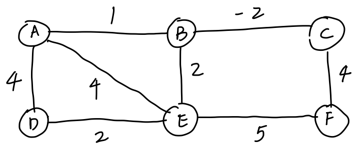
2 a)

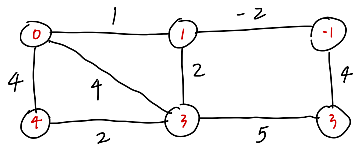
Consider the following counterexample G :



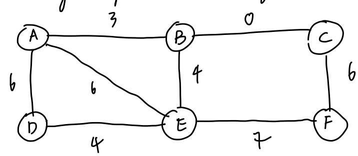
minimum weight edge = (B, C)

$w(B, C) = -2$

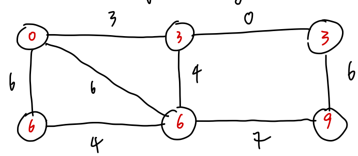
Assume we want to find the shortest path started from A:



Following the procedure in the question, we have G' :



Now, use Dijkstra's Algorithm on G' started from A we get :



And we can see that in G', the shortest path from A to E is : $A \to E$.

However, in G, the original shortest path from A to E is : $A \to B \to E$

And the reason of this "fix" cannot hold is because when adding $|w(\text{minimum weight edge})|$, the path with less edges will add up less total. Thus, this can make some "path" that is not the shortest in the origin G becomes the shortest in G'.

2 b)

We will make a modification of the grid-graph to make it work for Dijkstra's algorithm. We will use the similar strategy in 2a) and get G' which has all non-negative weighted edges. Now, we will use Dijkstra's algorithm on G' to find the shortest path from v_{11} to all other vertices.

We will prove that this algorithm is correct.

Let P be the shortest path from v_{11} to v_{ij} in G'. Let P' be some other path from v_{11} to v_{ij}.

Let $W(P)$ be the total weight for a path P in G, $W'(P)$ is the total weight in G'.

Then, $W'(P) \leq W'(P')$.

Next, we will try to show that P is exact the same shortest path in G, since the length of P and all other P' to v_{ij} are same.

Since G and G' are directed as shown, the length from v_{11} to v_{ij} would be exact $(i-1)$ to the right and $(j-1)$ going up, which is $(i+j-2)$.

Thus, assume w is the minimum weight, $W'(P) = W(P) + (i+j-2)\cdot|w|$ and $W'(P') = W(P') + (i+j-2)\cdot|w|$

We get $W(P) \leq W(P')$ for any other path P' from v_{11} to v_{ij}.

Therefore, we can make sure that the path we use Dijkstra's algorithm on G' is the correct shortes path in original G.

2c)

The algorithm is as follow: ($SD(V_1, V_2)$ = shortest distance from V_1 to V_2)

① for V_{1j} and V_{i1}, $i=1,\cdots,r$, $j=1,\cdots,c$,

$$SD(V_{11}, V_{1j}) = w(V_{11}, V_{12}) + w(V_{12}, V_{13}) + \cdots + w(V_{1(j-1)}, V_{1j})$$

$$SD(V_{11}, V_{i1}) = w(V_{11}, V_{21}) + w(V_{21}, V_{31}) + \cdots + w(V_{(i-1)1}, V_{i1})$$

② for other V_{ij},

$$SD(V_{11}, V_{ij}) = \min\{SD(V_{11}, V_{(i-1)j}) + w(V_{(i-1)j}, V_{ij}),\ SD(V_{11}, V_{i(j-1)}) + w(V_{i(j-1)}, V_{ij})\}$$

Proof of correctness:

We will try to prove by induction:

- Base cases:

for V_{i1}, $i=1,\cdots,r$, the only path from V_{11} to V_{i1} is $V_{11} \to V_{21} \to \cdots \to V_{i1}$. Thus $SD(V_{11}, V_{i1}) = w(V_{11}, V_{21}) + w(V_{21}, V_{31}) + \cdots + w(V_{(i-1)i}, V_{i1})$

Similarly,

for V_{1j}, $j=1,\cdots,c$, $SD(V_{11}, V_{1j}) = w(V_{11}, V_{12}) + w(V_{12}, V_{13}) + \cdots + w(V_{1(j-1)}, V_{1j})$

- Induction Hypothesis:

for $V_{(k-1)l}$ and $V_{k(l-1)}$, the shortest path is correct and shortest.

- Induction Conclusion:

for V_{kl}, since the only parents of V_{kl} is $V_{(k-1)l}$ and $V_{k(l-1)}$, the shortest path from V_{11} to V_{kl}, if exists, will include either $V_{(k-1)l}$ or $V_{k(l-1)}$. By IH, we know that the shortest paths to both $V_{k(l-1)}$ and $V_{k(l-1)}$ exist. Thus,

$$SD(V_{11}, V_{kl}) = \min\{SD(V_{11}, V_{(k-1)l}) + w(V_{(k-1)l}, V_{kl}),\ SD(V_{11}, V_{k(l-1)}) + w(V_{k(l-1)}, V_{kl})\}$$

And whichever vertex we choose to make the path shorter, the shortest path to V_{kl} will be the shortest path to it and the path from it to V_{kl}.

Therefore, by POMI, our algorithm is correct that it will make sure to find the shortes path from V_{11} to all other vetices.

