ADVANCED GRAPH ALGORITHMS

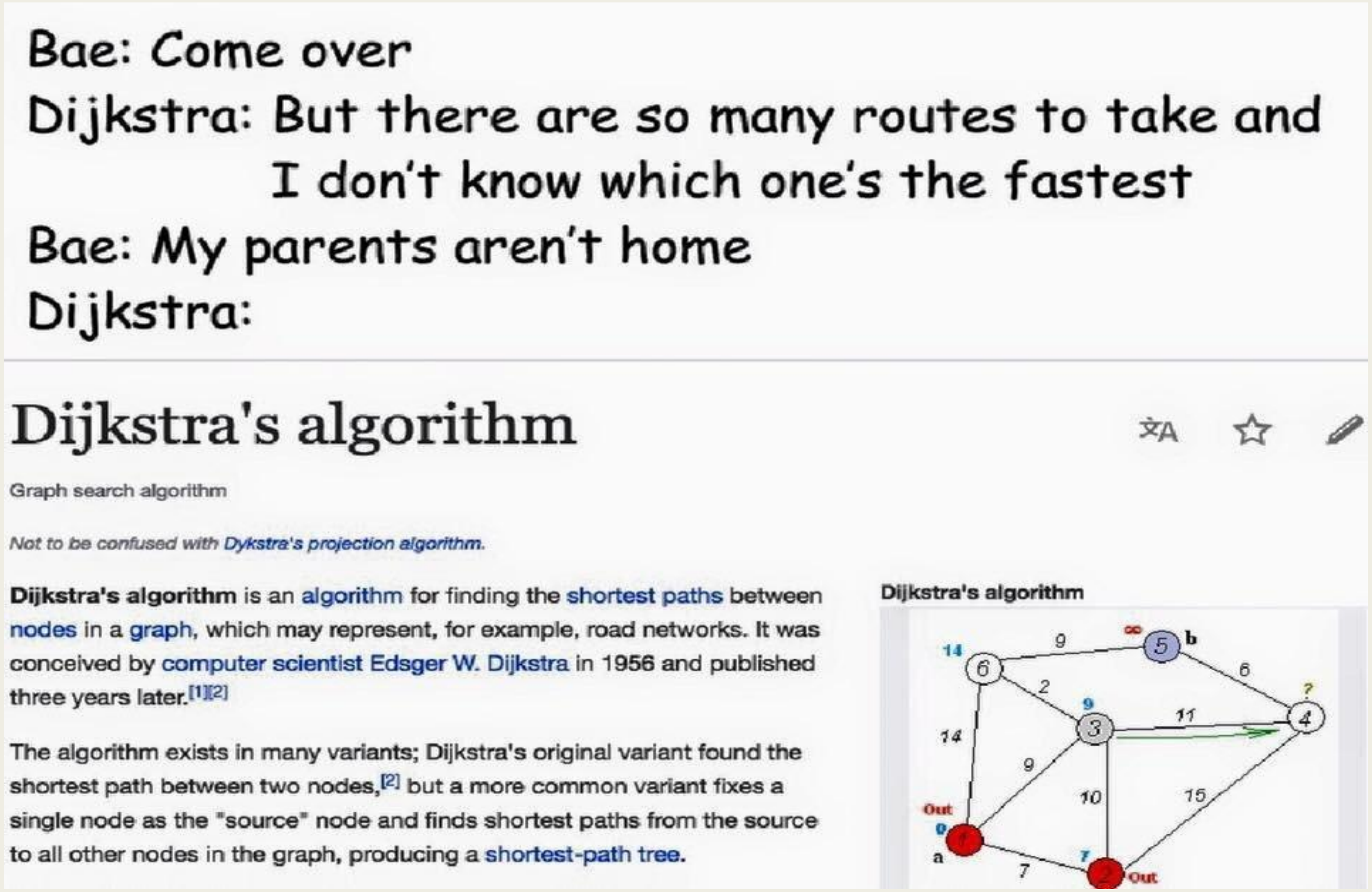
Quick Recap

- Graphs are weighted/unweighted and directed/undirected
- We should all know DSU (Disjoint Set Union) by now
- Paths and shortest paths
- We will solve SSSP (Single Source Shortest Paths) using Dijkstra and Bellman-Ford
- We will solve APSP (All Pairs Shortest Paths) using Floyd-Warshall
- We will learn to extract the MST (Minimum/Maximum Spanning Tree) using Prim's and Kruskal's algorithms

