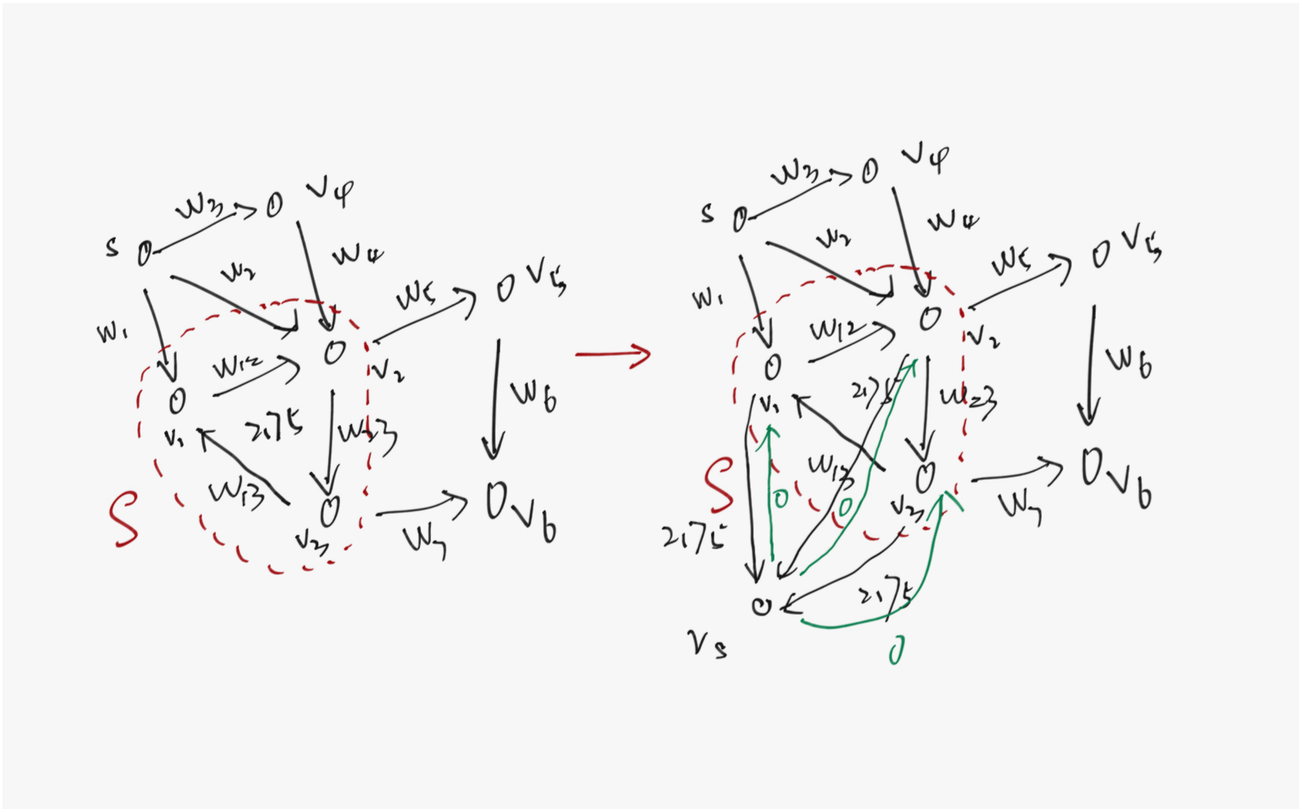
HW11

1. *Let G = (V,E) be a weighted directed graph and let s ∊ V be a designated vertex; assume that all the edge weights are non-negative. The task is to provide an algorithm to determine for each vertex v ∊ V whether there is a unique shortest path from s to v. There is a solution which avoids counting the number of paths; try to find it. Argue that your solution is correct. The resulting algorithm should have the same running time as Dijkstra's algorithm on a graph of n vertices and m edges, up to constant factors; argue that your algorithm does so. You may assume n ≤ m.*

