Why Study Shortest Path Problem ?

Why Study Shortest Path Problem ?

- **Traveling on a "Starving Student" budget:**
  - What is the cheapest multi-stop airline schedule from Kolkata to New York ?

Why Study Shortest Path Problem ?

- **Traveling on a "Starving Student" budget:**
  - What is the cheapest multi-stop airline schedule from Kolkata to New York ?
- **Optimizing routing of packets on the internet:**
  - vertices=routers, edges=network links with different delays
  - What is the routing path with smallest total delay ?

Why Study Shortest Path Problem ?

- **Traveling on a "Starving Student" budget:**
  - What is the cheapest multi-stop airline schedule from Kolkata to New York ?
- **Optimizing routing of packets on the internet:**
  - vertices=routers, edges=network links with different delays
  - What is the routing path with smallest total delay ?
- **Hassle-Free Commuting:**
  - Deciding which highways and roads to take to minimize total delay due to traffic ?

Unweighted Shortest Path Problem

Instance: An unweighted graph $G = (V, E)$ with $s \in V$, named as "source"

Goal: To find the shortest path (in terms of length of the path) from s to all vertices in G.

Unweighted Shortest Path Problem

Instance: An unweighted graph $G = (V, E)$ with $s \in V$, named as "source"

Goal: To find the shortest path (in terms of length of the path) from s to all vertices in G.

Solution

Unweighted Shortest Path Problem

Instance: An unweighted graph $G = (V, E)$ with $s \in V$, named as "source"

Goal: To find the shortest path (in terms of length of the path) from s to all vertices in G.

Solution

A BFS can do the job

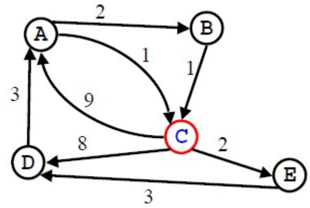
What if edges have weights

What if edges have weights

Does BFS still work for finding minimum cost paths ?

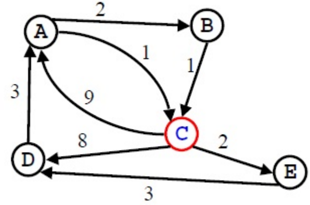
What if edges have weights

Does BFS still work for finding minimum cost paths ?



What if edges have weights

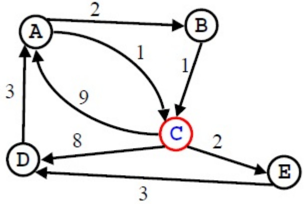
Does BFS still work for finding minimum cost paths ?



Shortest path from C to A

What if edges have weights

Does BFS still work for finding minimum cost paths ?

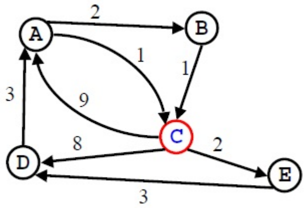


Shortest path from C to A

- BFS: C$\rightarrow$ A and cost is 9.

What if edges have weights

Does BFS still work for finding minimum cost paths ?

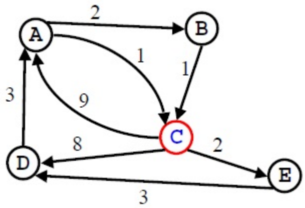


Shortest path from C to A

- BFS: C→ A and cost is 9.
- Minimum cost path: C → E → D → A

What if edges have weights

Does BFS still work for finding minimum cost paths ?

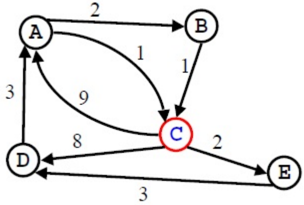


Shortest path from C to A

- BFS: C$\rightarrow$ A and cost is 9.
- Minimum cost path: C $\rightarrow$ E $\rightarrow$ D $\rightarrow$ A
  Cost is 8

What if edges have weights

Does BFS still work for finding minimum cost paths ?



