NOI 2013 – solutions to contest tasks.

School of Computing
National University of Singapore

Chang Ee-Chien

16 March 2013

Scientific Committee

<u>Chang</u> Ee-Chien (chair)
<u>Chin</u> Zhan Xiong
Steven <u>Halim</u>
<u>Harta</u> Wijaya
Martin <u>Henz</u>
Raymond <u>Kang</u>
<u>Ooi</u> Wei Tsang
Frank <u>Stephan</u>
Wing-Kin <u>Sung</u>
<u>Trinh</u> Tuan Phuong

Changes from previous years

1) Many subtasks. Constrains on each subtask explicitly stated.

2) Realtime feedback. "Realistic" input/output samples provided.

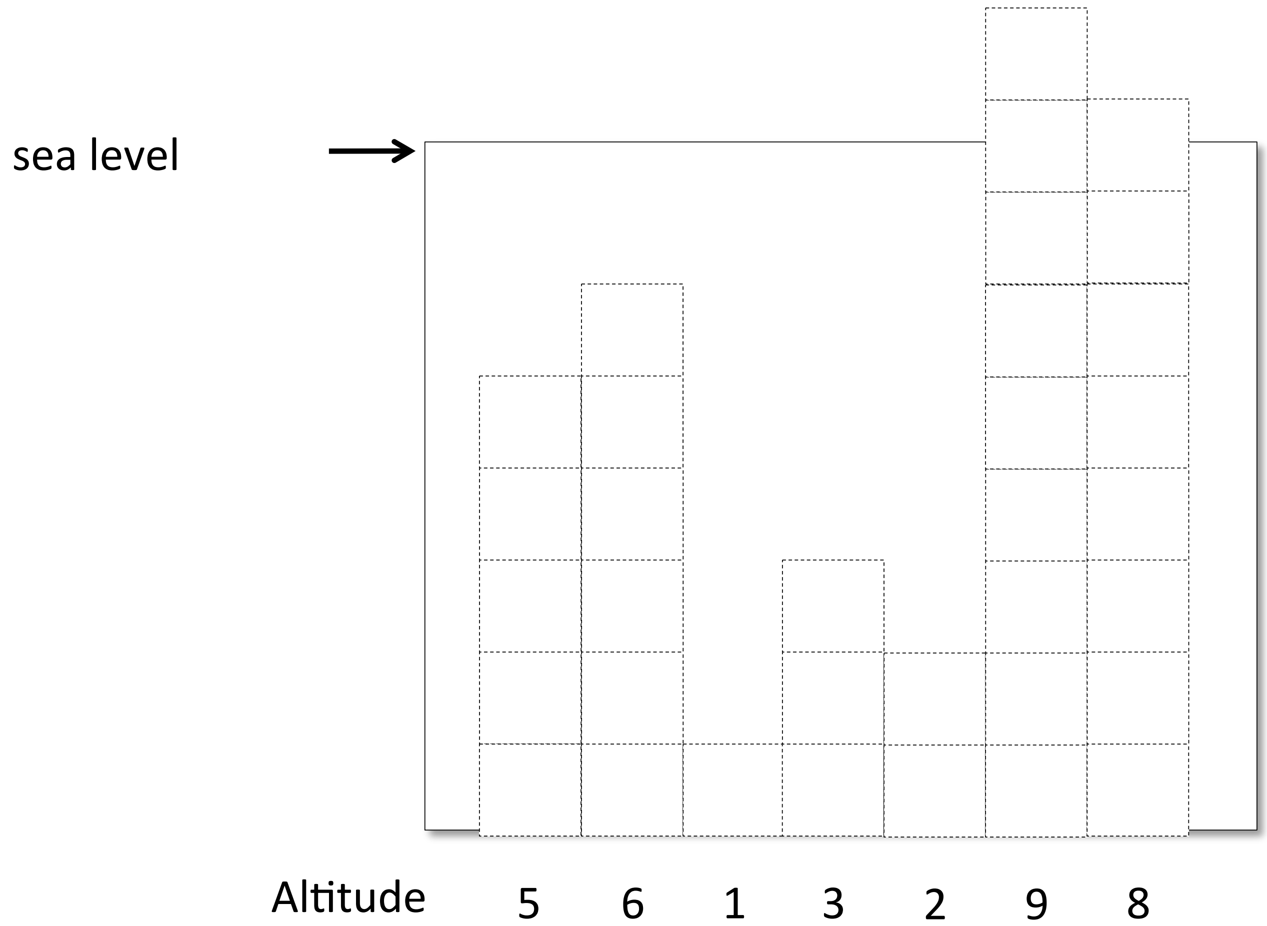
TASK 1: GLOBAL WARMING
TASK 2: TRUCKING DIESEL*
TASK 3: FERRIES

* Task 2 has a mistake and its solution will not be presented here.

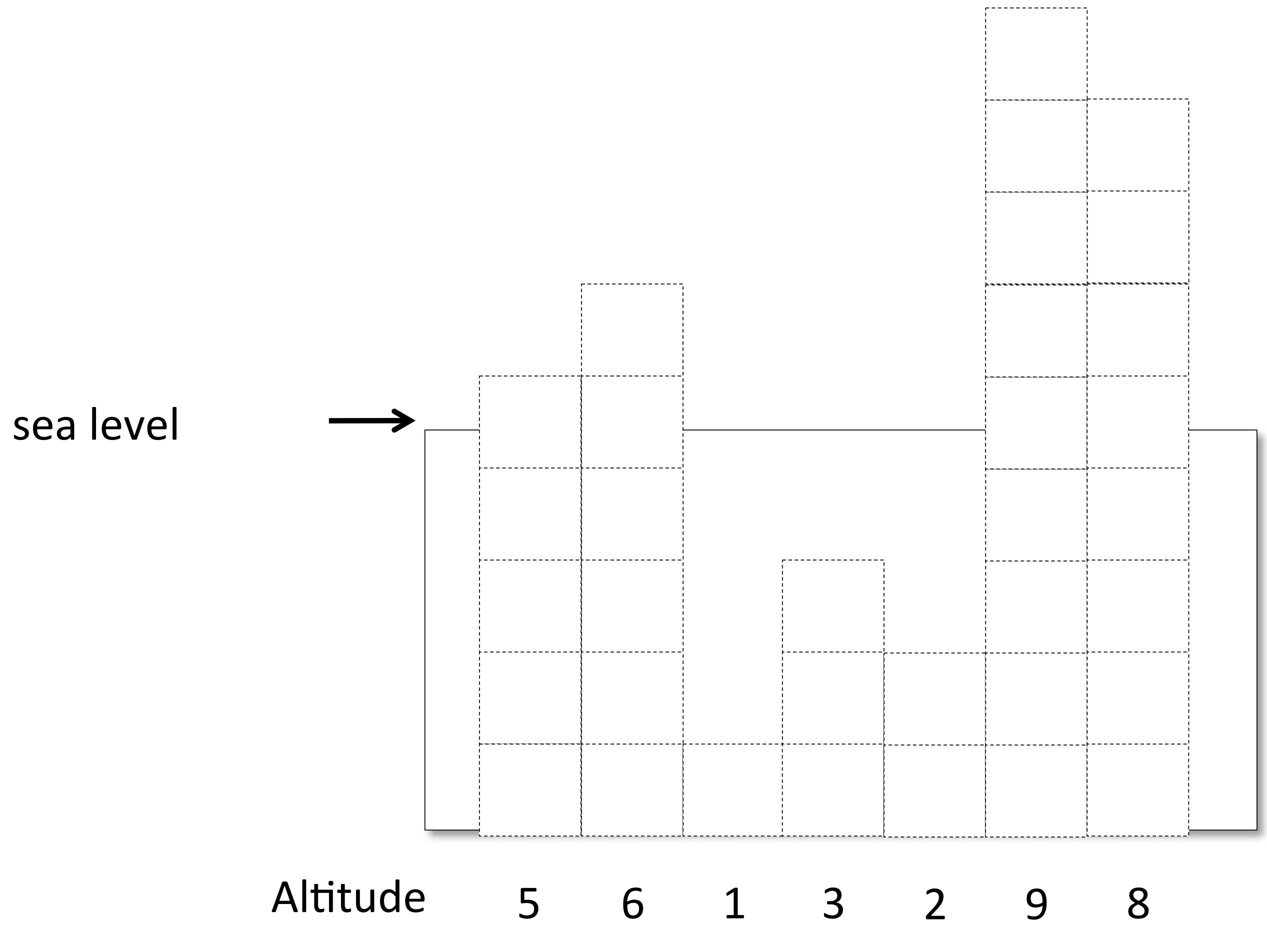
Task 1: GLOBAL WARMING

Problem

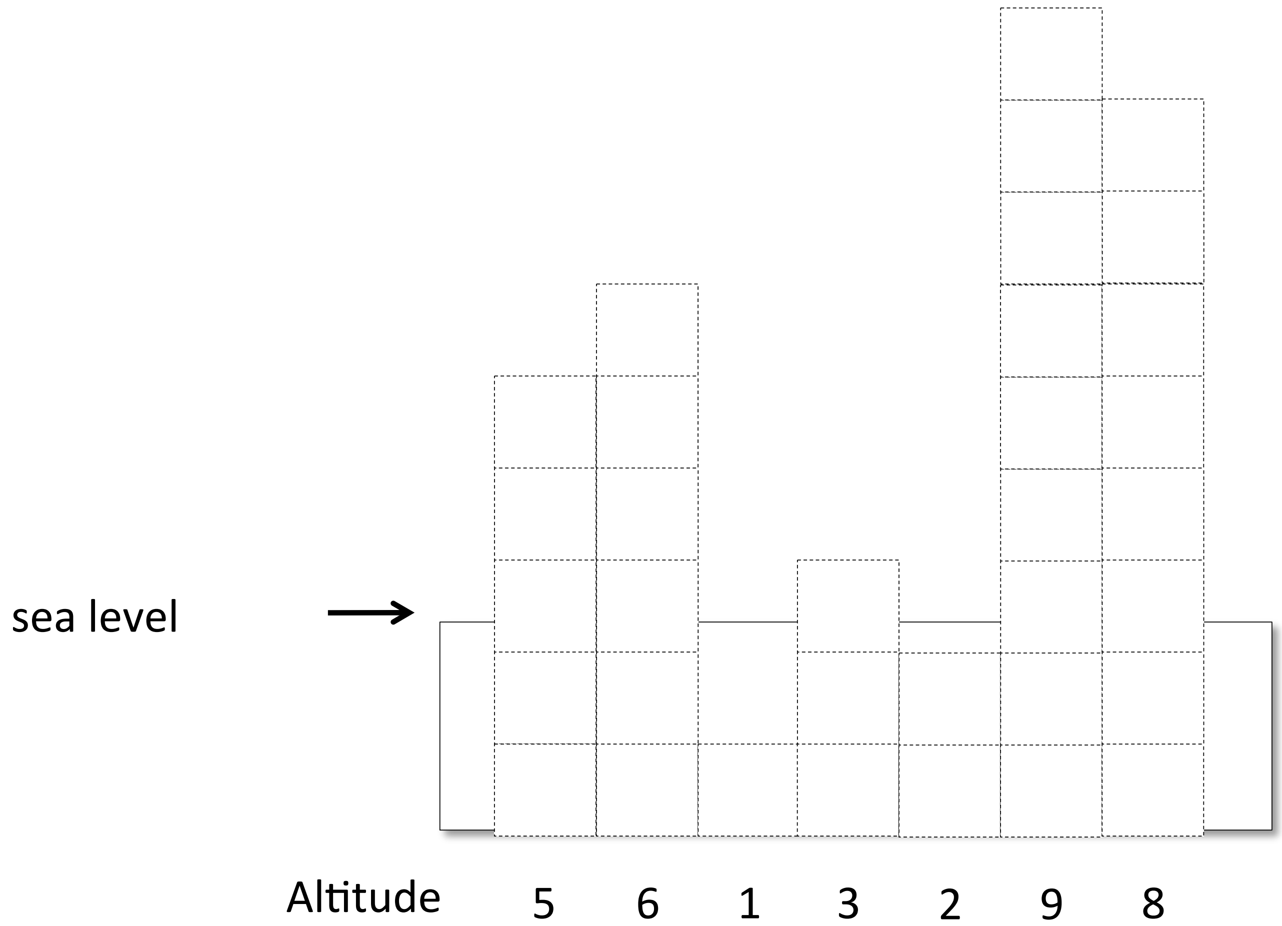
Given the altitude at each location, compute the max possible number of islands.



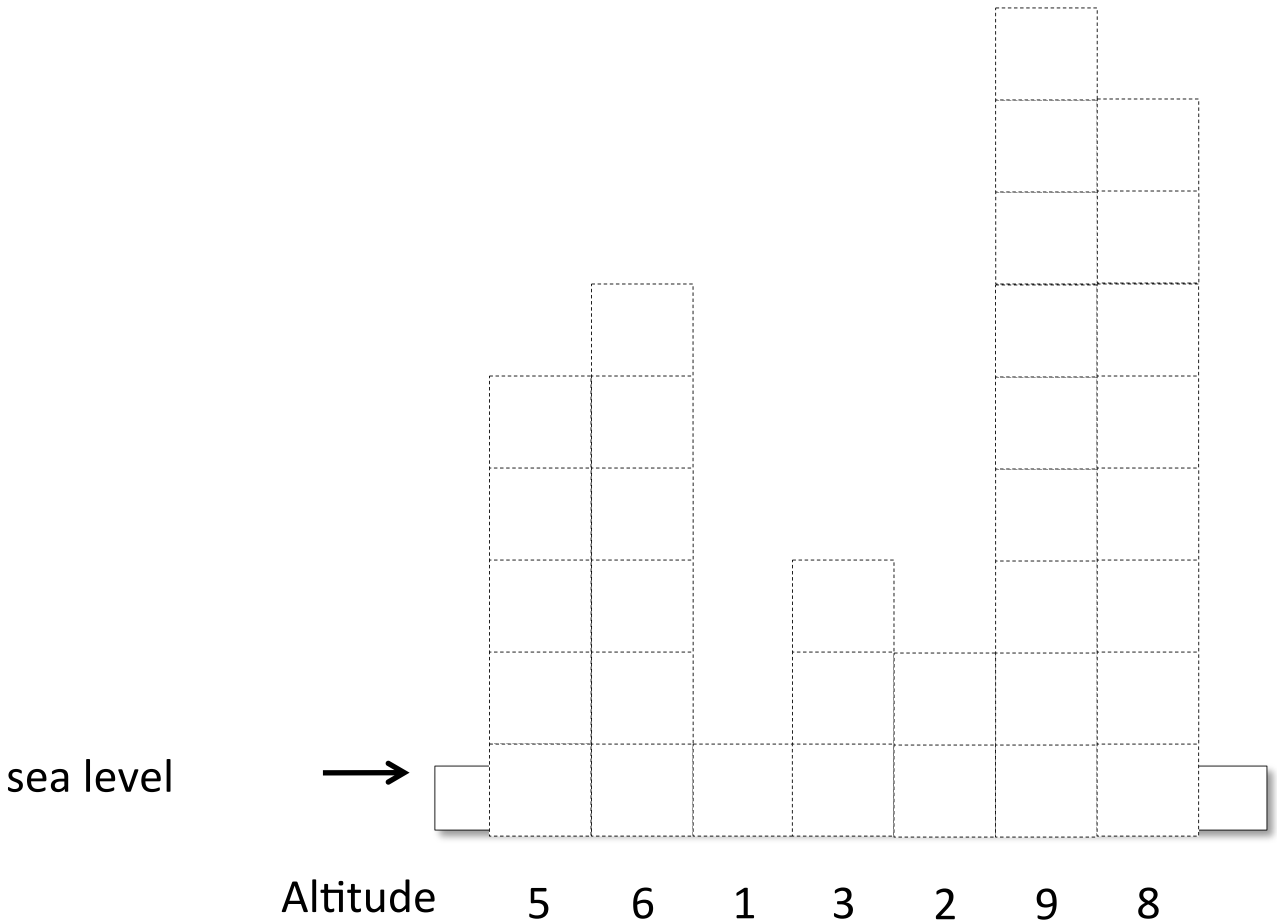
Problem



Problem



Problem



Partial solution

Try all sea levels, starting from 0.5, 1.5, 2.5, 3.5, …., (highest_point - 0.5).

At each sea level, scan the whole array to count the number of islands.

Running time O(highest_point × n)

Able to solve subtask 2.

Partial solution

Try sea levels at just below or above altitude at each ***location***. Trying

h_n $h_n + 0.5$.

Running time O(n^2)

Able to solve subtask 1.

Partial solution

Try sea levels at just below or above altitude at each ***location***. Trying

h_1 - 0.5, h_2 - 0.5, ..., h_n - 0.5 .

Running time O(n^2)

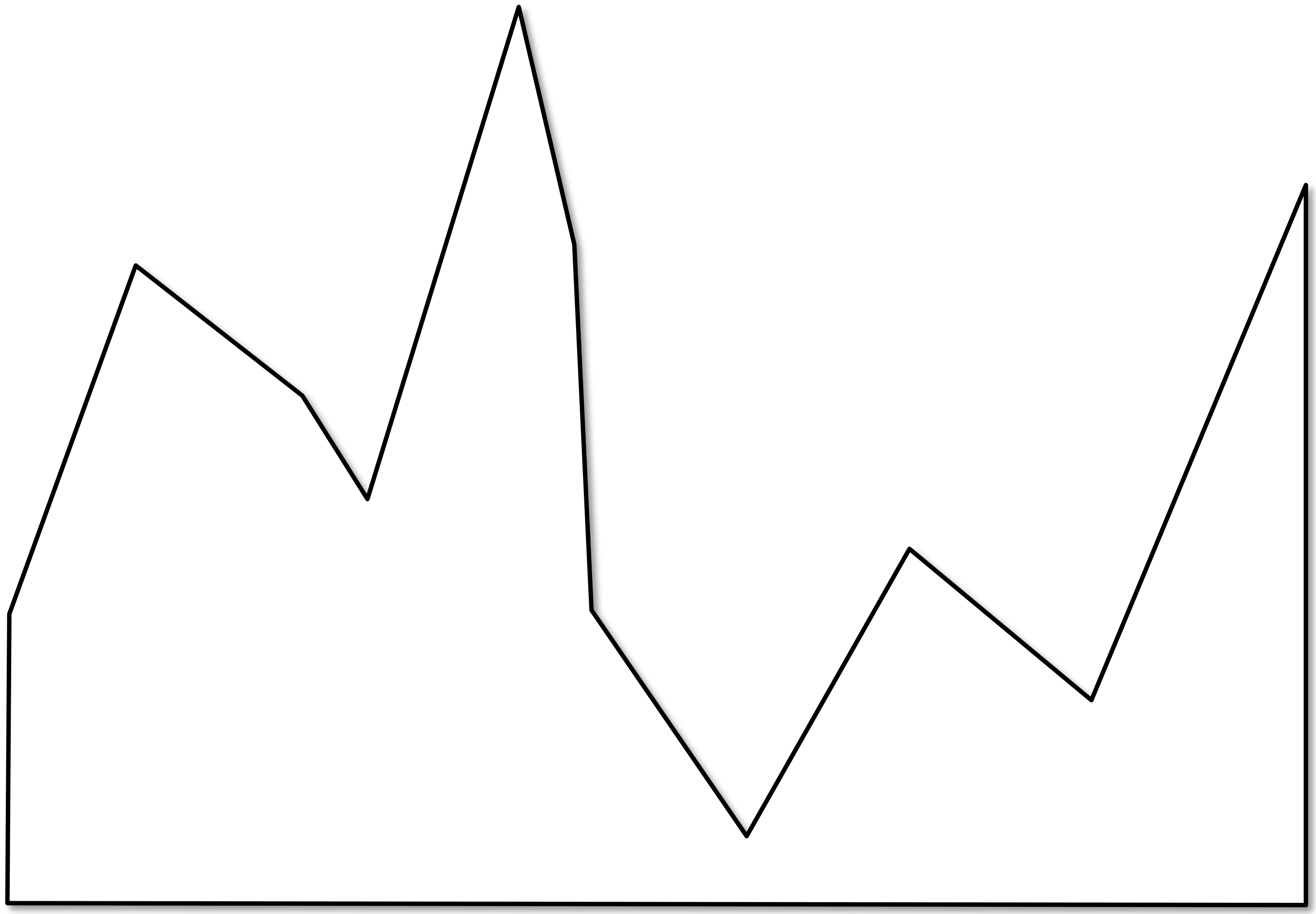
Able to solve subtask 1.

