Thursday, 26 October 2017

LAB DEMO 08

PS4 Debrief – Common Mistakes

Typical common mistakes in PS4:

- WA in ABC: Must be just minor mistake(s) as there is nothing fancy once you know that this is a ‘**MiniMax path problem**’ with classic solution:
  - Build the MST (not just MST weight) of the original graph with either Prim’s/Kruskal’s and then
  - Run DFS/BFS from the source vertex to target vertex on the MST (yeah, a tree)
  - It can be proven that taking the maximum edge along the only path in the MST (a tree) is the answer for this MiniMax path problem
- TLE in D: See the next slide

PS4 Debrief – Our Answer (1)

The ultimate? solution for PS4 Subtask D:

- Pre-compute all possible queries from **source vertex 0-9** to all other vertices by running DFS (or BFS) 10 times from each source vertex 0-9 after you obtained the MST
  - That is, the new technique that we want to share is "Pre-computation"
- Store the query results in a 2D array of answers of size 10*V
  - Using hash table or map (bBST) is possible but pointless
- This is an important trade-off:
  Trading memory (2D array of answers) for O(1) query speed ☺
  - This important key concept will be revisited in Dynamic Programming!

PS4 Debrief – Our Answer (2)

The ultimate? solution for PS4 Subtask D (continued):

- Minor enhancements (will not matter much):
  - Stop Prim's/Kruskal's as soon as the MST is found
    - I have a few small complete graphs in the test cases
  - Only run DFS/BFS from source s when asked… not always from [0..9]
    - On some small test cases V is < 10
    - On some test cases, I only ask from source s = 0
  - Kruskal's user only → The effort rating has a "small range" of integers [0 .. 1000] and this can be exploited by Kruskal's user: We can use **<u>Counting Sort</u>** to sort the edges so that Kruskal's runs in O(E) instead of O(E log V)
    - Learn Counting Sort (CS3230 syllabus) on your own
  - Change "Integer" to primitive "int" in IntegerPair/IntegerTriple class

Revisit APIO 2013 TASKSAUTHOR

Let’s concentrate on **Subtask 6** of APIO13_finalversion.pdf this week since you now know Dijkstra’s algorithm for solving SSSP

S6: “Kill” the Modified Dijkstra’s

```
// pre-condition: the graph is stored in an adjacency list L
counter = 0;
for each query p(s,t)
  dist[s] = 0; pq.push(pair(0, s)); // pq is a priority queue
  while pq is not empty
    increase counter by 1;
    (d, u) = the top element of pq and then remove it from pq;
    if (d == dist[u]) // that important check
      for each edge (u,v) in L
        if (dist[u] + weight(u,v)) < dist[v]
          dist[v] = dist[u] + weight(u,v);
          insert pair (dist[v], v) into the pq; // lazy
  output dist[t];
```

Challenge: Construct a Graph without –ve cycle that can make this Modified Dijkstra’s runs extremely slowly…

Think First ☺ & Discuss

Note: You have actually seen the picture in Lecture09…

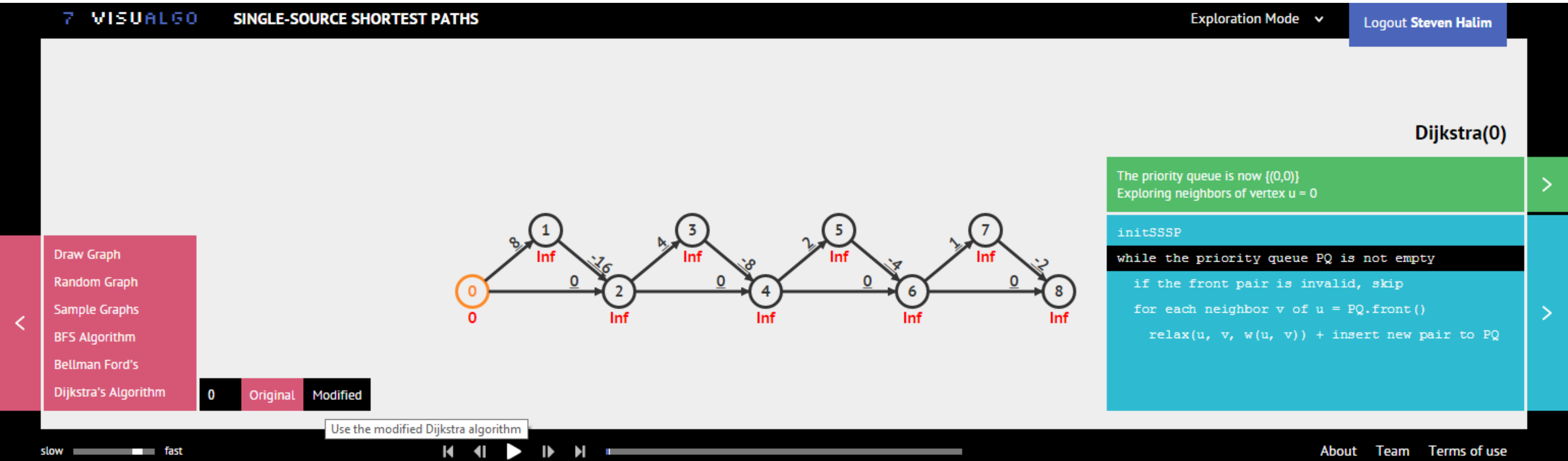
Wordy Solution
(sample drawing in the next slide)

