Fall 2019

Predicting Switch-like Behavior in Proteins using Logistic Regression on Sequence-based Descriptors

Benjamin Nathan Strauss
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Strauss, Benjamin Nathan, "Predicting Switch-like Behavior in Proteins using Logistic Regression on Sequence-based Descriptors" (2019). *Master's Theses*. 5082.
DOI: <https://doi.org/10.31979/etd.9n27-vj9d>
https://scholarworks.sjsu.edu/etd_theses/5082

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

PREDICTING SWITCH-LIKE BEHAVIOR IN PROTEINS USING LOGISTIC
REGRESSION ON SEQUENCE-BASED DESCRIPTORS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering
San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Benjamin N. Strauss

December 2020

© 2020

Benjamin N. Strauss

ALL RIGHTS RESERVED

The Designated Thesis Committee Approves the Thesis Titled

PREDICTING SWITCH-LIKE BEHAVIOR IN PROTEINS USING LOGISTIC
REGRESSION ON SEQUENCE-BASED DESCRIPTORS

by

Benjamin N. Strauss

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2020

Jon Pearce, Ph.D.

Department of Computer Science

Brooke Lustig, Ph.D.

Department of Chemistry

Sami Khuri, Ph.D.

Department of Computer Science

ABSTRACT

PREDICTING SWITCH-LIKE BEHAVIOR IN PROTEINS USING LOGISTIC REGRESSION ON SEQUENCE-BASED DESCRIPTORS

by Benjamin N. Strauss

Ligands can bind at specific protein locations, inducing conformational changes such as those involving secondary structure. Identifying these possible switches from sequence, including homology, is an important ongoing area of research. We attempt to predict possible secondary structure switches from sequence in proteins using machine learning, specifically a logistic regression approach with 48 N-acetyltransferases as our learning set and 5 sirtuins as our test set. Validated residue binary assignments of 0 (no change in secondary structure) and 1 (change in secondary structure) were determined (DSSP) from 3D X-ray structures for sets of virtually identical chains crystallized under different conditions. Our sequence descriptors include amino acid type, six and twenty-term sequence entropy, Lobanov-Galzitskaya's residue disorder propensity, Vkabat (variability with respect to predictions from sequence of helix, sheet and other), and all possible combinations. We find the optimal AUC values approaching 70% for the two models of just residue disorder propensity and separately Vkabat. We hope to follow up with a larger learning set and using residue charge as an additional descriptor.

ACKNOWLEDGMENTS

In Chemistry, first and foremost I would like to thank my research advisor, Dr Brooke Lustig. It was his insight that made this project possible in the first place, and it was through his guidance that I was able to understand the results obtained. I thank him for his patience and the time he spent with me, and his constant reassurance when second-guessing myself.

In Computer Science, I thank Dr. Jon Pearce for all his help and Dr. Sami Khuri for his thoughtful input into my project and agreeing to be on my thesis committee.

I also thank Dr. Ningkun Wang for insights. I thank Jonathan Oribello and Saira Montermoso, for the time they spent helping with modifying their python code. I also thank Vikram Padala, Khai Cao, Edgardo Millan, Mai Nguyen, Phuong Tran, Jason Graham, and Angelina Huynh for their contributions to the research.

TABLE OF CONTENTS

List of Tables	viii
List of Figures	xii
1 Introduction.....	1
1.1 Amino Acids	1
1.2 Protein Structure	2
1.3 Molecular Sequence	2
1.3.1 Secondary Structure	3
1.3.2 Tertiary Structure	5
1.3.3 Quaternary Structure.....	5
1.4 Allosterism and Protein Switches.....	6
1.5 Problem Statement.....	7
2 Objective.....	8
3 Methods.....	9
3.1 Identifying Sequence-Based Descriptors	9
3.1.1 Residue Sequence.....	9
3.1.2 Sequence Entropy	9
3.1.3 IsUnstruct (Lobanov-Galzitskaya)	10
3.2 Vkabat	11
4 Applying Logistic Regression	12
5 Data	14
5.1 Choosing Learning and Test Sets	14
5.2 Modeling Proteins With Java	15
5.3 Data Set Generation	18
5.3.1 Downloading FASTAs	18
5.3.2 Obtaining Secondary Structure Data.....	20
5.3.3 Aligning Sequences.....	20
5.3.3.1 A FASTA alignment algorithm.....	22
5.3.4 Lobanov-Galzitskaya's IsUnstruct	24
5.4 The Python Scripts	24
5.4.1 Entropy Scripts.....	26
5.4.2 Vkabat Script.....	26
6 Results	29
6.1 Evaluating Descriptor Performance.....	29

6.2	Which Descriptor Combinations Worked the Best?	30
7	Discussion	33
7.1	Analyzing the Learning Set	38
7.1.1	Vkabat Issues.....	40
7.1.2	Entropy Issues.....	41
7.2	Future Work.....	41
7.2.1	Residue Charge	42
8	Conclusions	43
	Literature Cited.....	44
	Appendix A: Complete Learning Set	47
	Appendix B: FASTAs and Switches	50
B.1	Learning Set.....	50
B.2	Test Set	99
	Appendix C: Descriptor Overlays	108
C.1	Learning Set.....	108
C.2	Test Set	132
	Appendix D: Box Plots	135

LIST OF TABLES

Table 1.	Levels of Protein Structure [2], [3].	2
Table 2.	The sequence-based descriptors for the search for allosteric regions.	9
Table 3.	Groupings of Amino Acids Relevant to E6 Sequence Entropy [13]. .	10
Table 4.	Variables of the Lobanov-Galzit'skaya Equations [15].	11
Table 5.	Test Set Clusters.	15
Table 6.	Characters Found in the .dssp Files Denoting Secondary Structure. .	20
Table 7.	4 Different Sequences of 4UA3:A.	21
Table 8.	Vkabat Patches (for both Learning and Test Sets.)	28
Table 9.	ROC AUC Values for Assigned+Unassigned Switches.	30
Table 10.	ROC AUC Values for Only the Assigned Switches.	31
Table 11.	ROC AUC Values for Only the Unassigned Switches.	32
Table 12.	Triplet Assigned: IsUnstruct.	33
Table 13.	Triplet Assigned: Vkabat.	35
Table 14.	Triplet Assigned: Residue Type + E6 + IsUnstruct + Vkabat.	37
Table 15.	P-Values with and without Entropy (E6).	38
Table 16.	Learning Set Clusters.	47
Table 17.	Key.	50

LIST OF FIGURES

Fig. 1.	Two-dimensional structure of an amino acid [1]. <i>R</i> indicates the side chain.....	1
Fig. 2.	The parts of an amino acid [1].....	2
Fig. 3.	Resonance structure of amide linkage [5].....	4
Fig. 4.	Schematics of α -helicies and β -sheets(anti-parallel) [6].....	4
Fig. 5.	Protein 4JJX: Spermidine N1-Acetyltransferase from Vibirio Cholerae.	6
Fig. 6.	Class diagram of the protein chains.	17
Fig. 7.	Class diagram of the clusters.	18
Fig. 8.	ROC plot of IsUnstruct at threshold of 0.00001.....	35
Fig. 9.	ROC plot of Vkabat at threshold of 0.001.	36
Fig. 10.	Box Plot of IsUnstruct Values from the Learning Set (Single/Assigned+Unassigned). (Non-Switch: 0, Switch: 1).	39
Fig. 11.	Box Plot of E6 Values from the Learning Set (Single/Assigned+Unassigned). (Non-Switch: 0, Switch: 1).	40
Fig. 12.	Decriptors for 1CJW:A	108
Fig. 13.	Decriptors for 1GX3:A	108
Fig. 14.	Decriptors for 1IB1:A.....	109
Fig. 15.	Decriptors for 1N71:A	109
Fig. 16.	Decriptors for 1S7F:A	110
Fig. 17.	Decriptors for 1TIQ:A	110
Fig. 18.	Decriptors for 1XEB:A	111
Fig. 19.	Decriptors for 2B5G:A	111
Fig. 20.	Decriptors for 2BL1:A.....	112

Fig. 21.	Decriptors for 2FIA:A	112
Fig. 22.	Decriptors for 2JLM:A	113
Fig. 23.	Decriptors for 2QV4:A	113
Fig. 24.	Decriptors for 2REE:A	114
Fig. 25.	Decriptors for 2VFB:A	114
Fig. 26.	Decriptors for 2VI7:A	115
Fig. 27.	Decriptors for 2WPW:A	115
Fig. 28.	Decriptors for 3FS8:A	116
Fig. 29.	Decriptors for 3MQH:A	116
Fig. 30.	Decriptors for 3N37:A	117
Fig. 31.	Decriptors for 3NE7:A.....	117
Fig. 32.	Decriptors for 3QB8:A	118
Fig. 33.	Decriptors for 3SMA:A.....	118
Fig. 34.	Decriptors for 3TFY:A.....	119
Fig. 35.	Decriptors for 3W6S:A	119
Fig. 36.	Decriptors for 4AVA:A	120
Fig. 37.	Decriptors for 4EAB:A	120
Fig. 38.	Decriptors for 4FD4:A.....	121
Fig. 39.	Decriptors for 4FD5:A.....	121
Fig. 40.	Decriptors for 4FD7:A.....	122
Fig. 41.	Decriptors for 4HNY:A	122
Fig. 42.	Decriptors for 4J3G:A	123
Fig. 43.	Decriptors for 4JJX:A.....	123

Fig. 44.	Decriptors for 4KVO:A	124
Fig. 45.	Decriptors for 4KVX:A	124
Fig. 46.	Decriptors for 4PV6:A	125
Fig. 47.	Decriptors for 4R3K:A	125
Fig. 48.	Decriptors for 4R9M:A	126
Fig. 49.	Decriptors for 4RI1:A	126
Fig. 50.	Decriptors for 4U9V:B	127
Fig. 51.	Decriptors for 4UA3:A	127
Fig. 52.	Decriptors for 5GI5:A	128
Fig. 53.	Decriptors for 5HGZ:A	128
Fig. 54.	Decriptors for 5JPH:A	129
Fig. 55.	Decriptors for 5K04:A	129
Fig. 56.	Decriptors for 5K9N:A	130
Fig. 57.	Decriptors for 5KTA:A	130
Fig. 58.	Decriptors for 5WIF:A	131
Fig. 59.	Decriptors for 5YGE:A	131
Fig. 60.	Decriptors for 2B4Y:A	132
Fig. 61.	Decriptors for 2HJH:A	132
Fig. 62.	Decriptors for 4IG9:A	133
Fig. 63.	Decriptors for 4L3O:A	133
Fig. 64.	Decriptors for 5BTR:A	134
Fig. 65.	IsUnstruct values from the learning set, broken down by residue type.	135
Fig. 66.	E6 values from the learning set, broken down by residue type.	136

Fig. 67. E20 values from the learning set, broken down by residue type..... 137

Fig. 68. IsUnstruct values from the learning set, broken down by residue type. 138

1 INTRODUCTION

Amino acids are molecules that are often called the building blocks of life. In a biological system, these molecules are covalently connected together to form proteins, larger molecules that perform a function in an organism. Since many biological processes are initiated by a protein interacting with another molecule, understanding proteins is critical to understanding biology. Other molecules (ligands) binding to a protein can change the protein's function in a variety of ways, and these changes in function are an area of ongoing research.

1.1 Amino Acids

The function of a protein is determined by the amino acids that comprise it. Amino acids themselves are smaller molecules, made of a standard part and what is known as an R-group or side chain, which differs between amino acids. These side chains give the protein their 3D structures. The basic structure of an amino acid is shown in Fig. 1.

Fig. 1: Two-dimensional structure of an amino acid [1]. *R* indicates the side chain.

There are four main parts to amino acids: the amine group, the α -carbon, the carboxyl group, and the R group (Fig. 2).

Fig. 2: The parts of an amino acid [1].

When amino acids bond together to form proteins, they are joined by peptide bonds to form what is called a polypeptide, which is a number of amino acids bound together in a chain.

1.2 Protein Structure

Proteins have four levels of structure (Table 1). For identifying possible switches and allosteric regions, we mainly look at the primary and secondary structures.

Table 1: Levels of Protein Structure [2], [3].

Primary	Amino Acid Sequence
Secondary	Local Structure (Helix, Sheet, et cetera)
Tertiary	Overall Chain Structure
Quaternary	Interaction between Multiple Chains

1.3 Molecular Sequence

When organized into proteins, amino acids are usually known as residues. This is when the amino acids bond to form the peptide chain, and they lose a water molecule. However, since this always happens in the forming of polypeptide chains, this detail is mostly irrelevant for computationally analyzing proteins. Note that the terms “protein

chain,” “residue chain,” and “peptide/polypeptide chain” are used interchangeably in biology.

The first layer of protein structure is the amino acid residue chain sequence. We typically represent this computationally as a string of capital letters, where each amino acid is represented as a capital letter. Capital letters work well because there are 20 standard amino acids. There are also two “non-standard” amino acids which are represented by ‘O’ and ‘U’, respectively. Some letters also can stand for multiple amino acids, such as ‘X’ that can stand for any. ‘B,’ ‘J,’ and ‘Z’ as well as some Greek letters can stand for one of multiple amino acid types [4].

The two ends of the amino acid residue chain are referred to as amino terminus (N-terminus) and carboxyl terminus (C-terminus), respectively, and these are determined by the “free group” at that specific end. Amino acids each have two molecular connectors, an amine $-NH_2$ group and a carboxyl $-COOH$ group, and bonds between amino acids always occur between these connectors. If an amino acid has one of these groups not connected to another amino acid, it is called a free group. By convention, strings representing amino acid chains always start with whatever amino acid has a free amine group, which is also known as position 1 on the chain. The next amino acid is position 2, and the one after that position 3, et cetera, until the carboxyl end of the chain.

The sequence of a protein is unique to that protein, and defines its structure and function. Despite this many proteins have multiple chains of identical sequences.

1.3.1 Secondary Structure

The secondary structure of a protein is essentially defined by hydrogen bonding between amino acid backbones. Hydrogen bonding occurs between the carbonyl oxygen in Fig. 1 and any hydrogen in the polypeptide. This is partly because oxygen is highly electronegative, and partly due to the resonance structure shown in Fig. 3:

Fig. 3: Resonance structure of amide linkage [5].

A resonance structure is one in which electrons can be described as both structures. This also stabilizes the backbone of the protein molecule.

There are two basic major types of secondary structure, helices and sheets. However, some proteins also have other structures. Both of these structures are fairly straightforward to visualize at the large scale. For example, α -helix appears as a helical coil, with each amino acid residue interacting with the residues 3.6 positions above or below it [3]. On the other hand, β -sheet appears as two polypeptide backbones being held parallel or anti-parallel to each other [5]. A rough visual of this is shown in Fig. 4.

Fig. 4: Schematics of α -helices and β -sheets(anti-parallel) [6].

However, experimentally obtaining the secondary structure in detail is non-trivial. Luckily there are many programs designed to characterize protein topology from X-ray and NMR data, including secondary structure.

1.3.2 Tertiary Structure

Tertiary structure is determined by the R-groups, the part of the amino acids that makes the molecule unique. R-groups are comprised of hydrogen, carbon, nitrogen, oxygen, and sometimes sulfur or selenium, and are sorted into two main groups, polar and non-polar. This makes a difference in the 3-dimensional structure of a protein chain since water is a polar molecule, and the cytoplasm of cells (where proteins typically reside) is mostly water. Polar molecules interact with a polar solvent (like water in cells) and non-polar molecules do not. As a result, the polar R-groups tend to be on the outside of the protein and the non-polar R-groups cluster together on the inside, away from the water.

Polar amino acids are then further divided into three categories: acidic, basic, or neutral. Among the first two, ionic interactions are possible, while the latter can participate in hydrogen bonding, including with water. These interactions can contribute to the protein's tertiary structure. Finally, if there are two cysteine residues that are physically near each other, the sulfur atoms in the R-groups can form a covalent bond, which also contributes to the tertiary structure, and possibly between chains with respect to quaternary structure [5].

1.3.3 Quaternary Structure

The final type of structure that proteins have, quaternary structure, is the result of interaction between multiple chains of amino acid residues. Note that typically the chains are not covalently bound to each other, so a protein with a quaternary structure is actually multiple molecules. (Proteins with only one chain are said to have no quaternary structure.) The chains involved may or may not all have the same primary sequence, but

having the same primary, secondary, and tertiary structure is not uncommon of the chains in a protein.

An example of quaternary structure is a protein named 4JJX (RCSB-PDB [7] model) shown in Fig. 5.

Fig. 5: Protein 4JJX: Spermidine N1-Acetyltransferase from Vibrio Cholerae.

4JJX has 12 chains with identical sequence of amino acid residues. 4JJX also has a double-layered structure with each layer being comprised of a hexamer, or structure consisting of six residue chains. These chains are labeled 4JJX:A to 4JJX:L respectively, even though they all have (effectively) the same residue sequence.

1.4 Allosterism and Protein Switches

Protein conformational switches are regions of a polypeptide chain that undergo a significant structural change upon receiving a signal. As a result of this structural change, the protein then interacts with its environment differently (for example, new areas of the protein could be exposed to solvent). The potential signals consist of molecules binding to the chain covalently, or otherwise (such as hydrogen bonding, absorbing a photon,

binding a drug, or engaging an entire cell via a surface receptor). And while many proteins bind other molecules and/or become covalently modified; in order for a protein to be a switch, the structural change must be stimulus induced [8].

The classic example of this is the protein hemoglobin (typically found in blood). In the protein-ligand reaction, the oxygen molecule O_2 binds to the hemoglobin, and as a result, makes it easier for other oxygen molecules to further bind. Other molecules that bind to proteins in this manner can affect protein structural features, such as side-chain packing, secondary structure mobility, domain movement, and oligomerization, depending on where and how they bind to the protein [9].

1.5 Problem Statement

Protein allostery presents a giant puzzle for researchers as the extent of allosteric behavior in proteins is not well understood. Furthermore, we do not know if protein allostery is a boolean on/off switch based on the presence of one specific ligand, or a switch statement based on one of many potential ligands. And of course it may be that in some situations multiple ligands can bind to one allosteric region on the same protein.

In this sea of unknowns, we have decided it is best to start unraveling how this mechanism works by trying to identify possible protein switches. We detail this in the next sections.

2 OBJECTIVE

Determining switch-like behavior requires us to define it. We decided upon the following definitions. Note that residues unassigned are classified by an absence of secondary structure type by the gold standard database DSSP [10].

Definition 1. *An amino acid residue exhibits switch-like behavior if and only if it, under different conditions, has two or more of the following categories of secondary structure: Helix, Sheet, and Other, or is Unassigned.*

From here, we split switch-like behavior into two types:

Definition 2. *An amino acid residue is an assigned switch if and only if under different conditions, has two or more of the following categories of secondary structure: Helix, Sheet, Other.*

Definition 3. *An amino acid residue is an unassigned switch if and only if it has one of only two classes of secondary structure, with one category being Unassigned.*

3 METHODS

3.1 Identifying Sequence-Based Descriptors

To look for allosteric regions, we chose a pattern-based approach of analyzing already known allosteric regions for patterns based on five specific criteria (Table 2).

Table 2: The sequence-based descriptors for the search for allosteric regions.

Residue Sequence	Sequence of types of side chains on the amino acids that make up the proteins.
6-Term Sequence Entropy (E6)	First order approximation of local flexibility of the molecule; high values suggest high mobility of that residue.
20-Term Sequence Entropy (E20)	Alternative first order approximation of local flexibility of the molecule; high values suggest high mobility of that residue.
Lobanov-Galzitskaya (IsUnstruct)	Probability than a residue will be disordered.
Vkabat	Variability of predicted secondary structure.

3.1.1 Residue Sequence

The residue sequence is our first descriptor, and our only qualitative one [11]. The Uniprot FASTAs we used classify amino acids with respect to the standard 20, so the residue sequence gets modeled as an array of 20-valued enums.

3.1.2 Sequence Entropy

Sequence Entropy (also known as “Shannon Information Entropy”) is a type of information entropy, which is related to thermodynamic entropy [12]. Additionally, in this 2005 paper by the Lustig group, low sequence entropy was correlated to inverse packing density (a physical measure of local flexibility). As it turns out, sequence entropy at a particular residue k , (denoted as S_k) is not difficult to calculate, as given by the equation

$$S_k = - \sum_{j=1}^{20} P(j,k) \log_2(P(j,k)) \quad (1)$$

where $P(j,k)$ is the probability of how often residue k is present in a set of aligned residues. The 20 values for j come from the 20 standard [12] amino acids (we do not

count pyrrollysine or selenocysteine). Note that an alternative version of this calculation can be done in which j goes from 1 to 6; in this case, each value for j represents a group of amino acids. The amino acids are categorized [13] in Table 3:

Table 3: Groupings of Amino Acids Relevant to E6 Sequence Entropy [13].

Aliphatic	Alanine, Cysteine, Isoleucine, Leucine, Methionine, Valine
Aromatic	Phenylalanine, Histidine, Tryptophan, Tyrosine
Polar	Asparagine, Glutamine, Serine, Threonine
Positive	Lysine, Arginine
Negative	Aspartic Acid, Glutamic Acid
Special	Glycine, Proline

3.1.3 *IsUnstruct* (Lobanov-Galzitskaya)

The Lobanov-Galzitskaya method, *IsUnstruct*, is a way of predicting with amino acid residues that are disordered solely from the residue sequence [14]. The output of the *IsUnstruct* method is a number from 0 to 1 for each residue, which is then interpreted as the probability that that residue will be disordered.

For our purposes, we use *IsUnstruct* as a black box function that takes in a protein chain sequence and outputs a value. This Ising model application comes from statistical physics (as Ising models were originally used to predict statistical mechanical phenomena) and the final value is derived as the following ratio:

$$p_{i,1} = \frac{Z_{i,1}}{Z} = \frac{\sum_{j=i}^M s_{ij} \cdot e^{\frac{-E_j}{kt}}}{\sum_{j=i}^M e^{\frac{-E_j}{kt}}} \quad (2)$$

where E_j is defined as

$$E_j = \sum_{i=1}^L w(a_i, s_{ij}) + e_b \cdot b + t_n \cdot e_n + t_c \cdot e_c \quad (3)$$

Equations 2 and 3 have been relabeled [15] from Lobanov and co-workers as shown in Table 4.

