Dijkstra's algorithm

Dijkstra's algorithm solves the shortest path problem on a weighted directed graph.

Dijkstra's algorithm is a greedy algorithm.

Before decribing the algorithm, we recap some terminology.

Terminology

Let $G = (V, E)$ be a digraph.

The shortest path b/w two vertices is a path with the shortest length (least # of edges)

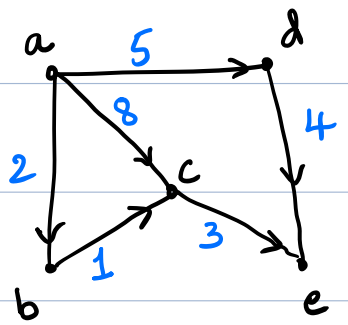
Let $G = (V, E)$ be a weighted digraph, with weight function $w : E \to \mathbb{R}$ mapping edges to real valued weights. If $e = uv$, we write w_e or w_{uv} or $w(u, v)$

The length of the path $P = v_0, v_1, \ldots v_k$ is the sum of the weights of its constituent edges.

$$\text{length}(P) = \sum_{i=1}^{k} w_{v_{i-1} v_i}$$

The distance from u to v, denoted $\delta(u, v)$ is the length of the minimum length path if there is a path from u to v; and is ∞ otherwise.

Example:



length of the path $a, c, e = 11$

distance from a to $e = \delta(a, e) = 6$

Single-Source Shortest Paths Problem [this lecture]

I/p: Directed weighted graph G and a source vertex S.

O/p: Find a shortest path from a given source vertex $s \in V$ to each vertex $v \in V$.

ie, For each $v \in V$, find $\delta(s, v)$.

Recap:

"length of path": Sum of edge weights of the path.

Applications

- Social Network : Finding the shortest path between two persons (degree of separation)

  Eg: In Linkedin, they provide some person is $2^{nd}/3^{rd}$ connection, here 2/3 is the shortest path from you to that person in the linkedin network (unweighted)

- Word ladder Puzzles:

  YES – YET – GET – GOT – GO – NO

  Given a starting word and an ending word, design an algorithm to transform one word into the other by changing, adding or deleting exactly one letter at a time, with the result being a valid English word at each step.

- Transport finished product from factory (single source) to all retail outlets.
- Courier company delivers items from distribution center to addressees.
- Google maps uses a version of Dijkstra's algorithm.

All pairs shortest paths

Find shortest paths b/w every pair of vertices i and j

(more on this after two lectures)

Initial attempt

Recap: BFS computes shortest paths from a source vertex.

Works on unweighted graphs (i.e., weight of an edge is 1)

One idea:

Replace each edge e by a directed path of length w_e.

- Edge weights may be too large, blows up the size of the graph.

So this approach may not give efficient algorithm

Dijkstra's Shortest Path Algorithm

Assumptions

① We assume that $\forall v \in V$, there is a s to v path.

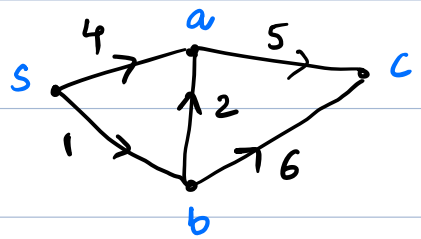
② Edge weights are non-negative.

ie, $w_e \geq 0 \quad \forall e \in E(G)$. (In some applications negative edge lengths are relevant)

↓

More on this in the next lecture.

Warmup Example:



Find shortest path distances from source vertex s to other vertices.

$\delta(s,a) = 4$, $\delta(s,b) = 1$, $\delta(s,c) = 7$

Overview of the Algorithm

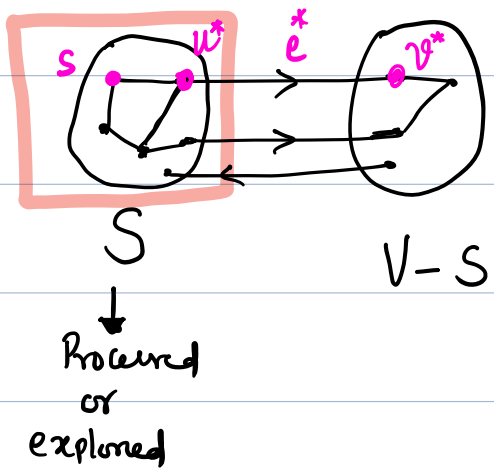
S : Vertices Processed so far and d(u) denote $\delta(s,u)$

