Algorithm Design & Analysis (CSE222)

Lecture-17

Outline

- Jarniks / Prims

- Kruskals

Outline

- Union Find data Structure
- Shortest Paths

Union Find Data Structure

Consider following operations in a data structure.

MakeSet(v): Create a set containing only the vertex v.
Find(v): Return unique identifier of the set containing v.
Union(u, v): Replace the sets containing u and v with their union.

Maintain an array *component* of size |V|, where component[x] is the name of the set that contains $x \in V$.

Running time of union(u, v): O(|V|)

Can we do better?

Union Find Data Structure

<u>Claim</u>: There is a way to perform the Union() operation such that its amortized running time for a sequence of k operations is O(k log k).

<u>Proof</u>: Union of two sets uses the name of the larger set.

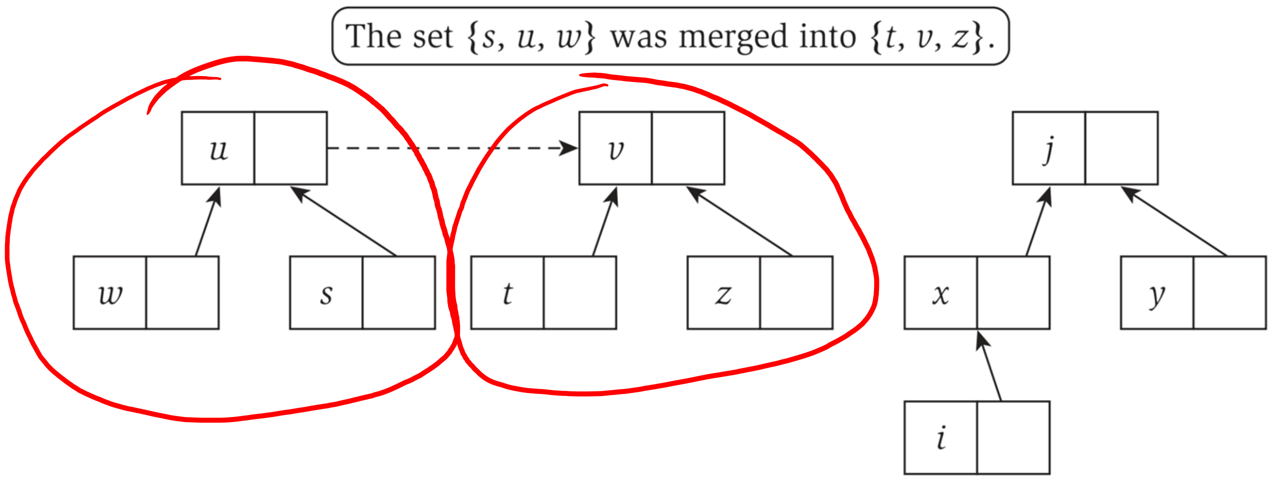
- First we bound the number of times component[u] gets updated.
- The algorithm starts with singleton set for each |V| elements.
- After first Union call the set size increased to 2. After k operations at most |V| - 2k elements are singleton.
- To change the name of the component[u] it size of its union set has to be at least twice the size of set containing u. Since, 2k elements are touched by Union() so name change happens for at most log 2k times.
- As there are 2k elements involved, the amortized running time is O(k log k).

Union Find Data Structure

Can we improve the worst case running time?

Use pointer to represent component names.

- For singleton sets the pointer is null. Hence, the element itself is the component name or root.
- Union(u, v): if size of set containing v is larger than the size of set containing u, then the pointer of u points to v else pointer of v points to u.



Union Find Data Structure

Claim: There is an improved data structure where the Union operation takes O(1) time and worst case running time of Find() is O(log |V|).

Proof:

- Consider an element u.
- The running time of Find(u) is the number of times its name changes during Union() operations.
- As we keep the name of the larger set among the two sets so the name is changed only if the other set is bigger than the set containing u.
- Every time the component name changes the size of the set at least doubles.
- Starting from size 1 to maximum size of |V|, the change can happen at max log|V| times.