Table 4: Variables of the Lobanov-Galzitskaya Equations [15].

a_i	type of amino acid residue
b	number of bordering residues
e_b	energy of the residue's borders
e_c	energy of initiation of the c-terminus
E_j	energy of the microstate
e_n	energy of initiation of the n-terminus
k	Boltzmann constant
L	length of the protein chain
s_{ij}	the state of the i th residue at the j th conformation
T	temperature
t_c	0 if c-terminus residue is ordered, else 1
t_n	0 if n-terminus residue is ordered, else 1
$w()$	given in a table in [15]

3.2 Vkabat

Vkabat is our way of calculating how diverse the range of secondary structures are [16] and is defined [13] by the following function:

$$Vkabat = \frac{k \times N}{n_1} \quad (4)$$

In this equation, k is the number of secondary structure classes predicted for that residue. Here we can classify up to three categories, helix, sheet and other. N is the total number of possible chains, or how many predictions we are making. The variable n_1 is the number of times that the most common type of secondary structure occurs in all of the predictions made. For our purposes, $N = 15$, since we use the 15 algorithms for calculating the prediction of secondary structure originally applied by Saravanan and Selvaraj [17]. The minimum and maximum values for Vkabat are 1 and 9, respectively.

4 APPLYING LOGISTIC REGRESSION

Logistic regression is a mathematical way of determining the probability of a binary event occurring based on one or more parameters. Applying the aforementioned descriptors, our binary event is whether or not a protein chain residue in question has switch-like characteristics. Using logistic regression with proteins is not new, and was done previously by the Lustig group predicting from sequence which R groups were on the inside of a folded protein [18], as well as residues of one protein interacting with residues on another.

To implement logistic regression, we first assemble learning and test sets of protein chains. From this we plan to calculate switch-like regions, depending on the descriptor value(s). Our quantitative descriptors are sequence entropy, IsUnstruct, and Vkabat. It is important to note that the odds is not the probability, although it can be derived from the probability p as such:

$$\text{Odds} = \frac{p}{1-p} = e^{\sum_{i=1}^n k_i x_i} \quad (5)$$

Here x_i is the quantitative descriptor and k_i is the associated weighting coefficient [19], [20]. However, the odds in this raw form do not give a linear correlation with the descriptor data. So to make matters simpler, most often the odds are transformed into the “log odds” (literally the natural log of the odds) which turns the formula into a linear polynomial:

$$\text{Log Odds} = \ln \text{Odds} = \ln \left(\frac{p}{1-p} \right) = \sum_{i=1}^n k_i x_i \quad (6)$$

The log odds for each residue are given by the following coefficients and descriptors [18]:

$$\eta = \beta_0 + (\beta_1 \times E6) + (\beta_2 \times E20) + (\beta_3 \times \text{IsU}) + (\beta_5 \times \text{VK}) + \sum_{i=1}^{19} (\beta_{i+4} \times X_i) \quad (7)$$

Here the β terms are constants (and η is the symbol for the log odds). The last terms of equation 7 are qualitative descriptors. Each of the different types of amino acid have their own unique identifier. Depending on what type of amino acid is present, one X_i will be 1, and all the rest will be 0. (Note that β_0 should be representative of valine.) One can rearrange the terms in equation 5 and noting equation 7 get the following:

$$p = \left(\frac{e^{\beta_0 + \beta_1 E6 + \dots}}{1 + e^{\beta_0 + \beta_1 E6 + \dots}} \right) \quad (8)$$

The other acronyms for descriptors are as follows:

- E6 is a six-term sequence entropy
- E20 is a 20-term sequence entropy
- IsU is IsUnstruct
- VK is Vkabat

5 DATA

5.1 Choosing Learning and Test Sets

We chose N-acetyltransferases and sirtuins respectively because there is some documentation of possible switch behavior among them [21]. In addition it should be noted that N-Acetyltransferases and Sirtuins share similar regulatory and functional features.

Originally, the chains were clustered with Pisces, at 98% or greater sequence identity [8]. We got our initial chains to do this clustering from Khun (personal communication) which lead to 34 usable clusters (some of which later were determined to be part of larger clusters). We added to these clusters using the results of RCSB-PDB keywords searches: “spermidine” and “n-acetyltransferase.”

To further refine the clusters, we discovered that JPred, used in the VKabat algorithm, lists all the protein chains that have a similar sequence to the sequence entered. (From here, it offers the opportunity to use a known chain’s secondary structure sequence instead of running the prediction). It became immediately obvious that this could be used to find similar chains, since they were listed in ascending order by blast E-value. (The lower the blast E-value, the more similar a chain was to the input sequence and vice versa.)

Thus, we entered each dominant chain’s sequence into JPred and then went down the list of chains, checking each chain’s RCSB-PDB FASTA sequence manually until a chain did not resemble the dominant chain. It was quickly discovered that once one sequence did not resemble the dominant chain, none of the ones after would. This latter method identified an additional cluster (4KVO and related sequences) resulting in a total of 48 working clusters (see Appendix 16). Note 44% of clusters are 5 or less, while 84% are ten or less. These are comparable to cluster sizes associated with determining C_{α} displacements for 3D X-ray structures crystallized under different conditions [22].

The test set clusters of Sirtuins were previously assembled [21]. However it was later determined to use “4IAO” instead of “4AIO.” The same JPred-based method was applied to the Sirtuins and no changes resulted when compared to the original clusters. We ended up with five usable clusters of Sirtuins, as shown in Table 5, where the dominant chain is the original chain of interest.

Table 5: Test Set Clusters.

Dominant Chain	Other Chains
2B4YA	2B4YB, 2B4YC, 2B4YD, 2NYRA, 2NYRB, 3RIGA, 3RIGB, 3RIYA, 3RIYB, 4F56A, 4F56B, 4F4UA, 4F4UB, 4G1CA, 4G1CB, 5BWL A, 4HDAA, 4HDAB, 6EQSA, 6EQSB, 6EQSC, 6EQSD, 5XHSA
2HJHA	2HJHB, 4IAOA
4IG9A	4IG9C, 4IG9E, 4IG9G, 4KXQA, 4IF6A, 4I5IA, 4I5IB
4L3OA	4L3OB, 4L3OC, 4L3OD, 4RMHA
5BTRA	5BTRB, 5BTRC, 4ZZHA, 4ZZIA, 4ZZJA

The next challenge was identifying switches. Since we define switch-like behavior as a residue changing its secondary structure, we had to determine which residues learning and test set exhibited this behavior.

To do this, we decided to compare each dominant chain (the chain for which we were trying to determine switch-like regions for) to chains with identical or nearly identical sequences, resulting in clusters. It is important to note that in order for us to use a protein chain in the regression, we needed to have at least one chain that was similar for comparison.

5.2 Modeling Proteins With Java

Starting out, we had python scripts to calculate the entropy and V_{kabat} of a given residue sequence, and we had the Lobinov-Galzitskaya IsUnstruct program written in C. So while we could generate descriptor data for protein chains, we had no automated way of organizing it. (And processing this much data by hand was extremely impractical). My goal was to automate the entire process, and create a program that if given a list of

RCSB-PDB IDs, could generate a data set to be used in the logistic regression without any manual assistance. While we did not achieve complete automation, (see 7.1.1), we came very close and automated most of the data set generation.

We used Java to write our program. Java's object-oriented approach meant one could model the components individually, with each component being a specific molecule or molecules, or part thereof. These objects would then hold descriptor data relevant to the modeled component (see Fig. 6). Once we had the model, it would then be simpler, if not trivial, to gather the requested data from it and display that data in whatever form we found most useful and intuitive.

Fig. 7: Class diagram of the clusters.

5.3 Data Set Generation

Generating the data sets used in the logistic regression was a multi-step process, but fairly straightforward in concept. Data needed to be downloaded, organized, and then written out into CSV format. The logistic regression python script would then use the CSV format of the data to run the logistic regression. The results would then be read back in to Java, which would perform further analysis.

5.3.1 Downloading FASTAs

We initially planned to use the FASTAs from the RCSB-PDB, and did use these for the early runs of our project. However there may be a problem: sometimes parts of the residue chains were missing, and the RCSB files in no way indicated where and how big were the gaps. Given how algorithms like IsUnstruct and Vkabat base their results on

sequence alone, this meant that the RCSB-PDB FASTAs were problematic for our purposes.

From here, we turned to GenBank and Uniprot, and we modified the program to allow the user to specify the FASTA source (and we set it to Uniprot by default). Uniprot FASTAs turned out to be more robust than their GenBank counterparts, as some of the Vkabat component algorithms ran into trouble when using the GenBank FASTAs. Sometimes because they could not handle a residue 'X' (Uniprot FASTAs did not have 'X' in them, but used the closest standard amino acid), and sometimes apparently for other reasons.

Using Uniprot however created another problem: the Uniprot did not use the RCSB-PDB identifiers. This meant that in order to get the Uniprot sequence from an RCSB-PDB identifier, one had to go to the RCSB-PDB page and parse the data to find where the Uniprot identifier was listed. For the most part this was easy since when parsing the RCSB-PDB HTML, the Uniprot identifier was always clearly marked. But with this approach, we still ran into trouble.

While this approach always obtained a Uniprot ID, the Uniprot ID was occasionally incorrect. This happened when a RCSB-PDB page had multiple Uniprot IDs for the same RCSB-PDB identifier, and from the RCSB-PDB identifier alone, there was no way to tell which Uniprot ID we wanted. To get around this issue, we modified the way that the cluster file is read: allowing the user to put a Uniprot ID after the RCSB-ID to ensure that that Uniprot ID will be used. (If no Uniprot ID is entered, the program will look up the Uniprot ID in the aforementioned way if necessary.) When doing this, the program will always obtain the first Uniprot ID on the page.

After downloading the FASTA sequence, the sequence will be saved in "files" under a directory named after where that FASTA came from, so it does not have to be downloaded again.

5.3.2 Obtaining Secondary Structure Data

To obtain the secondary structures of a protein chain, we used the DSSP files, which are held locally under "files/DSSP/". When assigning secondary structures to a protein chain, we ran into the problem that sometimes a DSSP file could not be found. (The reason, as it turned out, is that we did not have all of the DSSP files, and downloading the rest fixed this).

If secondary structures cannot be assigned from the DSSP files, the program is also equipped to assign them from the headers in the appropriate PDB files from RCSB (in which case the user will be alerted). This RCSB secondary structure information is stored in "files/ss.txt". Unless one is working with a FASTA sequence from PDB or GenBank, using PDB annotated secondary structures is sub-optimal, since a PDB sequence must be used as an intermediary (the sequence from the PDB corresponds with the PDB FASTA and may not match other FASTAs) and aligning RCSB sequences to other sequences can be error-prone, (see Table 7 for why this is a problem and 5.3.3.1 for how we deal with it).

Table 6 shows how the characters in the DSSP files were interpreted. The code that does this can be found in SecondaryStructure enum in the bio package.

Table 6: Characters Found in the .dssp Files Denoting Secondary Structure.

DSSP-Char	Secondary Structure	Structure Category
B	Beta Bridge	Sheet
E	Extended Strand	Sheet
G	Three Helix	Helix
H	Alpha Helix	Helix
I	Five Helix	Helix
S	Bend	Other
T	Turn	Other
(blank)	Other or Random Coil	Unassigned

5.3.3 Aligning Sequences

A huge problem when dealing with proteins is that different databases have slightly different sequences for the same protein chain, even when ignoring the sequence gaps in

the RCSB-PDB. To do this project, we used four different sources for protein sequences: RCSB-PDB, GenBank, Uniprot, and DSSP. This meant that for a given chain, such as 4UA3:A (a chain that gave us a lot of trouble), there could be four different sequences, as in Table 7:

Table 7: 4 Different Sequences of 4UA3:A.

Source	Sequence (4UA3:A)
RCSB-PDB	XSRRMKTQEIKNVTLEDVDFLKNLGVWVEIYHHLEKGLLQQCFNLVKKNMEALYRQS SFGWDDSEKLKEMEMEKLEYICIFEKTSKKLVGFLSFEDTVEAGLTCLYIYEIQLDE HIRGRNVGKWLKNASILAYRRNLKYIFLTVFSANLNALNFYHHFDFVPHESSPQEK KFRSGKVIHPDYYIILYTKSRKDW
GenBank	XSRRXKTQEIKNVTLEDVDFLKNLGVWVEIYHHLEKGLLQQCFNLVKKNXEALYRQS SFGWDDSEKLKEXEXEKLEYICIFEKTSKKLVGFLSFEDTVEAGLTCLYIYEIQLDE HIRGRNVGKWLKNASILAYRRNLKYIFLTVFSANLNALNFYHHFDFVPHESSPQEK KFRSGKVIHPDYYIILYTKSRKDW
Uniprot	MHFLKQYLFYSPRRMKTQEIKNVTLEDVDFLKNLGVWVEIYHHLEKGLLQQCFNLVK KNMEALYRQSSFGWDDSEKLKEMEMEKLEYICIFEKTSKKLVGFLSFEDTVEAGLTC LYIYEIQLDEHIRGRNVGKWLKNASILAYRRNLKYIFLTVFSANLNALNFYHHFDF VPHESSPQEKKFRSGKVIHPDYYIILYTKSRKDW
DSSP	SRRX VTLEDVDFLKNLGVWVEIYHHLEKGLLQQCFNLVK KNXEALYRQSSFGWDDSEKLKEXEXEKLEYICIFEKTSKKLVGFLSFEDTVEAGLTC LYIYEIQLDEHIRGRNVGKWLKNASILAYRRNLKYIFLTVFSANLNALNFYHHFDF VPHESSPQEKKFRSGKVIHPDYYIILYTKSRKDW

The fact that all these sequences could be different (‘*’ denotes something missing from the DSSP file) meant that when using the same sequence from any two sources, the sequences would have to be reconciled. Most of the possible problems that arose are visible from looking at the different sequences from 4UA3:A, and we break them down as follows:

The first kind of problems we faced were **Index Problems**. RCSB-PDB and GenBank sequences have the same indices, and DSSP has index numbers that align with Uniprot. The trouble is that RCSB-PDB/GenBank and Uniprot/DSSP **do not** match. These sequences not only tend to start/end at different residue positions, and as mentioned earlier, RCSB-PDB/GenBank can contain undeclared gaps in the FASTAs.

Unknown Residues also caused problems. GenBank and DSSP list non-standard residues as 'X' (selenomethionine is the most common reason for 'X') but RCSB-PDB and Uniprot use the closest standard residue. (4UA3:A is an exception to this rule, as the RCSB-PDB sequence contains an 'X' at the beginning, which corresponds not to an amino acid residue, but to an acetyl group.) Having an 'X' residue in a sequence would cause some of the Vkabat algorithms to crash.

Residue letter mismatches were also far from unheard of, even if neither was 'X'. We are unsure if this was a data-entry typo. DSSP uses **b** as a residue letter to denote a cysteine with a sulfur-sulfur bonded to another cysteine. This was an easy fix once we knew that it was happening.

5.3.3.1 A FASTA alignment algorithm: The solution to all of these problems was to create a Java class with static methods specifically for aligning residue sequences (bio.tools.SequenceAligner). Unlike normal string alignment algorithms, the purpose of this algorithm is to differentiate (as accurately as possible) between what is an unmarked gap in a sequence and what is a normal mis-match. Given that no table of sequence alignments exists (at least that we were able to find), one has to guess, and thus by the very nature of the algorithm, perfection is not guaranteed.

The first step in the sequence alignment algorithm is to figure out a base sequence that matches. The algorithm takes two strings of characters, each representing a residue sequence. From here, we look in the first sequence for a unique sub-sequence of K residues. Then we look for the same string in the second sequence. If the second sequence

also has a unique sub-sequence of K residues, we align the chains based on this sub-sequence, if not, we search for a new unique sub-sequence of K residues in the first sequence.

With the initial alignment, choosing a good K is critical. For a sequence of K residues, there is a 1 in 20^K chance that the sequence will happen randomly in the other protein, so a high K means a lower chance of error. However, if K is too high, errors in the sequences will mean that no match can be found and the alignment will fail. Thus the challenge is choosing the right value for K . We went with $K = 32$, (although a java programmer executing the alignment algorithm can specify a different value for K).

With $K = 32$, this means that the chance of giving a false alignment is 1 in 20^{32} , or roughly 1 in 4.3×10^{41} , which is fairly small. Even when looking at sub-sequences of a sequence of 1000+ residues, the chance that there will be a mis-match by chance is 1 in 4.3×10^{38} , which is still negligible.

The second part of the algorithm requires a bit more guesswork. `bio.tools.SequenceAligner` has a method called `superAlign()`, which is designed to differentiate between unmarked gaps in sequences and residues that are marked differently in different sources. There is no way to know for sure, but this algorithm makes estimates based on the length of mis-matched regions, using the assumptions that data mis-matches are fairly uncommon and that unmarked gaps are usually not small.

From this, if a large number of residues in one sequence do not match the residues at those same indexes in the other sequence, it is more likely that an unmarked gap is the cause rather than a data conflict. If it is only one or two residues that do not match, then it is most likely a data conflict and can be ignored. In areas that only match in part, the algorithm has to determine if more residues match or mis-match, and then decide whether it is a gap, or data conflict, based on the number of mis-matching residues and the closest matching regions.

When dealing with what are determined to be gaps, the `superAlign()` method aligns segments first by inserting blank residues, starting with the largest (anything size 32 or larger) and moving down aligning smaller and smaller segments until it reaches segments of size 4. We chose the number 4 specifically because it is the smallest number of residues seen here in any FASTA to be separated from the true sequence by an unmarked gap. The final step is then to go through both sequences, and remove all blank residues that have a blank residue at the same index in the other sequence.

5.3.4 *Lobanov-Galzitskaya's IsUnstruct*

Of the descriptors we used, `IsUnstruct` was the second-easiest to obtain (after the sequence). Lobanov and Galzitskaya [14], [15] conveniently made `IsUnstruct` a C program, which meant we could run it locally, and since the source code was available, it was not too difficult to translate the entire program into Java. The code for the Java version of `IsUnstruct` is located in the package `bio.tools.isUnstruct`, and gives the same results as its C counterpart. This was tested on multiple occasions, including against propensities calculated with the C version for the protein chain 5BTR:A by Lustig, Wong, and Graham [21].

5.4 The Python Scripts

Obtaining the descriptors regarding entropy and `Vkabat` involved using Lustig Group Python scripts that had been written by my colleagues, Jonathan Oribello and Saira Montermoso. These scripts were not translated into Java, but run from Java as Python scripts. Java would then read in the results.

Getting these scripts to run was difficult because they required a newer version of Python than was installed. We do not know what the minimum Python requirement is, but Python 3.7.0 is known to work. When running the Python scripts, it is critical to specify the path for Python; if the wrong version of Python is used, the scripts will NOT work! The program uses the default Python path of `“usr/local/bin/python3”` (which works on

Mac OS) but another path can be specified with the `bio.util.PythonScript.setPythonPath()` method.

Another requirement to run the Python scripts is to have all of the packages installed. The easiest way to do this is with `pip`, as this will tell you if the package is already installed. These packages include the following, although some may be installed by default.

- `argparse`
- `Bio`
- `Bio.Blast`
- `collections`
- `csv`
- `htmlement`
- `lib`
- `logging`
- `numpy`
- `os`
- `pandas`
- `requests`
- `requests_toolbelt.multipart.encoder`
- `subprocess`
- `sys`
- `time`

Finally, the scripts are designed to work with the Java program and not designed for standalone use. For these scripts to work properly, they must be run from the SwitchPrediction folder. All the paths to folders (and thus the data that they use) are based on this.

5.4.1 Entropy Scripts

To calculate the entropy values E6 and E20, the Java program executes a pair of Python scripts. The first of these is "getBlast.py" or "getBlastUniprot.py," which downloads the blast data. The script called depends on what the FASTA source (a variable internal to the program, e.g. RCSB-PDB FASTA) is set to: DSSP or Uniprot FASTAs will mean "getBlastUniprot.py" is called. Else, "getBlast.py" will be called. This downloads the blast file and saves it in the proper folder within "scripts/blast/". After this, "calculate_entropy.py" is called, which calculates the E6 and E20 from the blast file. The Java program will then read this in, and assign E6 and E20 to the chain.

Since there are 22 proteinogenic amino acids, the Java code also contains an E22 value. However, the program has no way to calculate an E22 value, and the feature was unused in our project. To use the E22 value, one will have to calculate E22 with some other method, and then modify the logistic regression script to make use of E22.

5.4.2 Vkabat Script

Our Vkabat script is called "neo-vkabat.py" (to differentiate it from the original version made for a different project). When run, neo-vkabat obtains Vkabat data in the following order: GOR, DPM, GOR3, PHD, PREDATOR, HNN, MLRC, SOPM, JPred, then PROF, SSPRO, YASPIN, JNET, and PSIPRED via SymPred, and finally DSC. Neo-vkabat can be finicky, and often times one or more prediction algorithm will fail.

These algorithms can fail for various reasons, and it is not always clear why any given algorithm failed. When the prediction algorithms fail, they do not always fail consistently, that is, the same algorithm might fail for a given protein chain at one time and succeed at another. We do not know the reason for this, however we think it might be linked to the workload of the server in question. (There was no way for us to know the workload of the server).

Some algorithms are more robust than others, and some have a limit to the length of the chain that they can analyze. JPred has a limit of 800 residues, and will not run on longer chains.

To solve this problem, we created the class `bio.tools.VkabatRecoveryModule`. `VkabatRecoveryModule` is used through the “patch” command of the `JBioMain` user interface, and will look in the “patches” directory, under the sub-folder that the fasta source is set to. To patch a Vkabat file, follow the steps below:

- 1) Run the secondary structure prediction algorithm online.
- 2) Copy the output into a text file with the name [protein+chain]-[algorithm].txt. (For example if 1S7F:A from a uniprot fasta fails JPred, then the patch would be “1S7FA-JPRED.txt”).
 - For JPred results, click “View simple results in HTML.”
 - For Sympred results use "SYMPRED+" for the algorithm.
- 3) Place that file in the "patches" directory under the sub-folder which corresponds to from where the sequence came.
- 4) From `JBioMain`, run “src” and enter the name of the sub-directory where your patch is. (Only the name, not the path).
- 5) From `JBioMain`, run “patch”.
- 6) From `JBioMain`, run “redo-vk” to re-assign the Vkabat values along with the patched data.