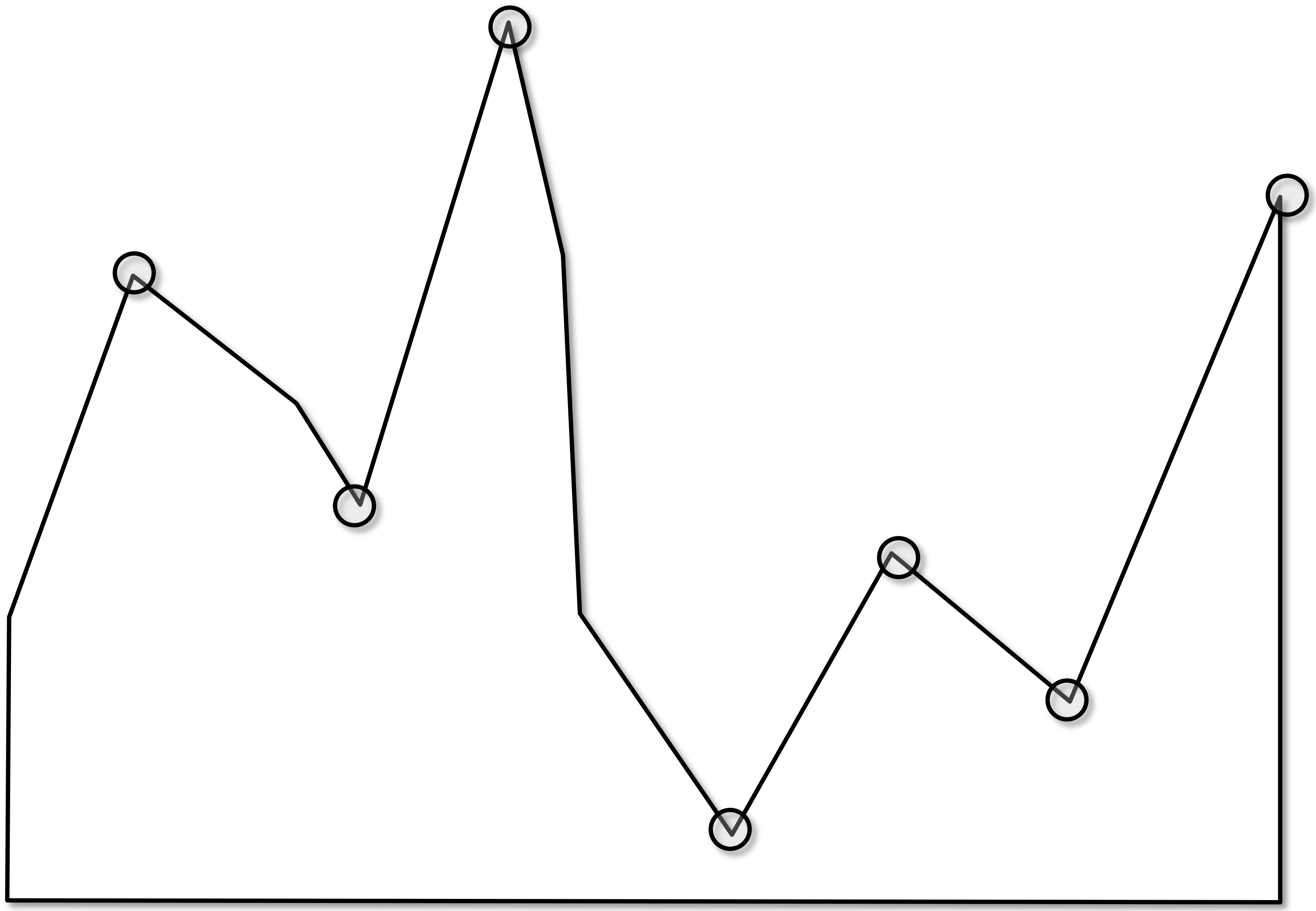
Solution (idea):

- Flood the world; lower the sea level and count the number of island.

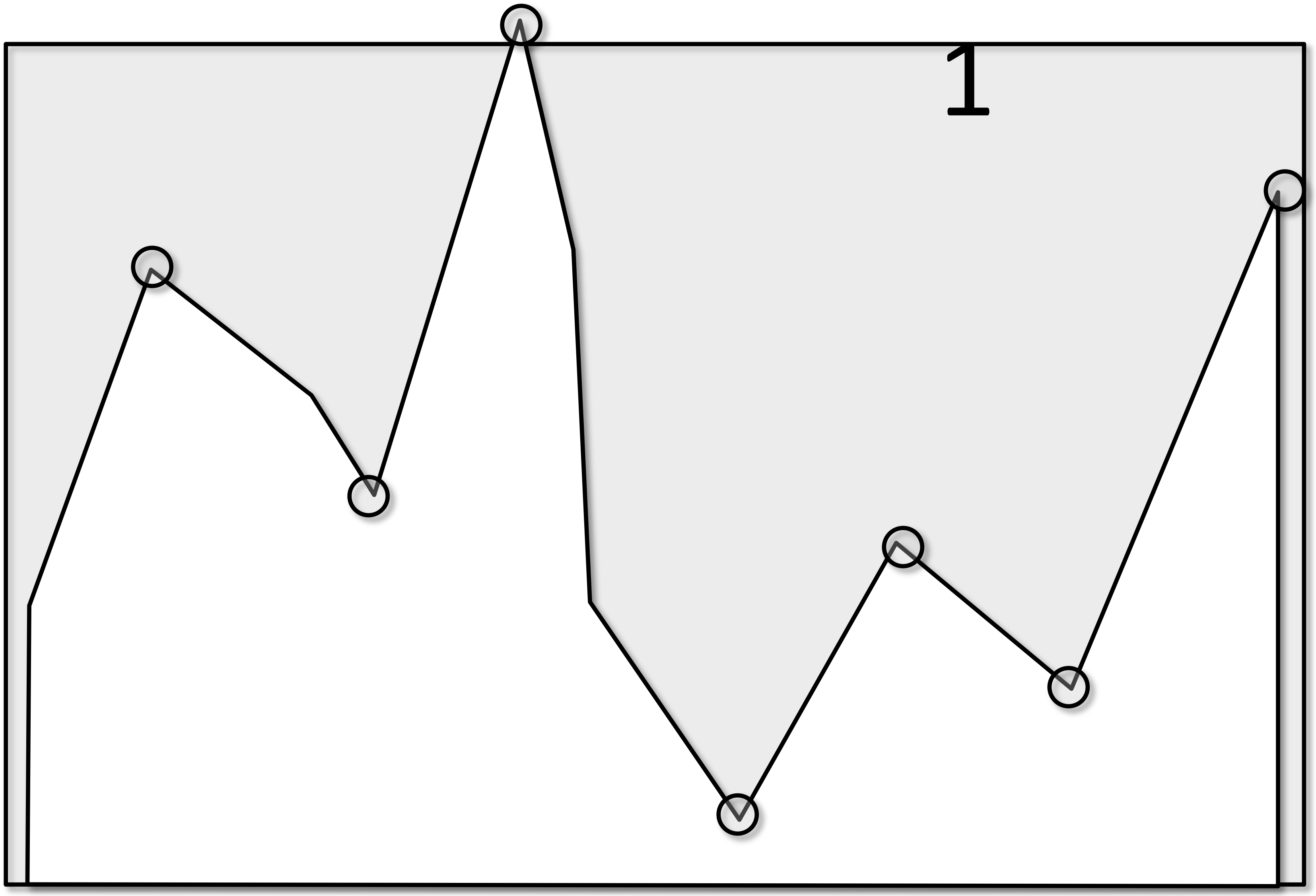
- Do not lower the sea level slowly, jump to the local peaks or local valleys.

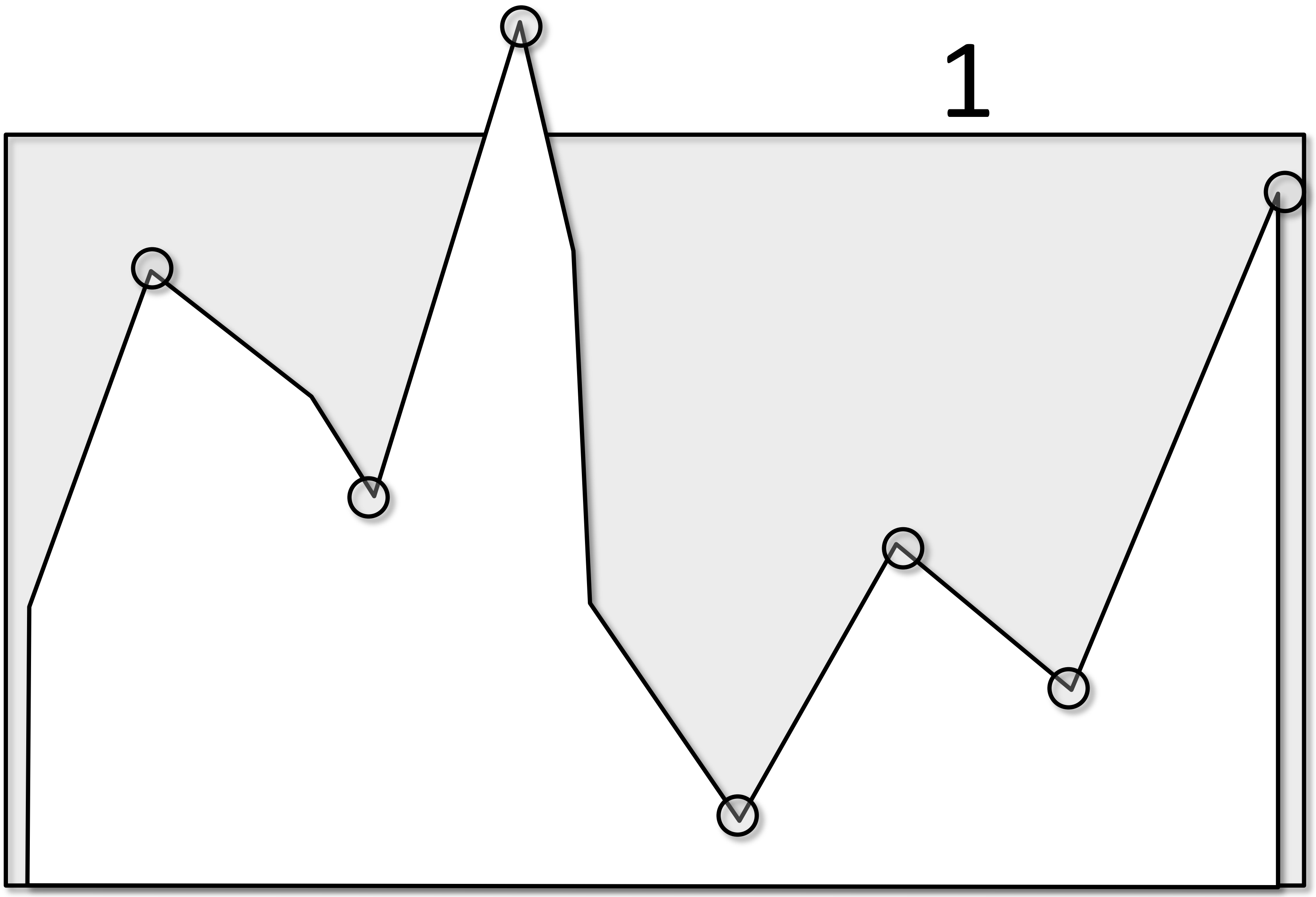
- Do not scan the whole array to count the islands. At each new sealevel, the number of islands can be differed by at most one.



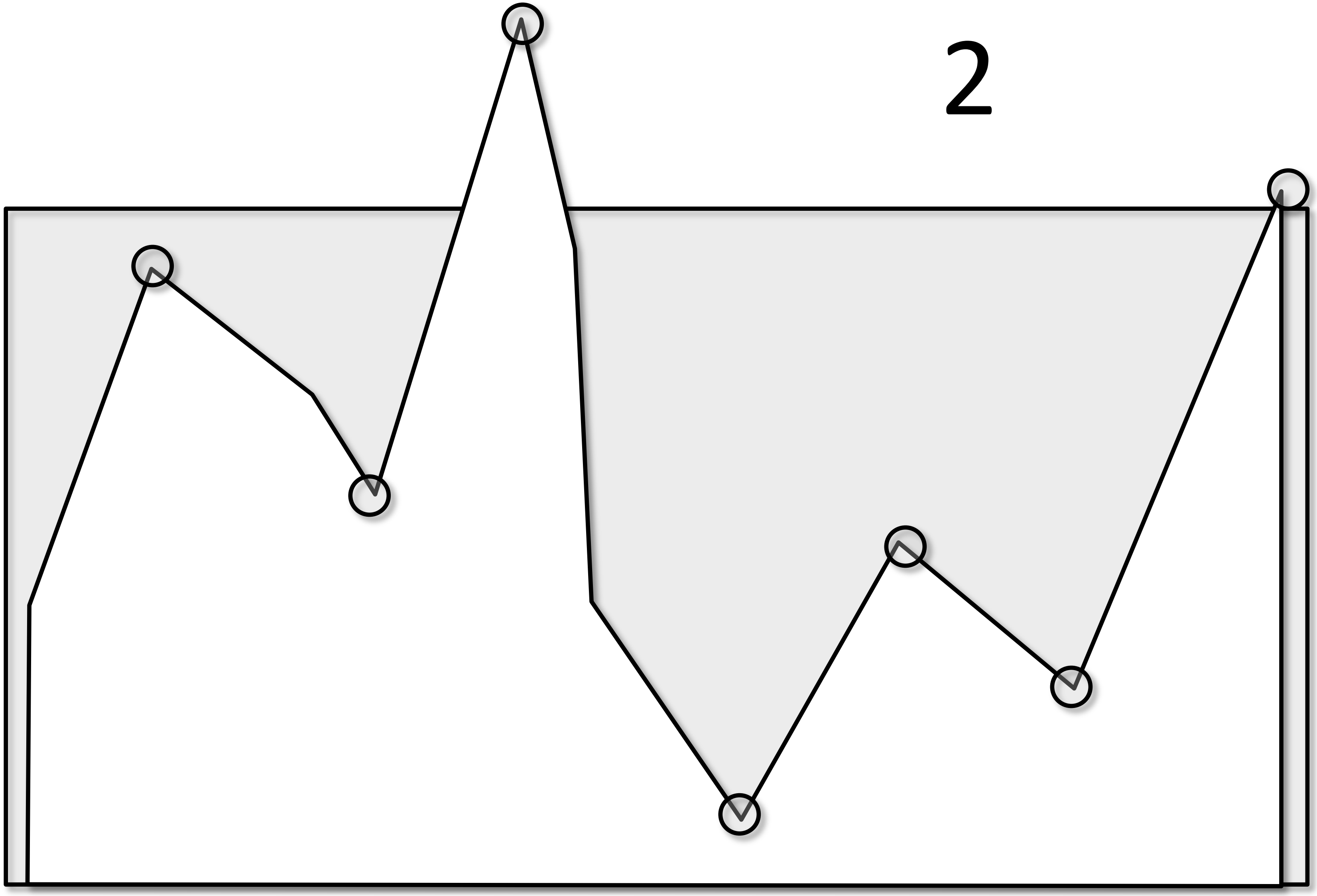


encounter peak (+)

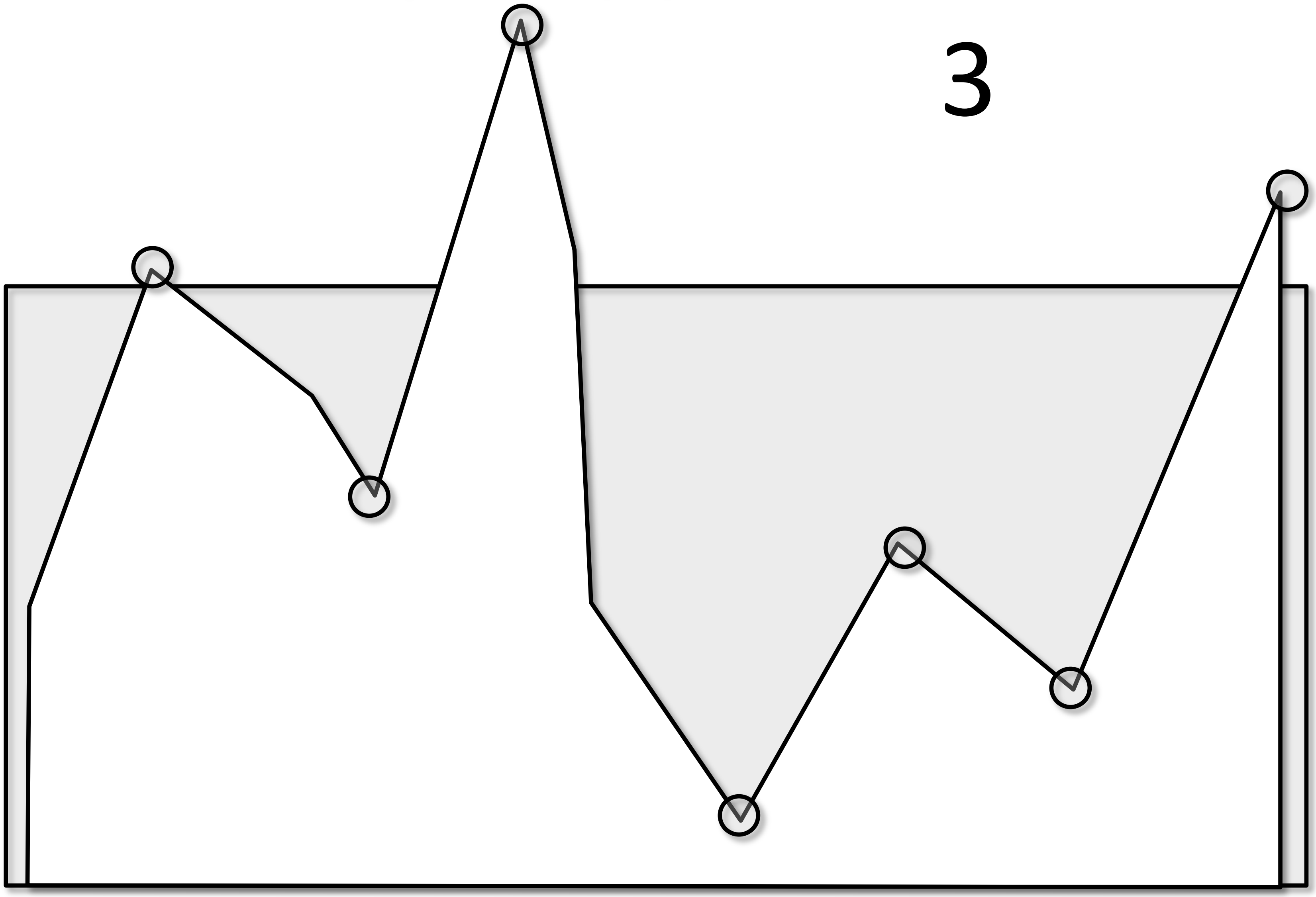



1

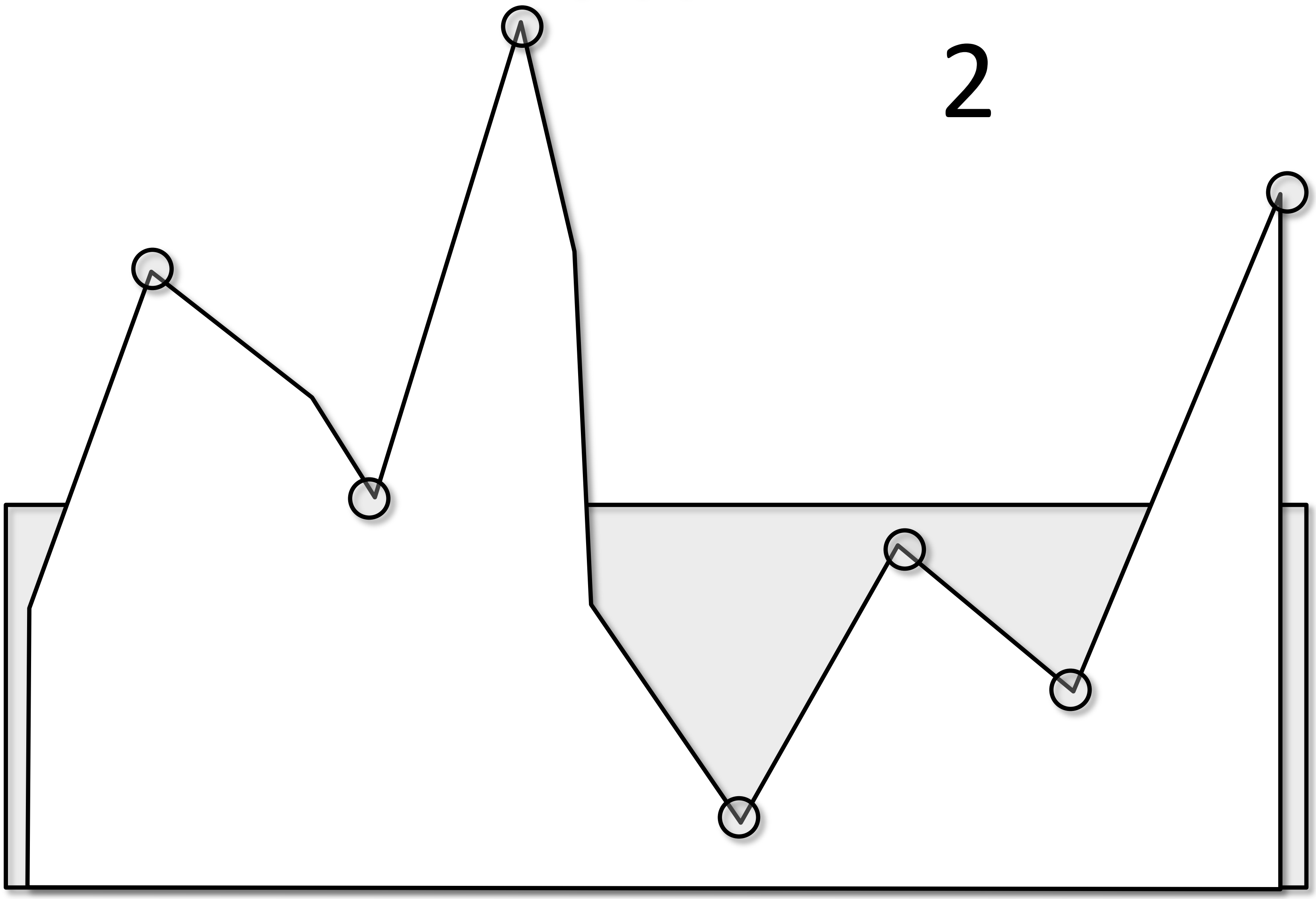
encounter peak (+)



