NUS
National University of Singapore
School of Computing

Tutorial 10: Shortest Paths I

October 31, 2022

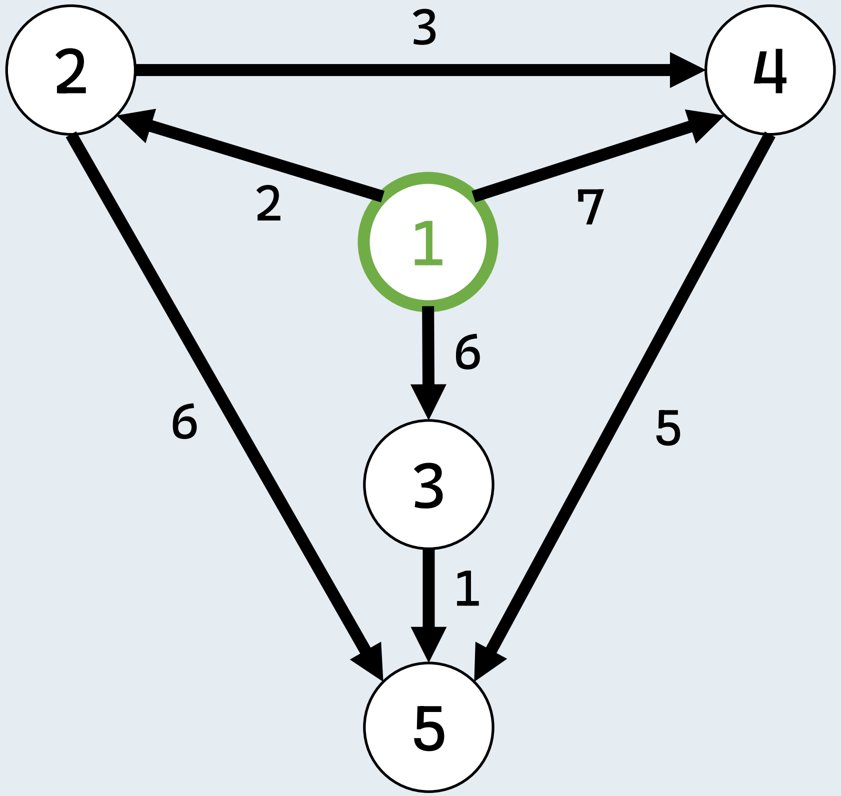
Gu Zhenhao

** Partly adopted from tutorial slides by Wang Zhi Jian.*

Shortest Path

How to find the shortest path between two nodes?

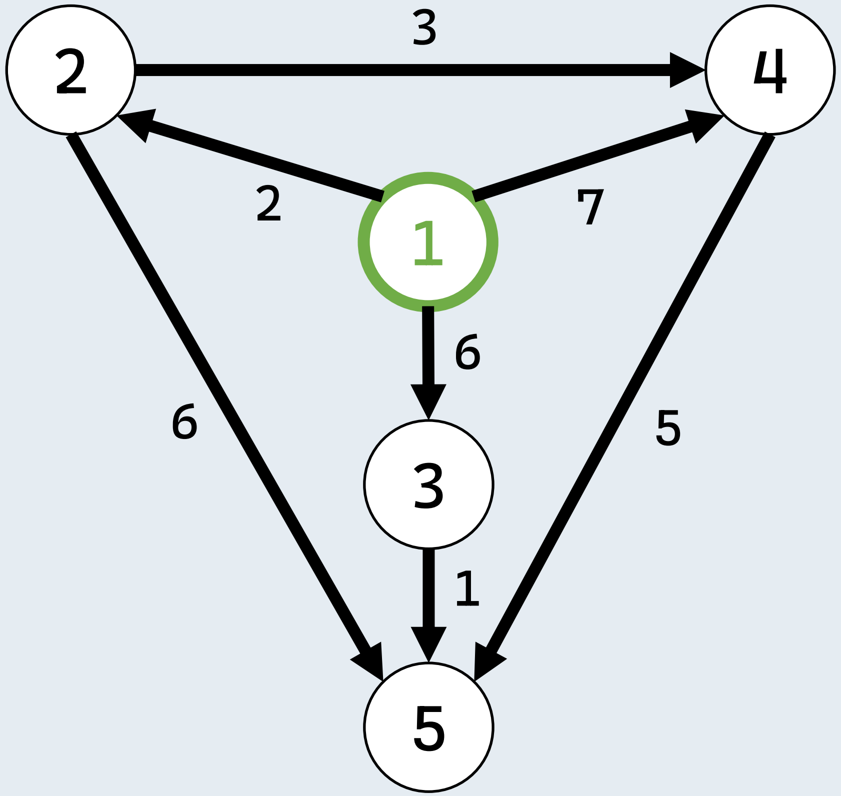
Single-Source Shortest Path



The ***Single-Source Shortest Paths (SSSP)*** problem tries to find the shortest path from one vertex u to all other vertices.

From 1 to ...	1	2	3	4	5
Shortest distance	0	2	6	5	7

Bellman-Ford Algorithm

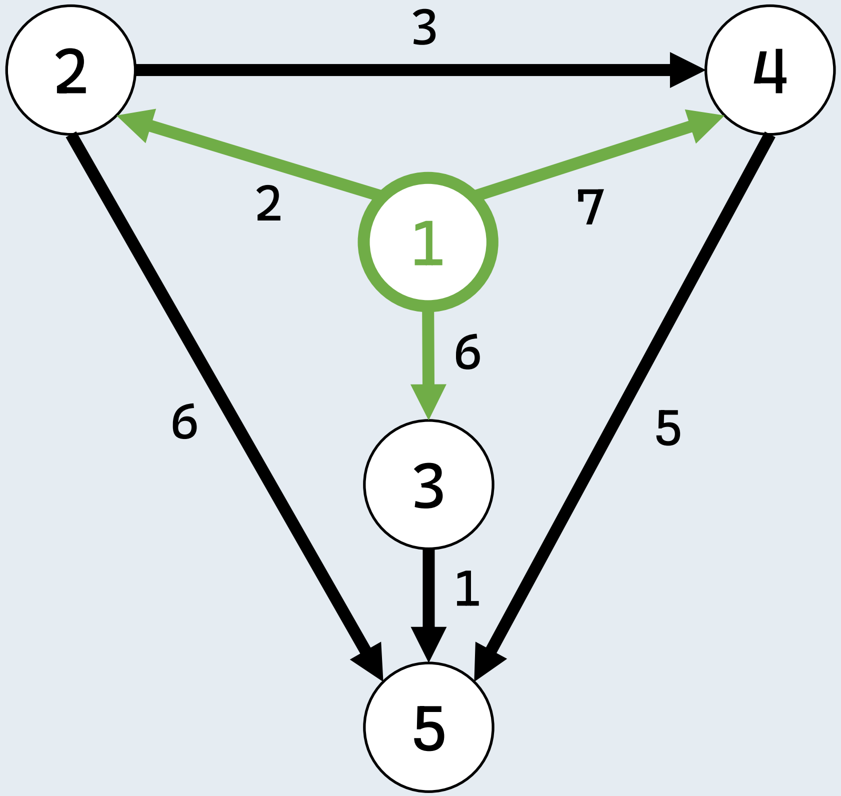


Idea:

