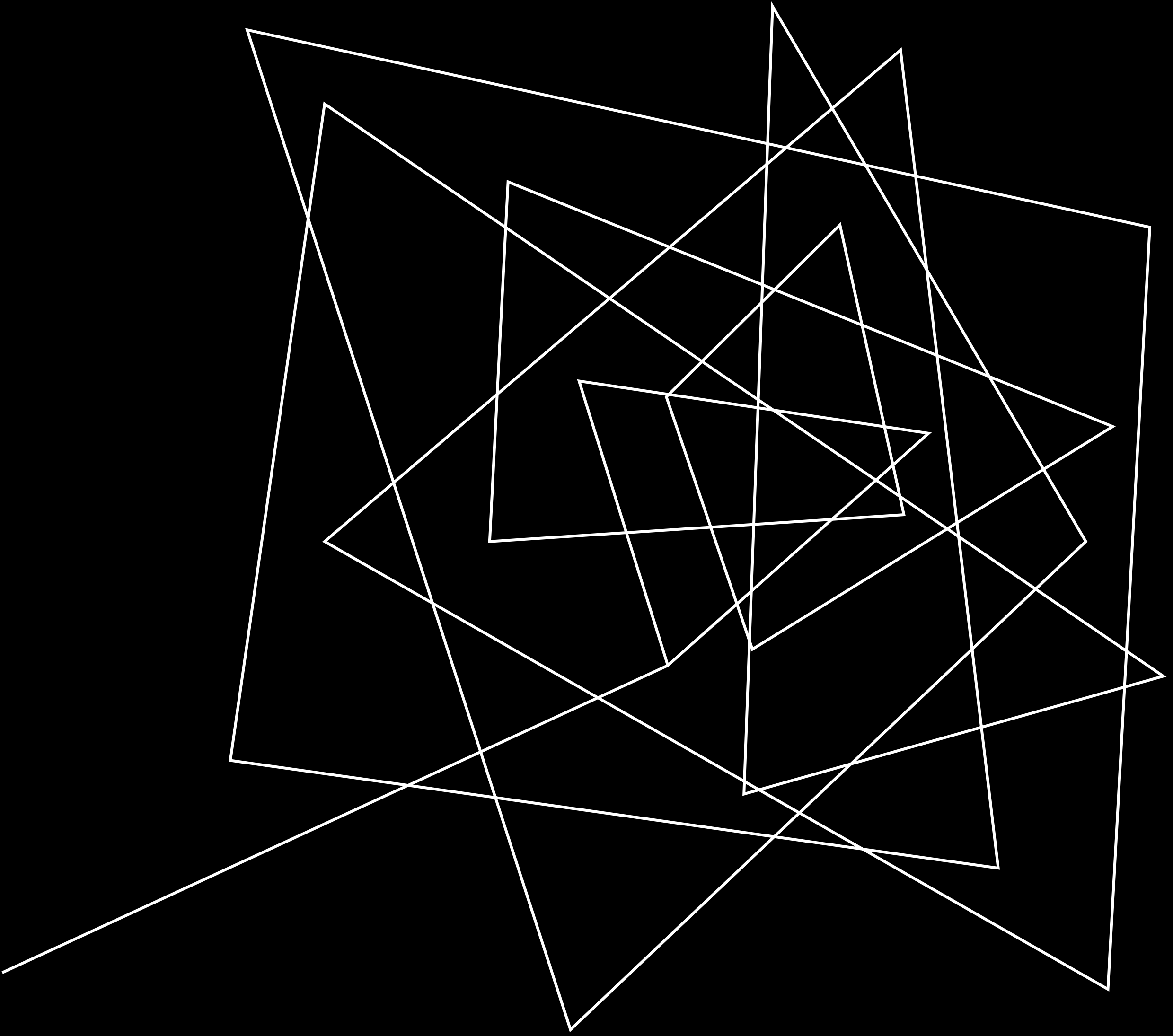


DIJKSTRA & PRIM'S ALGORITHMS

ALINA BURLACU

THE GRAPH MODEL

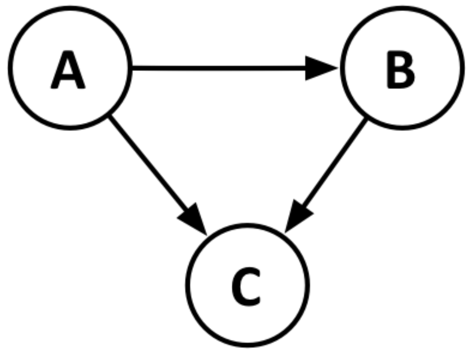
A **Graph Model** in computer science refers to an *abstract data structure* used to represent non-linear relationships between data items.