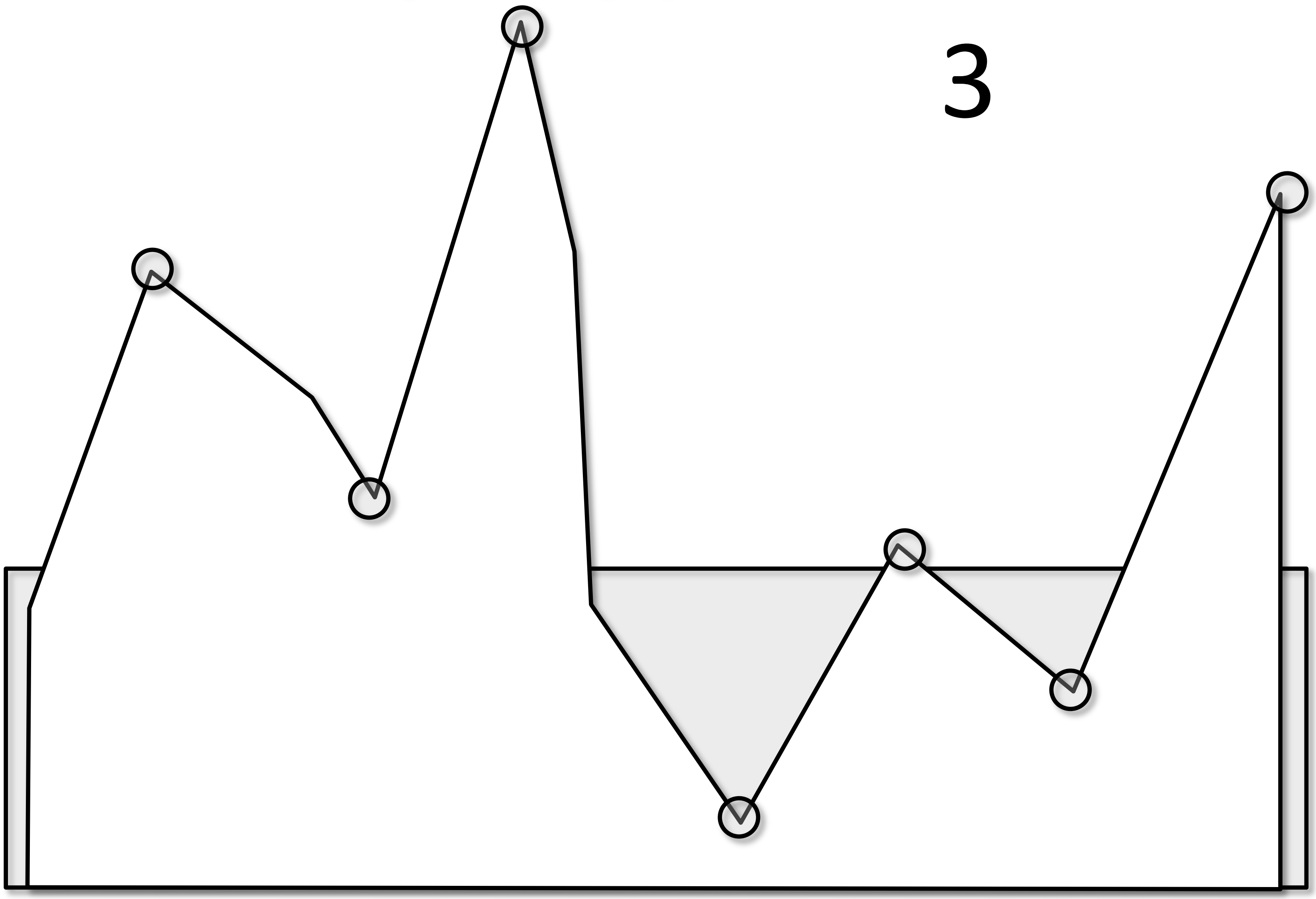
encounter peak (+)

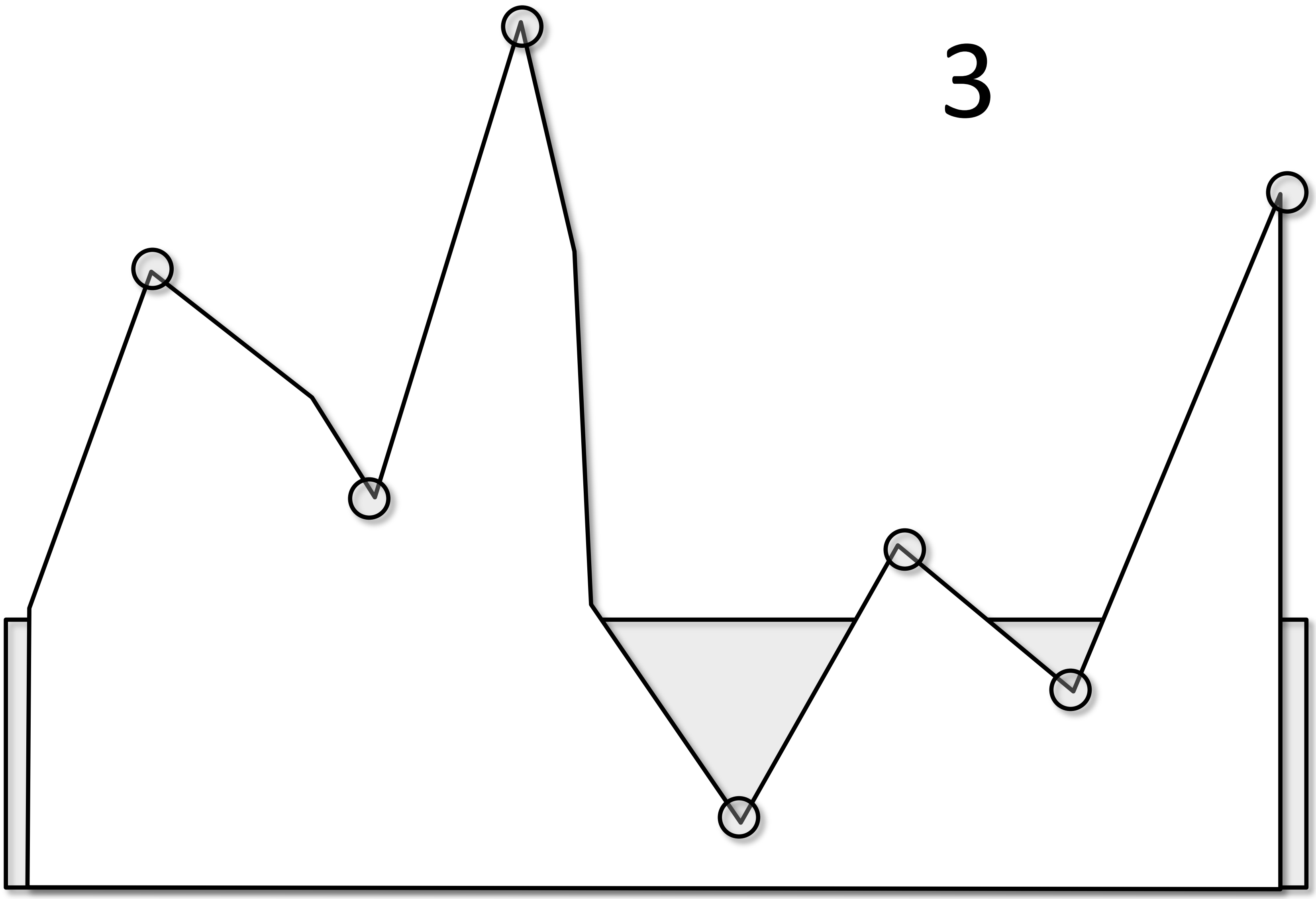


encounter valley (-)

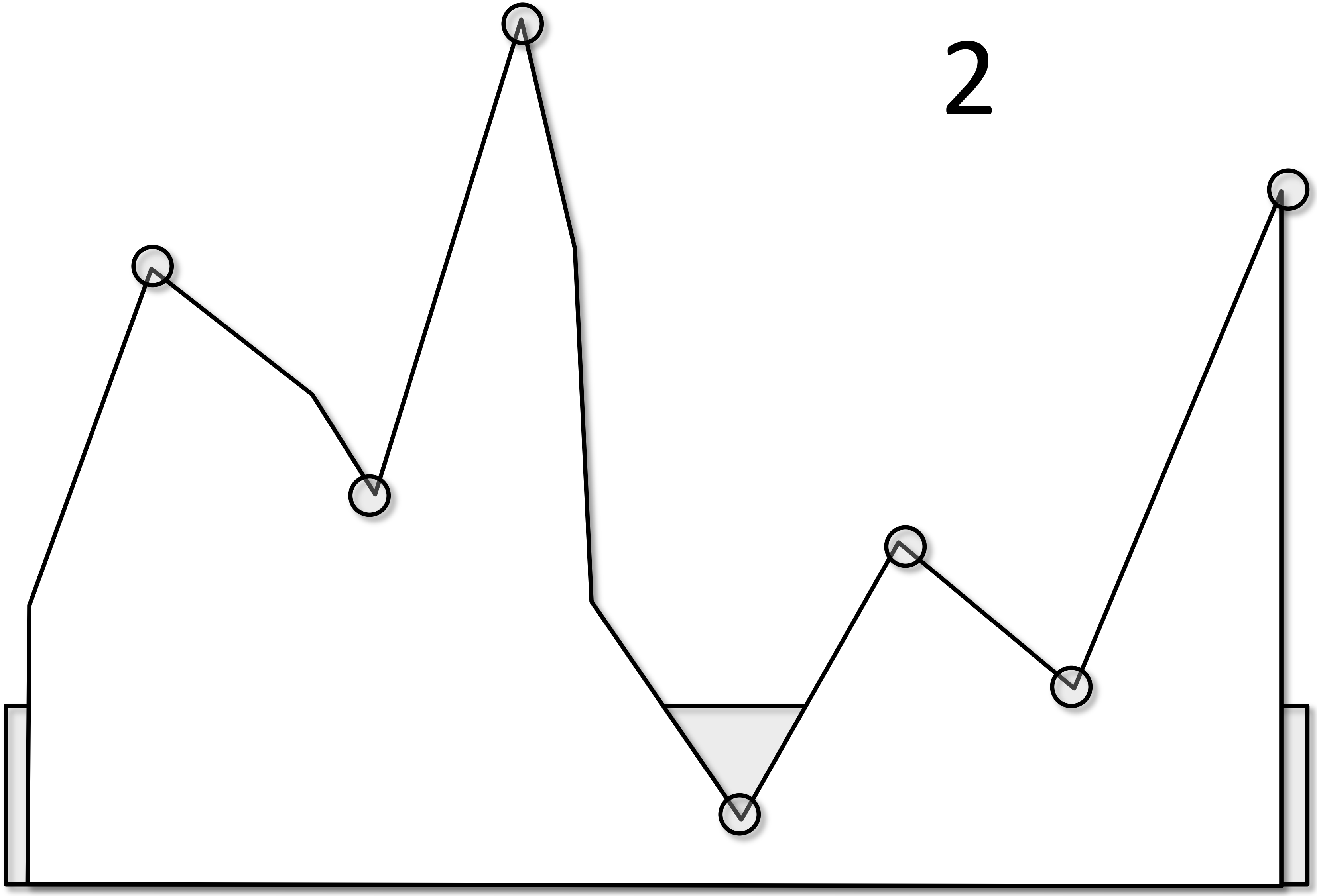


encounter peak (+)

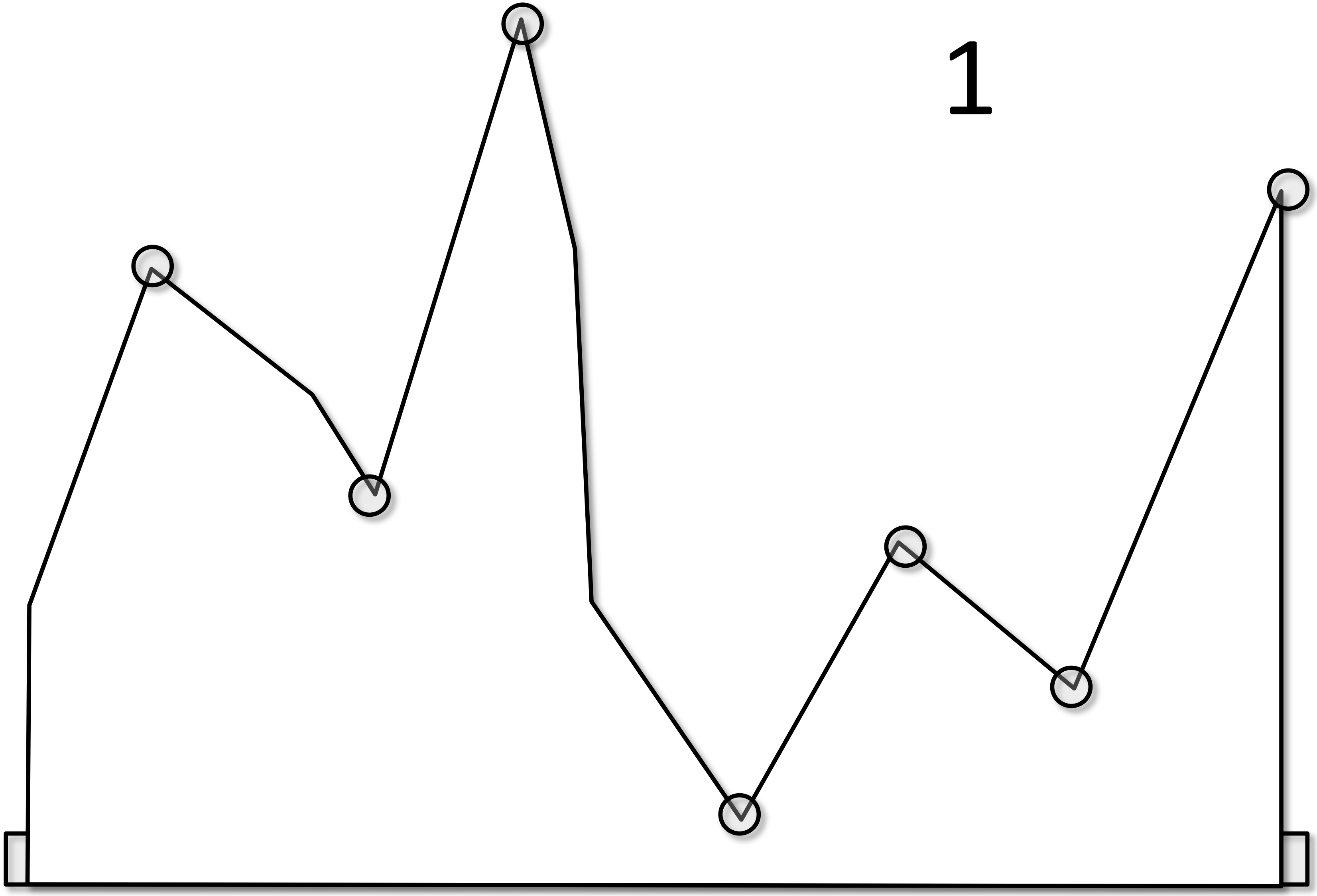



3

encounter valley (-)



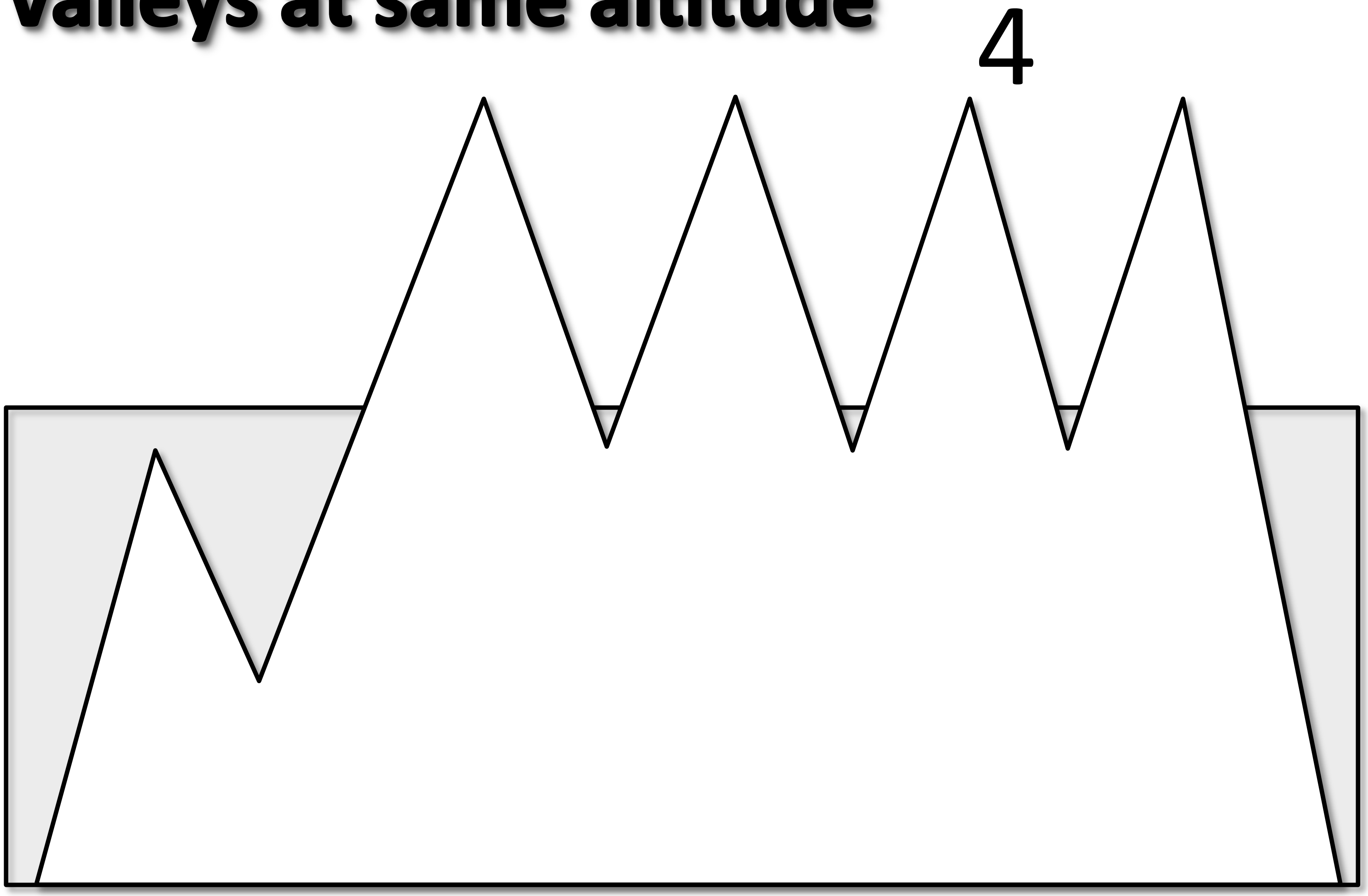
encounter valley (-)

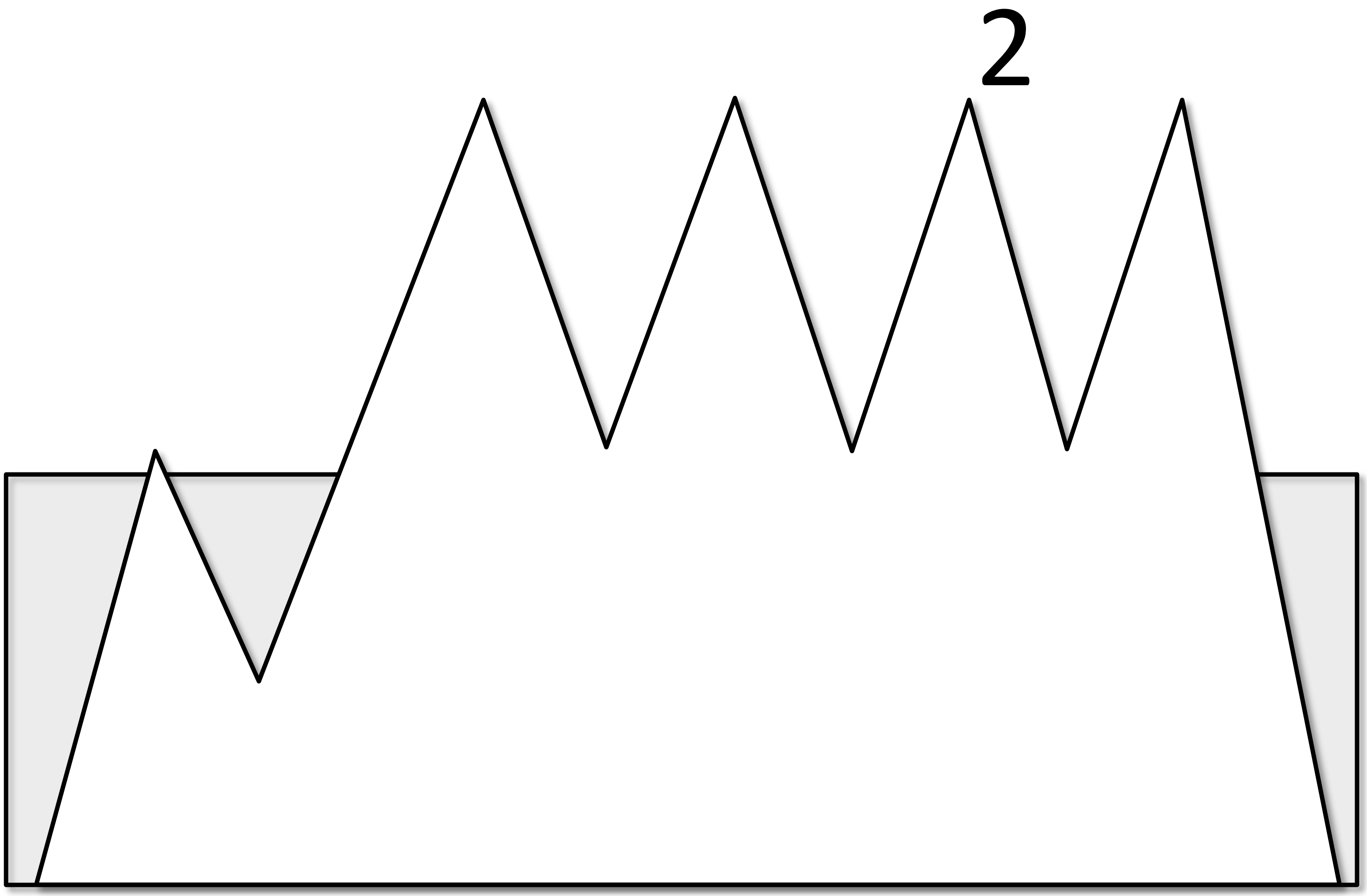


Running time O (n log n)

Able to solve input all subtasks.
Unable to solve subtasks 1,2,5 if handle a special case wrongly.

