Shortest Path Problem

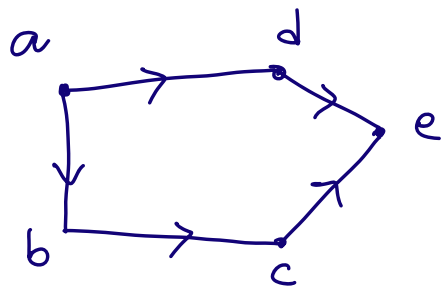
Today's class

- Dijkstra's algorithm (Greedy Algorithm)

Unweighted directed/undirected graphs

let $G = (V, E)$ be a graph.

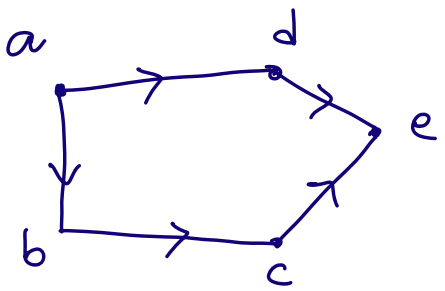
The **Shortest Path** b/w two vertices is a Path with the shortest length (least # of edges)



The shortest path b/w a & e is

$$P = a, d, e$$

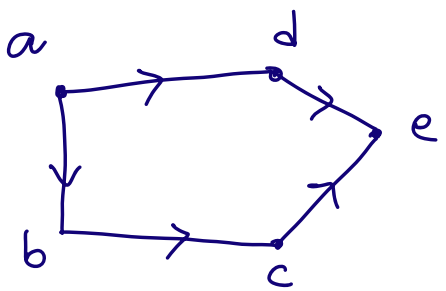
$$\text{length}(P) = 2$$



- There is NO directed Path b/w two vertices, then the length of the shortest Path b/w them is ∞.

Notation:

We use $\delta(u,v)$ to denote the length of the shortest path b/w u and v.



$\delta(a,b) = 1$

$\delta(a,e) = 2$

$\delta(d,a) = \infty$

Problem

I/p: Directed unweighted 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 $\delta(s, v)$.

Q: How to solve the above problem ?

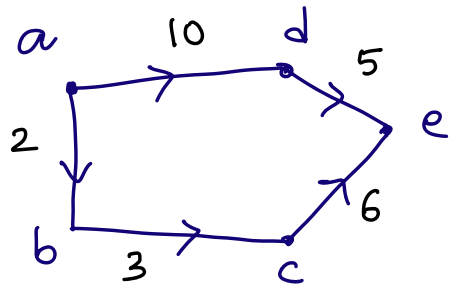
Weighted Graphs

Let $G = (V, E)$ be a weighted digraph, with weight function $\omega : E \longrightarrow \mathbb{R}$.

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

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

Example



The Shortest Path b|w a & e is

$$P = a, b, c, e$$

$$\text{length}(P) = 2+3+6$$
$$\delta(a,b) = 11$$

- $\delta(b,e) = 9$
- $\delta(b,a) = \infty$

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 $\delta(s, v)$.

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~~. Shortest Path algo

Back to the Problem.

↓

Single Source Shortest Path
on weighted directed graphs.

Initial Ideas are welcome.

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:

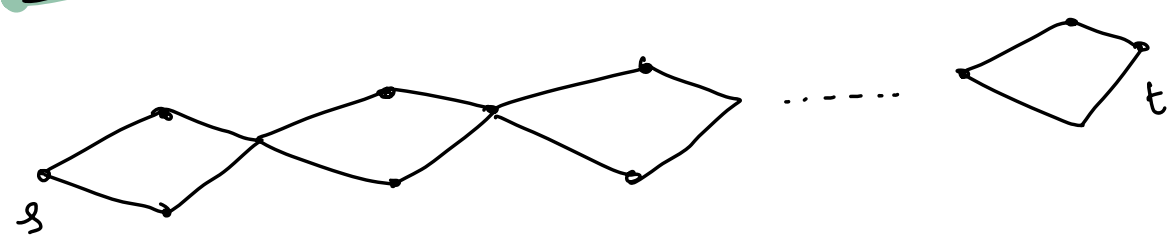
Replace each edge e by a directed path of length w_e.

