NUS
National University of Singapore
School of Computing

Tutorial 11: Shortest Paths II

November 7, 2022

Gu Zhenhao

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

Shortest Path (Continued)

How to find the shortest path between two nodes?

Bellman-Ford Algorithm

```
for each node $v \in V$ do
        $D_0[v] \leftarrow \infty$;
$D_0[u] \leftarrow 0$;
for $i \leftarrow 1$ to $n$ do
        for each edge $(t, v) \in E$ do
                $D_i[v] \leftarrow \min\{D_{i-1}[v], D_{i-1}[t] + w[t, v]\}$;
```

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

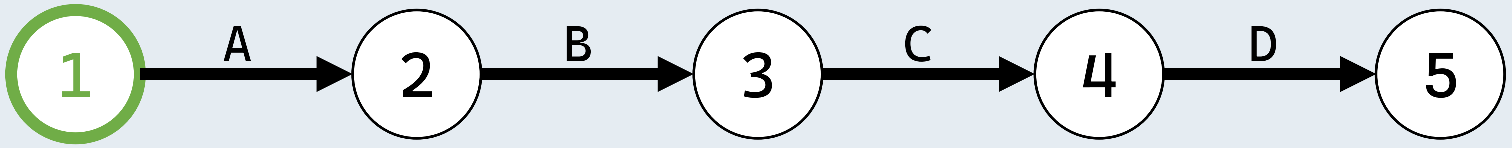
Bellman-Ford Algorithm

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

Does the order of edges here matter?

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

Bellman-Ford Algorithm

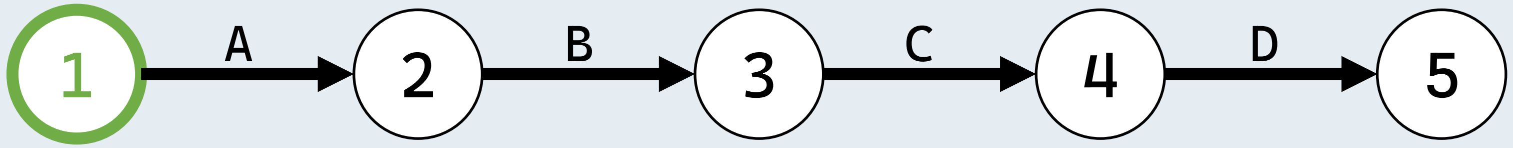


Iteration	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	1	∞	∞	∞
2	0	1	2	∞	∞
3	0	1	2	3	∞
4	0	1	2	3	4

If we relax the edges in the order of $D \to C \to B \to A$,

we will need **4 iterations** to complete the algorithm.

Bellman-Ford Algorithm

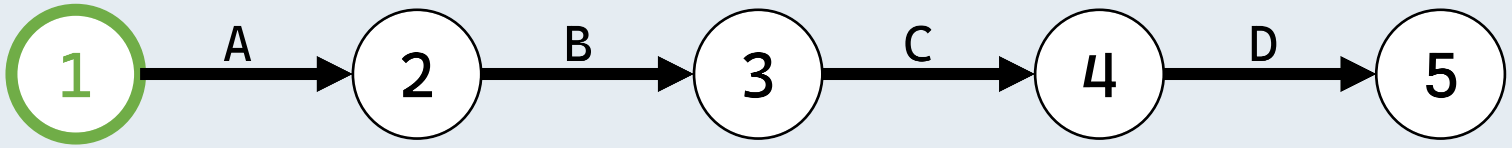


Iteration	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	1	2	∞	∞
2	0	1	2	3	∞
3	0	1	2	3	4
4	0	1	2	3	4

If we relax the edges in the order of $D \to C \to A \to B$,

we will need **3 iterations** to complete the algorithm.

Bellman-Ford Algorithm



Iteration	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	1	2	3	4
2	0	1	2	3	4
3	0	1	2	3	4
4	0	1	2	3	4

If we relax the edges in the order of $A \to B \to C \to D$,

we only need **1 iteration** to complete the algorithm.

Idea: relax the edges from vertices closest to the source vertex first!

Graph Traversal Revisited

- **Depth-First Search (DFS)**: Explore the closest nodes to <u>the last visited node</u> first. (LIFO)
- **Breadth-First Search (BFS)**: Explore the closest nodes to <u>the first visited node</u> first. (FIFO)
- **Dijkstra's Algorithm**: Explore the nodes that <u>have shortest path to the source so far</u> first.

Graph Traversal Revisited

For DFS: D is a stack
For BFS: D is a queue

```
traversal(G, v):
    D.push(v);
    while D is not empty do
        u = D.pop();
        if u is not visited then
            visit(u);
            for each neighbor v of y do
                if v is not visited then
                    D.push(v);
```

Dijkstra’s Algorithm

```
Dijkstra(G, v):
    D.insert(<0, v>); D.insert(<∞, all other nodes>);
    while D is not empty do
        <d, u> = D.extractMin();
        if u is not visited then
            for each neighbor v of u do
                if D[v] > D[u] + w[u,v] then
                    D[v] = D[u] + w[u,v]; v.pred = u;
                    D.update(<D[v], v>);
```

For Dijkstra’s: D is a min heap.

Dijkstra's Algorithm

```
Dijkstra(G, v):
    D.insert(<0, v>); D.insert(<∞, all other nodes>);
    while D is not empty do
        <d, u> = D.extractMin();
        if u is not visited then
            for each neighbor v of u do
                if D[v] > D[u] + w[u,v] then
                    D[v] = D[u] + w[u,v]; v.pred = u;
                    D.update(<D[v], v>);
```

u won't be revisited if there is no negative edge.