Special cases: Handling peaks and valleys at same altitude

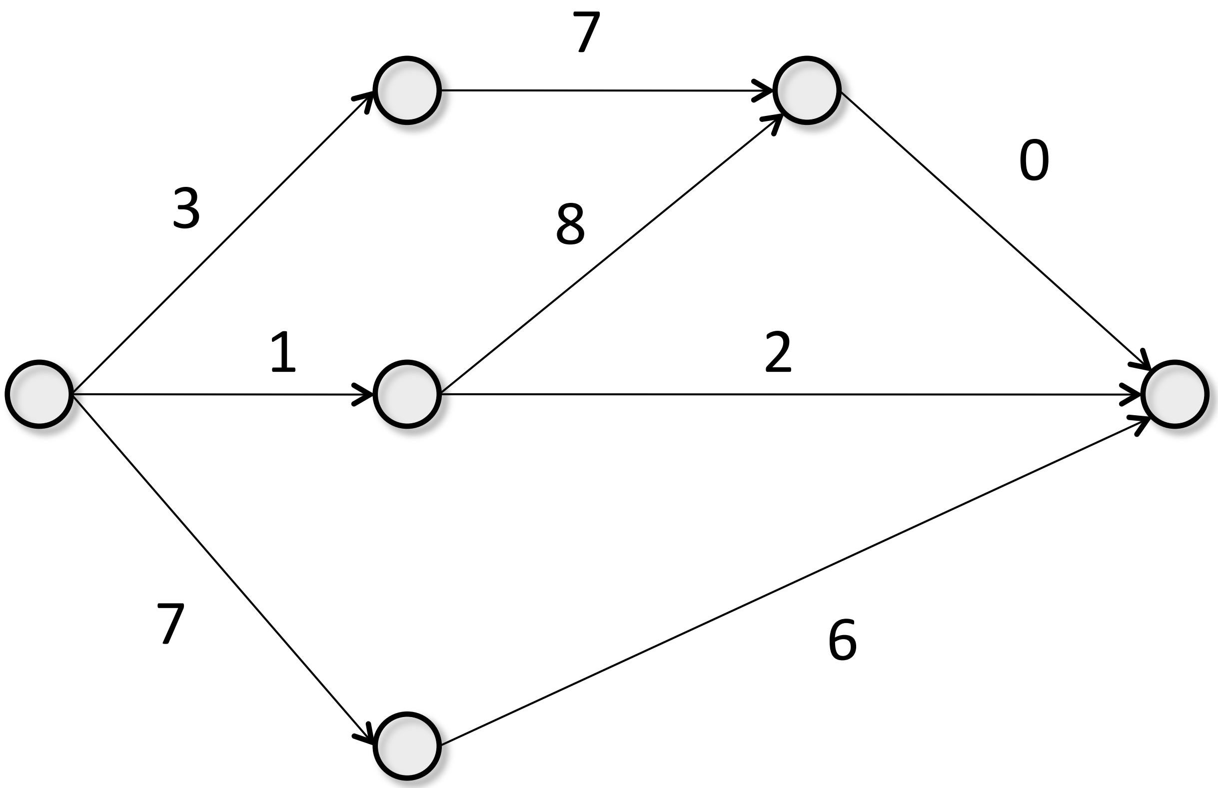



2

Task 3: FERRIES

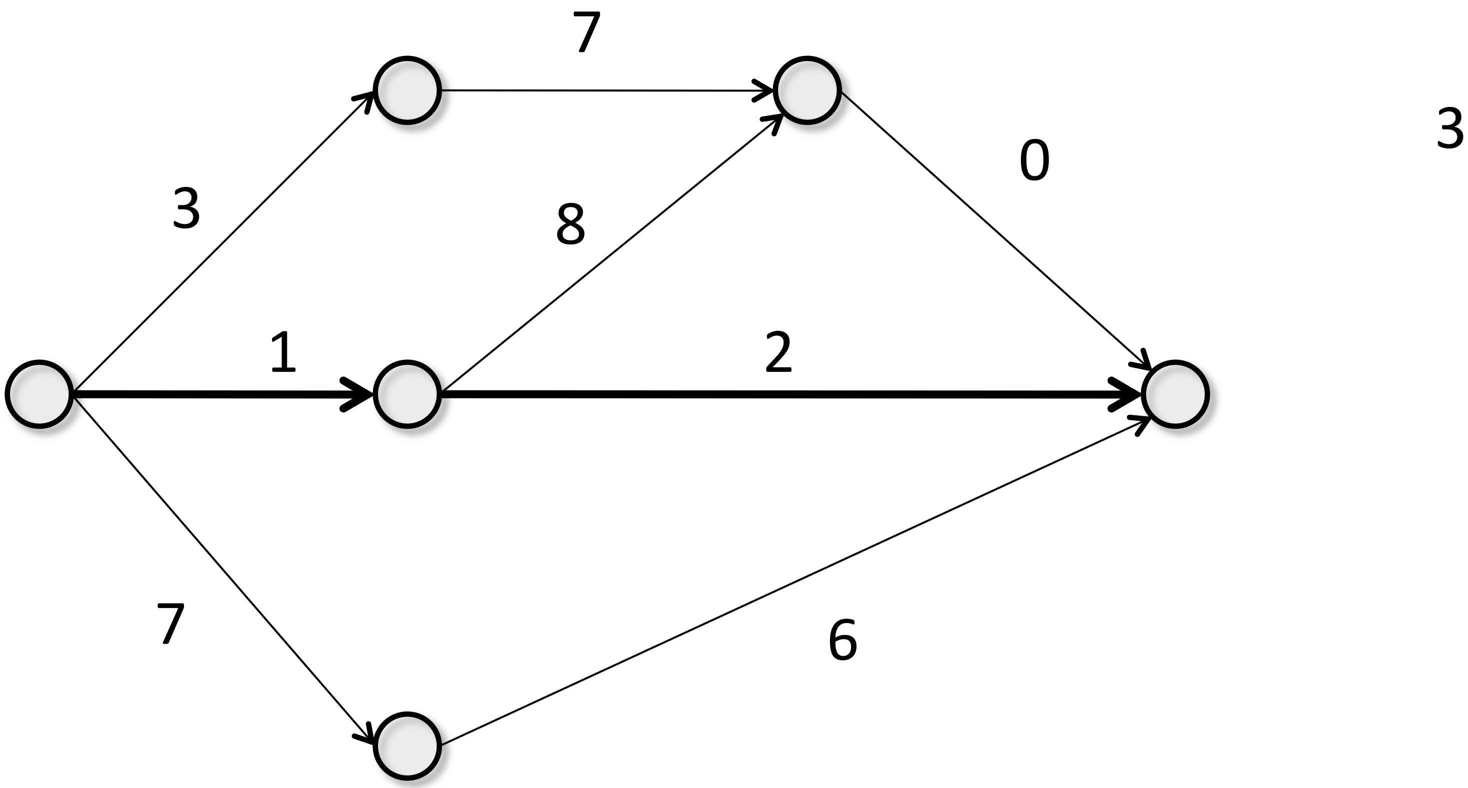
Problem

Shortest path (in term of cost).



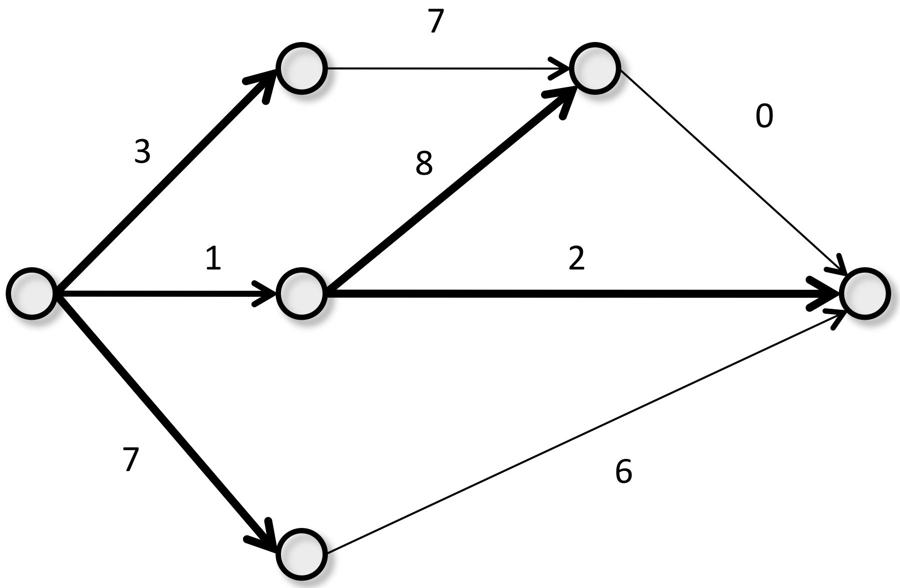
Problem

Shortest path (in term of cost).



Problem

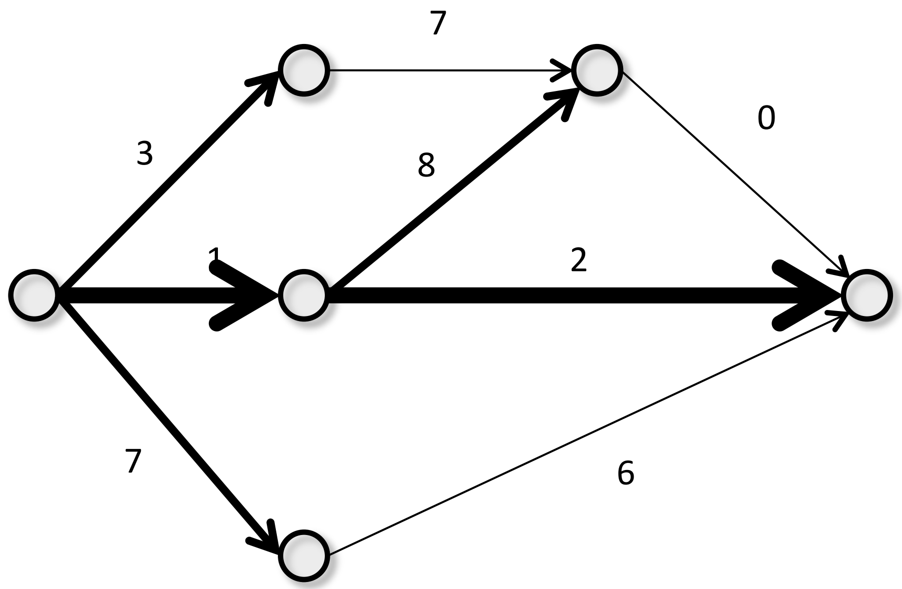
The weights of outgoing edges may be swapped. For each combination of swap, we can determine the cost of shortest path. Among all combinations, what is the maximum?



Problem

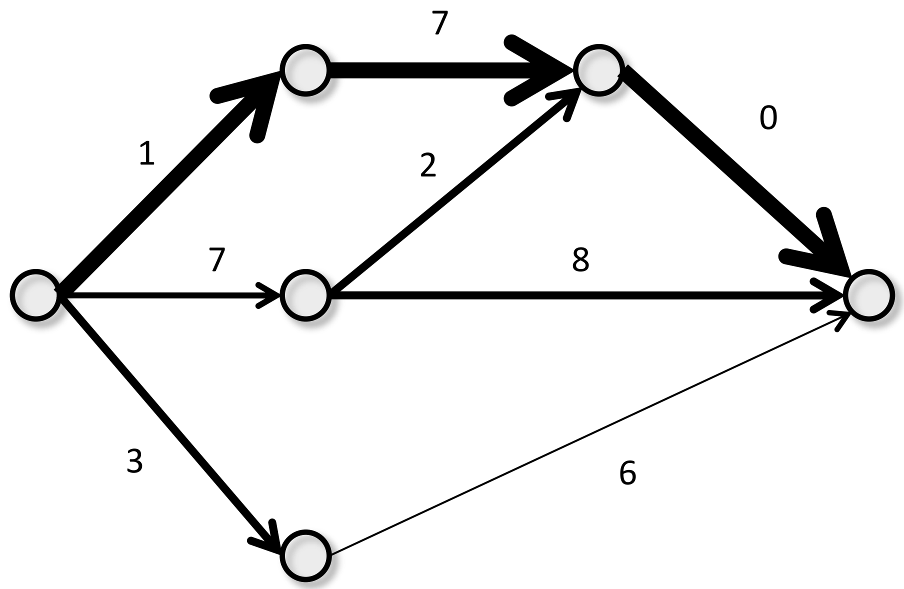
Shortest path (in term of cost).

combination 1



shortest : 3

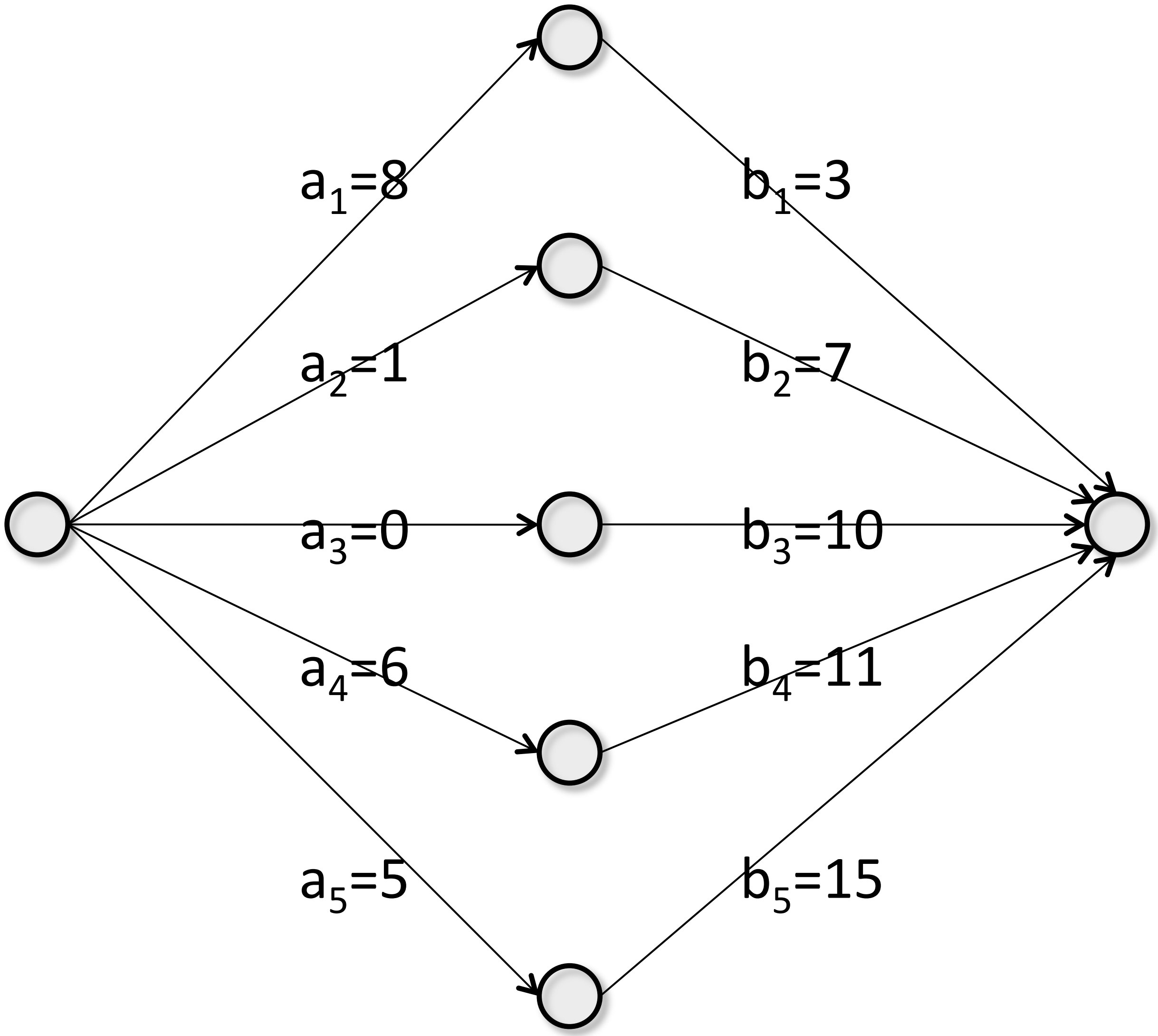
combination 2

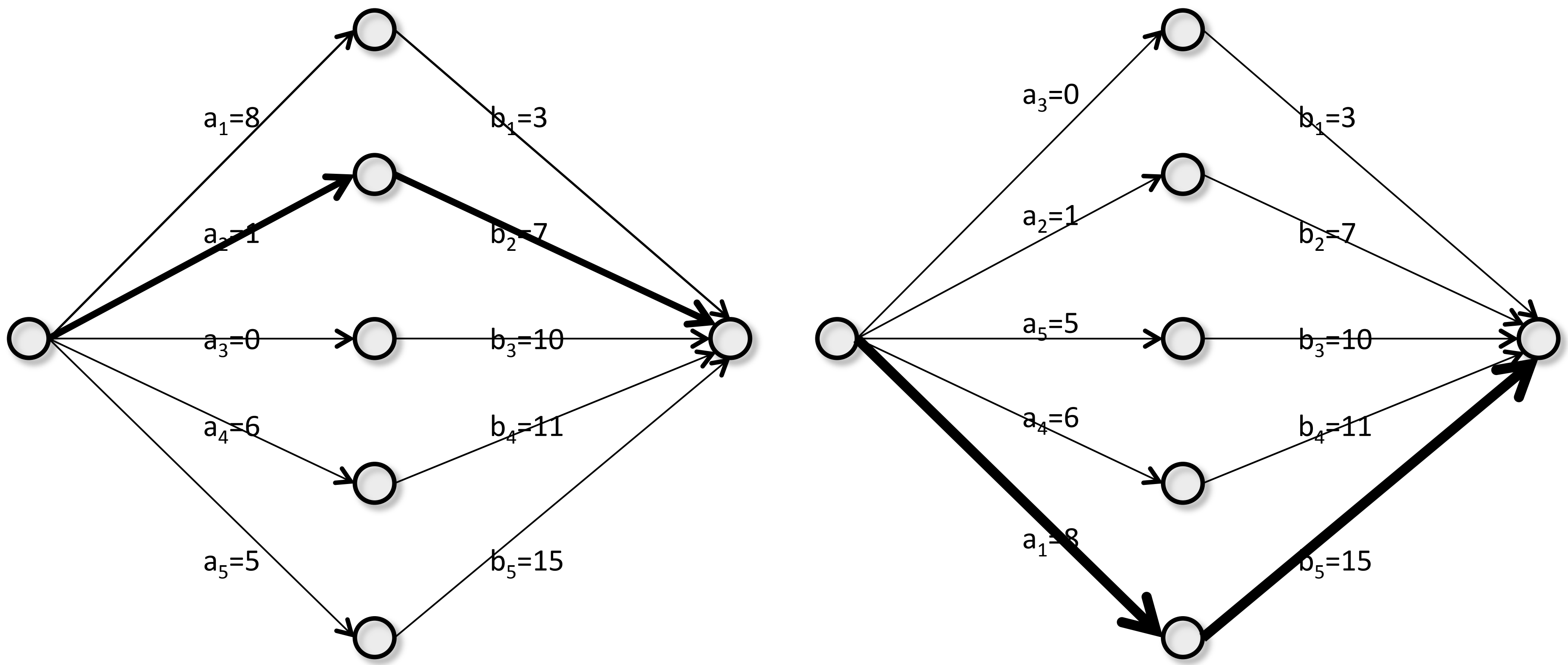


shortest: 8

Among the 12 combinations, 8 is the max.