Shortest path from C to A

- BFS: C$\rightarrow$ A and cost is 9.
- Minimum cost path: C $\rightarrow$ E $\rightarrow$ D $\rightarrow$ A
  Cost is 8

Therefore, BFS does not work anymore

Dijkstra's Algorithm

From now onwards, if we say shortest path in weighted graphs, then we mean that it is minimum cost path in weighted graph, not the shortest length path

Dijkstra's Algorithm

From now onwards, if we say shortest path in weighted graphs, then we mean that it is minimum cost path in weighted graph, not the shortest length path

- Classic algorithm for solving shortest path in weighted graphs (without negative weights)

Dijkstra's Algorithm

From now onwards, if we say shortest path in weighted graphs, then we mean that it is minimum cost path in weighted graph, not the shortest length path

- Classic algorithm for solving shortest path in weighted graphs (without negative weights)
- The algorithm is based on greedy choice.

Basic Idea

- Each vertex stores a cost for path from source (Initially ∞)
- Greedy Choice: always select the current best vertex
- Update costs of all neighbors of selected vertex

Notations

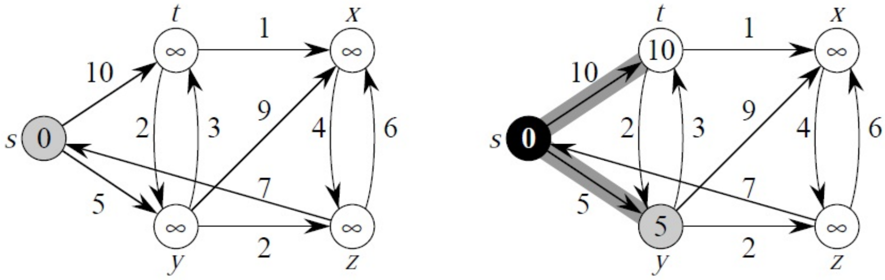
- weighted digraph: A digraph $G = (V, E, w)$, where $w : E \to R$
- weight of a path p: If $p =< v_0, v_1, \ldots, v_k >$ is a path, then $w(p) = \sum_{i=1}^{k} w(v_{i-1} v_i)$
- $\delta(u, v)$: shortest path weight of the paths from u to v and is defined as

  $$\delta(u, v) = \begin{cases} \min\{w(p) : p \text{ is a path from} u \text{ to } v\}, & \text{if there is a path from } u \text{ to } v; \\ \infty, & \text{Otherwise.} \end{cases}$$
- $d[v]$: an upper bound on the weight of a shortest path from source s to v.

Pseudocode of Dijkstra's algorithm

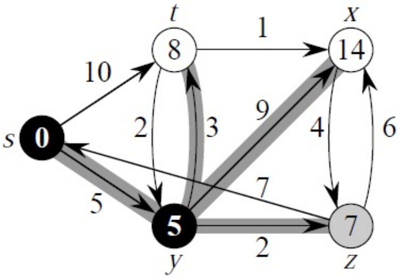
for *each* $v \in V$ **do**
    $d[v] = \infty$;
    $\pi[v] = \mathrm{NIL}$;
end
$d[s] = 0$;
$S = \emptyset$;
$Q = V$;
while $Q \neq \emptyset$ **do**
    $u = \textsc{Extract-Min}(Q)$;
    $S = S \cup \{u\}$;
    for *each* $v \in Adj[u]$ **do**
        if $(d[v] > d[u] + w(u, v))$ **then**
            $d[v] = d[u] + w(u, v)$;
            $\pi[v] = u$;
        end
    end
end

Example

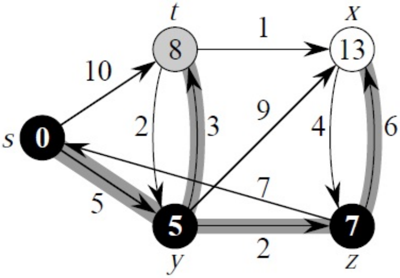


(a) (b)

Example

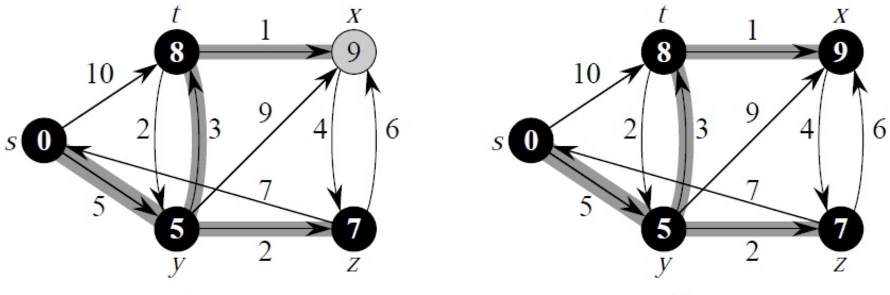


(c)