When run `VkabatRecoveryModule`, checks what secondary structure prediction data is in the Vkabat file by looking at the csv header, as so not to duplicate data already in the file if the patch command is run twice.

`VkabatRecoveryModule` does not read patches for all secondary structure prediction algorithms, just the ones that gave us trouble: PROF, SS-PRO, YASPIN, JNET, PSIPRED,

PHD (via Sympred), JPRED, and MLRC. Please make sure that the patch source is all uppercase on the patch file names.

The patches that we used are shown in Table 8:

Table 8: Vkabat Patches (for both Learning and Test Sets.)

Chain	Patches
1N71A	SYMPRED (PROF)
1S7FA	JPRED
1XEBA	SYMPRED (PROF)
2B5GA	JPRED
2HJHA	SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
2REEA	JPRED, MLRC, SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
2WPWA	SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
3N37A	PHD, SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
3W6SA	JPRED
4AVAA	SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
4EABA	JPRED
4HNYA	JPRED, SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
4IG9A	SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
4KVOA	JPRED, SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
4KVXA	JPRED
4L3OA	PHD, SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
4PV6A	JPRED
4UA3A	JPRED
5BTRA	SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
5K04A	SYMPRED (PROF, SSPRO, YASPIN, JNET, PSIPRED)
5WIFA	JPRED
5YGEA	JPRED

Note that for JPred, when we had FASTAs over 800 residues long, we broke sequences into 780-residue chunks because Uniprot FASTA lines are 60 residues long ($60 \times 13 = 780$).

6 RESULTS

6.1 Evaluating Descriptor Performance

We used ROC (receiver operating characteristic) statistics to evaluate which descriptors worked the best. Our logistic regression algorithm generates a probability (see equation 8) for every residue that specific residue is a switch. From this, we totaled up the true positives, false positives, true negatives, and false negatives.

Our learning set consisted of 13797 residues. In our test set, there were 198 assigned switches and 238 unassigned switches, for a total of 436 switches out of a total 2754 residues. We performed the logistic regression 6 times, twice on each of the switch types. Of each these two logistic regressions, one regression used the calculated E6, E20, and IsUnstruct values for the specific residue, whereas the other regression averaged E6, E20, and IsUnstruct values for each residue with its neighboring residues (but not V_kabat!) We label our tests as follows:

- **sa:** Single residue value, Assigned switches only
- **sb:** Single residue value, Both assigned and unassigned switches
- **su:** Single residue value, Unassigned switches only
- **ta:** Triplet residue value, Assigned switches only
- **tb:** Triplet residue value, Both assigned and unassigned switches
- **tu:** Triplet residue value, Unassigned switches only

For each descriptor combination in each regression, we plotted the true positive rate (TPR) against the false positive rate (FPR) (see supplemental files for fitting data) at thresholds of 0.01, and then we measured the AUC (area under the curve). High AUC values correspond with a good descriptor combination. An AUC of 0.5 means the descriptor combination is essentially random when making predictions. We found that the accuracy of a prediction itself was not very useful, as the highest accuracies came almost

exclusively from true negatives. We think this is because the signal is small compared to the background, since this is a similar problem to protein-protein interactions [23].

6.2 Which Descriptor Combinations Worked the Best?

Table 9: ROC AUC Values for Assigned+Unassigned Switches.

Descriptor Combination	Single residue value	Triplet residue value
E6 + IsU + VK	0.38588707443145387	0.3933273503732259
E6 + IsU	0.38187677608821274	0.39265451472718493
E6 + Res + IsU + VK	0.3841035652373527	0.3944889813268319
E6 + Res + IsU	0.3855600565181942	0.3933788025108643
E6 + Res + VK	0.4823751691983757	0.4790129698965416
E6 + Res	0.4820491407493014	0.47553203489246504
E6 + VK	0.4776455303923819	0.474197247706422
E6	0.4781412519492444	0.47597185172285505
E20 + IsU + VK	0.3905350824421559	0.39555760264701456
E20 + IsU	0.391089677118047	0.39598208278253166
E20 + Res + IsU + VK	0.3884290079236293	0.3970225043734316
E20 + Res + IsU	0.38738314427970966	0.3953948357885238
E20 + Res + VK	0.48421804624359815	0.48303415234582164
E20 + Res	0.48093005675566575	0.479067390426736
E20 + VK	0.4847360307446312	0.4790530431960484
E20	0.48068269070932707	0.47741102738045293
IsU + Res	0.3631575978975865	0.3630240202325637
IsU + VK	0.3654679967703889	0.3646858253318663
IsU	0.36700512938233687	0.3679085101835654
Res + IsU + VK	0.3635439836619674	0.3638739699677831
Res	0.4986889599544055	
VK + Res	0.5023014936951342	
VK	0.5295671687867587	

As one can see from Table 9, when trying to predict both types of switches, just about every descriptor combination fails miserably. Vkabat is the best at almost 0.53, but this is barely better than random. Lobanov-Galzit's IsUnstruct when combined with the sequence performed the worst at around 0.36. Averaging the E6, E20, and IsUnstruct values over a residue and its neighbors caused only the most minimal change.

Table 10: ROC AUC Values for Only the Assigned Switches.

Descriptor Combination	Single residue value	Triplet residue value
E6 + IsU + VK	0.5446473735792985	0.5300293624824142
E6 + IsU	0.4991207062803307	0.4779099682268696
E6 + Res + IsU + VK	0.5583900428383994	0.5501404894010528
E6 + Res + IsU	0.5144500561167228	0.5147889299884605
E6 + Res + VK	0.519042142868434	0.49978264649626153
E6 + Res	0.469330432652029	0.4578561435955802
E6 + VK	0.5178368188931569	0.5006332890722562
E6	0.4392546355574525	0.4261867501304121
E20 + IsU + VK	0.5387432620413841	0.5297280314885948
E20 + IsU	0.4688799181170073	0.48277473482872546
E20 + Res + IsU + VK	0.5499290637201436	0.5382700241855171
E20 + Res + IsU	0.5090537614011792	0.49887568960338924
E20 + Res + VK	0.5123545707465895	0.5044794581179557
E20 + Res	0.459464559523245	0.4468620081882993
E20 + VK	0.5176421887102638	0.4983332938145145
E20	0.43037080507737785	0.43857095999114776
IsU + Res	0.5615762476091115	0.5621374148369453
IsU + VK	0.5868356886549375	0.6002128088395694
IsU	0.6508571631811069	0.6590909090909092
Res + IsU + VK	0.5910345631589763	0.5886565182339831
Res	0.5115770379854887	
VK + Res	0.5635926953415217	
VK	0.6251491835412023	

When looking only at assigned switches (see Table 10), the AUC values appear to be better. However, entropy seems to be a bad predictor and residue type only does slightly better than random. Both IsUnstruct and Vkabat both seem fairly predictive, although triplet window averages do not seem to improve the results significantly. An unexpected result is that the IsUnstruct and Vkabat performed worse when combined than they did individually, but adding the qualitative descriptor RES (amino acid type) to the combination fared better than adding residues to the individual IsUnstruct and Vkabat.

Table 11: ROC AUC Values for Only the Unassigned Switches.

Descriptor Combination	Single residue value	Triplet residue value
E6 + IsU + VK	0.44086752347998026	0.45158715314424663
E6 + IsU	0.43960084033613445	0.44734205287838513
E6 + Res + IsU + VK	0.4425149630599457	0.4503505297190419
E6 + Res + IsU	0.43506599778226074	0.44783052330630185
E6 + Res + VK	0.5045957969833402	0.5062891611334518
E6 + Res	0.5028807230364324	0.5069379500607875
E6 + VK	0.5076685682222014	0.5167299034081041
E6	0.5190920962979786	0.5162314130739736
E20 + IsU + VK	0.44434443093612636	0.4523553459539619
E20 + IsU	0.44468510774739156	0.4467341785680886
E20 + Res + IsU + VK	0.44539401611200913	0.45798653324604877
E20 + Res + IsU	0.4472927883395011	0.45407626484616104
E20 + Res + VK	0.5105325914149443	0.5195029124527395
E20 + Res	0.5066039531869981	0.516408431417082
E20 + VK	0.5168760270403869	0.530434630131862
E20	0.5178078783182589	0.5316854484242027
IsU + Res	0.42355730050366736	0.42392553205702
IsU + VK	0.431681774458591	0.43251175668995734
IsU	0.43278980908738696	0.4334636477802568
Res + IsU + VK	0.4354901738119731	0.4365982084407691
Res	0.5023120933588061	
VK + Res	0.505927609517575	
VK	0.5273084193931945	

Predicting unassigned switches (see Table 11) also did not show much promise, with the best predictors doing only slightly better than random. It is notable, however, that we no longer see any values in the 0.3 to 0.4 range. Vkabat is once again the best. All of the ROC plots, along with all of the descriptor combinations, including those not displayed here, can be found in the supplemental files.

7 DISCUSSION

Looking at the tables of AUC values, predicting for unassigned switched seemed fairly difficult. We suspect that this might be because many of the blanks in the DSSP files that correspond to unassigned switches might be the result of missing data, and might not really be switches if they had known secondary structure data. The fact that all of the descriptors around the unassigned switches appear to be random supports this hypothesis. Furthermore, if this is true, it could help explain why the VKabat and IsUnstruct AUCs for assigned+unassigned switches compared to just the assigned switches is so low, since again, it is extremely likely that not all of the unassigned switches would actually be switches if we had all of the data.

We got the best results in terms of AUC when looking at switches with assigned secondary structure only. As one can see from Table 10, it did not make much difference whether averaged entropy and IsUnstruct values over neighboring residues, so anything determined about the triplet-averaged values can probably be assumed for the single values also.

Among the descriptor combinations for assigned switches, three values stood out to us, IsUnstruct, Vkabat, and the combination of all four descriptors (using E6 for entropy). IsUnstruct gave us the best AUC value outright, but the ROC plot only had three unique points on the graph, out of the 100 different thresholds we plotted. This may also have something to do with the fact that the P value of the fit is surprisingly high, meaning that there is a very good chance that a random sample would give similar results (see Table 12).

Table 12: Triplet Assigned: IsUnstruct.

	Beta	Standard Error	Z Value	$P > z $
Intercept	-2.2964	0.040	-57.098	0.000
IsUnstruct	-0.0216	0.123	-0.176	0.861

However, there is evidence that regions of disorder or regions that lend themselves to disorder under certain conditions may offer an additional alternative switch mechanism [24]. The Lobanov-Galzetskaya disorder propensity may be useful in computationally addressing the existence of such an alternative.

In a follow-up test, we made the threshold even smaller (0.001 instead of 0.01) and an additional point appeared on the graph, which brought the ROC value even higher, to 0.6939. The single residue version did even better, with an AUC of 0.6954. These values declined a little when we moved the threshold to 0.00001 (see Fig. 8), resulting in values of 0.6808 (single) and 0.6805 (triplet). These values are consistent with the lower end of the AUC values for other sequence-based prediction studies of protein residues binding to RNA [25], which have AUC values of 0.68 to 0.78, where their adding homology-based data did significantly improve AUC values in some cases.

Fig. 8: ROC plot of IsUnstruct at threshold of 0.00001.

The Vkabat from Table 10 gave us almost as good an AUC value as IsUnstruct, and it does give us a much better fit with a much lower p-value, see Table 13. Unlike the IsUnstruct, the Vkabat ROC plot had 7 unique points (see the supplemental files), with somewhat even spacing, which lends support to the notion that the number of unique points and p-value are inversely correlated.

Table 13: Triplet Assigned: Vkabat.

	Beta	Standard Error	Z Value	$P > z $
Vkabat	0.0761	0.018	4.234	0.000
Intercept	-2.5291	0.063	-40.079	0.000

In a follow-up test, we made the threshold even thinner (0.001 instead of 0.01, see Fig. 9), and another point appeared on the graph, which brought the AUC value down to 0.6365. Unlike with IsUnstruct, the AUC value did not change when the threshold was brought down to 0.00001 (also see Fig. 9, the plots are identical). This is because V_{kabat} values are discrete whereas the IsUnstruct are closer to continuous.

Fig. 9: ROC plot of V_{kabat} at threshold of 0.001.

The combination of all four descriptors (see Table 14) was interesting to us because it lets us look at all the descriptors side-by-side, including the qualitative descriptor.

Table 14: Triplet Assigned: Residue Type + E6 + IsUnstruct + Vkabat.

	Beta	Standard Error	Z Value	$P > z $
Intercept (Valine)	-2.9711	0.148	-20.022	0.000
Alanine	0.1699	0.172	0.990	0.322
Arginine	0.3294	0.179	1.837	0.066
Asparagine	0.3474	0.189	1.837	0.066
Aspartic Acid	0.4056	0.175	2.324	0.020
Cysteine	0.0942	0.317	0.297	0.766
Glutamine	0.5171	0.164	3.162	0.002
Glutamic acid	0.4653	0.185	2.511	0.012
Glycine	-0.3568	0.202	-1.765	0.078
Histidine	0.3064	0.220	1.391	0.164
Isoleucine	0.0837	0.185	0.453	0.650
Leucine	0.2121	0.160	1.323	0.186
Lysine	0.4416	0.170	2.593	0.010
Methionine	0.3012	0.240	1.254	0.210
Phenylalanine	0.4142	0.189	2.193	0.028
Proline	0.2646	0.202	1.310	0.190
Serine	0.3456	0.175	1.977	0.048
Threonine	0.0195	0.197	0.099	0.921
Tryptophan	0.5167	0.276	1.870	0.061
Tyrosine	0.2933	0.192	1.529	0.126
E6	0.4069	0.064	6.368	0
IsUnstruct	-0.1815	0.130	-1.398	0.162
Vkabat	0.0765	0.018	4.176	0.000

When we look at all four descriptors, the p-value for the IsUnstruct goes down significantly. However, this mainly applies to just the p-value for the IsUnstruct. As can be seen in Table 15, removing entropy from the calculation helps most of the p-values, (as well as increasing the AUC from 0.550 to 0.588). This time, the reducing the threshold caused the AUC values to stay pretty much the same at 0.548 (with E6) and 0.598 (without E6).

It is also interesting to note that the Vkabat p-value was not decreased when adding all of the other descriptors, and most of the other p-values are fairly small, with a couple notable exceptions like threonine at 0.921.

Table 15: P-Values with and without Entropy (E6).

Name	P-value (E6)	P-value (No E6)	Difference
Intercept (Valine)	0.000	0.000	0.000
Alanine	0.322	0.188	0.134
Arginine	0.066	0.030	0.036
Asparagine	0.066	0.033	0.033
Aspartic Acid	0.020	0.006	0.014
Cysteine	0.766	0.680	0.096
Glutamine	0.002	0.000	0.002
Glutamic acid	0.012	0.005	0.007
Glycine	0.078	0.126	-0.048
Histidine	0.164	0.107	0.057
Isoleucine	0.650	0.681	-0.031
Leucine	0.186	0.212	-0.026
Lysine	0.010	0.004	0.006
Methionine	0.210	0.099	0.111
Phenylalanine	0.028	0.028	0.000
Proline	0.190	0.108	0.082
Serine	0.048	0.020	0.028
Threonine	0.921	0.760	0.161
Tryptophan	0.061	0.074	-0.013
Tyrosine	0.126	0.144	-0.018
IsUnstruct	0.162	0.502	-0.340
Vkabat	0.000	0.000	0.000

As one can see from Table 15, IsUnstruct is a bit problematic. Where removing entropy usually caused an improvement in p-value (in fact, IsUnstruct is hurt the most by removing entropy). This is possibly related to the learning set being limited to the N-acetyltransferases which are themselves limited in number, and the test set being limited to the sirtuins which are even more limited in number.

7.1 Analyzing the Learning Set

In order for logistic regression to work, the distributions of descriptor values for switches and non-switches needs to be different. Below are box plots of the IsUnstruct values (courtesy of Jonathan Oribello). The as one can see in Fig. 10, the IsUnstruct tend to be a bit higher in switch-like regions. And because of this, the high p-value for IsUnstruct is not enough to invalidate it as a potential descriptor, although it is far from optimal.

Fig. 10: Box Plot of IsUnstruct Values from the Learning Set (Single/Assigned+Unassigned).
(Non-Switch: 0, Switch: 1).

The distributions of E6 values for switches and non-switches are more similar, which probably plays into why E6 (and by extension, E20) were not very good at telling switches and non-switches apart.

Both Fig. 10 and Fig. 11 are standard box plots, with the box being bounded above and below by the 75th and 25th percentiles, respectively. The horizontal black line through the box is the median value, while the shorter horizontal lines (“error bars”) at the top and bottom are the 75th and 25th percentile ranges plus, or minus $1.5 \times$ the interquartile range (the range between the 75th and 25th percentiles), respectively.

Fig. 11: Box Plot of E6 Values from the Learning Set (Single/Assigned+Unassigned). (Non-Switch: 0, Switch: 1).

7.1.1 Vkabat Issues

Even though it proved to be predictive of the assigned switches, Vkabat proved to be a very difficult descriptor to use. The 15 algorithms that Vkabat uses vary drastically in their robustness, with many crashing at some times and completing at others, even on the same sequence. JPred in particular was very finicky when run via python script, as were PROF, SSPPRO, YASPIN, JNET, and PSIPRED, all of which were run through Sympred. MLRC crashed on 2REE:A, although it proved fairly robust most of the time.

We had previously tried Vkabat with GenBank sequences, and immediately ran into trouble since some of the algorithms that always crashed when fed a sequence with an 'X'

residue. This, and the fact that some of the Sympred algorithms always crashed on certain sections of certain GenBank sequences, played a large part in why we eventually switched to Uniprot for our sequence data.

7.1.2 Entropy Issues

Entropy never crashed on us in the same way as Vkabat, however entropy proposed its own set of issues. Specifically, the entropy values that we calculated changed over time. To determine entropy values (both 6-term and 20-term) we used a python script that calculated said values from a blast file. (We downloaded the blast files with other python scripts: either "getBlast.py" or "getBlastUniprot.py" depending on where the FASTA of the chain came from). However, our query sequences blast outputs change over time, likely as new proteins are added to the subject databases. An example of this is that, the blast file for 4JJX:A from January 19, 2019 is slightly different from the blast file for 4JJX:A from February 21, 2019. Thus, as more proteins get added to the protein databases, it is possible that entropy will become a better predictor of switch-like behavior, although the reverse may also be true.

7.2 Future Work

Like with many research projects, time constraints prevented us from doing everything desired. Specifically, there were other descriptors that we would have liked to look at and other learning and test sets to be considered.

In the area of test sets, we would have liked to have more clusters: specifically at least around 100. It is possible that we may try something similar with all of the acetyltransferases in the future, and as more proteins get discovered, the number of N-acetyltransferases is bound to increase. Given the rarity of pyrrolysine and selenocysteine, the E22 is unlikely to make much of a difference, at least any time soon.

In and of itself, the DSSP cluster results (see Appendix B) are themselves interesting in describing possible switches for known x-ray and crystal structures and similarly

interesting are the overlays of descriptor values (see Appendix C) for the same x-ray structures.

7.2.1 *Residue Charge*

Residue Charge is a descriptor that we also looked at, but did not have time to fully explore. The Java program is fully equipped to deal with charge, and uses two types of charge values, the Amber95 and the charges of various atoms within the residue [26], which is stored in a SQLite database included with the program. The charges in this database are not exact, but are the average charge for a residue type with a given secondary structure.

Aside from being averages, the charge database has some limitations: first, it only contains charge averages for the standard 20 amino acid residues. (If one asks the program it to assign a charge to a non-standard residue, the program will print a warning instead.). Second, it only contains the charges for the following backbone atoms and atom combinations: N, NH, O, C(alpha), C(beta), and C(prime), however, users can add other atoms to the residue using our `addAtom()` method. These atom charges are used to compute the value of `averageCharge()`, but not the value of `amber95()`, which is listed in the program's database.

We actually made two versions of the logistic regression script. The script "logistic-switch2.py" is the version we used for our calculation, but "logistic-switch3.py" is an advanced version that contains the ability to do a logistic regression on the amber95 as well as the descriptors in "logistic-switch2.py." It is not recommended to use "logistic-switch3.py" without modification, as doing a logistic regression on the amber95 causes the script to crash, as the values fail to converge. For anyone who can get this script to work, the output parser can also handle this, just set the boolean parameter of `CSVParser.readFile()` to true. All code is available upon request.

8 CONCLUSIONS

The following are the most important results of this work. First, the clustering of possible secondary structure switches is, by itself, a useful algorithm, and the logistic regression results suggest that prediction of switch-like behavior is doable. Based on ROC values alone, isUnstruct made the best prediction. In the case of assigned switches, however, the Vkabat consistently gave us better-than-average results, and was the best descriptor option overall. The overall trend of low-confidence ROC values may be because of the small size of the data set. A larger data set (or precision-recall) may give better results.

As stated in 7.2, residue charge may also be useful as a descriptor, and over time entropy may become a better descriptor as more proteins are catalogued, which will allow for better BLAST results.

