CS2010 Semester 1 2012/2013

Data Structures and Algorithms II

Tutorial 07 - Shortest Path 2

For Week 09 (15 October - 19 October 2012)

Released: Wednesday, October 10, 2012

Document is last modified on: October 17, 2012

1 Introduction and Objective

This is the last tutorial for the second part of CS2010 (Graph). After this, we will switch to the last topic of CS2010 (DP).

We will continue discussing the SSSP problem, especially the graph modeling aspects. We will revisit Dijkstra's algorithm, both the original and the modified implementation variant. Finally, we will discuss PS5 Subtask 1-2-3-4 that highlight special cases that can be solved more efficiently using another algorithm than the generic one.

Note: Use `http://www.comp.nus.edu.sg/~stevenha/visualization/sssp.html` to *verify* the answers of some questions in this tutorial. However during written tests, you have to be able to do this by yourself.

2 Tutorial 07 Questions

Shortest Path application

Q1. What is the best algorithm to detect "the issue" in the following SSSP problem given the directed graph below and a source (highlighted as a black node)?

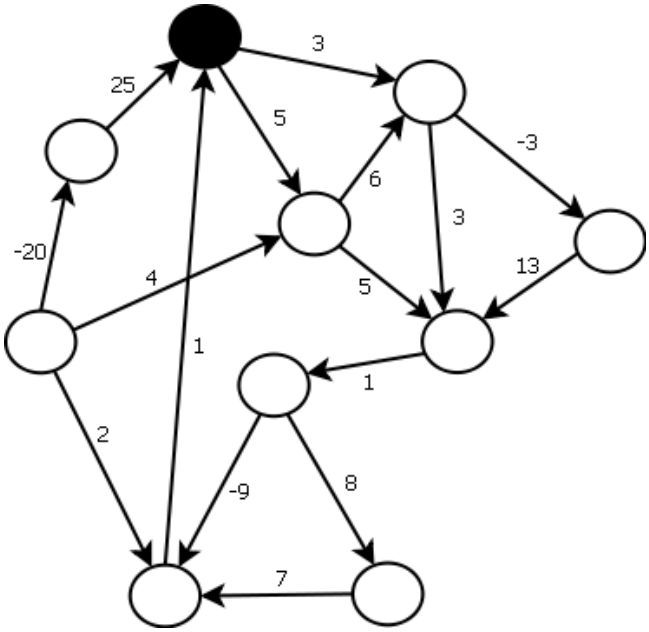


Figure 1:

1. Bellman Ford's
2. Original Dijkstra's
3. Modified Dijkstra's
4. BFS
5. DFS
6. None of the above

Ans: 1.

There is a negative weight cycle in the graph as shown below, $O(VE)$ Bellman Ford's is the tightest algorithm among the options above that can detect this negative weight cycle. (PS: We haven't see Floyd Warhsall's yet, but that one is also $O(V^3)$, which is slightly bigger than $O(VE)$ as in most graphs $E < V^2$.)

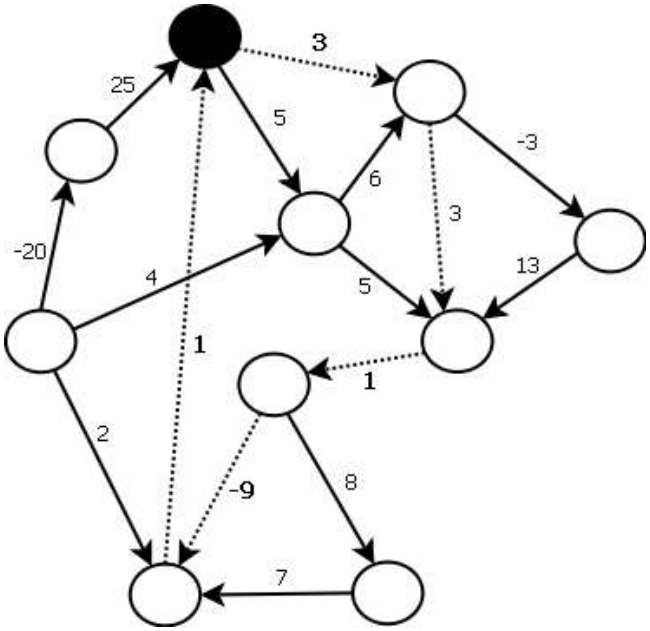


Figure 2:

