CS2020: Data Structures and Algorithms (Accelerated)

Problems 1–3

For Week10

1 Lecture 16/17 Material - Single Source Shortest Paths (SSSP)

1.1 Verifying Your Understanding of Bellman Ford's Algorithm

Drawing time :). Your DG leader will draw five **small random weighted directed graphs**:

1. One that has no negative weight edge at all, and connected
2. One that has no negative weight edge at all, and disconnected (say, it has 3 components)
3. One that has some negative weight edges, but no negative cycle, connected
4. One that has one or more negative cycles reachable from source, connected
5. One that has one or more negative cycles but those cycles are not reachable from source, i.e. the graph is disconnected

Then, your DG leader will ask five students, one graph per student, to perform **Bellman Ford's** algorithm from **source = vertex 0** in each graph. To make sure the order of relaxation is consistent, please use this edge-ordering criteria: the edges are ordered based on their first vertex number. For tie-breaking, order the edges based on their second vertex number.

Everyone should verify that after Bellman Ford's algorithm is executed, the correct result is obtained for each of the five graphs, i.e. the final $D[v] = \delta(v), \forall v \in V$.

PS: Please go through this exercise even if you feel that this is boring. Your DG leader is already instructed to spend only few minutes here if everyone already understand the basic concepts.

1.2 Verifying Your Understanding of Dijkstra's Algorithm

Using the same set of five random graph drawings from the previous section, run **Dijkstra's** algorithm on them. During lecture, Steven has shown that there are two ways to implement Dijkstra's algorithm, the standard/original way (the one that forbids re-tweaking of $D[v]$ once it has been confirmed) and the modified way (the one that avoid using `Heap_DecreaseKey` and thus forced to sometime re-enqueue an unprocessed vertex already in the priority queue but with better $D[v]$). Try both approaches and discuss the phenomenon on various type of graphs with your DG. To help making the discussion consistent, use the same source = vertex 0 and the edge-ordering criteria as above.

PS: As this new/modified technique is relatively new, please spend more time discussing the differences.

1.3 More SSSP Properties

1.3.1 Subpaths of shortest paths are shortest paths

Dijkstra's algorithm shown in Lecture17 relies on the property that a shortest path between two vertices contains other shortest paths within it. This allows Dijkstra's algorithm to do this: Suppose it already knows the shortest path from $source = a \sim\!\!\rightarrow b$. If edge (b, c) is the shortest path from b to c, then the path $source = a \sim\!\!\rightarrow b \rightarrow c$ is the shortest path from a to c.

Your task here is to prove this theorem: subpaths of a shortest path are shortest paths (too)!
Let p be the shortest path: $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$.
Let p_{ij} be the subpath of p: $p_{ij} = \langle v_i, v_{i+1}, v_{i+2}, \ldots, v_j \rangle, 0 \leq i \leq j \leq k$.
Then p_{ij} is a shortest path (from i to j)

1.3.2 Can a shortest path contain a *non-negative* cycle?

We have seen in Lecture16/17 that when the given graph contains negative cycle (reachable from the source vertex), then the shortest paths problem becomes ill-defined.

How about *non-negative* cycle? Can a shortest path contains a positive-weight cycle? Can a shortest path contain a 0-weight cycle? Prove it!

After that, go back to the proof of correctness of Bellman Ford's. Verify again that the shortest path from source to the furthest reachable vertex in the graph will at most contains $V - 1$ edges.

2 Quiz 2 Preparation (Steven's style): Graph Modeling + Choosing the *Best* Graph Algorithm

DG leaders will ask students in his group to read these University of Valladolid (UVa) online judge problems[1]. Look at Recitation of Week09 for four samples (note: that recitation shows four examples for SSSP only, but this time, the graph problem is not limited to SSSP). In case the graph in the given problem is *implicit*, look at Recitation of Week08 for some more details. Steven guarantees that these problems can be solved with the knowledge from Lecture13-14-15-16-17 (for SSSP problem during Quiz 2, it is OK if you always use the $O(VE)$ Bellman Ford's solution answer as it works in general, but may not be the fastest).