Literature Cited

- [1] B. Strauss, “Amino Acid Diagrams for SJSU Masters Thesis” 2019.
- [2] CrashCourse, “DNA, Hot Pockets, & The Longest Word Ever: Crash Course Biology #11.” <https://www.youtube.com/watch?v=itsb2SqR-R0>, April 2012.
- [3] J. D. Watson, T. A. Baker, S. P. Bell, A. Gann, M. Levine, and R. Losick, *Molecular Biology of the Gene*, ch. 5, pp. 75–76. Pearson Education, sixth ed., 2008.
- [4] Unknown, “Auspep.” <http://www.auspep.com.au/AminoAcidCodes>.
- [5] Professor Dave Explains, “Protein structure.” <https://www.youtube.com/watch?v=EweuU2fEgJw>, August 2016.
- [6] “Cell Biology by the Numbers.” <http://book.bionumbers.org/what-is-the-energy-of-a-hydrogen-bond/>.
- [7] Various, “RCSB Protein Data Bank.” <https://www.rcsb.org>.
- [8] J.-H. Ha and S. N. Loh, “Protein conformational switches: From nature to design,” *Chemistry*, pp. 7984–7999, June 2012.
- [9] E. V. Filippova, M. L. Khun, *et al.*, “A novel polyamine allosteric site of SpeG from *Vibrio cholerae* is revealed by its dodecameric structure,” *Journal of molecular biology*, pp. 1316–1334, 2015.
- [10] Various, “DSSP.” <https://swift.cmbi.umcn.nl/gv/dssp/>.
- [11] D. Rose, R. Nepal, R. L. R. Mishra, S. Gholizadeh, and B. Lustig, “Novel application of qualitative predictors: First stage calculation of solvent accessible residues using complementary protein sequence entropy and other parameters,” in *Proceedings of the 22nd International Workshop on Database and Expert Systems Applications*, pp. 70–74, 2011.
- [12] H. Liao, W. Yeh, D. Chiang, R. Jernigan, and B. Lustig, “Protein sequence entropy is closely related to packing density and hydrophobicity,” *Protein Engineering, Design & Selection*, vol. 0, pp. 1–6, 2005.

- [13] W. S. J. Valdar, “Scoring Residue Conservation,” *PROTEINS: Structure, Function, and Genetics*, p. 227–241, 2002.
- [14] M. Y. Lobanov and O. V. Galzitskaya, “The Ising model for prediction of disordered residues from protein sequence alone,” *Physical Biology*, June 2011.
- [15] M. Y. Lobanov, I. V. Sokolovsky, and O. V. Galzitskaya, “IsUnstruct: prediction of the residue status to be ordered or disordered in the protein chain by a method based on the Ising model,” *PLOS Computational Biology*, 2010.
- [16] S. Monteroso, J. Oribello, A. Huynh, J. Graham, N. Wong, M. L. Khun, and B. Lustig, “Sequence entropy and other descriptors identify a possible conformational switch: Case study of an allosteric dodecameric polyamine acetyltransferase speg.” 62nd Annual Meeting Biophysical Society, 2018.
- [17] K. M. Saravanan and S. Selvaraj, “Performance of Secondary Structure Prediction Methods on Proteins Containing Structurally Ambivalent Sequence Fragments,” *Peptide Science*, vol. 100, 2012.
- [18] R. Nepal, J. Spencer, G. Bhogal, A. Nedunuri, T. Poelman, T. Kamath, E. Chung, K. Kantardjieff, A. Gottlieb, and B. Lustig, “Logistic Regression Models to Predict Solvent Accessible Residues Using Sequence and Homology-Based Qualitative and Quantitative Descriptors Applied to a Domain-Complete X-ray Structure Learning Set,” *Journal of Applied Crystallography*, 2015.
- [19] D. W. Hosmer Jr., S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*. Hoboken: Wiley, 2013.
- [20] M. H. Kutner, C. J. Nachtsheim, and J. Neter, *Applied Linear Regression Models*. Boston: McGraw Hill, fourth ed., 2004.
- [21] J. G. Graham, “Computational characterization of sequence and structural indicators,” Master’s thesis, San Jose State University, 2018.
- [22] T. Hrabe, Z. Li, M. Sedova, P. Rotkiewicz, L. Jaroszewski, and A. Godzik, “Pdbflex: exploring flexibility in protein structures,” *Nucleic Acids Res.*, January 2016.
- [23] H. Hwang, D. Petrey, and B. Honig, “A hybrid method for protein–protein interface prediction,” *The Protein Society*, 2015.

- [24] D. Kovacs, B. Szabo, R. Pancsa, and P. Tompa, “Intrinsically disordered proteins undergo and assist folding transitions in the proteome,” *Archives of Biochemistry and Biophysics*, pp. 80–89, 2013.
- [25] L. C. Xue, D. Dobbs, A. M. Bonvin, and V. Honavar, “Computational prediction of protein interfaces: A review of data driven methods,” *FEBS Letters*, pp. 3516–3526, 2015.
- [26] A. Thomas, A. Milton, and R. Brasseur, “Partial atomic charges of amino acids in proteins,” *Proteins.*, pp. 102–109, July 2004.

Appendix A

COMPLETE LEARNING SET

Table 16: Learning Set Clusters.

Dominant Chain	Other Chains
1CJWA	1KUYA, 1KUXA, 1KUVA, 1IB1E, 1IB1F, 1IB1G, 1IB1H, 1B6BA, 1B6BB, 1L0CA
1GX3A	1GX3B, 1GX3C, 1GX3D, 1W6FC, 1W6FD, 1W6FA, 1W6FB, 1W5RA, 1W5RB
1IB1A	1iB1B, 1iB1C, 1iB1D, 4BG6A, 4BG6B, 3NKXA, 3NKXB, 3CU8A, 3CU8B, 2WH0A, 2WH0B, 2WH0C, 2WH0D, 1QJBA, 1QJBB, 1QJAA, 1QJAB, 1A4OA, 1A4OB, 1A4OC, 1A4OD, 1A38A, 1A38B, 1A37A, 1A37B, 5XY9A, 5XY9B, 5WXNA, 5WXNB, 5ULOA, 5ULOB, 4WRQA, 4WRQB, 4HKCA, 3RDHA, 3RDHB, 3RDHC, 3RDHD, 2V7DA, 2V7DB, 2V7DC, 2V7DD, 2C1NA, 2C1NB, 4ZDRA, 4ZDRB, 6FNCA, 6FNCB, 6FNBA, 6FNBB, 6FNAA, 6FNAB, 6FN9A, 6FN9B, 6F09P, 6F09Q, 6F09R, 6F09S, 6F08A, 6F08B, 6F08I, 6F08J, 6EWWA, 6EWWB, 6EWWC, 6EWWD, 6EJLA, 6EJLB, 5NASA, 5NASB, 5M37A, 5M37B, 5JM4A, 5JM4B, 5J31A, 5J31B, 5EXAA, 5EXAB, 5EWZA, 5EWZB, 5D3FA, 5D3FB, 5D2DA, 5D2DB, 4N7YA, 4N7YB, 4N7GA, 4IHLA, 4IHLB, 4FJ3A, 4FJ3B, 2O02A, 2O02B, 5M36A, 5M36B, 5M35A, 5M35B, 4N84A, 4N84B
1N71A	1N71B, 1N71C, 1N71D, 5E96A, 2A4NA, 2A4NB, 1B87A
1S7FA	1S7NA, 1S7NB, 1S7NC, 1S7ND, 1S7LA, 1S7KA, 1Z9UA, 1Z9UB
1TIQA	1TIQB
1XEBA	1XEBB, 1XEBC, 1XEBD, 1XEBE, 1XEBF, 1XEBG, 1XEBH,
2B5GA	2B5GB, 2B4DA, 2B4DB, 2B3UA, 2B3UB, 2B3VA, 2B4BA, 2B4BB, 2B58A, 2F5IA, 2F5IB, 2FXFA, 2FXFB, 2G3TA, 2G3TB, 2JEVA, 2JEVB, 3BJ7A, 3BJ7B, 3BJ7C, 3BJ7D, 3BJ8A, 3BJ8B, 3BJ8C, 3BJ8D

2BL1A	2J8MA, 2J8MB, 2J8NA, 2J8NB, 2J8RA, 2J8RB, 1YVOA, 1YVOB
2FIAA	2FIAB
2JLMA	2JLMB, 2JLMC, 2JLMD, 2JLME, 2JLMF
2Q4VA	2Q4VB, 2BEIA, 2BEIB
2REEA	2REEB, 2REFA, 2REFB
2VFBA	2VFCA, 2VFCB, 3LTWA, 4B55A, 4C5PA
2VI7A	2VI7B, 2VI7C
2WPWA	2WPWB, 2WPWC, 2WPWD, 2WPXA, 2WPXB
3FS8A	3FS8B, 3FSBA, 3FSBB, 3FSCA, 3FSCB
3MQHA	3MQHB, 3MQHC, 3MQHD, 3MQHE, 3MQHF
3N37A	4M1FA, 3N3BA, 3N3BB, 3N3AA, 3N3AB, 3N39A, 3N39B, 3N38A, 2R2FA, 2R2FB, 2BQ1I, 2BQ1J, 1R2FA, 1R2FB
3NE7A	3K9UA, 3K9UB, 3FIXA, 3FIXB, 3FIXC, 3FIXD, 3F0AA
3QB8A	3QB8B
3SMAA	3SMAB, 3SMAC, 3SMAD
3TFYA	3TFYB, 3TFYC, 4X5KA, 2PSWA, 2PSWB, 2PSWC, 2OB0A, 2OB0B, 2OB0C
3W6SA	3W6SB, 3W6SC, 3W91A, 3W91B, 3W91C, 3W6XA, 3W6XB, 3W6XC, 3W6XD, 3W6XE, 3W6XF, 3W6XG, 3W6XH, 3W6XI, 3W6XJ, 3W6XK, 3W6XL
4AVAA	4AVBA, 4AVBB, 4AVCA, 4AVCB
4EABA	4EAAA, 4EA9A, 4EA8A, 4EA7A
4FD4A	4FD4B, 5YAGA, 5YAGB
4FD5A	4FD6A
4FD7A	4FD7B, 4FD7C, 4FD7D
4HNYA	4HNYC, 4HNXA, 4HNWA, 4XPDA, 4XNHA, 4Y49A, 4Y49G, 4Y49M
4J3GA	4J3GB, 4J3GC, 4J3GD, 4JWPA, 4JWPB

4JJXA	4R57A, 4R57B, 4R57C, 4R57D, 4R57E, 4R57F, 4R57G, 4R57H, 4R57I, 4R57J, 4R57K, 4R57L, 4R87A, 4R87B, 4R87C, 4R87D, 4R87E, 4R87F, 4R87G, 4R87H, 4R87I, 4R87J, 4R87K, 4R87L, 4JJXB, 4JJXC, 4NCZA, 4NCZB, 4NCZC, 6CX8A, 6CX8B, 6CX8C, 6CX8D, 6CX8E, 6CX8F, 5CNPA, 5CNPB, 5CNPC, 5CNPD, 5CNPE, 5CNPF, 4MJ8A, 4MJ8B, 4MJ8C, 4JLYA, 4JLYB, 4JLYC, 4JLYD, 4JLYE, 4JLYF, 4MI4A, 4MI4B, 4MI4C, 4MHDA, 4MHDB, 4MHDC, 5UG4A, 5UG4B, 5UG4C, 4YGOA, 4YGOB, 4YGOC, 4YGOD, 4YGOE, 4YGOF, 6DAUA, 6DAUB, 6DAUC, 6DAUD, 6DAUE, 6DAUF
4KVOA	4KVOB, 4KVOC, 4KVOD, 4KVMA, 4KVMB, 4KVMC, 4KVMD
4KVXA	4KVXB
4PV6A	4PV6B, 4PV6C, 4PV6D, 4PV6E, 4PV6F, 4PV6G, 4PV6H, 4PV6I, 4PV6J, 4PV6K, 4PV6L, 4PV6M, 4PV6N, 4PV6O, 4PV6P
4R3KA	4R3LA, 5C88A, 5C88B, 4LX9A, 2X7BA
4R9MA	4R9MB, 4R9MC, 6CY6A, 3WR7A, 3WR7B, 3WR7C, 3WR7D
4RI1A	4RI1B, 4RI1C
4U9VB	4U9WA, 4U9WB, 4U9WC, 4U9WD
4UA3A	4UA3B
5GI5A	5GI8A, 5GI7A, 5GI6A, 3TE4A, 5GIHA, 5GIGA, 5GIFA, 5GIHA, 3V8IA, 3V8IB
5HGZA	5HH0A, 5HH1A, 5ICWA, 5ICWB, 5ICWC, 5ICWD
5JPHA	5JPHB, 5JPHC, 5JQ4A, 5JQ4B
5K04A	5K18A, 5K18C
5K9NA	5K9NB
5KTAA	5KTCA, 5KTDA
5WIFA	5WIFB, 5WIFC, 6D72A, 6D72B, 6D72C
5YGEA	5YGEB

Appendix B

FASTAS AND SWITCHES

In this appendix the layout is as follows:

Table 17: Key.

Term	Definition
1st	The primary structure or residue sequence. Determined by our alignment algorithm comparing the Uniprot and DSSP sequences
2nd(d)	The raw secondary structure of the sequence, as listed in the DSSP files. ‘*’ means that the residue was skipped in the DSSP files
2nd(s)	The secondary structure simplified to S=Sheet, H=Helix, O=Other, and D=Not Listed in DSSP.
Switch	0 = no switch 1 = assigned switch ? = unassigned switch
[#]	How many residues precede the below line of residues in the sequence, according to the DSSP file.

B.1 Learning Set

>1CJW:A

```
[0]
1st      : MSTPSVHCLKPSPHLPSGIPGSPGRQRRHTLPANEFRCLTPEDAAGVFE
2nd(d)   : *****????SSEE????GGGHHHHHH
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDOOOOOOSSSOOOHHHHHHHHH
Switch:  00000000000000000?????????????100000000011111
```

```
[50]
1st      : IERAFISVSGNCPLNLDEVQHFTLCPELSLGWFEVGRLVAFIIGSLWD
2nd(d)   : HHIIIIIIHHHS??S??HHHHHHHHHHH?GGGEEEEEEETEEEEEEEEEEEE??
2nd(s)   : HHHHHHHHHHHOOOOOOHHHHHHHHHHHOHHHSSSSSSSOOSSSSSSSSSSSO
Switch:  1111111111?????1111110000000010000100100000011000
```

```
[100]
1st      : EERLTQESLALHRPRGHS AHLHALAVHRSFRQQGKGSVLLWRYLHHVGAQ
2nd(d)   : SSS??GGGGG???TT??EEEEEEEEEE?TTSTTSSHHHHHHHHHHHHHHTS
2nd(s)   : OOOOOHHHHHOOOOOOOSSSSSSSSSSOOOOOOOHHHHHHHHHHHHHHHOO
Switch:  0000011111000000000011000011111100000000000000110
```



```
[199]
1st   : PGSHFVTGLTVAVVTDDARYNLRGRNLAVHRSGATEHIRFDSAAQVLDAI
2nd(d): TT?HHHHS?EEEE?SSEEEEEETEEEEETEEEEESSHHHHHHII
2nd(s): OOOHHHHOOSSSSSOOOSSSSSSOOSSSSSSOOSSSSSSOHHHHHHHH
Switch: 00000000000000001000000000001001000001001000000
```

```
[249]
1st   : VNRFGIDLGDLAGRVDQARVAEVLD
2nd(d): IIIS???GGGGTTS?HHHHHHH?TT
2nd(s): HHHOOOOHHHHOOOOHHHHHHHHOOO
Switch: 00100001111000000000000000
```

>1IB1:A

```
[-8]
1st   : MDKNELVQKAKLAEQAERYDDMAACMKSVTEQGAELSNEERNLLSVAYKN
2nd(d): *?HHHHHHHHHHHHHTT?HHHHHHHHHHHHHTT????HHHHHHHHHHHH
2nd(s): DOHHHHHHHHHHHHHOOHHHHHHHHHHHHHOOOOOOHHHHHHHHHHHH
Switch: ?1110000000000110000000000000110?000111?100000000
```

```
[42]
1st   : VVGARRSSWRVVSIEQKTEGAEEKQQMAREYREKIETELRDICNDVLSL
2nd(d): HHHHHHHHHHHHHHHHHH?SSSTTHHHHHHHHHHHHHHHHHHHHHHH
2nd(s): HHHHHHHHHHHHHHHHHHOOOOOOHHHHHHHHHHHHHHHHHHHHHH
Switch: 000000001?10000111????1111110000000000000000000000
```

```
[92]
1st   : LEKFLIPNASQAESKVFYLMKGDYYRYLAEVAAGDDKKGIVDQSQQAYQ
2nd(d): IIIIITTT??SHHHHHHHHHHHHHHHHHHHHHH??TTHHHHHHHHHHH
2nd(s): HHHHHOOOOOHHHHHHHHHHHHHHHHHHHHHHHOOOOHHHHHHHHHHHH
Switch: 000001110??11111?1000000000000110?11111100000001?
```

```
[142]
1st   : EAFEISKKEMQPTHPIRLGLALNFSVFYYEILNSPEKACSLAKTAFDEAI
2nd(d): HHHHHHHHHS?TT?HHHHHHHHHHHHHIIIIIS??HHHHHHHHHHHH
2nd(s): HHHHHHHHHHOOOOOHHHHHHHHHHHHHHHHHHHHHOOOHHHHHHHHHH
Switch: 100011111?????111000000001111?11?????1110000000111
```

```

[192]
1st      : AELDTLSEESYKDSLIMQLLRDNLTLWTSDTQGDEAEAGEGGE
2nd(d)   : HHHHTT?TT???S?HHHHHHHHHHHHH?*****
2nd(s)   : HHHHOOOOOOOOOHHHHHHHHHHHHHODDDDDDDDDDDDDDDDD
Switch: 11111??1111111?1000000011111????000?????????

>1N71:A

[0]
1st      : MIISEFDRNNPVLKQDLSDLLRLTWPEEYGDSSAEVEEMNPERIAVAA
2nd(d)   : ?EEEE??TT?HHHHHHHHHHHHHHH?TTSSSTHHHHHHHHHT?TTSEEEEE
2nd(s)   : OSSSSOOOOOHHHHHHHHHHHHHHHOOOOOOOHHHHHHHHHOOOOOSSSSS
Switch: 0100000000000000000000011011110000000001000000000

[50]
1st      : VDQDELVGFIGAIPQYGITGWELHPLVVESSRRKNQIGTRLVNYLEKEVA
2nd(d)   : EETEEEEEEEEEEEEETTTEEEEEEEEE?TTS?SSSHHHHHHHHHHHHHH
2nd(s)   : SSOOSSSSSSSSSSSSSOOSSSSSSSSSOOOOOOOHHHHHHHHHHHHHHH
Switch: 000000000000000000000000000000011011100000000000000

[100]
1st      : SRGGITIYLGTDLDHGTTLTSLQTDLYVHTFDKVASIQNLREHPYEFYEKL
2nd(d)   : HTT??EEEEEEE?SSS?BTSSS?TTSSHHHHHHT??BSS??THHHHHHT
2nd(s)   : HOOOSSSSSSSSSOOOOOSOOOOOOOOOHHHHHHHOOOSOOOOOHHHHHHO
Switch: 0000000000010000????????????????????????0010000010

[150]
1st      : GYKIVGVLPNANGWDKPDIIWMAKTIIPRPDS
2nd(d)   : T?EEEEETTTTSTT??EEEEEEE?S????*
2nd(s)   : OOSSSSSSOOOOOOOOOSSSSSSSOOOOOOD
Switch: 00000000000000000000000000000000??

>1S7F:A

[-8]
1st      : GLVPRGSHMVEIIPVSTTLELRAADESHVPALHQLVLKKNKAWLQQSLDWP
2nd(d)   : ??EEETEEEE?EEE?SSSEEEEE??GGGHHHHHHHHHHHHHHHTTSSS??
2nd(s)   : OOSSSOSSSSOSSSSOOOSSSSSSSOHHHHHHHHHHHHHHHHHOOOOOOO
Switch: 00000000110000110000000000000000000001????1????11

```

```
[42]
1st      : QYVTSQEETRKHVQGNILLHQRGYAKMYLIFCQNEMAGVLSFNAI EP INK
2nd(d)   : ***??HHHHHHHHHHHHHHHHHHTS?EEEEEEETTEEEEEEEEEEEETTTT
2nd(s)   : DDDOOHHHHHHHHHHHHHHHHHHHOOOSSSSSSSOOSSSSSSSSSSSSSOOOO
Switch:  ????????1000000000000100000000000000000000000001110
```

```
[92]
1st      : AAYIGYWLDESFQGQGIMSQSLQALMTHYARRGDIRRFVIKCRVDNQASN
2nd(d)   : EEEEEEEE?GGGTSSHHHHHHHHHHHHHHTTS??EEEEEEETT?HHHH
2nd(s)   : SSSSSSSSOHHHOOOOHHHHHHHHHHHHHHHHHOOOOSSSSSSSSOOHHHH
Switch: 000000000000000000000000000000000000000000000000000
```

```
[142]
1st      : AVARRNHFTLEGCMKQAEYLNQDYHDVNMARIIDA
2nd(d)   : HHHHHTT?EEEEEEEEEEEEETEEEEEEEEEEEE?**
2nd(s)   : HHHHHOOOSSSSSSSSSSSSSOOSSSSSSSSSSSSODD
Switch:  0000000000000000000100100000000000??
```

>1TIQ:A

```
[0]
1st      : MSVKMKKCSREDLQTLQQLSIETFNDTFKEQNSPENMKAYLESAFNTEQL
2nd(d)   : *?EEEEEE?G GGGHHHHHHHHHHHHHHHHHHSTTS?HHHHHHHHHHHHSSHHHH
2nd(s)   : DOSSSSSOHHHHHHHHHHHHHHHHHHHHHHOOOOOHHHHHHHHHHHHOOHHHH
Switch: 00000000000000000000000000000001????????1000000011000000
```

```
[50]
1st      : EKELSNMSSQFFFIYFDHEIAGYVKVNIDDAQSEEMGAESLEIERIYIKN
2nd(d)   : HHHHHH?TTEEEEEEEETTTEEEEEEEEEEGGGSSS???TTEEEEEEEEE?G
2nd(s)   : HHHHHOOSOSSSSSSSOOSSSSSSSSSSH HHOOOOOOOOSSSSSSSSSOH
Switch:  000000000000000000000000000000000000000000000000000
```

```
[100]
1st      : SFQKHGLGKHL LNKAIEIALERNKKNIWLG VWEKNENAI AFYKKMGFVQT
2nd(d)   : GG?SSSHHHHHHHHHHHHHHHTT?SEEEEEEEETT?HHHHHHHHHHTT?EEE
2nd(s)   : HHOOO OHHHHHHHHHHHHHHHHHO OOOOSSSSSSSO OOH HHHHHHHHHHOOOSSS
Switch: 0000000000000000000000000000000000000000000000000000000
```

[150]
1st : GAHSFYMGDEEQTDLIMAKTLIL
2nd(d): EEEEEETTEEEEEEEEEEEE??
2nd(s): SSSSSSSOOSSSSSSSSSSSSOOO
Switch: 0000000000000000000000

>1XEB:A

[0]
1st : MSLDWTCKHHADLTCLKELYALLQLRTEVFVVEQKCPYQEV DGLDLVG DTH
2nd(d): *??EEEEEGGG??HHHHHHHHHHHIIIIIIITT??S??S?TTSTT?E
2nd(s): DOOSSSSSHHHOOHHHHHHHHHHHHHHHHHHHHOOOOOOOOOOOOOOOOOS
Switch: 0?000000011100000000000000000000000000000000000000

[50]
1st : HLMAWRDGQLLAYLRLLDPVRHEGQVVIGRVVSSSAARGQGLGHQLMERA
2nd(d): EEEEEETTEEEEEEEEE?STTTTT?EEEEEEEE?GGGTTSSHHHHHHHHH
2nd(s): SSSSSSOOSSSSSSSSSOOOOOOOOSSSSSSSSSOHHHOOOOHHHHHHHHH
Switch: 0000010010000000001111000000000001110?1110000000

[100]
1st : LQAAERLWLDTPVYLSAQAH LQAYYGRYGFVAVTEVYLEDDIPHIGMRR
2nd(d): HHHHHHHHTT?EEEEESTTHHHHHTTTEEE?S??EEETEEEEEEEE
2nd(s): HHHHHHHHOOOOSSSSSSOOHHHHHOOOSSSOOOOSSSOOSSSSSSSS
Switch: 00000001000000000011111101101001111011??11000001

>2B5G:A

[0]
1st : MAKFVIRPATAADCSDILRLIKELAKYEYMEEQVILTEKDLLEDGFGEHP
2nd(d): *???EEEE??GGGHHHHHHHHHHHHT?***?????HHHHHHHHSSSS?
2nd(s): DOOSSSSSOHHHHHHHHHHHHHHHHHOODDDDOOOOOHHHHHHHHHOOOO
Switch: 0??0000000000000000000000111??110000110000001????

