CS106B Chris Piech

Winter 2016 Feb 7th, 2016

CS106B Midterm

This is an open-note, open-book exam. You can refer to any course handouts, textbooks, handwritten lecture notes, and printouts of any code relevant to any CS106B assignment. You may not use any laptops, cell phones, or internet devices of any sort. You will be graded on functionality—but good style helps graders understand what you were attempting. You do not need to #include any libraries and you do not need to forward declare any functions. You have 2 hours. We hope this exam is an exciting journey.

Last Name: **__SOLUTION____**

First Name: _____

Section Leader: _____

I accept the letter and spirit of the honor code. I've neither given nor received aid on this exam. I pledge to write more neatly than I ever have in my entire life.

(signed) _____

		Score	Grader
1. Mystery	[15]	_____	_____
2. Metric Space	[15]	_____	_____
3. Pandora	[15]	_____	_____
4. Facebook Degree	[15]	_____	_____
Total	**[60]**	**_____**	

Problem 1: Tracing programs and big-O (15 points)

Assume that the functions **`mystery`** and **`riddle`** have been defined as follows:

```
int mystery(int n) {
    int sum = 0;
    for(int i = n; i >= 1; i--) {
        sum += riddle(i);
    }
    return sum;
}

int riddle(int n) {
    int sum = 0;
    for(int i = 0; i < n; i++) {
        sum += i;
    }
    return sum;
}
```

(a) [3 points] What is the value of **`riddle(4)`**?

Answer: 6

(b) [4 points] What is the value of **`mystery(4)`**?

Answer: 10

(c) [4 points] What is the worst case computational complexity of the **`riddle`** function expressed in terms of big-O notation, where *N* is the value of the argument **`n`**?

Answer: O(n)

(c) [4 points] What is the worst case computational complexity of the **`mystery`** function expressed in terms of big-O notation, where *N* is the value of the argument **`n`**?

Answer: $O(n^2)$

Problem 2: Grids (15 points)

For a list of *N* elements, a "dissimilarity grid" of size *N x N* stores how different each element is from another. The value of **`grid[row][col]`** represents the dissimilarity between element with index **`row`** and element index **`col`**. If we can show that a dissimilarity grid is a "metric" then we can perform a set of important analyses.

Write a function that tests if a grid is a "metric". A grid is a metric if it obeys these rules:

1. **`numRows`** equals **`numCols`**. `// Square`
2. All elements in the **`grid`** are >= 0. `// Non-negative`
3. All cells where **`row == col`** are 0. `// Diagonal-zero`
4. Only cells where **`row == col`** are 0. `// Off-diagonal-non-zeros`
5. The **`grid`** must be symmetric. `// Symmetric`
6. The **`grid`** must obey triangle inequality. `// Triangle Inequality`

A grid is symmetric if for all values of **`row`** and **`col`** in the range 0 to N:
`grid[row][col] == grid[col][row].`

A grid obeys triangle inequality if for all values of **`a, b, c`** in the range 0 to N:
`grid[a][c] <= grid[a][b] + grid[b][c]`

Here is an example of a metric grid:

Lets say we collected the four DNA strands:

`"ATG", "GTG", "AATC", "AATG"`

Then using the DNA alignment algorithm seen in class we computed the distance for every pair of DNA. Those distances are store in a grid where **`grid[row][col]`** represents the distance between DNA strand index **`row`** and index **`col`**:

	"ATG"	"GTG"	"AATC"	"AATG"
"ATG"	**0**	**1**	**3**	**1**
"GTG"	**1**	**0**	**3**	**2**
"AATC"	**3**	**3**	**0**	**1**
"AATG"	**1**	**2**	**1**	**0**

Since the distance between "ATG" (index 0) and "AATG" (index 3) is 1, both **`grid[0][3]`** and **`grid[3][0]`** have the value 1. This grid is a "metric." It obeys all rules. That means we can do cool things like compute a hierarchy of relationships between the DNA strands.

Hint: the six criteria for a grid being a "metric" are listed in order of complexity. Even if you can't compute the hardest criteria, you will get partial credit for all criteria you check. The details of DNA distance are not important for this problem.

Answer to Problem 2:

```
bool isDistance(Grid<double> & m) {
   if(m.numRows() != m.numCols()) return false;
   for(int r= 0; r < m.numRows(); r++) {
       for(int c = 0; c < m.numCols(); c++) {
           if(m[r][c] < 0) return true;
           if(m[r][c] == 0 && r != c) return false;
           if(r == c && m[r][c] != 0) return false;
           if(m[r][c] != m[c][r]) return false;
           if(!cellTriangle(m, r, c)) return false;
       }
   }
   return true;
}

// helper
bool cellTriangle(Grid<double> & m, int r, int c) {
   double d_rc = m[r][c];
   // triple for loop to check for triangles
   for(int i = 0; i < m.numRows(); i++) {
       double d_ri = m[r][i];
       double d_ic = m[i][c];
       if(d_ri + d_ic < d_rc) return false;
   }
   return true;
}
```

Problem 3: Maps and Sets (15 points)

Pandora is an online radio station that is backed by what they call the "Music Genome Project". Your job is to use the Map that represents the Music Genome Project to generate a radio station corresponding to a seed song.