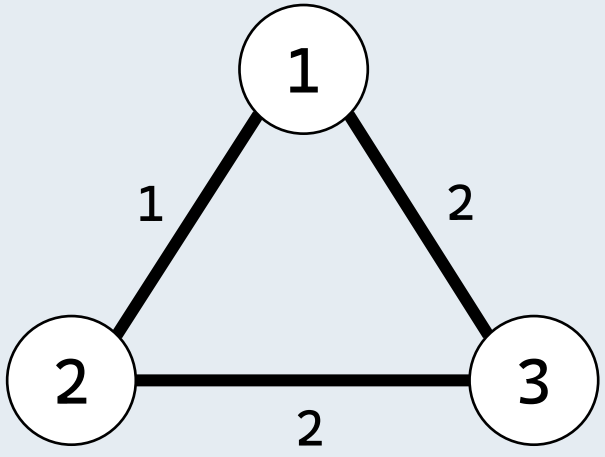
Dijkstra's Algorithm: Modified

```
ModifiedDijkstra(G, v):
    D.insert(<0, v>); D.insert(<∞, all other nodes>);
    while D is not empty do
        <d, u> = D.extractMin();
        if d == D[u] then
            for each neighbor v of u do
                if D[v] > D[u] + w[u,v] then
                    D[v] = D[u] + w[u,v]; v.pred = u;
                    D.insert(<D[v], v>);
```

Only consider the last inserted pair in D.

Problem 1.a

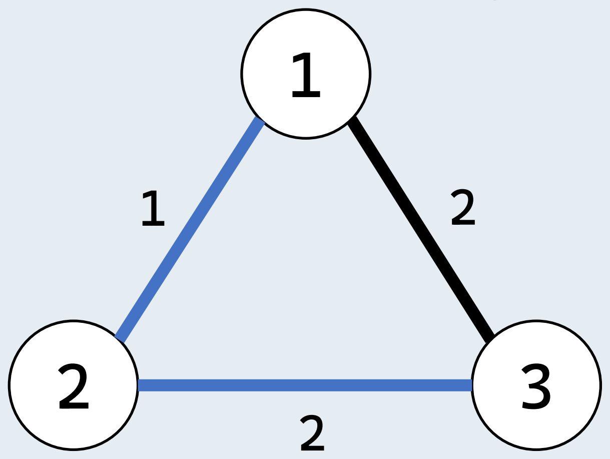
True or False: *If an undirected graph has a unique minimum spanning tree, and we run Prim's or any single source shortest path algorithm from the same source, the final result of edges chosen will form the same spanning tree.*



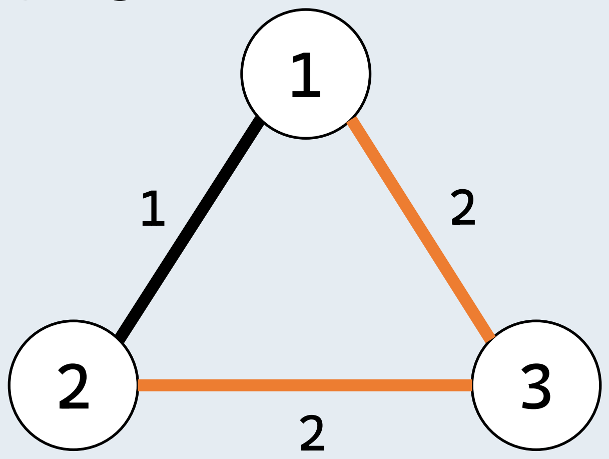
Problem 1.a

True or False: *If an undirected graph has a unique minimum spanning tree, and we run Prim's or any single source shortest path algorithm from the same source, the final result of edges chosen will form the same spanning tree.*

From vertex 3...



Minimum Spanning Tree

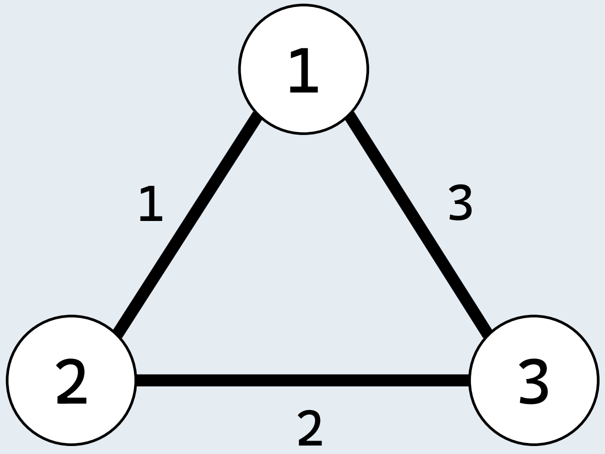


Shortest path Spanning Tree

False. The minimum spanning tree is not necessarily the same as shortest path spanning tree.

Problem 1

True or False: *If the weights of all edges in a positively weighted graph are unique, there is always a unique shortest path (the smallest total cost is unique) from a source to a destination in such a graph*



False. For example, here both $1 \to 2 \to 3$ and $1 \to 3$ are shortest paths from 1 to 3.

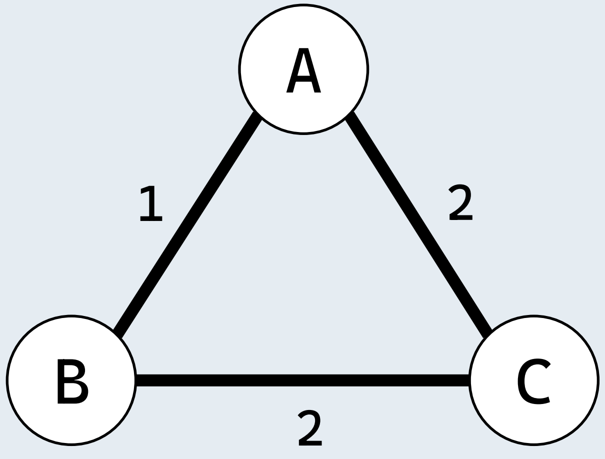
Problem 1.c

True or False: *A connected, undirected graph will always form a shortest path tree with* $V-1$ *edges.*

True. In the subgraph we obtained from shortest path finding algorithm, for every vertex, there is only one path to the source. So what we get is a tree with $V-1$ edges.

Problem 1.d

True or False: *The shortest path from any vertex A to any vertex B in the combination of the shortest path from A to any vertex C and the shortest path from C to B.*



False. The shortest path from A to B doesn't necessarily go through vertex C.

Question: What if I know that the shortest path from A to B passes through C?