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

So this approach may not give efficient algorithms

Q1: Given two nodes s & t in G, how many s-t Paths can be there in G in the worst case.

Example



k-diamonds

↓

3k+1 vertices

↓

2^k s-t Paths (Shortest)

ie, $n = 3k+1$

$k = \frac{n-1}{3}$

ie, $2^{\frac{n-1}{3}}$ s-t Paths

↓

Exponential. } too many

In this lecture

we look at a special case of

the Problem

Assumptions

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

② Edge weights are **non-negative**.

ie, $w_e \geq 0 \quad \forall e \in E(G)$.

↓

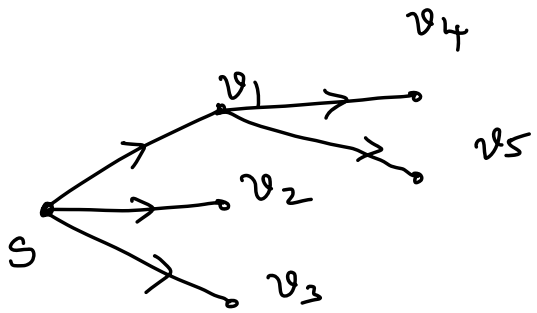
IMPORTANT

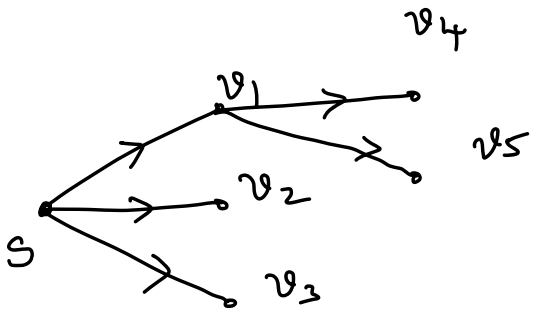
(In some applications negative edge lengths are relevant)

↓

More on this in the next ~~lecture~~ course.

Warmup Ideas





T/F:

For every out-neighbor v_i of s

$$\delta(s, v_i) = \text{weight of the edge } sv_i$$
$$= w(s, v_i)$$

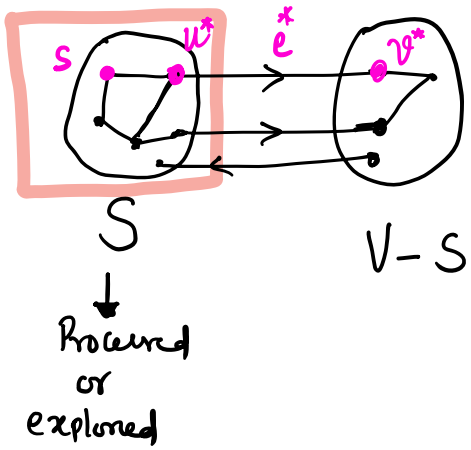
Overview of the Algorithm

S : vertices processed so far and $d(u)$ denote $\delta(s,u)$

Initialization:

$$S = \{s\} \qquad d(s) = 0$$

Main Part



Greedy choice

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

  Pick the one that minimizes

  $$d(u) + \omega_e$$

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

- Add v^* to S

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

let S be the set of explored vertices

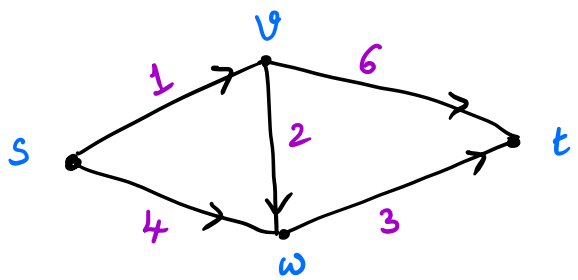
For each $u \in S$, we store $d(u)$

Algorithm

Dijkstra (G, w)

