Greedy Algorithms

Neelima Gupta
ngupta@cs.du.ac.in

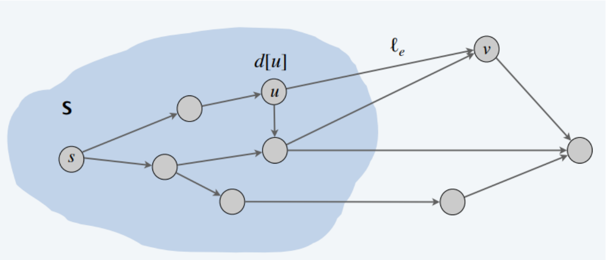
Table of Contents

Shortest Path Problem

Given a directed graph $G = (V, E)$ with edge lengths ℓ and a pair s, t of the vertices. Aim is to find a shortest path from s to t.

Djikstra's Algorithm

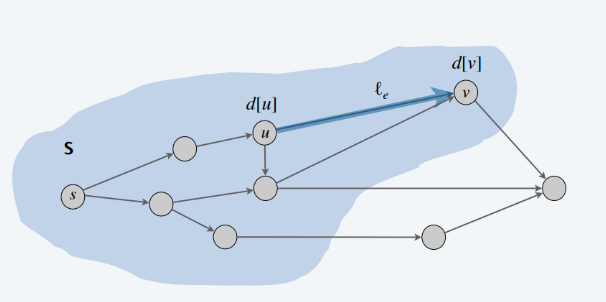
- At any point of time, maintain a set S of explored nodes, for which, shortest path has been computed. Initially, $S \leftarrow \{s\}, d[s] \leftarrow 0$.[1]
- Greedy Choice: Repeatedly choose unexplored node $v \notin S$ which minimizes, $\pi(v) = min_{e=(u,v):u \in S} d[u] + \ell_e$.
- Add v to S, and set $d[v] \leftarrow \pi(v)$.



[1] Slides are based on https://www.cs.princeton.edu/ wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsII.pdf

Djikstra's Algorithm

- Initialize $S \leftarrow \{s\}, d[s] \leftarrow 0$.
- Greedy Choice: Repeatedly choose unexplored node $v \notin S$ which minimizes, $\pi(v) = min_{e=(u,v):u \in S} d[u] + \ell_e$.
- Add v to S, and set $d[v] \leftarrow \pi(v)$.
- To recover path, set $pred[v] \leftarrow u$ that achieves the min.



Proof of Invariant

Let P_u denotes the $s - u$ path consisting of Djikstra's edges. Then clearly, $d[u] = \ell(P_u)$ from the algorithm.

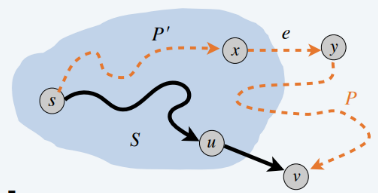
Invariant: For each node $u \in S$:P_u is a shortest $s - u$ path i.e., $d[u] =$ length of a shortest sâu path.

Proof by Induction:

Base Case: $|S| = 1$ is easy since $S = \{s\}$ and $d[s] = 0$

Induction Hypothesis: Assume true for $|S| \geq 1$.

- Let v be the next node added to S. Suppose v is added via u i.e. using the edge (u, v).



- $P_v = P_u$ followed by (u, v).
$\ell(P_v) = \ell(P_u) + \ell_{(u,v)} = d[u] + \ell_{(u,v)} = \pi(v)$

Proof of Invariant

Let P_u denotes the $s - u$ path consisting of Djikstra's edges. Then clearly, $d[u] = \ell(P_u)$ from the algorithm.

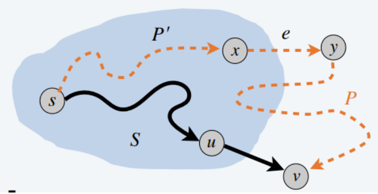
Invariant: For each node $u \in S$:P_u is a shortest $s - u$ path i.e., $d[u]$ = length of a shortest sâu path.

Proof by Induction:

Base Case: $|S| = 1$ is easy since $S = \{s\}$ and $d[s] = 0$

Induction Hypothesis: Assume true for $|S| \geq 1$.

