02-201 Homework 8:
Protein Folding & Simulated Annealing

Due: Nov 25, noon

1. Set up

The setup is the same as in previous assignments. To do the assignment and extra credit, you will need the `canvas.go` file along with the `code.google.com` directory in your `src` directory. If you have your setup from previous experiments, you should have most of this completed already, and will only need to create the `fold` directory (step 3) and copy `canvas.go` into it.

1. Create a directory called "`go`" someplace (different than where you have installed Go) [If you've already done this, you don't have to do it again.]
2. Inside of that directory create a directory called `src` [If you've already done this for a previous assignment, you don't have to do it again.]
3. Inside of the `src` directory, create a directory called `fold`.
4. Download the template `canvas.go` from BlackBoard (or copy it from a previous assignment) and put it into the `fold` directory. Also copy over the `code.google.com` directory from a previous assignment if it isn't already in your `src` directory.
5. Set your GOPATH environment variable to the location of your `go` directory that you made above. On a Mac:

   ```
   export GOPATH=/Users/carlk/Desktop/go
   ```

   where you replace the directory name after the `=` with the location of the `go` directory you just made.

   On Windows use

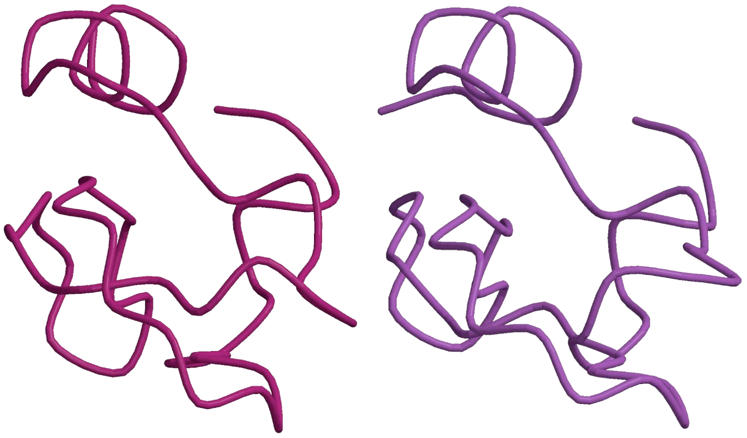
   ```
   set GOPATH=C:\Users\carlk\Desktop\go
   ```

2. Assignment

2.1 Protein folding

Proteins are linear molecules that primarily consist of a chain of amino acids strung together. There are 20 commonly occurring amino acids in most organisms. An important challenge in biology and computational biology is to predict the three-dimensional structure of a protein given its sequence of amino acids. The assumption that is made is that a protein will fold up into its lowest-energy conformation. So if we assume we have a function energy(S) that takes a structure and evaluates its energy, we are looking for a structure S that minimizes energy(S). While a lot of progress has been made on this problem over the years (including by the winners of the 2013 Nobel prizes in Chemistry), the problem remains hard.

Here are two example protein folds and their amino acid sequences:



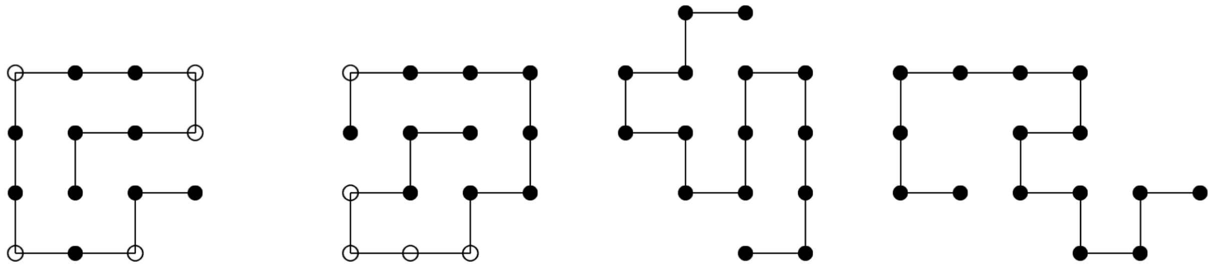
(a) 1dtk (b) 1dtk

```
1dtk  XAKYCKLPLRIGPCKRKIPSFYYKWKAKQCLPFDYSGCGGNANRFKTIEECRRTCVG-
5pti  RPDFCLEPPYTGPCKARIIRYFYNAKAGLCQTFVYGGCRAKRNNFKSAEDCMRTCGGA
```