1. Initially $S = \{s\}$, $d(s) = 0$, $d(v) = \infty$, $v \in V - \{s\}$
   $v.\pi = NIL$
2. While $S \neq V(G)$
3. 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
4. Add v to S and set $d(v) = d'(v)$ &
   $v.\pi = u$
5. End while

Example:



Implementation & Running time

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.

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

Q Can we do better than naive implementation ?

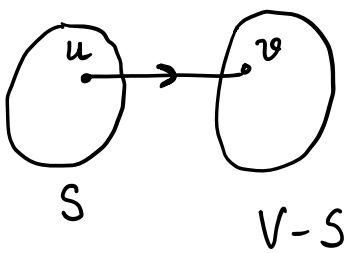
Ans YES, Using Priority queue.

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.

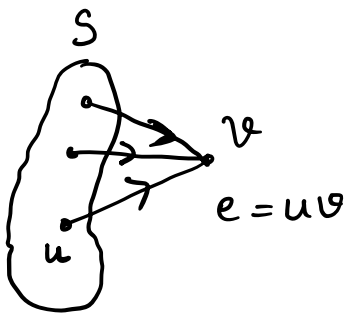
Updating the keys:



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)

Q: What about graphs having negative weight edges?

graphs having negative edge weights:

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

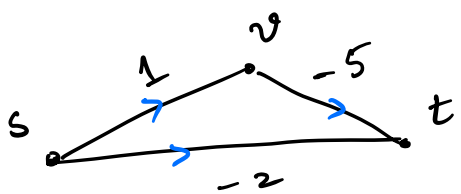
Ans See Next Page.

Idea: (to get away with negative edge weights)

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

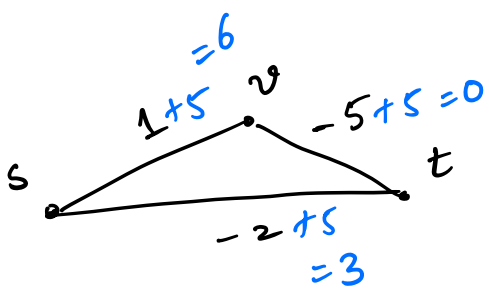
(X) But this doesn't Preserve Shortest Paths.

Example:



$d(s,t) = -4$

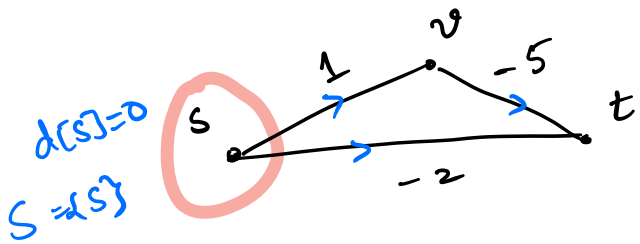
$s \rightsquigarrow v \rightsquigarrow t$



$d(s,t) = 3$

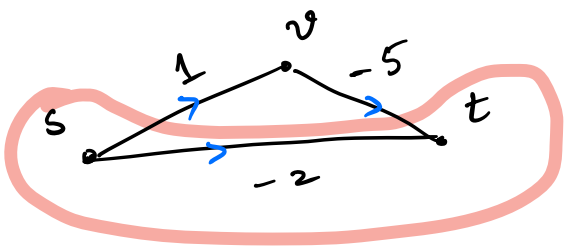
$s \rightarrow t$

Run Dijkstra's algorithm on the same example



d[s]=0

S = {s}



d[s]=0

S = {s}

S = {s, t}

d[t] = -2

↓

This is wrong

Dijkstra fails here.