- Let v be the next node added to S. Suppose v is added via u i.e. using the edge (u, v).



- $P_v = P_u$ followed by (u, v). $\ell(P_v) = \ell(P_u) + \ell_{(u,v)} = d[u] + \ell_{(u,v)} = \pi(v)$

Proof of Invariant

Let P_u denotes the $s - u$ path consisting of Djikstra's edges. Then clearly, $d[u] = \ell(P_u)$ from the algorithm.

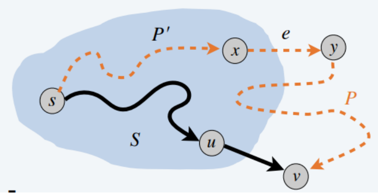
Invariant: For each node $u \in S$:P_u is a shortest $s - u$ path i.e., $d[u]$ = length of a shortest sâu path.

Proof by Induction:

Base Case: $|S| = 1$ is easy since $S = \{s\}$ and $d[s] = 0$

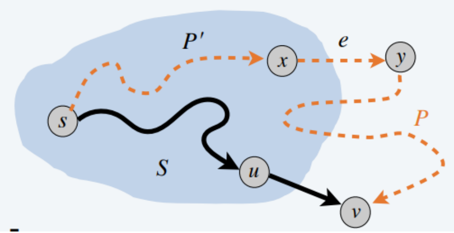
Induction Hypothesis: Assume true for $|S| \geq 1$.

- Let v be the next node added to S. Suppose v is added via u i.e. using the edge (u, v).

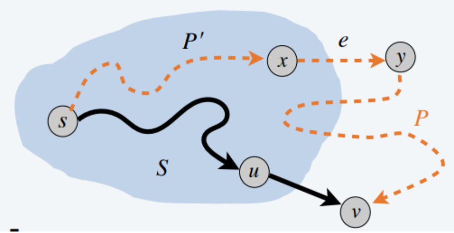


- $P_v = P_u$ followed by (u, v).
  $\ell(P_v) = \ell(P_u) + \ell_{(u,v)} = d[u] + \ell_{(u,v)} = \pi(v)$

Proof of Invariant

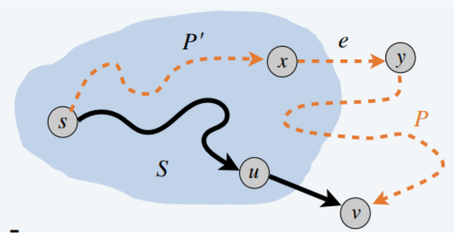


- Consider any other sâv path P (need not consist of Djikstra's edges). Claim: $\ell(P) \geq \pi(v)$.
- Let $e = (x, y)$ be the first edge in P that leaves S, and let P' be the sub-path from s to x.



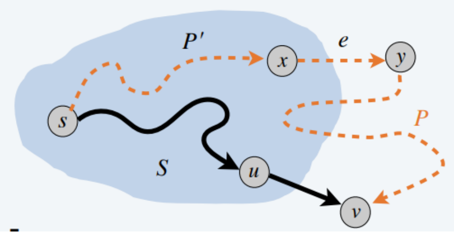
- Consider any other s⌢v path P (need not consist of Djikstra's edges). Claim: $\ell(P) \geq \pi(v)$.
- Let $e = (x, y)$ be the first edge in P that leaves S, and let P' be the sub-path from s to x.

Proof of Invariant



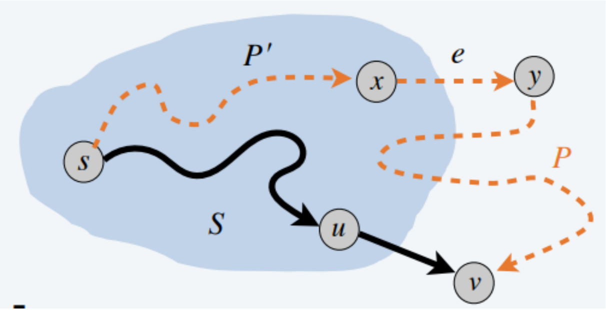
- The length of P is already $\geq \pi(v)$ as soon as it reaches y. (why?)
  Recall that $\pi(y) = min_{e=(u,y):u \in S}\{d[u] + \ell_e\}$. Hence
  $\pi(y) \leq d[x] + \ell_{(x,y)}$
  $\quad \leq \ell(P') + \ell_{(x,y)}$ (by induction hypothesis)
  (P' is some path from s to x not necessarily consisting of the edges picked by DA).
  And, since DA chose v and not y, we have $\pi(v) \leq \pi(y)$.

