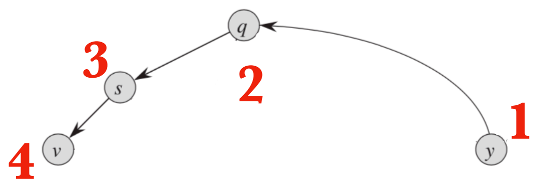
ALGORITHM DESIGN AND ANALYSIS - HOMEWORK 3

B07902143 陳正康

Problem D

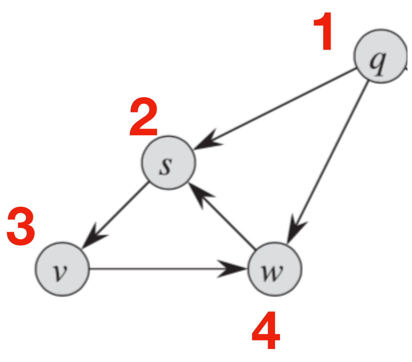
1. (q, s, v, w/y correspond to A, B, C, D)

   Same order:

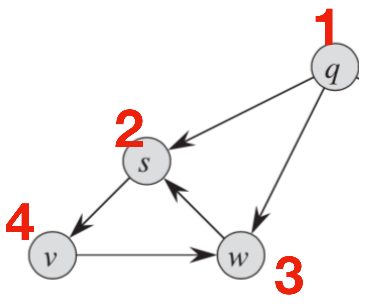


   Different order:

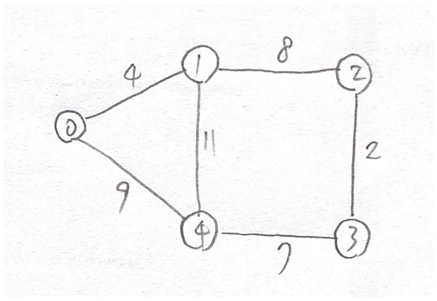
   DFS:



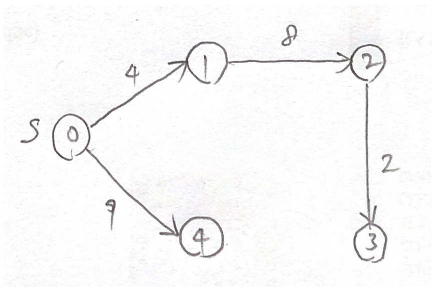
   BFS:



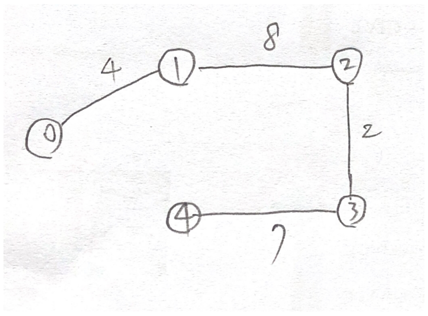
2. Graph:



   Shortest Path Tree:



   Minimum Spanning Tree:



   The shortest path tree MAY NOT be the same as minimum spanning tree.

3. ===^_^===

(a) Construct a graph G where vertex set V is all the cities, edge set E is the road between cities. The edge weight function w is defined as:
$w(u,v) = r(u,v) + c(v) \quad \forall (u,v) \in E$

Now if we apply the algorithm of Single-source Shortest Paths in Directed Acyclic Graphs, we can find $M(u) = c(s) + \delta(s,u)$ for every non-capital city u , where s is capital city and $\delta(s,u)$ represent shortest distance from s to u

Pseudo Code:

```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices of G
  for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
  for each vertex u, taken in topologically sorted order
    for each vertex  v ∈ G.Adj[u]
      if v.d > u.d + w(u,v)
        v.d = u.d + w(u,v)
        v.π = u
```

(b) The topological sort takes $O(V+E)$ time, then a traversal through all vertices is $O(V+E)$, so overall $O(V+E)$

Since G is a DAG by definition (all roads being one-way and no cycle), for any city u , define $D(u,P)$ as the number of days required for some simple path $P = \{(v_1,v_2),(v_2,v_3),\cdots,(v_{n-1},v_n)\}$ from capital city s to u , $u = v_n$, $s = v_1$:

$$D(u,P) = \sum_{i=1}^{n-1} r(v_i,v_{i+1}) + \sum_{i=1}^{n} c(v_i)$$

$$= c(s) + \sum_{i=1}^{n-1} r(v_i,v_{i+1}) + \sum_{i=1}^{n-1} c(v_{i+1})$$

$$= c(s) + \sum_{i=1}^{n-1} w(v_i,v_{i+1})$$

Thus, $M(u) = \min\{D(u,P) \mid \text{any simple path } P\}$. To minimize $D(u,P)$ is to minimize $\sum_{i=1}^{n-1} w(v_i,v_{i+1})$, that is the shortest distance from s to u

Problem E

(1) Define: G is the graph, G.V is the array of factories, G.E is the set of roads, w(u,v) represents the mailing fee between factory u and factory v.

