CS106B Handout

Winter 2016 Feb 6th, 2016

Practice Midterm #1 Solution

This is an open-note, open-reader 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 handheld 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: _____

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	[10]	_____	_____
2. Smooth Image	[15]	_____	_____
3. Who Has A Silly Password?	[15]	_____	_____
4. Mirror Temple	[15]	_____	_____
Total	**[55]**	**_____**	

Problem 1: Tracing C++ programs and big-O (10 points)

Assume that the functions **`Mystery`** and **`Enigma`** have been defined as follows:

```
int Mystery(int n) {
   if (n == 0) {
        return 0;
   } else {
        return Mystery(n - 1) + Enigma(n/2) + Enigma(n/2);
   }
}

int Enigma(int n) {
    int index = 0;
    int sum = 0;
    while (index < n * 2) {
        sum += index;
        index++;
    }
    return sum;
}
```

(a) [3 points] What is the value of **`Enigma(2)`**?

Enigma(2) = 6

(b) [3 points] What is the value of **`Mystery(4)`**?

Mystery(4) = 16

(c) [2 points] What is the worst case computational complexity of the **`Enigma`** function expressed in terms of big-O notation, where *N* is the value of the argument **`n`**? In this problem, you may assume that **`n`** is always a nonnegative integer.

O(n)

(d) [2 points] What would be the effect on the worst case big-O of Enigma if the line:

```
if(RandomChance(0.5)) break;
```

had been added to the beginning of the while loop in Enigma?

It will still run in O(n) in the worst case

Problem 2: Grids, Vectors, Stacks, and Queues (15 points)

A common operation on 2D data (think images or scientific data) is to smooth out the values of each cell. In images it blurs the picture, on scientific data it reduces the effect of outliers.

Your job is to write a function **`smooth`** that takes in a **`Grid<double>`** by reference, and smooths out every cell in the grid. On a single cell from the data, the smoothed value is the average of its neighbors and its own value. Most cells have eight neighbors, except for cells at the edge of the grid. Consider the following cell in blue, and its eight neighbors:

1.0	1.0	2.0
6.0	5.0	3.0
1.0	2.0	2.0

The average of the cell and its eight neighbors is $(1 + 1 + 2 + 6 + 5 + 3 + 1 + 2 + 2) / 9$ which equals 2.56. Thus the new value of the cell will be 2.56.

You should apply such an averaging to every cell in the grid (which could have any size). Importantly, when you compute the average of a cell, it should use the original value of the neighbors, not a smoothed value. In the example above, if you were calculated the smoothed value of one of the blue cell's neighbor it should use the old value of the cell (5) instead of the smoothed value (2.56).

Answer to problem 2:

```
void smooth (Grid<double> & values) {
   Grid<double> newValues(values.numRows(), values.numCols());
   for(int r = 0; r < values.numRows(); r++){
      for(int c = 0; c < values.numCols(); c++) {
         double smoothed = smoothCell(values, r, c);
         newValues[r][c] = smoothed;
      }
   }
   values = newValues;
}

double smoothCell(Grid<double>& values, int r, int c){
   double sum = 0.0;
   int num = 0;
   for(int deltaR = -1; deltaR <= 1; deltaR++){
      for(int deltaC = -1; deltaC <= 1; deltaC++){
         int currR = r + deltaR;
         int currC = c + deltaC;
         if(values.inBounds(currR, currC)) {
            sum += values[currR][currC];
            num++;
         }
      }
   }
   return sum/num;
}
```

Problem 3: Maps and Sets (15 points)

In any map, multiple keys can have the same value. The Most Common Value is the value in the map with the largest number of associated keys.

Your job is to write a function **`KeysForMostCommonValue`** that takes in a **`Map<string, string>`**, finds the Most Common Value and returns the set of keys that map to the Most Common Value.

```
Set<string> KeysForMostCommonValue (Map<string, string> & map)
```

Consider the following map which stores username password combinations:

Key	Value
"cpiech"	"password"
"mfaulk"	"pizza"
"zuckerberg"	"12345"
"haxor2000"	"password"
"sheen"	"12345"
"rebeccablack"	"friday"
"neo"	"password"

The Most Common Value in the map is **`"password"`** and the keys that map to the Most Common Value are **`"cpiech"`**, **`"haxor2000"`** and **`"neo"`**.