Initialization:

$S = \{s\}$

$d(s) = 0$

Main Part



Greedy choice

- Among all edges $e=uv$ with $u \in S$ and $v \in V-S$

  Pick the one that minimizes

  $d(u) + w_e$

  [Say $e^* = u^*v^*$ is the edge which minimizes]

- Add v^* to S

  $d(v^*) = d(u^*) + w_{e^*}$

Algorithm

w: weight fun
$w: E(G) \to \mathbb{R}$

Dijkstra (G, w)

Let S be the set of explored vertices

For each $u \in S$, we store $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V(G)$

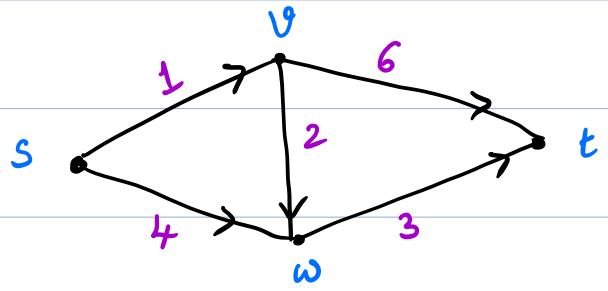
Select a vertex $v \notin S$ with at least one edge from S for which

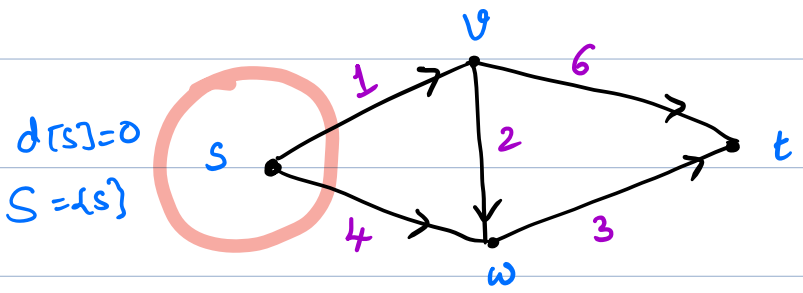
$$d'(v) = \min_{\substack{e = uv \\ u \in S}} \{d(u) + w_e\}$$ is as small as possible