Single Source Shortest Paths (SSSP)

- Extract the shortest/longest paths from a single node (the source) to all the other nodes
- We can get asked about the length of the path or the nodes to the path
- There might be more than one weight or constraint on edges

Dijkstra



Bae: Come over

Dijkstra: But there are so many routes to take and I don't know which one's the fastest

Bae: My parents aren't home

Dijkstra:

Dijkstra's algorithm

Graph search algorithm

Not to be confused with Dykstra's projection algorithm.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.[1][2]

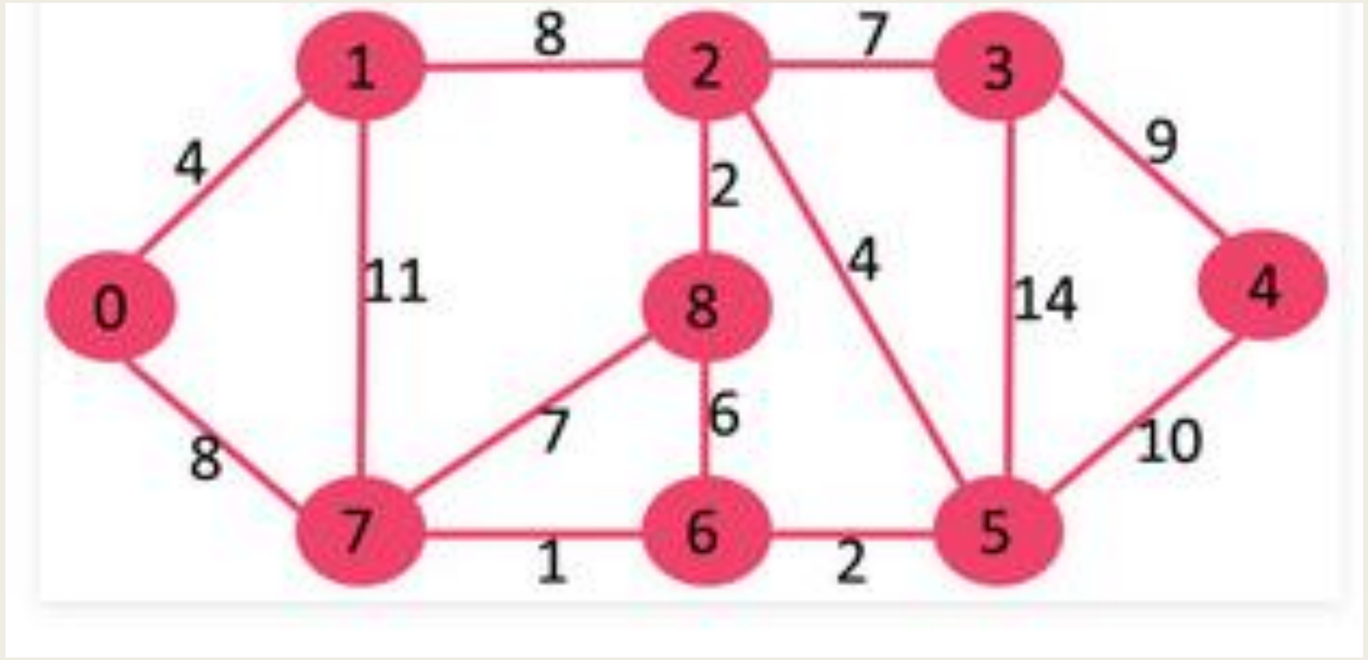
The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,[2] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

Dijkstra's algorithm

Dijkstra’s Algorithm

- It is used to solve SSSP (Works for longest paths too)
- It has an astonishing complexity of $O((E + V)logV)$
- We can extract not only the length of the shortest path, but also the nodes that are form that path
- Dijkstra’s algorithm doesn’t work with negative numbers (we will discuss this after implementing the algorithm)

Dijkstra’s Algorithm Illustration



Pic Courtesy of GeeksforGeeks

Priority Queue

- Priority queue is a crucial data structure in Dijkstra’s algorithm
- It is a C++ data structure based on Fibonacci’s heap
- It is similar to a normal queue by the principle of **popping**, but with the difference that it sorts the elements inside it (biggest by default)
- For longest paths

```
priority_queue<pii> pq;
```