1. UVa 10653 - Bombs! NO they are Mines!!
   http://uva.onlinejudge.org/external/106/10653.html
2. UVa 11747 - Heavy Cycle Edges
   http://uva.onlinejudge.org/external/117/11747.html
3. UVa 10336 - Rank the Languages
   http://uva.onlinejudge.org/external/103/10336.html

After reading these problems, three students will answer these questions.

1. What is the graph (the vertices, edges, weighted/unweighted, directed/undirected, etc)?
2. What is the graph problem?
3. What should be the best graph data structure and algorithm to solve these problems? State their space/time complexity! Argue why you choose those DS/algorithm.
4. Try to sketch a *pseudo code* that implements the chosen data structure and algorithm. If possible, write a working Java code during the DG :).

[1]We put this skill into practice in CS3233. Given interesting programming problems that can be modeled as graph problem (not just SSSP), solve it as fast as you can. Consider taking CS3233 in S2 AY2011/2012 if you like these kind of problems :)

3 Kingdoms At War (100 marks)

Long long time ago, there was a very fierce war between two kingdoms: A and B.

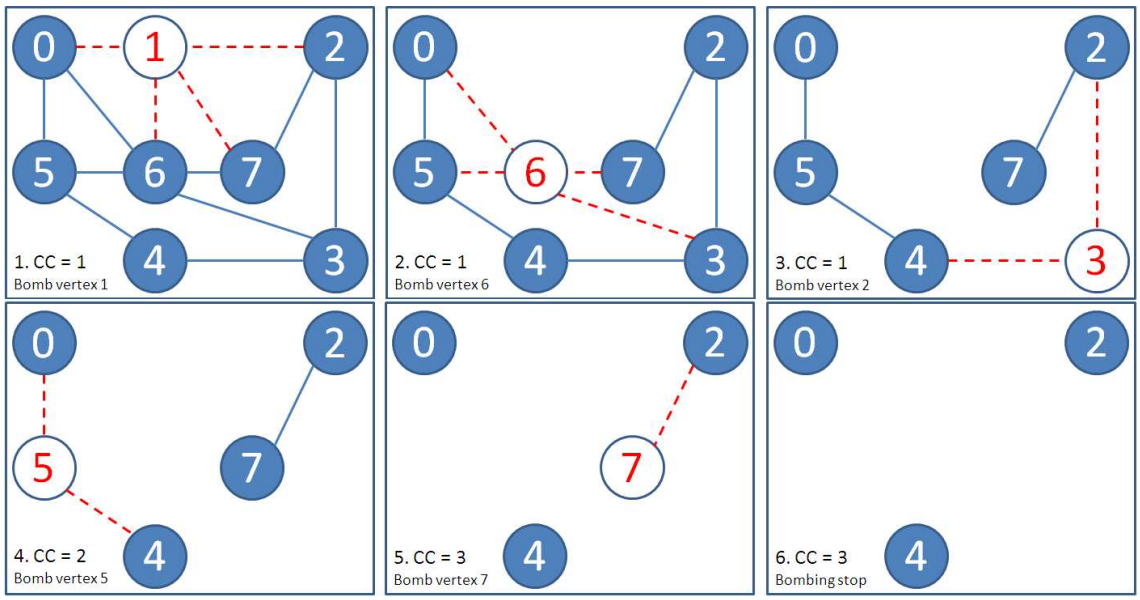
Kingdom A consisted of N *connected* cities. There were M bi-directional roads between these N cities ($1 \leq N, M \leq 200000$) – yes, it was **a rather large** kingdom...

In that war, the enemy soldiers from kingdom B used a special bomb, which can totally destroy a whole city that it is thrown into. If a city was destroyed, no one could pass through it anymore. History recorded that kingdom B soldiers thrown K special bombs into K distinct cities of kingdom A ($1 \leq K \leq 200000$) – yes, that was **a lot of** bombs...