Subtask 1: Bipartite Graph

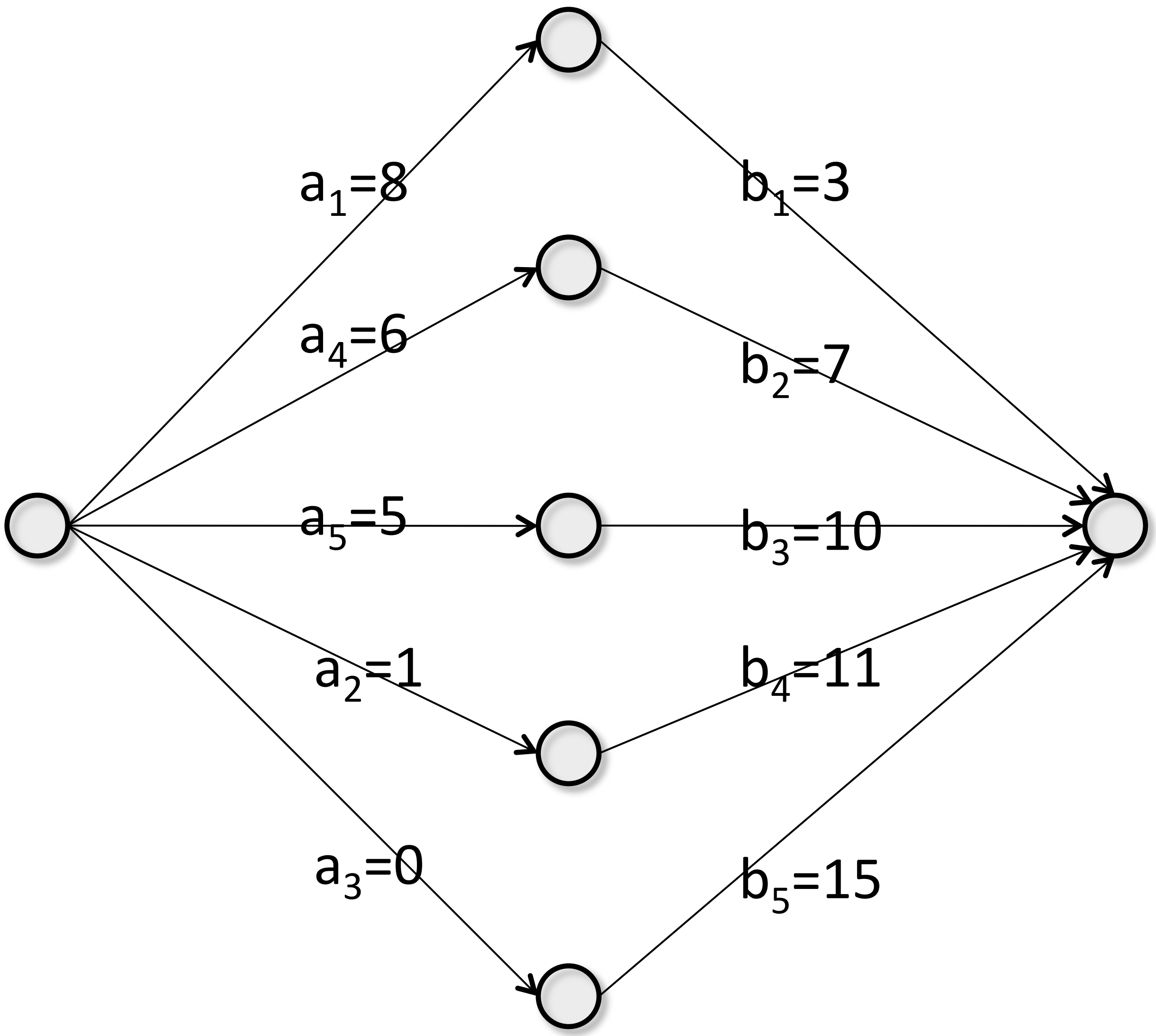




shortest: 8

shortest: 15

among the 120 combinations (matchings), 15 is the max



a_1=8
b_1=3
a_4=6
b_2=7
a_5=5
b_3=10
a_2=1
b_4=11
a_3=0
b_5=15

Solution (subtask 1)

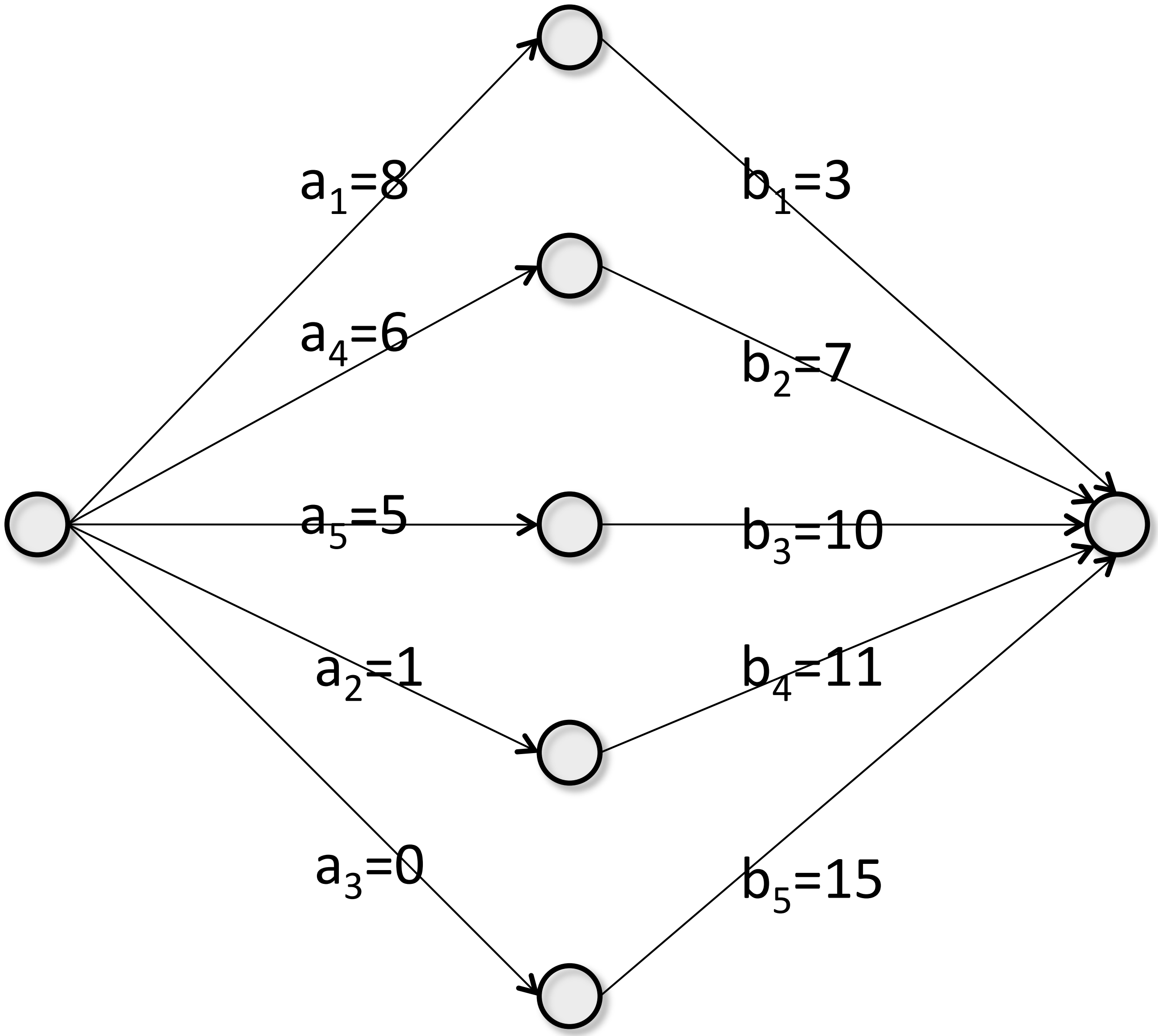
- Sort A in increasing order.
- Sort B in decreasing order.
- Output min(A[i] + B[i])   among all i's.

Running time: Depend on the sorting algo.

O(n log n)

Correctness: Why?

Correctness:



+	0	1	5	6	8
3					11
7				13	
10			15		
11		12			
15	15				

Claim: *The answer must appear in the diagonal of the above table.*

+	a_1	a_2	a_3	a_4	a_5
b_1					
b_2					
b_3					
b_4					
b_5					

a_i and b_i are sort increasing.

Claim: *The answer must appear in the diagonal of the above table.*

proof

+	a_1	a_2	a_3	a_4	a_5
b_1					
b_2					
b_3					
b_4					
b_5					

The answer cannot appear only in the lower triangle, for e.g a_5+b_3.
Suppose so, let us consider the matching that gives $a_5 + b_3$.
Note that elements in $\{a_1,a_2,a_3,a_4\}$ must not match with elements in $\{b_1,b_2\}$, as this give a value smaller than $a_5 + b_3$.
So $\{a_1,a_2,a_3,a_4\}$ must match with $\{b_4,b_5\}$. It is impossible to match 4 items to 2 items (contradiction).

+	a_1	a_2	a_3	a_4	a_5
b_1					
b_2					
b_3					
b_4					
b_5					

The answer cannot appear only in the upper triangle.

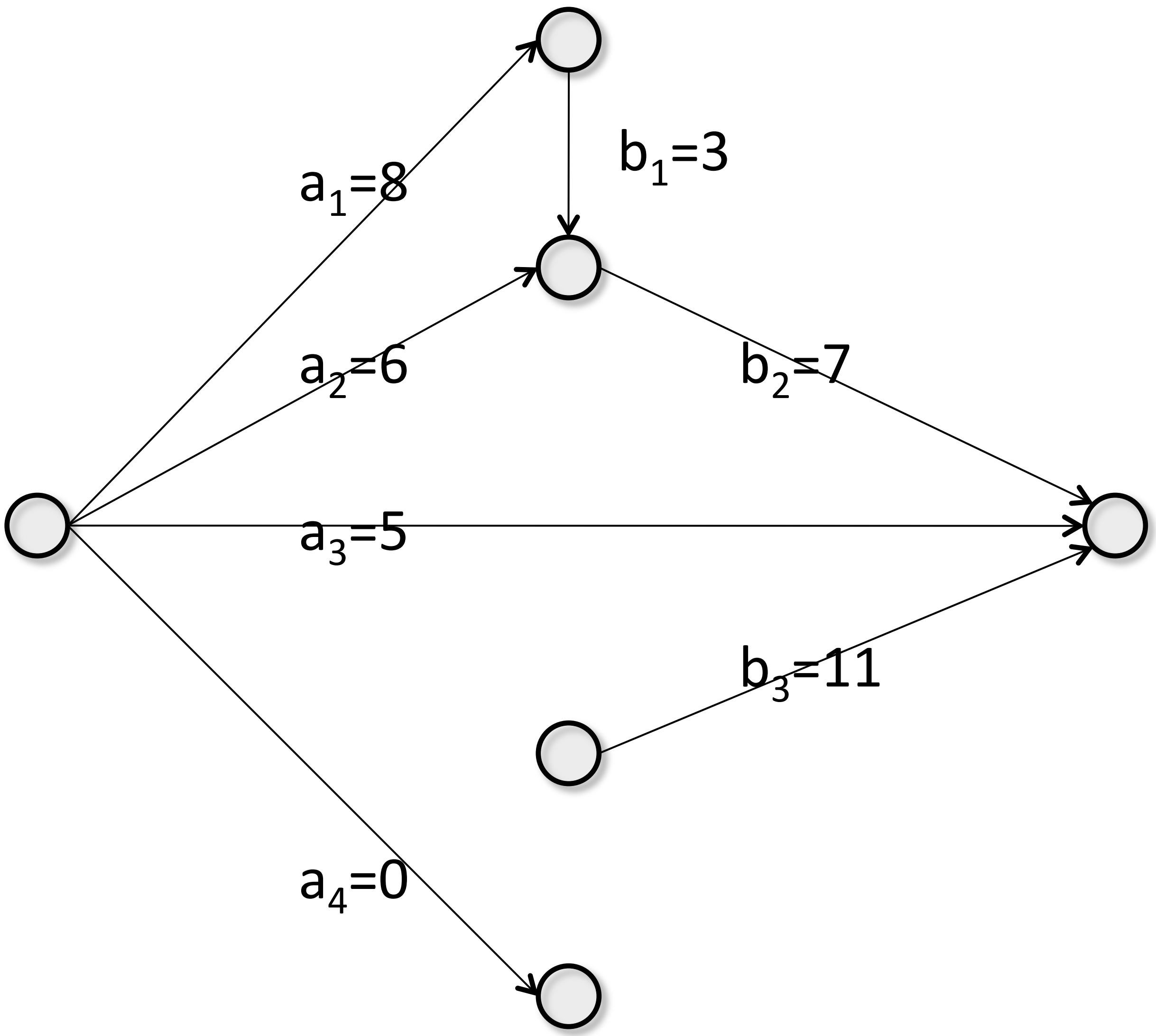
Suppose the answer appear only in $b_2 + a_{3.}$

Case 1: both $a_1 + b_5$ and $a_2 + b_4$ are larger then $(b_2{+}a_3)$. This is impossible as we can find a matching whose min is larger than $(b_2{+}a_3)$

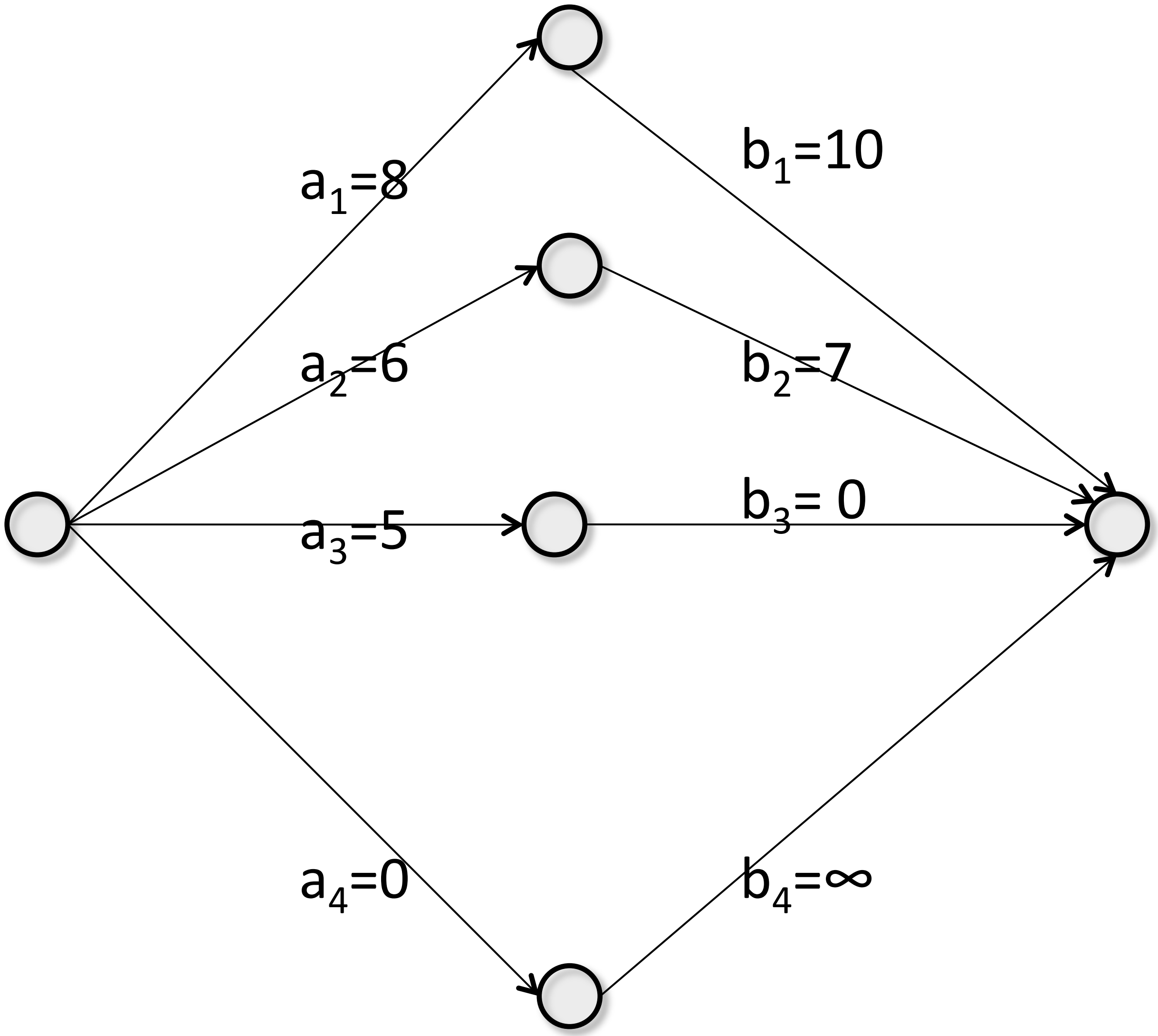
Case 2: $a_1{+}b_5$ is smaller than $(b_2{+}a_3)$. Impossible, because any matching will take one element from the leftmost column.

Case 3: $a_2 + b_4$ is smaller than $(b_2{+}a_3)$. Also impossible…

Subtask 2



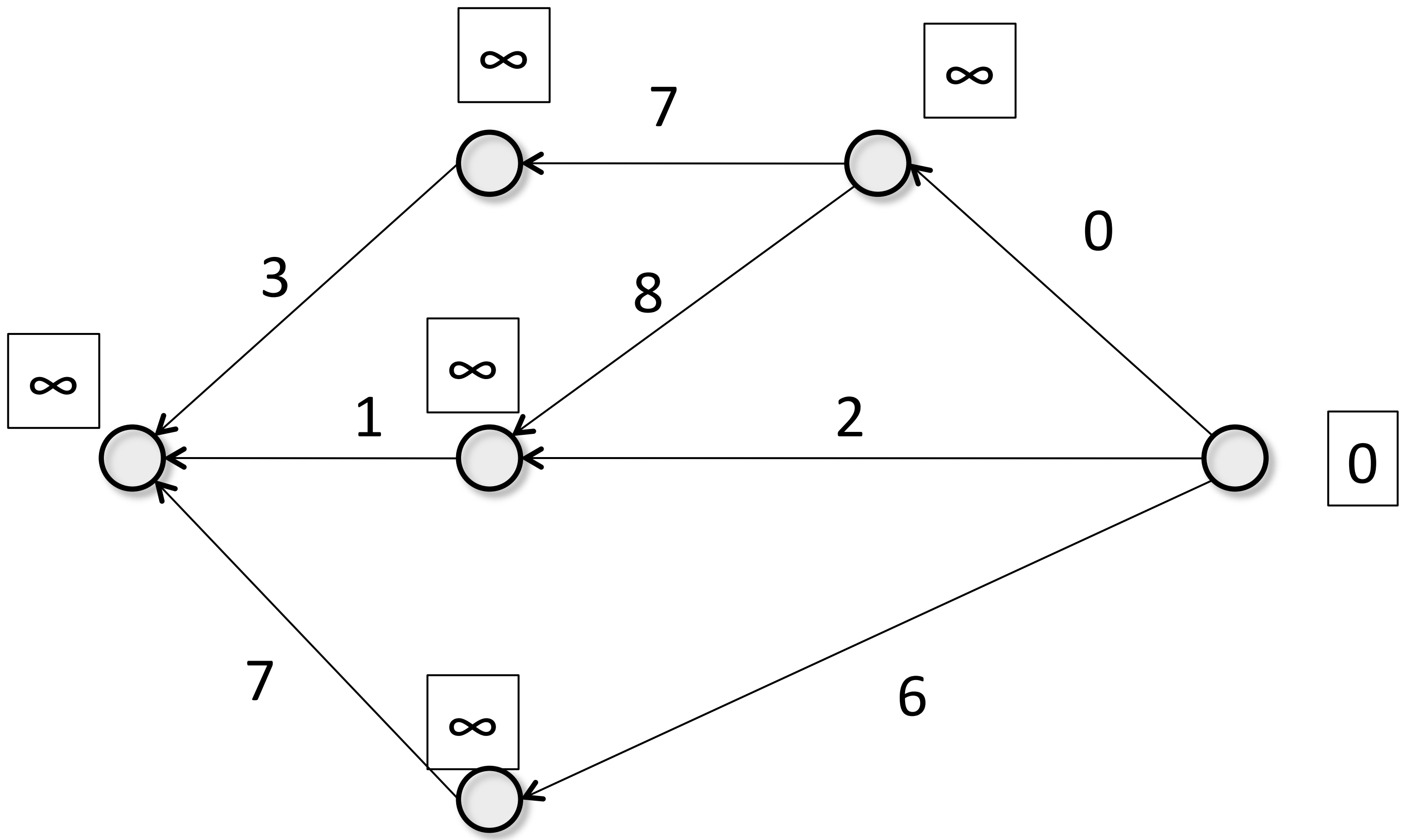
Subtask 2 – reduce it to subtask 1



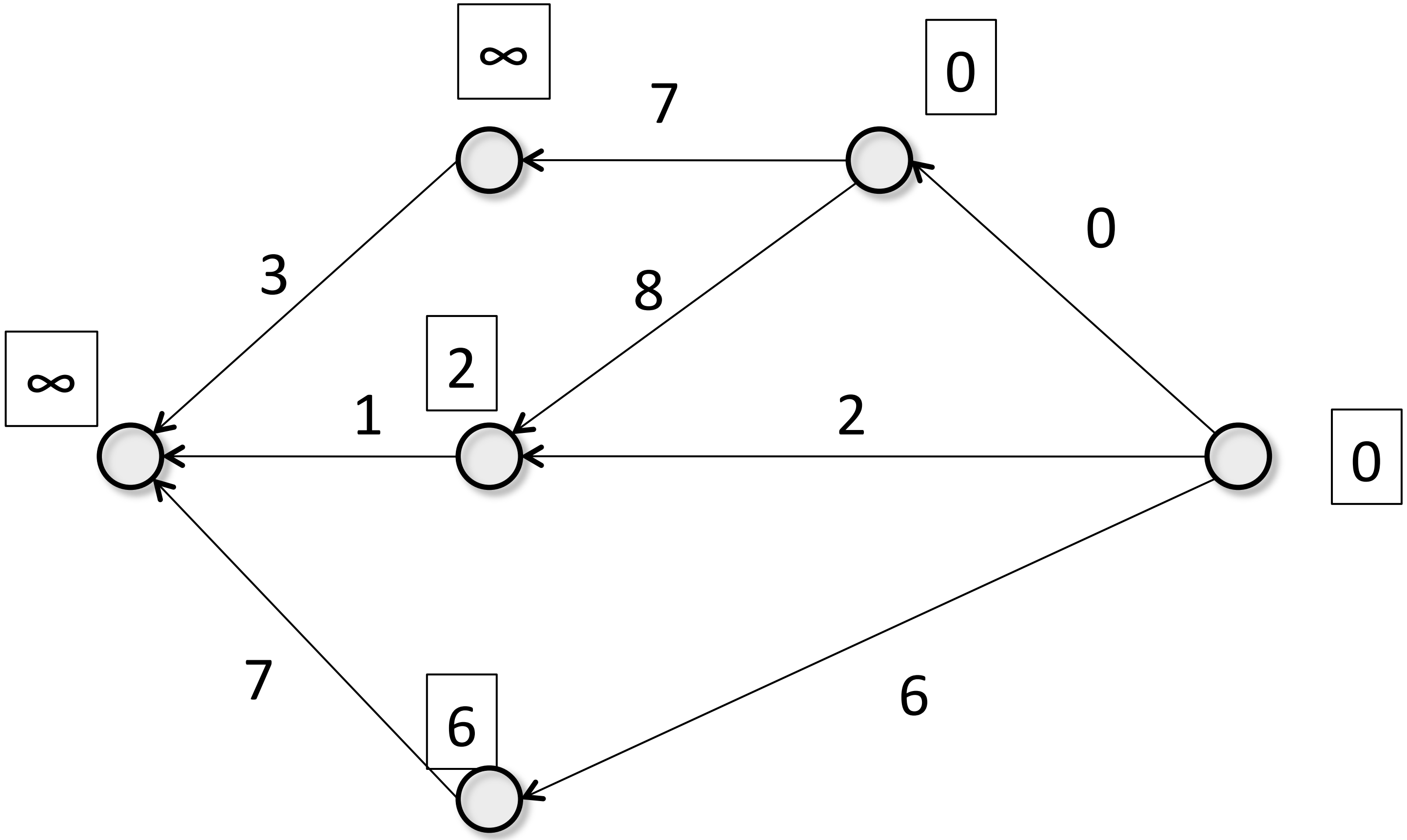
Subtask 3

- Slight modification of Dijkstra's algorithm.
- Reverse the graph (by reversing the direction of the edge). Find the shortest path from the destination to the source.
- Modify the Dijkstra's algorithm.