- Those who are familiar with the Modified Dijkstra’s should immediately realize that the solution has to be related with the fact that Modified Dijkstra’s **re-enqueue and re-process** new vertex information pair.
- On a graph with no negative weight edge, such action is rare (but exists, as we employ Lazy Deletion)
- On a graph with negative weight but no negative weight cycle, one can construct a ‘dual path’ graph as shown in the next slide so that Modified Dijkstra’s reprocess many vertices many times (exponentially), whereas the Optimized Bellman Ford’s simply runs in O(VE).

Not an all-conquering algorithm…

What is the time complexity of this test case?

- Answer: # of paths from 0 to last vertex, it is 16 for this DAG
  - You will learn an algorithm to count # of paths in a DAG in Lecture10
- There are 9/2 = 4 "triangles" and 2^4 = 16 paths (exponential)…
  - This is a bad news for modified Dijkstra's ☹, but this case is rare ☺☺

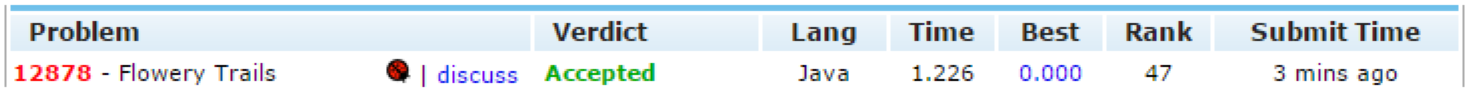


The priority queue is now {(0,0)}
Exploring neighbors of vertex u = 0

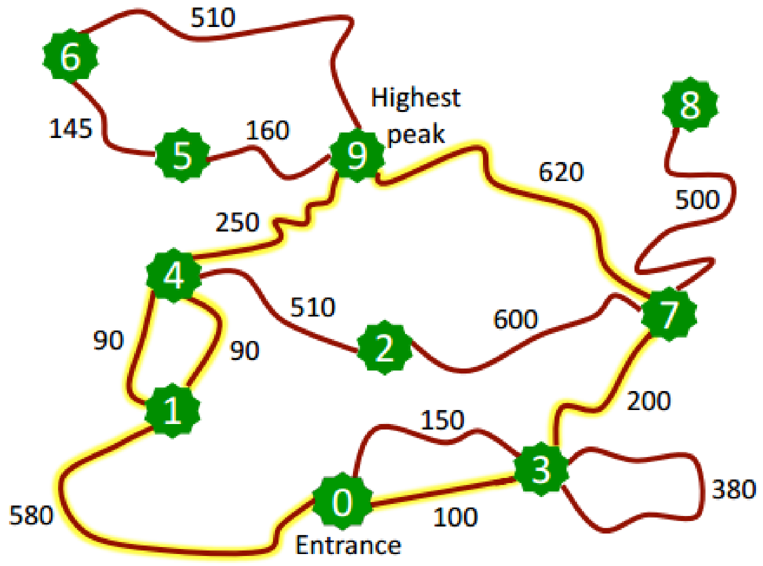
```
initSSSP
while the priority queue PQ is not empty
  if the front pair is invalid, skip
  for each neighbor v of u = PQ.front()
    relax(u, v, w(u, v)) + insert new pair to PQ
```

Recap Flower Trails Problem from last week (1)

We have modelled UVa 12878 as an SSSP problem last week – this week we will show the code for solving it by "tweaking" Modified Dijkstra's.