You are a historian and you want to analyze how many *connected components* (CC) were there in kingdom A at the beginning (before the war was started) and after each of the K bombs was thrown into a certain city by kingdom B soldiers[2]. *Your program must run in under 1 minute.*

See below for a sample scenario where initially kingdom A had $N = 8$ cities, $M = 13$ bi-directional roads, and connected (CC = 1). Then kingdom B soldiers thrown 5 special bombs...

1. Kingdom B soldiers thrown a special bomb to city 1.
2. After city 1 was destroyed, no one could pass through it anymore, but CC was still 1.
   Then kingdom B soldiers thrown a special bomb to city 6.
3. After city 6 was destroyed, no one could pass through it anymore, but CC was still 1 (a tree!).
   Then kingdom B soldiers thrown a special bomb to city 3.
4. After city 3 was destroyed, there were two CCs: (0-5-4) and (7-2).
   Then kingdom B soldiers thrown a special bomb to city 5.
5. After city 5 was destroyed, there were three CCs: (0), (4), and (7-2).
   Then kingdom B soldiers thrown a special bomb to city 7 (the last one).
6. After city 7 was destroyed, there were still three CCs: (0), (4), and (2).



[2]This problem was originally created by Acer Wei Jing for CS3233 AY 2009/10. This problem was originally planned for Quiz 2, but it is moved to DG7 instead because it is found not suitable for CS2020 level yet :O. If you found this problem to be easy, please join CS3233 next AY :).

3.1 Understanding the Problem (20 marks)

Let's reuse the same graph as the initial graph (Figure 1, with $N = 8$, $M = 13$, and $CC = 1$).

If K and the sequence of bombings were different, what was the number of connected components after each bombing?

Each test case worth 5 marks, the first one is the sample above and thus not counted.

1. Sample: $K = 5$, sequence = $[1, 6, 3, 5, 7]$
   Output $K = 5$ numbers, the number of CC after each bombing = 1, 1, 2, 3, 3
2. Test case 1: $K = 6$, sequence = $[0, 1, 2, 3, 4, 5]$
   Your output for test case 1 =
3. Test case 2: $K = 4$, sequence = $[6, 7, 1, 2]$
   Your output for test case 2 =
4. Test case 3: $K = 3$, sequence = $[7, 4, 2]$
   Your output for test case 3 =
5. Test case 4: $K = 5$, sequence = $[6, 7, 0, 2, 4]$
   Your output for test case 4 =

3.2 Further Understanding of the Problem (10 marks)

Assume that your computer can process 1000000 (1 million) operations in 1 second. Thus, within 1 minute of allocated time, you can do up to 60000000 (60 million) operations. Can you solve this problem by calling $O(N + M)$ DFS to count the number of connected components K times?
Circle one answer (5 marks): (YES / NO).
This is because (5 marks) = ----------.

3.3 Outlining a Solution Sketch (10 marks)

Write down – in few sentences – the graph problem that you are asked to solve here (5 marks). Then, give a rough outline and time complexity of your solution (5 marks). For this rough outline, simply use pseudo-code and algorithm name(s) without stating the algorithm details.

```
The graph problem is:

The rough outline of the solution and its time complexity is:
```

3.4 Writing Your Java Solution (60 marks)

Suppose the input graph has been stored in an `AdjList` as shown in lecture. As the graph is unweighted, you can ignore the second value in integer pair `ii` as defined in lecture (`ii.java`). The sequence of bombings has been stored in a Vector `Seq`. Write a Java function: `private static Vector < Integer > CC(Vector < Vector < ii > > AdjList, Vector < Integer > Seq)` that returns a `Vector < Integer >` of the number of connected components after each bombing.

4 Discussion on Current Problem Set (PS8)

If you still have time, you can spend some time during DG7 to discuss the solutions for problems in PS8 (the short one). Note that you are allowed to discuss! You are only prohibited from coding the solution together/copying the solution. Write down the list of collaborators in your solution.