A graph consists of a set of **vertices** (nodes containing data) and **edges** that connect the vertices.

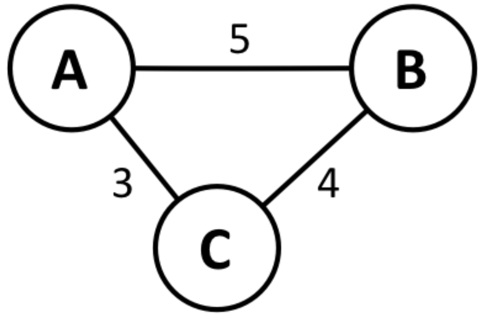
Graphs use the following notation: G = {Vertex, Edge}

Types of Graphs:

- Undirected Graph
  - can move in any direction between vertices (nodes)
  - like a 2-way street
- Directed Graph
  - can only move in direction specified by arrows on edge lines
  - like a 1-way street
- Weighted Graph
  - can be either undirected or directed
  - has values (weights) associated with the edges
  - similar to distances between places



directed graph



undirected, weighted graph

DIJKSTRA'S ALGORITHM

The Single Source Shortest Path (SSSP) Problem consists of finding the shortest path from a specific vertex to all vertices in the given graph.

- The given graph must be:
  - directed & weighted
  - have only positive weights

Dijkstra's algorithm is a way to solve this problem, by implementing a greedy algorithm.

A greedy algorithm is one that builds a solution step by step, where at each step the "best" current option is chosen.

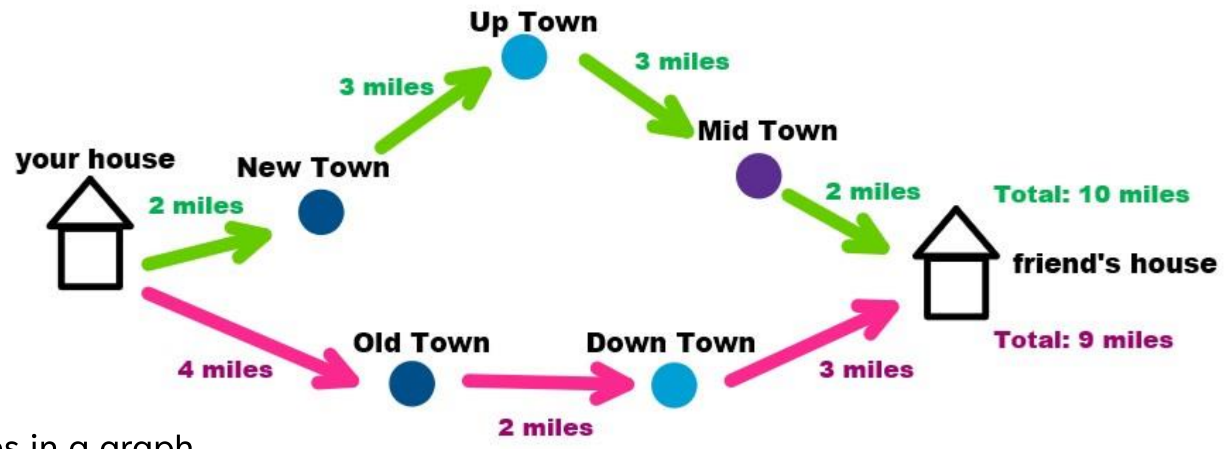
In the way we are approaching the SSSP problem using Dijkstra's greedy algorithm,

we can think of it like the following example

DIJKSTRA’S ALGORITHM EXAMPLE

- You want to go to your friend's house in a different town.
- You ask someone for directions. They tell you that you can either go through **New Town** (2 miles away) or **Old Town** (4 miles away)
- You pick the shortest route through **New Town**.
- Once you get there, ask someone else where to go next. Then again, you choose the shortest path.
- You repeat these steps till you get to your friend's house.
- It took you ***10 miles*** to get there.
- While you think this is the right path to take, you didn't know that if you had gone to **Old Town** first,
- you would have gotten to your friend's house in ***9 miles*** instead.

This is basically the same idea Dijkstra's algorithm follows in order to solve the SSSP problem.
However, it does one more thing that makes it beneficial to problem-solving: using edge relaxation.



Edge relaxation refers to updating the path based on the weight of the edges in a graph.