Proof of Invariant



- The length of P is already $\geq \pi(v)$ as soon as it reaches y. (why?) Recall that $\pi(y) = min_{e=(u,y):u \in S}\{d[u] + \ell_e\}$. Hence
  $\pi(y) \leq d[x] + \ell_{(x,y)}$
  $\leq \ell(P') + \ell_{(x,y)}$ (by induction hypothesis)
  (P' is some path from s to x not necessarily consisting of the edges picked by DA).
  And, since DA chose v and not y, we have $\pi(v) \leq \pi(y)$.

Proof of Invariant



Thus,

$$\ell(P) \underset{\substack{\uparrow \\ \text{non-negative} \\ \text{lengths}}}{\geq} \ell(P') + \ell_e \underset{\substack{\uparrow \\ \text{inductive} \\ \text{hypothesis}}}{\geq} d[x] + \ell_e \underset{\substack{\uparrow \\ \text{definition} \\ \text{of } \pi(y)}}{\geq} \pi(y) \underset{\substack{\uparrow \\ \text{Dijkstra chose } v \\ \text{instead of } y}}{\geq} \pi(v) \quad \blacksquare$$

Efficient Implementation

Critical optimization 1. For each unexplored node $v \notin S$:
explicitly maintain $\pi[v]$ instead of computing directly from definition

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\;:\;u \in S} \; d[u] + \ell_e$$

- For each $v \notin S$: $\pi(v)$ can only decrease (because set S increases).

- More specifically, suppose u is added to S and there is an edge $e = (u, v)$ leaving u. Then, it suffices to update:

$$\pi[v] \leftarrow \min \{ \pi[v],\; \pi[u] + \ell_e) \}$$

recall: for each $u \in S$,
$\pi[u] = d[u] =$ length of shortest $s \rightsquigarrow u$ path

[2]

[2] Slide is taken from
https://www.cs.princeton.edu/ wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsII.pdf

Analysis

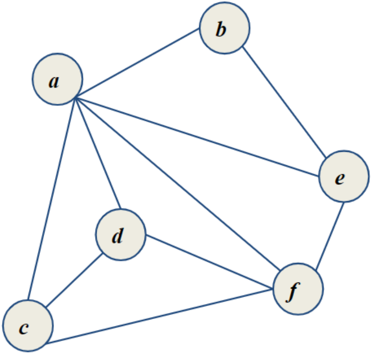
Operation	Number of times the operation is called for algorithm under consideration	Time taken by the operation	Total Time
Enqueue	V	$O(\log V)$	$O(V \log V)$
Decrease-key	E	$O(\log V)$	$O(E \log V)$
Extract-Min	$V - 1$	$O(\log V)$	$O(V \log V)$

Table: Data Structure: Priority Queue

Total time = $O((E + V)\log V) = O(E \log V)$ for a connected graph.

Spanning Tree

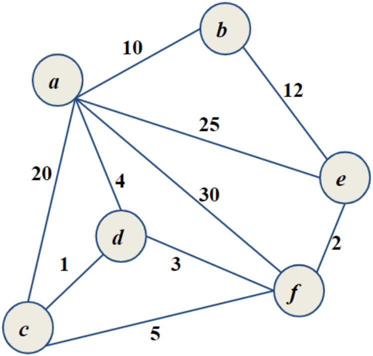
Given a connected undirected graph $G = (V, E)$, a spanning tree is a tree that spans all the vertices.



Minimum Spanning Tree

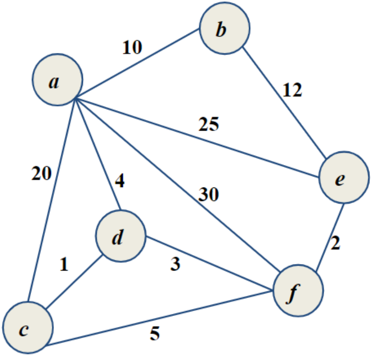
Minimum Spanning Tree

Given a connected undirected graph $G = (V, E)$ with weights on edges, a minimum spanning tree is a spanning tree with minimum total weight.