Pseudo code:

```
Dijkstra(G, N)
   for i=1 to N
      G.V[i].d = $\infty$
   G.V[0].d = 0
   Q = G.V
   while Q $\neq \emptyset$
      u = extract-min(Q)
      for each vertex v $\in$ G.Adj[u]
         if v.d > u.d + w(u,v)
            v.d = u.d + w(u,v)

solve(G, N)
   Dijkstra(G, N)
   for i=1 to N
      result[i] = G.V[i].d $\times$ 2
   return result
```

(2) Pseudo code:

```
solve(G, N)
   Dijkstra(G, N)
   G' = reverse all edges of G
   Dijkstra(G', N)
   for i=1 to N
      result[i] = G.V[i].d + G'.V[i].d
   return result
```

A shortest path about a factory consists of two parts: "to" part must be the shortest path from start point to the factory, and "back" part must be the shortest path from the factory to the start point, otherwise it cannot be optimal path (argued by copy-and-paste argument)

So we have two subproblems: single-source-shortest-path and single-destination-shortest-path
The latter can be reduced to single-source if we reverse all edges
Then, since there is no negative edge, we apply Dijkstra's Algorithm on both ($O(E \log E)$ time), after which we sum up the result of two (linear time)
Since the graph is connected, $N = O(E) = O(E \log E)$, the overall time complexity is $O(E \log E + N) = O(E \log E)$

(3) Pseudo code:

```
BellmanFord(G, N)
   for i=1 to N
      G.V[i].d = ∞
   G.V[0].d = 0
   for i=0 to N-1
      for each edge (u,v) ∈ G.E
         if v.d > u.d + w(u, v)
            v.d = u.d + w(u, v)
   for each edge (u,v) ∈ G.E
      if v.d > u.d + w(u, v)
         return False
   return True

solve(G, N)
   if not BellmanFord(G, N)
      Print "I am rich!"
   G' = reverse all edges of G
   BellmanFord(G', N)
   for i=1 to N
      result[i] = G.V[i].d + G'.V[i].d
   return result
```

Bellman-Ford's Algorithm can handle negative edges and detect negative cycles, and runs in $O(NE)$ time

(4)

Subproblem	2	3
Algorithm	Dijkstra's	Bellman Ford's
Space Complexity (Using adjacency list)	$O(V+E)$	$O(V+E)$
Time Complexity	$O((V+E)\log V)$	$O(VE)$
Pros	Faster	Can handle negative edge and cycle, simple in implementation
Cons	Cannot handle negative edge or cycle	Slower

(5) The key is to re-weigh the graph.

Define:

f : mailing fee function of any edge

r : reliability function of any vertex

Suppose a cycle is composed of vertices $v_1, v_2, \cdots, v_n$ and edges

$e_1 = (v_1, v_2), e_2 = (v_2, v_3), \cdots, e_{n-1} = (v_{n-1}, v_n), e_n = (v_n, v_1)$

By definition, the cycle is a trusted cycle if and only if:

$$\sum_{i=1}^{n} r(v_i) > K \sum_{i=1}^{n} f(e_i)$$

$$\iff \sum_{i=1}^{n} r(v_i) > \sum_{i=1}^{n} K f(e_i)$$

$$\iff \sum_{i=1}^{n} (r(v_i) - K f(e_i)) > 0$$

Define edge weight function w as: $w(u, v) := K f(u, v) - r(u)$

$$\iff \sum_{i=1}^{n} w(e_i) < 0$$

That is, a cycle is a trusted cycle if and only if it's a negative cycle

So we can use Bellman-Ford's Algorithm to detect negative cycle (same as (3))

The time complexity of Bellman-Ford's Alg. is $O(NE)$

Problem F

1. Observation: Let $s_i = \sum_{i=1}^{n} x_i$,$1 \le i \le n$ and $s_0 = 0$, solving $x_1, x_2, \cdots, x_n$ is exactly the same problem as solving $s_1, s_2, \cdots, s_n$, since each x_i can be obtained from $s_i - s_{i-1}$, and each s_i can be obtained by adding up some x (i.e. there exists bijection mapping)

2. Modeling: Then construct a undirected graph with $n + 1$ vertices indexed from 0 to n, representing s_0 to s_n
   For each element a in the equation pool, suppose $a = x_i + x_{i+1} + \cdots + x_j$, construct an edge about a by connecting s_{i-1} and s_j , whose weight is the cost of a
   Then we can apply Kruskal's Algorithm to obtain a minimum spanning tree
   The edges of the minimum spanning tree is the minimum-cost solution. If any vertex cannot be reached, the problem cannot be solved

3. Correctness: consider a directed path $\{(s_0, s_{b_1}), (s_{b_1}, s_{b_2}), \cdots, (s_{b_{k-1}}, s_{b_k})\}$ on the minimum spanning tree, for an edge (s_i, s_j) on this path:
   If $i < j$, it represents the element $x_{i+1} + \cdots + x_j$ in the pool
   If $i > j$, it represents the element $-(x_{j+1} + \cdots + x_i)$ in the pool (the negative sign means we take it out of the pool and negate it)
   WLOG, consider there exists a "reverse point" s_{b_m} , such that
   $\forall 1 \le i < m \ (b_i < b_{i+1}) \land \forall m \le i \le n \ (b_i > b_{i+1})$
   Then we add up these elements from the pool:
   $(x_1 + \cdots + x_{b_1}) + (x_{b_1+1} + \cdots + x_{b_2}) + \cdots + (x_{b_{m-1}+1} + \cdots + x_{b_m}) - (x_{b_m+1} + \cdots + x_{b_{m+1}}) - \cdots - (x_{b_{k-1}+1} + \cdots + x_{b_k})$
   $= x_1 + x_2 + \cdots + x_{b_k} = s_{b_k}$
   that is, if s_{b_k} is reachable from s_0 on graph, its value can be solved
   We want to make all s_i reachable from s_0 by selecting edges with minimum cost, so minimum spanning tree will give us correct solution

4. Time complexity: Kruskal's Algorithm takes $O(E \log E)$
   We add a test in the beginning: if $N < n \rightarrow$ this problem is unsolvable
   so n is bounded by N
   vertex number V is $n + 1$, edge number E is N
   After that, we solve each x_i from $s_i - s_{i-1}$, which takes $O(n)$
   Overall: $O(E \log V + n) = O(N \log n + n) = O(N \log N + N) = O(N^2)$