In our example, this would mean that instead of just choosing the shortest distance for our first town visited, we would also look at the following town and the one after that, to see if the path really is shorter, or just seems to be.

CODE FOR DIJKSTRA’S ALGORITHM

```python
def dijkstra(graph, s):
    visited = {s: 0}; path = {}
    nodes = set(graph.nodes)
    while nodes:
        min_node = None
        for node in nodes:
            if node in visited:
                if min_node is None: min_node = node
                elif visited[node] < visited[min_node]: min_node = node
            if min_node is None: break
            nodes.remove(min_node)
            current_weight = visited[min_node]
            for v in graph.edges[min_node]:
                weight = current_weight + graph.distances[(min_node, v)]
                if v not in visited or weight < visited[v]:
                    visited[v] = weight
                    path[v] = min_node
    return visited, path
```

PRIM’S ALGORITHM

The Minimum Spanning Tree (MST) Problem consists of a **spanning tree** which is a form of a graph.

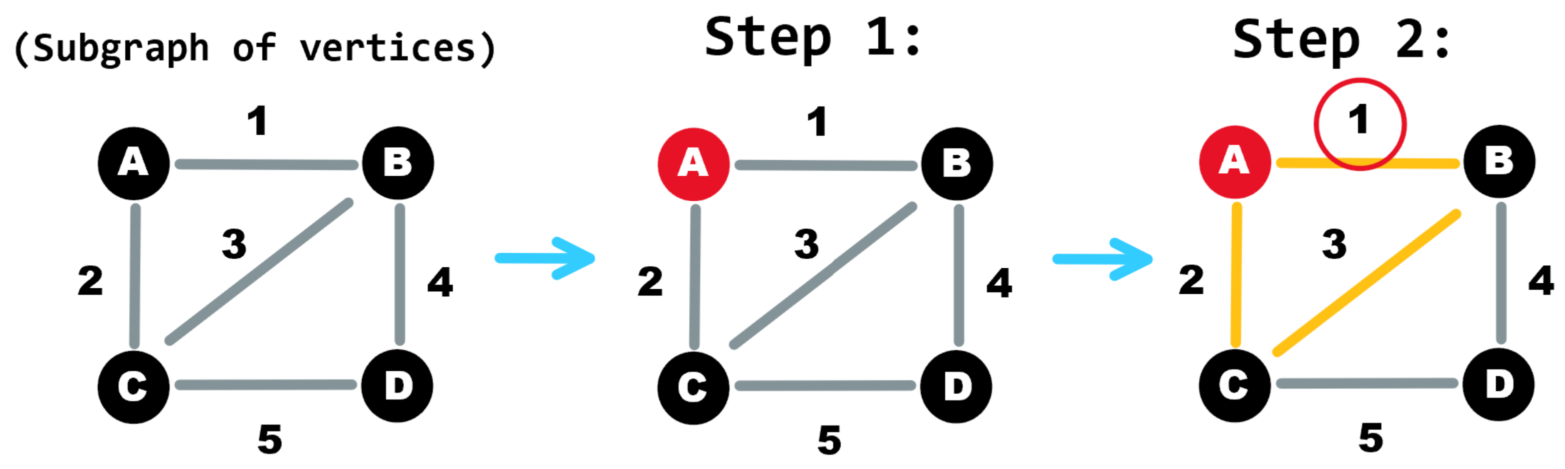
A tree is simply a disjoint graph, and a spanning tree is a subgraph of that graph, which only includes the vertices (nodes containing data) of the graph.

- **Properties of spanning tree (subgraph):**
  - Contains all vertices from main graph
  - All vertices are connected(there is a path of edges going to each vertex)
  - There can't be any cycles (loops where vertices are connected in a circular fashion)

Prim’s algorithm utilizes a greedy algorithm just like Dijkstra's. But in Prim's algorithm, we use it to find the minimum spanning tree.

In the MST Problem, we are given an undirected & weighted graph.
The goal is to achieve a minimum total weight while visiting every vertex in the graph.

