# Graph Drill session - 2

## 1. High Score

Use Bellman - Ford

In what case we will get a larger positive value? Whenever there is a presence of positive cycle.

→ If there is a positive cycle, you can keep iterating over it to increase the overall score definitely.

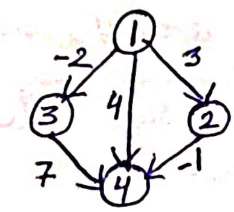
→ Bellman Ford helps compute shortest path even with negative weights.

→ If a negative - weight cycle is reachable, shortest paths aren't well-defined since cost can decrease infinitely.

→ Here, we use edge relaxation to implement this.

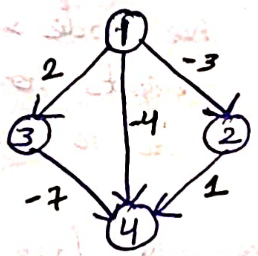
Exa: 4 nodes, 5 edges

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 4 | -1 |
| 1 | 3 | -2 |
| 3 | 4 | 7 |
| 1 | 4 | 4 |



- Consider edge values as negative of current values
- Apply Bellman - Ford

⇓



Edge distances:

$1 \to 2 = -3$

$1 \to 3 = 2$

$3 \to 4 = -7$

$2 \to 4 = 1$

$1 \to 4 = -4$

Value of Edge Relaxation:

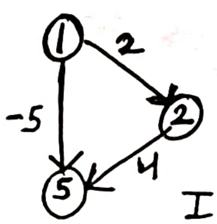
| Node: | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| dist arr: | 0 | ∞ | ∞ | ∞ |

Iterations :-

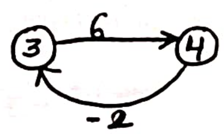
- Initial dist : [0, ∞, ∞, ∞]
- After 1st iteration : [0, -3, -2, -4]
- After 2nd iteration : [0, -3, 2, -5]
- After 3rd iteration : [0, -3, 2, -5]

Ans = -(-5) = 5

Edge Case:



I component



II component

The positive loop component (II) is different than the source → destination component. This suggest that if n is reachable through 1 ($1 \to n$), it doesn't mean that every node b/w 1 to n is also reachable through 1.

For exa: Node 5 is reachable from 1 (source), but 3 isn't (it is b/w 1 & 5)

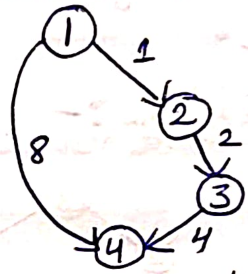
Approach :-

1. Run a DFS to check nodes available in component containing source & destination
2. Explore only those edges in Bellman Ford which share their component no. with 1 & n.

So, from previous example, consider only:

$1 \to 2$
$2 \to 5$
$1 \to 5$

2. Flight Discount

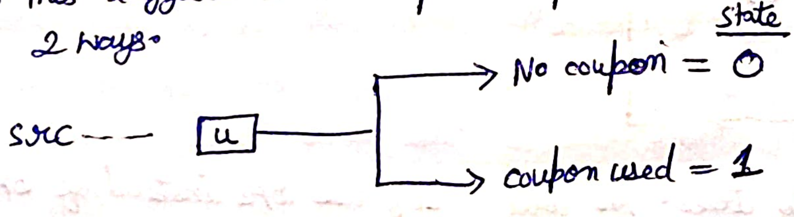



→ A standard Dijkstra algorithm won't work because the shortest path might not be the best one to apply the discount to.

For exa: we will get the shortest path b/w 1 & 4 (7). So we will apply coupon most expensive price which will affect our total as $7 - \{\frac{4}{2}\} = 7 - 2 = 5$

For reverse path: $1 \to 4$ the price $= 8$, but when we apply coupon, the overall price will be $\frac{8}{2} = 4$

∴ This suggests that all path exploration b/w 1 to n is required can be visited in 2 ways.




Now, we can run dijkstra's algorithm on this extended state-space graph which has 2N nodes. Where you are at node u:

1. If you are arrived in state 0, you can travel to neighbour v:
→ Without using the coupon: This is a transition to v in state 0.
→ By using the coupon: " " " " to v " state 1.

2. If you arrived in state 1 (coupon used) you can only travel to neighbour v without the coupon, remaining in state 1.

## 3. Max Weighted K-edge Path

1. ~~Initialize a 2D array,~~ Set max_weights [0][source] = 0
2. Loop from i = 0 to k-1.
3. Iterate through every edge (u, v) with weight w in the graph. (Inside loop)
4. If a path to u with i edges exist, update the path to v using i+1 edges:

   max_weights [i+1][v] = max (max_weights [i+1][v], max_weights [i][u] + w)

5. ans = max_weights [k] [destination]. check if this value is less than t. If it is, that's your ans. otherwise return -1.