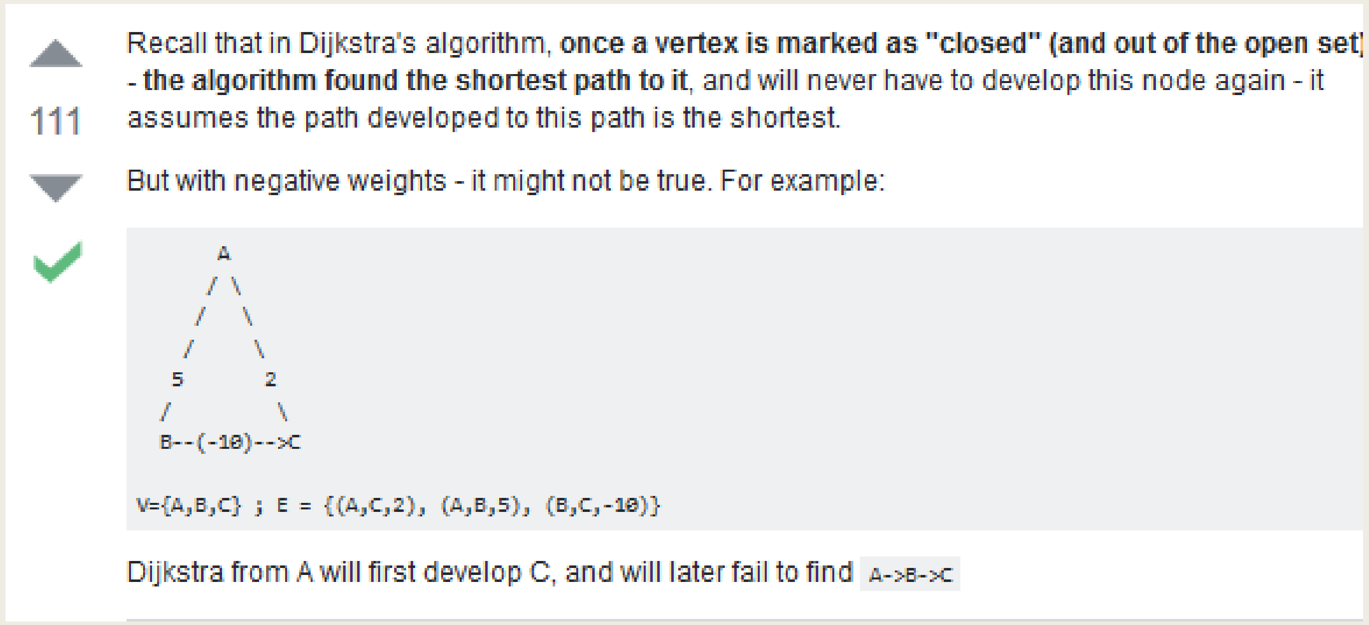
- For shortest paths

```
priority_queue<pii, vector<pii>, greater<pii> > pq;
```

Complexity Discussion

- Remember that we mentioned the complexity is $O((E + V)logV$

So Why Doesn't It Work on Negative Weights?



111

Recall that in Dijkstra's algorithm, **once a vertex is marked as "closed" (and out of the open set) - the algorithm found the shortest path to it**, and will never have to develop this node again - it assumes the path developed to this path is the shortest.

But with negative weights - it might not be true. For example:

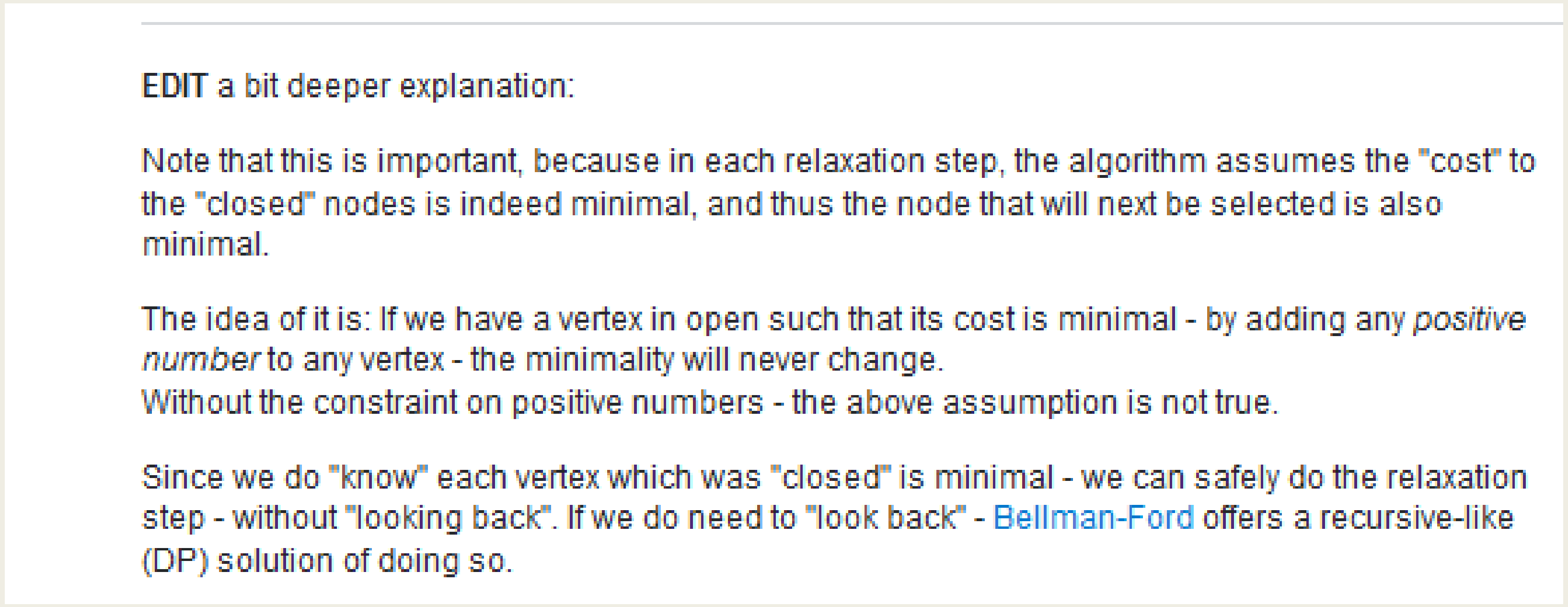
```
       A
      / \
     /   \
    /     \
   5       2
  /         \
  B--(-10)-->C

V={A,B,C} ; E = {(A,C,2), (A,B,5), (B,C,-10)}
```

Dijkstra from A will first develop C, and will later fail to find `A->B->C`

Pic Courtesy of stackoverflow

So Why Doesn’t It Work on Negative Weights?



EDIT a bit deeper explanation:

Note that this is important, because in each relaxation step, the algorithm assumes the "cost" to the "closed" nodes is indeed minimal, and thus the node that will next be selected is also minimal.

The idea of it is: If we have a vertex in open such that its cost is minimal - by adding any *positive number* to any vertex - the minimality will never change.
Without the constraint on positive numbers - the above assumption is not true.

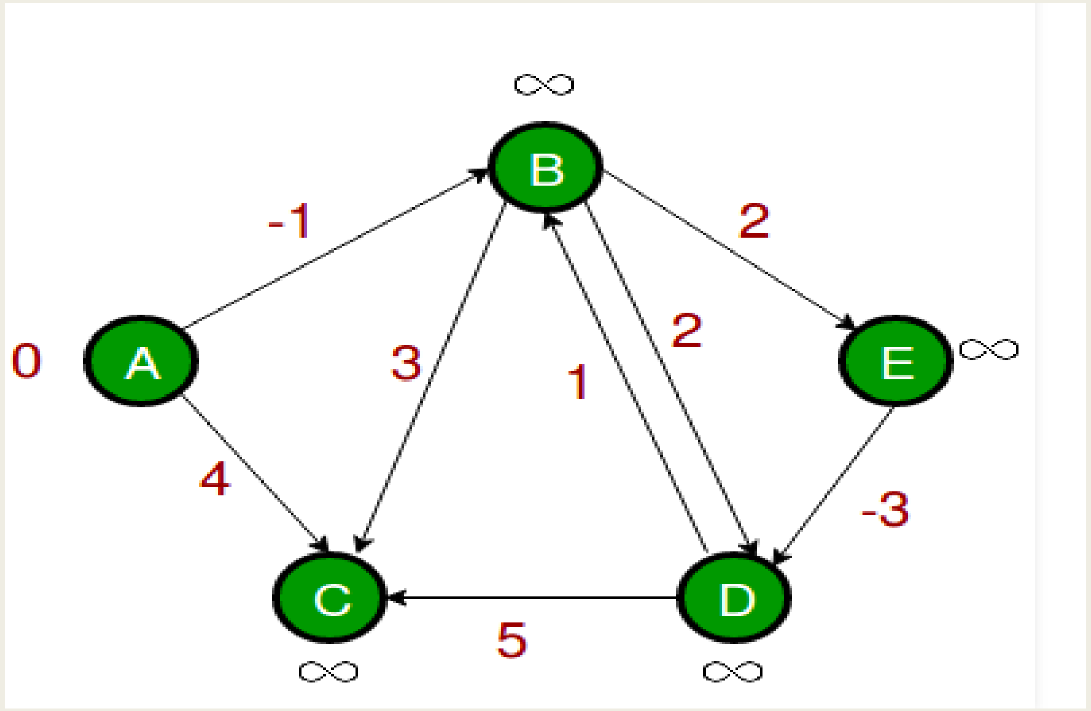
Since we do "know" each vertex which was "closed" is minimal - we can safely do the relaxation step - without "looking back". If we do need to "look back" - Bellman-Ford offers a recursive-like (DP) solution of doing so.