Problem 1.e

True or False: *In Dijkstra's algorithm, we can replace the min heap with a set of vertices.*

True. But finding the item with minimum value in the set will take $O(|V|)$ time, while updating the value takes in average $O(1)$ time.

Variations of SSSP

How to modify shortest path finding algorithms to solve certain problems?

Modifications on Dijkstra's Algorithm

Common Modifications:

```
Dijkstra(G, v):
    D.insert(<0, v>); D.insert(<∞, all other nodes>);
    while D is not empty do
        <d, u> = D.extractMin();
        for each neighbor v of u do
            if D[v] > D[u] + w[u,v] then
                D[v] = D[u] + w[u,v]; v.pred = u;
                D.update(<D[v], v>);
```

Modifications on Dijkstra's Algorithm

Common Modifications:

1. Keys that are used in the priority queue D.
   How would you like to rank your edges? Is there anything else you want to minimize (maximize)?

```
Dijkstra(G, v):
    D.insert(<0, v>); D.insert(<∞, all other nodes>);
    while D is not empty do
        <d, u> = D.extractMin();
        for each neighbor v of u do
            if D[v] > D[u] + w[u,v] then
                D[v] = D[u] + w[u,v]; v.pred = u;
                D.update(<D[v], v>);
```

Modifications on Dijkstra's Algorithm

Common Modifications:

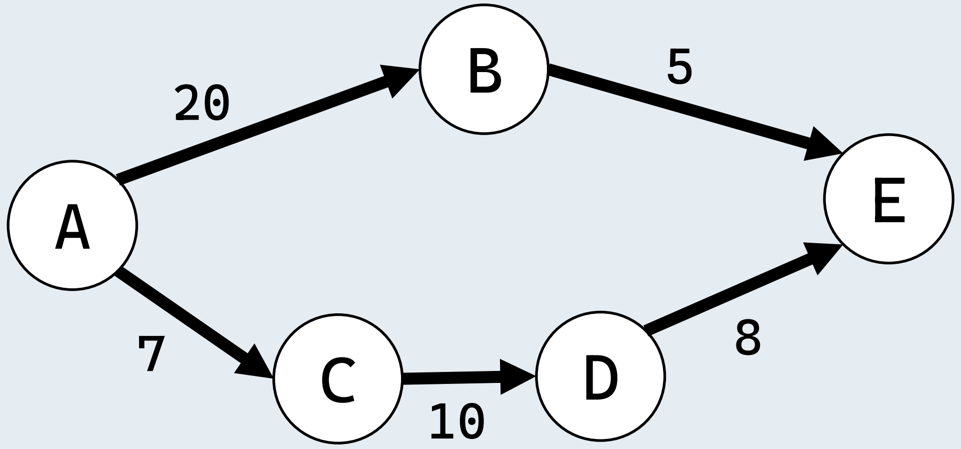
1. Keys that are used in the priority queue D.
2. **Conditions** for edge relaxation.

   When can we set predecessor of v to be u?

```
Dijkstra(G, v):
    D.insert(<0, v>); D.insert(<∞, all other nodes>);
    while D is not empty do
        <d, u> = D.extractMin();
        for each neighbor v of u do
            if D[v] > D[u] + w[u,v] then
                D[v] = D[u] + w[u,v]; v.pred = u;
                D.update(<D[u] + w[u,v], v>);
```

Problem 2

Goal: *Find the shortest path from one node to another node. If there are multiple paths with same weight, choose the one containing fewer vertices.*



Problem 2

Recall: Dijkstra's algorithm only takes distance from source into account.

```
Dijkstra(G, v):
    D.insert(<0, v>); D.insert(<∞, all other nodes>);
    while D is not empty do
        <d, u> = D.extractMin();
        for each neighbor v of u do
            if D[v] > D[u] + w[u,v] then
                D[v] = D[u] + w[u,v]; v.pred = u;
                D.update(<D[u] + w[u,v], v>);
```

Problem 2

Idea: Add an attribute in keys, that denotes **the number of vertices in the shortest path**.

```
Dijkstra(G, v):
    D.insert(<(0,0), v>); D.insert(<(∞,∞), all other nodes>);
    while D is not empty do
        <(d,h), u> = D.extractMin();
        for each neighbor v of u do
            if D[v] > D[u] + w[u,v] then
                D[v] = D[u] + w[u,v]; v.pred = u; H[v] = H[u] + 1;
                D.update(<(D[v], H[v]), v>);
```

How to compare 2 keys here?

Problem 2

Idea: modify the conditions, so that **if there are multiple shortest paths of same weight, we take the one with fewer vertices**.

```
Dijkstra(G, v):
    D.insert(<(0,0), v>); D.insert(<(∞,∞), all other nodes>);
    while D is not empty do
        <(d,h), u> = D.extractMin();
        for each neighbor v of u do
            if (D[v] > D[u] + w[u,v] or
                (D[v] == D[u] + w[u,v] and H[v] > H[u] + 1)) then
                D[v] = D[u] + w[u,v]; v.pred = u; H[v] = H[u] + 1;
                D.update(<(D[v], H[v]), v>);
```

Applications

How to utilize the shortest path finding algorithms?

Problem 3

- We have an $m \times n$ grid, `#` represent a wall while `.` represents a passable spot.
- A robot starts at top-left corner `(0, 0)` and facing **`east`**. It can choose to:
  1. Move forward by 1 cell, which takes 3 seconds,
  2. Turn right, which takes 2 seconds.

