Dijkstra's Algorithm

09114319: Data Structures and Algorithms

Ratthaprom PROMKAM, Dr. rer. nat.

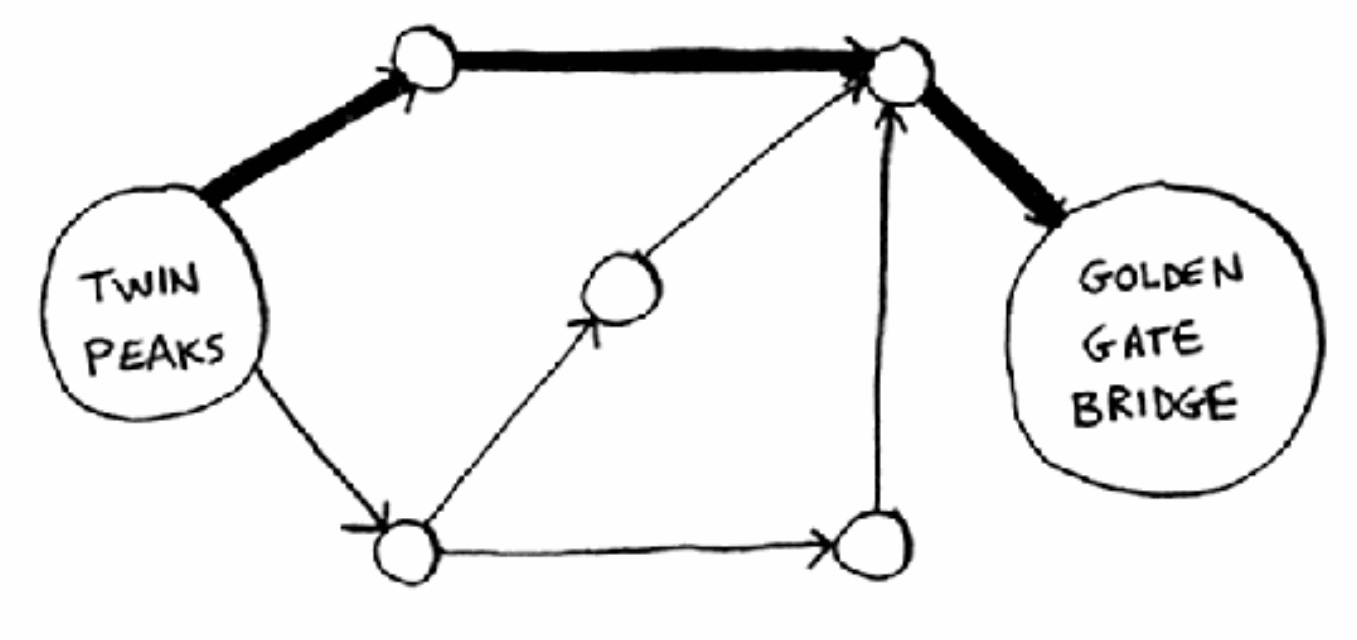
Department of Mathematics and Computer Science, RMUTT

Outline

- Weighted graphs
- Dijkstra's algorithm:
  - Find the shortest path in a weighted graph.
- Cycles in graphs:
  - Dijkstra's algorithm doesn't work with negative weight cycles

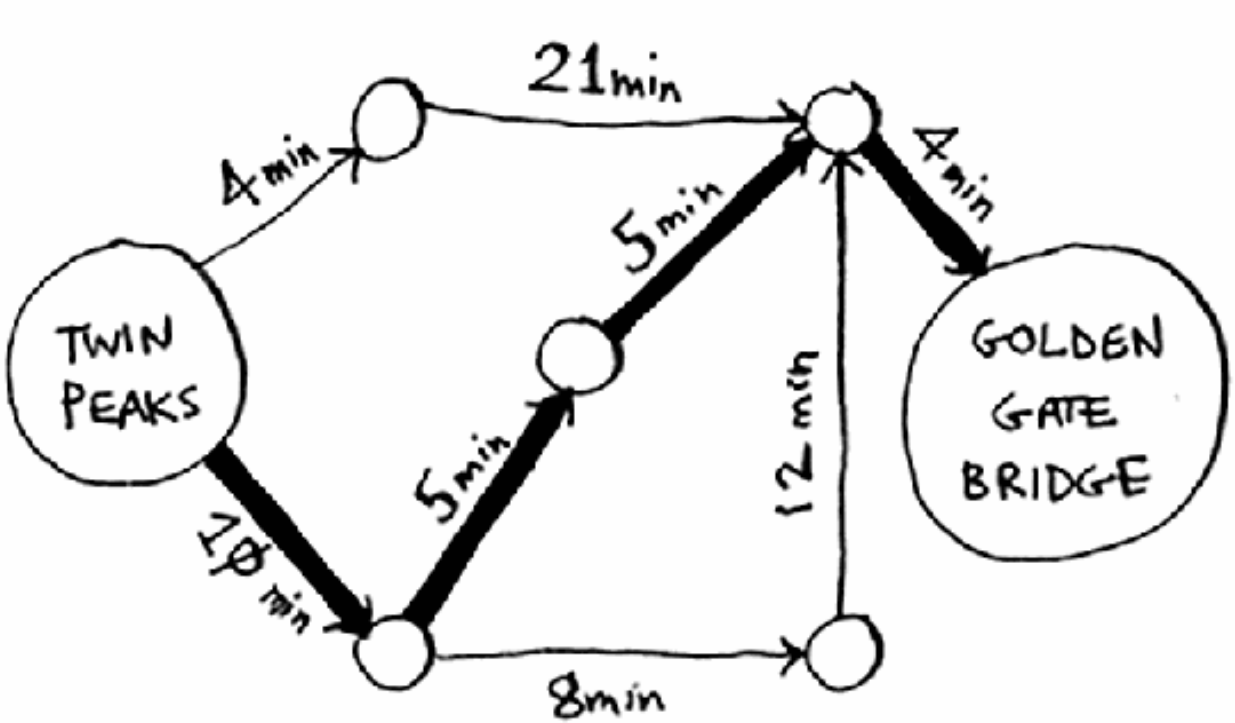
Shortest Path

- Breadth-first search will find you the path with the fewest segments.



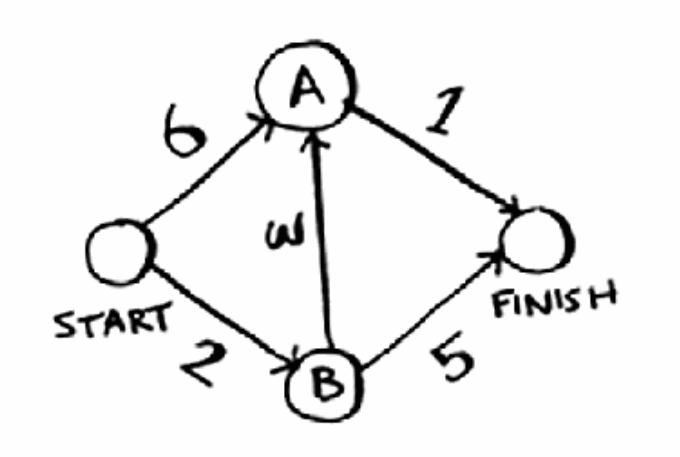
Fastest Path

- What if each segment has a travel time in minutes?
- You may need another algorithm to go from start to finish in the shortest possible time.