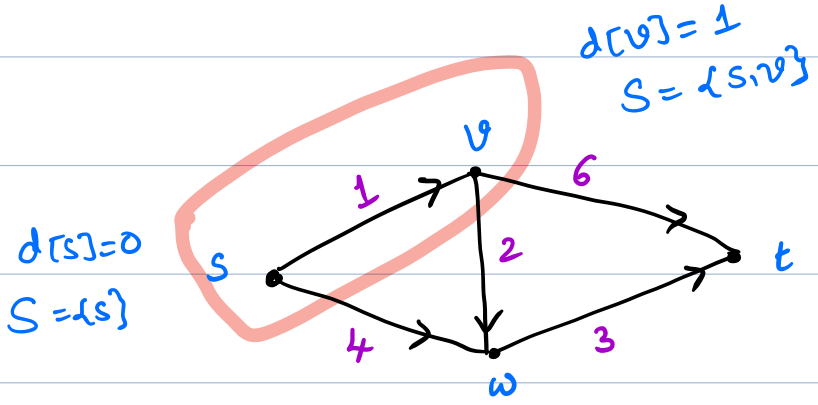
Add v to S and set $d(v) = d'(v)$

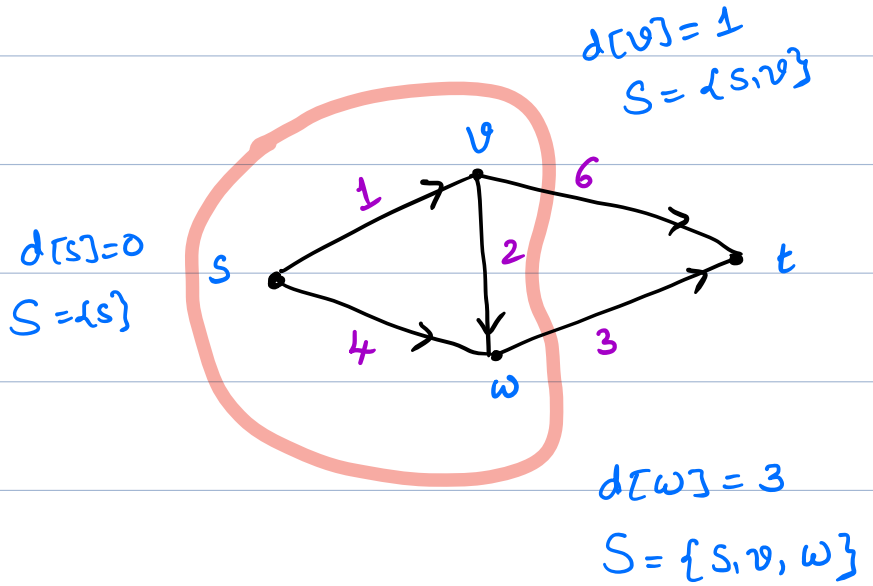
End while

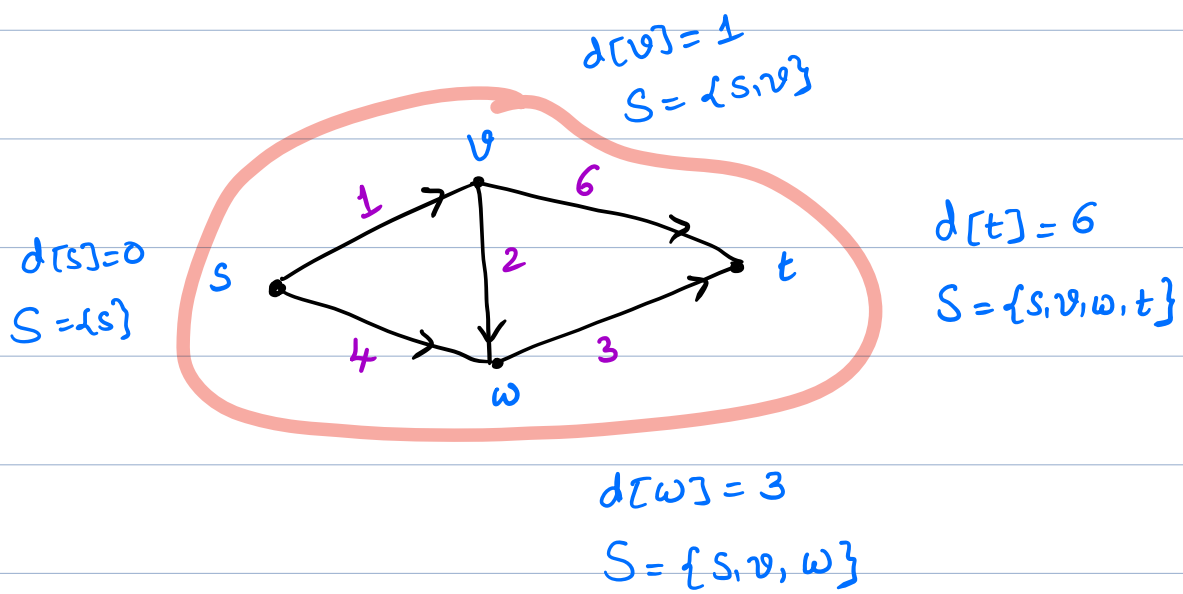
Example:











By simple book keeping we can compute the shortest paths.

Before proving the correctness of the Dijkstra's algorithm, we need the optimal-substructure property of shortest paths, which is stated in the following lemma.

Lemma ***(Subpaths of shortest paths are shortest paths)***
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Correctness of Dijkstra's Algorithm

We need to show that

$$d(u) = \delta(s,u) \quad \text{for all } u \in V(G)$$

Proof by induction on the # of iterations (or) on size of S.

