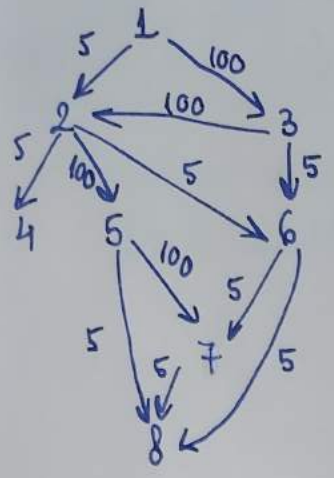
# Topological sorting using predecessor counting algorithm

Pb 4: we have a directed graph with 8 vertices and 12 edges



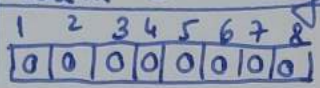
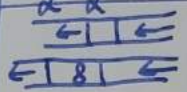
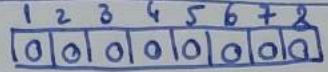
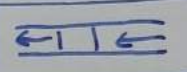

Input file:

```
8 12
1 2 5
1 3 100
2 4 5
2 5 100
2 6 5
3 2 100
3 6 5
5 7 100
5 8 5
6 7 5
6 8 5
7 8 5
```

topological sorting orders:

1, 3, 2, 5, 4, 6, 7, 8
1, 3, 2, 4, 6, 5, 7, 8
1, 3, 2, 5, 6, 4, 7, 8
1, 3, 2, 6, 4, 5, 7, 8
1, 3, 2, 6, 5, 4, 7, 8

| | x | y | count: Dictionary | q: queue | sorted: list |
|---|---|---|---|---|---|
| iteration | | | 1 2 3 4 5 6 7 8<br>0 2 1 1 1 2 2 3 | ← 1 ← | [ ] |
| iteration 1 | 1 | 2<br>3 | 1 2 3 4 5 6 7 8<br>0 1 0 1 1 2 2 3 | ← 3 ← | [1] |
| iteration 2 | 3 | 2<br>6 | 1 2 3 4 5 6 7 8<br>0 0 0 1 1 1 2 3 | ← 1 ←<br>← 2 ← | [1, 3] |
| iteration 3 | 2 | 4<br>5<br>6 | 1 2 3 4 5 6 7 8<br>0 0 0 0 0 0 2 3 | ← ←<br>← 4 ←<br>← 4 5 ←<br>← 4 5 6 ← | [1, 3, 2] |
| iteration 4 | 4 | | 1 2 3 4 5 6 7 8<br>0 0 0 0 0 0 2 3 | ← 5 6 ← | [1, 3, 2, 4] |
| iteration 5 | 5 | 7<br>8 | 1 2 3 4 5 6 7 8<br>0 0 0 0 0 0 1 2 | ← 6 ← | [1, 3, 2, 4, 5] |
| iteration 6 | 6 | 7<br>8 | 1 2 3 4 5 6 7 8<br>0 0 0 0 0 0 0 1 | ← ←<br>← 7 ← | [1, 3, 2, 4, 5, 6] |

| | x | y | count : Dictionary | q : queue | sorted : list |
|---|---|---|---|---|---|
| iteration 7 | 7 | 8 |  |  | [1, 3, 2, 4, 5, 6, 7] |
| iteration 8 | 8 | |  |  | [1, 3, 2, 4, 5, 6, 7] |

The algorithm is over, the graph is a DAG having size of (sorted) = 8

Highest cost walk between 2 given vertices:

starting vertex: 1

ending vertex: 7

| | u | v | dist : Dictionary | next : Dictionary |
|---|---|---|---|---|
| initialization | | | 1 2 8<br>0 \| -1000000 \| .... \| -1000000 | |
| iteration 1 | 1 | 2 | 1 2 3 4 5 6 7 8<br>0 \| 5 \| \| \| \| \| \| \| | 1 2 3 4 5 6 7 8<br>\| 1 \| \| \| \| \| \| \| |
| | | 3 | 0 \| 5 \| 100 \| \| \| \| \| \| | \| 1 \| 1 \| \| \| \| \| |
| iteration 2 | 3 | 2 | 1 2 3 4 5 6 7 8<br>0 \| 200 \| 100 \| \| \| \| \| \| | 1 2 3 4 5 6 7 8<br>\| 3 \| 1 \| \| \| \| \| |
| | | 6 | 0 \| 200 \| 100 \| \| \| 105 \| \| | 1 2 3 4 5 6 7 8<br>\| 3 \| 1 \| \| \| 3 \| \| |
| iteration 3 | 2 | 4 | 1 2 3 4 5 6 7 8<br>0 \| 200 \| 100 \| 205 \| \| 105 \| \| | 1 2 3 4 5 6 7 8<br>\| 3 \| 1 \| 2 \| \| 3 \| \| |
| | | 5 | 0 \| 200 \| 100 \| 205 \| 300 \| 105 \| \| | \| 3 \| 1 \| 2 \| 2 \| 3 \| \| |
| | | 6 | 0 \| 200 \| 100 \| 205 \| 300 \| 205 \| \| | \| 3 \| 1 \| 2 \| 2 \| 2 \| \| |
| iteration 4 | 4 | - | — | — |
| iteration 5 | 5 | 7 | 1 2 3 4 5 6 7 8<br>0 \| 200 \| 100 \| 205 \| 300 \| 205 \| 400 \| | 1 2 3 4 5 6 7 8<br>\| 3 \| 1 \| 2 \| 2 \| 2 \| 5 \| |
| | | 8 | 0 \| 200 \| 100 \| 205 \| 300 \| 205 \| 400 \| 305 \| | \| 3 \| 1 \| 2 \| 2 \| 2 \| 5 \| 5 \| |
| iteration 6 | 6 | 7 | — | — |
| | | 8 | — | — |
| iteration 7 | 7 | | ⇒ stop u = 7 (ending vertex)<br>the highest walk is dist[u] = 400<br>and using next, we can build the walk:<br>1 → 3 → 2 → 5 → 7 | |