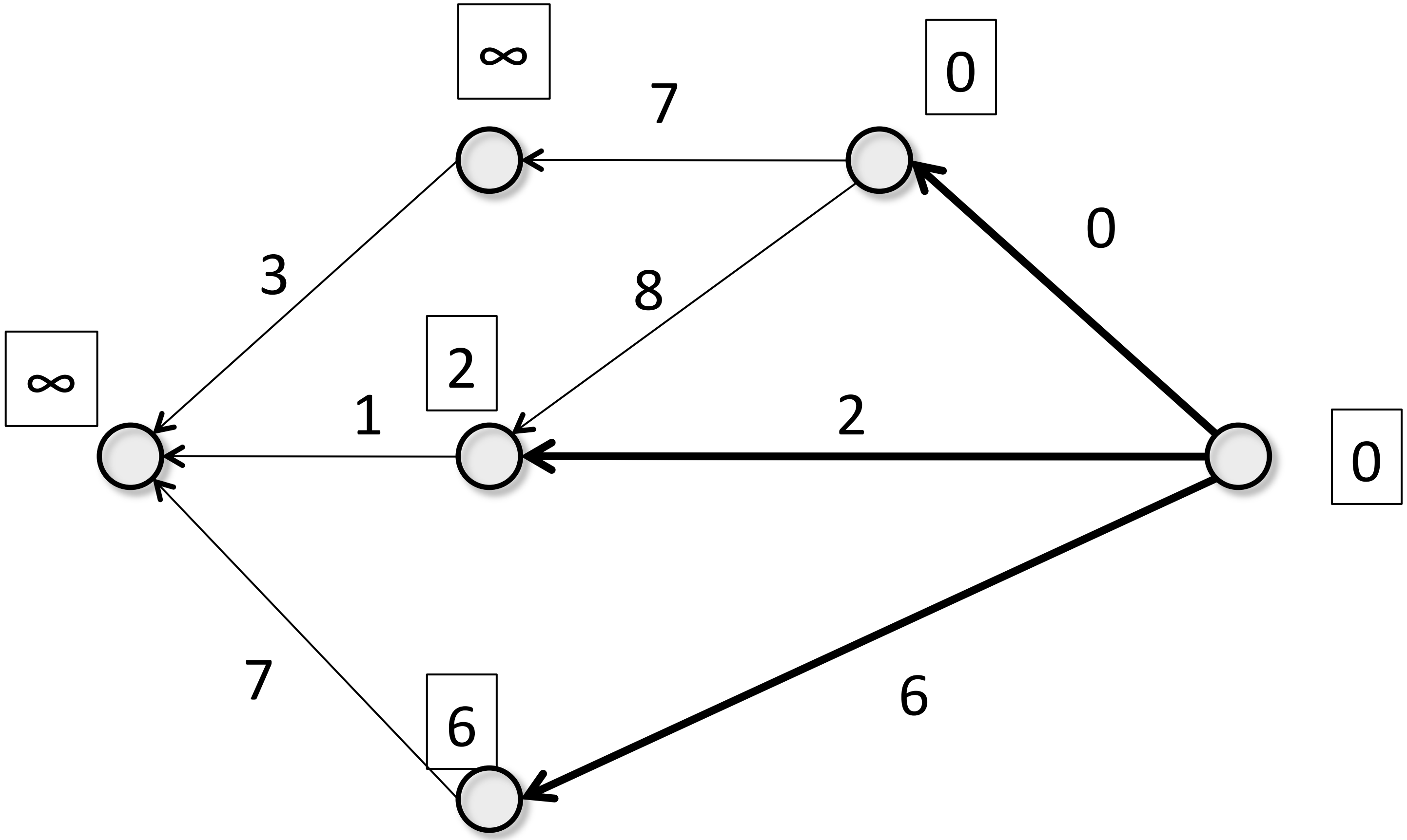
The original Dijkstra's algo: find the shortest path from source to destination.



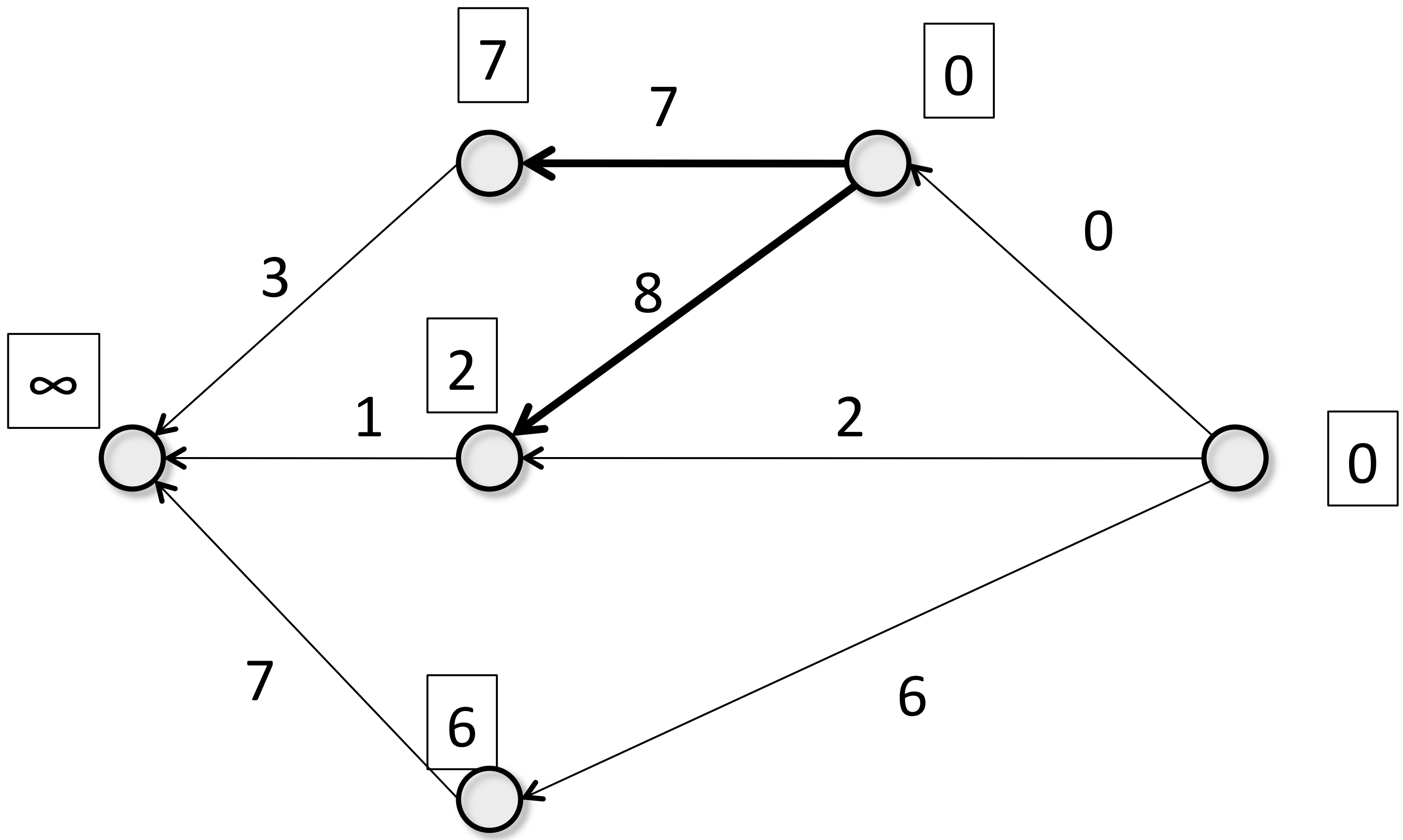
Extract the min, update neighbours



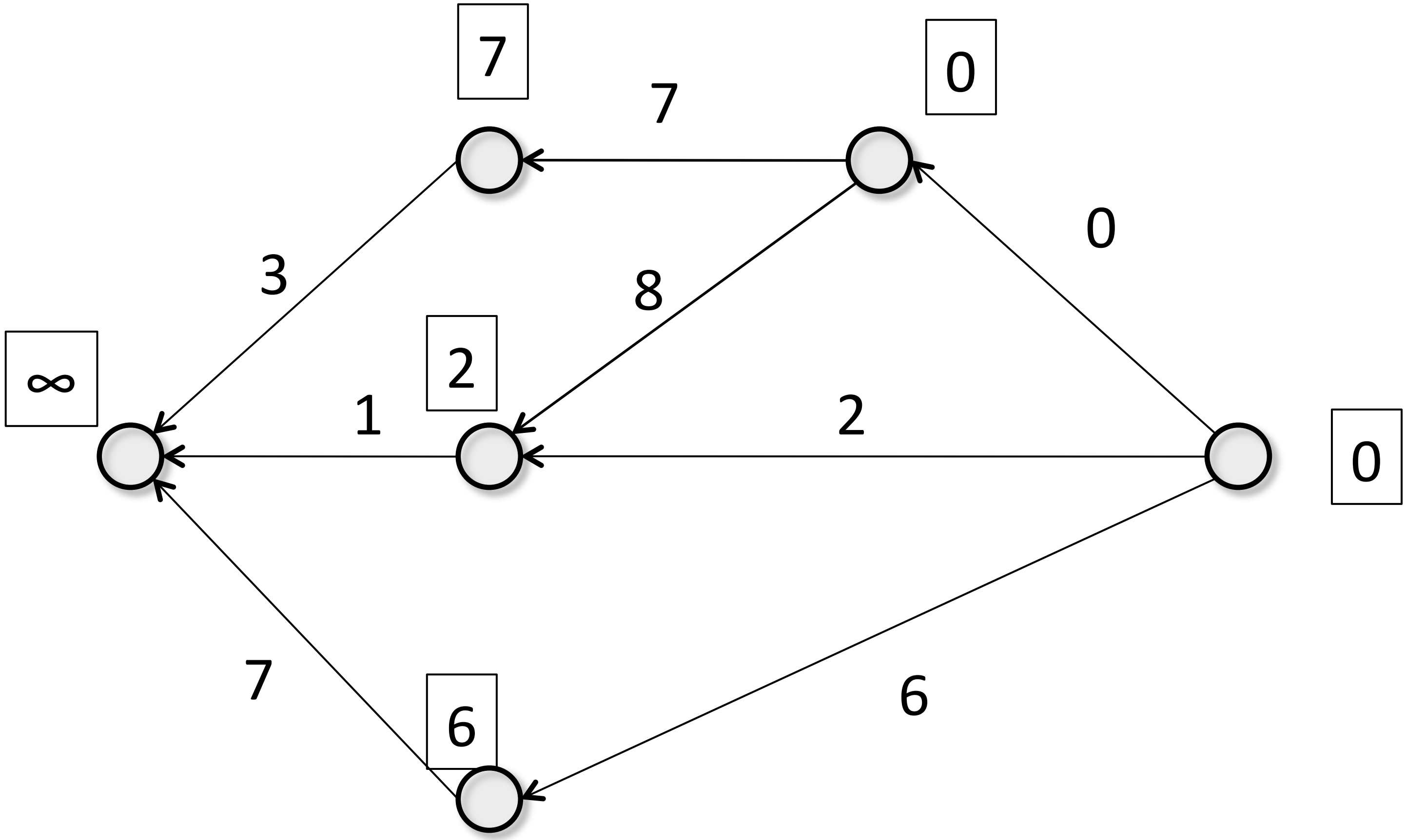
Extract the min



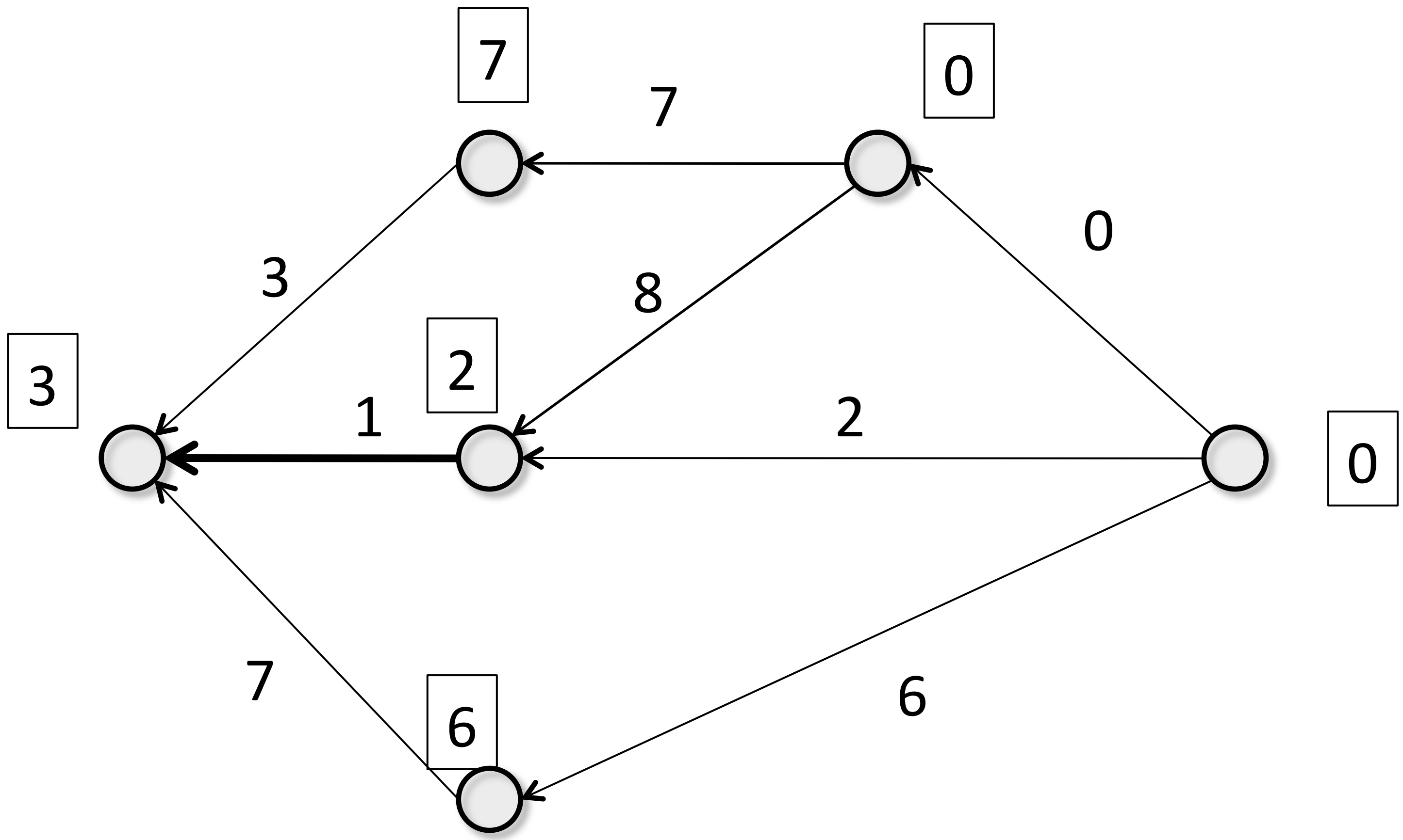
Extract the min, update neighbours



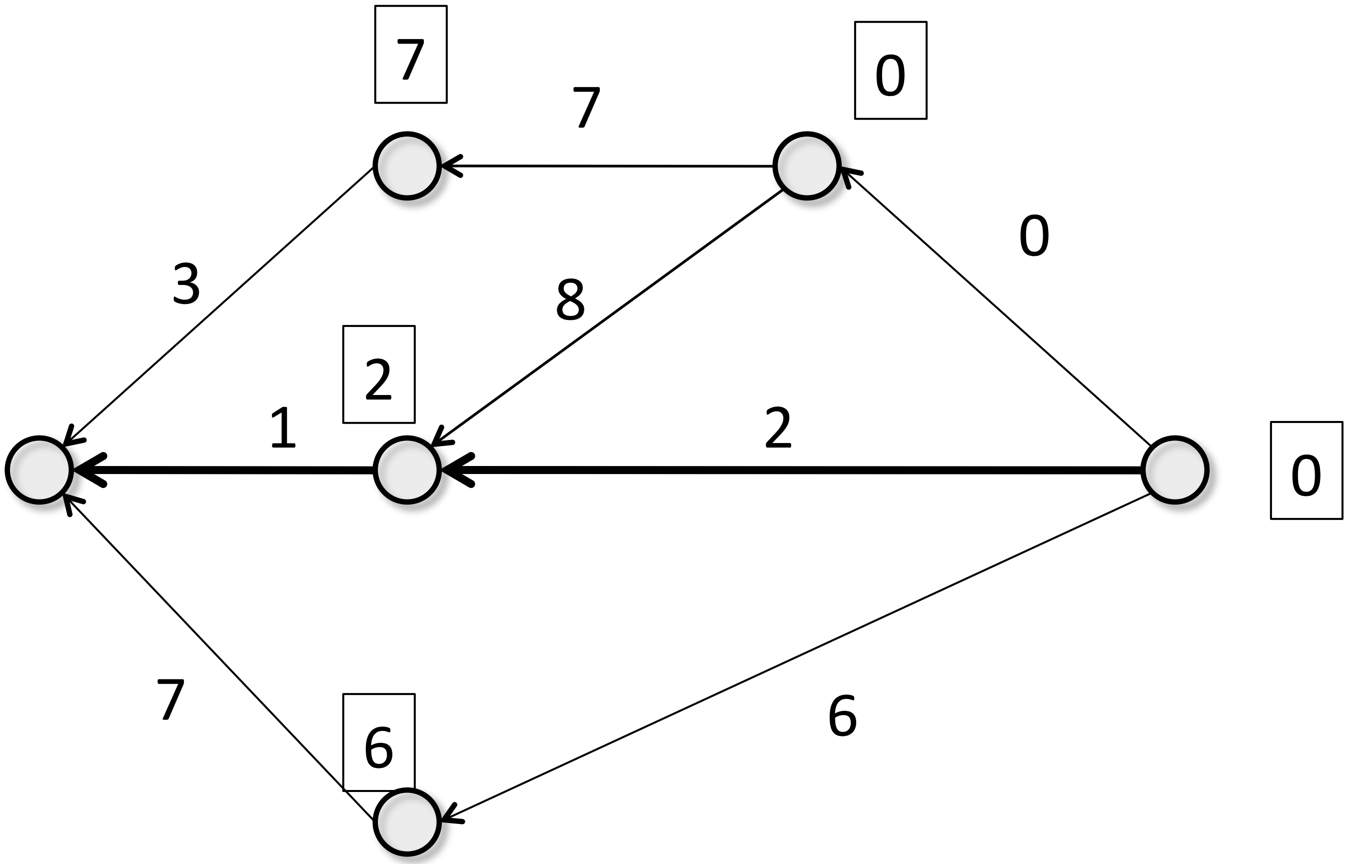
Extract the min,



Extract the min, update neighbours



Extract the min. Done.



Let $G = (V, E, c)$ be a weighted directed graph.
Let $N_{in}(v)$ be the set of incoming neighbors of v.
Let $N_{out}(v)$ be the set of outgoing neighbors of v.
Let $c(u, v)$ be the cost of the edge from u to v.

---

for all $u \in V$ **do**
    $d(u) \leftarrow \infty$
    mark u as unvisited
end for
$d(source) \leftarrow 0$
$Q \leftarrow \{source\}$
while $Q \neq \phi$ **do**
    $v \leftarrow \arg\min_{x \in Q} d(x)$
    $Q \leftarrow Q \setminus \{v\}$
    mark v as visited
    for all $u \in N_{out}(v)$ **do**
        if $d(u) < c(v, u) + d(v)$ **then**
            $d(u) \leftarrow c(v, u) + d(v)$
            $Q \leftarrow Q \cup \{u\}$
        end if
    end for
end while
return $d(destination)$

Solution (subtask 3)

- Modify the Dijkstra's algorithm.