- In the i-th iteration, go through all edges and calculate the shortest path from u to v using <u>at most i edges</u> $D_i[v]$.

v	1	2	3	4	5
$D_0[v]$	0	∞	∞	∞	∞

Bellman-Ford Algorithm



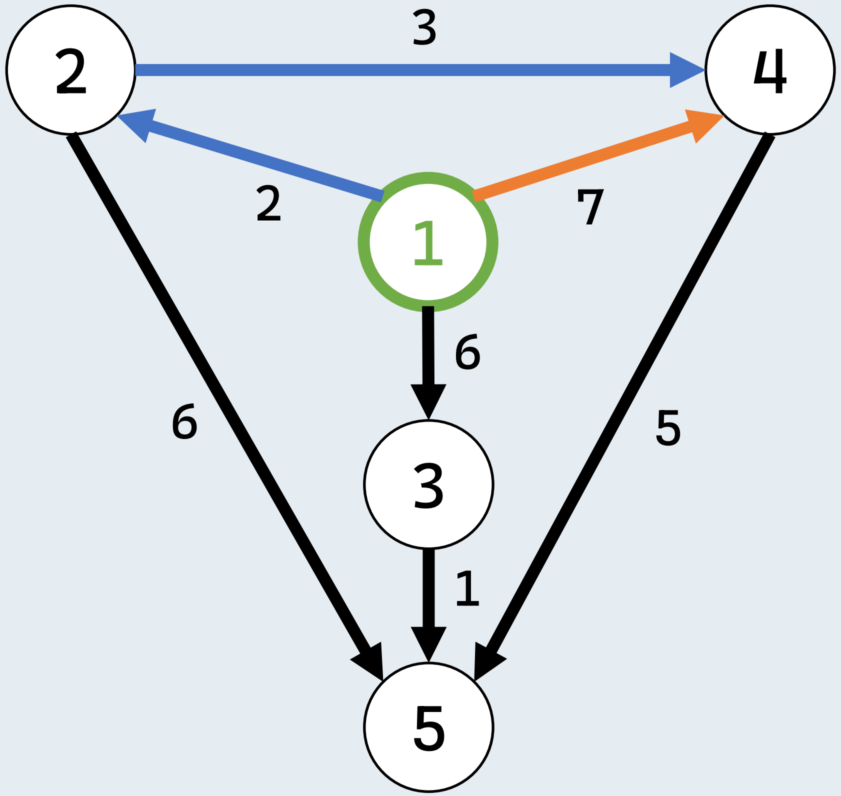
Idea:

- In the i-th iteration, go through all edges and calculate the shortest path from $\boldsymbol{u}$ to v using **at most $\boldsymbol{i}$ edges** $D_i[v]$.

v	1	2	3	4	5
$D_1[v]$	0	2	6	7	∞

Question: How do we calculate $D_2[4]$?

Bellman-Ford Algorithm

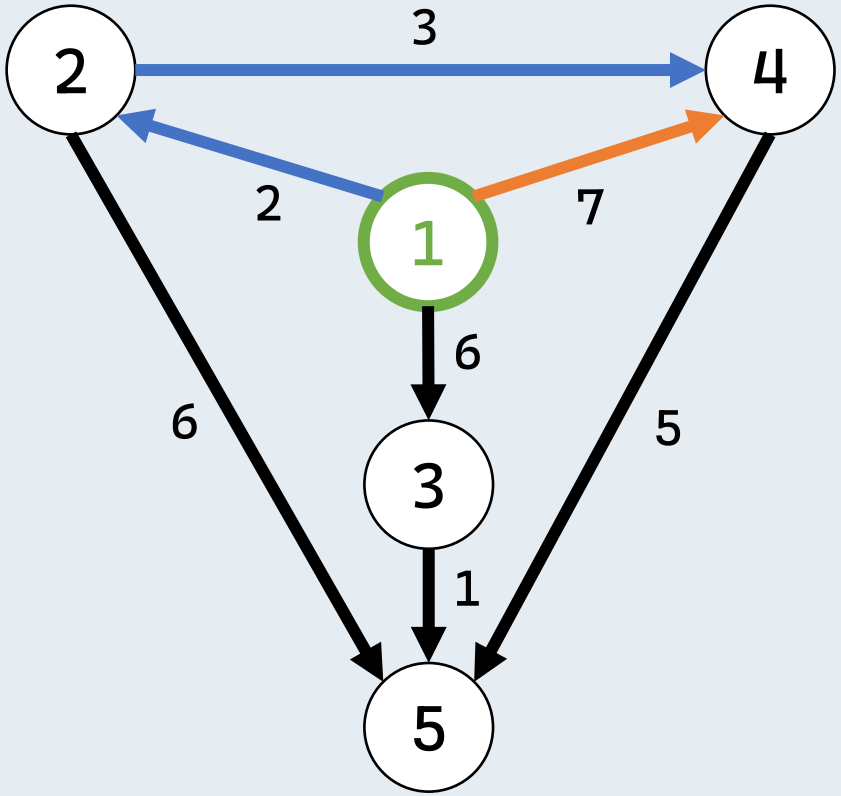


The shortest path P from 1 to 4 using $\leq$ 2 edges can be:

- P uses only 1 edge, $D_2[4] = D_1[4]$.
- P uses 2 edges, and the last edge is (2, 4). $D_2[4] = D_1[2] + w[2,4]$.

v	1	2	3	4	5
$D_2[v]$	0	2	6	5	∞

Bellman-Ford Algorithm



The shortest path P from u to v using $\leq i$ edges can be:

- P uses only $i-1$ edges, $D_i[v] = D_{i-1}[v]$.
- P uses i edges, and the last edge is (t, v). $D_i[v] = D_{i-1}[t] + w[t, v]$.

$$D_i[v] = \min \begin{cases} D_{i-1}[v] \\ \min\limits_{(t,u)\in E} \{D_{i-1}[t] + w[t, v]\} \end{cases}$$

Bellman-Ford Algorithm

for each node $v \in V$ **do**

$\qquad D_0[v] \leftarrow \infty;$

$D_0[u] \leftarrow 0;$

for $i \leftarrow 1$ **to** n **do**

$\qquad$ **for each** edge $(t, v) \in E$ **do**

$\qquad\qquad D_i[v] \leftarrow \min\{D_{i-1}[v], D_{i-1}[t] + w[t, v]\};$

Need to store an additional row storing $D_{i-1}[v]$!

Bellman-Ford Algorithm: Improvement

for each node $v \in V$ **do**

  $D[v] \leftarrow \infty;$

$D[u] \leftarrow 0;$

for $i \leftarrow 1$ **to** n **do**

  for each edge $(t, v) \in E$ **do**

    $D[v] \leftarrow \min\{D[v], D[t] + w[t, v]\};$

Can terminate early if D is no longer updating!

// If D keeps updating on the n-th iteration → we have a negative cycle!

Solving Problems Using Graph

How to apply graph algorithms to solve problems?

Graph-related Problem Solving

1. **Model the problem using graph.**
   What should the vertices and edges be representing?

2. **Identify equivalent graph problem.**
   What standard graph problem can you reduce the problem to?
   MST? SSSP? #(S)CC? Graph Traversal?

3. **Find suitable algorithm.**
   What is the fastest algorithm for the problem, based on properties of graph?

Problem 1

- We have a four digit lock with code U (4-digit integer between 0000 and 9999).
- The lock is initially set to L $(0000 \leq L \leq 9999)$.
- We have R buttons. Pressing the i-th button adds K_i to the lock.
- Only the last 4 digits of the sum are kept.

Goal: find the minimum number of button pressing to reach U.

Problem 1

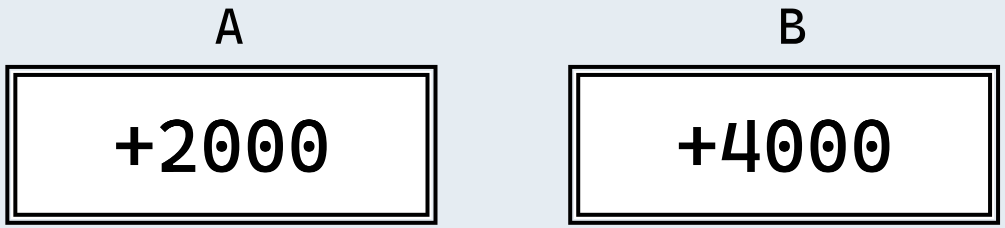
A

$\boxed{+2000}$

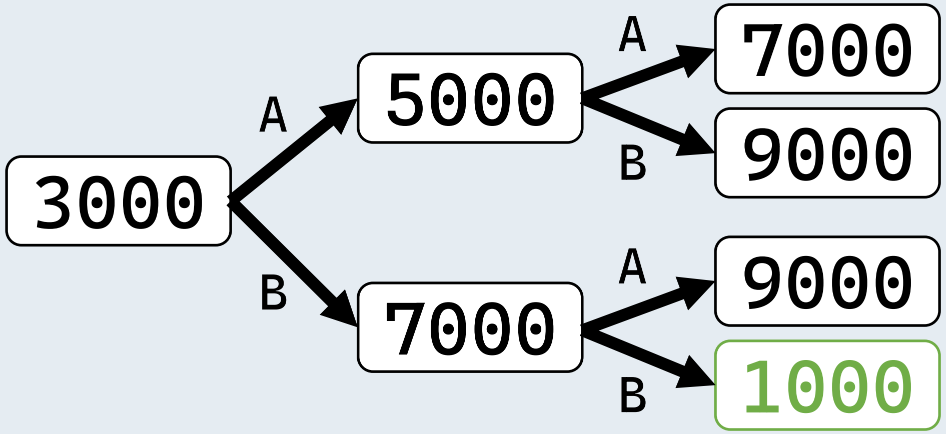
- **Example**: we have only one button.
- Initial value $L = 3000$, goal $U = 1000$.

$$3000 \xrightarrow{A} 5000 \xrightarrow{A} 7000 \xrightarrow{A} 9000 \xrightarrow{A} 1000$$

Problem 1



- **Example**: we have two buttons.
- Initial value $L = 3000$, goal $U = 1000$.



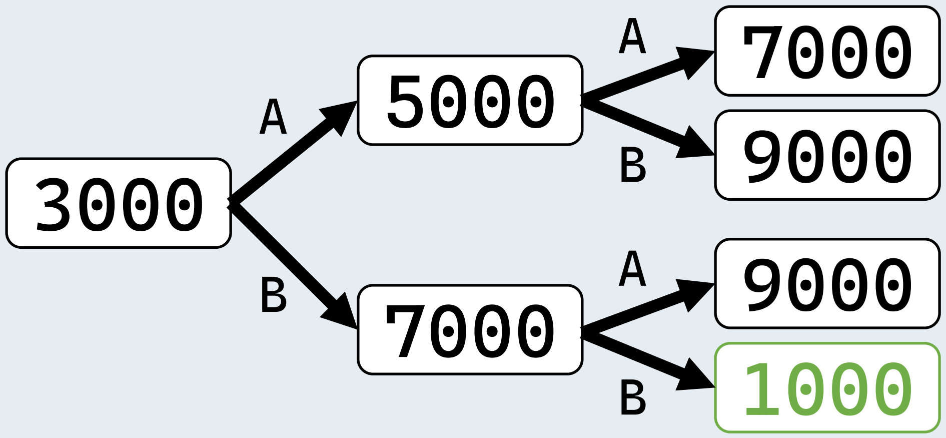
Problem 1

Idea: Model the process as a **tree**, with root being L.

- A node can get to its children with one button press.
- Go through each level in the tree and search for U.

Question: Can there be a problem?

What if L is not reachable...
do we go on forever?