[50]
1st : FYHCLVAEVPKEHWTPEGHSIVGFAMY YFTYDPWIGKLLYLEDDFFVMSDY
2nd(d): S?EEEEEE??GGG??TT??EEEEEEEEEEEEETTTEEEEEEEEEEEE?GGG
2nd(s): OOSSSSSSOOHHHOOOOOOOSSSSSSSSSSSSSOOSSSSSSSSSSSOHHH
Switch: 000000000111??0000000000000000000000000000000000

>2FIA:A

[0]

1st : MKIRVADEKELPMILQFLTEVKAYMDVVGITQWTKDYPSQGDIQEDITKK
2nd(d): ??EEE??GGGTTHHHHHHHHHHHHHHHHT????SSSS?HHHHHHHHHTT
2nd(s): OOSSSOOHHHOHHHHHHHHHHHHHHHHHOOOOOOOOOOHHHHHHHHHOO
Switch: 00000000001100

[50]

1st : RLYLLVHEEMIFSMATFCMEQEQDFVWLKRFATSPNYIAKGYGSLLFHEL
2nd(d): ?EEEEETTEEEEEEEEEEE?TT?SEEEEEEEEE?GGGTTHHHHHHHHHH
2nd(s): OSSSSSOOSSSSSSSSSSSOOOOSSSSSSSSSOHHHOOOOHHHHHHHHH
Switch: 0000011001110000001000000000000000001000000000000

[100]

1st : EKRAVWEGRRKMYAQTNHTNHRMIRFFESKGFTKIHESLQMNRLDFGSFY
2nd(d): HHHHHTTT??EEEEEEETT?HHHHHHHHHTT?EEEEEE??TT?GGG??EE
2nd(s): HHHHHOOOOOSSSSSSSOOOHHHHHHHHHOOSSSSSSSOOOOHHHOOSS
Switch: 0000000000000000000000000000010000000000000000000

[150]

1st : LYVKELENQSI
2nd(d): EEEEE??****
2nd(s): SSSSSOODDDD
Switch: 0000000??00

>2JLM:A

[0]

1st : MFSPSTTTLFRFVECTEDQHALEILEILNDAIINSTALYDYKPRSKESMA
2nd(d): *****??EEEEE??HHHHHHHHHHHHHHHHHHH?SSS??SS??HHHHH
2nd(s): DDDDDOOOOOSSSSSOHHHHHHHHHHHHHHHHHHHOOOOOOOOOOHHHHH
Switch: ?????1111100000000000000000000000000000000000011000

[50]

1st : AWFATKRQNNFPIIGAVNEVGQLLGFAWGSFRAFPAYKYTVEHSVYIHK
2nd(d): HHHHHHHHTT??EEEEEEETTS?EEEEEEEEESSSSGGGTTEEEEEEEEE?T
2nd(s): HHHHHHHHOOOOSSSSSSSOOOOSSSSSSSSSOOOHHHOOSSSSSSSSSOO
Switch: 0000000110000000010000000000000000000000000000001


```
[0]
1st      : MLELINRYEHGFVYIPVILACREKGLFELIKKKRITHRQIANTLGANTGN
2nd(d)   : *****
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 00000000000000000000000000000000000000000000000000000
```

```
[100]
1st      : LGKEKPGIIKNWIELSQQQWNINDRTIADLLDGVLIIPIMLGLHKHNLLA
2nd(d)   : *****
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 000000000000000000000000000000000000000000000000000
```

```
[200]
1st      : CLTDIGKCLGQRTIIKSISLNCFENNYYNLRHPKIEDLRDLIALETLCWS
2nd(d)   : *****??EEEE???GGGHHHHHHHHHHHS?
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDOOSSSSOOOHHHHHHHHHHHHHOO
Switch:  000000000000000000000000000000000000000000000000
```

59

[300]
1st : TQVPLLHTESGVVVQLLAVNILPELQNQGLGDRLLLEFMLQYCAQISGVEK
2nd(d): TTGGGG??TT?SEEEEEEEEE?GGG?SSSHHHHHHHHHHHHTTSTT??E
2nd(s): OOHOOHHOOOOOOSSSSSSSSSOHHOOOOHHHHHHHHHHHHHHOOOOOOOS
Switch: 001000000000000000000000111000000000000000010000000

[350]
1st : VVAVTLCRNYPDYSMPMAEYIHQKNESGLLVDP LLRFHQIHGAKIEKLL
2nd(d): EEEEE??SSGGGTTTS?HHHHTT?B?TTS?BSSHHHHHHHHHTT?EEEEEE
2nd(s): SSSSSOOOOHHHOOOOHHHHOOOSOOOOO SOHHHHHHHHHHOOOSSSSSS
Switch: 00000000000010000000000000000000000000000001001000000

[400]
1st : PGYRPKDWENQTCGVLVSYDIQHRQRF DGATVEKNRQTKVKLTEDIDEVV
2nd(d): TTS?TT?GGGTT?EEEEEE??TT?*****
2nd(s): OOOOOOHHHOOOSSSSSSSOOOO ODDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 000000011100

[450]
1st : ASCVRKVMHQKKSFDAQRALMEMGIESLELLELRYLLQKKLGVEIDPNLF
2nd(d): *****
2nd(s): DDD
Switch: 000

[500]
1st : FKYGTAAAIASYFKGETTERS GEVSEVLPKPEAIKTD SGFSTAQLENGVA
2nd(d): *****
2nd(s): DDD
Switch: 000

[550]
1st : IIGMACRFPGDADSPQQYWSLLHDGIDGITEVPPTRWDIEQYYHPEKNQP
2nd(d): *****
2nd(s): DDD
Switch: 000

[600]
1st : GKIANRYGGFLTEVDKFEPDFFRISAREALYMDPQQRLLLEHWKALEDA
2nd(d): *****
2nd(s): DDD
Switch: 000

[illegible][illegible][illegible][illegible]

```
[850]
1st      : YVEAHGTGTS LGDPVEIKAIEAVYGKDRSADRPLMIGSVKTNIHTEAAA
2nd(d)   : *****
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 000000000000000000000000000000000000000000000000000
```

[illegible][illegible]

[illegible]

```
[1750]
1st      : STKIFPGQFYYSVFDLHLKNNIKWDESQGDLNEYVANANTIYINKITEDVI
2nd(d)   : *****
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 00000000000000000000000000000000000000000000000000000
```

[illegible][illegible][illegible][illegible]

```
[2000]
1st      : QTYNLNLKATKLYDYPTLLELSGYIAQILSSQGTKPISSSSQTQQSLKTL
2nd(d)   : *****
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch:   000000000000000000000000000000000000000000000000000
```


>2VFB:A

[0]

1st : MALDLTGYLDRINYRGATDPTLDVLRDLVSAHTGAI AFENLDPLMGVPVD
2nd(d): ***?HHHHHHHTT??S??SSSHHHHHHHHHHHHS?EE?HHHHTT???
2nd(s): DDDOHHHHHHHHOOOOOOOOOHHHHHHHHHHHHHHHOOSSOHHHHOOOOOO
Switch: 00?01000000000?00000000000000000000000000000000

[50]

1st : DLSAEALADKLVDRRRGGYCYEHNGLIGYVLAELGYRVRRLAGRVVWLAP
2nd(d): ??SHHHHHIIIIIT??B?HHHHHHHHHHHHHHHT?EEEEEEEE?TT??
2nd(s): OOOHHHHHHHHHHOOOOOOSHHHHHHHHHHHHHHHOSSSSSSSSSOOOOO
Switch: 000

[100]

1st : PDAPTPAQTHTVLAVTFPGCQGPYLV DVGFGGMTPTAPLRLETGTVQOTA
2nd(d): TTS????S?EEEEEEE?TT?SS?EEE?S?SGGG??SS?EES?BS??EE?S
2nd(s): OOOOOOOOOSSSSSSSOOOOOOOSSSOOOHHHOOOOOSSOOSOOOSSOO
Switch: 0000000110000000000000000000000001110000000010010100

[150]

1st : LEPYRLDDRGDGLVLQAMVRDEWQALYEFSTLTRPQVDLRVGSWFVSTHP
2nd(d): SSEEEEEEEETEEEEEEEEETEEEEEEEE??S??HHHHHHHHHHHHH?T
2nd(s): OOSSSSSSSOOSSSSSSSSOOSSSSSSSSOOOOOHHHHHHHHHHHHHO
Switch: 001000000000000000100100000000000000000000000000

[200]

1st : TSHFVTGLMAATVADDARWNL MGRNLAIHRRGGTEKILLEDAAAVVDTLG
2nd(d): T?HHHHS?EEEE?SS?EEEEETEEEEEEETEEEEEE??SHHHHHHHHH
2nd(s): OOH HHHOSSSSSSOOOOSSSSSSOOSSSSSSOOSSSSSSOOHHHHHHHHHH
Switch: 000000000000010010000000000001001100000000000000

[250]

1st : DRFGINVADVGERGRLEARIDKVCFGAEN
2nd(d): HTT??G GGS?HHHHHHHHHHH??****
2nd(s): HOOOOHHHHHOHHHHHHHHHHHHHOODDDD
Switch: 100000000000000000000001100000

>2VI7:A

[0]
1st : MTAEPTIRLERYSERHVEGLTALYNDPAVARQVLQMPYQSVEQRRKRLH
2nd(d): *****?EEEE?GHHHHHHHTSHHHHTTSS??SS??HHHHHHHT
2nd(s): DDDDDDOSSSSSOHHHHHHHHHHHHOOHHHHOOOOOOOOHHHHHHHO
Switch: 0000??010000000000000000000000000000000000000001

[50]
1st : DSADDDRLLILVALHQGDVIGSASLEQHPRIRSHSGSIGMGVAVAWQ GK
2nd(d): T?*??TTEEEEEETTEEEEEEEEE?SSGGTTEEE?TT??EESSTTT
2nd(s): OODOOOSSSSSSSOSSSSSSSSSSSOOHHHOSSSSOOOOOSSSOOOO
Switch: 1000

[100]
1st : GVGSRLLGELLDIADNWMNLRRVELTVYTDNAPALALYRKFGFETEGEMR
2nd(d): THHHHHHHHHHHIIIIIS??SEEEEEETTHHHHHHHHTT?EEEEEE
2nd(s): OHHHHHHHHHHHHHHHHOOOSSSSSSSOOHHHHHHHHHHOOOSSSSSS
Switch: 0000000000000001000000000000000000000000000000

[150]
1st : DYAVRDGRFVDVYSMARLRRVEGRVG
2nd(d): EEEEEETEEEEEEEEEE??*****
2nd(s): SSSSSOOSSSSSSSSSSSSOODDDDDD
Switch: 000000000000000000000000000000

>2WPW:A

[0]
1st : MNDTAGELEFVPLAANDDETVGQWLDLMALAAETGPRAAPPCNVDMVGSL
2nd(d): ***??SSEEEEE??TT?HHHHHHHHHHHHHHHTSSS????HHHHHTT
2nd(s): DDDOOOSSSSSOOOOHHHHHHHHHHHHHHHHOOOOOOOOHHHHHHOO
Switch: 000??010000000000000000000000010000000000000011

[50]
1st : RFAPPATALDDWVVRSGGRVVGALRLALPDGAPTARVDQLLVHPGRRRRG
2nd(d): T???TTEEEEEEEEEETTEEEEEEEEEESS?SEEEEEEEEE?TTS?SSS
2nd(s): OOOOOOSSSSSSSSSSSOSSSSSSSSSSSOOOOSSSSSSSSSOOOOOOOO
Switch: 100000000000000001000000000000000000000001110000


```
[50]
1st      : SILGEYLVDFYNDRIKKHPLIIGENALIRTENVIYGDTHIIGDNFQTGHK
2nd(d)   : ?EEEE??TTHHHHT?????EEE?TT?EE?TT?EESS?EE?TT?EE?SS
2nd(s)   : OSSSSOOOOHHHHOOOOOOOSSSSOOOOSSOOOOSSSOOOSSOOOOSSOOO
Switch: 0000000110001000000000000000000000000000000100000000000
```

```
[100]
1st      : VTIRENTKIGNNVKIGTILSDIQHHVYIGNYVNIHSNVFVGEKSI IKDFVW
2nd(d)   : ?EE?SS?EE?SS?EE?TT?EE?SS?EE?SS?EE?TT?EE?TT?EE?TT?E
2nd(s)   : OSSOOOOSOSSOOOOSOSSOOOOSOSSOOOOSOSSOOOOSOSSOOOOSOSSOOOS
Switch: 0000000000000000000000000000000000000000000000000000000
```

```
[150]
1st      : LFPHVVLTDPTPPSNELLGVTIELFAVIAARSVVLPGIHINEDALVGAG
2nd(d)   : E?TT?EE???SSSS?S????EE?TT?EE?TT?EE?TT?EE?TT?EE?TT
2nd(s)   : S000OSS0000000000000000SS000OSS000OSS000OSS000OSS000
Switch: 000000000000000000000000000000000000000000000000000
```

```
[200]
1st      : AVVTKDVPKETVVVGNPAREICSIKIKNKITGEQVYPWRYTFKRGMPEWE
2nd(d)   : ?EE?S?B?TTEEEEEETTEEEEEEGGG?B?TTT??B??SGGGT??TTSTTT
2nd(s)   : OSSOOOSOOOSSSSSOOSSSSSSHHSOOSOOSOOSHHSOOSOOSOOSO
Switch: 000000000000000100100000000000000000000000000000
```

```
[250]
1st      : ETDYDTWIKNISIE
2nd(d)   : TS?HHHHHHH?***
2nd(s)   : OOOHHHHHHHODDD
Switch:   00000000111?00
```

$$> 3MQH : A$$

```
[ -2 ]  
1st      : HMATIHPTAIVDEGARIGAHSRIWHVHICGGAEIGEGCSLGQNVFVGNR  
2nd(d)   : ???EE?TT?EE?TT?EE?TT?EE?TT?EE?TT?EE?TT?EE?SS  
2nd(s)   : OOOSSOOOOSOOOOSSOOOOSSOOOOSSOOOOSSOOOOSSOOOOSSOOO  
Switch: 0?010000000000000000000000000000000000000000000000
```

```
[48]
1st      : VRIGNRVKIQNNVSVYDENVFLEDDVFCGPSMVFTNVYNPRAAIERKSEYR
2nd(d)   : ?EE?TT?EE?TT?EE?TTEEE?TT?EE?TT?B??S?SS?BTTB??GGG??
2nd(s)   : OSSOOOSSOOOSSOOOSSSOOOOSSOOOSSOOOOOOSOOSOHHHOO
Switch: 00000000000000000000000000000000010000000000000000
```

```
[98]
1st      : DTIVRQGATLGANCTVVCGATIGRYAFVGAGAVVNKDVPDFALVVGVPAR
2nd(d)   : ?EEE?TT?EE?TT?EE?TT?EE?TT?EE?TT?EE?S?B?TTEEEETTTTE
2nd(s)   : OSSSOOOOSSOOOSSSOOOOSSSOOOOSSSOOOOSSSOOOSOOOSSSSOOOOS
Switch: 00000000000000000000000000000000000000000000000000
```

```
[148]
1st      : QIGWMSRHGEQLDLPLRGNAEATCPHTGERYILTDGVCRL
2nd(d)   : EEEEB?TTS?B?S??SSSEEEE?TTT??EEEEETTEEEE
2nd(s)   : SSSSSOOOOOSOOOOOOOOSSSSOOOOOOSSSSSOOSSSS
Switch: 00000000000000000000000000000000000000000000000000
```

>3N37:A

```
[0]
1st      : MKLSRISAINWNKISDDKDLEVWNRLTSNFWLPEKVPLSNDIPAWQTLTV
2nd(d)   : ****??????TTS?S?HHHHHHHHHHHHT??GGS?GGGGHHHHTTS?H
2nd(s)   : DDDDOOOOOOOOOOOHHHHHHHHHHHHOOOOHHHOOHHHHHHHHHHOOOOH
Switch: 0?????000000000000000000000110000111001111000011000
```

```
[50]
1st      : VEQQLTMRVFTGLTLLDTLQNVIGAPSLMPDALTPHEEAVLSNISFMEAV
2nd(d)   : HHHHHHHHHHHHHHHHHHHHHHIIIIHHHHGGG?SSHHHHHHHHHHHHHHHHH
2nd(s)   : HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHOOHHHHHHHHHHHHHHHHH
Switch: 000000000000000000000000010011110000000000000000000
```

```
[100]
1st      : HARSYSSIFSTLCQTKDVDAAYAWSEENAPLQKAQIIQQHYRGDDPLKK
2nd(d)   : HHHHHHHHHHHHS?HHHHHHHHHHHHH?HHHHHHHHHHHHHHHSS?HHHH
2nd(s)   : HHHHHHHHHHHHHOOHHHHHHHHHHHHHHOHHHHHHHHHHHHHHHHOOHHHHH
Switch: 0000000011110000000000001110000000000001110000000
```


[94]
1st : KKIGKTLLLEAEKIMKKKGILECRLYVHRQNSVGFSFYKNGFKVEDTDG
2nd(d): SSHHHHHHHHHHHHHHHHTT??EEEEEEETT?HHHHHHHHHTT?EEEEET
2nd(s): OOHHHHHHHHHHHHHHHHOOOOSSSSSSSOOHHHHHHHHHOOOOSSSSSSO
Switch: 11000000000000000100000000000000000000010000000110

[144]
1st : SDFIMEKKYE
2nd(d): TEEEEEEE?*
2nd(s): OSSSSSSSOD
Switch: 0000000000

>3QB8:A

[0]
1st : MYTLIKLTSEYTSRAISFTSRNFVASEPTSIALKLTTCDFTTSFQONIMKQ
2nd(d): ??EEEE??GGGHHHHHHHHHHHHHHH?HHHHHTT??HHHHHHHHHHHHHH
2nd(s): OOSSSSOHHHHHHHHHHHHHHHHHHHOHHHHHOOOHHHHHHHHHHHHHH
Switch: 000

[50]
1st : CVDYGHSAFAFVDADDNIKAQILNIPYDAYENMHYGNIRETDPMFDLFGNL
2nd(d): HHHTT??EEEE?TT??EEEEEEEEHHHHHT?????GGGHHHHHHHHHG
2nd(s): HHOOOOSSSSSOOOOSSSSSSSSSHHHHHOOOOOOHHHHHHHHHHHHHH
Switch: 000

[100]
1st : DSYTPDDKCLYVFAIGSEVTGKGLATKLLKKTIEESSSHGFKYIYGDCTN
2nd(d): GGS??SS?EEEEEEEEESS?SSSHHHHHHHHHHHHHHTT??EEEEEE?S
2nd(s): HHOOOOSSSSSSSSSSOOOOOHHHHHHHHHHHHHHHOOOOSSSSSSO
Switch: 000

[150]
1st : IISQNMFEKHGFETVGSVKYKGYQYGITKPFDSINCTEYIKRMVKT
2nd(d): HHHHHHHHTT?EEEEEEESTT??BTTB?TTTT??S?SEEEEEEEE
2nd(s): HHHHHHHHOOSSSSSSSSOOOOOSOOSOOOOOOOOOOSSSSSSS
Switch: 000

>3SMA:A

[0]

1st : MIRHIDARREDLEPDRQDRELVTRDRLASDLAALGVRPGGVLLVHASLSA
2nd(d): *****??EE?HHHHHHHHHHT??TT?EEEEEE?STT
2nd(s): DDDDDDDDDDDDDDDDDDOOSSOHHHHHHHHHHHOOOOOOSSSSSSOOOO
Switch: 0000000000000????00000000000000000000000000000011

[50]

1st : LGWVCGGAQAVVLALQDAVGKEGTLVMPTFSGDLSDPSTWRRPPVPEDWW
2nd(d): S?EETTHHHHHHHHHHHHHH?TT?EEEE???TTSS?GGG??SS??G GGH
2nd(s): OOSOOHHHHHHHHHHHHHHHOOOOSSSSSOOOOOOOOHHHOOOOOOHHH
Switch: 1000

[100]