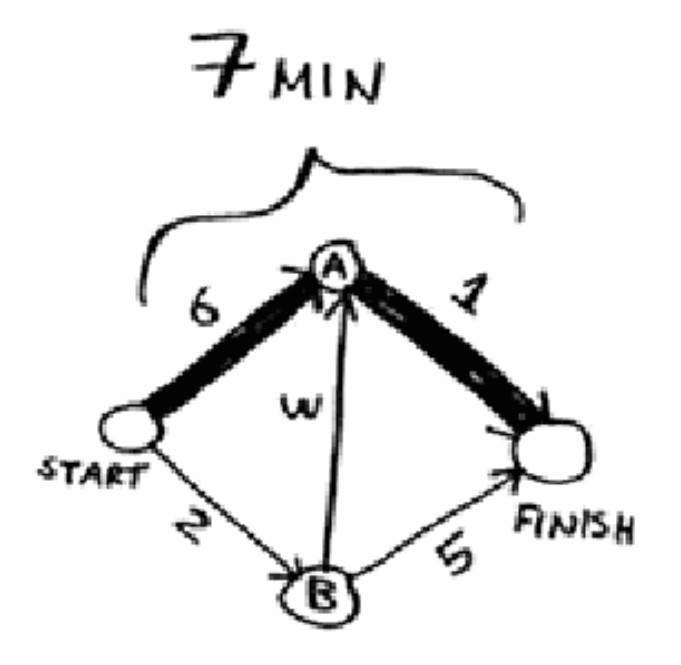


Working with Dijkstra's algorithm

Find the way to go from start to finish in the shortest possible time.



✎ BFS gives us this shortest path:



Working with Dijkstra's algorithm

Dijkstra's Algorithm

There are four steps to this algorithm:

1. Find the 'cheapest' node. This is the node you can get to in the least amount of time.
2. Update the costs of the neighbors of this node.
3. Repeat until you have done this for every node in the graph.
4. Calculate the final path.

Working with Dijkstra's algorithm

Step 1: Find the cheapest node.

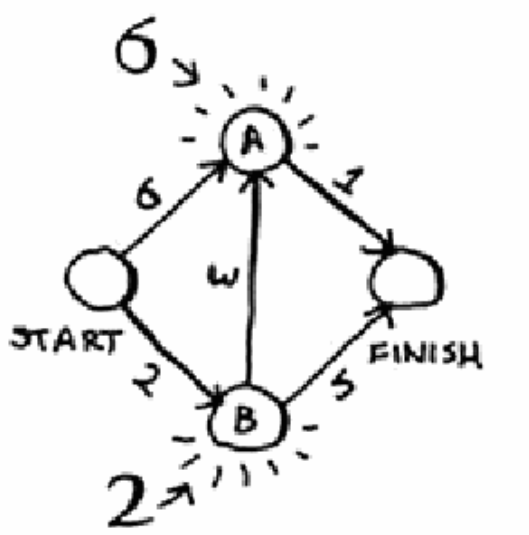
- ✎ You are standing at the start, wondering if you should go to node A or node B.

Working with Dijkstra's algorithm

Step 1: Find the cheapest node.

- ✎ You are standing at the start, wondering if you should go to node A or node B.
- ✎ How long does it take to get to each node?

NODE	TIME TO NODE
A	6
B	2
FINISH	∞

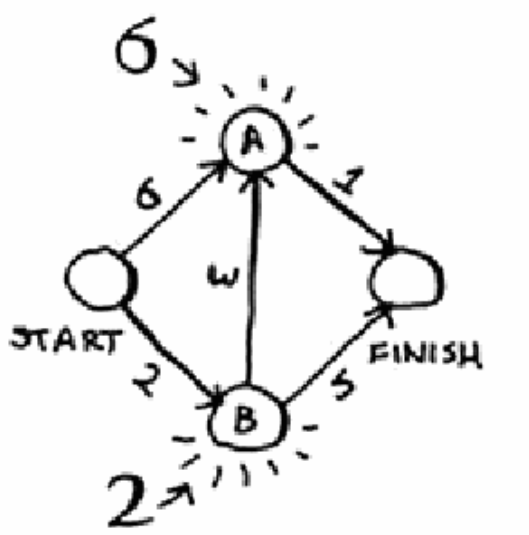


Working with Dijkstra's algorithm

Step 1: Find the cheapest node.

- ✎ You are standing at the start, wondering if you should go to node A or node B.
- ✎ How long does it take to get to each node?

NODE	TIME TO NODE
A	6
B	2
FINISH	∞

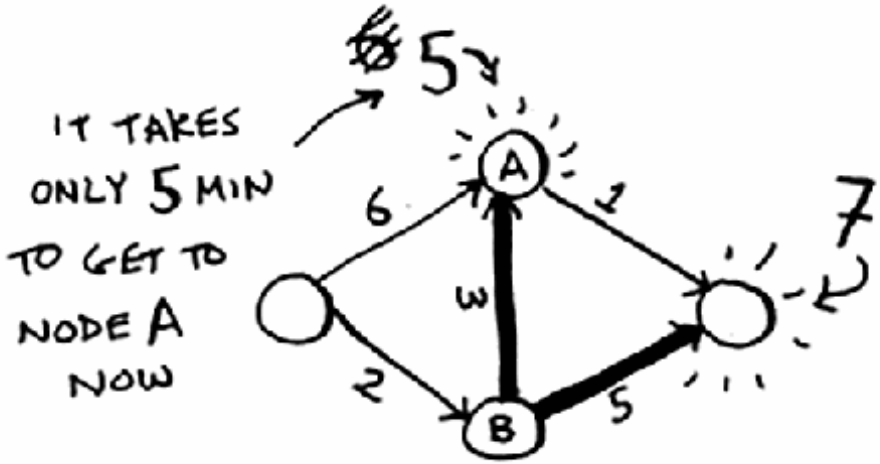


- ✎ Node **B** is cheapest.

Working with Dijkstra's algorithm

Step 2: Update the costs of the neighbors of node **B**.

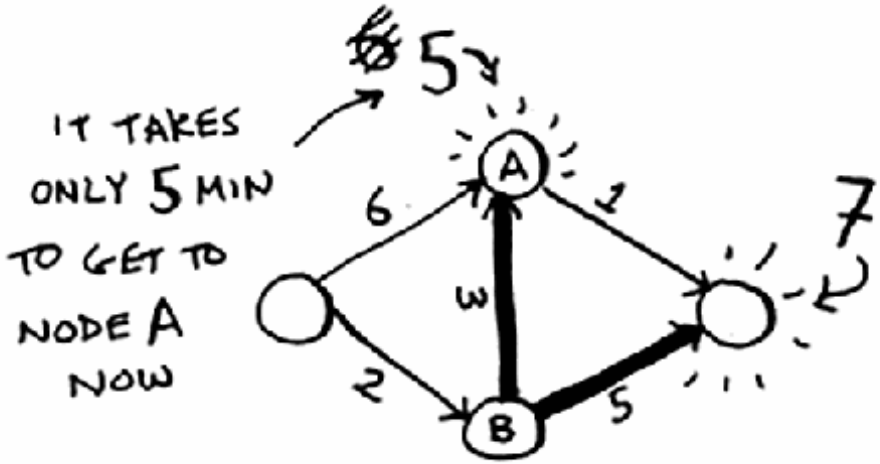
NODE	TIME
A	~~6~~ 5
B	2
FINISH	7



Working with Dijkstra's algorithm

Step 2: Update the costs of the neighbors of node **B**.

NODE	TIME
A	~~6~~ 5
B	2
FINISH	7

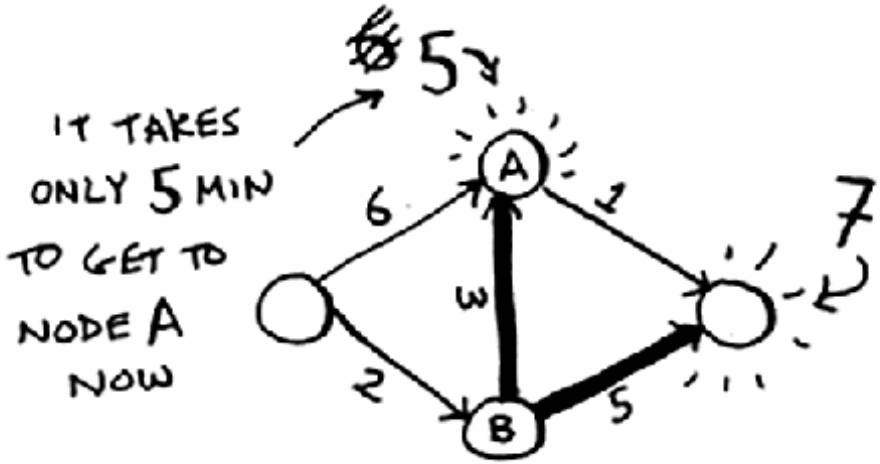


✎ You just found the shortest path to node A!