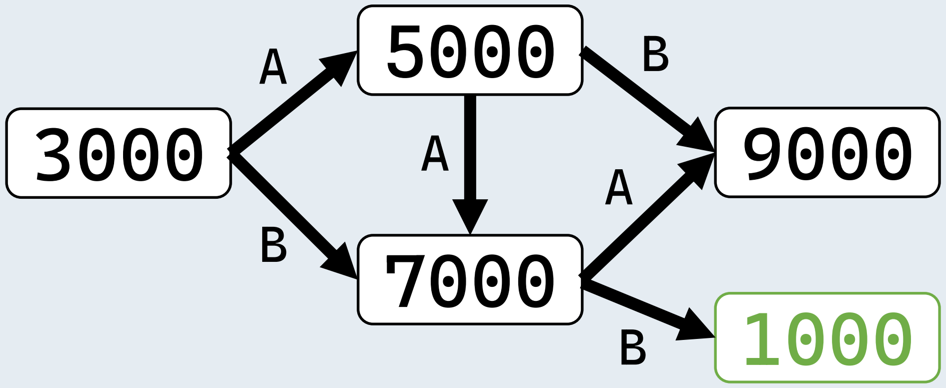


Problem 1

Better Idea: Model the process as a **graph**,

- Vertices: different code on the lock.
- Edges: unweighted, directed edges. An edge (u, v) means u gets to v with one button press.

Equivalent problem: find the shortest path from vertex L to vertex U.



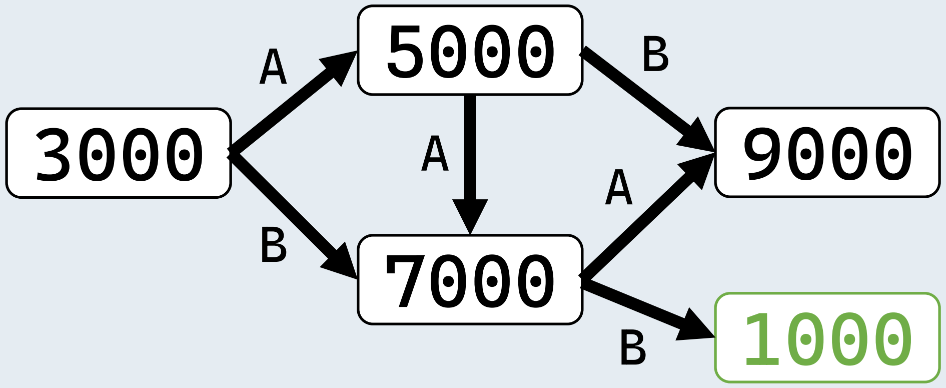
Problem 1

Better Idea: Model the process as a **graph**, each vertex being a code.

- This is an unweighted graph: use **BFS**!

Question: How to check if U is reachable from L?

- If we complete the BFS without finding U, we conclude that we can never reach U!



Problem 2

You are given a maze (R rows and C columns), filling with

- #: a wall.
- .: a passable spot.
- Y: position of you, which is also passable.
- F: the spot is on fire.

Goal: Find the earliest time that you can safely exit the maze, if possible.

```
#####
#YF.#
#..##
#...#
```

Problem 2

You are given a maze (R rows and C columns), filling with

- #: a wall.
- .: a passable spot.
- Y: position of you, which is also passable.
- F: the spot is on fire.

Goal: Find the earliest time that you can safely exit the maze, if possible.

```
#####
#FFF#
#YF##
#...#
```

Problem 2

You are given a maze (R rows and C columns), filling with

- #: a wall.
- .: a passable spot.
- Y: position of you, which is also passable.
- F: the spot is on fire.

Goal: Find the earliest time that you can safely exit the maze, if possible.

```
#####
#FFF#
#FF##
#YF.#
```

Problem 2

You are given a maze (R rows and C columns), filling with

- #: a wall.
- .: a passable spot.
- Y: position of you, which is also passable.
- F: the spot is on fire.

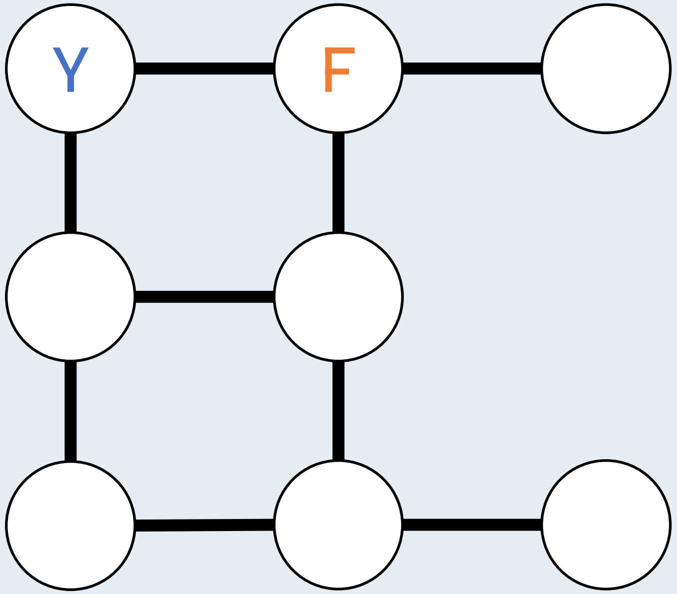
Goal: Find the earliest time that you can safely exit the maze, if possible.

```
#####
#FFF#
#FF##
#FFF#
 Y
```

Problem 2.a

Idea: represent the problem using a graph.

- Vertices: spots that are not walls.
- Edges: unweighted, undirected edge if two spots are adjacent to each other.



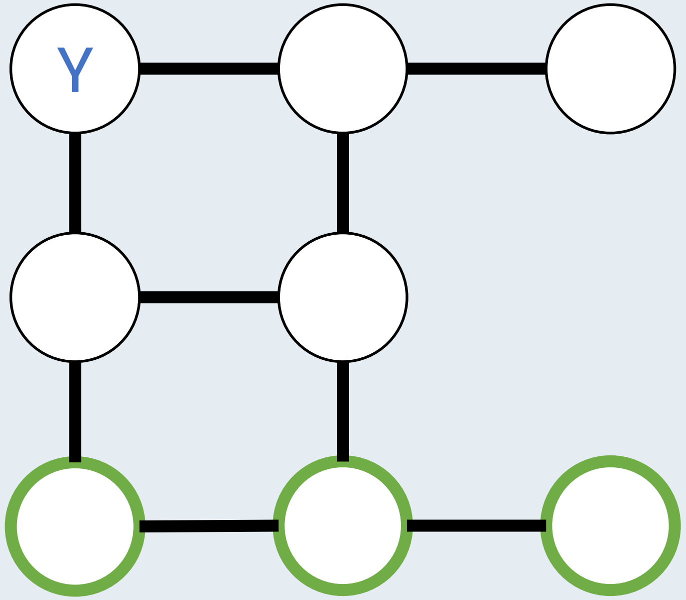

```
#####
#YF.#
#..##
#...#
```

Problem 2.b

Suppose there is no fire...