Base case : $|S| = 1$

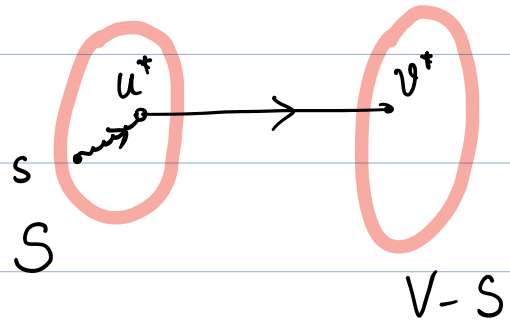
ie, $S = \{s\}$. $d(s) = 0 = \delta(s,s)$

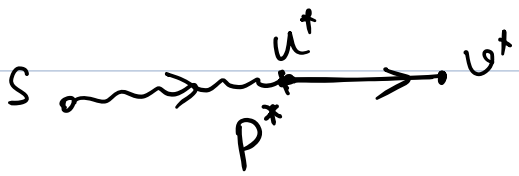
Assume the hypothesis for $k-1$ iterations, $|S| = k-1$

ie, for all $v \in S$, $d(v) = \delta(s,v)$. } Inductive hypothesis.

In the k^{th} iteration, we choose v^* according to the algorithm.

Claim : $d(v^*) = \delta(s,v^*)$





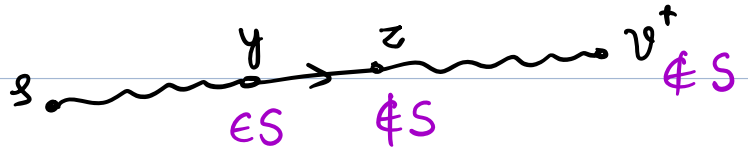
By Induction hypothesis, $d(u^*) = \delta(s, u^*)$

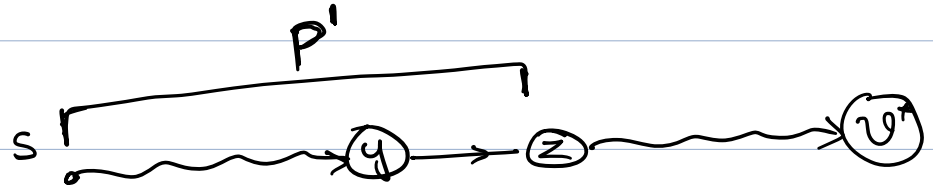
and $d(v^*) = d(u^*) + w_{u^*v^*}$

We show that length of any s to v^* path is at least $d(u^*) + w_{u^*v^*}$.

Let P be any $s - v^*$ Path.

this Path must cross S.





length of the subpath $P' \geq d(y) + w_{yz}$

By Dijkstra's greedy criterion

$$d[u^*] + w_{u'y'} \leq d[y] + w_{yz}$$

[the alternate S-v path P through y and z is already too long by the time it has left the set S]

Since the edge lengths are non-negative,

the full path P is at least as long as P',

which is at least as long as P^*

$\therefore$ length of $P \geq$ length of P^*

Implementation & Running time:

Naive implementation:

There are $n-1$ iterations of the While loop as each iteration adds a new vertex to S.

So in each iteration we consider each vertex $v \notin S$, and go through all the edges between S and v to determine the minimum $\min_{\substack{e=uv \\ u \in S}} (d(u) + w_e)$, so that we can select the node v for which this minimum is smallest.

∴ This would lead to an implementation that runs in $O(mn)$ time.

Q Can we do better than naive implementation?

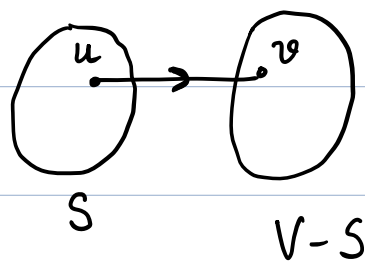
Ans YES, using Priority queue.

HEAP DATA STRUCTURE

We Put the vertices of V in a min Priority queue with $d'(v)$ as the key for $v \in V$.

To select the node v that should be added to the set S, we need the EXTRACTMIN operation.