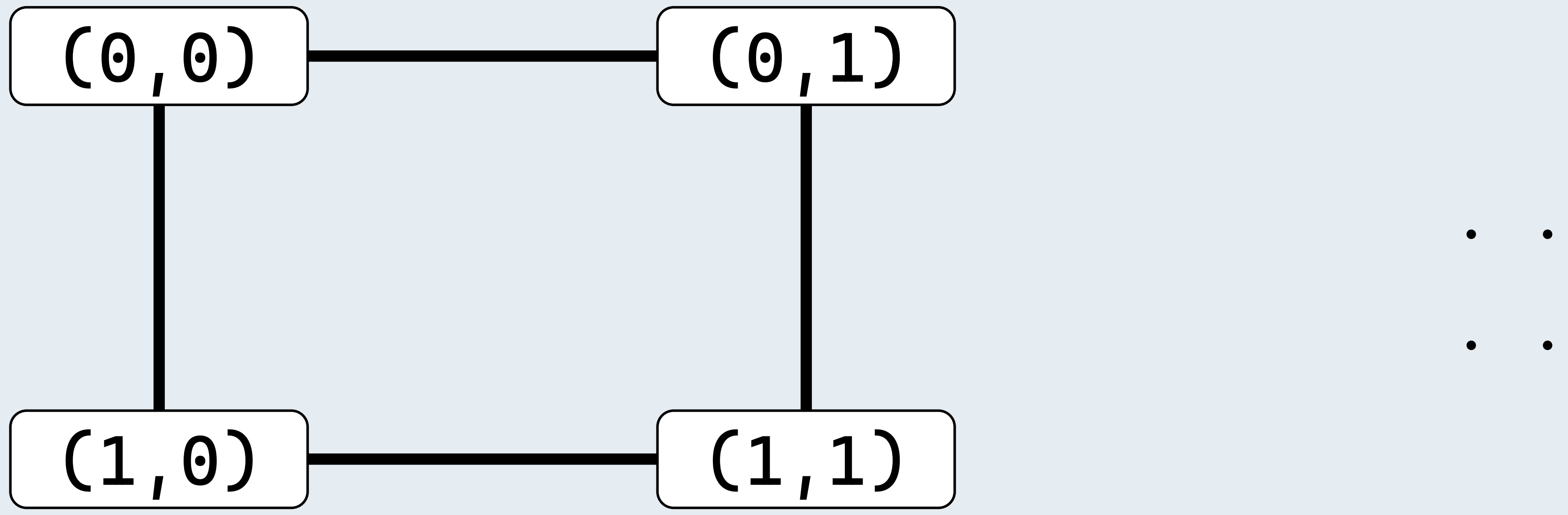
Goal: find the shortest time to reach bottom-right corner `(m-1, n-1)`.

```
. .
. .
```

Problem 3

Attempt: represent the problem as a graph,

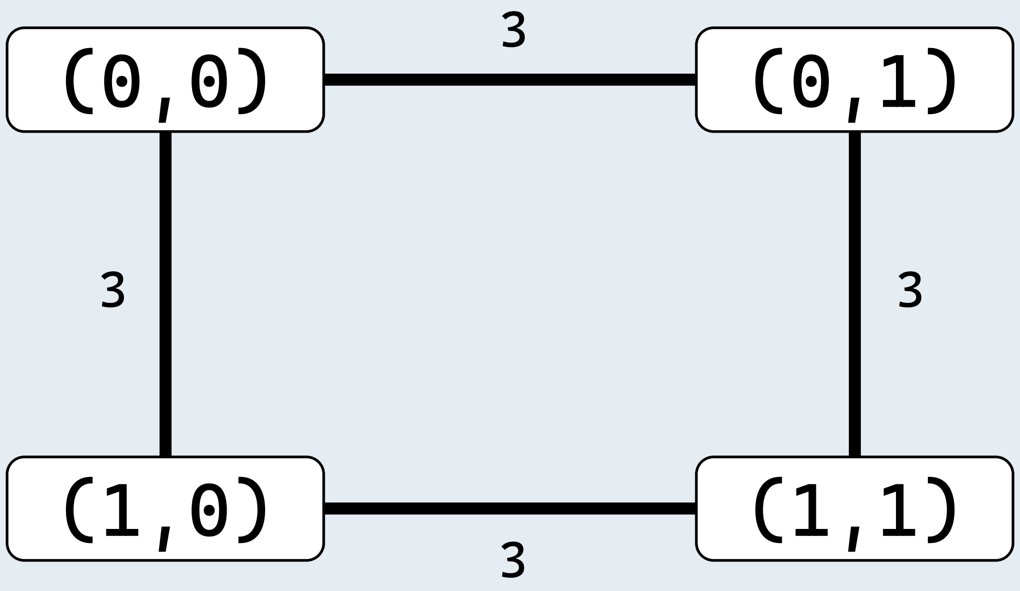
- Vertices: passable cell.
- Edges: represents we can reach from a spot to another spot.

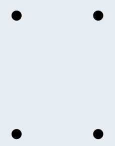


Problem 3

Questions:

1. What are the edge weights?
2. How to represent the current direction we are facing?

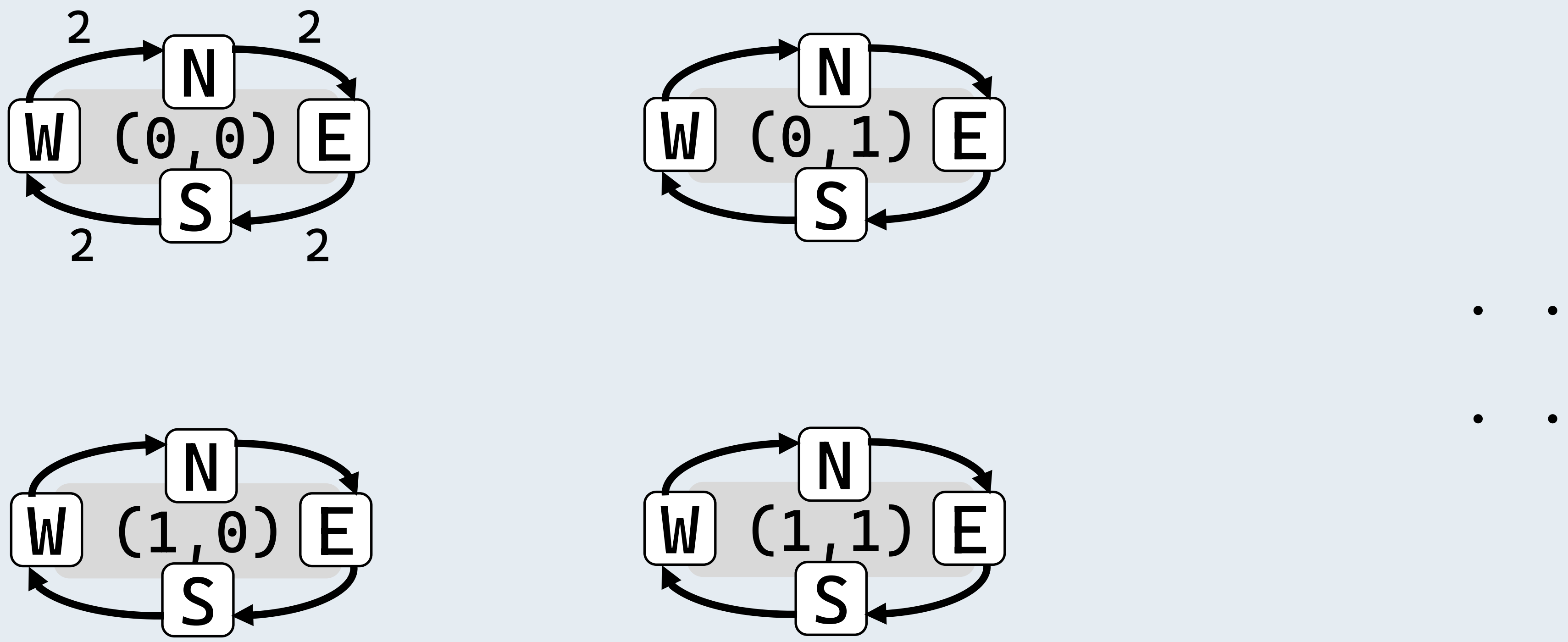




Problem 3

Idea: take both the robot's position and direction into account.

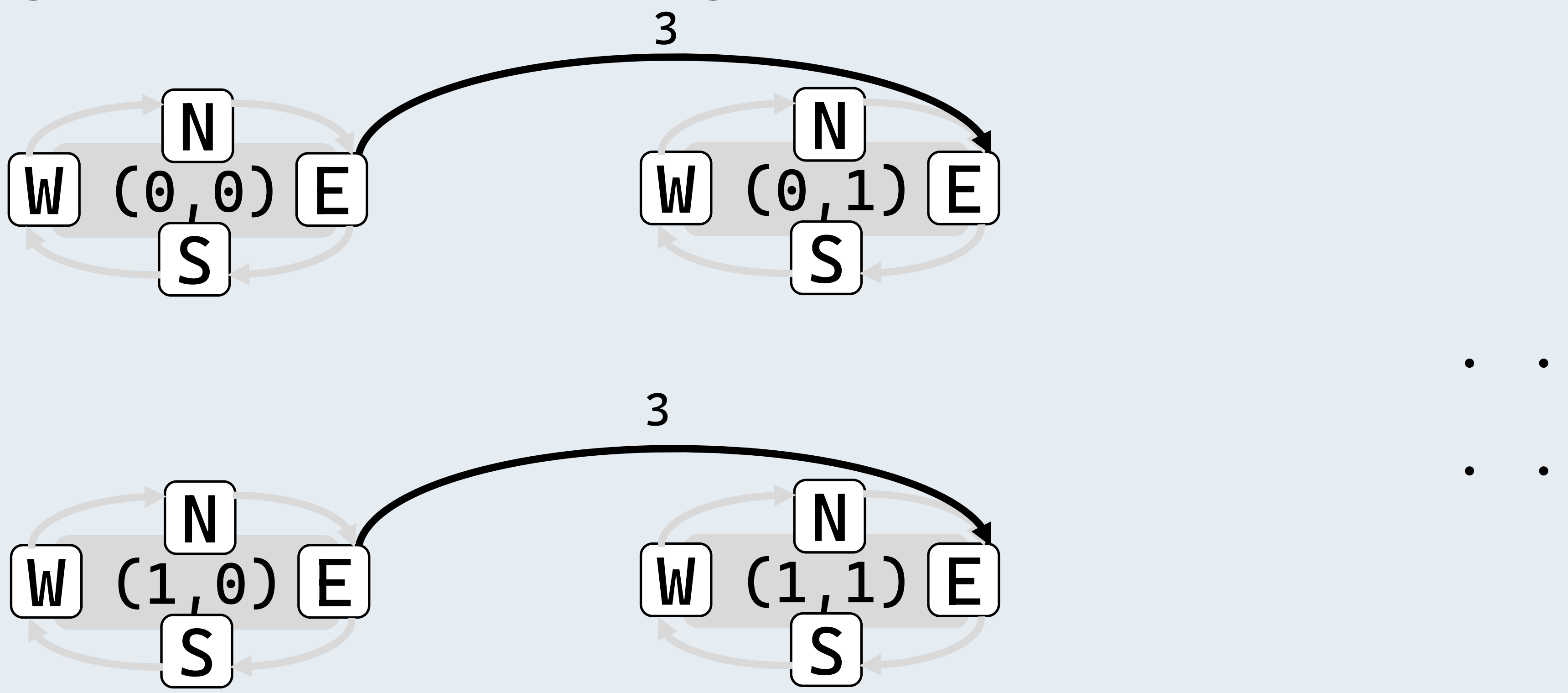
- Vertices: robot's state `<position, direction>`.
- Edges: whether we can change from one state to another state.



Problem 3

Idea: take both the robot's position and direction into account.