Working with Dijkstra's algorithm

Step 2: Update the costs of the neighbors of node **B**.

NODE	TIME
A	~~6~~ 5
B	2
FINISH	7

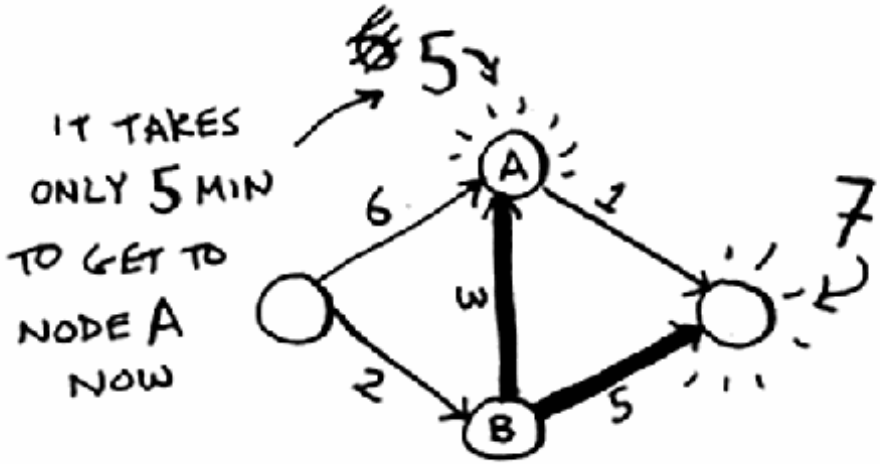


- ✎ You just found the shortest path to node A!
- ✎ A shorter path to node A (down from 6 mins $\rightarrow$ 5 mins)

Working with Dijkstra's algorithm

Step 2: Update the costs of the neighbors of node **B**.

NODE	TIME
A	~~6~~ 5
B	2
FINISH	7



- You just found the shortest path to node A!
- A shorter path to node A (down from 6 mins $\rightarrow$ 5 mins)
- A shorter path to the finish (down from $\infty \rightarrow$ 7 mins)

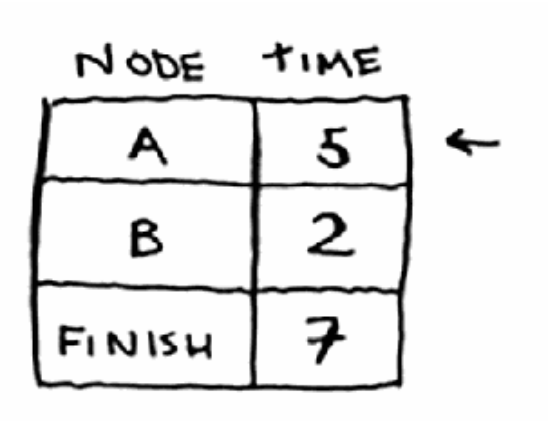
Working with Dijkstra's algorithm

Step 3: Repeat!
Step 1 again: Find the cheapest node.

Working with Dijkstra's algorithm

Step 3: Repeat!
Step 1 again: Find the cheapest node.

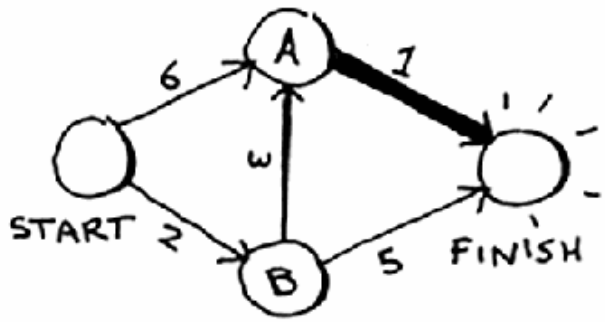
- ✎ You are done with node B, so the node A has the smallest time estimate.



Working with Dijkstra's algorithm

Step 2 again: Update the costs of the neighbors of node **B**.

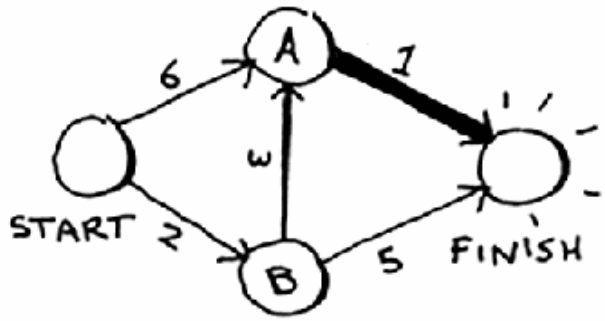
NODE	TIME
A	5
B	2
FINISH	6



Working with Dijkstra's algorithm

Step 2 again: Update the costs of the neighbors of node **B**.

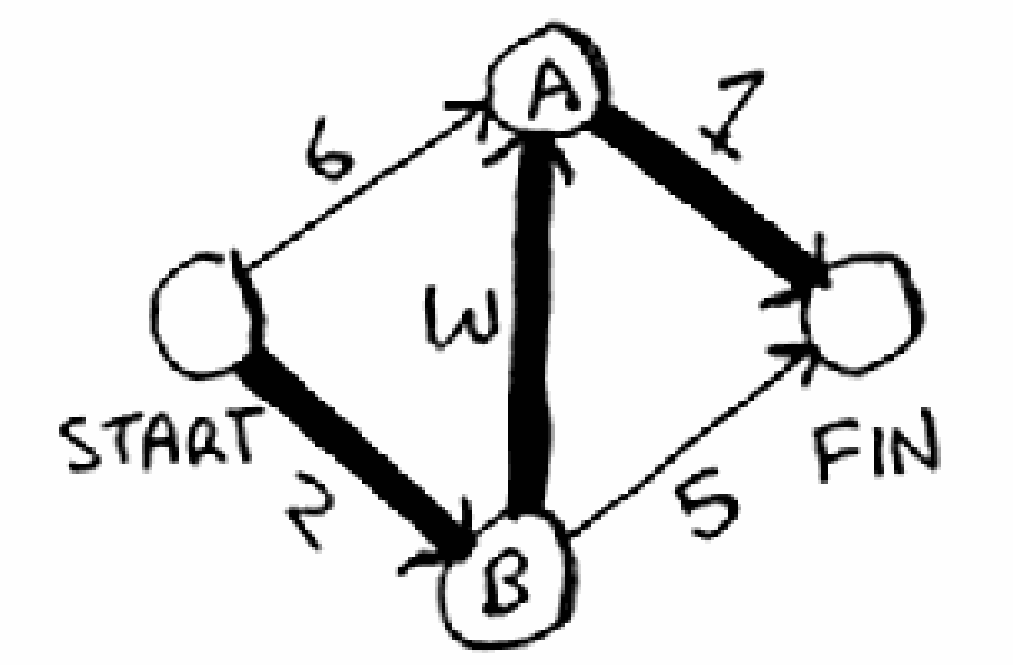
NODE	TIME
A	5
B	2
FINISH	6



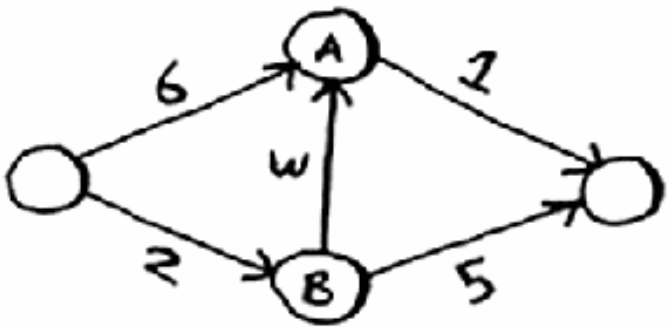
✎ Woo, it takes 6 minutes to get to the finish now!