Kruskal Algorithm

1. Sort the edges in the increasing order of their weights - $e_1, e_2 \ldots e_m$.
2. While there are more edges and we have selected $< n - 1$ edges do
   Select the next edge if it does not form a cycle and discard it otherwise.



Min-Cut Property

Cut

A cut is a non trivial partition of the node set V into S and $V \setminus S$, where $S \neq \phi, V$.

Cutset

The cutset $(S, V \setminus S)$ defined by $S \subset V$ is the set of edges connecting S to $V \setminus S$.

Cut Property

The cheapest edge in every cutset belongs to the MST.

Min-Cut Property

Cut

A cut is a non trivial partition of the node set V into S and $V \setminus S$, where $S \neq \phi, V$.

Cutset

The cutset $(S, V \setminus S)$ defined by $S \subset V$ is the set of edges connecting S to $V \setminus S$.

Cut Property

The cheapest edge in every cutset belongs to the MST.

Correctness of Kruskal Algorithm : The Plan

- Acyclic ...by design
- Claim: Every edge selected by KA belongs to the MST.
  Proof:
  - We will prove that the edge picked by Kruskal is the cheapest edge in a cutset. Hence the claim follows by the Min-Cut Property
- (n -1) edges picked by KA and the fact that they do not form a cycle implies that the set of edges are same as that of MST. Hence proved.

Correctness of Kruskal Algorithm : The Plan

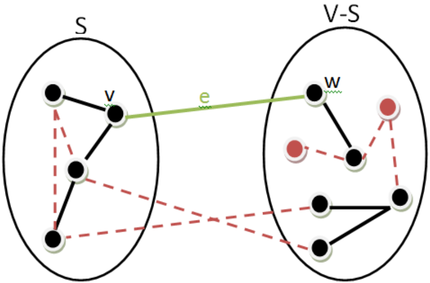
- Acyclic ...by design
- Claim: Every edge selected by KA belongs to the MST.
  Proof:
  - We will prove that the edge picked by Kruskal is the cheapest edge in a cutset. Hence the claim follows by the Min-Cut Property
- (n -1) edges picked by KA and the fact that they do not form a cycle implies that the set of edges are same as that of MST.
  Hence proved.

Proof of Claim 1: Edge picked by KA is a cheapest edge in a cutset

Let $e_j = (v, w)$ be an edge picked by Kruskal at some point of time.

Proof of Claim 1: Edge picked by KA is a cheapest edge in a cutset

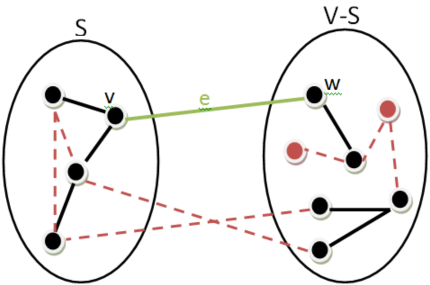
Let $e_j = (v, w)$ be an edge picked by Kruskal at some point of time.

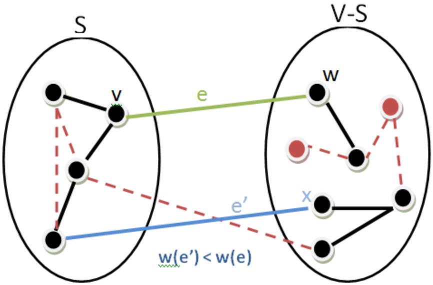


S : Set of vertices reachable from v using Kruskal edges when e_j was picked

Proof of Claim 1: Edge picked by KA is a cheapest edge in a cutset

Let $e_j = (v, w)$ be an edge picked by Kruskal at some point of time.



