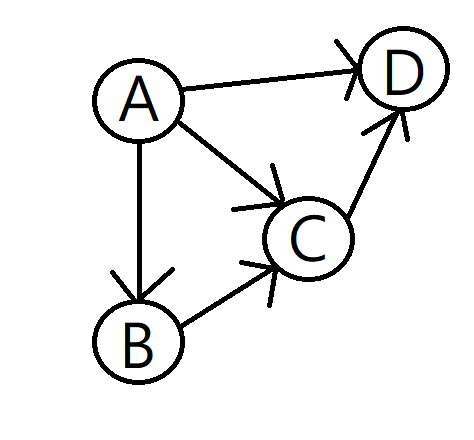
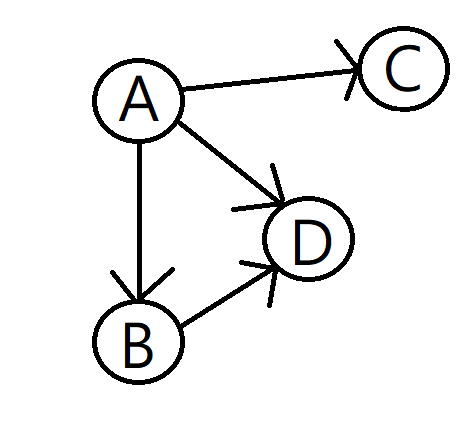
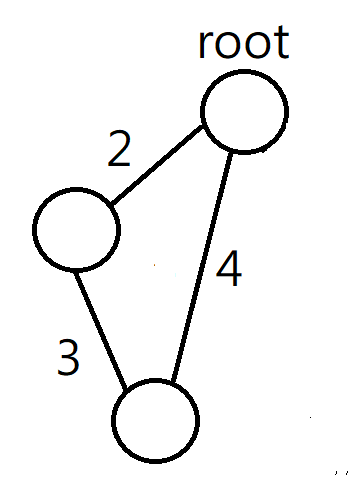
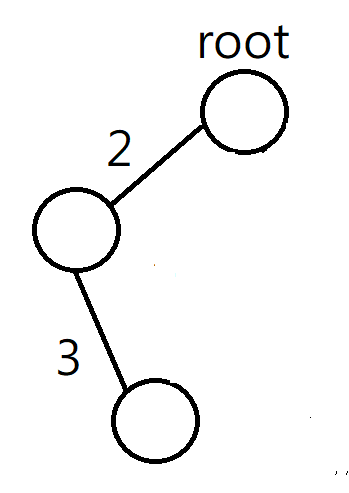
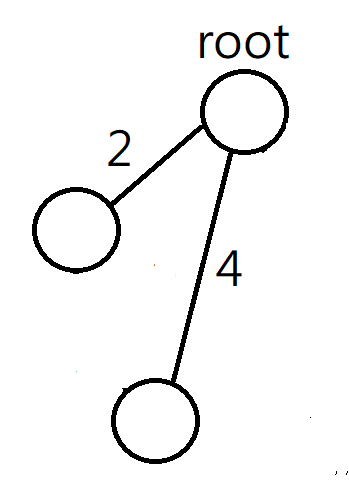
ADA HW3 b07902126 資工二 謝宗儒

Problem D

1. Same order different order

1. Original MST shortest path tree

As graph showed, MST might not be the same with shortest path tree when select some of nodes as root

3.(a)

Topological-sort(graph) //cost O(V+E)

sort by adjacency list

INITIALIZATION(Graph, c, M, start) //cost O(V)

For i in Graph //for each node

M(i)= inf

M(start)=c(start)

RELAX(u, v, c, r, M) //cost O(1)

if M(v) > M(v) + r(u,v) + c(v)

M(v) = M(v) + r(u,v) + c(v)

lawler(Graph, c, r, M,start)

topological-sort(Graph)

INITIALIZATION(Graph, c, M, start)

for each vertex u taken in topologically sorted order // totally V operation

for each vertex v in Graph.adjacency[u] //totally E operation

RELAX(u, v, c, r, M)

(b) according to the pseudo code above

Time complexity = O(V+E) + O(V) + O(V + E) 🡺 O(V + E)

Since the roads are one-way, M of a city has n possibilities (n = in-degree of the city) and each of the possibilities M(x) = M(i)+r(i, x)+ c(x).