2.2 The HP model

In order to make progress on several aspects of the protein folding process, a simplified model — introduced by Dill[1] — has been studied. This model is called the "HP model" because there are only two types of amino acids: "H", which are hydrophobic (like to be away from water), and "P" which are polar and don't mind being near the water. So, the first simplification of the HP model is that there are 2 instead of 20 types of amino acids.

The second simplification in our case is that proteins are modeled in 2-dimensions instead of 3, and they fold on a square lattice. So the following are examples of a HP protein fold, where the circles represent amino acids and the solid nodes are the "H" amino acids:



(Here the leftmost HP protein has the sequence HHPHPHHPHHPPHHH.) We can think of these folds as (non-random) walks along a square lattice.

We represent such a walk by a sequence of relative directives: right (r), left (l), forward (f). The walk is described as though you were giving driving directions. For example, a walk that looks like a "z" would be represented as $\langle r, r, l \rangle$ (we assume that initially, the "walker" is facing up). Almost all walks can be described in this way since at any point in the walk there are three choices for the next point — you cannot go backwards. The only exception to this is the first point: in the above scheme you cannot represent the walk that looks like a "u" because there is no way to

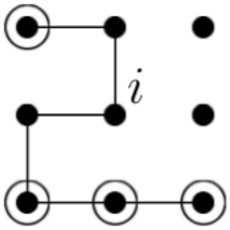
[1] Dill. Theory for the folding and stability of globular proteins. *Biochemistry* 24(6):1501–9, 1985

specify "go backwards" at the first point. Fortunately, this does not matter since we don't care about the orientation of the protein. For example, the leftmost structure in the row of 4 above, is specified by $\langle l, l, r, f, r, f, f, r, f, f, r, r, f, l \rangle$. This sequence of instructions is called the *fold* or the *structure*.

The last simplification is the energy function. When dealing with real proteins, the energy function usually is a combination of many physical energies. For the HP model the energy function measures how buried the "H" atoms are. Let the amino acid sequence be given by $\langle p_1, p_2, \ldots, p_n \rangle$, where $p_i = 1$ if the ith amino acid is an "H" and $p_i = 0$ otherwise. Then the energy of a fold is:

$$\text{energy}(S, \vec{p}) = 10x - \sum_{i=1}^{n} p_i s_i \tag{1}$$

where s_i equals the number of amino acids at neighboring lattice points, including diagonals, that are not adjacent to the ith point in the walk, and x is the number of times the walk crosses itself. In this example,



$s_i = 4$ because of the 8 nearby points, 4 contain amino acids and are not adjacent to i in the structure.

Our goal in this assignment will be to write a program that is given an HP protein sequence $\vec{p}$ and finds a structure S (walk along a square lattice) that minimizes energy$(S, \vec{p})$. Alternatively, we want to find a *non-crossing* structure S that minimizes energy$(S, \vec{p})$.

2.3 Simulated annealing

To find a low energy structure, we will use simulated annealing, which is a widely useful optimization framework. It depends on several parameters: T, k, m, and it works as follows:

1. Set $i = 1$; Choose a random structure S_i (i.e. a random sequence of l, r, f).
2. Compute $e_i = \text{energy}(S_i, \vec{p})$.
3. Change a random letter of the walk S_i to a random different letter to obtain a slightly different structure S_i'.
4. Compute $e_i' = \text{energy}(S_i', \vec{p})$.
5. If $e_i' < e_i$, set $S_{i+1} = S_i'$; otherwise, if $e_i' > e_i$ let $q = \exp(-(e_i' - e_i)/kT)$ and with probability q set $S_{i+1} = S_i'$, and with probability $(1 - q)$ set $S_{i+1} = S_i$.
6. Let $i = i + 1$ and if $i\%100 = 0$ let $T = 0.999T$.
7. If the structure hasn't changed for m iterations, stop; otherwise go to step 2.

