IT5003 Mar-May 2023

Data Structures and Algorithms

Tutorial+Lab 08

Finale

Document is last modified on: November 21, 2023

1 Introduction and Objective

In this last session, we will wrap up the semester. First, we will spend some more time reviewing any loose ends about the SSSP problem.

Then, we will discuss a few past paper questions involving this SSSP problem.

Finally, we will use the remaining time to discuss the ongoing PS8 tasks.

2 Questions

SSSP Open QnA, Continued

We need to clear two more SSSP-related algorithms that are covered during Lecture 8a (the Modified Dijkstra's algorithm and the topological sort solution if the input graph is a DAG). Please use `https://visualgo.net/en/sssp` during the discussion.

For this session, focus on:

1. In IT5003, we only covered the modified implementation of Dijkstra's algorithm (implemented purely using Python heapq, but in a 'lazy fashion'). For this course, it is sufficient to say that if the edge weights are non-negative, then modified Dijkstra's runs in $O((V + E) \log V)$, on par with the original version of Dijkstra's algorithm (that we cannot easily implement in Python due to the lack of built-in balanced BST data structure). TA will go through with students if anyone is still unclear with this famous CS algorithm.

2. If the given directed weighted graph is a DAG, we can regain the $O(V + E)$ time complexity by relaxing the edges not via 'Breadth-First' manner with BFS (WA) or 'greedy' manner with modified Dijkstra's ($O(\log V)$ slower), but by the topological order manner (remember, we can only find a topological ordering in a DAG). So we first find the topological order (any valid

order is fine) of the DAG using postorder DFS or BFS modification (Kahn's algorithm), and then relax the outgoing edges of each vertex of the DAG according to the topological order. This is $O(V + E)$ overall.

Last Graph Modeling Exercise, via Past Paper Discussions

There are a few graph questions in recent related final assessment papers (of CS2040/C/S or IT5003). Let's discuss three of them (considering that SSSP will have to be asked in the Final Assessment):

1. `https://www.comp.nus.edu.sg/~stevenha/cs2040/tests/IT5003-2021-22-S1-final.pdf`, Application 2 - Book Translation

   For Q14, we can use AL (using defaultdict) to help us map string (vertex = language) to list of neighboring languages and associated translation cost.

   For Q15, A=15, B="Impossible",C=24

   This is an unweighted SSSP problem with a small variation. We do BFS from "English", keep the BFS SSSP spanning tree. But if there is a tie (two different paths reaches vertex v with the same number of hops), we keep the one with lower edge weight (cost). At the end, if any of the n target languages is not reachable, we print "Impossible", otherwise, we output the sum of the weights in the BFS SSSP spanning tree constructed as above.

   Note that using Dijkstra's by sum of costs is WA, as we want to minimize the number of translation steps first, cost is secondary.

   BFS prioritizing just number of translation steps only can have wrong cost

   Using Dijkstra's that uses unweighted edge (like BFS) and if tie then use the cost is 'correct' but has an extra log factor

2. `https://www.comp.nus.edu.sg/~stevenha/cs2040/tests/CS2040-2017-18-S4-final.pdf`, Question C.1, SSSP in a Special "SLL".

   The solution is like this:
   We don't actually need the Modified Dijkstra's algorithm to solve this problem...
   We also don't actually need toposort based algorithm although this "SLL" is a DAG...

```
n, K = map(int, input().split())
print(min(n-1, S+2+(n-1)%S)) # look ma, no need to run Dijsktra's...
# n-1 for going direct using normal edges, or
```

```
# S+2+(n-1)%S for going from vertex 0 to vertex S (S steps,
# then 1 jump to near vertex n-1 (weight 2),
# then walk (n-1)%S more steps to vertex n-1
```

3. `https://www.comp.nus.edu.sg/~stevenha/cs2040/tests/CS2010-2015-16-S4-final.pdf`, Question C.3, Avengers Initiative

   For Avengers Initiative

   A quick idea is to use All-Pairs Shortest Paths (APSP) solution: The $O(V^3)$ Floyd-Warshall algorithm (not discussed). This way, we can know what is the shortest path from each of the A Avenger positions to each of the H Helicarrier positions in $O(1)$ per query. We can then test who has the slowest (that is, largest) shortest path time T among the A Avengers to reach each of the H Helicarrier positions even after each Avenger use the shortest paths (doable in $O(A)$ per Helicarrier position so $O(HA)$ overall). Avenger that cannot reach a certain Helicarrier will cause T = 'Infinity' for that Helicarrier. Then, we report the position of the Helicarrier that has the smallest T value and report T as the shortest possible meeting time in $O(H)$, but this is negligible as we already have $O(HA)$, and $O(HA)$ is smaller than $O(V^3)$ so it is also not considered in the overall time complexity. If the overall T is still 'Infinity', we have to unfortunately report 'Doomsday'... The overall time complexity is $O(V^3 + HA) = O(V^3)$ but this has problem as $V \leq 100\,000$ in this problem.

   Another idea is to do A calls of SSSP algorithm: Dijkstra's (either version, but let's use Modified Dijkstra's in IT5003) to compute the same APSP information above. The time complexity is still slow $O(A * (V + E) * \log V + HA) = O(A * (V + E) * \log V$. The problem is that $A \leq 10\,000$ and $V + E \leq 300\,000$...

   A better idea is to realize that H is small (up to 10 only). We can therefore transpose the entire world map G in $O(V + E)$, then run Dijkstra's (either version, but let's use Modified Dijkstra's in IT5003) from each of the H Helicarrier positions (this is $O(H * (V + E) * \log V)$). After we do this, we then also have the shortest paths from each of the H Helicarrier positions to each of the A Avenger positions. The rest is the same as above. The overall time complexity is $O((V + E) + H * (V + E) * \log V + HA) = O(H * (V + E) * \log V)$, AC.

Hands-on 8

TA will run most of this session to help students with PS8.

- PS7 Quick Debrief,
- Then, use the remaining time to help students with PS8... (in high level only), see below

Do PS7 quick debrief according to the context of your lab group and concentrate on helping students with PS8 (up to a certain limit of hints) - but for students in your lab group that has completed

PS8, ask them to try `https://nus.kattis.com/problems/texassummers`, another easy SSSP problem as their last hands-on problem.

Problem Set 8

We will end the tutorial and this module with high level discussion of PS8 A+B.

For PS8 A (/bryr), you first need to fully understand the meaning of the problem. Single-lane bridge has $c = 1$ (cost is 1) and Stefan doesn't like this. Double-lane bridge has $c = 0$ (cost is 0/free), Stefan is ok. So Stefan prefers taking 0-weighted edges (double-lane bridges) as most as he can and only resorted to 1-weighted edge (single-lane bridges) only if necessary. This can either be modeled as a weighted graph and let Dijkstra's prioritizes 0-weighted edges, or if you know about this BFS variant, to use BFS+deque to solve this 0/1-weighted SSSP variant.

PS8 B (/hopscotch50) was used for final assessment last year (December 2022). This problem can be modeled as a weighted SSSP problem. There are multiple sourceS (all cells with value 1). You can go to a cell x to any other cell y if the value of y is $x + 1$. The cost of that movement is the Manhattan distance between x and y. A standard Dijkstra's algorithm can solve this problem as the constraints are small. All the best.