Problem	Verdict	Lang	Time	Best	Rank	Submit Time
12878 - Flowery Trails | discuss	Accepted	Java	1.226	0.000	47	3 mins ago



Recap Flower Trails Problem from last week (2)

- Points of interest – **Vertices**
- Trails connecting points of interest – **Edges**
- Non-simple graph – **can have more than 1 trail connecting two points of interest**
- SSSP problem with a twist – **want to know edges involved in all SPs from entrance to highest peak**

PS5 Overview

PS5 is already out since Saturday, 15 Oct 2016, 12noon

- It is about Single-Source Shortest Paths++

PS5 Status (as of 22 Oct)

Subtasks A+B are not that hard actually…

# of students	A	B	C
2	AC	AC	AC
3	AC	AC	
3	AC		
The rest?			

Yes, Subtask C is a bit insane …

PS5 Subtask A+B

Big Question: Does parameter **k** matters?

PS5 Subtask A+B Summary

Nothing to hide here (as it will be too late if we hide it until next Lab Demo 09), parameter **k** does not matter for these two subtasks…

PS5 Subtask A and B are there to force you to code **<u>at least one</u>** SSSP algorithm:

- Subtask A → the graph is a tree, **V** < 1K, **Q** ≤ 100K therefore the best algorithm is _____________
- Subtask B → the graph is weighted, **V+E** < 201K, **Q** ≤ 10K therefore the best algorithm is _____________

But how to deal with that big **Q**?

Dijkstra’s is a standard algorithm, but to avoid plagiarism check…

- Choice of Dijkstra’s version
  - Original or Modified **(2 options)**
- Choice of Priority Queue implementation for Dijkstra’s
  - TreeSet, TreeMap :O, PriorityQueue (Lazy DS), your own PS2 code (AVL Tree), your own PS1 code (Binary Heap), or even Fibonacci Heap (or other exotic Heap DS)… **(6 options)**
- Or 1 other alternatives outside Dijkstra’s **(+1 option)**
- Technically, we have **2*6+1 = 13 combinations** ☺

PS5 Subtask C

Why it is hard(er)?

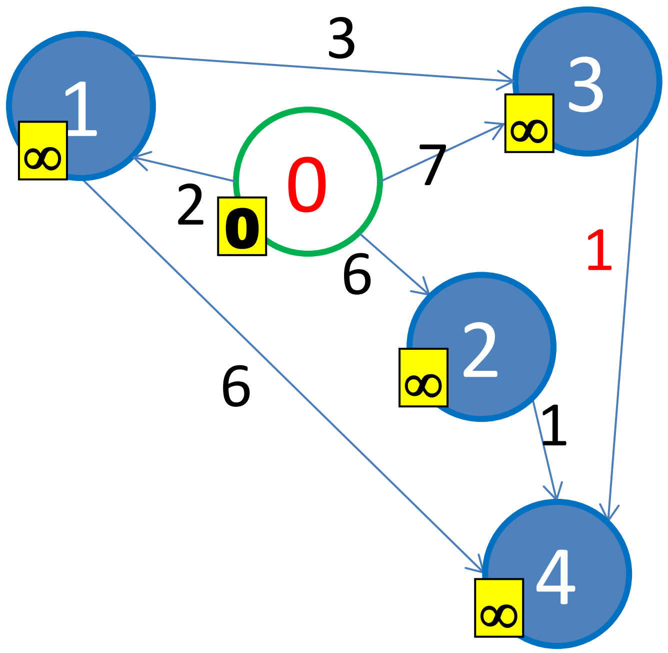
The additional constraint is quite relevant in real life (other than the requirement due to ket fah's condition)

- Google around for recent (taxi) accidents videos in SG
  - Many happened around junctions
- Note that some of those accidents are fatal so please watch those videos with caution