8. Return the S_j with the lowest energy.

In other words, we keep making random small changes to the structure. If the change improves the structure (lower energy), we keep the change. If the change doesn't improve the energy, we keep the change with probability $\exp(-\Delta)/kT)$ where Δ is the amount by which the energy went up. The idea here is that we always accept improvements, and we accept "bad" changes with probability inversely proportional to how bad they are. The constant k is a normalizing factor to scale the energy differences.

Variable T is the interesting one: when T is large, we are more likely to accept changes that increase the energy, and when T is very small, we nearly never do. T can be viewed as a temperature of the system: when the system is "hot", we bounce around a lot, and when it is "cool" we head towards low-energy solutions.

Reasonable choices for the parameters for a protein of length n are $m = 10n$, $k = 6n$, and the initial $T = 10n$. But you may need to modify the exact algorithm / parameters to have your program find good structures.

2.4 What you should do

You should write a Go program that can be run with the following command line:

```
./fold HP
```

where `HP` is a string of H and Ps representing the HP protein sequence.

Your program should output two lines of text:

```
Energy: ENERGY
Structure: lrf
```

where `ENERGY` is the energy (integer) of the final structure, and `lrf` is a sequence of l, r, f that specifies the lowest energy structure you found.

Extra credit: You should also write a file called `fold.png` that draws the structure. You can use any nice format for the drawing that makes the Hs dark colored and the Ps light colored. For example using edges of length 10 pixels, and drawing the H-residues as solid black squares and P-residues as solid white squares with a black boarder.

You can use *any algorithm you want* to find a low-energy structure. Of course, simulated annealing is a good choice, but you can modify simulated annealing if you need to in order to obtain a better structure. You can also modify aspects of the basic simulated annealing algorithm above. For example, you can change how often T is changed (maybe even sometimes increasing it), or you could run several independent simulated annealing runs from different starting points, and return the best solution found, etc.

2.5 Tips on how to start

First write the main function to parse the arguments from the command line.

Then write a function `randomFold` that generates a random fold as a sequence of l, r, and f commands.

Then, write a function `drawFold` that "draws" the fold onto a sufficiently large 2D matrix.

Next, write and test a function `energy(S, p)` that computes the energy (according to equation 1) of a fold S and sequence p. This function probably calls `drawFold`.

Next, write a `randomFoldChange` function that takes a fold and randomly changes one of its commands.

Next, write `optimizeFold(p)` that returns the lowest-energy fold you can find for p (likely by running the simulated annealing algorithm). Play around with the parameters and the algorithm to see if you can get better and better folds.

Finally, if you want to do the extra credit, write `paintFold` that draws the fold onto a canvas and saves the canvas to `fold.png`.

2.5.1 Tip for getting your program to run faster:

In the simulated annealing code, if you find a new structure with the **same** energy, don't switch to it.

2.5.2 Tips for making the code easier to write:

3. For the drawing (extra credit): you can use any format you want for the drawing so long as it shows the structure in a reasonable way.

4. The data structure the solution uses for laying out the fold is `[][]string`. Then `M[x][y]` lists all the residues that are in that position (if crossings are allowed). You can use

```
strings.Contains(M[x][y], "H")
```

to check if a cell position contains an H amino acid.

5. If you'd prefer, you can simply reject structures that have crossings (i.e. they have infinite energy)

6. Don't try to track which amino acids are adjacent to one another. Rather, lay them out in a 2D array, compute the score ignoring which residues are adjacent in the walk, and then subtract 2 for every H in the middle of the sequence, and 1 for every H at the ends of the sequence.

3. Learning outcomes

After completing this homework, you should have

- learned about and implemented a simulated annealing optimizer (or other approach for optimizing)
- learned about the HP protein folding model
- gained additional practice writing Go programs