CSCI 104 - Fall 2018

Data Structures and Object Oriented Design

Homework 6

- Due: Friday, November 2, 11:59pm
- Directory name in your github repository for this homework (case sensitive): `hw6`.
- Provide a `README.md` file.

Problem 1 (Graph Search, 20%)

Write a program to find the largest country in a map. Typing `make search` should create an executable called `search` which solves the following problem.

The input file (whose name will be passed at the command line) will have three parameters on the first line: how many different letters there are in the map, the number of rows in the map, and the number of columns in the map. The countries are identified by contiguous identical characters; in other words, starting from an arbitrary point in a country, its borders are defined by where you can walk by moving horizontally or vertically while remaining on the same character. Here is an example of what the input might look like:

```
3 6 13
AAAAAABBBBBBB
AAAAAABBBBBBA
BBBAAABBBBBAA
BBAAAABBCCAAA
BBBBBABACCCAA
BBBBBBAAAAAAA
```

In this example, there are two countries designated by `A`: one in the top left (of size 20), and another in the bottom right (of size 16). They indicate two different countries because you cannot get from one set of A's to the other by

only walking horizontally or vertically within their bor
There are another two countries designated by B (of
21 and 16), and one country designated by C (of size

You should output to terminal a single number indicating the size of the largest country (in the above example, 21).

Your algorithm must run in `r*c` time, where r is the number of rows, and c is the number of columns.

Problem 2 (Recursion and Backtracking - Graph Coloring, 35%)

Write a program to 4-color a map using recursion and backtracking. Typing `make coloring` should create an executable `coloring` which solves the following problem.

The input file (whose name will be passed at the command line) will have three parameters on the first line: how many "countries" there are in the map, the number of rows in the map, and the number of columns in the map. This time, the map will never be bigger than 80 by 80 characters, and will contain at most 15 countries. The countries are denoted by characters 'A', 'B', ... up to the letter for the highest country number: for this problem, each country gets its own designated letter (two disjoint countries will never share a letter). Here is an example of what the input might look like:

```
5 6 13
AAAAAACCCCCCC
AAAAAACCCCCCD
BBBAAACCCCCDD
BBAAAACCEEDDD
BBBBBACDEEEDD
BBBBBBDDDDDDD
```

Each country will always be contiguous; in other words, you can always walk from any point in a country to any other just going horizontally or vertically. We will say that two countries share a border if at least one square of one country is horizontally or vertically adjacent to at least one square of the other. In the example above, A shares a border with B and C, and D shares a border with B and C and E, and C and E also share a border. A and D do not share a border, and neither do B and C, because they only touch diagonally.

A map is properly colored if each country is assigned
color, and no two countries which share a border hav
same color. (Otherwise, you couldn't tell apart where
starts and the other ends.) Of course, one can always color a map by giving each country its own color, but we want to reuse colors as much as possible, in the sense that the total number of distinct colors is minimized. For instance, in the example above, we could make A and D cardinal, B and C gold, and E green, minimizing the total number of colors to 3.

The famous Four-Color Theorem guarantees that there is always a way to color any map with just 4 colors. You are to write a program that actually **finds** such a coloring. Unless you were going to read thousands of pages of math proofs to understand how to do this differently, we strongly recommend using recursion and backtracking to solve this problem. While backtracking is slow, our guarantee that you have at most 15 countries (and only 4 colors) means that you only need to check at most about a billion cases, and backtracking will truncate this significantly. So it should run fast enough if you are careful.

The output should be a valid coloring that outputs the color assigned to each country, one per line. For the example coloring we gave for the input above, you would output (numbering the colors 1, 2, 3, 4):

```
A 1
B 2
C 2
D 1
E 3
```

Of course, there are many other colorings. You are welcome to output any one you like, or multiple solutions if you prefer.

In case you want more non-trivial test cases, here are a few. (If your solution does not use Backtracking, there is a decent chance it will fail on one or both of these inputs.)