Equivalent problem: find the shortest path from the initial position to the exits.

Suitable algorithm: BFS on undirected and unweighted graph!



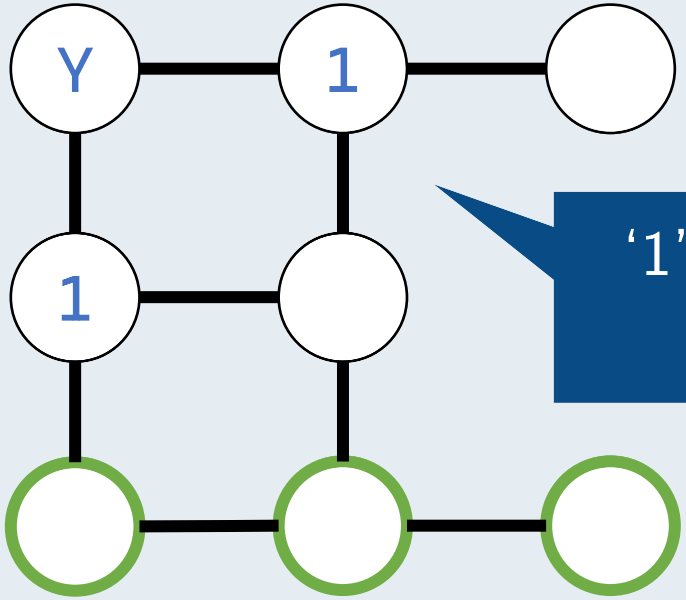

```
#####
#Y..#
#..##
#...#
```

Problem 2.b

Suppose there is no fire...

Equivalent problem: find the shortest path from the initial position to the exits.

Suitable algorithm: BFS on undirected and unweighted graph!



'1' fills the place you can reach at $t = 1$ min.

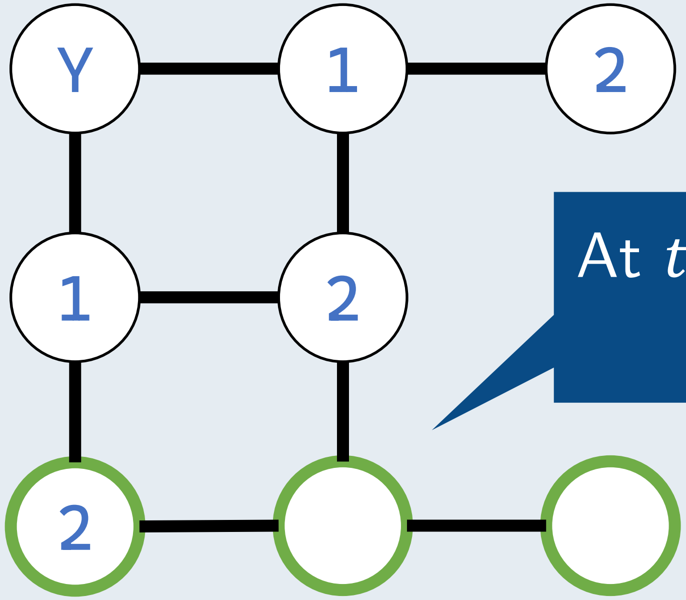
```
#####
#Y..#
#..##
#...#
```

Problem 2.b

Suppose there is no fire...

Equivalent problem: find the shortest path from your initial position to the exits.

Suitable algorithm: BFS on undirected and unweighted graph!



At $t = 2$ min, you can reach an exit vertex!

```
#####
#Y..#
#..##
#...#
```

Problem 2.b

Now there is fire. How to find when a spot will be on fire?

Equivalent problem: multi-source shortest path from initial positions of fires.

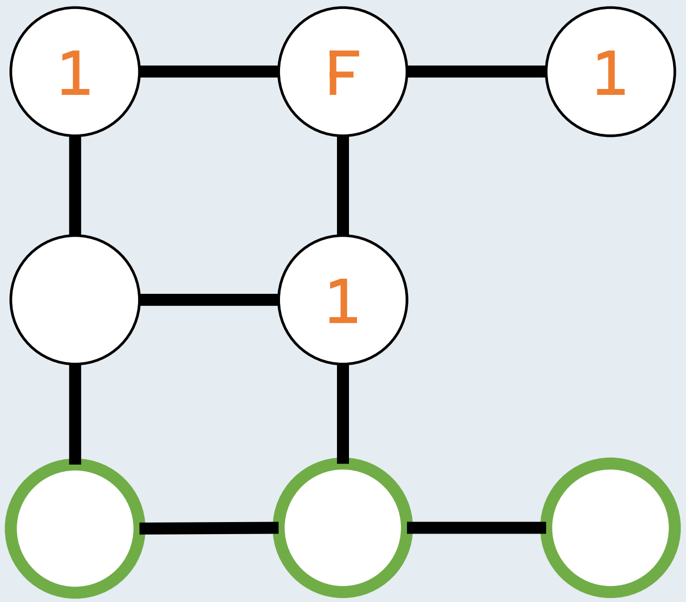
Suitable algorithm: still BFS on undirected and unweighted graph!