Working with Dijkstra's algorithm

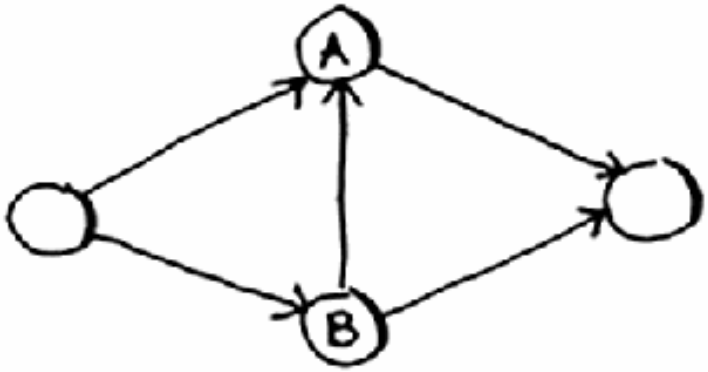
Step 4: Calculate the final path.



Weighted graphs

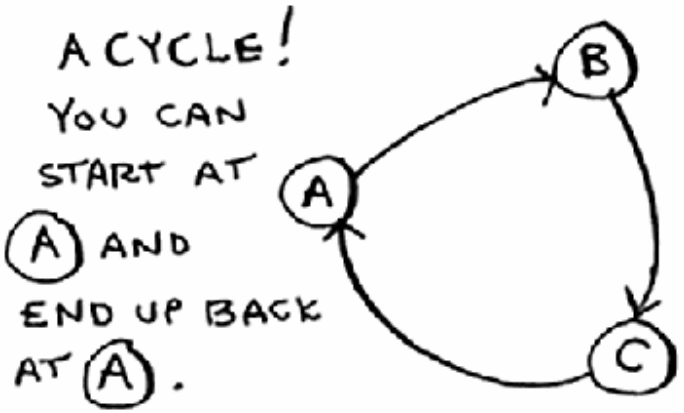


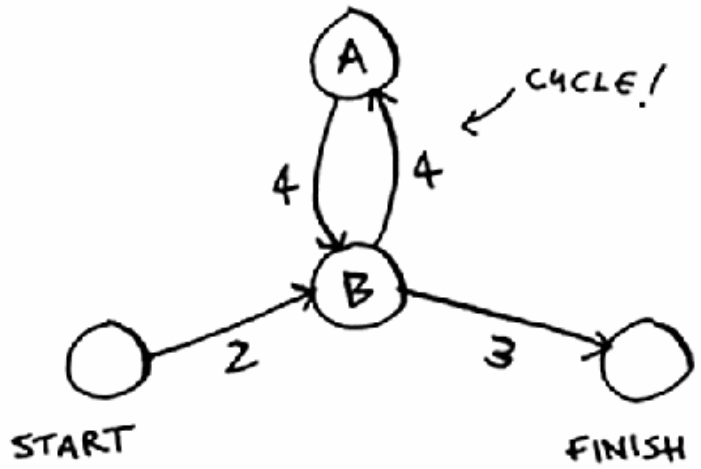
WEIGHTED GRAPH

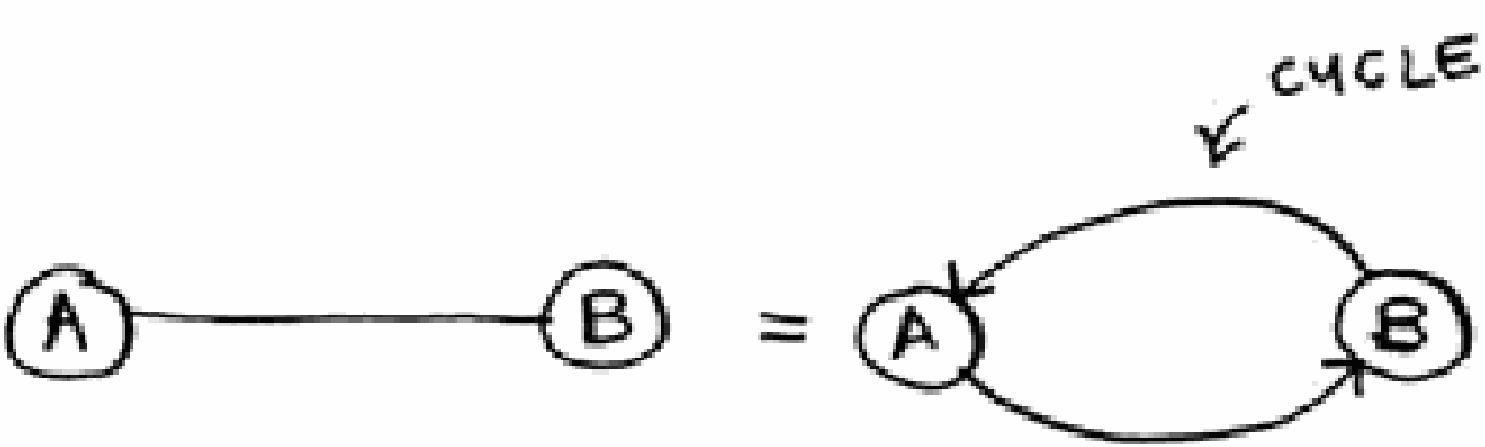


UNWEIGHTED GRAPH

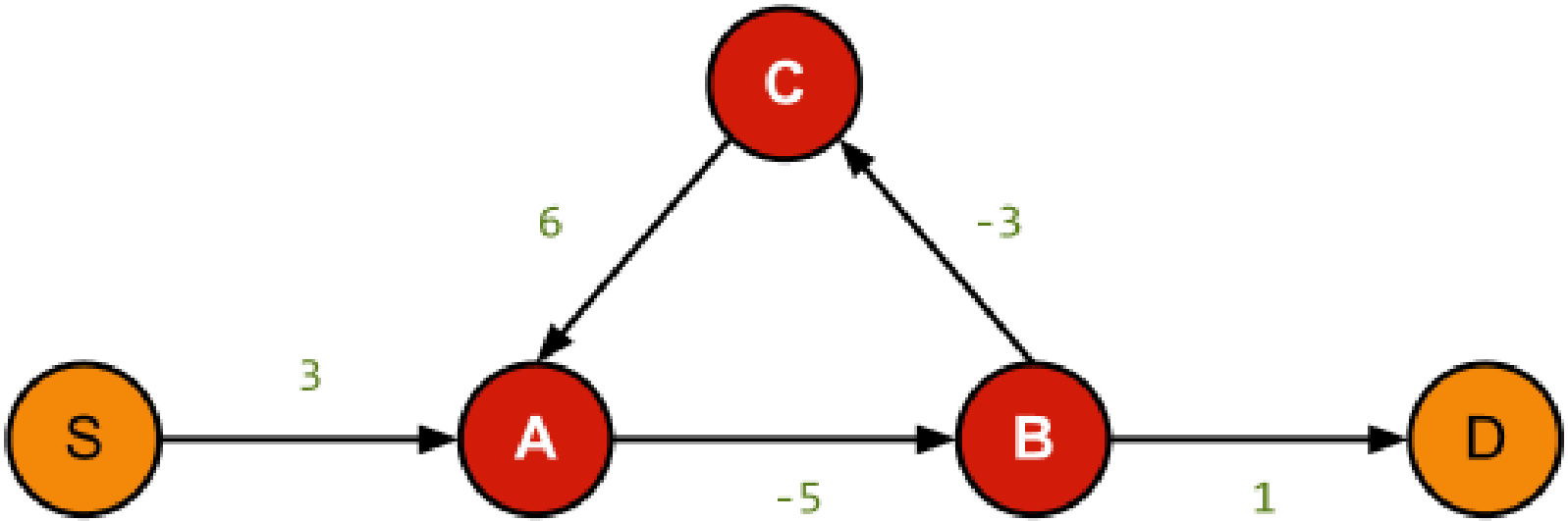
Cycles


A CYCLE!
YOU CAN START AT A AND END UP BACK AT A.
A
B
C


A
CYCLE!
4
4
B
2
3
START
FINISH


CYCLE!
A
B
=
A
B

Negative weight cycles



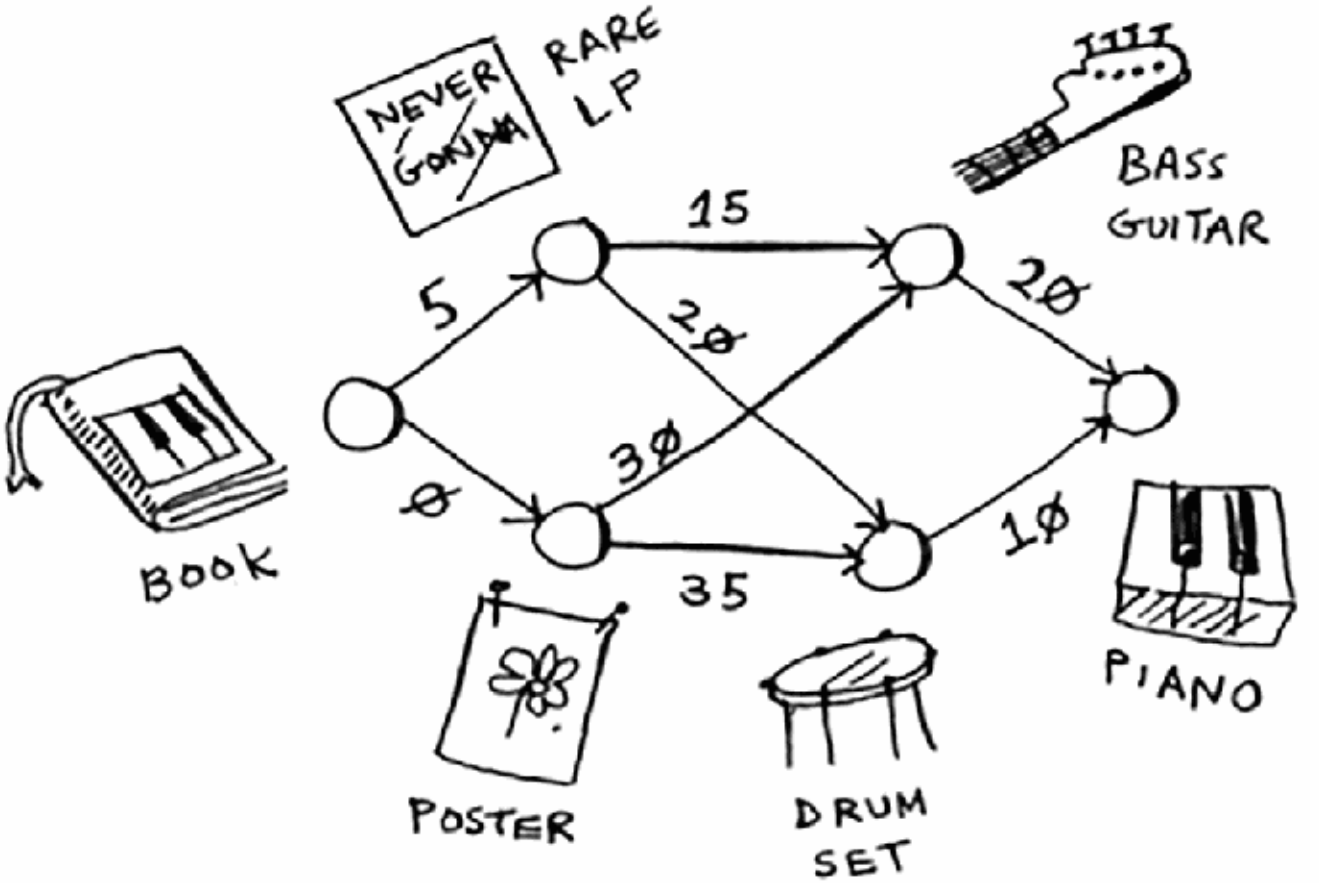
✎ The cycle

$$A \xrightarrow{-5} B \xrightarrow{-3} C \xrightarrow{6} A$$

makes the path from S to D shorter and shorter!

Trading for a piano

Try to trade a book for a piano!



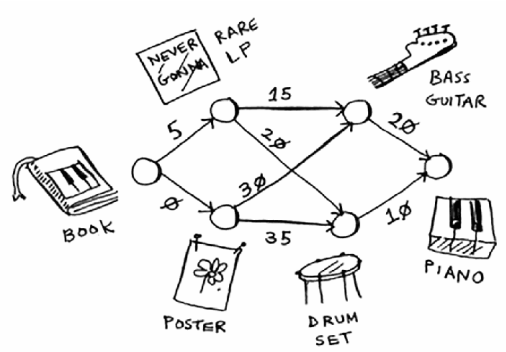
Trading for a piano

- ✎ Before you start, you may need some setup.
- ✎ To calculate the final path, you also need to memorize parents of the chosen nodes.

Node = {LP, Poster, Guitar, Drum }

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
–	Guitar	∞
–	Drum	∞
–	Piano	∞

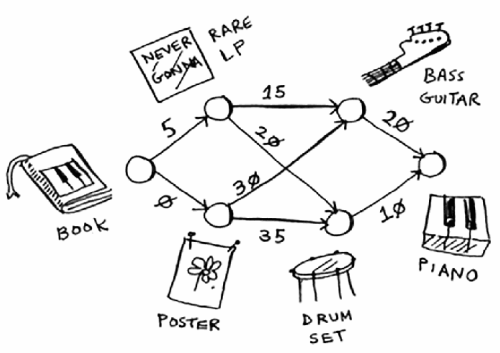