Then, we can regard r(i, x)+c(x) as the weight of the edge, so the problem will be simplified to be a shortest path problem without cycle which can be solved by lawler algorithm.

Problem E

(1)

Initialization(Graph, start) //O(N)

for v in Graph.node

v.d = inf

start.d = 0;

Relax(u, v, w) //O(1)

If v.d > u.d + w(u,v)

v.d =u.d + w(u,v)

Dijkstra(Graph, W, start)

Initialization(Graph. start)

put start in Set //Keep Set as a min-heap tree //O(1)

while Set is not empty //totally N times

u = extract-min(Set) //O(log N)

for v in Graph.adj[u] //totally E times

Relax(u, v, w(u,v))

run Dijkstra(Graph, mailing fee, 0) and then the mailing fees have to pay for sending from 0 to node x (x = any one factory) = 2\*x.d

(2)

Since it becomes a positive weighted directed graph, Dijkstra algorithm can solve this problem with that used in (1).

Correctness : We get the shortest path distance from 0 to any other nodes for a positive weighted directed graph by applying Dijkstra algorithm. Therefore, we can apply this algorithm to the graph and its reverse graph to find the shortest path distance to and from each factory.

Time complexity : O(N) + O(1) + O(NlogN + E) = O(NlogN + E)

Since the graph is connected, N <= E, O(NlogN + E) <= O(ElogE + E) = O(ElogE)  
 belongs to O(ElogE).

(3)

Initialization(Graph, start) //O(N)

for v in Graph.node

v.d = inf

start.d = 0;

Relax(u, v, w) //O(1)

If v.d > u.d + w(u,v)

v.d =u.d + w(u,v)

Bellman-Ford(Graph, W, start)

Initialization(Graph, start)

for i = 1 to |G.V| - 1 //totally N-1 times

for (u, v) in G.E //O(E)

Relax(u, v, w(u,v))

for (u, v) in G.E //O(E)

if v.d > u.d + w(u, v)

print(I am rich)

run Bellman-Ford(Graph, W, 0) and Bellman-Ford(Graph\_inverse, W, 0)

Then, mailing fees have to pay for sending from 0 to node x = adding the two results of x.d (x = any one factory) together.

correctness : Since Bellman-Ford algorithm can deal with general graph and weight , and also detect negative cycles. Therefore, I can use Bellman-Ford algorithm to find the shortest path distance from 0 to factories and use it again with reversed graph to find the shortest path distance from factories to 0. Then, check whether there is negative cycles.

Time complexity : 2(O(N) + O(1) + O((N-1)\*E) + O(E)) = O(NE)

(4)

Subproblem2 :

Time complexity : O(NlogN + E) 🡪 O(ElogE)

Space : O(N) // or O(N+E) if include the graph

Advantage : better time complexity

Disadvantage : can only apply graph with positive weight

Subproblem3 :

Time complexity : O(N\*E)

Space : O(N) // or O(N+E) if include the graph

Advantage : can apply to general graph (with negative weight) and detect

negative cycles

Disadvantage : worse time complexity

(5)

Reweight w(u,v) as w(u, v) – r(v)/K // r(v) means reliability of v.

By this modification, we can simplify the problem to be detecting negative cycles.

And then, we can apply Bellman-Ford algorithm to detect negative cycles.

Initialization(Graph, start) //O(N)

for v in Graph.node

v.d = inf

start.d = 0;

Relax(u, v, w) //O(1)

If v.d > u.d + w(u,v)

v.d =u.d + w(u,v)

Reweight(r,w,K) //O(E)

for(u, v) in Graph.edge

w(u, v) = w(u, v) – r(v)/K

Bellman-ford(Graph, r , w, K, s)

Initialization(Graph, start)