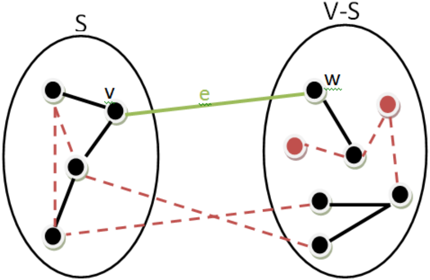


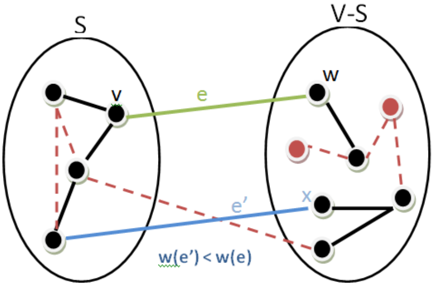
S : Set of vertices reachable from v using Kruskal edges when e_j was picked

Edges $e_1, e_2 \ldots e_{j-1}$ could not have connected S and $V \setminus S$ for else x would be reahable from v when e_j was picked.

Proof of Claim 1: Edge picked by KA is a cheapest edge in a cutset

Let $e_j = (v, w)$ be an edge picked by Kruskal at some point of time.



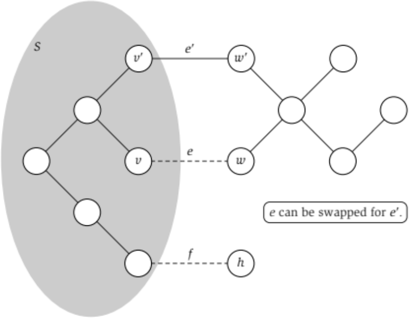


S : Set of vertices reachable from v using Kruskal edges when e_j was picked

Edges $e_1, e_2 \ldots e_{j-1}$ could not have connected S and $V \setminus S$ for else x would be reahable from v when e_j was picked.

Property: Let $e = (v, w)$ be the minimum weight edge in a cut-set $(S, V \setminus S)$. Then, MST contains e.

T is an MST not containing e.



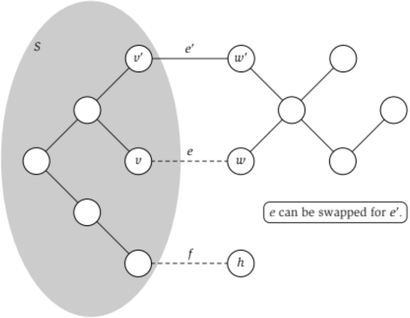
Claim: $T' = T - e' + e$ is a spanning tree with $w(T') < w(T)$.

A contradiction.

Proof of Cut Property

Property: Let $e = (v, w)$ be the minimum weight edge in a cut-set $(S, V \setminus S)$. Then, MST contains e.

T is an MST not containing e.

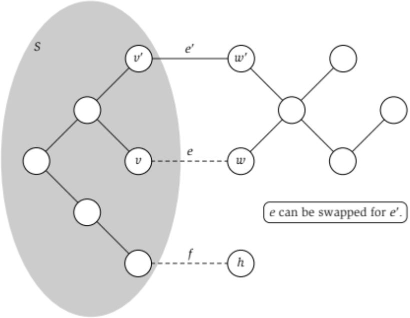


Claim: $T' = T - e' + e$ is a spanning tree with $w(T') < w(T)$.

A contradiction.

Proof of Cut Property

Property: Let $e = (v, w)$ be the minimum weight edge in a cut-set $(S, V \setminus S)$. Then, MST contains e.
T is an MST not containing e.



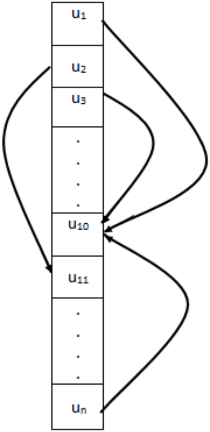
Claim: $T' = T - e' + e$ is a spanning tree with $w(T') < w(T)$.

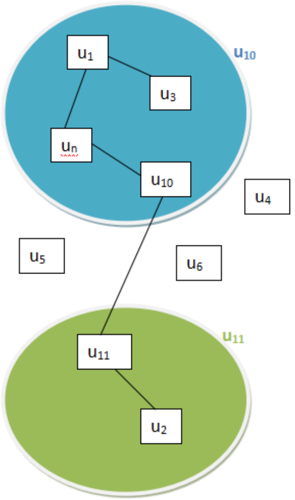
A contradiction.