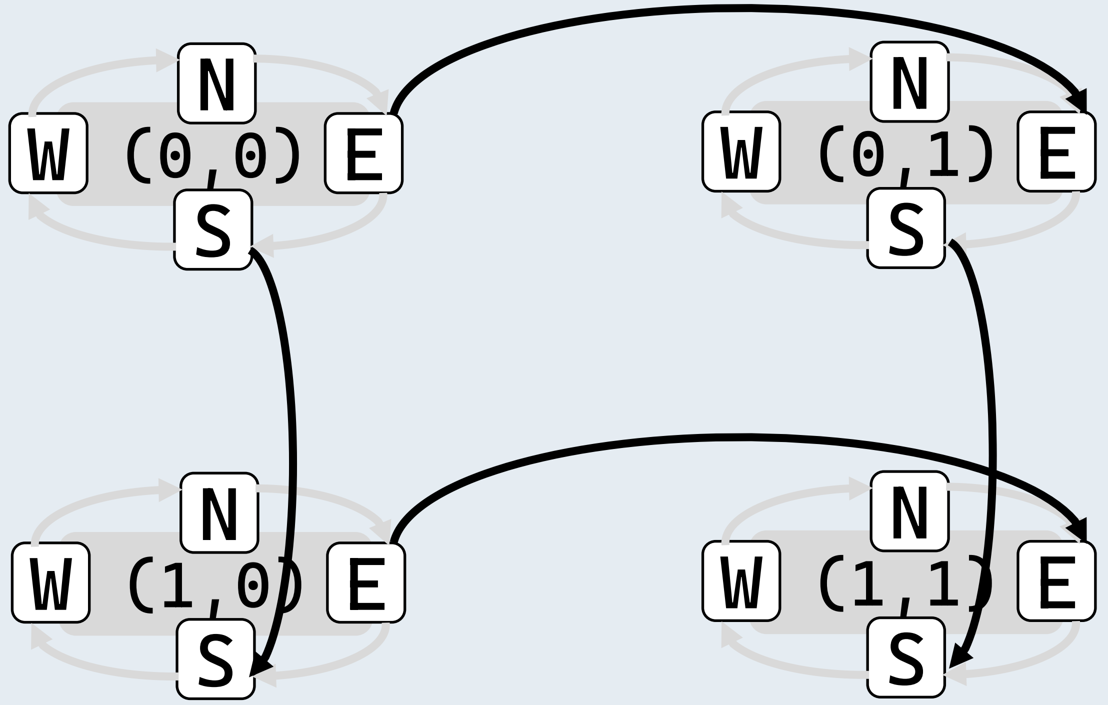
- Vertices: robot's state `<position, direction>`.
- Edges: whether we can change from one state to another state.



Problem 3

Idea: take both the robot's position and direction into account.

- Vertices: robot's state **`<position, direction>`**.
- Edges: whether we can change from one state to another state.



Number of vertices: $4mn$.

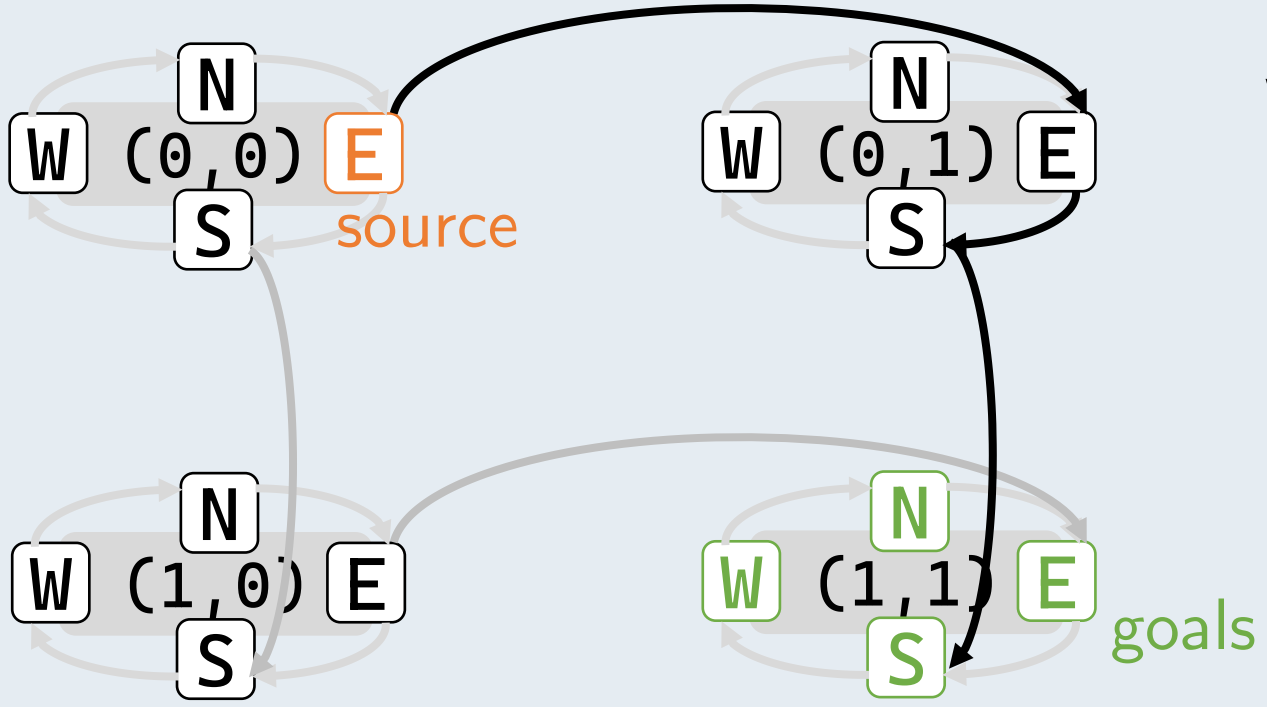
Number of edges: each vertex has 2 outgoing edges, $8mn$.

** We omit some edges here.*

Problem 3

Idea: take both the robot's position and direction into account.

Equivalent problem: SSSP from the source vertex `<(0, 0), east>` to one of the four vertices at `(m-1, n-1)`.



Weighted and directed graph.

Use (modified) Dijkstra's algorithm!