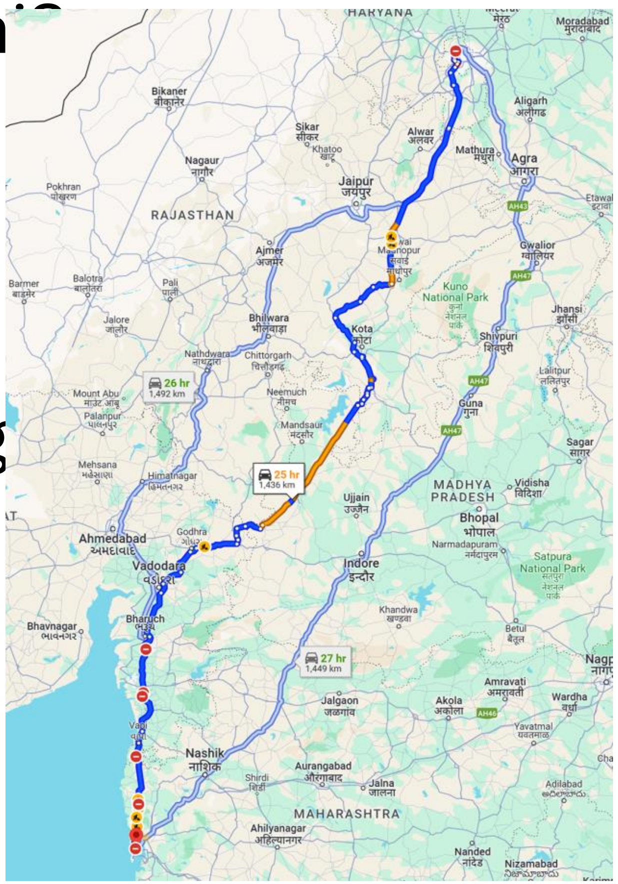
Outline

- Union Find data Structure

- Shortest Paths

Shortest Path

What is the fastest way to drive from Delhi to Mumba



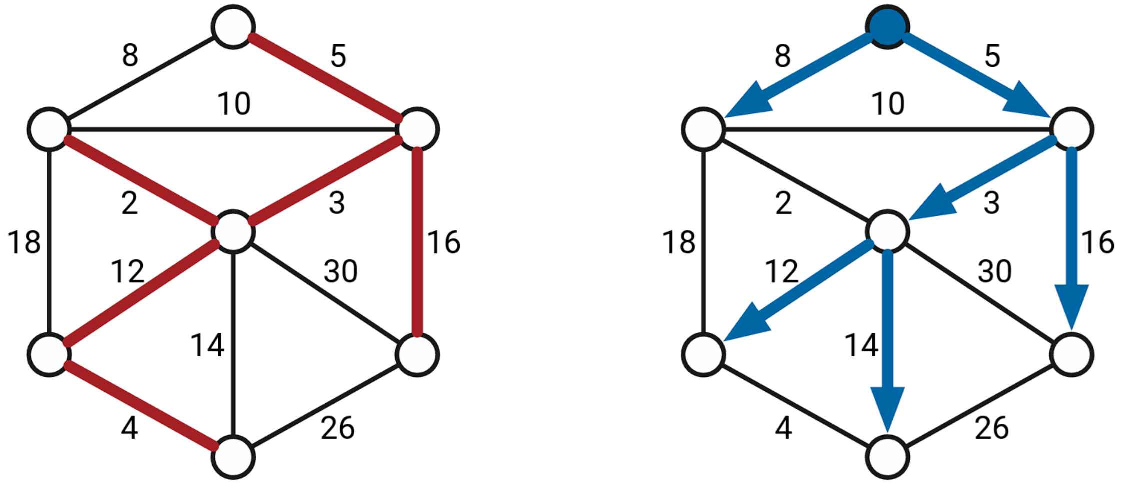
- Cities as vertices
- Roads as edges
- Time to travel between two vertices as edge weig

$$w(P) := \sum_{u \to v \in P} w(u \to v)$$

Shortest Path

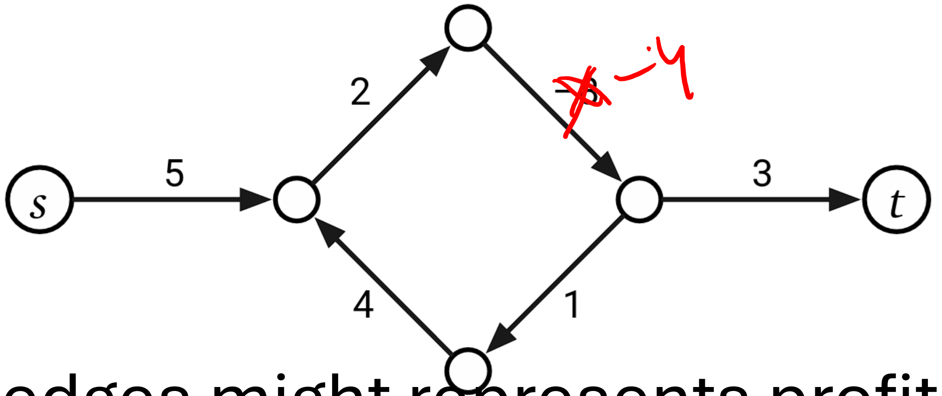
Single Source Shortest Path: It is the shortest path from source vertex (say s) to every other vertex in the graph.