Reweight(r,w,K)

for i = 1 to |Graph.vertex|-1

for (u, v) in Graph.edge

Relax(u, v, w)

for(u, v) in graph.edge

for (u, v) in G.E //O(E)

if v.d > u.d + w(u, v)

print(trustful cycle exist)

correctness : For a trustful cycle, (total reliability value)/(total weight) > K

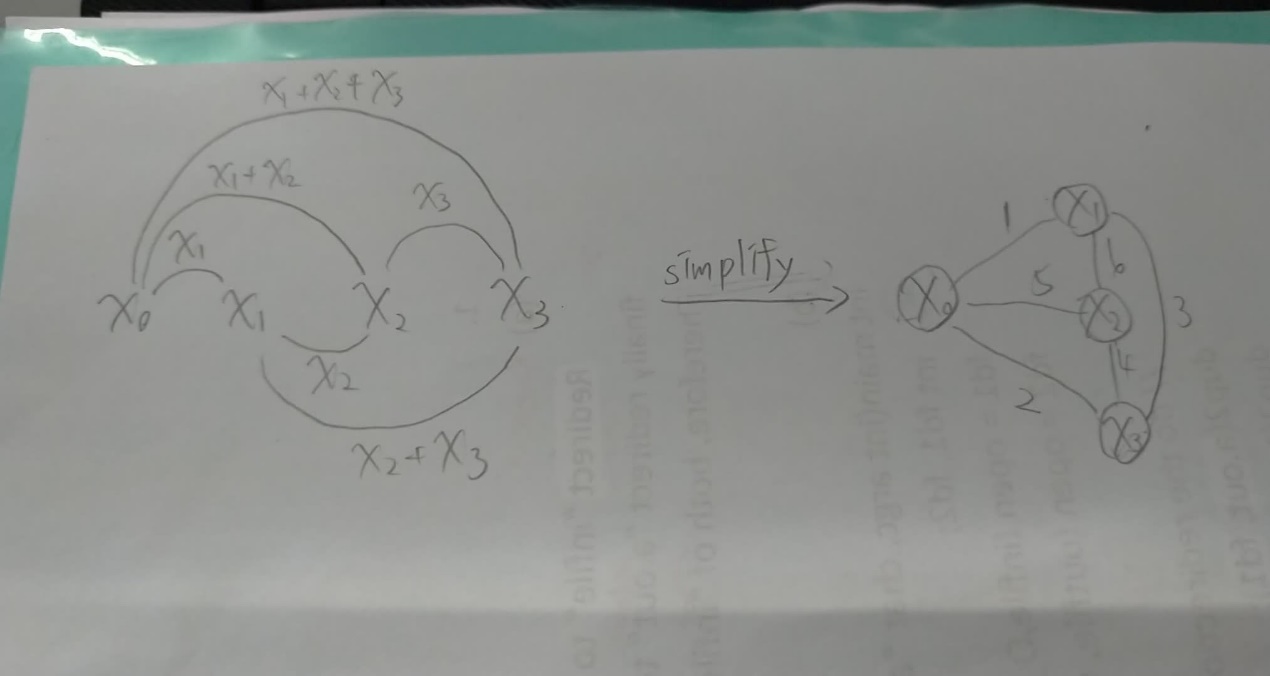
* (total reliability value/K )> (total weight)
* (total weight) -(total reliability value/K ) < 0
* sum(w(u, v)) – sum(r(v))/K < 0 --------(\*)

We can modify w(u,v) as w(u, v) – r(v)/K for each edge. If there exists any negative cycles, that means there exist trustful cycle for original graph because of the equation(\*).

Time complexity : O(N)+O(1)+O(E)+O(NE) = O(N\*E)

Problem F

We can transfer the equation pool to a graph (for example)



By observation, all xi (i=1 to N) are used two times in inconsistent solutions which form a cycle in the graph. Therefore, spanning tree can select consistent solutions. To minimize the total cost, we can simplify this problem to MST which can be solved by Prim’s algorithm.

Build the graph cost O(V + E) = O(N+1+C(N+1, 2)) = O(N+N^2) = O(N^2)

Prim’s algorithm with fabonacci heap cost O(E + VlogV) = O(N^2 + (N+1)log(N+1)) = O(N^2).

Time complexity = O(N^2) + O(N^2) = O(N^2)