Trading for a piano



$$\text{Node} = \{\text{LP}, \text{Poster}, \text{Guitar}, \text{Drum}\}$$

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
–	Guitar	∞
–	Drum	∞
–	Piano	∞

Trading for a piano

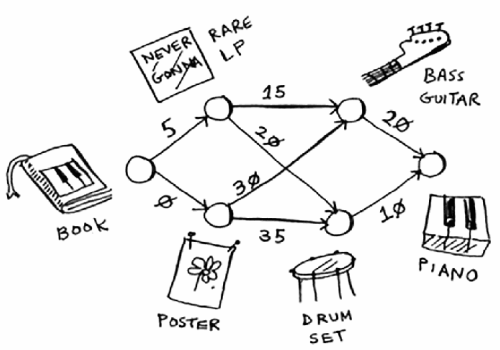


$$\text{Node} = \{\text{LP}, \text{Poster}, \text{Guitar}, \text{Drum}\}$$

✎ Choose node **Poster**.

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
–	Guitar	∞
–	Drum	∞
–	Piano	∞

Trading for a piano

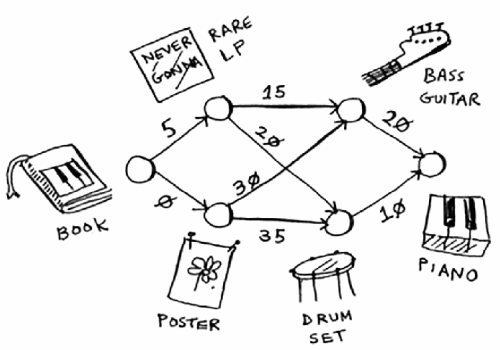


Node = {LP, Poster, Guitar, Drum}

- Choose node **Poster**.
- Update cost of nbd(Poster) = {Guitar, Drum}

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
–	Guitar	∞
–	Drum	∞
–	Piano	∞

Trading for a piano

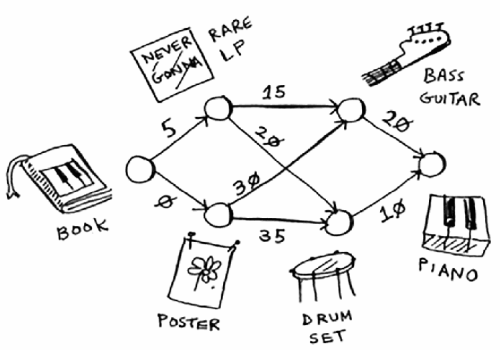


Parent	Node	Cost
Book	LP	5
Book	Poster	0
Poster	Guitar	~~∞~~ 30
–	Drum	∞
–	Piano	∞

$$\text{Node} = \{\text{LP}, \text{Poster}, \text{Guitar}, \text{Drum}\}$$

- Choose node **Poster**.
- Update cost of nbd(Poster) = {Guitar, Drum}
  - Guitar: $0 + 30 = 30 < \infty$, so assign the parent of Guitar to Poster.