1st : PVIREQMPPFDPLTPTRGMGAVAECFRAAGAVRSGHPQNSFAAWGAHA
2nd(d): HHHHHH????TTT??SSS?HHHHHHTTSTT?EE?S?TTT?EEEEETH
2nd(s): HHHHHHOOOOOOOOOOOOOOHHHHHHHOOOOOOSOOOOOOSSSSSOOH
Switch: 000

[150]

1st : EQVVAEHGLTERLGRGSPLEQVYRLDGQVLLLGCGFESNTSFHLAEYRTA
2nd(d): HHHHTT??STTTTSTTSHHHHHHHTT?EEEEES??GGG?TTHHHHHHH??
2nd(s): HHHHOOOOOOOOOOOOHHHHHHHOOOOSSSSSOOOHHHOOHHHHHHHOO
Switch: 0001000000000

[200]

1st : YPGRRSHRRRVPVPEGDRVRWVEQEDIVYFEEDFQTMGESCLTRTPGHSR
2nd(d): ?TT?EEEEEEEEETEEEEEEEEEE??TTHHHHHHHHHHHSTT?EE
2nd(s): OOOOOSSSSSSSSSSOOSSSSSSSSSSSOOOOOHHHHHHHHHHHOOOOSS
Switch: 0001000000

[250]

1st : GTVGEAAVLYGQRAFVDLACEWMTAHRDLARAVG
2nd(d): EEETEEEEEEHHHHHHHHHHHHHHH???*
2nd(s): SSSOOSSSSSSHHHHHHHHHHHHHHHHOOOODDDD
Switch: 00000000000000000000000000000000??000

```
[0]
1st      : MKGSRIELGDVTPHNIKQLKRLNQVIFPVSYNDKFYKDVLEVGE LAKLAY
2nd(d)   : ??S?SEEEEE??TTTHHHHHHHHHHHH?SS???HHHHHHHTTSGGGEEEEEE
2nd(s)   : OOOOOSSSSSOOOOOHHHHHHHHHHHHOOOOOHHHHHHHHOOHHHSSSSS
Switch:   ???000000000000000000000000000000000000000000001111100000
```

```
[100]
1st      : EKDGTFDNIYLVHVSINESAIDFYRKFGFEIIETKKNYYKRIEPADAHVL
2nd(d)   : HHH???SEEEEEETT?HHHHHHHHHTT?EEEEESS?SSSSSS??EEEE
2nd(s)   : HHOOOOSSSSSSOOOHHHHHHHHHOOSSSSSSOOOOOOOOOOOSSSS
Switch: 111000000000000000000000000000000000000000000000000
```

```
[150]
1st      : QKNLKVPSGQNADVQKTD
2nd(d)   : EEE?*****
2nd(s)   : SSSOODDDDDDDDDDDDD
Switch:  0000????????????
```

```
[0]
1st      : MDAESIEWKLTANLRNGPTFFQPLADSIIEPLQFKLIGSDTVATAFPVFDT
2nd(d)   : ****??GGGBGGG??S??SS???SSSB???EEEE?TTS??EEEEEEESSG
2nd(s)   : DDDDOOHSHSHHHOOOOOOOOOOOOOOSOOOSSSSOOOOOOOSSSSSSSOOH
Switch:  ?????00000000000000000000001000000000000000000000000
```

```
[50]
1st      : KYIPDSLINYLFKLFNLEIESGKTYPQLHSLTKQGFLNYWFHSFAVVVLQ
2nd(d)   : GGS?HHHHHHHHHHHHHHHHHTSSSS??S???HHHHHHHHTSSEEEEEES
2nd(s)   : HHOOHHHHHHHHHHHHHHHHHHOOOOOOOOOOHHHHHHHHHOOSSSSSSO
Switch:  0000000000000000000010000000000000000000000000001
```

[100]
 1st : TDEKFIQDNQDWNSVLLGTFYIKPNYAPRCSHNCNAGFLVNGAHRGQKVG
 2nd(d): ?S?SS??SS??HHHEEEEEEEEESSSGGGTTEEEEEEEE?GGGTTS?HH
 2nd(s): OOOOOOOOOOHHHHSSSSSSSSSSOOOHHHOSSSSSSSSSOHHHHOOOOHH
 Switch: 000

[150]
 1st : YRLAQVYLNWAPLLGYKYSIFNLVFTNQASWKIWDKLNFRIGLVPHAG
 2nd(d): HHHHHHHHHTGGGT??EEEEEEEETT?THIIIIITTTT?EEEEEEEEEE
 2nd(s): HHHHHHHHHHOHHHHOOOSSSSSSSSSOOOOHHHHHHOOOOOSSSSSSSSSS
 Switch: 000000000010000000000000000000001111?111110000000000000

[200]
 1st : ILNGFSEPVDAlIYGKDLTKIEPEFLSM
 2nd(d): ??TT?SS?EEEEEEEEETTS??HHHH??
 2nd(s): OOOOOOOOSSSSSSSSSSOOOOOHHHHOO
 Switch: 00000000000000000000000000000000

>4AVA:A

[0]
 1st : MDGIAELTGARVEDLAGMDVFQGCPAEGLVSLAASVQPLRAAAGQVLLRQ
 2nd(d): *???????S??HHHHTTSGGGTT??HHHHHHHHHH?EEEE?TT?EEE?T
 2nd(s): DOOOOOOOOOHHHHOOOHHHOOOOHHHHHHHHHHHOSSSSSSOOOOSSSOO
 Switch: 0???????000000010011100000001000000000000000000000000

[50]
 1st : GEPAVSFLLISSGSAEVSHVGDDGVAIIARALPGMIVGEIALLRDSPRSA
 2nd(d): TSB???EEEEEE??EEEEEE?TT??EEEEEE?TT?EESHHHHHHT?B?SS
 2nd(s): OOSOOOSSSSSSSOOSSSSSSSOOOOOSSSSSSOOOOSSOHHHHHHHOOSOOO
 Switch: 0010010000001100000000000000000000000000000100000001000

[100]
 1st : TVTTIEPLTGWTGGRGAFATMVHIPGVGERLLRTARQRLAAFVSPVIPVRL
 2nd(d): EEEESS?EEEEEE?HHHHHHHHHSTTHHHHHHHHHHHHHHHHH??EEEE?
 2nd(s): SSSSOOSSSSSSSOHHHHHHHHHHOOOHHHHHHHHHHHHHHHHHHOOOSSSSO
 Switch: 00001110000001000000001000000000000000000001100000000

[150]
 1st : ADGTQLMLRPVLPGDRETRVHGHIQFSGETLYRRFMSARVPSPALMHYLS
 2nd(d): TT??EEEEEE??TT?GGGT???SS???HHHHHGGG?*****?HHHHHHHH
 2nd(s): OOOOSSSSSSOOOOHHHOOOOOOOHHHHHHHHHODDDDOHHHHHHHH
 Switch: 000000000000001000111000000100000000?000000000

[200]
 1st : EVDYVDHFVWVVTGSDPVADARFVRDETDPTVAEIAFTVADAYQGRGIG
 2nd(d): HH??SSEEEEEEEETEEEEEEEEEE?SS?TEEEEEEEEE?GGGTTSSH
 2nd(s): HHOOOSSSSSSSSSOOSSSSSSSSSSSOOOOOOSSSSSSSSSOHHHOOOOHH
 Switch: 11000000000001001000000000000000000000000000000000

[250]
 1st : SFLIGALSVAARVDGVERFAARMLSDNVPMRTIMDRYGAVWQREDVGVIT
 2nd(d): HHHHHHHHHHHHTT??EEEEEEET?HHHHHHHHHTT??EE?STTEEE
 2nd(s): HHHHHHHHHHHHHOOOSSSSSSSOOHHHHHHHHHOOOOOSSSOOOOSS
 Switch: 00000000000001000000000000000000000010000001100000

[300]
 1st : TMIDVPGPGELSLGREMVDQINRVARQVIEAV
 2nd(d): EEEE??TTTSSS?HHHHHHHHHHHHHIIIII?
 2nd(s): SSSSOOOOOOOOOHHHHHHHHHHHHHHHHHHHO
 Switch: 00000001110000100000000000000110

>4EAB:A

[0]
 1st : MSASLAIGGVVIIGGGGHAKVVIESLRACGETVAAIVDADPTRRAVLGVP
 2nd(d): **????TT?EEEE??SHHHHHHHHHHHHTT??EEEE?SS??*?EETTEE
 2nd(s): DDOOOOOOSSSSOOHHHHHHHHHHHHHOOOSSSSSSOOOOODOSSSOSS
 Switch: 00?0?100001

[50]
 1st : VVGDDLALPMLREQGLSRLFVAIGDNRLRQKLGRKARDHGFSLVNAIHPS
 2nd(d): EEESGGGHHHHHHHTT??EEEE??HHHHHHHHHHHHHTT?EE??EE?TT
 2nd(s): SSSOHHHHHHHHHOOOSSSSSOOOHHHHHHHHHHHHHOOOSSSOSSOOO
 Switch: 000

[100]
1st : AVVSPSVRLGEGVAVMAGVAINADSWIGDLAIINTGAVVDHDCRLGAACH
2nd(d): ?EE?TT?EE?SS?EE?TT?EE?TT?EE?TT?EE?TT?EE?TT?EE?TT?E
2nd(s): OSSOOOSSOOOSSOOOSSOOOSSOOOSSOOOSSOOOSSOOOSS
Switch: 000

[150]
1st : LGPASALAGGVSVGERAFLGVGARVIPGVTIGADTIVGAGGVVVRDLPDS
2nd(d): E?TT?EE?TT?EE?TT?EE?TT?EE?TT?EE?TT?EE?TT?EE?S?B?TT
2nd(s): SOOOOSSOOOSSOOOSSOOOSSOOOSSOOOSSOOOSSOOOSSOOOSSOO
Switch: 000

[200]
1st : VLAIGVPAKIKGDR
2nd(d): ?EEETTTTEE???*
2nd(s): OSSSOOOOSSOOD
Switch: 00000000000?0

>4FD4:A

[0]
1st : MVAPESIVLRVARLDELEQVREILHRIYYPEEGITISYVHGKSHTLDDER
2nd(d): *****?EEEE?GGGHHHHHHHIIIIITTT?HHHHHBTT?SS??HHHHH
2nd(s): DDDDDOSSSSSOHHHHHHHHHHHHHHHOOOHHHHHSOOOOOOHHHHH
Switch: ?????1000000000000000000000000000000010000000000000

[50]
1st : FLSFVEQGTVVVAEDSAAKKFIGVSIAGPIQPGDPDAMVEEAATTETKK
2nd(d): HHHTTTTTT?EEEEETTTTEEEEEEEEEEE?TTHHHHHHHHHHHSS?HH
2nd(s): HHOOOOOOOSSSSSSSOOOOSSSSSSSSSSSOOHHHHHHHHHHHOOOHH
Switch: 000001110000000000000000000000000000000000001100000

[100]
1st : WGDILKLLALLERTADVCGRYGLEKAYHVHILAVDPTYRGHSLGQRLLOF
2nd(d): HHHHHHHHHHHHHHH?HHHHHT?S?EEEEEEEEEE?TTS?SS?HHHHHHHH
2nd(s): HHHHHHHHHHHHHHHOHHHHHOOOOSSSSSSSSSSOOOOOOOHHHHHHHH
Switch: 000000000000000000000000000000000000111000000000000

```
[150]
1st      : QMDLSKKLGFKAISGDFTSVFSVKLAEKLGMECISQLALGDYRDEKGEKL
2nd(d)   : HHHHHHHT?SEEEEE?SHHHHHHHHTT?EEEEEEGGG???TT????
2nd(s)   : HHHHHHHHOOSSSSSSSOHHHHHHHHHOOSSSSSSSSHOOOooooooooo
Switch:  0000000100000000000000000000000000000000000111010000010
```

```
[200]
1st      : FEPLDVHQVIKTCVKL
2nd(d)   : B??SSSS?EEEEEEEE
2nd(s)   : SOOOOOOOSSSSSSSS
Switch:  0000111000000000
```

>4FD5:A

```
[0]
1st      : MLDSKLNIRFETISSKYYDDVIEHLRQTFFADEPLNKAVNLTRPGQGHF
2nd(d)   : *****??EEEE??GGGHHHHHHHHHHHTTTTSHHHHHTT??T??H
2nd(s)   : DDDDDDOOSSSSOOHHHHHHHHHHHHHHHOOOOOHHHHHHOOOOOOOOOOH
Switch:  0000??000000000000000000000000010000000000000000000
```

```
[50]
1st      : LLEQHSLSTLKD NVSIMAISNDGDIAGVALNGILYGN TDieKSREKLNEI
2nd(d)   : HHHHHHHHHTTS??EEEE?TTS?EEEEEEEEEEETTS?TTHHHHHHHH?
2nd(s)   : HHHHHHHHHHO OOOOOSSSSOOOOOSSSSSSSSSSSSOOOOOOH HHHHHHHHO
Switch:  00000000000000000000000000000000000000000001000000110
```

```
[100]
1st      : QDESFKKIFKLLYEQNLKINLFKQFDVDKIFEIRILSVDSRFRGKGLAKK
2nd(d)   : ??HHHHHHHHHHHHHHTT??HHHHHT?SEEEEEEEEEEE?GGGTSSSHHHH
2nd(s)   : OOHHHHHHHHHHHHHHHOOOHHHHHOOOSSSSSSSSSSSOHHHOOOOHHHH
Switch: 000000000000000000000000000000000000000000000000000
```

```
[150]
1st      : LIEKSEELALDRGFQVMKTDATGAFSQRVVSSLGFITKCEINYTDYLDEN
2nd(d)   : HHHHHHHHHHHTT?EETEEE?SHHHHHHHHHTT?EEEEEEEGGG?B?TT
2nd(s)   : HHHHHHHHHHHO OOOSSSSSSSOOH HHHHHHHHHHO OOS SSSSSSSH H OSOO
Switch: 000000000000000000000000000000000000000000000000000
```

[200]
1st : GEQIFVVDPPHEKLIKIMCKVI
2nd(d): SSBS????TT??EEEEEEEE?
2nd(s): OOSOOOOOOOOOSSSSSSSSO
Switch: 00000000000000000000

>4FD7:A

[0]
1st : MKWTRSVKVPFPSVWHRFQAKDLTSQQLVWYRVQDLPEDRFEDAIRHMCD
2nd(d): *?????TTS?SS?EEEEEEEE?SSSS?EEEEEEEE??GGGHHHHHHHHIIII
2nd(s): DOOOOOOOOOOOOSSSSSSSSSOOOOOOSSSSSSSSSOHHHHHHHHHHHHHHH
Switch: 000

[50]
1st : YFARDELNMQAKGLAKDLVAMGDVVALWKAMLPDRMSLVCFREGSDEIVG
2nd(d): IHHHSHHHHHHTGGG?HHHHHHHHHHHHHHHGGGS??EEEEETT??SEEE
2nd(s): HHHHHOHHHHHHOHHHHOHHHHHHHHHHHHHHHHHHOOOSSSSSOOOOSSS
Switch: 00000000000001110000000000000000000000000000000000

[100]
1st : VNILDVASRSDKDNAQFNSAIFQAIYDTIEYVSHQANIFDRYNVDHYLNA
2nd(d): EEEEEEEETT??????S?HHHHHHHHHHHHHHHHHHHTHHHHHT?SEEEEE
2nd(s): SSSSSSSSOOOOOOOOOOHHHHHHHHHHHHHHHHHHHOHHHHHOOSSSSS
Switch: 000100000100000000

[150]
1st : MGLSVDPKYRGRGIATEILRARIPLCRAVGLKLSATCFTGPNSQTAATRV
2nd(d): EEEEE?GGGTTSSHHHHHHHTHHHHHHHT??EEEEEE?SHHHHHHHHHH
2nd(s): SSSSOHHHOOOHHHHHHHOHHHHHHHHHOOSSSSSSSOHHHHHHHHHHH
Switch: 000

[200]
1st : GFQEDFTITYGELARVDQRFNYPGIEENFCKYMSLRV
2nd(d): T?EEEEEEHHHHHHH?TT??TT?TTSEEEEEEEE?
2nd(s): OOSSSSSSHHHHHHHOOOOOOOOOOOSSSSSSSSO
Switch: 000000000000000001110000000000000000

[illegible]

```
[100]
1st      : MRNTKEYKESIKWFTAALNNGSTNKQIYRDLATLQSQIGDFKNALVSRKK
2nd(d)   : HHHHTT?HHHHHHHHHHHHHHHTT??TTHHHHHHHHHHHHHHTT?HHHHHHHHHH
2nd(s)   : HHHOOOHHHHHHHHHHHHHHHOOOOOHHHHHHHHHHHHHHOOOHHHHHHHHHH
Switch: 011000100000000000010000001000011111111001000000001
```

```
[200]
1st      : HSECLMYKNDIMYKAASDNQDKLQNVLKHLNDIEPCVFDKFGLLERKATI
2nd(d)   : HHHHHHHHHHHHHHHHHHTT?HHHHHHHHHHHHHHHGGG??HHHHHHHHHHHH
2nd(s)   : HHHHHHHHHHHHHHHHHHOOOHHHHHHHHHHHHHHHHHHHOOOHHHHHHHHHHHH
Switch: 11111100000111110001111000000001111100011111111111
```

80

[650]
1st : AKKEAAAMNKRKETEAKSVAAYPSDQDNDVFGEKLIETSTPMEDFATEFY
2nd(d): *****?TT?HHHHT?S?HHHHHIIIII
2nd(s): DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDOOOOHHHHOOOOHHHHHHHHHH
Switch: ?????????????????????????0000111110001111000010

[700]
1st : NNYSMQVREDERDYILDFEFNYRIGKLALCFASLNKFAKRFGTTSGLFGS
2nd(d): HHHHHH??TTT??HHHHHHHHHHTT?HHHHHHHHHHHHHHH?SSSHHHHH
2nd(s): HHHHHHOOOOOOHHHHHHHHHHOOOHHHHHHHHHHHHHHHHOOOOHHHHH
Switch: 001111001110011111111111100100000001110001000010000

[750]
1st : MAIVLLHATRNDTPFDPILKKVVTKSLEKEYSENFPLNEISNNSFDWLNF
2nd(d): HHHHHHHHT?***??HHHHHHHHHHHGGGS??*****??HHHH
2nd(s): HHHHHHHHOODDDOOOHHHHHHHHHHHHHHHHOOODDDDDDDDDDDOOHHHH
Switch: 000111111?????111???111???111111????????????????1111

[800]
1st : YQEKFGKNDINGLLFLYRYRDDVPIGSSNLKEMIISLSPLEPHSQNEIL
2nd(d): HHHHT?S??HHHHHHHHHTTTTTT?HHHHHHHHHHHTTTS?HHHHHHHH
2nd(s): HHHHOOOOOHHHHHHHHHHOOOOOOOHHHHHHHHHHHHOOOOOHHHHHHHH
Switch: 11111??00000000011000000?0100010001110000000001111

[850]
1st : QYYLYPYDVDP
2nd(d): HHHTGGGTS??
2nd(s): HHHOHHHOOOO
Switch: 111?1100000

>4J3G:A

[0]
1st : MTLIRHATEADLPALLAIYNDAVENTLAIWNETLVDLENRHQWLENRNR
2nd(d): ???EEEE??GGGHHHHHHHHHHHHHHH?SSS??????HHHHHHHHHHHHHH
2nd(s): OOOSSSSOHHHHHHHHHHHHHHHHHHHHOOOOOOOOOHHHHHHHHHHHHHHHH
Switch: ??00

```
[50]
1st      : DGFPVLVAEREGQVVGYSYGPFRPFEGFRHSSELSVYVASNARGGGIGR
2nd(d)   : HT??EEEEETTEEEEEEEEEESSSSGGGTTEEEEEEEE?GGGTTSSH HH
2nd(s)   : HOOSSSSSSSOOSSSSSSSSSSO OOH HHOOSSSSSSSSSOHHHOOOOHHH
Switch:  100000000000000000000000000000000000000000000111000000
```

```
[100]
1st      : TLLAELIEEARERKVHVLIAGIEAGNAASIALHRSQGFEECGTLKQVGQK
2nd(d)   : HHHHHHHHHHHHTT??EEEEEEETT?HHHHHHHHHTT?EEEEEEEEEEEEEE
2nd(s)   : HHHHHHHHHHHHOOOOSSSSSSSOOHHHHHHHHHOOOOSSSSSSSSSSSSS
Switch:  00000000000000000000000000000000000000000000000000000
```

```
[150]
1st      : FGRWLDLLFMQKI
2nd(d)   : TEEEEEEEEEEEEEE
2nd(s)   : OOSSSSSSSSSSSS
Switch:  00000000000000
```