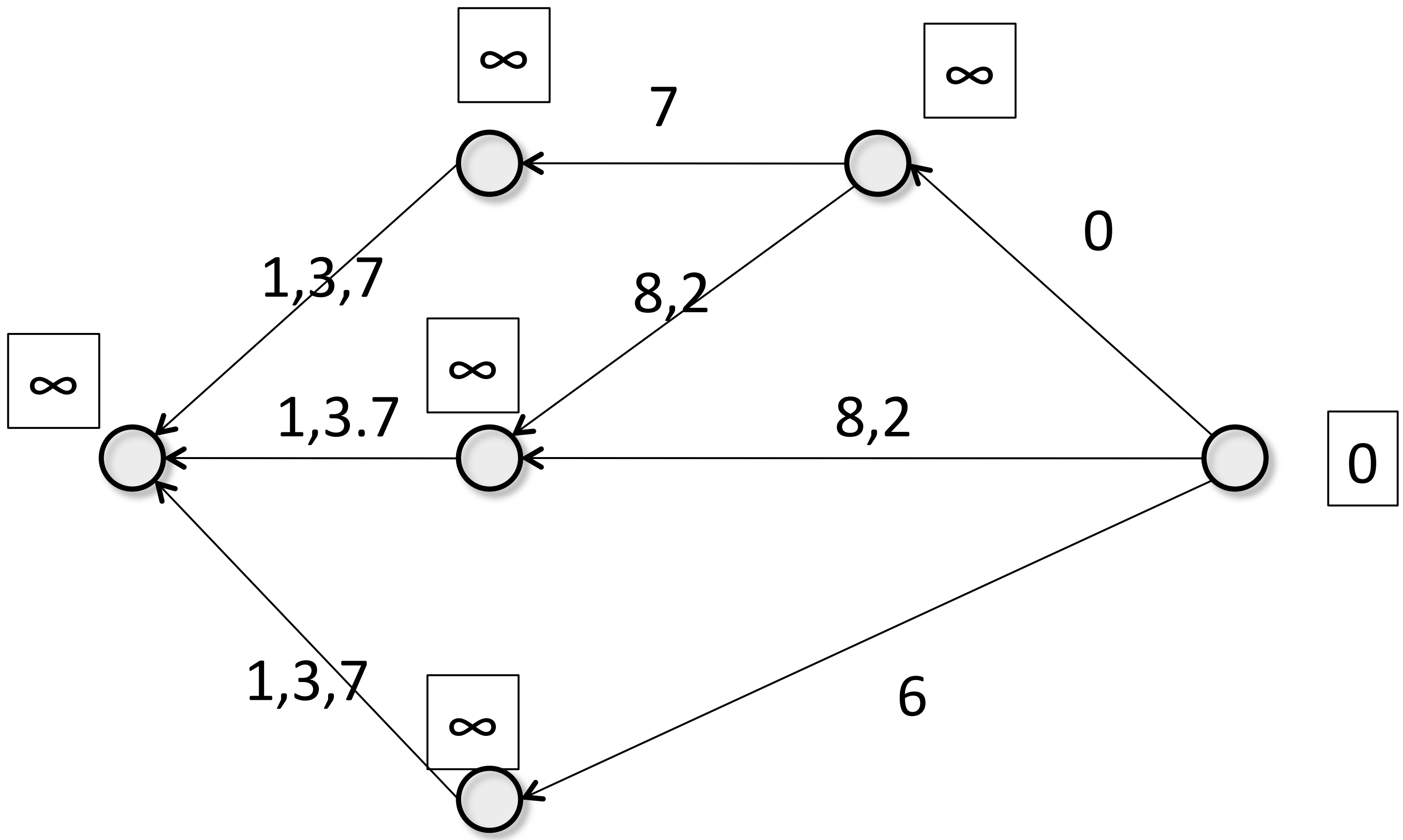
Dijkstra's

Let $G = (V, E, c)$ be a weighted directed graph.
Let $C(v)$ be the multiset of unassigned costs of incoming edges to v.
Let $N_{in}(v)$ be the set of incoming neighbors of v.
Let $N_{out}(v)$ be the set of outgoing neighbors of v.

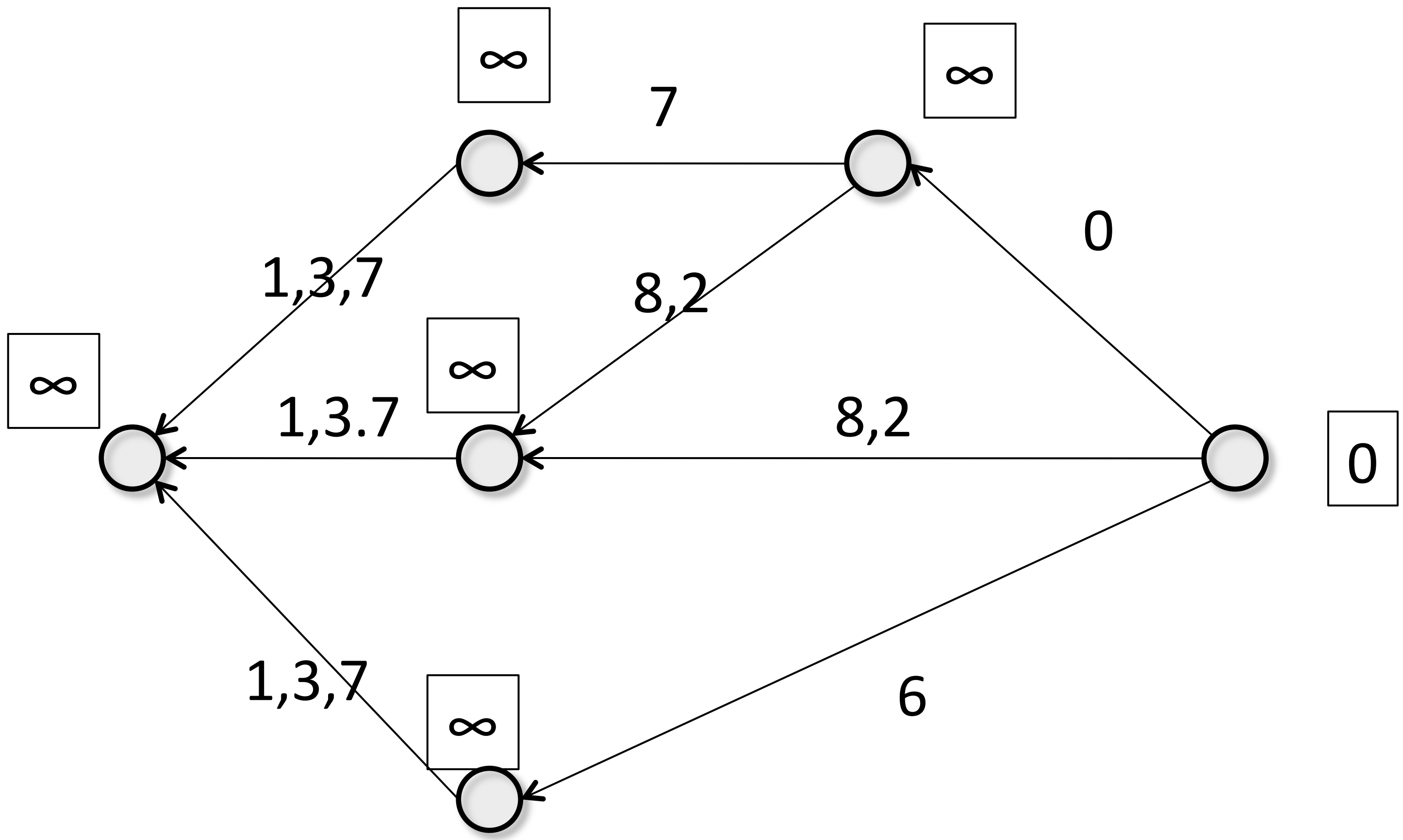
1: **for all** $u \in V$ **do**
2:     $d(u) \leftarrow \infty$
3:     mark u as unvisited
4: **end for**
5: $d(source) \leftarrow 0$
6: $Q \leftarrow \{source\}$
7: **while** $Q \neq \phi$ **do**
8:     $v \leftarrow \arg\min_{x \in Q} d(x)$
9:     $Q \leftarrow Q \backslash \{v\}$
10:     mark v as visited
11:     **for all** $u \in N_{out}(v)$ **do**
12:         $m \leftarrow \max\{x | x \in C(u)\}$
13:         $\widetilde{c}(v, u) \leftarrow m$
14:         $C(u) \leftarrow C(u) \backslash m$
15:         **if** $d(u) < \widetilde{c}(v, u) + d(v)$ **then**
16:             $d(u) \leftarrow \widetilde{c}(v, u) + d(v)$
17:             $Q \leftarrow Q \cup \{u\}$
18:         **end if**
19:     **end for**
20: **end while**
21: **return** $d(destination)$

insert line 12, 13, 14.

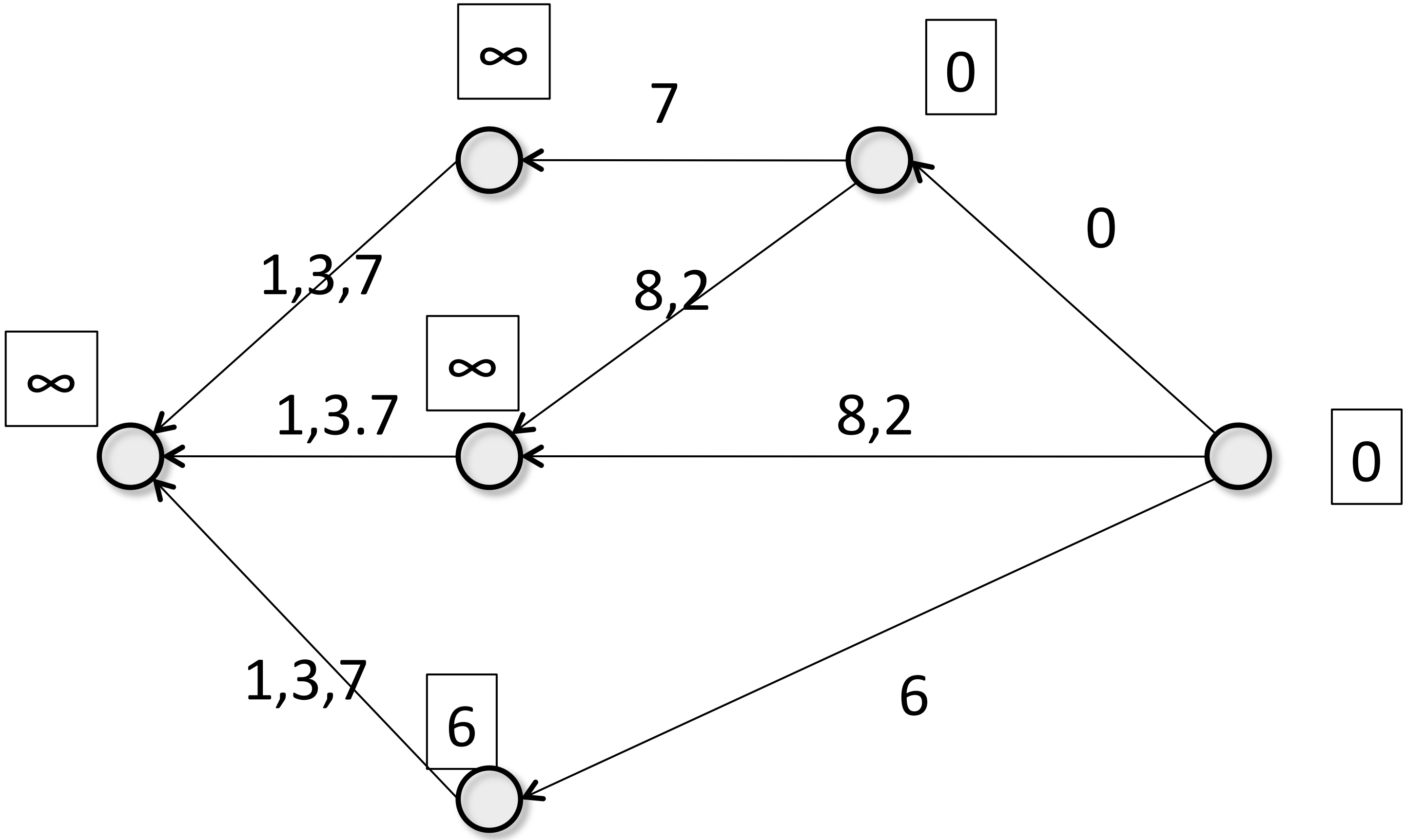
Modified Dijkstra's algo.