Hint: This problem is much easier to solve if you populate internal set(s) and/or map(s) that will help you find the Most Common Value.

(space for the answer to problem 3 appears on the next page)

Answer to problem 3:

```
Set<string> keysForMostCommonValue (Map<string, string> & map) {
     Set<string> valueSet = getValueSet(map);
     int maxCount = -1;
     string mcv = "";
     for(string value : valueSet) {
          int count = getKeySet(value, map).size();
          if (count > maxCount) {
               maxCount = count;
               mcv = value;
          }
     }
     return getKeySet(mcv, map);
}

Set<string> getValueSet(Map<string, string> & map) {
     Set<string> valueSet();
     for(string key : map) {
          string value = key[value];
          valueSet.add(value);
     }
     return valueSet;
}

Set<string> getKeySet(string goal, Map<string, string> & map) {
     Set<string> keySet();
     for(string key : map) {
          string value = key[value];
          if (value == goal) keySet.add(key);
     }
     return keySet;
}
```

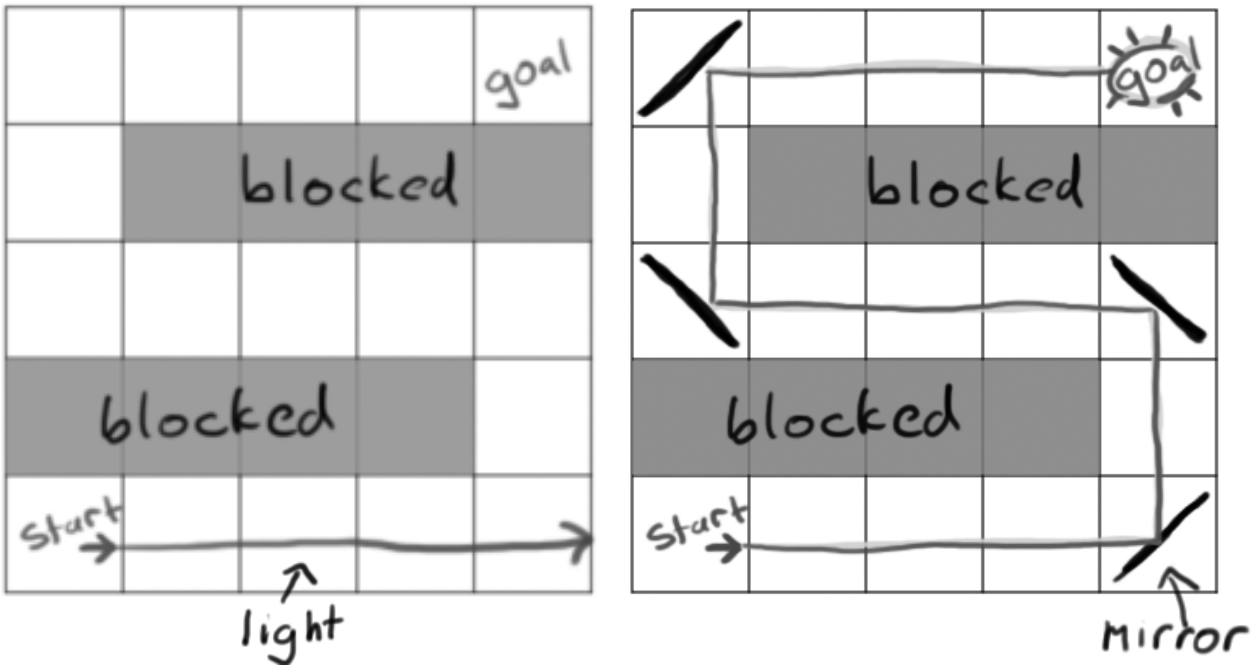
Problem 4: Recursive Exploration (15 points)

It has been thought that Ancient Egyptians used systems of mirrors to bring light into the inner chambers of their temples! Light normally travels in a straight line (think of a laser) but you can turn the path of the light using reflective surfaces.

Your job is to write a predicate function `IsSolvable` that returns whether or not there is a system of mirrors for a given temple such that light traveling in a specific direction from a starting position can be directed to a goal position using a *limited* number of mirrors.