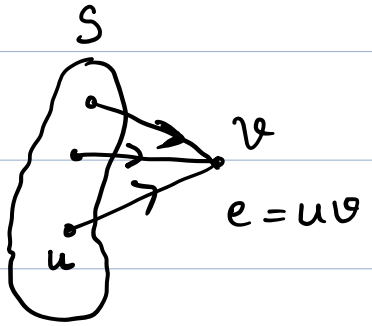
Updating the keys:



Consider an iteration in which node u is added to S, and let $v \notin S$ be a node that remains in the Priority queue.

We don't have to update $d'(v)$ if uv is not an edge, as the set of edges considered in the minimum $\min_{\substack{e=uv \\ u \in S}} d(u) + w_e$ is exactly the same before and after adding u to S.

If $uv \in E(G)$ then the new value for the key is $\min(d'(v), d(u) + w_e)$



If $d'(v) > d(u) + w_e$ then we need to use the CHANGE KEY operation to decrease the key of node v accordingly.

CHANGEKEY operation can occur at most once per edge.

Running time:

$O(m)$ + n Extract min + m change key operations

$= O(m) + n\,O(\log n) + m\,O(\log n)$

$= O((n+m)\log n)$

$= O(m \log n)$.

($m \geq n$ as all vertices are reachable from s)

graphs having negative edge weights:

(Q) Does Dijkstra's algorithm work in this case?

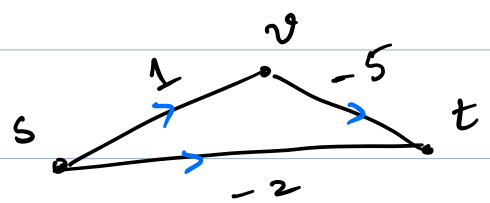
Ans See Next Page.

Idea: (to get away with negative edge weights)

Add some large constant to each weight so that all edge weights becomes non-negative.

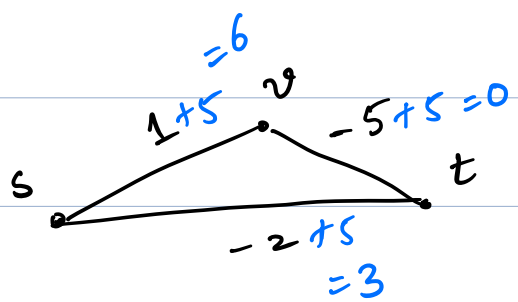
(*) But this doesn't Preserve Shortest Paths.

Example:



$d(s,t) = -4$

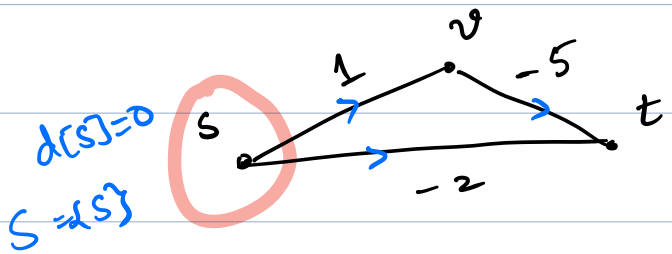
$s \rightsquigarrow v \rightsquigarrow t$

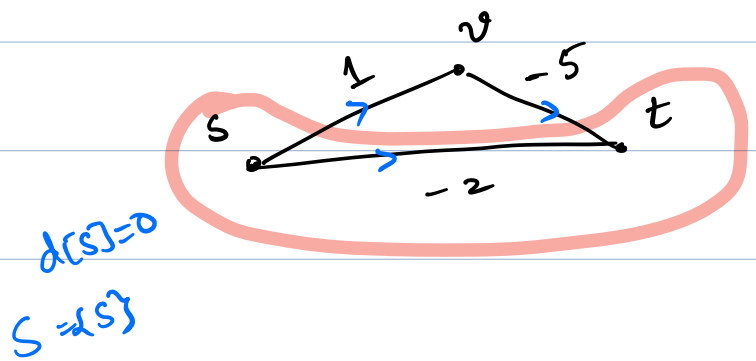


$d(s,t) = 3$

$s \rightarrow t$

Run Dijkstra's Algorithm on the same example





$S = \{s, t\}$

$d[t] = -2$

↓

This is wrong

Dijkstra fails here.