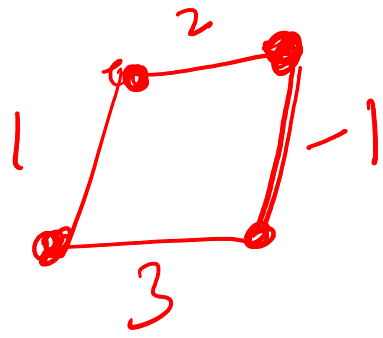
Minimum spanning tree and shortest path tree are not the same!



Shortest Path

In case of negative cycle shortest path is undefined.



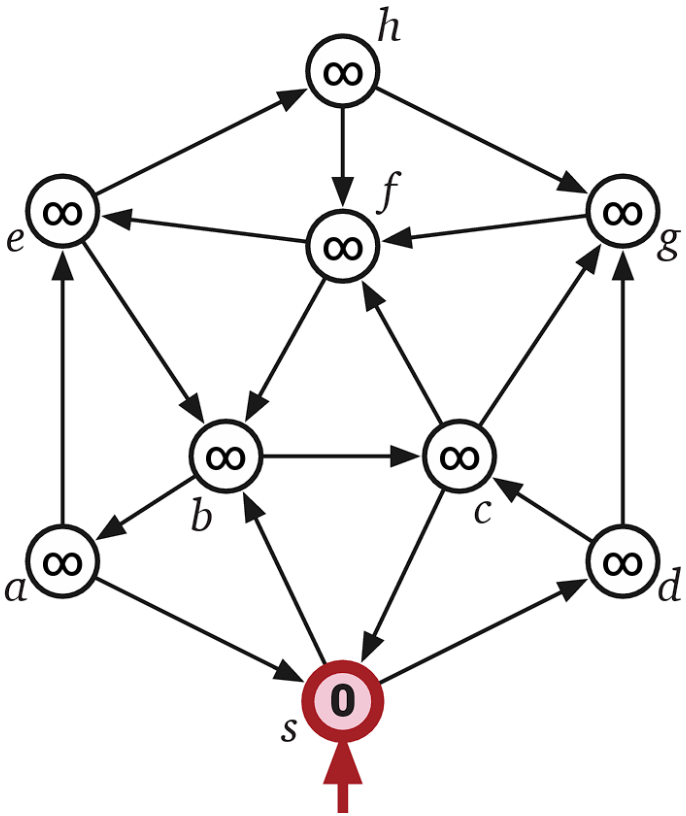


Negative and positive edges might represents profit and loss. Change -8 to -4.

In case of undirected graphs the negative edge needs to handled carefully to while computing shortest path.

Shortest Path

Consider a diagraph having all the edge weights weights as 1.

