Homework #5

Deadline 11:59pm Thursday 28 December 2013

*Please present a **printed** report with all your answers, explanations, and sample plots. Submit also a soft copy of the source code and binaries used to generate these results. Please note that copying of any results or source code will result in ZERO credit for the whole homework.*

Problem 1

You are given an image, and you are asked to return the *number* of connected components in the image, where a connected component is a set of pixels that are of the same color and can be reached from one another through pixels of the same color i.e. a *segment* or *cluster*. We will consider only 4-connectivity, that is each pixel can be reached to the pixels on top, to the right, to the left, and to the bottom of it. Mathematically, a pixel (i, j) is connected to pixels $(i–1, j)$, $(i+1, j)$, $(i, j–1)$, and $(i, j+1)$. The image implicitly represents a graph, where each pixel is connected to the four pixels on the *east*, *west*, *north*, and *south* of it.

1. How many vertices and how many edges are there in an image of dimension n x n?
2. Explain in pseudo-code how you would implement Union(x, y) and Find(x) if you are only allowed to use a matrix to store the set representative of each pixel (the cluster id).
3. Analyze the time complexity of both operations.
4. Suggest a way to reduce the time complexity of the Union operation with more storage using a heuristic like that in the lectures, and analyze the new amortized complexity and how it is improved.
5. Write a C++ program that, given an input image, will output the number of connected components in the image, as defined above. You should use the algorithm from part (4) above.

   The input is given on stdin as follows: one line contains the size of the input image n, followed by n lines of n numbers each. The output should be the number of connected components. For example, the input

   ```
   5
   
   32  32  21  21  10
   0   32  21  10  10
   0   0   32  9   10
   0   0   5   10  20
   5   5   5   10  20
   ```

   should have as output:

9

Problem 2

Explain a linear time algorithm that, given a directed acyclic graph $G = (V, E)$ and two vertices s and t, would return the number of paths from s to t in G. (You just need to count that paths and not list them).

Problem 3

You know that Dijkstra's algorithm can not be used to compute shortest paths in graphs with negative weight edges.

1. Prove this by finding a counter example i.e. a graph with negative weights (and no negative-weight cycles) for which Dijkstra won't find the correct shortest distances.
2. Explain where in the proof of correctness of Dijkstra the property that weights have to be positive is needed.

Problem 4

Let G = (V, E) be a graph with a real weight $w(u, v)$ assigned to each edge $(u, v) \in E$. Let $p=v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k$ be a path in G. In class we defined the weight of p as

$$w(p)=\sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

Although this definition of path weight is natural, in some applications alternative definitions may be more appropriate. In this problem you will see two alternative definitions of path weight. Your goal is to design efficient algorithms for the single-source shortest-paths problem under each of these definitions. That is, in each of the three cases below, give an efficient algorithm to find the shortest-path weights from a given source node s to all other nodes in G. The running time of each algorithm should be bounded by a polynomial in |V | and |E|.

Hint: create a new graph G' with weight function w', and show that running your algorithm on the new graph yields the desired result on the original graph G.

1. Let $G = (V, E)$ be a directed graph and $w: E \rightarrow R$ be a weight function. Assume that w takes only *positive* values. Define the weight of a path $p=v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k$ in G to be the *product* of all edge weights along the path, i.e.,
$$w(p)=\prod_{i=1}^{k-1} w(v_i, v_{i+1})$$
2. Consider a directed graph $G = (V, E)$ where weights are assigned to vertices rather than edges. Let $w: E \rightarrow R$ be such a weight function. Define the weight of a path $p=v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k$ in G to be the sum of all node weights along the path, i.e.,
$$w(p)=\sum_{i=1}^{k} w(v_i)$$

Problem 5

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 49 Indian rupees, 1 Indian rupee buys 2 Japanese yen, and 1 Japanese yen buys 0:0107 U.S. dollars. Then, by converting currencies, a trader can start with 1 U.S. dollar and buys 49 x 2 x 0.0107 = 1.0486 US dollar, thus turning a profit of 4.86 percent.

Suppose that we are given *n* currencies $c_1, c_2, \ldots c_n$ and an *n* x *n* table R of exchange rates, such that one unit of currency c_i buys R[*i*, *j*] units of currency c_j.

1. Give an efficient algorithm to determine whether or not there exists a sequence of currencies $c_{i1}, c_{i2}, \ldots, c_{ik}$ such that
$$R[i_{1,}i_2]\cdot R[i_{2,}i_3]\cdot\ldots\cdot R[i_{k-1},i_k]\cdot R[i_k,i_1]>1$$
and analyze the running time of your algorithm.
2. Implement your algorithm using C++. The input is also given on stdin, with the first line giving the number of currencies *n*, followed by *n* lines of *n* exchange rates each. The output should be either Yes or No and given on stdout. For example, the input

```
4
1.0     3.1       0.0023     0.35
0.21    1.0       0.00353    8.13
200     180.559   1.0        10.339
2.11    0.089     0.06111    1.0
```

should have as output:

```
Yes
```

Instructions

Please submit a soft copy of the solutions together with source code and binaries in one zip file. The file should be named as **CMP302.HW##.First.Last.zip** where HW## is the homework number e.g. HW01, First and Last are your first and last name. So, if your name is Mohamed Aly, and this is homework #1, the file should be named **CMP302.HW01.Mohamed.Aly.zip.** Failing to follow these instructions will cost you points.

Acknowledgment: Some problems are adapted from Erik Demaine.