University of Waterloo
CS341 - Winter 2016
Assignment 2

Due Date: Monday February 8 at 11:59pm

Problem 1 Cut Vertices and Cut Edges

Ans: The code is in the file a2q1.cpp.

Problem 2 Sorting by Reversals

Ans:
Model this problem as a graph problem:
For the given sequence of size n, we have $n!$ permutations. As we know, there will be $\frac{n(n-1)}{2}$ edges for each node.
We can build a graph with $n!$ vertices and since it is a $\frac{n(n-1)}{2} - regular$ graph, the number of edges of the graph is $n! \cdot \frac{n(n-1)}{2}$.
By doing BFS from the start and label every nodes with their location, we can find the label which located the sorted permutation.
So, the cost is $O(n! + n! \cdot \frac{n(n-1)}{2}) = O(n! \cdot n^2)$.

If the least swapping times is small, which is for transforming the permutations. Then for the location that the sorted permutation is located, the destination can reached from the start in k steps. Because there are $\frac{n(n-1)}{2}$ edges connecting the location to next locations for each node, we can get there are $(\frac{n(n-1)}{2})^k$ edges need to get destination permutation.
So, the cost for the small number of steps taken is $O(n^{2k})$.

So, the complexity of algorithm to find the shortest reversals, the minimum step need to convert, is $O(min\{n^{2k}, n! \cdot n^2\})$.

Shortest-Reversals
Given a permutation of size n, $(1, ..., n)$
Build a permutation graph with edges connecting to nodes if they could be transformed by reservals operations
1. do the BFS:
2. if the relatively large number of swapping times
3. go through all vertices and reversal all edges $(n! + n! \cdot \frac{n(n-1)}{2})$
4. else
5. reversal edges k times $((\frac{n(n-1)}{2})^k)$
6. choose which is the minimum steps to convert a permutation to destination permutation

Problem 3 Number of Shortest s-t Path

Ans:

Shortest-Path-Count

Given a undirected graph G and 2 vertices s,t

Output a number of different shortest paths between s and t

1. mark nodes as visited as usuall with BFS
2. add nodes plus number of incoming paths to the queue
3. if a node that has been visited
4. add a mark ig (ignore it)
5. if find a node which is currently in the queue (meet ig)
6. do not add it again, instead add the counts together.

(Because it has been encountered before, all paths that can follow it will be at least 1 longer than the previous shortest path. So none of these paths can contribute to the sum.)

7. accumulate the counts on the queue when adding new node
8. when encounter the final, return the counts

Because the algorithm just run by BFS, it should depends on the complexity of BFS. So the complexity of algorithm is $O(|V| + |E|)$.

Problem 4 One Way Streets

Ans:

As we know, there exists a way that any streets can go to any other streets, in other words, any street is reachable by any other street. So, all of vertices in this graph G are strongly connected each other.

So, given a vertex s, we can check whether it can get to all vertices v by doing DFS vertex by vertex and edges by edges in graph G costing $O(|V| + |E|)$. Then reverse the direction of all edges in G to get G^R. Then do the DFS again to check whether vertex s can is reachable for all other vertices v in graph G^R costing $O(|V| + |E|)$ as well. Finally, if both show *true*, we can get the design is possible and the direction; otherwise, the design is not possible.

As with DFS, we track the status of each node: new, visited but still open, visited and finished. In addition, we store the depth when we first reached a node, and the lowest such depth that is reachable from the node (after we finish a node, we will know this). A node is the root of a strongly connected component if the lowest reachable depth is equal to its own depth.

Initiate the DFS from any single node, and when we have finished, if the lowest reachable depth is 0, and every node was visited, then the whole graph is strongly connected.

Street-Design

Given a graph G and reverse it to get a graph G^R

Output true and direction; or false

1. make nodes as new
2. do DFS in G, make nodes as visited as usual with DFS
3. if finished and lowest reachable depth is 0
4. mark $bool1$ as $true$
5. if finished and lowest reachable depth is not 0
6. mark $bool1$ as $false$
7. do DFS in G^R, make nodes as visited as usual with DFS
8. if finished and lowest reachable depth is 0
9. mark $bool2$ as $true$
10. if finished and lowest reachable depth is not 0
11. mark $bool2$ as $false$
12. if $bool1$ and $bool2$ are both $true$
13. return $true$ and output the directions
14. else
15. return $false$

So, the total complexity is $2O(m+n) = O(m+n)$, where m is the number of edges and n is the number of vertices.