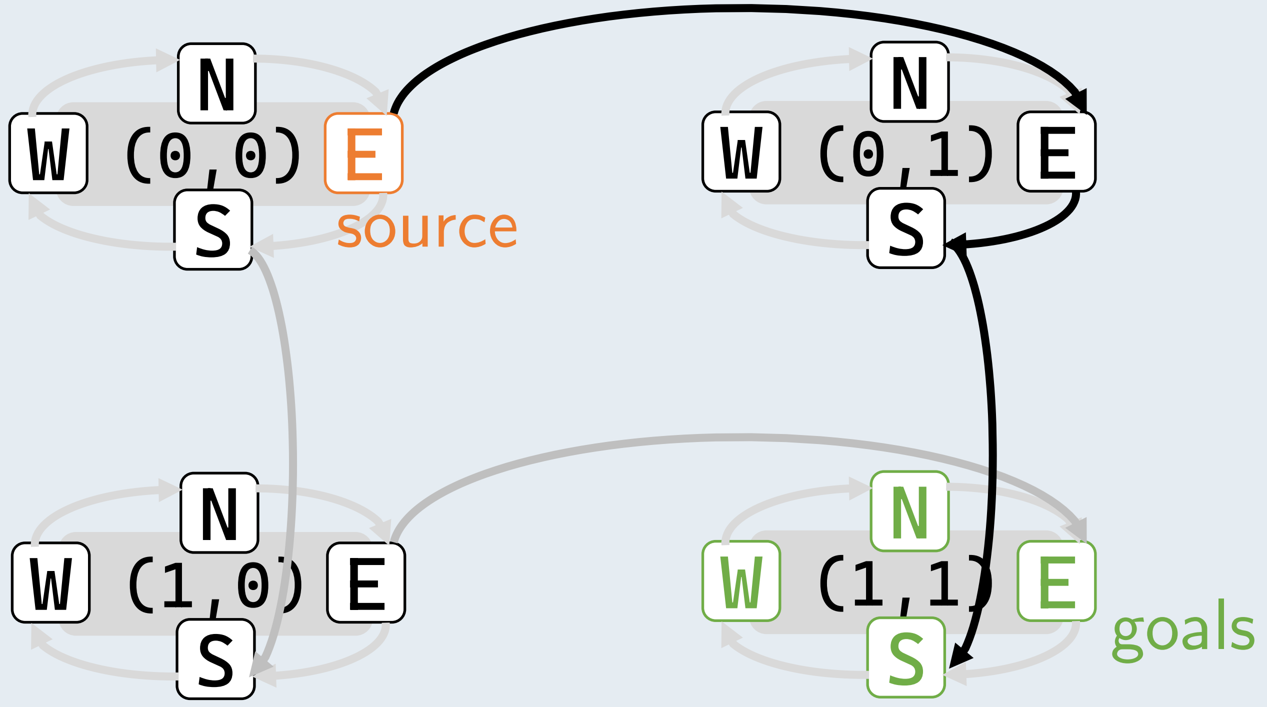
Total time: $O(mn \log(mn))$.

** We omit some edges here.*

Problem 3

Idea: take both the robot's position and direction into account.

Equivalent problem: SSSP from the source vertex `<(0, 0), east>` to one of the four vertices at `(m-1, n-1)`.



Question:

- Can we use BFS instead?
- Can we turn this into an unweighted graph?

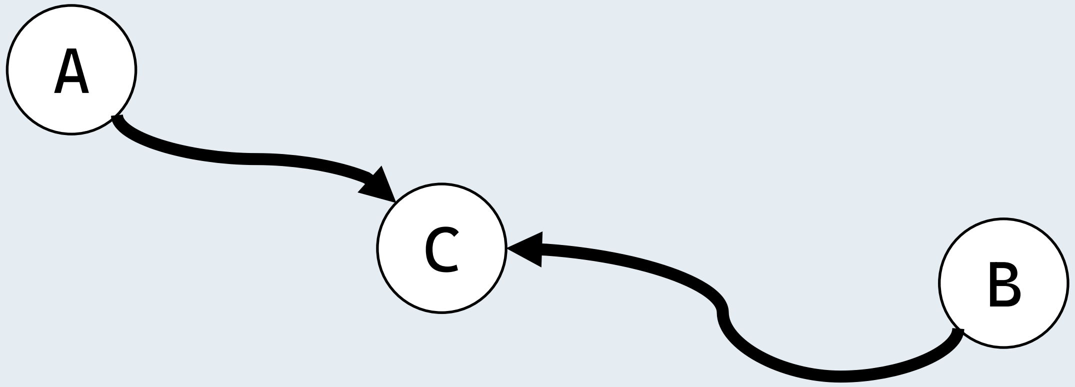
** We omit some edges here.*

Problem 4

Two friends live in cities A and B respectively.

Goal: find a city C such that

- Sum of shortest distance $A \to C$ and $B \to C$ is minimized. (primary concern)
- Difference in the two distances is minimized.

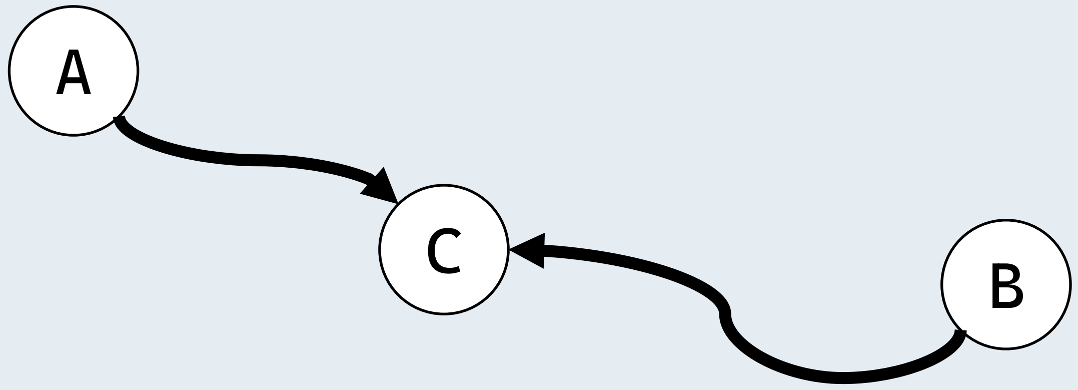


Problem 4

Idea: for each city,

- Find their shortest path to A and B, respectively,
- Calculate the sum $d_A + d_B$ and difference $|d_A - d_B|$.

Find the city with minimum sum!



Problem 5.a

Which of the following properties of a Binary Search Tree (BST) is **false**?

A. The right descendants of the root will contain keys larger than the key of the root.

True. by the properties of BST.

B. Every node of a BST will have two child nodes.

False. leaf nodes don't have children.

C. The time complexity of inserting into a BST is $O(n)$ time.

True. consider worst case where BST is not balanced.