Q2. Run the modified Dijkstra's algorithm (as shown in Lecture 08) on the graph below using vertex A as the source. Show the PQ in each step of the algorithm and also give the final shortest path spanning tree.

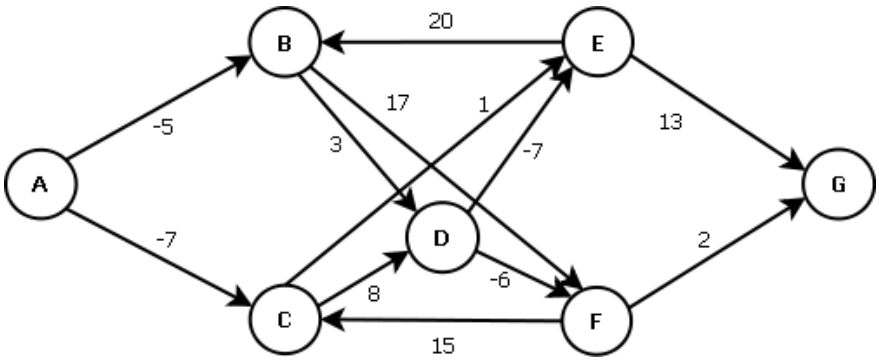


Figure 3:

Ans: (**bold and underline** - the inferior vertex information in PQ that are not immediately deleted due to 'lazy deletion' strategy)

PQ = {(0,A)}
PQ = {(-7,C),(-5,B)}
PQ = {(-6,E),(-5,B),(1,D)}
PQ = {(-5,B),(1,D),(7,G)}
PQ = {(-2,D),**(1,D)**,(7,G),(12,F)}
PQ = {(-9,E),(-8,F),**(1,D)**,(7,G),**(12,F)**}
PQ = {(-8,F),**(1,D)**,(4,G),**(7,G)**,**(12,F)**}
PQ = {(-6,G),**(1,D)**,**(4,G)**,**(7,G)**,**(12,F)**}
PQ = {**(1,D)**,**(4,G)**,**(7,G)**,**(12,F)**} all these are inferior vertex information that will now be deleted

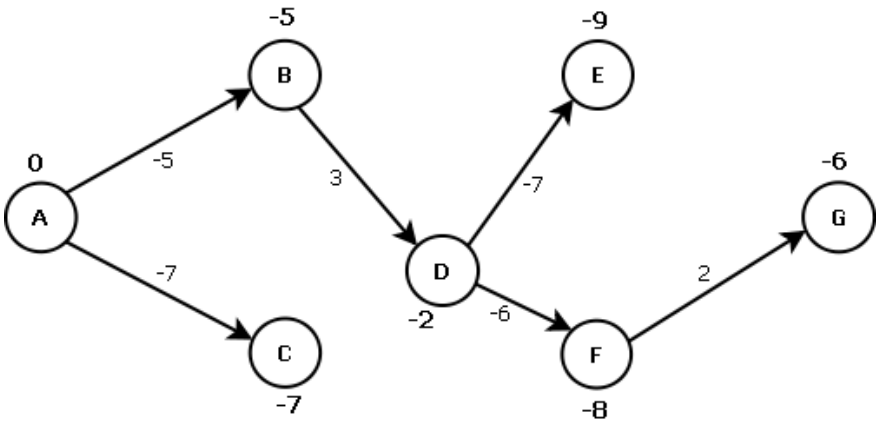
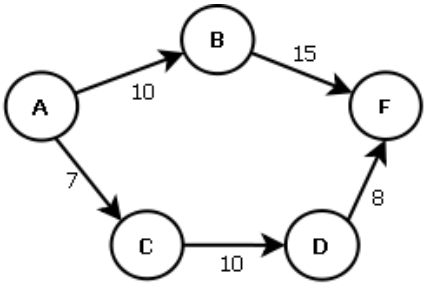