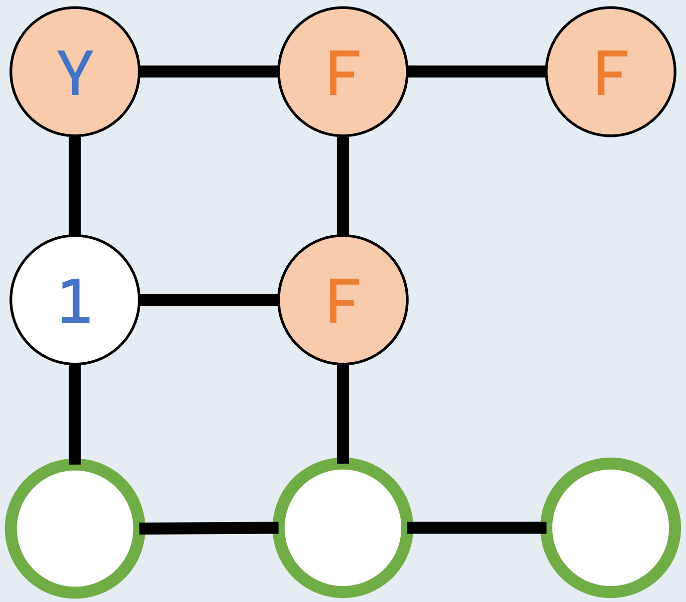
Problem 2.b

Now there is fire. How to find when a spot will be on fire?

Equivalent problem: multi-source shortest path from initial positions of fires.

Suitable algorithm: still BFS on undirected and unweighted graph!



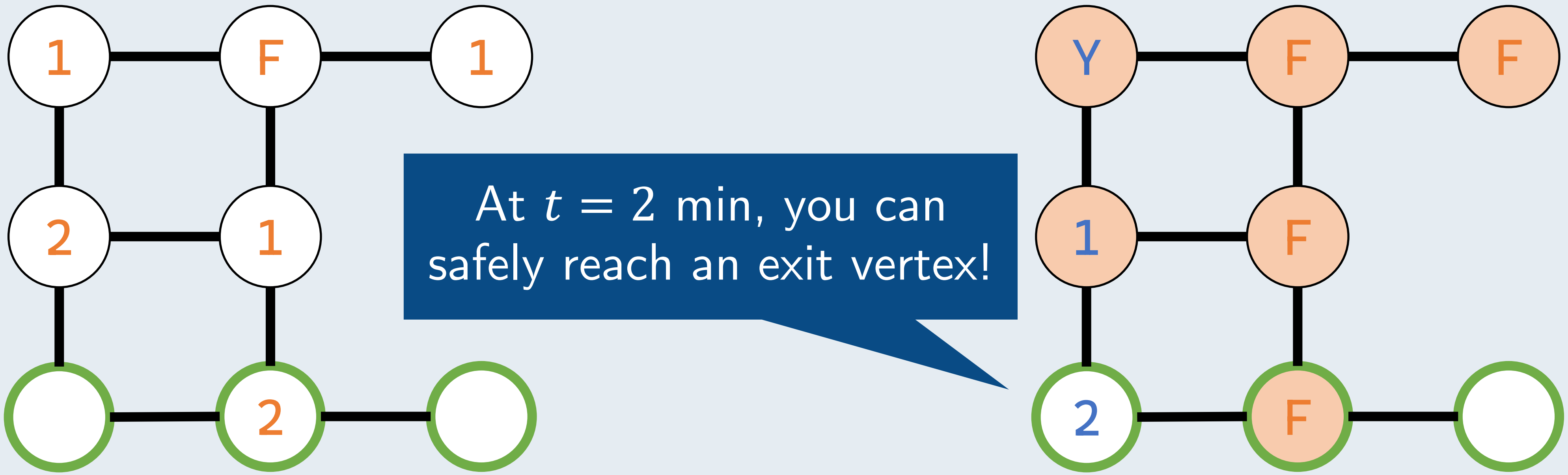


Problem 2.b

Now there is fire. How to find when a spot will be on fire?

Equivalent problem: multi-source shortest path from initial positions of fires.

Suitable algorithm: still BFS on undirected and unweighted graph!



Problem 3

- n currencies and m exchange rates.

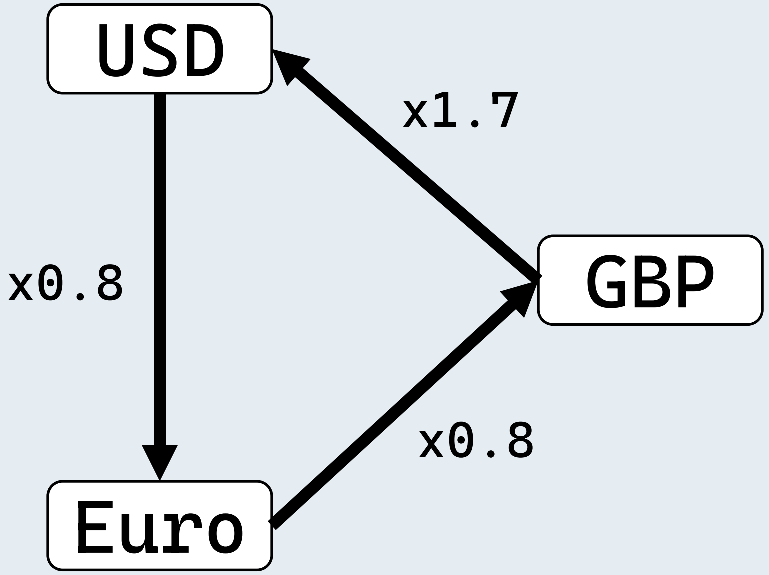
```
 1 USD = 0.8 Euro
1 Euro = 0.8 GBP
 1 GBP = 1.7 USD
```

- **Goal**: find if there is a way to end up at the same currency but with more money than we had at first.

Problem 3

- **Idea**: Model the problem as a graph.

```
1 USD = 0.8 Euro
1 Euro = 0.8 GBP
1 GBP = 1.7 USD
```

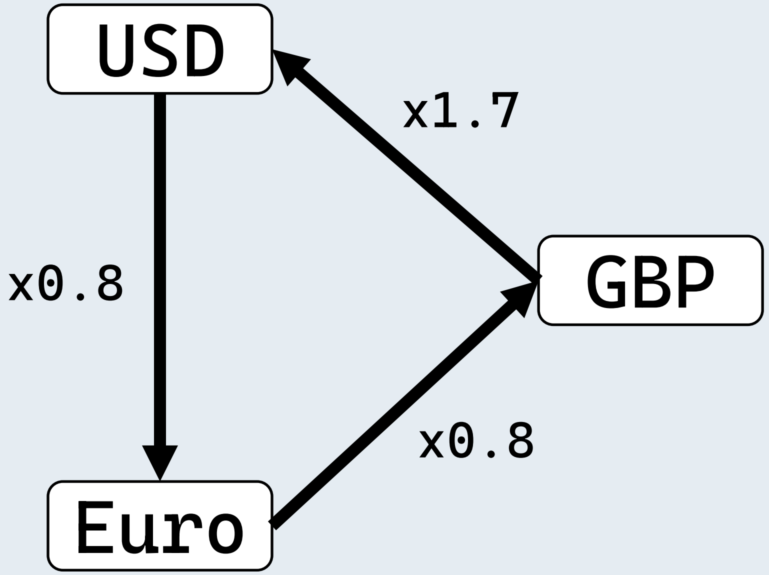


- Vertices: different currencies,
- Edges: directed and weighted edges representing exchange relationship.