Example 1

```
6 4 8
FFFFFFFF
FEEEDDDF
```

```
FAABBCCF
FFFFFFFF
```

Here is a valid output for this input:

```
A 1
B 2
C 3
D 1
E 3
F 4
```

Example 2 (Larger)

```
6 10 15
FFFFFFFFFFFFFFF
FFFFFFFFFFFFFFF
FFEEEDDDDDDDDFF
FFEEEDDDDDDDDFF
FFEEEDDDDDDDDFF
FFAABBBBBBCCCFF
FFAABBBBBBCCCFF
FFAABBBBBBCCCFF
FFFFFFFFFFFFFFF
FFFFFFFFFFFFFFF
```

Valid output:

```
A 1
B 2
C 3
D 1
E 3
F 4
```

Feel free to generate your own map files and post them on Piazza along with the possible valid color assignments.

Problem 3 (Heaps, 10%)

For the following pen-and-paper exercise, please solve them by hand and only use online visualization tools to verify your work.

For each of the following two arrays that are interpre
binary (2-ary) "min-heaps", state whether they are in
correct binary min-heaps or violate a property. If one
correct min-heap, then show what it looks like after you insert the key "2", and then show it again after you now remove the minimum (after having inserted "2"). You don't have to show the intermediate steps, but if you don't show them and the final answer is wrong, we will not be able to give you a lot of partial credit.

If one of the two heaps we gave you is not a correct binary min-heap, then point out everything that is wrong with it. In that case, you don't need to insert "2" or remove anything.

- [1, 4, 3, 7, 10, 6, 5, 9, 8, 11]
- [1, 6, 3, 9, 7, 4, 5, 8, 10]

Your answer should not be in array form like ours, but instead be drawn as a tree, either for the trees you want us to see, or by annotating the mistakes in the tree(s) we gave you. Your answer should be in a file called `hw6q3.pdf` or `hw6q3.jpg`. (If you need more space, you can give us multiple JPGs, but then you need to tell us in your README file where to find them and in what order to look at them.)

Problem 4 (Build a templated d-ary heap, 35%)

Build your own d-ary `MinHeap` class in `MinHeap.h` with the interface given below. You learned in class how to build a binary `MinHeap`, where each node had 2 children. For a d-ary `MinHeap`, each node will have d children.

```
template <class T>
class MinHeap {
   public:
     MinHeap (int d);
     /* Constructor that builds a d-ary Min Heap
        This should work for any d >= 2,
        but doesn't have to do anything for smaller d.*/

     ~MinHeap ();

     void add (T item, int priority);
       /* adds the item to the heap, with the given pric
```

```
        const T & peek () const;
          /* returns the element with smallest p
             Break ties however you wish.
             Throws an exception if the heap is empty. */

        void remove ();
          /* removes the element with smallest priority.
             Break ties however you wish.
             Throws an exception if the heap is empty. */

        bool isEmpty ();
          /* returns true iff there are no elements on the

    private:
       // whatever you need to naturally store things.
       // You may also add helper functions here.
  };
```

In order to build it, you may use internally the vector container (you are not required to do so). You should of course not use the STL `priority_queue` class or `make_heap`, `push`, `pop` algorithms.

In order to guide you to the right solution, think first about the following questions. We strongly recommend that you start your array indexing at 0 (that will make the following calculations easier). In order to figure out the answers, we suggest that you create some examples and find a pattern.

1. If you put a complete d-ary tree in an array, what is the index of the parent of the node at position i?
2. In the same scenario as above, what are the indices of the children of the node at position i?
3. What changes in the heap functions you learned in class when you move to d-ary arrays?

Submission Link

You can submit your homework here. Please make sure you have read and understood the submission instructions.

WAIT! You aren't done yet. Complete the last section below to ensure you've committed all your code.

Commit then Re-clone your Repository

Be sure to add, commit, and push your code in your
directory to your `hw-usc-username` repository. Now c
check what you've committed, by following the direc
below (failure to do so may result in point deductions):

1. Go to your home directory: `$ cd ~`
2. Create a `verify` directory: `$ mkdir verify`
3. Go into that directory: `$ cd verify`
4. Clone your hw-username repo: `$ git clone git@github.com:usc-csci104-fall2018/hw-username.git`
5. Go into your hw6 folder `$ cd hw-username/hw6`
6. Recompile and rerun your programs and tests to ensure that what you submitted works.

USC > UCLA Muahaha