>4JJX:A

```
[-2]
1st      : AMNSQLTLRALERGDLRFIHNLNNNRNIMSYWFEEPPYESFDELEELYLNKH
2nd(d)   : ?GGGG?EEEE??GGGHHHHHTTTTTT?S??SSS????SHHHHHHHHHTT
2nd(s)   : OHHHHOSSSSOOHHHHHHHHHOOOOOOOOOOOOOOOOOOHHHHHHHHHHOO
Switch: 11111100000011100000111111111111100111000000000111
```

```
[48]
1st      : IHDNAERRFVVEDAQKNLIGLVELIEINYIHRSAEFQIIIIAPEHQKGKFA
2nd(d)   : TT?SSEEEEEEEE?TT??EEEEEEEEEEEEETTTTEEEEEEEEE?GGGGGG??H
2nd(s)   : OOOOOSSSSSSSSOOOOOSSSSSSSSSSSSOOOOOSSSSSSSSOHHHHHHHOOH
Switch:  11000111000010000110000001110000110000000111111011
```

```
[98]
1st      : RTLINRALDYSFTILNLHKIYLHVAVENPKAVHLYEECGFVEEGHLVEEF
2nd(d)   : HHHHHHHHHHHHHTS??SEEEEEEEETT?HHHHHHHHHTT?EEEEEEEEEEE
2nd(s)   : HHHHHHHHHHHHHHOOOOOSSSSSSSOOOHHHHHHHHHHOOOSSSSSSSSSS
Switch: 10000000000011100000000000001110000011000000000000
```

```

[148]
1st      : FINGRYQDVKRMYYILQSKYLNRS
2nd(d)   : EETEEEEEEEEEEEEHHHHH??*
2nd(s)   : SSOOSSSSSSSSSSSHHHHHOOD
Switch: 010010000000001?11111??

>4KVO:A

[0]
1st      : MAKVQLSPKEITLFR TALKCYETKQYKKGLKAIEPLLERHPEHGESLAIK
2nd(d)   : ****???HHHHHHHHHHHHHHHTS?THHHHHHHHHHHSS?TT?HHHHHHH
2nd(s)   : DDDDOOOHHHHHHHHHHHHHHHOOOOHHHHHHHHHHHHHOOOOOHHHHHHH
Switch: ???0001110111000000000001000001100011000000000000

[50]
1st      : GILLHSLGNTKEGYDNVRLGLRNDVGSVCWHIFGLISRADKDYVQAAKC
2nd(d)   : HHHHHHTS?SHHHHHHHHHHHHHHTT?HHHHHHHHHHHHHTT?HHHHHHH
2nd(s)   : HHHHHHOOOOHHHHHHHHHHHHHHHOOOOHHHHHHHHHHHHHOOOHHHHHHH
Switch: 0000001000100000000011100001000000000000000000000000

[100]
1st      : YINAHKLEKNSSLLRDLALLQSQLRQYKALADTRNALLQDNPGVRANWS
2nd(d)   : HHHHHHH?TT?HHHHHHHHHHHHHTT?HHHHHHHHHHHHHHH?TTSHHHHH
2nd(s)   : HHHHHHHHOOOOHHHHHHHHHHHHHHHOOOOHHHHHHHHHHHHHHHOOOHHHHH
Switch: 0000001000000000000000000100000000000000000000000000

[150]
1st      : ALAVAQFLRGEYASAYKIVDAFESTINQGV PVD TQEESEAMLFMNLVILK
2nd(d)   : HHHHHHHHTT?HHHHHHHHHHHHHT?SS?S?S?SHHHHHHHHHHHHHHHHH
2nd(s)   : HHHHHHHHOOOOHHHHHHHHHHHHHHHOOOOOOOOOHHHHHHHHHHHHHHHH
Switch: 00000000100000000000000000000000000000000000000000000

[200]
1st      : KDGVEDAYKHLLSIEKKVLDRVAFLETRA EYELYLSKMEEAKSTIYLLLD
2nd(d)   : HTHHHHHHHHHHSGGG???HHHHHHHHHHHHHTT?HHHHHHHHHHHHHHH
2nd(s)   : HOHHHHHHHHHHHOHHHOOOHHHHHHHHHHHHHHHOOOHHHHHHHHHHHHHHH
Switch: 01000000000001111000000000000000000000000000000000000

```

```
[250]
1st      : RNPDNHQYYYNLQRAYGYEDASGKVLDSAEWLNLYSQLAKRYPKSECPTR
2nd(d)   : H?TT?HHHHHHHHHHTTSS?SSS??S?HHHHHHHHHHTTTS?TT?SHHHH
2nd(s)   : HOOOOHHHHHHHHHHHOOOOOOOOOOOOHHHHHHHHHHHHOOOOOOOOOHHHH
Switch: 000001000000000000000000000000000000000000000000000000
```

```
[300]
1st      : LPLEKLEGDEFLTHVDLYLRKKLKRGI PSVFVDVKS LYKDTKKCKVVEDL
2nd(d)   : THHHH??THHHHHHHHHHHHHHHHHHT?TTHHHHHGGGGGSHHHHHHHHH
2nd(s)   : OHHHHOOOHHHHHHHHHHHHHHHHHHHOOOOHHHHHHHHHHHHHOHHHHHHHH
Switch: 100000011100000000000000010000000000000010100000000
```

```
[350]
1st      : VSKYASSLSTTNKFSEDDDNSQIEIPTTLLWTTYFLAQHFDHVGELEKAE
2nd(d)   : HHHHHHTTTTTSSSSTTGGSSTTTSSHHHHHHHHHHHHHHHHHT?SHHHH
2nd(s)   : HHHHHHOOOOOOOOOOHHHOOOOOOOHHHHHHHHHHHHHHHHHHOOOHHHH
Switch: 000000000000000000001000000000000000000000000000
```

```
[400]
1st      : KYVDLAIDHTPTLVELFMTKARISKHKGELQTAMEIMDHARKLDLQDRFI
2nd(d)   : HHHHHHHHHH?TT?HHHHHHHHHHHHHHTT?HHHHHHHHHHHHHHHH?SS?HHH
2nd(s)   : HHHHHHHHHHHOOOOHHHHHHHHHHHHHHHHOOOOHHHHHHHHHHHHHHHHOOOOHHH
Switch:   000000000000000000000000000100000000000000010000000
```

```
[450]
1st      : NGKCAKYMLRNDENELAAKTVSLFTRNEAVGGAVGDLADMQCLWYMLEDG
2nd(d)   : HHHHHHHHHHTT?HHHHHHHHTTTS??SSSSHHHHHHHHTT?HHHHHHHH
2nd(s)   : HHHHHHHHHHOOOHHHHHHHHOoooooooooHHHHHHHHHOOOHHHHHHHH
Switch: 000000000000000000000000000000000000000000000000000
```

```
[500]
1st      : KSFARQKKFALALKRFSTVFKIFDTWADDQFDFHFFAFRKGSLRTYLDLM
2nd(d)   : HHHHHHTT?HHHHHHHHHHHHHHHHHHHHHTTTTHHHHHHHHHT?HHHHHHHH
2nd(s)   : HHHHHHOOOHHHHHHHHHHHHHHHHHHHHHHOOOOHHHHHHHHHOOHHHHHHHH
Switch: 00000000000000000000000000000000100000000000000000000
```

```
[550]
1st      : SWEDSVYDDPSFREAAGSGSIEIYFALFDLPFAKYSPKLPDFEKLSSGEIN
2nd(d)   : HHHHTTT?SHHHHHHHHHHHHHHHHHHHH?TTGGG???HHHHHHHHHT?S?
2nd(s)   : HHHHOOOOHHHHHHHHHHHHHHHHHHHHHOOOHOOHHHHHHHHHHOOOO
Switch: 000100000000000000000000000000000000000000000000
```

```
[600]
1st      : EEEEEKKIYKKLKKDLSKRRLERA EK LKEADKSRAK SEDGMPVKYDEDPLGE
2nd(d)   : HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH??*****????S?SSS
2nd(s)   : HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHOODDDDDDDDDDOOOOOOOO
Switch: 0000000000000000000000000000000000000000000000000000
```

```
[650]
1st      : NLVATSEPLKEAQKCLEKLLPYGDKNPSAYILAAQLYTRLKNFDTASKYL
2nd(d)   : TTTT?S?HHHHHHHHHHTTTGGGTTT?THHHHHHHHHHHHTT?HHHHHHHH
2nd(s)   : OOOOOOHHHHHHHHHHOOOHHHOOOOHHHHHHHHHHHHHHOOOHHHHHHHH
Switch:  111000000000000010000000000000000000000000000000
```

```
[700]
1st      : EQAKVILGQNDPTVISTEKFYNSIKTQ$NA
2nd(d)   : HHHHHSS?SS?HHHHHHHHHHHHHHHTTSS?
2nd(s)   : HHHHHOooooooooHHHHHHHHHHHHHHHoooooooo
Switch:  000000000000000000000000000000000000
```

>4KVX:A

```
[0]
1st      : MDIRPARISDLTGMQNCNLHNLHPENYQLKYLYHAISWPMLSYVATDPKG
2nd(d)   : *?EEE??GGGHHHHHTTTT?GGGGTTHHHHHHHHTSTTS?EEEE?TTS
2nd(s)   : DOSSSOOHHHHHHHHHOOOOOHHHHOOHHHHHHHHHOOOOOOSSSSOOOO
Switch:  000000000000000000000000000000000000000000000000
```

```
[50]
1st      : RVVGYVLAKMEEEPKDGIPHGHITSVSVMRSYRHLGLAKRLMVQSQRAMV
2nd(d)   : ?EEEEEEEEEE?SS?GGG???EEEEEEEE?GGGTTSSHHHHHHHHHHHHHHH
2nd(s)   : OSSSSSSSSSOOOOHHHHOOOSSSSSSSSSOHHHHOOOOHHHHHHHHHHHHHHH
Switch:  0000000000000000000100000000000000000000000000000000
```

```
[100]
1st      : EVYGAKYMSLHVRKSNRAAIHL YRDTLQFDVQGIESKY YADGEDAYAMHK
2nd(d)   : HHH??SEEEEEEEETT?HHHHHHIIIIIS??EEEEEEETT?STTS??EEEEEE
2nd(s)   : HHHOOOSSSSSSSOOHHHHHHHHHHHOOSSSSSSSOOOOOOOOOOSSSSSS
Switch:  00000000000000000000000000000000000000000000000000000
```

[150]
1st : DFSTLKFDTPETNDELAKTVQSLALN
2nd(d): E??*****
2nd(s): SOODDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 0000000000000000000000000000

>4PV6:A

[0]
1st : MVYMAINAVAGTIREFSPKDIESVYRIAQTSLTEYYTQALILDLHREWPE
2nd(d): *****??SS?EEEE??GGGHHHHHHHHHHH?SS??HHHHHHHHHHHSGG
2nd(s): DDDDDO O O O O O S S S S O O H H H H H H H H H H H H O O O O O O H H H H H H H H H H O H H
Switch: 00000??001100000000000000000000110000001100000111011

[50]
1st : SFMVYTVAGSVVGFIVGSKYSRTEARILLFAVDERFRRMGVGSALMDAFL
2nd(d): G?EEEEETEEEEEEEEESSSSEEEEEEEEE?GGG?SSSHHHHHHHHHH
2nd(s): HOSSSSSOSSSSSSSSSSSOOOOSSSSSSSSSOHHHO O O O H H H H H H H H H H
Switch: 10000100100000000011000000000001111110100000000

[100]
1st : SLCREQNMLSVRLEVRTDNDEAIRFYKKYGFVITAMLPNYYSDSSNAYTM
2nd(d): HHHHHTT??EEEEEEETT?HHHHHHHHHTT?EEEEETT?STT??EEEE
2nd(s): HHHHHO O O O S S S S S S S O O H H H H H H H H H H O O O S S S S S S O O O O O O O O S S S S
Switch: 000001000000000000000000000000000000000000010000000000000

[150]
1st : WRIVLEHH
2nd(d): EEES?B??
2nd(s): SSSOOSOO
Switch: 00000100

>4R3K:A

[0]
1st : MELAEKDKGRDFTLRNARMDDIDQIIKINRLTLPENYPYFFVEHLKEYG
2nd(d): *****??EEEE??GGGHHHHHHHHHHH?S???HHHHHHHHHHHGG
2nd(s): DDDDDDDDDDOOSSSSSOHHHHHHHHHHHHHHH O O O O O O H H H H H H H H H H
Switch: 00000000000000000011100000000000000000000000000010

[150]
1st : FINGQYRNAIRMCIFQHQYLAEHKTPGQTLLKPTA
2nd(d): E?SSSEEEEEEEEEHHHHHH?*****
2nd(s): SOOOSSSSSSSSSSHHHHHHODDDDDDDDDDDDD
Switch: 0100100000000000000011?0000000000

>4RI1:A

[0]
1st : MKKNYSYKNIQAIDFTNLNDGEKLLVLEFRNHPNTALWMYSTFISLKTHL
2nd(d): *???EEETEEEEEGG??HHHHHHHHHHHHSHHHHTTSSSS???HHHHH
2nd(s): DOOOSSSOSSSSSSHHHOHHHHHHHHHHHOHHHHOOOOOOOOHHHHH
Switch: 000

[50]
1st : QFIEDLKNSPNHRYFLFKEEGVYLGVGSIKINFFHKHGYLGIYKNPFLK
2nd(d): HHHHHHTT?TT?EEEEEEETEEEEEEEEEEEEETTTTEEEEEEEE?TTSS
2nd(s): HHHHHHOOOOOOSSSSSSSOSSSSSSSSSSSSSOOOOSSSSSSSSOOOOO
Switch: 00000011000

[100]
1st : NGGETILKALEFIAFEFQLHSLHLEV MENNFKAIAFYEKNHYELEGRK
2nd(d): ??HHHHHHHHHHIIIIIS??SEEEEEETT?HHHHHHHHHTT?EEEEEE
2nd(s): OHHHHHHHHHHHHHHHOOOOSSSSSSSOOHHHHHHHHHHHOOSSSSSSS
Switch: 000

[150]
1st : GFISKDKEFIDVLLYYKDKKGYNDQSLK
2nd(d): EEEEEETEEEEEEEEEEEEHHHHHHHHH??
2nd(s): SSSSSOSSSSSSSSSSSSHHHHHHHHHO
Switch: 000000000000000000000000111?1?

>4U9V:B

[0]
1st : MGRKSSKAKEKKQKRLEERAAMDAVCAKVDAANRLGDPLEAFPVFKKYDR
2nd(d): *****?HHHHHHHHHT?S?GGGG?GGGGEE??
2nd(s): DDDDDDDDDDDDDDDDDDDDDDOHHHHHHHHHOOOHHHHHOHHHSSOO
Switch: 000000000000????????10000000010001111000000011

[50]
 1st : NGLNVSIECKRVSGLEPATVDWAFDLTKTNMQTMYEQSEWGWKDREKREE
 2nd(d): TT??EEEEEEEGGGS?HHHHHHHHHHHHHHHHHHHTSSS??HHHHHHH
 2nd(s): OOOOSSSSSSSHHHOOHHHHHHHHHHHHHHHHHHHHOOOOOOHHHHHHH
 Switch: 001100

[100]
 1st : MTDDRAWYLIAWENSSVPVAFSHFRFDVECGDEVLYCYEVQLESKVRKRG
 2nd(d): HH?TT?EEEEEEETTTEEEEEEEEEEEEEETTEEEEEEEEEEE?GGG?SSS
 2nd(s): HHOOOSSSSSSSSOOOSSSSSSSSSSSSSSSOOSSSSSSSSSSSOHHHHOOOO
 Switch: 01000000000011001000000000000000000000000000000000

[150]
 1st : LGKFLIQILQLMANSTQMKKVMLTVFKHNHGAYQFFREALQFEIDDSSPS
 2nd(d): HHHHHHHHHHHHHHTT?SEEEEEETT?HHHHHIIIIIS??EE?TTS??
 2nd(s): HHHHHHHHHHHHHHHOOOSSSSSSSSOOOHHHHHHHHHHHHOOOSSOOOOOO
 Switch: 00000000000000010000000000000000000000000000000000

[200]
 1st : MSGCCGEDCSYEILSRRTKFGDSHHSHAGGHCGGCC
 2nd(d): TTTBTTB??SEEEEEEE??*****
 2nd(s): OOSOOSSOOSSSSSSSSOOODDDDDDDDDDDDDDDDDDD
 Switch: ???????00000000000000000000000000000000

>4UA3:A

[0]
 1st : MHFLKQYLFYSPRRMKTQEIKNVTLEDVDFLKNLGVWVEIYHHLEKGLLQ
 2nd(d): *****????*****??S?HHHHHTTTEEEEEESS??HHHHH
 2nd(s): DDDDDDDDDDDDOOOODDDDDDDOOOOHHHHHHOOOSSSSSSOOOOHHHHH
 Switch: 000000000000111??????0000000000000000000000000000000

[50]
 1st : QCFNLVKKNMEALYRQSSFGWDDSEKLEMEMEKLEYICIFEKTSKKLVG
 2nd(d): HHHHHHHHHHHHHHHSTT??HHHHHHHT?TT?EEEEEEETTT?EEE
 2nd(s): HHHHHHHHHHHHHHHHHOOOOOOHHHHHHHHOOOOOSSSSSSSOOOOOSSS
 Switch: 00

```

[100]
1st      : FLSFEDTVEAGLTCLYIYEIQLDEHIRGRNVGKWLLKNASILAYRRNLKY
2nd(d)   : EEEEEEEETTEEEEEEEEEEEEE?GGGTTSSHHHHHHHHHHHHHHHTT??E
2nd(s)   : SSSSSSSSSOOSSSSSSSSSSSSOHHHOOOOHHHHHHHHHHHHHHHHOOOOS
Switch: 0000000000000000000000000000000000000000000000000000000

```

```

[150]
1st      : IFLTVEFSANLNALNFYHHFDFVPHESSPQEKKFRSGKVIHPDYIYILYTKS
2nd(d)   : EEEEEETT?HHHHHHHHHTT?EE?TTS??EE?TTS?EE??SEEEEEEE?
2nd(s)   : SSSSSSOOOHHHHHHHHHHOOOSSOOOOOOOSSOOOOOSSOOSSSSSSSSO
Switch: 0000000000000000000000000000000000000000000000000000000

```

```

[200]
1st      : RKD
2nd(d)   : ?S?
2nd(s)   : OOO
Switch: 000

```

>5GI5:A

```

[0]
1st      : MEVQKLPDQSLISSMMLDSRCGLNDLYPIARLTQKMEDALTVSGKPAACP
2nd(d)   : *****
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 0000000000000000000000000000000000000000000000000000000

```

```

[50]
1st      : VDQDCPYTIELIQPEDGEAVIAMLKTFFFKDEPLNTFLDLGECKELEKYS
2nd(d)   : ?GGG??EEEE??GGGHHHHHHHHHHHTHHHSHHHHHHT??S?HHHHHHH
2nd(s)   : OHHHOOSSSSSOOHHHHHHHHHHHHHHOHHHHOHHHHHHOOOOOHHHHHHH
Switch: ???1?0100000000000000000000000000000000000000000000000000000

```

```

[100]
1st      : LKPLPDNCSYKAVNKKGEIIGVFLNGLMRRPSPDDVPEKAADSCEHPKFK
2nd(d)   : HTTGGGS??EEEEETS?EEEEEEEEEEEE????*****?TT??HHHH
2nd(s)   : HOOHHHOOOSSSSSSOOOOSSSSSSSSSSSSSOOOOODDDDDDOOOOOOHHHH
Switch: 0000000000000100000000000000000000000000000000000000000

```

```
[150]
1st      : KILSLMDHVEEQFNIFDVYPDEELILDGKILSVDTNYRGLGIAGRLTERA
2nd(d)   : HHHHHHHHHHHH? ? HHHHSTT? SEEEEEEEEEE? GGGTTSSHHHHHHHHHH
2nd(s)   : HHHHHHHHHHHHHOHHHHHOOOOOSSSSSSSSSSOHHHHOOOOHHHHHHHHHH
Switch: 000000000000000000000000000000000000000000000000000
```

[illegible]

```
[250]
1st      : PAAPHVGIQVMAKEVGPAAQAQTK
2nd(d)   : ??TT??EEEEEEEE?*****
2nd(s)   : OOOOOOSSSSSSSSSODDDDDDDDD
Switch:  001110100000000000000000
```

>5HGZ:A

```
[0]
1st      : MTEVVPSSALSEVSLRLLCHDDIDTVKHLCGDWFPPIEYPDSWYRDITSNK
2nd(d)   : ??HHHHHHHTTSEEEEE??GGGHHHHHHHHHHH?SS??HHHHHHHHH?T
2nd(s)   : OOHHHHHHHOOSOSSSSSOHHHHHHHHHHHHHHHHOOOOOOHHHHHHHHHHOO
Switch:   ??1111110111000000000000000000000000000000000000000001100
```

```
[50]
1st      : KFFSLAATYRGAIVGMIVAEIKNRTKIHKEDGDILASNFSVDTQVAYILS
2nd(d)   : TEEEEEEEEETEEEEEEEEEEEEEGGGS?GGGTTSS?TTS?TT?EEEEEEE
2nd(s)   : OSSSSSSSSSOSSSSSSSSSSSSSHHHOOHHHOOOOOOOOOOOOSSSSSSS
Switch:  01000000000000000000000000000000001110???00000000000
```

```
[100]
1st      : LGVVKEFRKHGIGSLLLESKDHISTTAQDHCKAIYLVLTNNNTAINFY
2nd(d)   : EEE?GGGTTSSHHHHHHHHHHHHHHHHHHHTTTEEEEEEEETT?HHHHHHH
2nd(s)   : SSSOHHHHOOOOHHHHHHHHHHHHHHHHHHHHOOSSSSSSSSSSOOOHHHHHHH
Switch: 00000000000000000000000000000000111111100000000000000000000
```

```
[150]
1st      : ENRDFKQHLYLPYYYSIRGVLKDGFTYVLYINGGHPPTILDYIQHLGSA
2nd(d)   : HTTT?EEEEEEEEEEEEETEEEEEEEEEEEE?TT?????HHHHHHHHHHHH
2nd(s)   : HOOOOSSSSSSSSSSSSSSOSSSSSSSSSSSSSOOOOOOOOOHHHHHHHHHHHH
Switch:  01000000000110001010011000000000????????????????1?
```

```
[200]
1st      : LASLSPCSIPHRVYRQAHSLLCSFLPWSGISSKSGIEYSRT
2nd(d)   : HTTS?GGGS?*****
2nd(s)   : HOOOOHHHOOODDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch:  ??????????0000000000000000000000000000000000
```

>5JPH:A

```
[ -1 ]  
1st      : AMFSKVNNQKMLEDCFYIRKKVFVEEQGVPEESEIDEYESESIHLIGYDN  
2nd(d)   : ??EEE??SHHHHHHHHHHHHIIIIIIIT???HHHHS?TTTTTSEEET  
2nd(s)   : OOSSOOHHHHHHHHHHHHHHHHHHHOOOHHHHOooooooooSSSSSSSO  
Switch: 01000000000000000000000000000000000000000000000000
```

```
[49]
1st      : GQPVATARIRPINETTVKIERVAVMKSHRGQGMGRMLMQAVESLAKDEGF
2nd(d)   : TEEEEEEEEEEE?SSSEEEEEEEEEEE?GGGTTSSHHHHHHHHHHHHHHHTT?
2nd(s)   : OSSSSSSSSSSSOOOOSSSSSSSSSSOHHHOOOOHHHHHHHHHHHHHHHHOOO
Switch: 000000000000000000000000000000000000000000000000000
```

```
[99]
1st      : YVATMNAQCHAIPFYESLNFKMGRGNIFLEEGIEHIEMTKKLTSL
2nd(d)   : ?EEEEEEEGGGHHHHHHTT?EEEEEEEEETEEEEEEEEEE??*
2nd(s)   : OSSSSSSSHHHHHHHHHOOOSSSSSSSSSOOSSSSSSSSSOODD
Switch: 000000111110000000000001111110011110000000??
```