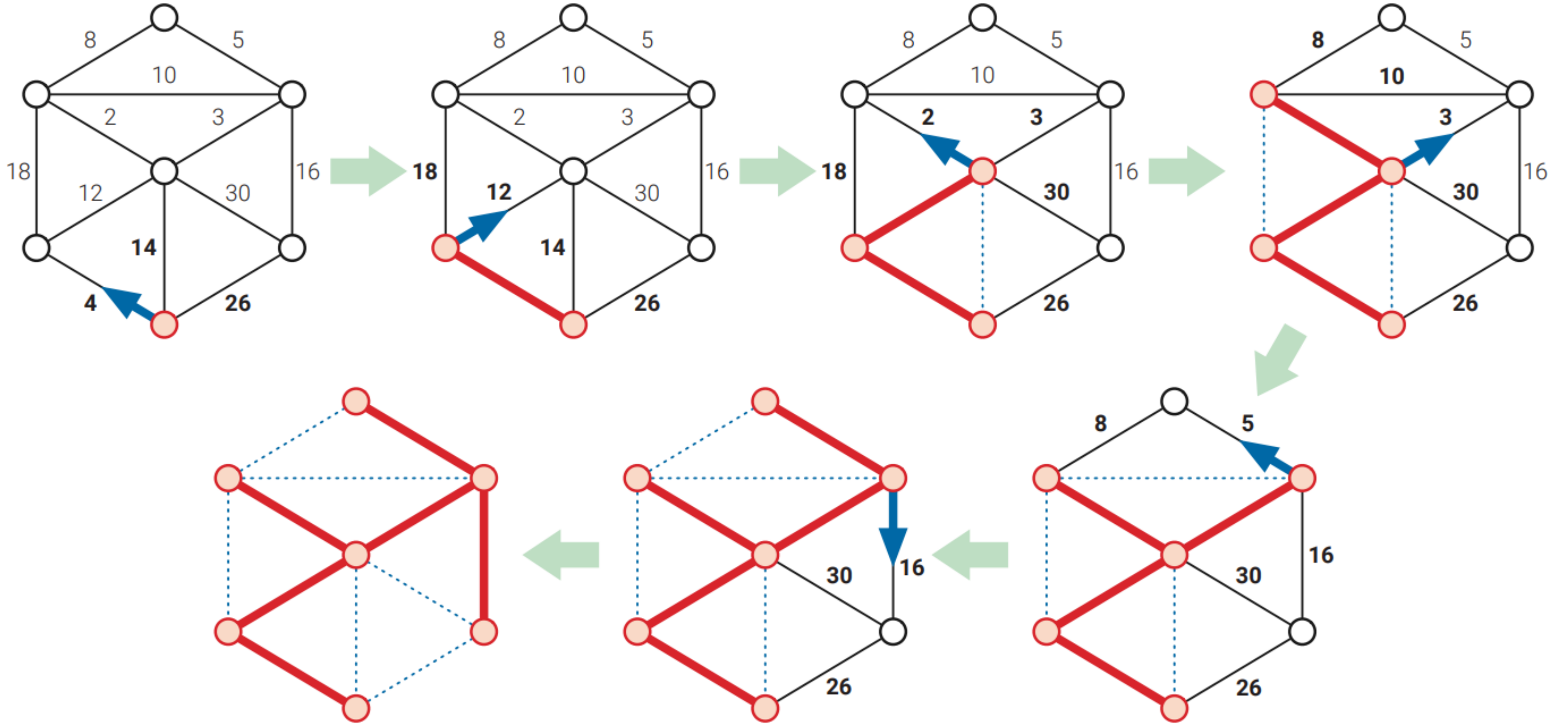


$\underline{\text{BFS}(s):}$
  $\text{INITSSSP}(s)$
  $\text{PUSH}(s)$
  while the queue is not empty
    $u \leftarrow \text{PULL}()$
    for all edges $u \to v$
      if $dist(v) > dist(u) + 1$
        $dist(v) \leftarrow dist(u) + 1$
        $pred(v) \leftarrow u$
        $\text{PUSH}(v)$

BFS Shortest Path

```
BFSWithToken(s):
  InitSSSP(s)
  Push(s)
  Push(✠)                          ⟨⟨start the first phase⟩⟩
  while the queue contains at least one vertex
      u ← Pull()
      if u = ✠
          Push(✠)                  ⟨⟨start the next phase⟩⟩
      else
          for all edges u→v
              if dist(v) > dist(u) + 1          ⟨⟨if u→v is tense⟩⟩
                  dist(v) ← dist(u) + 1
                  pred(v) ← u                   ⟨⟨relax u→v⟩⟩
                  Push(v)
```

BFS Shortest Path




```
BFSWithToken(s):
    InitSSSP(s)
    Push(s)
    Push(✠)                              ⟨⟨start the first phase⟩⟩
    while the queue contains at least one vertex
        u ← Pull()
        if u = ✠
            Push(✠)                      ⟨⟨start the next phase⟩⟩
        else
            for all edges u→v
                if dist(v) > dist(u) + 1         ⟨⟨if u→v is tense⟩⟩
                    dist(v) ← dist(u) + 1
                    pred(v) ← u                  ⟨⟨relax u→v⟩⟩
                    Push(v)
```

BFS Shortest Path

Claim: For every integer $i \geq 0$, and every vertex v at the end of ith phase, either $dist(v) = \infty$ or $dist(v) \leq i$ and v is the queue if and only if $dist(v) = i$.