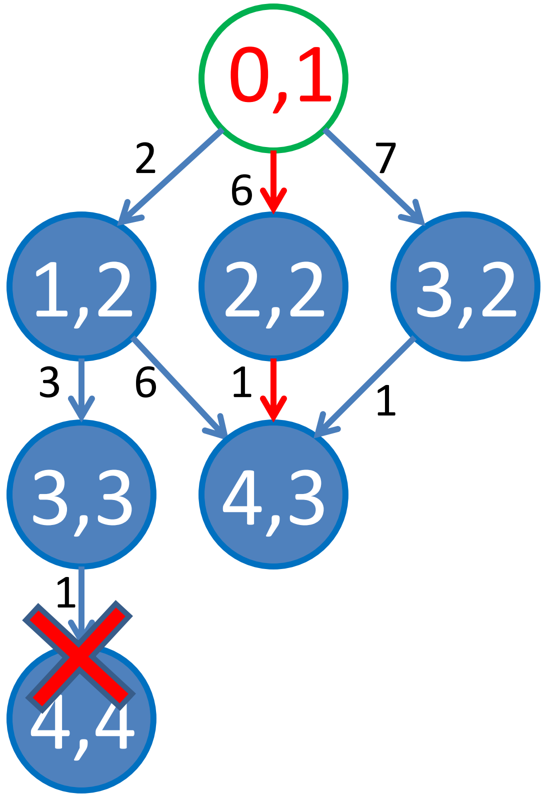
What to do in order to handle this seemingly simple additional constraint that the shortest path cannot have more than **k** vertices on it?

- Hint: Related to what your tutor said in tut08…

Restrict SP to 3 vertices only



The transformed graph is a DAG!



Note that if I do not restrict the number of vertices in the SP, the answer is 0→1→3→4 of weight 2+3+1 = 6 instead of 7 from path 0→2→4

Is that a DP Problem?

Yes, we can classify PS5 Subtask C as a DP problem

This is because the transformed graph is actually a DAG (more details in the 3rd part of CS2010 ☺)

In fact, after you finish Lecture 10+11, you may want to re-do this problem with DP technique instead of using Dijkstra's algorithm

– But it is 'slower' due to the usage of recursive calls…

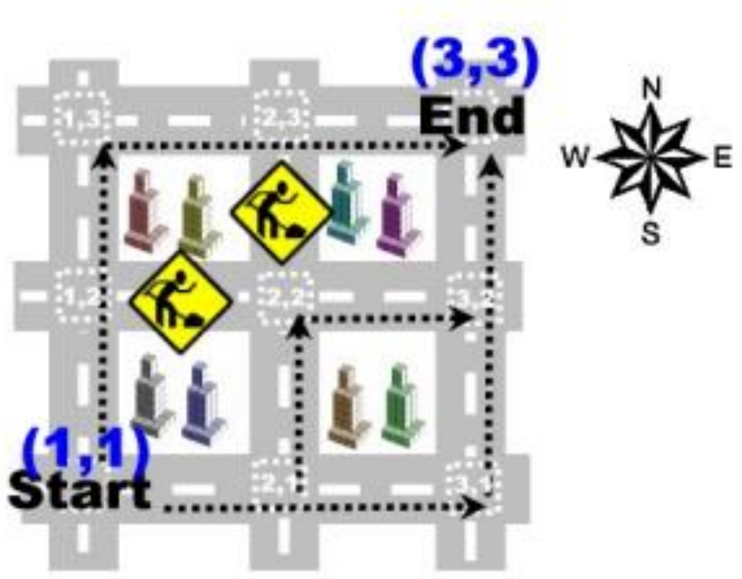
Another Demo for Today

Discussion of one classic algorithm on DAG

UVa 926 - Walking Around Wisely

https://uva.onlinejudge.org/external/9/p926.pdf

This is part of CS2010 final exam question in the past 4 years (the final exam question is the simplified version)



Online Quiz 2 next week

- Please remember to come to lab next week for online quiz 2
- Content covered will be from lecture 6 to 9.
- Format is same as OQ1
  - 20 questions
  - 35 mins
  - Difficulty level -> Hard
- Train hard on Graph Traversal, Minimum Spanning Tree and SS Shortest Path modules on Visualgo
  - Click on the “CS2010 Quiz 2” link and train train train !