>5K04:A

```
[0]
1st      : MATERDQEIIDFIDQGNITYAQSLITKKLAKSPQKLFYHVLQNEIHLKSG
2nd(d)   : *****?HHHHHHHHH?*?HHHHHHHHHHHHHHH??*?HHHHHHHHHHHHHHHH
2nd(s)   : DDDDDDDOHHHHHHHHHODOHHHHHHHHHHHHHHHOODDOHHHHHHHHHHHHHHHH
Switch: 0000000???????????1????????????????00?10000000000001
```

[50]
1st : QRELAIKKNLELLNRYPNPLTIEKLSDFFSKMEMEKESSLVYENAIKKY
2nd(d): ?SHHHHHHHHHHHHT???HHHHHHHHHHHHHTT?HHHHHHHHHHHHHH?
2nd(s): OOHHHHHHHHHHHHHOOOOHHHHHHHHHHHHHHOOOHHHHHHHHHHHHHO
Switch: ??1000000000001?00010000000000010011000000000000

[100]
1st : PVSTETLCLSWFDNSIEKYDFKVFNRIFMYLNKNGKSRLHTLWYAFSFHL
2nd(d): GGGHHHHHHHHHHHTTS??HHHHHHHHHHHHHT??HHHHHHHHHHHHH
2nd(s): HHHHHHHHHHHHHHHOOOOHHHHHHHHHHHHHHOOOOHHHHHHHHHHHHH
Switch: 000000000000000111001110000000001??00000000000000

[150]
1st : LLQEGETDKASLYNSLGKKLMEGLQPFENTQEIIYVYTLFLSSKEIEQVLS
2nd(d): HHTT??HHHHHHHHHHHHHHHTT?S??SHHHHHHHHTT??HHHHHHHT
2nd(s): HHOOOOHHHHHHHHHHHHHHHHOOOOOOHHHHHHHHOOOOHHHHHHHHO
Switch: 0000?001

[200]
1st : GVTLPDLLELKLKLYMKAMKENASFEALHAYTEKLLFKEKFDDFDTWKLWI
2nd(d): T??S??HHHHHHHHHHHHHTT?HHHHHHHHHHHHHS??HHHHHHH
2nd(s): OOOOOOHHHHHHHHHHHHHHOOOHHHHHHHHHHHHHHOOOOHHHHHHHH
Switch: 000

[250]
1st : LSGKEIGKSFEELDQKLTSPTRNISLLKIELDILYSRNIETSVENYYQKF
2nd(d): HHHHHTT??HHHHHTT??S??HHHHHHHHHHHHHTT??SHHHHHHHHHHH
2nd(s): HHHHHOOOOHHHHHHOOOOOOHHHHHHHHHHHHHHOOOOHHHHHHHHHHH
Switch: 00000000000001000000000000000000000000000000000

[300]
1st : NTKLCCYADLSQYELPTSFIGSLKNSTSEENLITVNNRKFVNQTDNWDV
2nd(d): TTSTTHHHHHTTS??HHHHH??*?*??SHHHHHHHHHHTT??HHH
2nd(s): OOOOOHHHHHOOOOOOHHHHHHOOOODDOOOHHHHHHHHHHOOOOOOHHH
Switch: 00001000000000000001????000000000000000100000000

[350]
1st : YERFSTKEGAEYDSNPVNELTLRTIVSDLDSSPQNTIKNIVLLKHLLEQD
2nd(d): HHHS?S??SSTTS??THHHHHHHHHHHH??S?HHHHHHHHHHHHHHHS
2nd(s): HHOOOOOOOOOOOOHHHHHHHHHHHHOOOOHHHHHHHHHHHHHHHHO
Switch: 0001000

[400]
1st : KYN YKLKLWLMKLYSQLNTNDLIFPIYNGLKIRMTQHETLNYYLTTTNPS
2nd(d): TT?HHHHHHHHHHHTTBS?GGGSHHHHHHTT??THHHHHHGGGGT????
2nd(s): OOOHHHHHHHHHHHOOSOHHHOHHHHHHHOOOOHHHHHHHHHHHOOOOO
Switch: 000000000000000000000000010000000000000000000000000

[450]
1st : KINLDAWVDIYRFYLT SKQEIKESI IQGFDNGVFNKLEGFINFSKRMQNS
2nd(d): HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHTT?HHHHHHHHHHHHHHHTB
2nd(s): HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHOOOHHHHHHHHHHHHHHHOS
Switch: 000

[500]
1st : ISLNFTVAKILQISTILGTDGYLNYFIHYLKTNEALIVSDYTDNRDFKSE
2nd(d): HHHHHHHHHHHHHHHHTT?HHHHHHHHHHHHH?HHHHTS??B????SSGG
2nd(s): HHHHHHHHHHHHHHHHHHOOOHHHHHHHHHHHHHHHOHHHHHOOOOSOOOOOOHH
Switch: 000000000000000000001110000000000000000000000000000

[550]
1st : WNGLEKIDCIDVPVNDVATKLLVYSIVFEDQDASRLKLVFNKITSNAK
2nd(d): GTTS???S?????HHHHHHHHHHHHHT?SS??HHHHHHHHHHHTT??
2nd(s): HOOOOOOOOOOOOHHHHHHHHHHHHHHHOOOOOHHHHHHHHHHHHHOOOO
Switch: 0001000

[600]
1st : FSVFDNLLYKLYFNLLKITKTKLNPQETQSLYNYLQKNLKTDKLKILIP
2nd(d): ??HHHHHHHHHHHHHHHHHHHS??HHHHHHHHHHHHHSSHHHHGGGS?S
2nd(s): OOHHHHHHHHHHHHHHHHHHHHOOOHHHHHHHHHHHHHHHOHHHHHHHOOO
Switch: 00000000000000000000??0000000000000000000000000000

[650]
1st : NLLSGELTQNLTNLVEFIKIVKLLAKRHPSSYMNQLVNLVKPFGKEFKNL
2nd(d): STTBTHHHHHHHHHHHHHHHHHHHHS??HHHHHHHHHHTTHHHHHHTT
2nd(s): OOSOHHHHHHHHHHHHHHHHHHHHHHOOOHHHHHHHHHHHOHHHHHHHO
Switch: 0001000000000

[700]
1st : KLVQRQHEIIDSMDFEPPISVDISQTKLEIKSSIEDCVVALLNS
2nd(d): ?HHHHHHHHHHH????SS??SHHHHHHHHHHHHHHHHHHHHTT
2nd(s): OHHHHHHHHHHHHOOOOOOOOOHHHHHHHHHHHHHHHHHHHHHO
Switch: 000000000001000000000010000000000000000000000011

>5K9N:A

[0]

1st : MAKPIADDIVVRQVDVGETEQLMTFLLAHYYPEEPLTAGTHPPEPEAADK
2nd(d): *****??E???TTS?HHHHHHHHHHHTTTT?HHHHTSSS?S??HHHH
2nd(s): DDDDDDOOSSSSSOOOOOHHHHHHHHHHHOOOOHHHHOOOOOOHHHH
Switch: 000000?000000001111000000000000000000100000000000

[50]

1st : EFLLSNVPFGTCFVALHEGRIVA AVVAGPKDSHEPEHMAEEARKYAGGKW
2nd(d): HHHHTTGGGT??EEEEETEEEEEEEEEEEE?TTHHHHHHHHHHHHTTSHH
2nd(s): HHHHOOHHHOOOSSSSSSOOSSSSSSSSSSSSOOHHHHHHHHHHHHHOOHH
Switch: 000

[100]

1st : GSILHLLSAVETATDVCRRFSVPSCLVHVALGVDPQLRGRNLGGRLMETV
2nd(d): HHHHHHHHHHHHHH?HHHHTT?S?EEEEEEEE?GGGTTS?HHHHHHHHH
2nd(s): HHHHHHHHHHHHHHOOHHHHOOOOOSSSSSSSSSOHHHOOOOHHHHHHHHH
Switch: 000

[150]

1st : AQRGRDLGHQLVSVDCTSVYSARLVQRLGYQLINTLRYVDHLDASGQQVI
2nd(d): HHHHHHHT??EEEEEE?SHHHHHHHHHHTT?EEEEEEGGG?B?TTS?BS?
2nd(s): HHHHHHHOOOSSSSSSSOHHHHHHHHHHHOOOSSSSSSSHHHHOSOOOOOSOO
Switch: 0000001000

[200]

1st : RPPPPHESVQTFVLH
2nd(d): ???TT??EEEEEEEE
2nd(s): OOOOOOSSSSSSSS
Switch: 0000000000000000

>5KTA:A

[0]

1st : MVNFNKANTTYLRLVEENDA EFICTLRNNDKLNTYISKSTGDIKSQAEW
2nd(d): *????EE?SSEEEEE??GGGHHHHHHHHHHHHH?STT??S????HHHHHHH
2nd(s): DOOOOSSOOOSSSSSSOOHHHHHHHHHHHHHHHOOOOOOOOOOHHHHHHH
Switch: 0000010000000000000000000000111100000000000000000

[50]
 1st : IRNYKNRENNGEEYYFIIIFRSDDQSPIGTVRLYDFHENPKSFCWGSWILN
 2nd(d): HHHHHHHHHTTS?EEEEEEETTT??EEEEEEEEEEETTTTEEEEEEEEE?
 2nd(s): HHHHHHHHHOOOOSSSSSSSSOOOOSSSSSSSSSSSSOOOOSSSSSSSSSO
 Switch: 00000000000010000000000010000000000000000000000

[100]
 1st : EHKT KYAAVESALLVYEAGFSTLGFEQSHFEVMKGNDKVHSFHLKMGAQK
 2nd(d): TT??TTHHHHHHHHHHHIIIIIS??SEEEEEETT?HHHHHHHHHHT?EE
 2nd(s): OOOOOHHHHHHHHHHHHHHHHOOOOSSSSSSSSOOHHHHHHHHHHHOSS
 Switch: 000

[150]
 1st : ISEDDENIYYIFPKSKYKENKIQYARFLG
 2nd(d): EEE?SSEEEEEEEHHHHHHHHHHHTHHHH?
 2nd(s): SSSOOOSSSSSSSHHHHHHHHHHOHHHHO
 Switch: 00000000000000000001111000000

>5WIF:A

[-3]
 1st : MSTTSSVRLRPLEREDLPFVHQLDNNASIMRYWFEEPYEAFVELCDLYDK
 2nd(d): *****?EEEE?GGGHHHHHHT?***??EEETTEE?SHHHHHHHHH
 2nd(s): DDDDDOSSSSSOHHHHHHHHHOODDDOOOSSSOOSSSOHHHHHHHHHH
 Switch: ?????010000000000000100???0000000000000000000000

[47]
 1st : HIHDQSERRFIIESQGTKIGLVELVEINHIHRRAEFQIIIDPTHQKGKYA
 2nd(d): TTT?TT?EEEEEEETEEEEEEEEEEETTTTEEEEEEE?GGGTTSSH
 2nd(s): OOOOOOSSSSSSSOOSSSSSSSSSSSSSOOOOSSSSSSSOHHHOOOOH
 Switch: 000111000

[97]
 1st : GAAAKLAMEYGFSVLNLYKLYLIVDKENKAIHIYSKLGFEIEGELKQEF
 2nd(d): HHHHHHHHHHHHHTS??SEEEEEETT?HHHHHHHHHHT?EEEEEEEEEE
 2nd(s): HHHHHHHHHHHHHOOOOOSSSSSSSSOOHHHHHHHHHHHOOSSSSSSSSSS
 Switch: 000000000000100000000000000000000000000100000000000

>5YGE:A

```
[50]
1st      : WVAEHPDLYGKVVGCGALHVLWSDLGEIRTVAVDPAMTGHGIGHAIVDRL
2nd(d)   : EEEE?TTTTT?EEEEEEEEEEETEEEEEEEEEE?GGGTSSHHHHHHHHHH
2nd(s)   : SSSSOOOOOSOSSSSSSSSSSSOOSSSSSSSSSOHHHOOOOHSHHHHHHHHH
Switch: 0000111100100000000000000000000000000000000000000000
```

```
[150]
1st      : VAEFLDLSYVKPNILGNSRMLLV
2nd(d)   : HHTT??HHHS??SS??EEEEEE
2nd(s)   : HHOOOOHHHHOOOOOOOSSSSSS
Switch:  000000000000000000000000
```

B.2 Test Set

>2B4Y:A

```
[0]
1st      : MRPLQIVPSRLISQLYCGLKPPASTRNQICLKMARPSSSMADFRKFFAKA
2nd(d)   : *****?B??HHHHHHHHHHH?
2nd(s)   : DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDOOSOOHHHHHHHHHHHO
Switch:  00000000000000000000000000000000???????010000000000110
```

```
[50]
1st      : KHIVIIISGAGVSAESGVPTFRGAGGYWRKWQAQDLATPLAFAHNPSRVWE
2nd(d)   : SEEEEEE?GGGGGGGT?*****??BTTB?HHHHSSHHHHHH?HHHHHH
2nd(s)   : OSSSSSSOHHHHHHHOODDDDDDDOOSOOSHHHHHOHHHHHHHOHHHHHH
Switch:  000000001111111?????????11100110001001100000111000
```

```
[100]
1st      : FYHYRREVMGSKEPNAGHRAIAECETRLGKQGRRVVITQNIDELHRKAG
2nd(d)   : HHHHHHHHHHT????HHHHHHHHHHHHHHHTTT?EEEEEE??SS?HHHHHT
2nd(s)   : HHHHHHHHHHOOOOOHHHHHHHHHHHHHHHOOOOSSSSSSOOOOOHHHHHO
Switch:  0000000111100000000000000011111000000000000110010
```

```
[150]
1st      : TKNLLEIHGSLFKTRCTSCGVVAENYKSPICPALSGKGAPEPGTQDASIP
2nd(d)   : ??SEEEETEEEEEEEEETTT??EEE???SSSSGGGT????STT???????
2nd(s)   : OOSSSSOOSSSSSSSSSOOOOOSSSOOOOOOOHHHOOOOOOOOOOOOOOOOO
Switch: 0000000000000000000000000000000000000000000000000000000
```

```
[200]
1st      : VEKLPRCEEAGCGLLRPHVWFGENLDPAIL EEVDRELAHCDLCLVVG T
2nd(d)   : GGG S?B??STTT??BEEEEEE??TT????HHHHHHHHHHHHH?S EEEEEES?
2nd(s)   : HHHOOSOOOOOOOOOSSSSSSSOOOOOOOOHHHHHHHHHHHHHHHOOSSSSSSOO
Switch:  11100000111000000000000001000000000100011000000010
```

```
[250]
1st      : SSVVYPAAMFAPQVAARGVPVAEFNTEETTPATNRFRHFHQGPCGTTLPEA
2nd(d)   : ?S?STTGGGHHHHHHHHTT??EEEEESS??TTGGGSSEEEES?HHHHHHHH
2nd(s)   : OOOOOHHHHHHHHHHHOOOOSSSSSOOOOOHHHHOOSSSSOHHHHHHHH
Switch:  00011011111000011000000010?????111?010010011100000
```

```
1st      : LACHENETV
2nd(d)   : T???????
2nd(s)   : OODDDDDDD
Switch:  1?0000000
```

[illegible][illegible][illegible][illegible]

```
[200]
1st      : SAISVTNVEDPLAKKQTVRLIKDLQRAINKVLCTRLRLSNFFTIDHFIQK
2nd(d)   : *****??E????B?EE?*****??????TT??SHHHHHHH
2nd(s)   : DDDDDDDDDOOSSSSSOOSOSODDDDDOooooooooooooooooHHHHHHH
Switch:  000000000?00111111111111111111111111111100000000000000000000
```


[illegible]

```
[299]
1st      : EYFRKDPRPFFKFAKEIYPGQFQPSLCHKFIALSDKEGKLLRNYTQNIDT
2nd(d)   : HHHHHH??HHHHHHHHHHH?TTSS??H?HHHHHHH?*?TTS?EEEEEE??S?
2nd(s)   : HHHHHOOHHHHHHHHHHHHOOOOOOOOHHHHHHHHODOOOOOSSSSSSSOOOOO
Switch: 000000000000000111000000000000001?1100000000000000
```

```
[349]
1st      : LEQVAGIQRIIQCHGSFATASCLICKYKVDCEAVRGDIFNQVVPRCPRCP
2nd(d)   : HHHHHT??SEEETEEEEEEETT??EEEGGGGHHHHHTT???B?SSS?
2nd(s)   : HHHHHOOOOSSSSOOSSSSSSSSSOOOOOSSSHHHHHHHHHHOOOOOSOOOOO
Switch: 0000100000000000000000000000000000000000000000000000000
```

```
[399]
1st      : ADEPLAIMKPEIVFFGENLPEQFHRAMKYDKDEVDLLIVIGSSSLKVRPVA
2nd(d)   : TT?TT?BEEEEEE??BT????HHHHHHHHHHTTT?SEEEEEES????STTGG
2nd(s)   : OOOOOSOSSSSSOOSOOOOHHHHHHHHHHHOOOOOSSSSSSOOOOOOOOH
Switch: 00000000000000000000000000000000000000000000000000
```

[illegible][illegible][illegible][illegible]


```
[200]
1st      : RHEYPLSWMKEKIFSEVTPKCEDCQSLVKPDIVFFGESLPARFFSCMQSD
2nd(d)   : ??EE?HHHHHHHHHTT???B?SSS??BEEEE?BTSTB?HHHHHHHHHH
2nd(s)   : OOSSOHHHHHHHHHOOOOOSOOOOOOSSSSSSOOSOOSOHHHHHHHHHH
Switch:  000010000000011000000000000000000000001000000001111
```

```
[250]
1st      : FLKVDLLLVMGTSLQVQPFASLISKAPLSTPRL LINKEKAGQSDPFLGMI
2nd(d)   : TTS?SEEEEEES????EETTGGGGGSS?TT??EEEEESS????S??GGG?
2nd(s)   : OOOOOSSSSSOOOOOSSOOHHHHHOOOOOOOSSSSSSOOOOOOOOOOHHHO
Switch:  000000000000000110100000100000000000000000000000000011??
```

```
[300]
1st      : MGLGGGMDFDSKKAYRDVAWLGECDQGCLALAE LLGWKKELEDLVRREHA
2nd(d)   : SS?????SSSTT??SEEEEEES?HHHHHHHHHHHHT?HHHHHHHHHHHHHH
2nd(s)   : OOOOOOOOOOOOOOOOSSSSSOOHHHHHHHHHHHHHOHHHHHHHHHHHHHH
Switch:   ?????00000000000000000000000000000000000000000000
```

```
[350]
1st      : SIDAQSGAGVNPSTSASPKKSPPPAKDEARTTEREKP
2nd(d)   : HHHHH?*****
2nd(s)   : HHHHHODDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Switch: 0001100000000000000000000000000000000000
```

>5BTR:A

[illegible][illegible]

Appendix C

DESCRIPTOR OVERLAYS

C.1 Learning Set

Fig. 12: Decriptors for 1CJW:A

Fig. 13: Decriptors for 1GX3:A

Fig. 14: Descriptors for 1IB1:A

Fig. 15: Descriptors for 1N71:A

Fig. 16: Descriptors for 1S7F:A

Fig. 17: Descriptors for 1TIQ:A

Fig. 18: Decriptors for 1XEB:A

Fig. 19: Decriptors for 2B5G:A

Fig. 20: Descriptors for 2BL1:A

Fig. 21: Descriptors for 2FIA:A

Fig. 22: Decriptors for 2JLM:A

Fig. 23: Decriptors for 2QV4:A

Fig. 24: Decriptors for 2REE:A

Fig. 25: Decriptors for 2VFB:A

Fig. 26: Decriptors for 2VI7:A

Fig. 27: Decriptors for 2WPW:A

Fig. 28: Decriptors for 3FS8:A

Fig. 29: Decriptors for 3MQH:A

Fig. 30: Descriptors for 3N37:A

Fig. 31: Descriptors for 3NE7:A

Fig. 32: Decriptors for 3QB8:A

Fig. 33: Decriptors for 3SMA:A

Fig. 34: Decriptors for 3TFY:A

Fig. 35: Decriptors for 3W6S:A

Fig. 36: Decriptors for 4AVA:A

Fig. 37: Decriptors for 4EAB:A

Fig. 38: Decriptors for 4FD4:A

Fig. 39: Decriptors for 4FD5:A

Fig. 40: Descriptors for 4FD7:A

Fig. 41: Descriptors for 4HNY:A

Fig. 42: Descriptors for 4J3G:A

Fig. 43: Descriptors for 4JJX:A

Fig. 44: Decriptors for 4KVO:A

Fig. 45: Decriptors for 4KVX:A

Fig. 46: Descriptors for 4PV6:A

Fig. 47: Descriptors for 4R3K:A

Fig. 48: Descriptors for 4R9M:A

Fig. 49: Descriptors for 4RI1:A

Fig. 50: Decriptors for 4U9V:B

Fig. 51: Decriptors for 4UA3:A

Fig. 52: Decriptors for 5GI5:A

Fig. 53: Decriptors for 5HGZ:A

Fig. 54: Descriptors for 5JPH:A

Fig. 55: Descriptors for 5K04:A

Fig. 56: Decriptors for 5K9N:A

Fig. 57: Decriptors for 5KTA:A

Fig. 58: Decriptors for 5WIF:A

Fig. 59: Decriptors for 5YGE:A

C.2 Test Set

Fig. 62: Descriptors for 4IG9:A

Fig. 63: Descriptors for 4L3O:A

Fig. 64: Decriptors for 5BTR:A

Appendix D

BOX PLOTS

Fig. 65: IsUnstruct values from the learning set, broken down by residue type.

Fig. 66: E6 values from the learning set, broken down by residue type.

Fig. 67: E20 values from the learning set, broken down by residue type.

Fig. 68: IsUnstruct values from the learning set, broken down by residue type.