Time Complexity:

We will go through all vertices once to see if its parent for shortes path is the one on the left or the one below. Thus, we will go through all $r \times c$ vertices. And at each vertex, we will use $O(1)$ to do the comparison. Therefore, $T(n) = c \times r \times O(1) = O(cr)$

As for the algorith to find the longest distance, we will modify the algorithm above at part ②, (we will change the name SD into LD refering to the longest distance)

②* $LD(V_{11}, V_{ij}) = \max\{SD(V_{11}, V_{(i-1)j}) + w(V_{(i-1)j}, V_{ij}),\ SD(V_{11}, V_{i(j-1)}) + w(V_{i(j-1)}, V_{ij})\}$

3a) My algorithm would be as follow:

```
SinglePath ( G-set, s, t ):
  // G-set = {G_1(V,E_1), ..., G_k(V,E_k)}
  E ← E_1
  For i ← 1 to k:
      For each e in E:
        if e is not in E_i
           remove e from E
      done
  done
  Let G(V,E) with all edges weighting 1.
  return  ssspDAG( G(V,E) )
```

(ssspDAG is the procedure from slides that can find the shortest s-t path)

First, we will try to find the intersection of all edge sets E_i, denoted E, and construct a new graph $G(V,E)$.

Since $E \subseteq E_i$ for $i=1,\dots,k$, the path we find in G must exist in all graphs G_i. Therefore, if we find the shortest path in G, it will be the common path which can minimize the cost.

As for the time complexity, the first loop that we find the intersection of all E_i is $\sum_{i=1}^{k} |E_i| \le \sum_{i=1}^{k} |V|^2 \in O(k \cdot |V|^2)$

Calling ssspDAG is $O(|V|+|E|) = O(|V|+|V|^2) = O(|V|^2)$. Therefore, the total time complexity is $O(k \cdot |V|^2)$

3 b)

We will design a dynamic programming algorithm

As the hint mentions, for each G_i, it either doesn't switch or does switch. Let minimizePaths $((G_1, \dots, G_k), cost, i)$ be the algorithm where i means the ith graph is the last switch. Then here is the recurrence expression:

minimizePath $((G_1, \dots, G_k), cost, i) = \min\{$ minimizePath$((G_1, \dots, G_k), cost, i-1)$, minimizePath$((G_1, \dots, G_{i-1}), cost + \text{Single}((G_i, \dots, G_k)) \times (k-i+1) + \lambda, i-1)\}$

where Single$((G_i, \dots, G_k), s, t)$ is in 3a) and it will return the minimum weight path if there is no switch among $G_i, \dots, G_k$.

If Single fails, it means there is the switch and we will have to choose the other option.

Then, if we go through $(G_1, \dots, G_k)$ from the back to run minimizePath, we will get the minimum cost that satisfies the condition.

Therefore, my algorithm is going through $(G_1, \dots, G_k)$ by calling minimizePath$((G_1, \dots, G_k), cost, k)$.

This algorithm is correct since whenever there is a switch, we will call Single to calculate the minimum weights for graphs after the last switch graph plus λ and compare it with not switching. After computation, we will get the minimum cost.

Time Complexity:

Every recurrence, we will reduce the index of graphs, and there will be total k times

minimizePath $(\dots, k)$ — k

minimizePath $(\dots, k-1)$ — minimizePath $(\dots, k-1)$ + Single(1) — $k-1$ — $1 \cdot |V|^2 \cdot 1$

m$(\dots k-2)$ — m$(k-2)$ + S(2) — min$(k-2)$ — min$(k-2)$ + S(1) — $\vdots$ — $2 \cdot |V|^2 + 1 \cdot |V|^2$

$\vdots$

... — ... — 1 — $k \cdot |V|^2 + \dots + 1|V|^2$

$$T(n) = |V|^2 + (2|V|^2 + |V|^2) + \dots + (k|V|^2 + \dots + |V|^2)$$

$$= (1 + 1+2 + 1+2+3 + \dots + 1+2+3+\dots+k)\,|V|^2$$

$$= \sum_{i=1}^{k} \frac{(1+i)i}{2} \cdot |V|^2 \in O(k^3 \cdot |V|^2)$$

4a)

Input: $G(V,E)$, an instance of HP (as HAM-CYCLE to the converter)

HC−TSP converter:

Let $G^*(V,E^*)$ be a complete graph with edge weights:

$w(u,v) = 0$ if $(u,v) \in E$

$w(u,v) = 1$ if $(u,v) \notin E$

Output:

$G^*(V,E^*)$, an instance of PTSH (as TSP) and the target value 0.

If the solution S_{TSP} to Π_{TSP} is YES, then in G^*, there is a cycle that all weights of the edges are 0 so that total ≤ 0 and it passes through all vertices. And since all those edges have weight 0, they $\in E$. Therefore, this cycle is also in G and it passes through V. Thus, $S_{HAM\text{-}CYCLE}$ to $\Pi_{HAM\text{-}CYCLE}$ is YES.

If the solution S_{TSP} to Π_{TSP} is NO, then in G^*, then there doesn't exist a cycle with weight ≤ 0. Either among all cycles there is some edge with weight $= 1$, or there is no cycle in G^*. In the first case, those edges with weight $= 1$ don't exist in G. In the second case, G will has no cycle passing all vertices V. Thus, $S_{HAM\text{-}CYCLE}$ to $\Pi_{HAM\text{-}CYCLE}$ is NO.

4b) My algorithm will be as follow:

```
easy Convert ( G(V,E) as Π_HAM-CYCLE ).
    (G*(V,E*), k) ← HC-TSP( G(V,E) )
    return MAGICAL-POLYTIME_TSP_ALG( G*(V,E*), k )
```

My algorithm convert $\Pi_{HAM-CYCLE}$ to Π_{TSP} and since we have the magical poly-time algorithm, we pass the Π_{TSP} and return the solution S_{TSP} to it. And based on the argument we have in 4a), the solution S_{HAM_CYCLE} is solved.