Trading for a piano

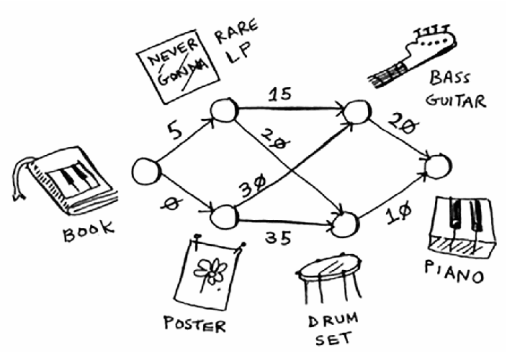


Parent	Node	Cost
Book	LP	5
Book	Poster	0
Poster	Guitar	~~∞~~ 30
Poster	Drum	~~∞~~ 35
–	Piano	∞

Node $= \{\text{LP}, \text{Poster}, \text{Guitar}, \text{Drum}\}$

- Choose node **Poster**.
- Update cost of nbd(Poster) $=$ {Guitar, Drum}
  - Guitar: $0 + 30 = 30 < \infty$, so assign the parent of Guitar to Poster.
  - Drum: $0 + 35 = 35 < \infty$, so assign the parent of Drum to Poster.

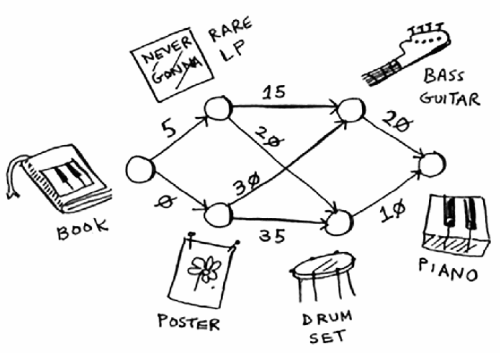
Trading for a piano



$$\text{Node} = \{\text{LP}, \sout{\text{Poster}}, \text{Guitar}, \text{Drum}\}$$

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
Poster	Guitar	30
Poster	Drum	35
–	Piano	∞

Trading for a piano

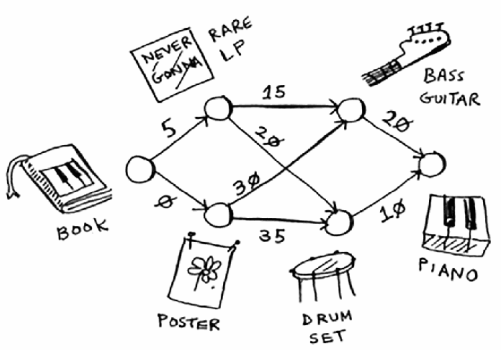


$$\text{Node} = \{\text{LP}, \sout{\text{Poster}}, \text{Guitar}, \text{Drum}\}$$

✎ Choose node **LP**.

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
Poster	Guitar	30
Poster	Drum	35
–	Piano	∞

Trading for a piano

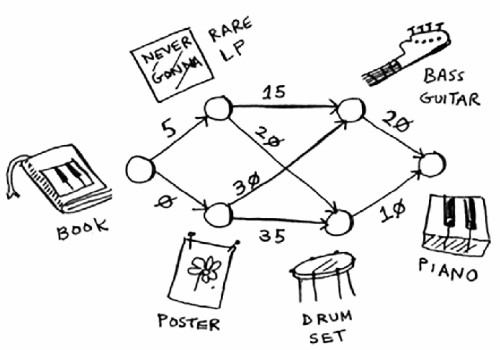


Parent	Node	Cost
Book	LP	5
Book	Poster	0
Poster	Guitar	30
Poster	Drum	35
–	Piano	∞

Node $= \{\text{LP}, \cancel{\text{Poster}}, \text{Guitar}, \text{Drum}\}$

- Choose node **LP**.
- Update cost of nbd(LP) $= \{$Guitar, Drum$\}$

Trading for a piano

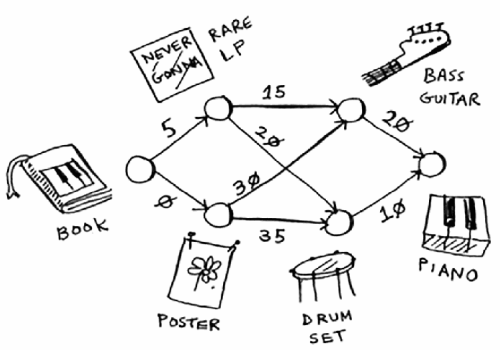


Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	~~30~~ 20
Poster	Drum	35
–	Piano	∞

Node $= \{\text{LP}, \sout{\text{Poster}}, \text{Guitar}, \text{Drum}\}$

- Choose node **LP**.
- Update cost of nbd(LP) $= \{$Guitar, Drum$\}$
  - Guitar: $5 + 15 = 20 < 30$, so assign the parent of Guitar to LP.

Trading for a piano

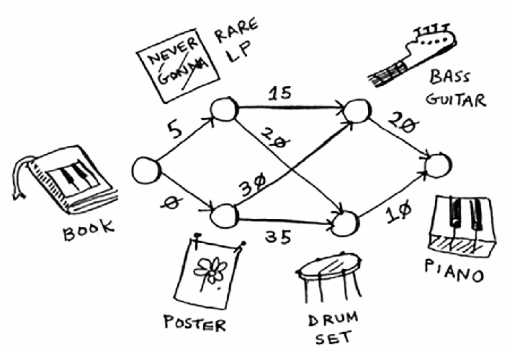


Node $= \{\text{LP}, \sout{\text{Poster}}, \text{Guitar}, \text{Drum}\}$

- Choose node **LP**.
- Update cost of nbd(LP) $= \{$Guitar, Drum$\}$
  - Guitar: $5 + 15 = 20 < 30$, so assign the parent of Guitar to LP.
  - Drum: $5 + 20 = 25 < 35$, so assign the parent of Drum to LP.

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
LP	Guitar	~~30~~ 20
LP	Drum	~~35~~ 25
–	Piano	∞

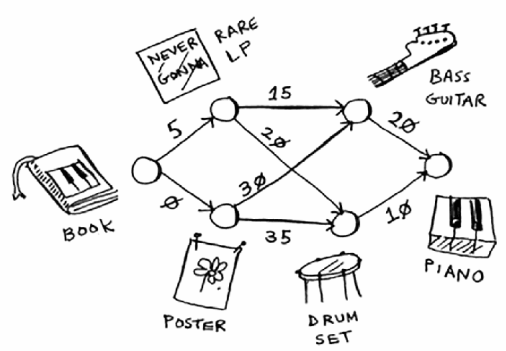
Trading for a piano



$$\text{Node} = \{\sout{\text{LP}}, \sout{\text{Poster}}, \text{Guitar}, \text{Drum}\}$$

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
–	Piano	∞

Trading for a piano

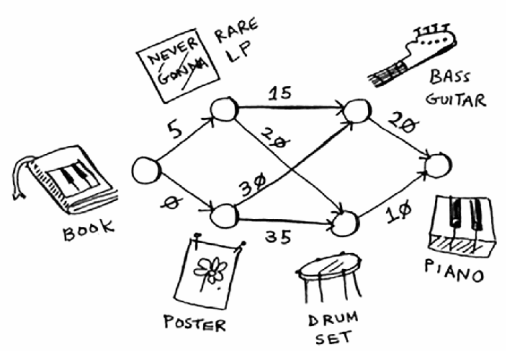


$$\text{Node} = \{\sout{\text{LP}}, \sout{\text{Poster}}, \text{Guitar}, \text{Drum}\}$$

✎ Choose node **Guitar**.