Figure 4: The SSSP Spanning Tree

Q3. A salesman frequently needs to drive from one city to another to promote his products. Since time is of the essence, he will want the shortest route to get from one city to another. However in every city he passes he will have to pay a **toll fee** (assuming toll fee is the same for every city). Therefore, given two different routes of the same distance to get from city A to city B, he will prefer the one which passes through fewer cities. An example is shown



To get from A to E, route A,B,E is preferred over route A,C,D,E even though both have the same cost 25, since A,B,E goes through fewer cities.

Figure 5:

Propose the best algorithm using what you have learnt so far, so that the salesman will choose a route from any source city A to any destination city B such that it has the shortest distance and also passes through the fewest cities. What is the running time for your algorithm?

Ans:

Modify the priority queue of Dijkstra's algorithm a bit, from storing pairs $(d[v], v)$ to storing triplets $(d[v], h[v], v)$, where $h[v]$ stores how many hops/edges have been used in the shortest path so far.

On the sample graph above, the execution is like this:
PQ = {(0,0,A)}
PQ = {(7,1,C), (10,1,B)}
PQ = {(10,1,B), (17,2,D)}
PQ = {(17,2,D), (25,2,E)}
PQ = {(25,2,E), (25,3,E)} // (25,3,E) will be considered as "inferior vertex information triple" and ignored with code like this

if ($d == D[u]$ && $h == H[u]$)
// then process this vertex information triple

Then, there is one more test to check when relaxation results in the same min distance estimate. If it is the same, we will check whether the min Hop estimate can be decreased decreased.

```
if (D[v] == D[u]+w(u,v))
  if (H[v] > H[u]+1)
    H[v] = H[u]+1;
    PQ.enqueue(D[v],H[v],v);
```

For the case where min distance estimate is reduced. We will update H without checking.

Running time = same as original/modified Dijkstra's, $O((V + E) \log V)$, easier analysis with original Dijkstra's as there is only one vertex information triple per vertex.

Q4. A travel agent has access to a database which lists for all airports around the world, their flight schedule. The flight schedule consists of the flight plans of all airplanes coming into and leaving a given airport. Specifically, the flight plan for each airplane is as follows

1. plane number
2. destination airport
3. departure time from airport
4. arrival time at destination airport

Help the travel agent develop a software which will determine a flight route for her clients, such that given a source airport **SA** and starting time **ST** (departure time of a flight must be >= ST), it will produce the flight route giving the earliest arrival time at the destination airport **DA**.

*you can assume that smallest unit of time is in minutes.

Ans:

Model the problem as a directed graph as follows:
Nodes:-
A normal node represents the flight plan of an airplane at an airport. It contains the following data

1. Airport **sa** (airport plane is in)
2. plane number **pn**
3. destination airport **da**
4. departure time from airport **dt**
5. arrival time at destination airport **at**

There are 2 more special nodes, the source node **sn** and the destination node **dn**.

Edges:-

Weight on the edges represents time in minutes.