D. The left child of the root has a key that is smaller than the key of the root.

True. By the properties of BST.

Problem 5.b

Consider a connected, undirected graph G which can have two or more cycles, and a vertex X in the graph. Which of the following statements is **true**?

A. G will **always** have more than one possible minimum spanning tree.

False. If edge weights are unique, MST is unique.

B. The shortest paths from X to all other vertices are **always** unique.

False. Can find a counter example.

C. We can obtain a spanning tree of G in $O(V + E)$ time.

True. We can use DFS/BFS to find a spanning tree.

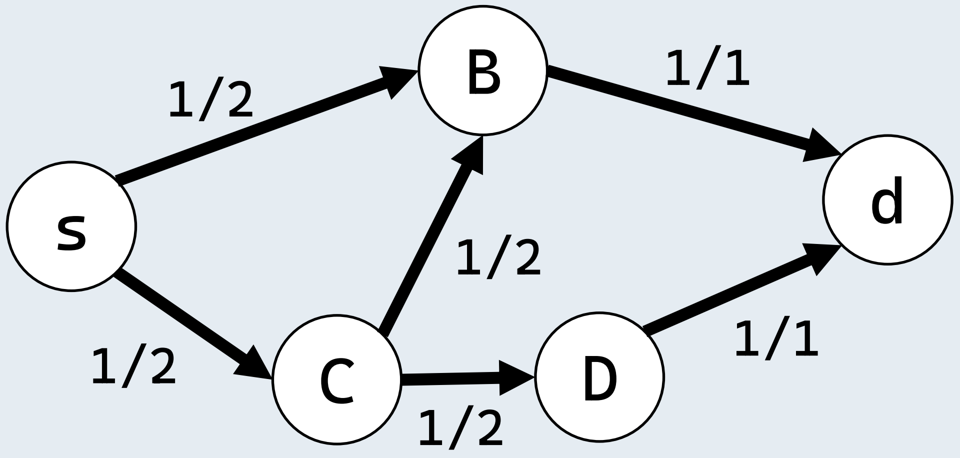
D. Bellman-Ford algorithm applied on G from X will **never** produce the correct shortest paths.

False. As long as we don't have negative cycles, Bellman-Ford works.

Problem 5.c

- V vertices and E edges,
- Source vertex s and destination vertex d,
- Start with some number K, each time we move from vertex u to v, we divide K by number of outgoing edges of u.

Goal: find minimum starting value of K such that when we reach d, $K > 0$.

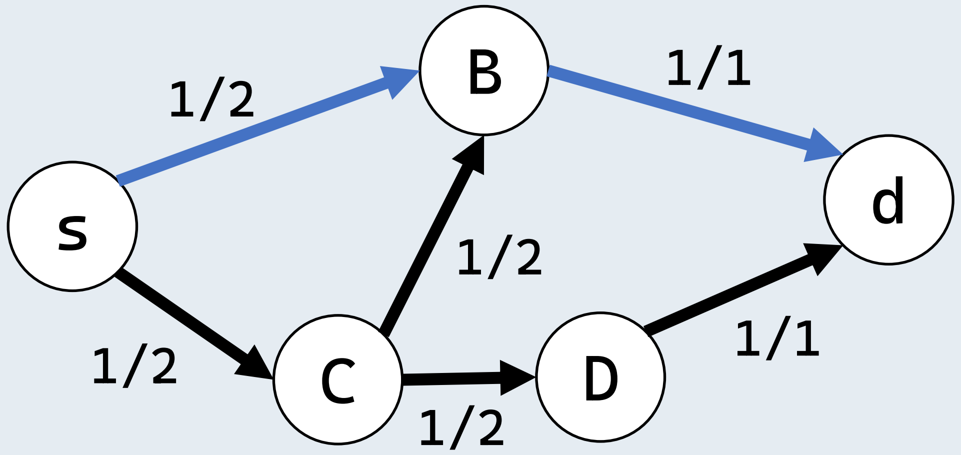


Problem 5.c

Goal: find minimum starting value of K such that when we reach d, $K > 0$.

Alternative goal: find the path from s to d that has the largest product (or smallest product of inverse).

Idea: Use Dijkstra's algorithm, but replace the addition of edge weights to product of edge weights.

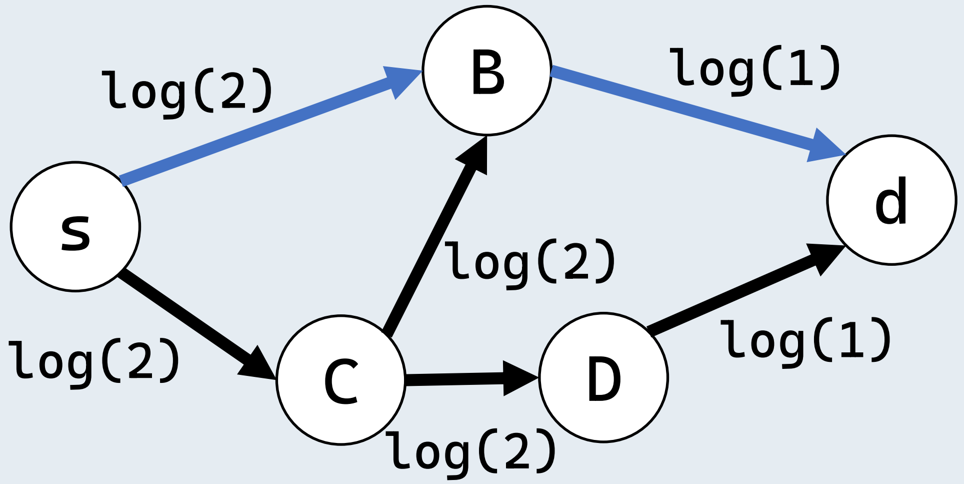


Problem 5.c

Goal: find minimum starting value of K such that when we reach d, $K > 0$.

Alternative goal: find the path from s to d that has the largest product (or smallest product of inverse).

Alternative Idea: Use Dijkstra's algorithm, but replace the edge weights with their logarithms.



End of Tutorials

Thank you and good luck in the finals! :-D