Implementation

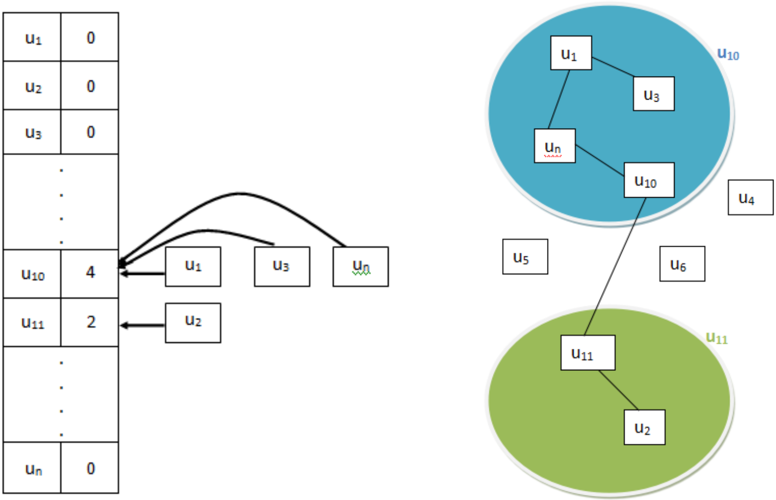
- **Disjoint Union-Find** Structure
- *Find*(u): Given a node u, the operation *Find*(u) will return the name of the set containing u.
- *Union*(A, B): Take two sets A and B and merge them to a single set.

Union Find

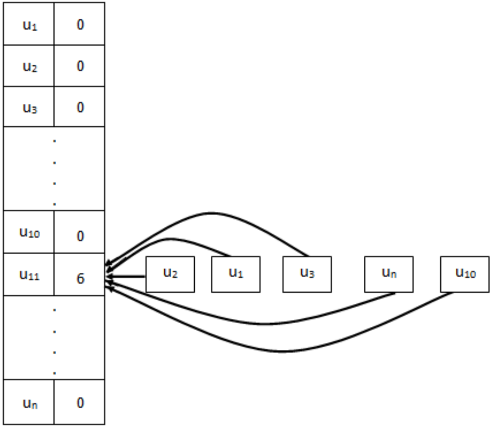


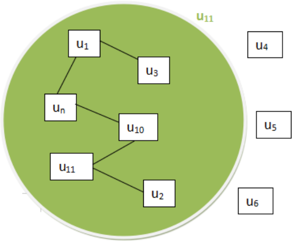


Alternate Representation



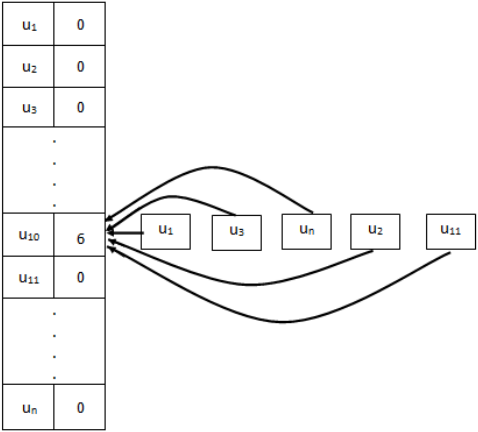
Merge all into u_{11}

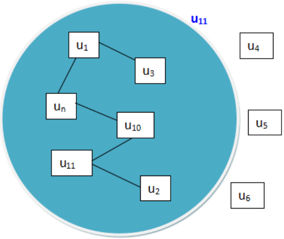




Time Complexity

Or Merge all into u_{10}





Time Complexity

Implementation

If find(u)!= find(v)
then Union (find(u),find(v))
//include that edge in the set

Time Complexity

- Sorting takes *mlogm* time, where *m* is the number of edges.
- *Find* takes constant time.
- *Union* is performed at most $n - 1$ times, where *n* is the number of vertices.
- Total number of pointer updates over all Union operations is $O(nlogn)$.
- Thus total time is $O(mlogn)$.