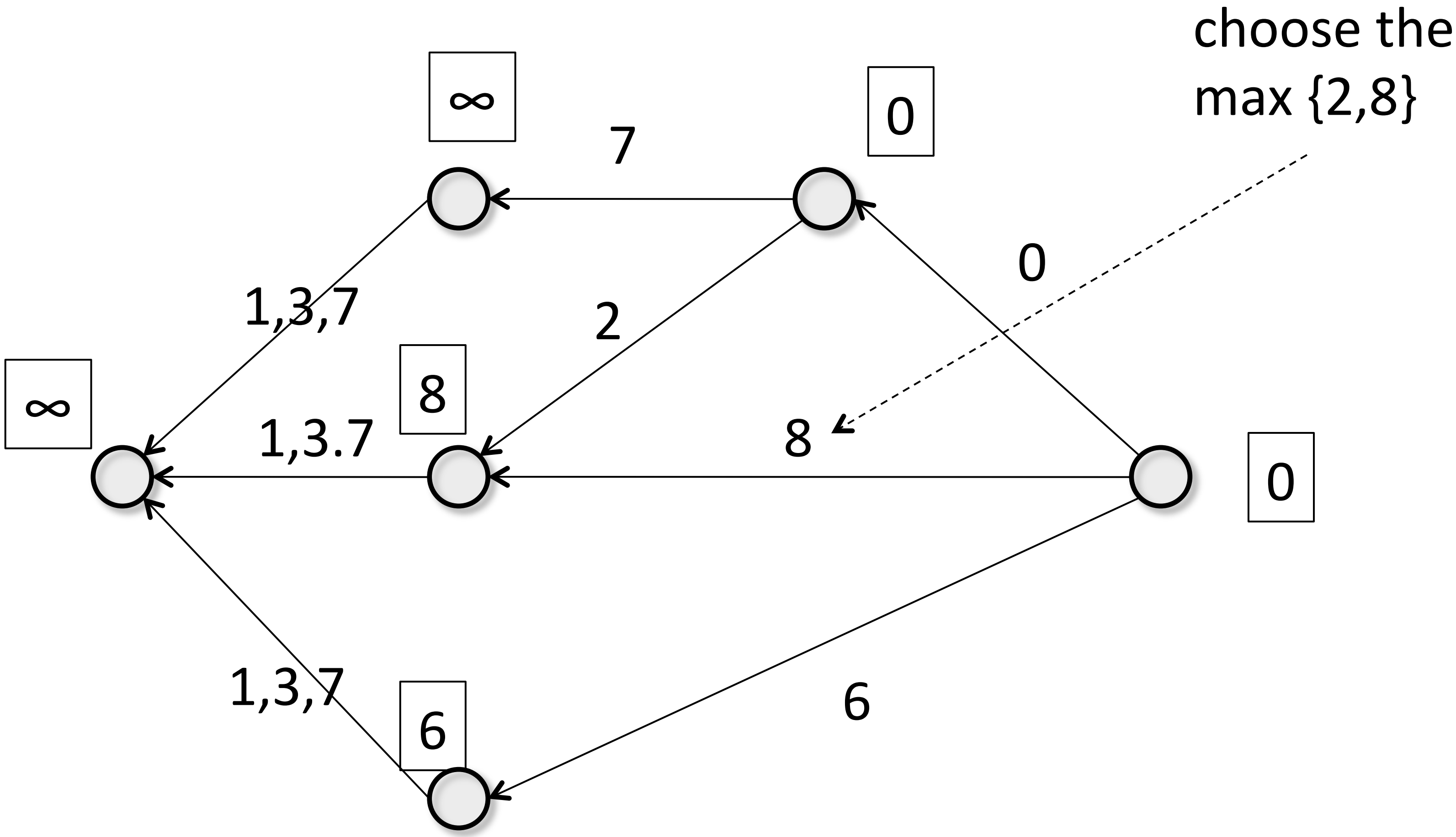
Extract min



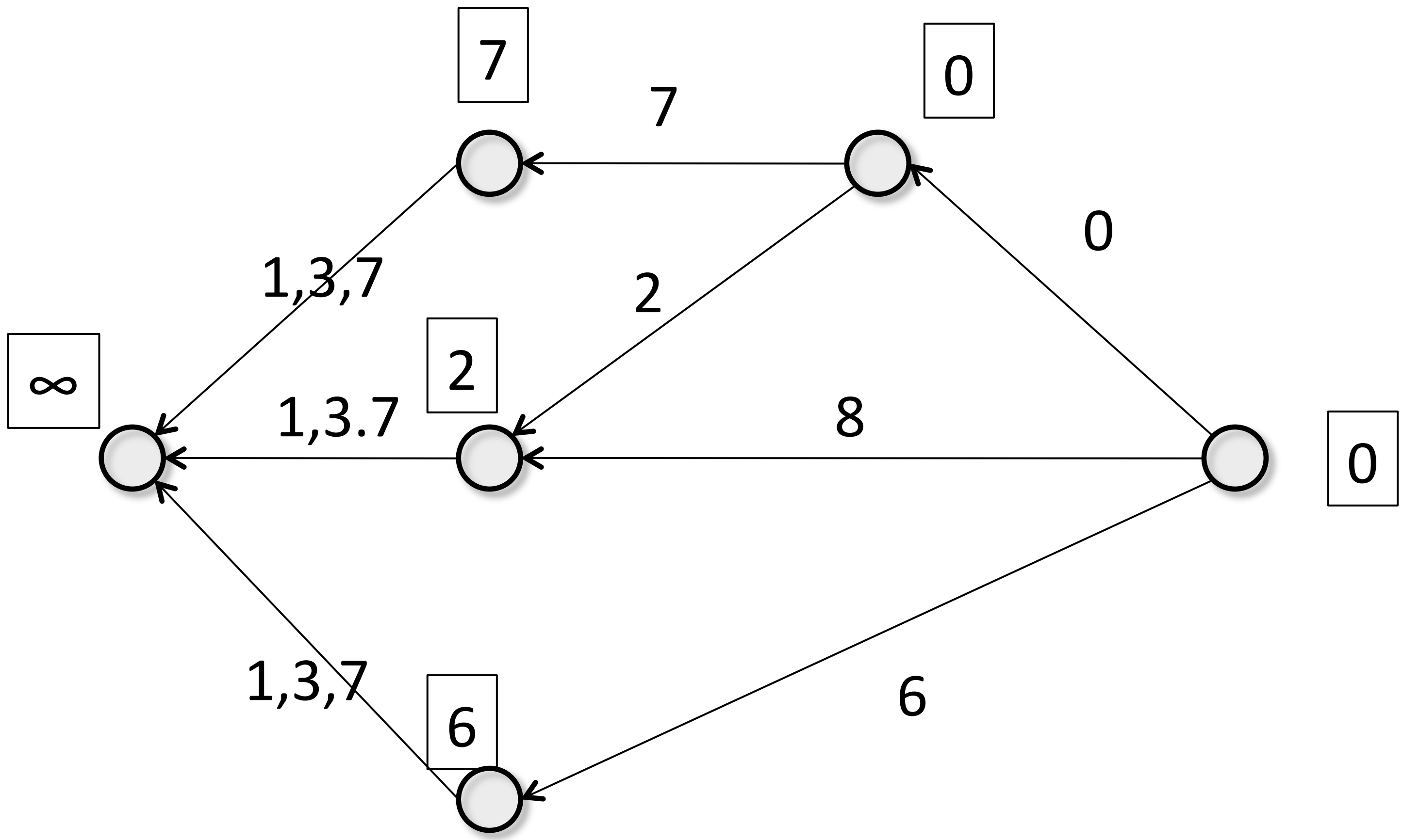
Extract min, update neighbours



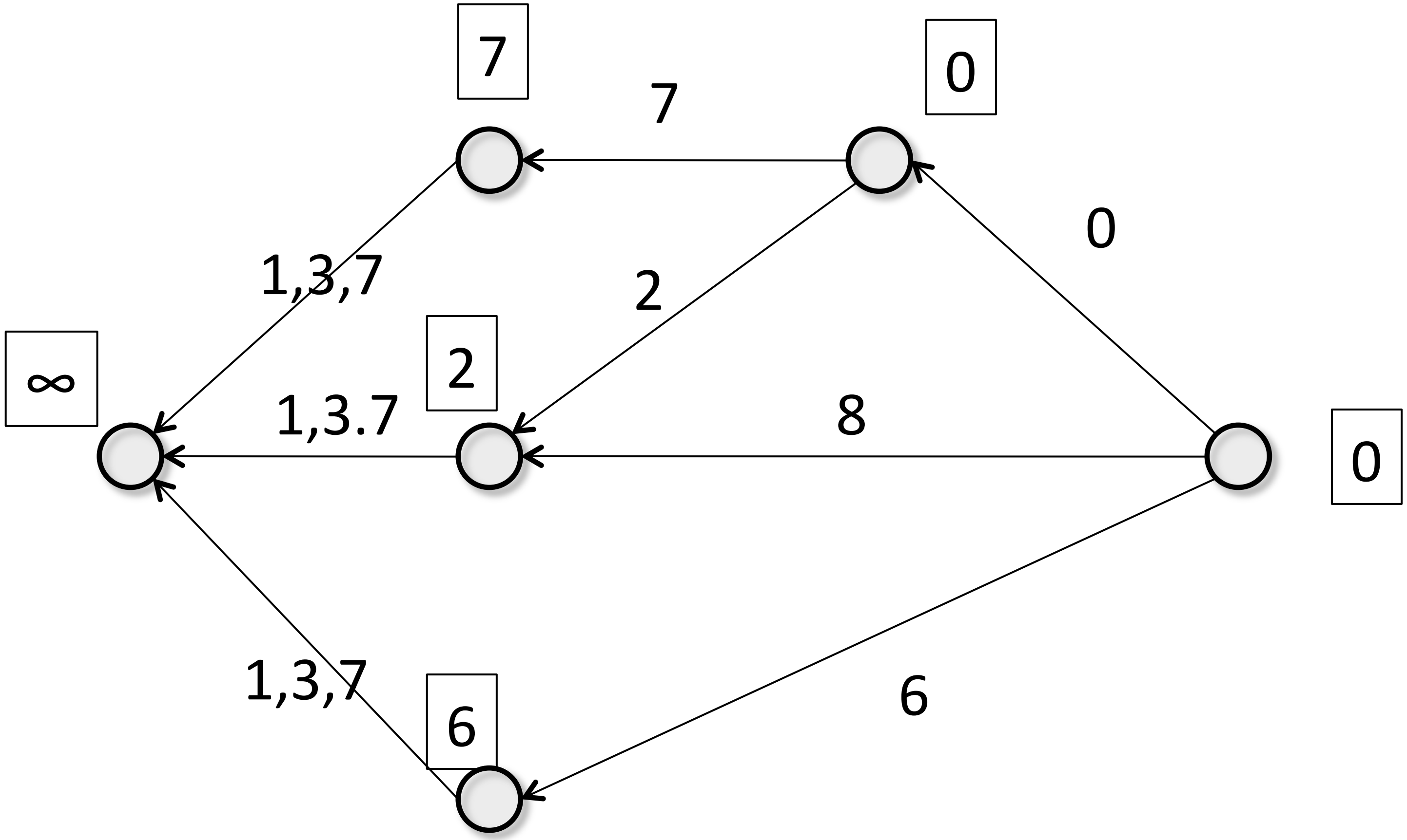
Extract min, update neighbours



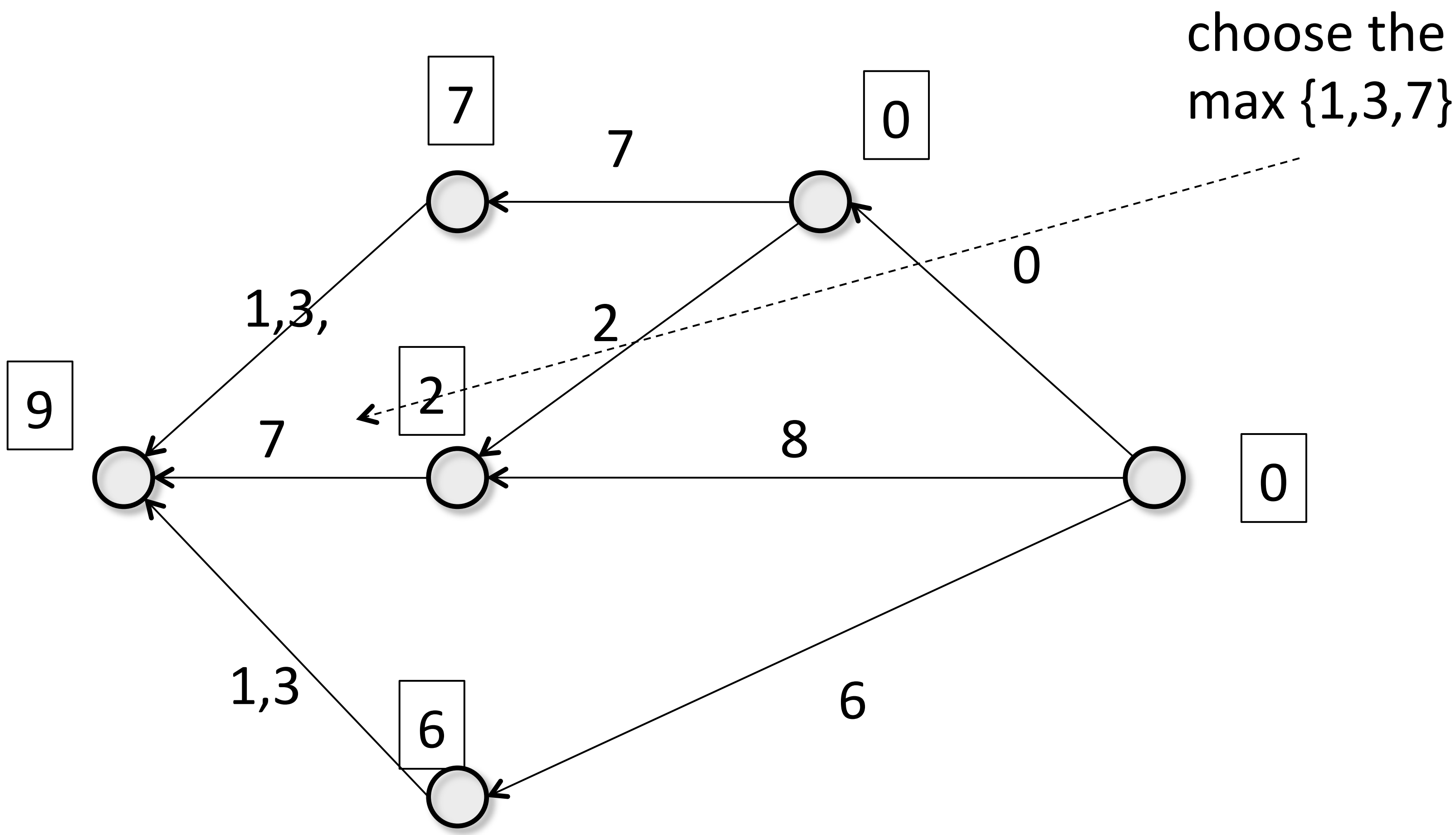
Extract min, update neighbours



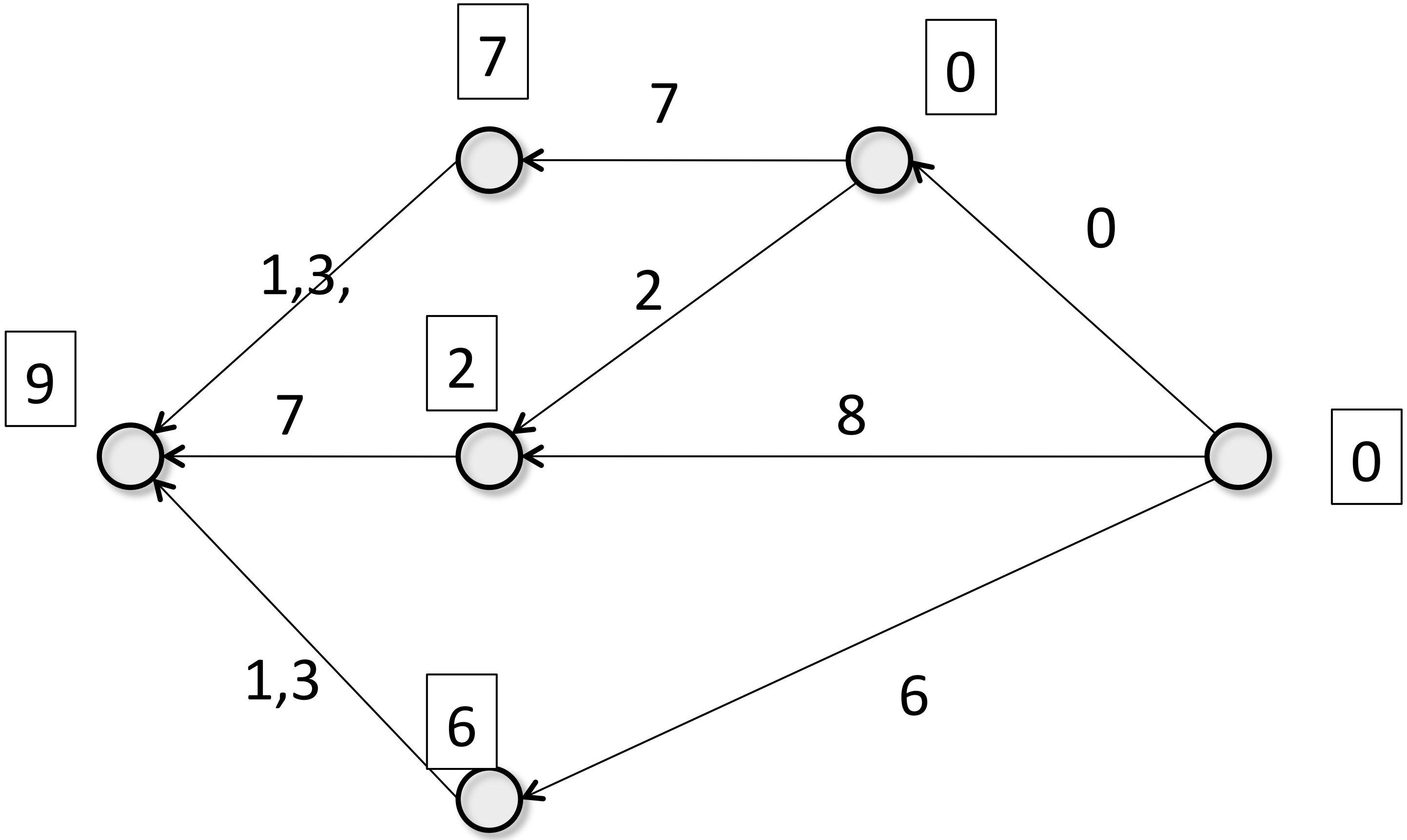
Extract min,



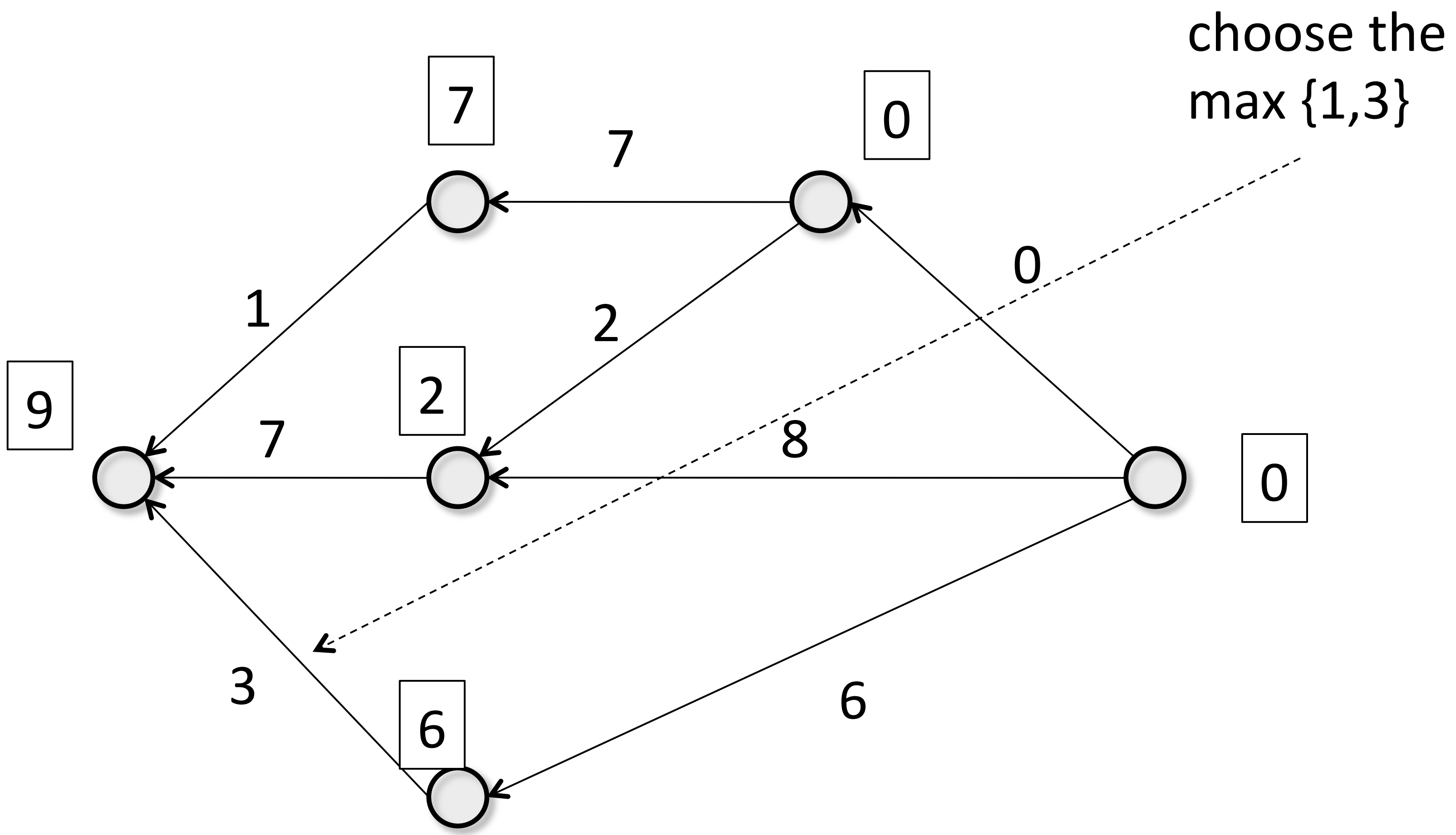
Extract min, update neighbours



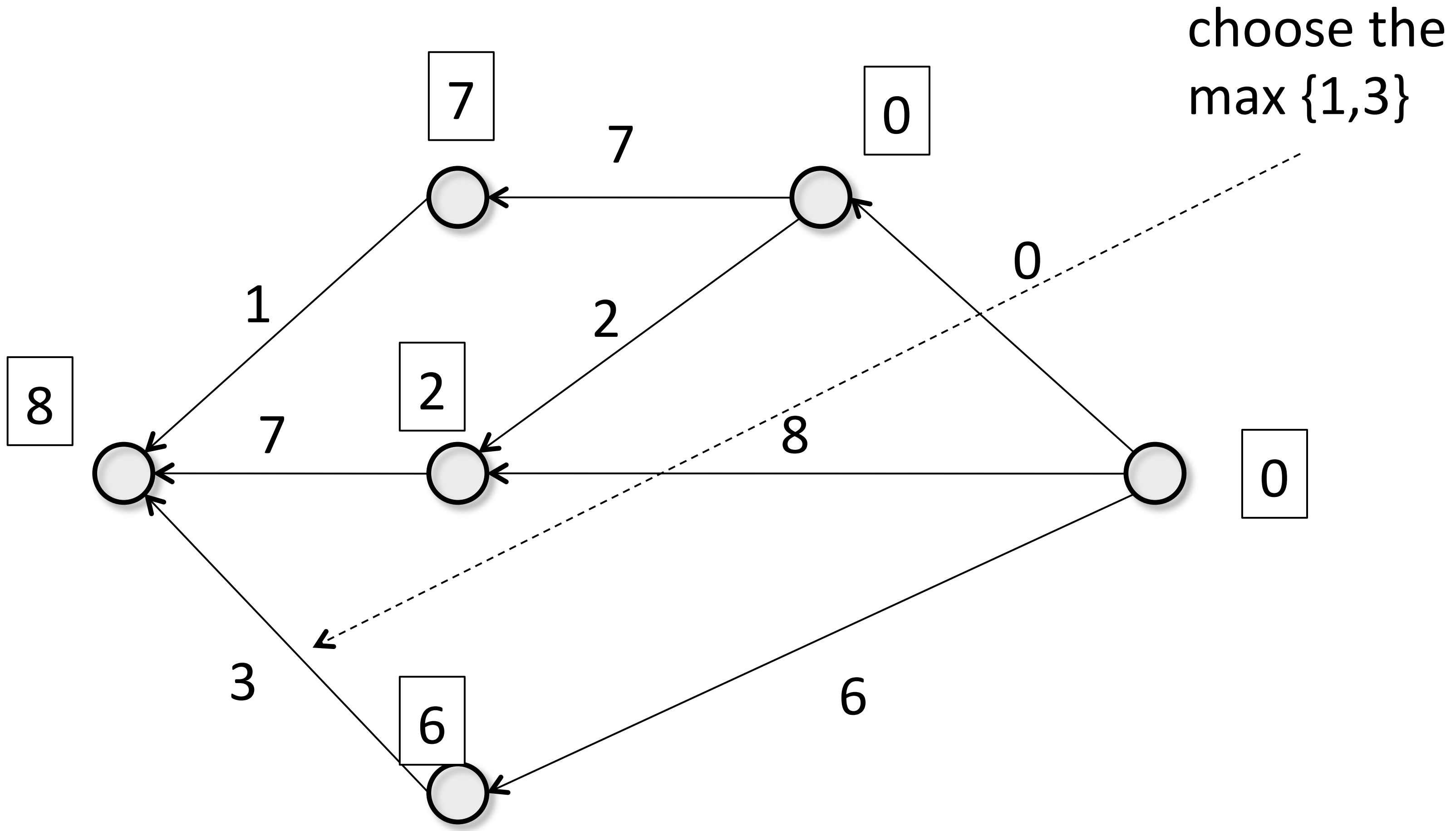
Extract min



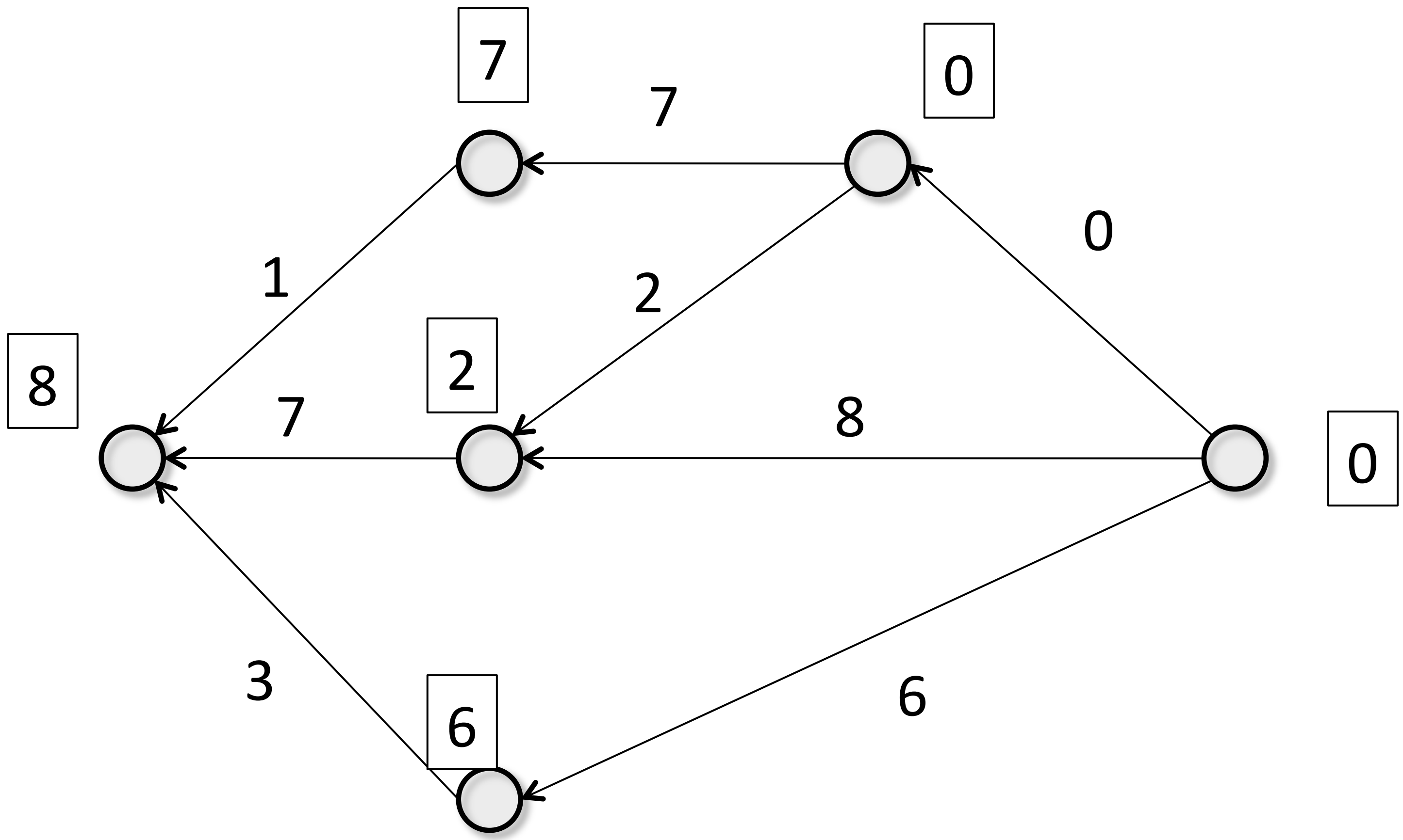
Extract min, update neighbours



Extract min, update neighbours



Extract min, done.



Ans: 8

That's great. Why it is correct?

See attached pdf file for proof.

Subtask 4: graph contain cycle

Need a special check. Such check also required in the original Dijkstra's algorithm.

For simplicity, let us consider the original Dijkstra's algorihtm. If the graph contains a cycle, a vertex may get added to the queue Q twice. Need an extra "if" statement to prevent that.

The original Dijkstra's algo.