September 22, 2025

GRAPH DRILL SESSION - 2

Ques 1. High Score

https://cses.fi/problemset/task/1673

You play a game consisting of n rooms and m tunnels. Your initial score is 0, and each tunnel increases your score by x where x may be both positive or negative. You may go through a tunnel several times.

Your task is to walk from room 1 to room n. What is the maximum score you can get?

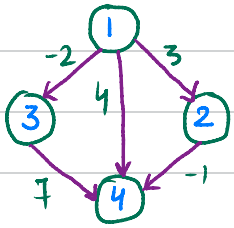
→ can use Bellman Ford.

In what case, we will get a larger positive value : whenever there is a presence of the positive cycle.
So whenever we get a component with positive cycle, we can simply iterate over it again & again to increase the overall score.
The idea of Bellman Ford is to compute shortest paths from a source, even when edges have negative weights.
If a -ve weight cycle is reachable from the source, then shortest paths are not well defined because you can keep decreasing the path cost indefinitely.
Use of Edge relaxation to help implement it.

Example :

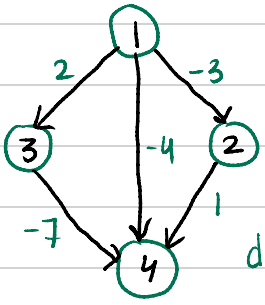
4 nodes, 5 edges

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



- Consider edge values as -ve of the current values.
- Then Apply Bellman Ford

(Use of Reverse Bellman Ford)



Values of Edge Relaxation :

Node :	1	2	3	4
dist. arr :	0	∞	∞	∞

Edge distances :
1 → 2 = -3
1 → 3 = 2
3 → 4 = -7
2 → 4 = 1
1 → 4 = -4

In 1st iteration, all the nodes connected to 1 will get updated (like 2, 3, 4)

Node :	1	2	3	4
dist array :	0	∞	∞	∞
1st iteration :	0	-3	2	-4

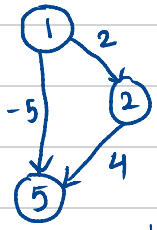
(refer edge dist. table)

Node :	1	2	3	4
dist array :	0	∞	∞	∞
1st iteration :	0	-3	2	-4
2nd iteration :	0	-3	2	-5
3rd :	0	-3	2	-5

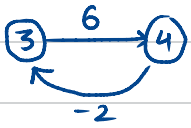
Ans = -(-5) = 5

AlgoZenith

Edge Case :



I component



II component

The positive loop component (II) is different than the source → destination component (I).

This suggest that if n is reachable through 1 (1 → n), it doesn't necessarily mean that every node between 1 to n is also reachable through 1. In this example, 5 is reachable from 1 (source), but 3 isn't (it is b/w 1 & 5).

Approach in such a case :

1. Run DFS to check the nodes available to the component containing source and destination. (there is always a path from 1 → n, given, so 1 & n will be part of the same component)

Explore only these founded edges in Bellman Ford, The ones which share their component no. with 1 & n.

So, in our previous case, we will look at the edges :

- 1 → 2
- 2 → 5
- 1 → 5

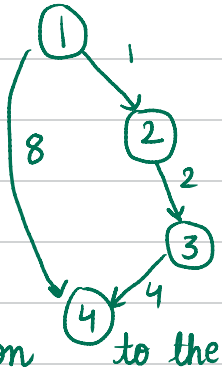
Ques 2. Flight Discount

https://cses.fi/alon/task/1195

Your task is to find a minimum-price flight route from Syrjälä to Metsälä. You have one discount coupon, using which you can halve the price of any single flight during the route. However, you can only use the coupon once.

When you use the discount coupon for a flight whose price is x, its price becomes $\lfloor x/2 \rfloor$ (it is rounded down to an integer).

Example :



If we apply Dijkstra directly here, we will get the shortest path b/w 1 → 4 (7)

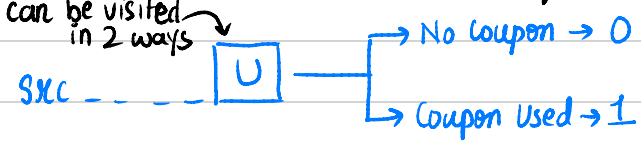
So, we will apply coupon to the most expensive price which will affect our total as $7 - \left(\frac{4}{2}\right) = 7 - 2 = 5$

For Reverse path , we will have the path 1 → 4 with the price = 8.

Here, when we apply coupon, our overall price will be $= \frac{8}{2} = 4$

So, we learn that overall price 2 which is 4 is better than shortest path's overall price which is 5.

∴ This suggests that all path exploration b/w 1 to n is required.



So, in order to visit any node, we have 2 states (0 & 1). We can judge whats the min price considering both states.

Starting from node U, we have to transit to node V.

State = 0 (No coupon) — U (took d distance till now) — e_0 / e_1

(State = 0) V: $d + e_1$ — V (State = 1): $d + \frac{e_1}{2} < D' \leftrightarrow 1$

If $d + e_1$ is smaller only then it will be considered

This should also only be considered if its strictly less.