Edges are computed as follows:

1. There is a directed edge (sn, v) from sn to every normal node v, where $v.sa = SA$ and $v.dt >= ST$ (valid starting flight). weight$(sn, v) = v.dt - ST$ (Time to wait for the first flight)

2. There is a directed edge (u, v) from normal node u to v as follows:

Algorithm 1 edge-generation

```
if u.da = v.sa then                                          ▷ plane flying to correct airport
    if v.sa != DA then                                       ▷ not destination airport
        if u.at < v.dt then                                  ▷ next flight must be catchable
            insert directed edge (u, v) with weight(u, v) =
            (u.at − u.dt)+ (v.dt − u.at)                     ▷ travel time + waiting time for next flight
        end if
    else                                                     ▷ arrive at final destination airport
        insert directed edge (u, v) with weight(u, v) =
        (u.at − u.dt)   ▷ only the travel time   ▷ No more waiting time as destination is reached
    end if
end if
```

3. There is a directed edge (v, dn) from normal node v to dn, where v.sa = DA. weight$(v, dn) = 0$

The graph looks like the following (without showing edge weights):

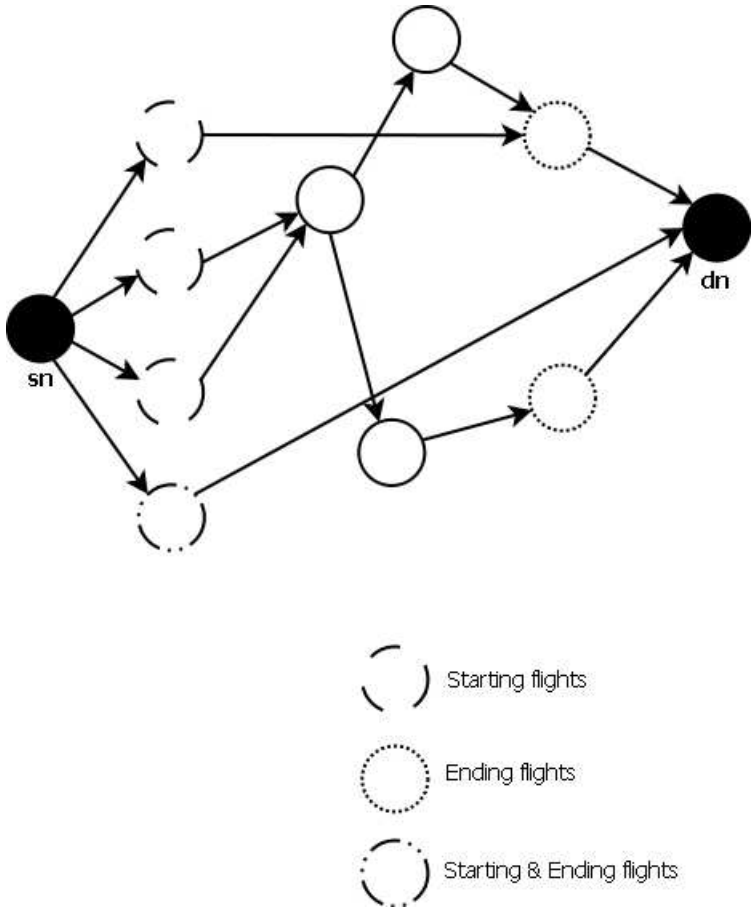


Figure 6:

Algorithm:-
Perform modified Dijkstra's algorithm starting from sn. At the end, backtrack from dn. Discarding sn and dn from the path and reversing the path gives the flight route from SA to DA with the earliest arrival time.

*adding a special source vertex sn is to simplify our starting point where sa = SA, adding a special target vertex dn is to simplify our ending point where sa = DA.

Other graph modeling may be possible. The important takeaway message from this problem is the graph modeling skill.

Also since the graph is actually a DAG (since you can't travel back in time), the best algorithm is the 1-pass Bellman Ford's (SSSP on DAG) instead of Dijkstra's. However, it is usually OK to just use Dijkstra's as $O((V + E) \log V)$ is not very different compared to $O(V + E)$.

Problem Set 5 – A Very Quick Discussion

Q5. Discussion of PS5 Subtask 1-2-3-4 :O...

Ans: Subtask 1-2-3-4: The graphs given are essentially a tree (Subtask 1), an unweighted graph (Subtask 2), a small weighted graph (Subtask 3), and a medium weighted graph (Subtask 4)... All these have been discussed in Lecture 07 and 08. This is a very easy PS =).