Proof: By induction.
at $i = 0$, $dist(s) = 0$ and $dist(v) = \infty$ for every $v \neq s$.
Fix $i > 0$, the queue has $dist(u) = i\text{-}1$ followed by the token. $\rightarrow \; \maltese \; i-1 \; i-1 \; \cdots \; i-1 \; \rightarrow$
Before we pull out the token we pull out every vertex u s.t. $dist(u) = i\text{-}1$
For every $u \rightarrow v$, if $dist(v) > \infty$ then set $dist(v) = i$ and push v.
So, before we pull the token we have, $\rightarrow \; i \; i \; \cdots \; i \; \maltese \; \rightarrow$
Finally after phase i we have $\rightarrow \; \maltese \; i \; \cdots \; i$

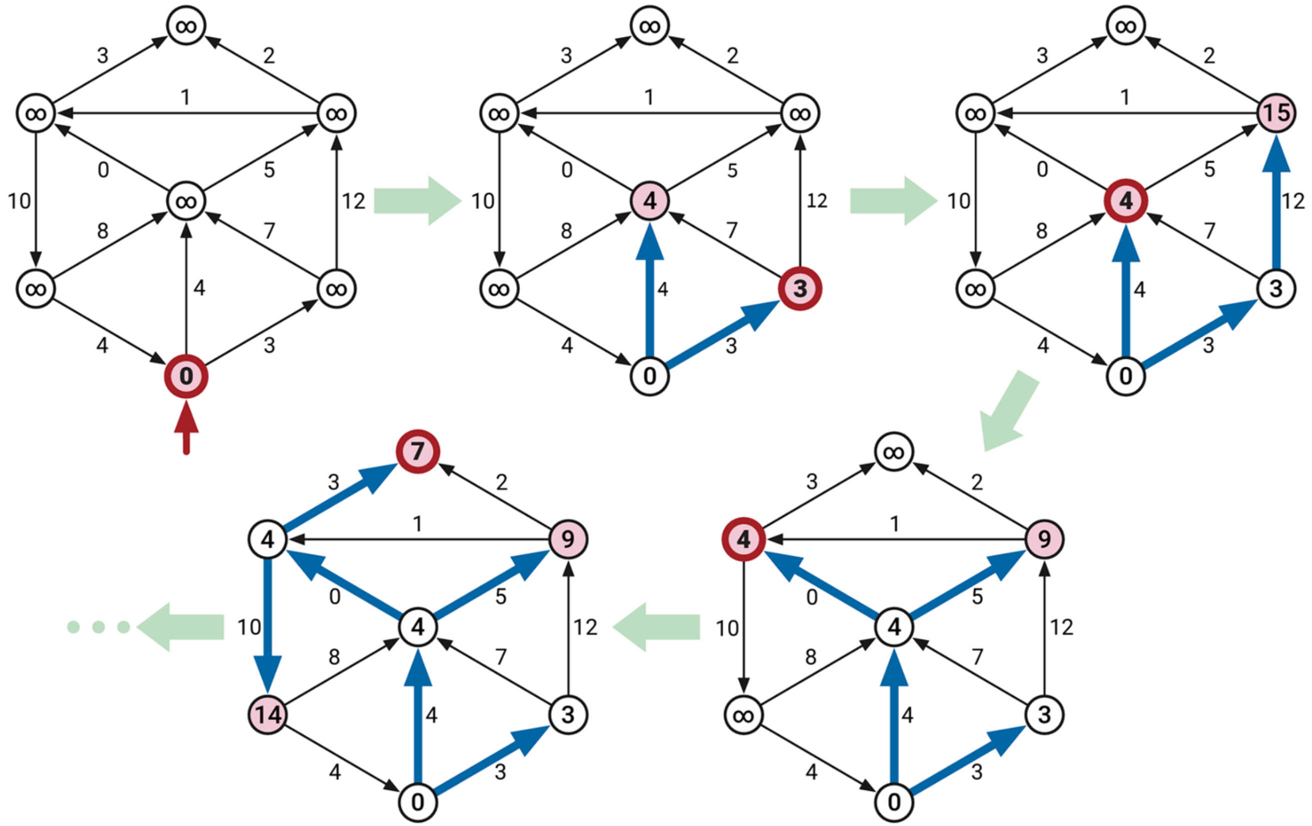
BFS Shortest Path

Claim: The dist(v) is the length of the shortest path from s to v in the graph G.

Proof: Fix an arbitrary path then prove by induction.

Dijkstra’s Algorithm

Replace the FIFO queue in BFS shortest path with a priority queue.



Dijkstra's Algorithm

```
DIJKSTRA(s):
    INITSSSP(s)
    INSERT(s, 0)
    while the priority queue is not empty
        u ← EXTRACTMIN( )
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                if v is in the priority queue
                    DECREASEKEY(v, dist(v))
                else
                    INSERT(v, dist(v))
```

Reference

Slides

Jeff Erickson Chp-8

Algorithms Design by Kleinberg & Tardos - Chp 4.6