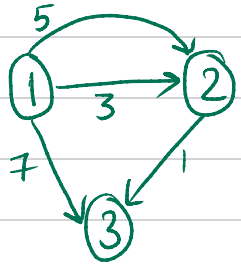
State = 1 (Coupon used) — U

Not possible with state = 0

V | 1 → $d + e$ (coupon is already utilised so we cant utilise it anymore)

Input:

```
3 4
1 2 3
2 3 1
1 3 7
2 1 5
```



Per node, we will have 2 states:

NODES	STATE 0 NO COUPON	STATE 1 COUPON USE
1	0	∞
2	∞	∞
3	∞	∞

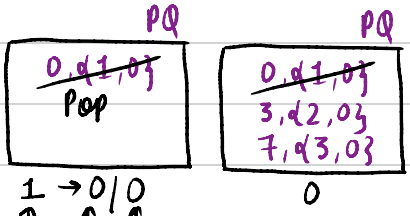
Initial form

Priority Queue:

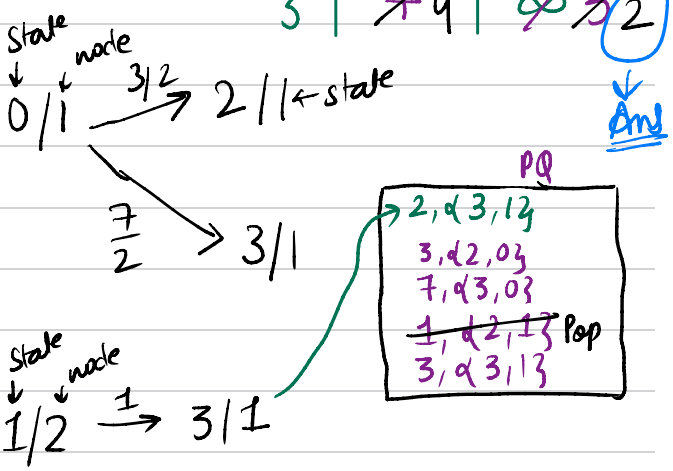
→ distance, with what dist a node can be visited

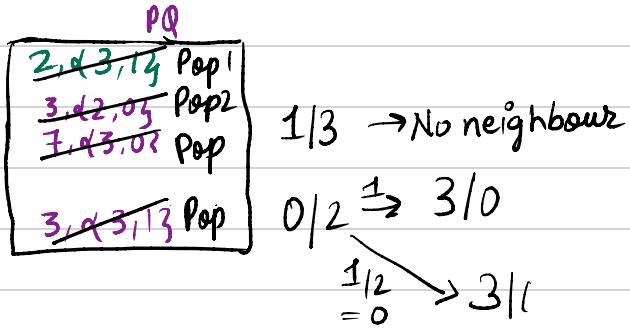
→ pair of node & associated state.

Cost, node, state



NODES	STATE 0 NO COUPON	STATE 1 COUPON USE
1	0	∞
2	3	~~∞~~ 1
3	~~7~~ 4	~~∞~~ ~~3~~ 2 (min)





T.C = $2V + 2E \log V$

$\Rightarrow (V+E) \log V$

* SPFA, Shortest Path Faster Algo

Utilise queue instead of PQ

Ques 3. Max. Weighted K-Edge Path

https://leetcode.com/problems/maximum-weighted-k-edge-path/

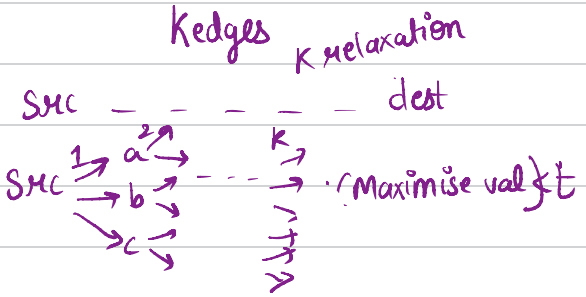
You are given an integer `n` and a **Directed Acyclic Graph (DAG)** with `n` nodes labeled from 0 to `n - 1`. This is represented by a 2D array `edges`, where `edges[i] = [u_i, v_i, w_i]` indicates a directed edge from node `u_i` to `v_i` with weight `w_i`.

You are also given two integers, `k` and `t`.

Your task is to determine the **maximum** possible sum of edge weights for any path in the graph such that:

- The path contains **exactly** `k` edges.
- The total sum of edge weights in the path is **strictly** less than `t`.

Return the **maximum** possible sum of weights for such a path. If no such path exists, return `-1`.

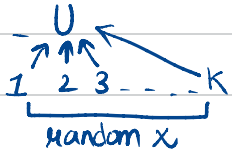


→ exact K edges needed b|w any source & any destination.

src ---- U ---- dest

From source to node U, how many edges are b|w src & U?

→ src ---- U



'x' no of paths to reach u.

0 ---- K

{a, b, c} → < t