> PathNum[v]: record the number of shortest paths of v
>
> IsUnique[v]: record if v has a unique shortest path, the results are represented by this array.
>
> The idea is that if we find a shorter distance of v, assign the number of the shortest path of u to v. If the distance is equal, which means it may not have only one shortest number, so the number of the shortest path of v is PathNum[v]+PathNum[u]. Finally, we iterate the array PathNum, if PathNum[v] = 1, then it has a unique shortest path, setting IsUnique[v] to true.
>
> We add two arrays in the initial loops and in the last iteration and extra if conditions cost $O(1)$ time, so the total time complexity is $O((n+m)\log n + 3n) = O((n+m)\log n)$

```
Dijkstra(G,s)
for v=1 to n do
  Dist[v]<-MaxInt
  PathNum[v] = 0
  IsUnique[v] = false
end for
Dist[s] ← 0,PathNum[s]<-1;
EnQueue all v ∈ V on a PriorityQueue Q with key Alarm[v]
While Q is not empty do
  u<-DeleteMin(Q)
  for each edge (u,v) do
    if Dist[v] > Dist[u] + length(u,v) then
      Dist[v]<-Dist[u]+length(u,v)
      PathNum[v]<-PathNum[u]
      ReduceKey(Q,v,Dist[v])
    else if Dist[v] = Dist[u] + length(u,v) then
      PathNum[v]<-PathNum[v]+PathNum[u]
      ReduceKey(Q,v,Dist[v])
    end if
  end for
end while
for v=1 to n do
  if PathNum[v] = 1 then IsUnique[v] = true
end for
```

2. *Taxi and subway. Let G = (V,E) be a weighted directed graph and let s ∊ V be a designated vertex. All the weights are positive and indicate the dollar cost of traversing an edge by taxi. Let S be a subset of the vertices, the subway stops. You can travel from any vertex in S to any other vertex in S for a cost of $2.75. Your task is to compute, for every vertex v ∊ V , the cost of the least cost path from s to v, using any mode of transport (some combination of taxis and subway). You are to do this by means of a reduction to the single source shortest path problem. Argue that your reduction is correct. Your algorithm should run in the same time as Dijkstra's algorithm on a graph of n vertices and m edges, up to constant factors; argue that it does so.*

> We can consider subset S as strongly connected component and the weight of this component, let's say v_s, is 2.75(the cost of subway). Therefore we built the following graph $H = (W, F)$. For each vertex in subset S, make them point to v_s with edge weight 2.75. Then v_s points to each edges in subset S with weight 0.
>
> 
>
>
> For example, subset S is the red circle in the left part, and start from s. First, for each vertex in subset S, make it point to v_s with weight 2.75. Then make v_s point to each vertice in S with weight 0. Therefore, we can take the subway to any other vertices in subset S with cost 2.75, though they could choose to take the taxis to do this.
>
> Clearly, we can build H $O(n_S + m_S)$ time because we can make it by traversing all the vertices and edges in S. We claim that the reduction is correct. To see this, we can see for each path passing through S, we keep the original graph that go through like v_2 but not take the subway and reduce taking the subway into a vertex with weight 2.75 and go to any other vertices for free. Therefore, we can conclude that the shortest path from s to v in H is no longer than the shortest path from s to v in G

> The reduction procedure cost $O(n_S + m_S)$ time, so the total cost is still $O((n+m)\log n)$.

3. *Let G = (V, E) be a directed graph with two weights, we and re, on each edge e; assume that all the edge weights are non-negative. There are two special vertices in this graph: c, the chemical plant, and a, the amphitheater. The task is to create an evacuation plan from a to v for every vertex v in V in the event that the chemical plant catches fire. The time it takes the fire to spread along an edge e is re. The time to travel along an edge is we. We want to find the shortest safe path from a to v for each v∈V: a path is safe if it does not include any vertex on fire at the time that vertex is traversed. Your algorithm should run in the same time as Dijkstra's algorithm on a graph of n vertices and m edges, up to constant factors. Justify your running time. Also, argue that your construction is correct.*

   Hint. i. First compute the time at which each vertex catches fire, assuming the chemical plant catches fire at time 0.
   ii. Now modify Dijkstra's algorithm so that it does not use any vertex on fire on the paths it computes.