Pic Courtesy of stackoverflow

Bellman-Ford Algoritm (SSSP)

- It depends on the concept of dynamic programming
- Can detect negative cycles
- Works with negative numbers

Bellman-Ford Illustration



Pic Courtesy of GeeksforGeeks

Bellman-Ford Implementation

Bellman-Ford Complexity

- $O(VE)$

Floyd-Warshall (All Pairs Shortest Paths)

- Solves APSP problems
- Can also detect negative cycles
- Beware: high complexity

Floyd-Warshall Illustration

Does Floyd-Warshall Detect Negative Cycles the Same Way?

- No.

Floyd-Warshall Implementation

Floyd Warshall Complexity

- $O(V^3)$

So How Do They Fare in Comparison to Each Other

	Dijkstra	Bellman-Ford	Floyd-Warshall
Solves	SSSP	SSSP	APSP
Negative	Can't	Can	Can
Complexity	$O((E+V)logV)$	$O(VE)$	$O(V^3)$

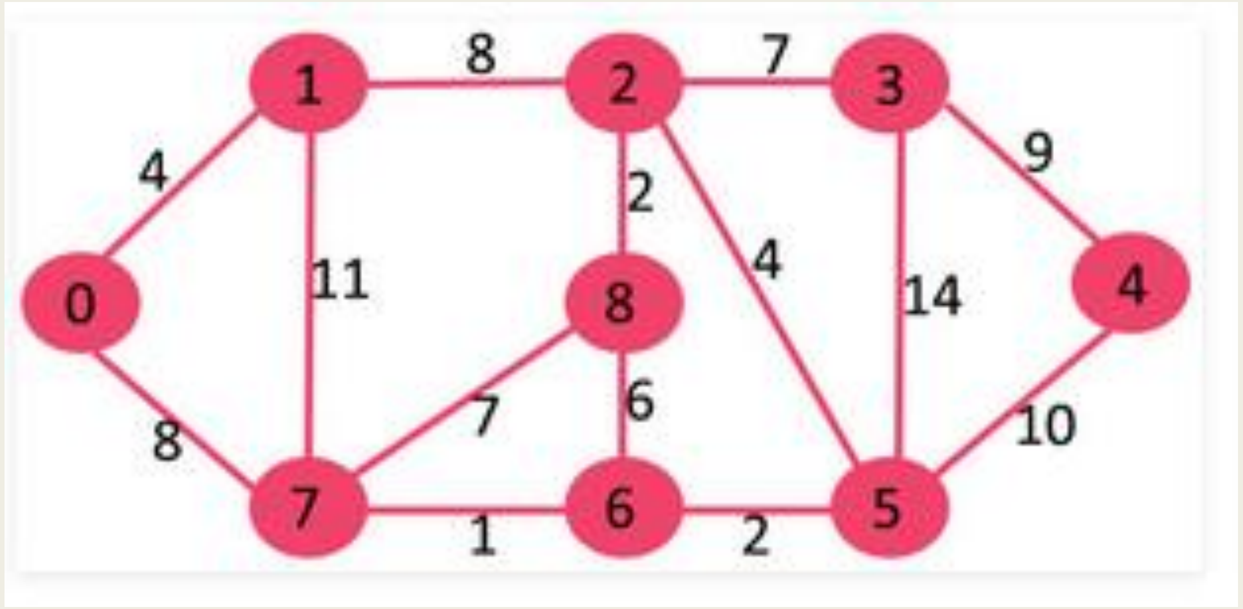
Spanning Trees

- It is a tree created from the graph
- Therefore, it doesn’t contain cycles
- Applications are Minimum/Maximum Spanning Trees