Problem 3

Let's try! Suppose we have 1 USD. Will we make money going through this cycle?

$$1 \times 0.8 \times 0.8 \times 1.7$$
$$= 1.088 > 1$$



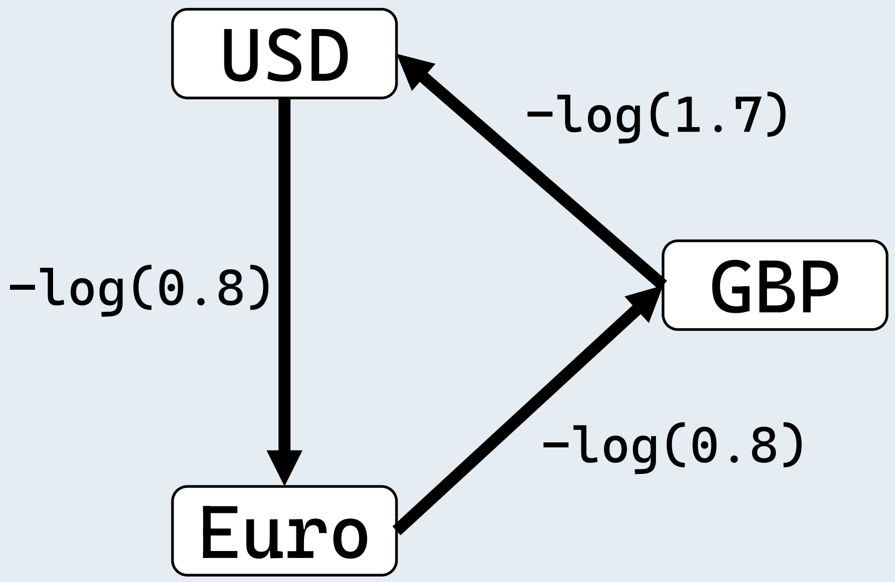
Equivalent problem: detect a cycle so that the product of edge weights is > 1.

Is there a way to find the sum instead?

Problem 3

Idea: Let the edge weight be negative log of exchange rate,

$$1 \times 0.8 \times 0.8 \times 1.7 > 1$$
$$\Leftrightarrow \log 0.8 + \log 0.8 + \log 1.7 > 0$$
$$\Leftrightarrow -\log 0.8 - \log 0.8 - \log 1.7 < 0$$



Equivalent problem: find if there is a negative cycle in the graph.

Use **Bellman-Ford** algorithm!

Problem 4

- N areas of different height, and $N-1$ descriptions like "area B_i higher than area A_i by N_i centimeters".

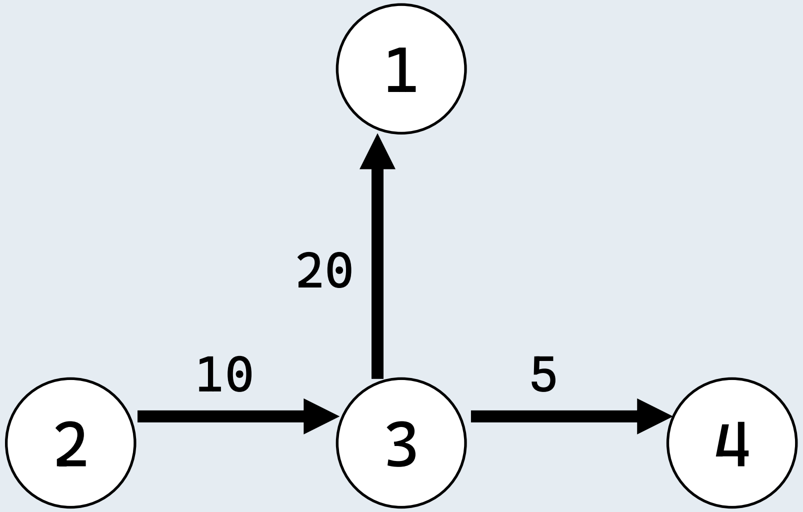
```
2 higher than 3 by 10cm
3 higher than 1 by 20cm
3 higher than 4 by 5 cm
```

- **Goal**: Answer Q queries of whether an area X is higher than an area Y, $1 \leq X, Y \leq N$.

Problem 4

- **Idea**: Represent the problem as a graph.

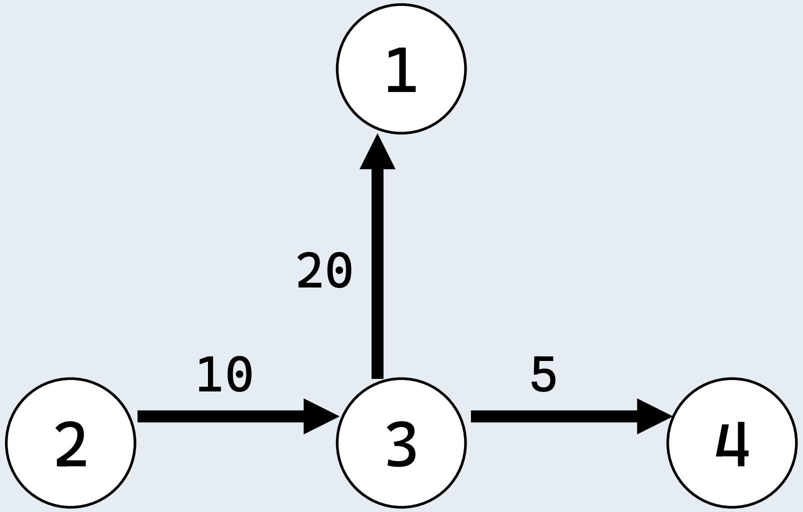
```
2 higher than 3 by 10cm
3 higher than 1 by 20cm
3 higher than 4 by 5 cm
```



- Vertices: different areas,
- Edges: weighted, directed. The weight represent relative height.