   > $FireTime[v]$: We use this to store the time at which each vertex catches fire.
   >
   > $CanEvac[v]$: Initially False for all $v \in V$, use to check if this vertex is safe.
   >
   > The idea has two steps:
   >
   > 1. Use c as the start vertex, find the shortest path from c to v for every vertex $v \in V$. If there are multiple c in the graph, use a start vertex s to connect all the c with weight 0. After reduction, this can be solved by Dijkstra's algorithm directly. The result is the time at which each vertex catches fire. We store them in an array $FireTime[v]$.
   > 2. Use a as the start vertex, find the shortest path from a to v for every vertex $v \in V$. If there are multiple a in the graph, reduce it like step 1 as well. Use modified Dijkstra's algorithm to find the shortest safe path. The difference is that for $u \leftarrow DeleteMin(Q)$, if $CanEvac[u]$ is true, we continue the algorithm, else we try to use another u. Then for each edge (u, v), if $Dist[u]$ + $w_e < Dist[v]$ and $FireTime[v] > Dist[v]$, $Dist[v] \leftarrow Dist[u] + w_e$ and set $CanEvac[v] \leftarrow True$. This means if travel time is less than fire spread time, this vertex is safe, then it can be used to traverse in the following steps as well.
   >
   > Therefore, $CanEvac[v]$ store the vertices which have the shortest safe path and the length stores in the $Dist[v]$. In this algorithm, we run Dijkstra's algorithm twice and add $O(1)$ comparison in the second Dijkstra's algorithm, so the total time complexity is
   > $O(2(m+n)\log n) = O((m+n)\log n)$

4. *There is an algorithm for the all pairs shortest path problem that runs in O(n3) time, called the Floyd-Warshall algorithm. Here, the input is a directed weighted graph H = (VH , EH) given in adjacency matrix format. You may assume there is a matrix WH[1 : n,1 : n] which stores the weights of the edges, where, for pairs of vertices u,v with (u,v) ∈/ EH, WH [u, v] = ∞. The output is the length of the shortest path from u to v for each pair u, v of vertices. The assumption needed here is that there are no negative length cycles.*

   Suppose you are given a directed weighted graph G = (VG,EG), with the edge weights stored in matrix WG[1 : n,1 : n]. The task is to give an algorithm to determine, for every pair of vertices u, v ∈ VG, the length of a shortest path from u to v that uses an even number of edges. Solve this problem by means of a reduction to the standard all pairs shortest path algorithm. Your algorithm should run in O(n3) time; justify your running time.

> This is similar to the additional question 4 in week 11. We can build a graph $H(F, W)$. H has two copies of each vertex in G, the even copy, and the odd copy, i.e., for each vertex $v \in V$,F has two vertices v^e and v^o. For each edge $(u, v) \in E, W$ has two edges (u^e, v^o) and (u^o, v^e). Here, we use the same method. The shortest path from s to v is actually $s^e, v_1^o, v_2^e, \cdots, v_{2k+1}^e$. Then we should only report the length of the shortet path to v^e in H for each v in G.
>
> By using a method similar to question 4, we actually have $2n$ vertices and $2m$ edges in H, so the matrix W_G will become $W_G[1 : 2n, 1 : 2n]$. We can use first n rows/columns to represent the even vertices, the second n rows/columns to represent the odd vertices. Now, we can use the edges in H to initialize the weights in W_G. After that, we can directly use the Floyd-Warshall algorithm to solve the problem now, the result will be stored in the left-topmost part of the matrix, i.e., the shortest path from s^e to v^e. The runtime is $O(2^3n^3)$ depending on the W_G, which is still $O(n^3)$.