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
–	Piano	∞

Trading for a piano

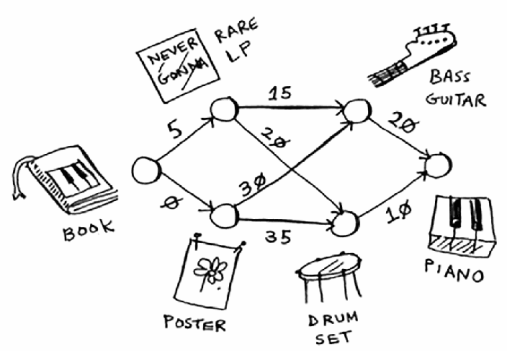


Node = {~~LP~~, ~~Poster~~, Guitar, Drum}

Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
–	Piano	∞

- Choose node **Guitar**.
- Update cost of nbd(Guitar) = {Piano}

Trading for a piano

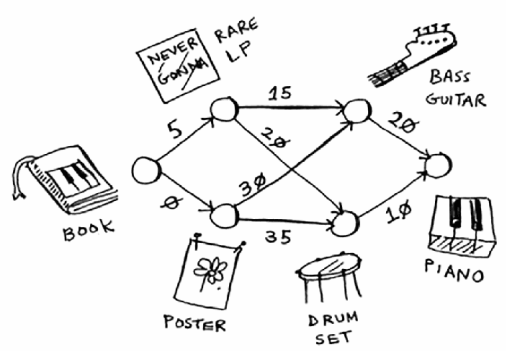


Node $= \{\sout{\text{LP}}, \sout{\text{Poster}}, \text{Guitar}, \text{Drum}\}$

Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Guitar	Piano	$\sout{\infty}$ 40

- Choose node **Guitar**.
- Update cost of nbd(Guitar) = {Piano}
  - Piano: $20 + 20 = 40 < \infty$, so assign the parent of Piano to Guitar.

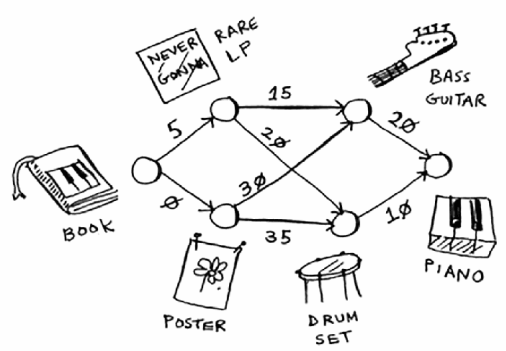
Trading for a piano



$$\text{Node} = \{\cancel{\text{LP}}, \cancel{\text{Poster}}, \cancel{\text{Guitar}}, \text{Drum}\}$$

Parent	**Node**	**Cost**
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Guitar	Piano	40

Trading for a piano

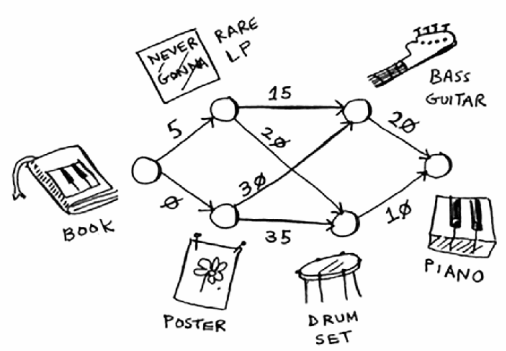


$$\text{Node} = \{\cancel{\text{LP}}, \cancel{\text{Poster}}, \cancel{\text{Guitar}}, \text{Drum}\}$$

✎ Choose node **Drum**.

Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Guitar	Piano	40

Trading for a piano

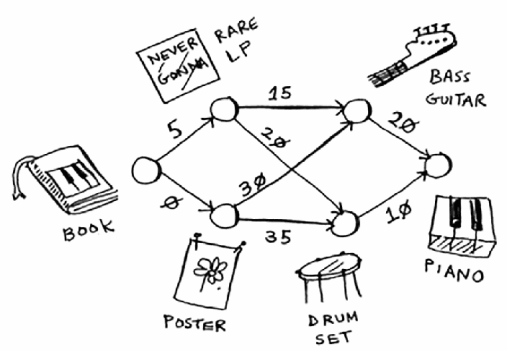


Node = {~~LP~~, ~~Poster~~, ~~Guitar~~, Drum}

Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Guitar	Piano	40

- Choose node **Drum**.
- Update cost of nbd(Guitar) = {Piano}

Trading for a piano

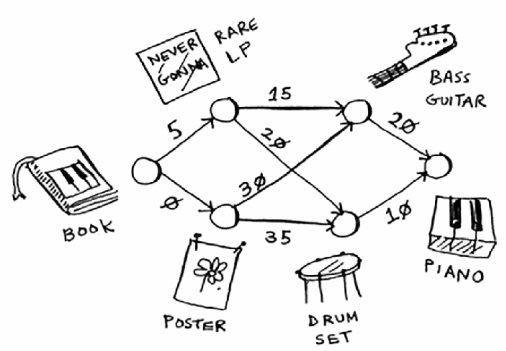


Node = {~~LP~~, ~~Poster~~, ~~Guitar~~, Drum}

Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Drum	Piano	~~40~~ 35

- Choose node **Drum**.
- Update cost of nbd(Guitar) = {Piano}
  - Piano: $25 + 10 = 35 < 40$, so assign the parent of Piano to Drum.

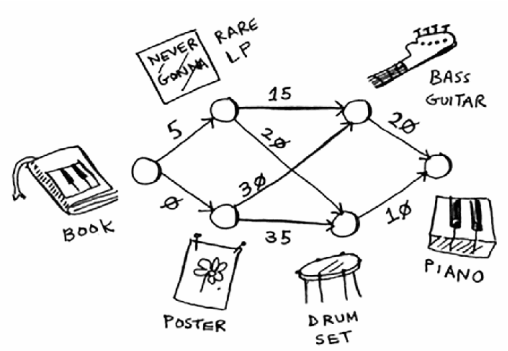
Trading for a piano



Node = {~~LP~~, ~~Poster~~, ~~Guitar~~, ~~Drum~~}

Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Drum	Piano	35

Trading for a piano



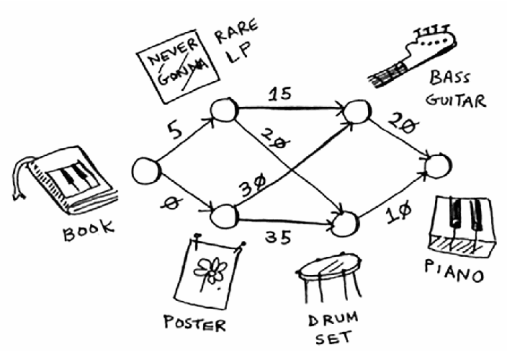
Node = {~~LP~~, ~~Poster~~, ~~Guitar~~, ~~Drum~~}

✎ Read the final path:

$$\text{Book} \xrightarrow{5} \text{LP} \xrightarrow{20} \text{Drum} \xrightarrow{10} \text{Piano}$$

Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Drum	Piano	35

Trading for a piano



Parent	Node	Cost
Book	LP	5
Book	Poster	0
LP	Guitar	20
LP	Drum	25
Drum	Piano	35

$$\text{Node} = \{\sout{\text{LP}}, \sout{\text{Poster}}, \sout{\text{Guitar}}, \sout{\text{Drum}}\}$$

- Read the final path:

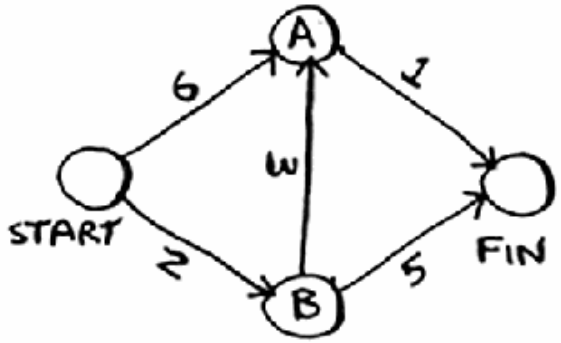
$$\text{Book} \xrightarrow{5} \text{LP} \xrightarrow{20} \text{Drum} \xrightarrow{10} \text{Piano}$$

- Calculate the cost of the final path:

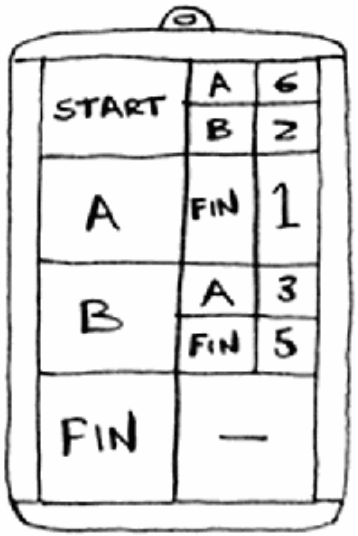
$$5 + 20 + 10 = 35.$$

Implementation

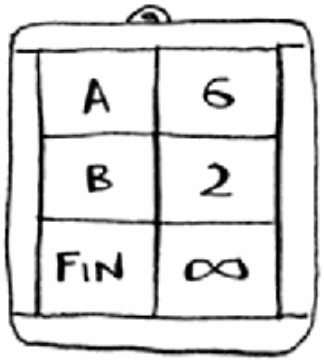
Let's see how to implement Dijkstra'a algorithm for this example:



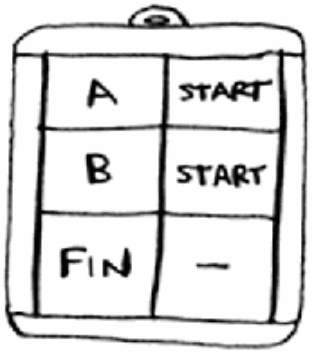
To code this example, you will need 3 hash tables.



GRAPH

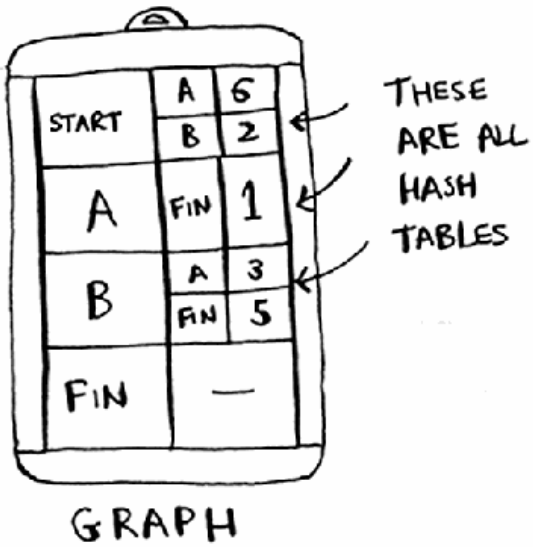


COSTS



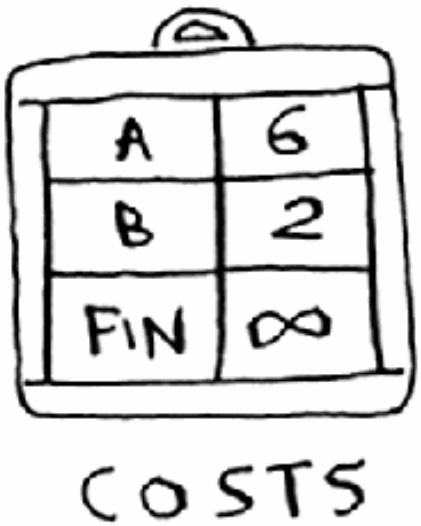
PARENTS

Implementation



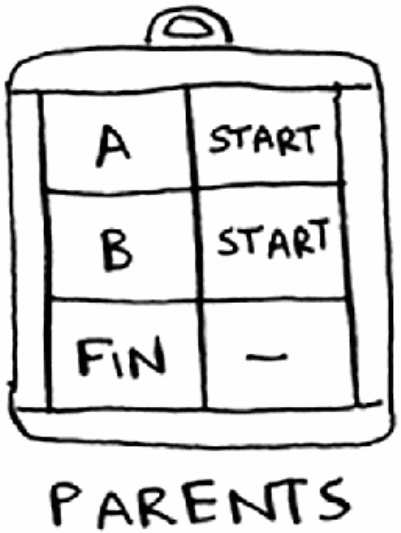

```
1  graph = {}
2
3  graph['start'] = {}
4  graph['start']['a'] = 6
5  graph['start']['b'] = 2
6
7  graph['a'] = {}
8  graph['a']['fin'] = 1
9
10 graph['b'] = {}
11 graph['b']['a'] = 3
12 graph['b']['fin'] = 5
13
14 graph['fin'] = {}
```

Implementation




```python
import numpy as np

costs = {}
costs['a'] = 6
costs['b'] = 2
costs['fin'] = np.inf
```




```python
parents = {}
parents['a'] = 'start'
parents['b'] = 'start'
parents['fin'] = None
```

Implementation

```
processed = [ ]
node = find_lowest_cost_node(costs, processed)
while node is not None:
    cost = costs[node]
    neighbors = graph(node)
    for n in neighbors.keys():
        new_cost = cost + neighbors[n]
        if costs[n] > new_cost:
            costs[n] = new_cost
            parents[n] = node
    processed.append(node)
    node = find_lowest_cost_node(costs)
```

Implementation

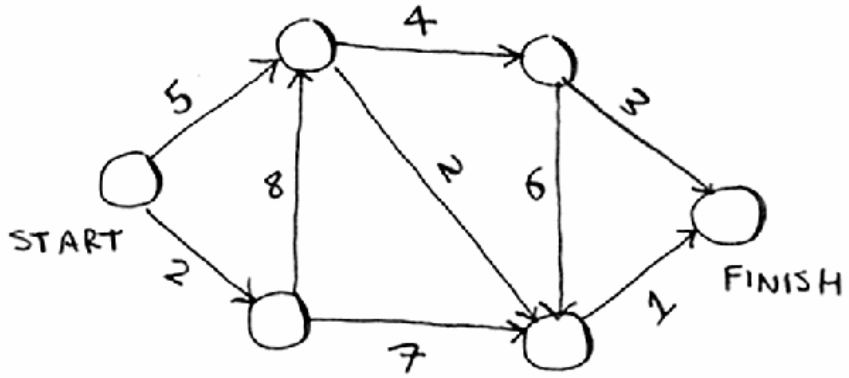
```python
import numpy as np

def find_lowest_cost_node(costs, processed):
    lowest_cost = np.inf
    lowest_cost_node = None
    for node in costs:
        cost = costs[node]
        if cost < lowest_cost and node not in
        ↪  processed:
            lowest_cost = cost
            lowest_cost_node = node
    return lowest_cost_node
```

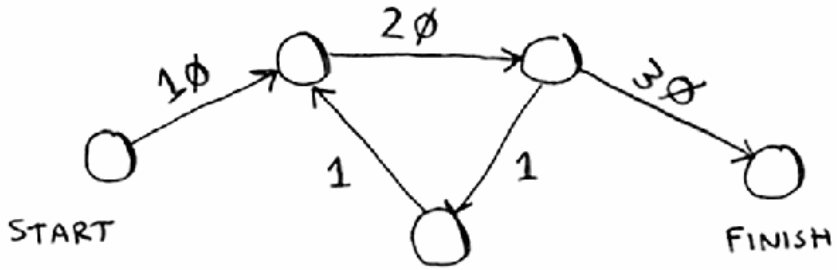
Exercise

What is the weight of the shortest path from start to finish?

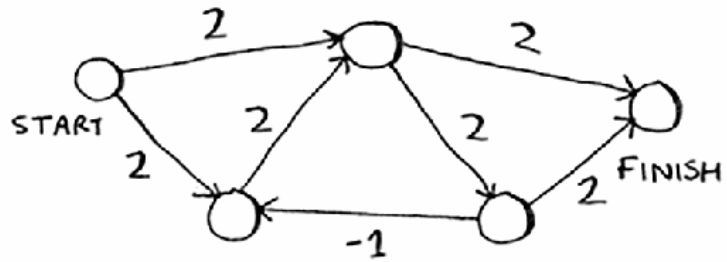
A.



B.



C.



Recap

- Breadth-first search is used to calculate the shortest path for an unweighted graph.
- Dijkstra's algorithm is used to calculate the shortest path for a weighted graph.
- Dijkstra's algorithm works when all the weights are positive.