Minimum Spanning Tree (MST) Prim’s Method

- Very similar to Dijkstra’s algorithm
- Always picks the vertex with the minimum distance from the source
- The source can be arbitrarily chosen

Prim’s MST Illustration



Pic Courtesy of GeeksforGeeks

Prim’s MST Implementation

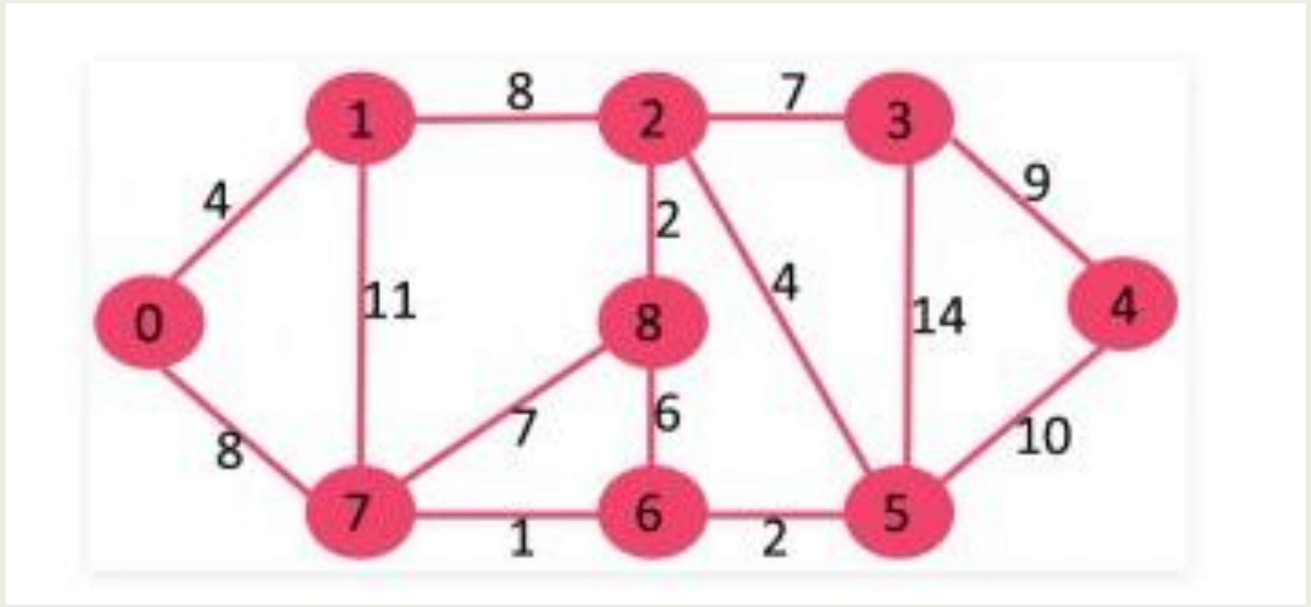
Prim’s MST Complexity

- $O(ElogV)$

Kruskal’s MST

- Uses DSU
- Sorts edges according to their weight and ignores edges that connect nodes that are already connected in DSU

Kruskal’s MST Illustration



Pic Courtesy of GeeksforGeeks

Kruskal’s MST Implementation

Kruskal’s MST Complexity

- $O(ElogE + ElogV)$ where $ElogE$ for sorting and $ElogV$ for DSU
- Note that $E = V^2$ at most then $logE = 2logV$
- This translates to $O(ElogV) = O(ElogE)$

Problems to Solve

- http://www.codeforces.com/problemset/problem/20/C
- http://codeforces.com/problemset/problem/676/D
- http://codeforces.com/gym/100935/problem/J
- https://www.spoj.com/problems/MST (Prim)
- https://www.spoj.com/problems/IITWPC4I/ (Kruskal)

THANK YOU FOR YOUR ATTENTION! ^^