Shortest Path Algorithm

Shortest Path

A **shortest path** from u to v is a path of minimum weight from u to v.

The **shortest path weight** from u to v is defined as

δ(u, v) = min{w(p) : p is a path from u to v}.

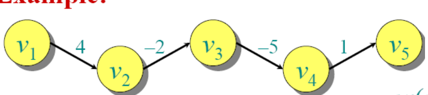
Note: δ(u, v) = ∞ if no path from u to v exists.

Paths In Graph

- Directed graph (digraph) $G = (V,E)$
- Weight function W
- Weight of path $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

Example:



$w(p) = -2$

Shortest Paths

Finding the shortest path between two nodes in a graph arises in many different applications:

- Transportation problems – finding the cheapest way to travel between two locations.
- Motion planning – what is the most natural way for a cartoon character to move about a simulated environment.
- Telephone communication costs
- Computer networks response times.

Shortest-Path Problems

– **Single-source (single-destination).** Find a shortest path from a given source (vertex *s*) to each of the vertices.

– **Single-pair.** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently.

– **All-pairs.** Find shortest-paths for every pair of vertices. Dynamic programming algorithm.

Dijkstra's Algorithm

- Non-negative edge weights
- Uses Greedy Appoarch, similar to Prim's algorithm
- A weighted version of breadth-first search.
  - Instead of a FIFO queue, uses a priority queue.
- Basic idea
  - maintain a set *S* of solved vertices
  - at each step select "closest" vertex *u*, add it to *S*, and relax all edges from *u*

Algo

Have two sets of vertices:
• *S* = vertices whose final shortest-path weights are determined,
• *Q* = priority queue = *V* − *S*.

DIJKSTRA*(V, E,w, s)*
INIT-SINGLE-SOURCE*(V, s)*
$S \leftarrow \emptyset$
$Q \leftarrow V$ // i.e., insert all vertices into *Q*
while *Q* != ∅
do $u \leftarrow$ EXTRACT-MIN*(Q)*
$S \leftarrow S \cup \{u\}$
for each vertex $v \in Adj[u]$
do RELAX*(u, v,w)*

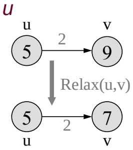
DECREASE-KEY

Initialization

INIT-SINGLE-SOURCE*(V, s)*
for each $v \in V$
do $d[v] \leftarrow \infty$
$\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

Relaxation

- Relaxing an edge (*u*,*v*) means testing whether we can improve the shortest path to *v* found so far by going through *u*



RELAX(*u, v,w)*
if *d*[*v*] > *d*[*u*] + *w(u, v)*
then *d*[*v*] ← *d*[*u*] + *w(u, v)*
π[*v*]← *u*

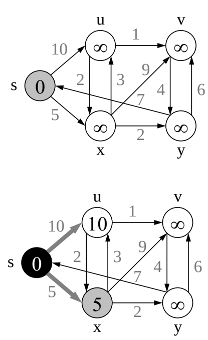
Algo

Have two sets of vertices:
• *S* = vertices whose final shortest-path weights are determined,
• *Q* = priority queue = *V* − *S*.

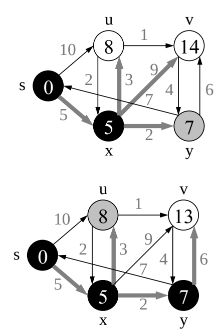
DIJKSTRA*(V, E,w, s)*
INIT-SINGLE-SOURCE*(V, s)*
S ← ∅
Q ← *V* // i.e., insert all vertices into *Q*
while *Q* != ∅
do *u* ← EXTRACT-MIN*(Q)*
S ← *S* ∪ {*u*}
for each vertex *v* ∈ *Adj*[*u*]
do RELAX*(u, v,w)*

DECREASE-KEY

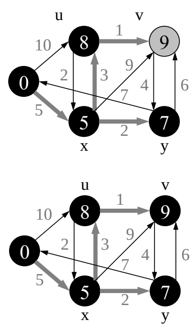
Dijkstra's Example



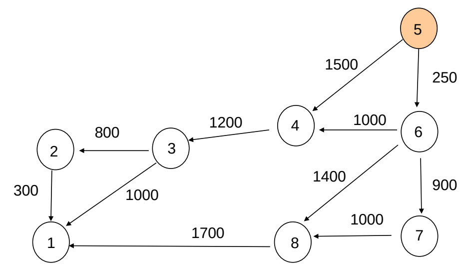
Dijkstra's Example (2)



Dijkstra's Example (3)



Example



SOLUTION

Vertex	Sel	1	2	3	4	5	6	7	8
5	-	X	X	X	1500	0	250	X	X
5,6	6	X	X	X	1250	0	250	1150	1650
5,6,7	7	X	X	X	1250	0	250	1150	1650
5,6,7, 4	4	X	X	2450	1250	0	250	1150	1650
5,6,7, 4,8	8	2350	X	2450	1250	0	250	1150	1650
5,6,7, 4,8,3	3	2350	3250	2450	1250	0	250	1150	1650
5,6,7, 4,8,3, 2	2	2350	3250	2450	1250	0	250	1150	1650

Dijkstra's Running Time

- Extract-Min executed $|V|$ time
- Decrease-Key executed $|E|$ time
- Time = $|V|\ T_{\text{Extract-Min}} + |E|\ T_{\text{Decrease-Key}}$
- T depends on different Q implementations

Q	T(Extract-Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$	$O(V \lg V + E)$