This problem is constrained so that light is always traveling north, south, east or west and mirrors turn light clockwise (to the right relative to the previous direction) or counterclockwise (to the left). The temple is stored as a `Grid<bool>` where `true` means that the location is blocked—light cannot travel through this location—and `false` means that the location is open—light can travel through this location.

As an example: in the temple depicted in the image below-on-the-left, light starts in the bottom left corner traveling to the east and the goal is to get the light to the upper right corner. In this example there is no solution with three or fewer mirrors. However using four mirrors the system in the image below-on-the-right will direct light to the goal:



Hint: You should think of this problem step by step from the perspective of the light. At each square you can: move straight, turn left or turn right. It costs nothing to move straight—in the direction you were already facing—but it costs a mirror to turn left or to turn right..

The problem uses the Point class to represent a position in the grid and the Direction enum to represent light direction:

```
// Directions that light can travel
enum Direction = {NORTH, EAST, SOUTH, WEST};

// Position in the temple
Point is a class in the cs106b library that stores an x and y position as
ints. The two methods it supports are getX() and getY().

Point myPoint(2, 3);
int x = myPoint.getX();
int y = myPoint.getY();
```

To help you with the updating of positions and directions, we provide you with the following functions (you **don't** have to write them):

```
Point move(Point curPos, Direction curDir);
Direction turnLeft(Direction curDir);
Direction turnRight(Direction curDir);
```

`Move` returns the new position obtained by taking one step from the passed-in-position oriented in the passed-in-direction. `TurnLeft` and `TurnRight` return the orientation obtained from rotating left and right respectively from the passed-in-direction.

Using these predefined functions write a recursive predicate `IsSolvable` that takes a grid, a number of mirrors, a start configuration and a goal position as arguments and returns `true` if there is a system of mirrors to direct light to the goal and `false` otherwise.

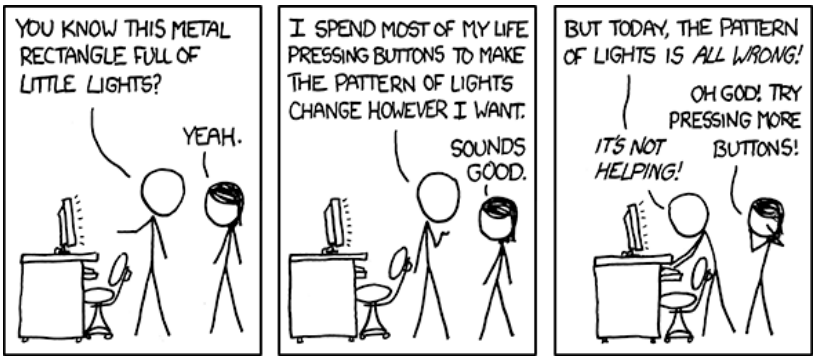
(space for the answer to problem 4 appears on the next page)

Answer to problem 4:

```
// predefined helper functions (you can use and don't have to implement)
Point Move(Point curPos, Direction curDir);
Direction TurnLeft(Direction curDir);
Direction TurnRight(Direction curDir);


bool IsSolvable(Grid<bool> templeBlocks, int numMirrors, Position startPos,
                  Direction startDir, Position goalPos) {

   // check if you are outside the grid
   if(!inBounds(startPos.getX(),startPos.getY())) return false;
   // check if you have hit a blocked cell
   if(!blocks[startPos.getX()][startPos.getY()]) return false;
   // check if you are out of mirrors
   if(numMirrors < 0) return false;
   // check if you have found the goal
   if(lightPos == goalPos) return true;

   // compute all the decisions
   Direction left = TurnLeftDir(startDir);
   Direction right = TurnRightDir(startDir);
   Position forward = Move(startPos, startDir);

   // check if there is a solution from any decisiom
   if(IsSolvable(blocks, numMirrors-1, startPos, left, goalPos) return true;
   if(IsSolvable(blocks, numMirrors-1, startPos, right, goalPos) return true;
   if(IsSolvable(blocks, numMirrors, forward, startDir, goalPos) return true;
   return false;
}
```

Problem 5: Optional Fun Problem (0 points)

Make your own paneled style stick figure comic pontificating or commenting about CS, computers, life, etc. Here's one for reference:



This problem isn't required and there are no points associated with doing any work here, but any great gems will be very much appreciated by the course staff :).