Problem 4

- **Example**: How to find if 2 is higher than 1?
- **Idea**: Can use BFS/DFS starting from 2, and find a path to 1.



Problem 4

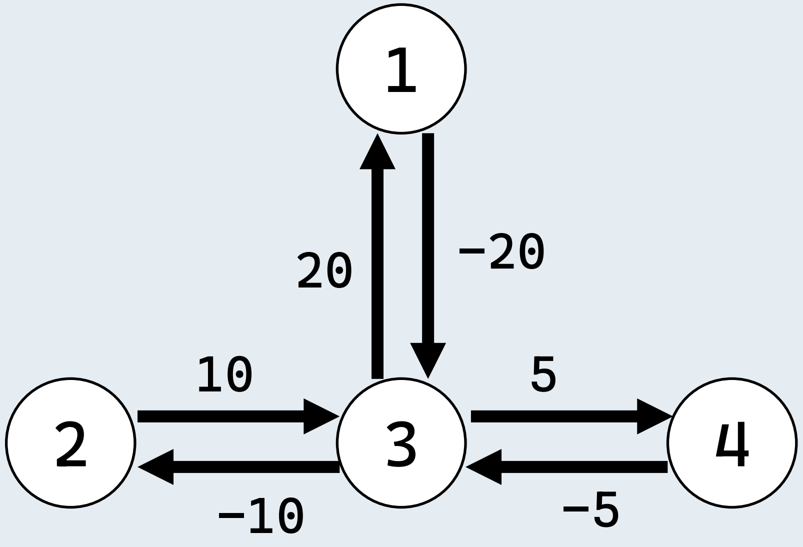
- **Example**: How to find if 1 is higher than 4?

No path between 1 and 4... BFS/DFS doesn't work!

- **Question**: How to link 1 and 4 together?
- **Idea**: We can also include edges in reverse directions.

Each query requires one BFS/DFS, in total $O(QN)$ time.

Can we do better?



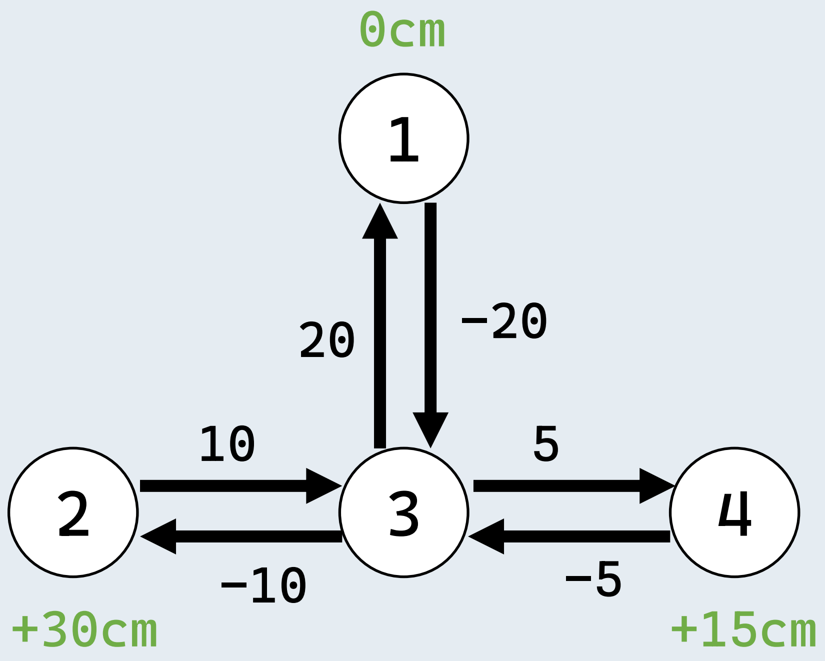
Problem 4

- **Example**: Suppose we already know that
  - 2 is higher than 1 by 30cm,
  - 4 is higher than 1 by 15cm,

Do we know the relative position of 2 and 4?

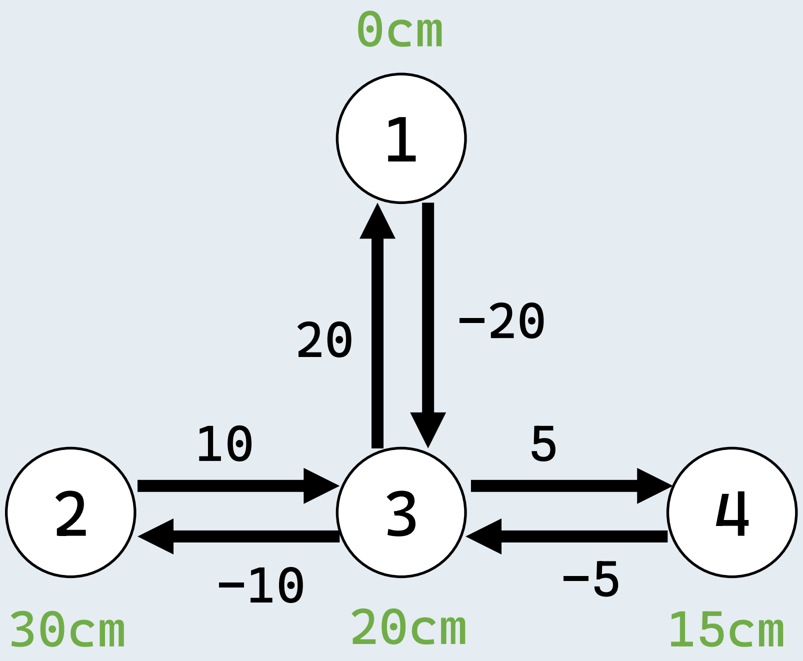
Idea: Simply find the relative position of all areas to one area (e.g. **1**)!

Equivalent problem: find the sum of weight on the path from all vertices to **1**.



Problem 4

- **Idea**: as we don't need to find the shortest path, we can just use BFS/DFS.
  - Traverse through all nodes starting from 1,
  - obtain the relative height and store in an array **`pos`**.
  - when we **`query(x, y)`**, simply return **`pos[x] - pos[y]`**.
- Pre-processing takes $O(N)$ time and each query will take $O(1)$ time. In total $O(N+Q)$ time.



End of File

Thank you very much for your attention :-)