STEPS TO PRIM'S ALGORITHM



Step 1

To build our tree, choose *any vertex* from the graph as a starting point

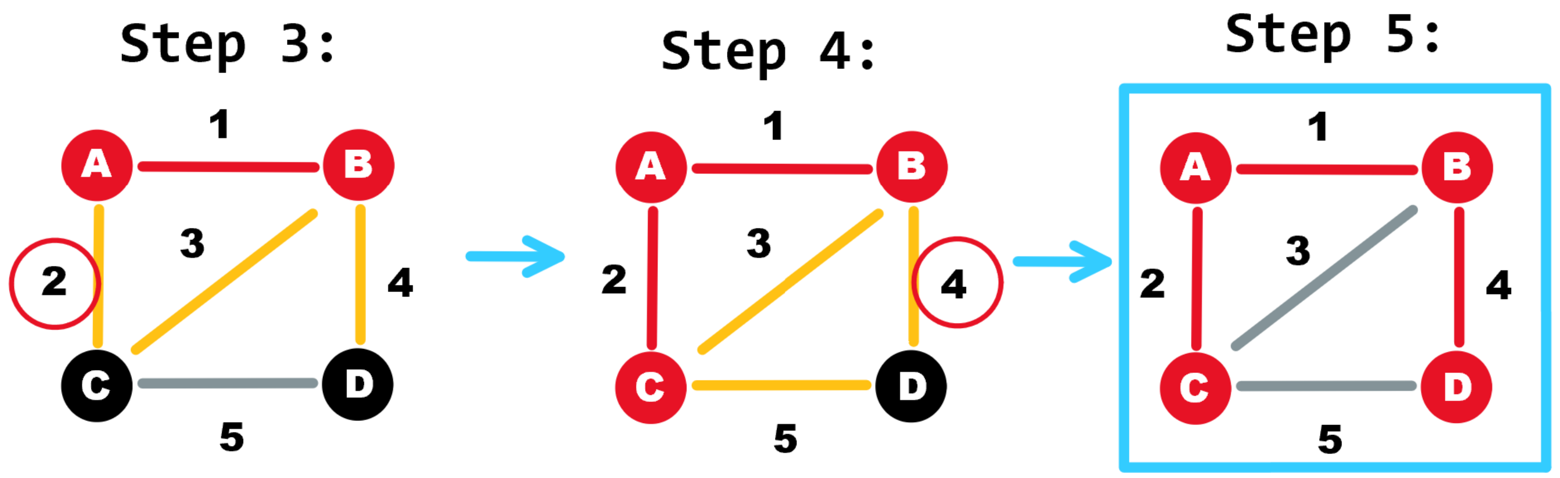
- Here, we'll pick node **A** to begin
- When a vertex is visited, it's added to the MST: **{A}**

Step 2

From the starting node, look at all the *weights* of the edges connected to it

- For **A** the edges have the values *1, 2, 3*
- Pick the edge with the smallest weight
- Here, that is the edge with weight 1 leading to node **B**
- Add the new node to the MST: **{A, B}**

STEPS CONTINUED



Step 3

Now look at next set of edges that connect visited nodes to unvisited nodes

- We want to pick only edges that lead to unvisited nodes
- So the edge with value *1* is out
- Choose the smallest weighted edge again
- In this case, the value is *2* for the edge connecting **A** to **C**
- Visit the new node, add it to the MST: **{A, B, C}**

Step 4

Now we've reached the last set of edges, with values *3, 4, 5*

- Usually, we would pick the smallest weight, which is *3*
- This is not the case here, since the edge with weight *3* would lead us back to an already visited node, **B**
- The next lowest weight is *4*, so we choose this edge

Step 5

Visit the final node and add it to the MST: **{A, B, C, D}**

- The MST is complete! All nodes have been visited only once, and the minimum weighted path was found.
- The minimum spanning tree has a total weight, or *cost* of 1 + 2 + 4 = **7**

CODE FOR PRIM'S ALGORITHM

```python
def primMST(self): #self is a graph (G)
    key = [sys.maxsize] * self.V # set to infinity
    parent = [None] * self.V # Array to store
    key[0] = 0
    mstSet = [False] * self.V
    parent[0] = -1 # First node is always the root
    for cout in range(self.V):
        u = self.minKey(key, mstSet)
        mstSet[u] = True
        # self.graph[u][v] is an adjacent matrix
        for v in range(self.V):
            if mstSet[v] == False and self.graph[u][v] > 0 and key[v] > self.graph[u][v]:
                key[v] = self.graph[u][v]
                parent[v] = u
```

RESOURCES

- Graphs - Isaac Computer Science
- Graphs: Single-Source-Shortest-Path
- Greedy Algorithm - Programiz
- Edge Relaxation in Dijkstra's Algorithm - Baeldung
- Activity Selection Problem | Greedy Algo-1 - GeeksforGeeks
- Minimum Spanning Tree Tutorials & Notes | Algorithms
- Spanning Tree and Minimum Spanning Tree - Programiz