The Music Genome Project produced a map that associates each songName with a set of songProperties (such as fastRhythm, heavyBass, femaleVocals). Consider this example genome.

Example Music Genome map (associates song names with properties):

```
"Wake Me Up"          -> ["Syncopation", "Guitar", "Electonic"]
"Love Yourself"       -> ["Syncopation", "SoftMaleVocals"]
"Sky Full of Stars"   -> ["Syncopation", "SoftMaleVocals", "Dancing"]
"Hey Soul Sister"     -> ["Syncopation", "SoftMaleVocals", "Rock"]
"Riptide"             -> ["Guitar", "Upbeat", "Dancing"]
```

Given a seed song, you should return all songs in the music genome that have ***all*** of the properties of the seed song.

For example **`"Love Yourself"`** has the properties "Syncopation" and "SoftMaleVocals". Thus if Love Yourself was the seed song, you should produce the set of songs:

`["Love Yourself", "Hey Soul Sister", "Sky Full of Stars"]`.

Since each of those songs also has the properties Syncopation and SoftMaleVocals**.** The song **`"Wake Me Up"`** is not included because even though it has Syncopation it doesn't have SoftMaleVocals.

Answer to Problem 3:

```
Set<string> radioStation(Map<string,Set<string>>& genome, string seed) {
   Set<string> qualities = genome[seed];
   Set<string> songs;
   for(string key : genome) {
       if(qualities.isSubsetOf(genome[key])) {
           songs.add(key);
       }
   }
   return songs;
}
```

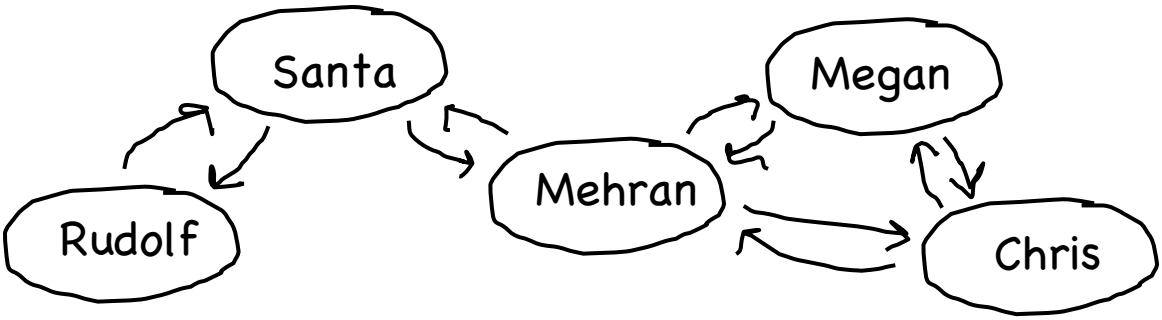
Problem 4: Recursive Exploration (15 points)

Facebook recently released an analysis of their friend network. They claimed that each person in the world (at least among the 1.59 billion people active on Facebook) is connected to every other person by an average distance of only three and a half connections! Distance is the minimum number of friend connections between two users.

Your job is to calculate the minimum distance between two given users. You can assume that everyone is connected to everyone else.

Consider this simple network with the following **`friendMap`**:

```
Santa   -> [Mehran, Rudolph]
Rudolph -> [Santa]
Mehran  -> [Santa, Megan, Chris]
Megan   -> [Mehran, Chris]
Chris   -> [Megan, Mehran]
```



This picture shows the network of friends described by the **`friendMap`**

The minimum distance between Megan and Chris is 1	[They are friends].
The minimum distance between Megan and Rudolf is 3	[Megan -> Mehran -> Santa -> Rudolf]
The minimum distance between Santa and Chris is 2	[Santa -> Mehran -> Chris]
The minimum distance between Mehran and Mehran is 0.	[Mehran *is* Mehran]

To find a *minimum* distance between a start name and a goal name use a type of recursive exploration called iterative deepening. First recursively check if you can find a path from start to goal with distance at most 0. Then if that fails try with distance at most 1. Continue recursively looking for a path with successively increasing distance. Eventually you will find a path with distance at most *n*. The minimum distance between start and goal is *n*.

For example, to find the min distance between Megan and Santa:
Is there a path from Megan to Santa with distance 0? No
Is there a path from Megan to Santa with distance 1? No
Is there a path from Megan to Santa with distance 2? Yes
Thus the min distance from Megan to Santa is 2.

Answer to Problem 4:

```
int minDistance(Map<str, Set<str>>& friendMap, str start, str goal){
   int distance = 0;
   while(true) {
       if(distance(friendMap, start, goal, distance)) {
           return distance;
       }
       distance++;
   }
}

// helper
bool distance(Map<str, Set<str>>& friendMap, str a, str b, int dist){
   if(a == b) return true;
   if(dist < 0) return false;
   for(string key : friendMap[a]) {
       if(distance(friendMap, key, b, dist - 1)) return true;
   }
   return false;
}
```