(d)

Example



(e) (f)

Analysis

```
for each v ∈ V do
    d[v] = ∞;
    π[v] = NIL;
end
d[s] = 0;
S = ∅;
Q = V;
while Q ≠ ∅ do
    u = EXTRACT-MIN(Q);
    S = S ∪ {u};
    for each v ∈ Adj[u] do
        if (d[v] >
          d[u] + w(u, v))
          then
            d[v] =
              d[u] +
              w(u, v);
            π[v] = u;
        end
    end
end
```

Analysis

```
1  for each v ∈ V do
2      d[v] = ∞;
3      π[v] = NIL;
4  end
5  d[s] = 0;
6  S = ∅;
7  Q = V;
8  while Q ≠ ∅ do
9      u = EXTRACT-MIN(Q);
10     S = S ∪ {u};
11     for each v ∈ Adj[u] do
12         if (d[v] > d[u] + w(u, v)) then
13             d[v] = d[u] + w(u, v);
14             π[v] = u;
15         end
16     end
17 end
```

- Lines 1-4 takes $O(|V|)$ time

Analysis

```
1  for each v ∈ V do
2      d[v] = ∞;
3      π[v] = NIL;
4  end
5  d[s] = 0;
6  S = ∅;
7  Q = V;
8  while Q ≠ ∅ do
9      u = EXTRACT-MIN(Q);
10     S = S ∪ {u};
11     for each v ∈ Adj[u] do
12         if (d[v] > d[u] + w(u, v)) then
13             d[v] = d[u] + w(u, v);
14             π[v] = u;
15         end
16     end
17 end
```

- Lines 1-4 takes $O(|V|)$ time
- While loop runs for $|V|$ times

Analysis

1 **for** *each* $v \in V$ **do**
2     $d[v] = \infty$;
3     $\pi[v] =$ NIL;
4 **end**
5 $d[s] = 0$;
6 $S = \emptyset$;
7 $Q = V$;
8 **while** $Q \neq \emptyset$ **do**
9     $u =$ EXTRACT-MIN(Q);
10     $S = S \cup \{u\}$;
11   **for** *each* $v \in Adj[u]$ **do**
12   **if** $(d[v] > d[u] + w(u, v))$ **then**
13             $d[v] = d[u] + w(u, v)$;
14             $\pi[v] = u$;
15   **end**
16   **end**
17 **end**

- Lines 1-4 takes $O(|V|)$ time
- While loop runs for $|V|$ times
  - For loop runs for $d(u)$ times

Analysis

```
1  for each v ∈ V do
2      d[v] = ∞;
3      π[v] = NIL;
4  end
5  d[s] = 0;
6  S = ∅;
7  Q = V;
8  while Q ≠ ∅ do
9      u = EXTRACT-MIN(Q);
10     S = S ∪ {u};
11     for each v ∈ Adj[u] do
12         if (d[v] > d[u] + w(u, v)) then
13             d[v] = d[u] + w(u, v);
14             π[v] = u;
15         end
16     end
17 end
```

- Lines 1-4 takes $O(|V|)$ time
- While loop runs for $|V|$ times
  - For loop runs for $d(u)$ times
  - time for decrease key i.e., to update new value of $d[v]$

Analysis

1 **for** *each* $v \in V$ **do**
2      $d[v] = \infty$;
3      $\pi[v] =$ NIL;
4 **end**
5 $d[s] = 0$;
6 $S = \emptyset$;
7 $Q = V$;
8 **while** $Q \neq \emptyset$ **do**
9      $u = \textsc{Extract-Min}(Q)$;
10      $S = S \cup \{u\}$;
11   **for** *each* $v \in Adj[u]$ **do**
12   **if** $(d[v] > d[u] + w(u, v))$ **then**
13              $d[v] = d[u] + w(u, v)$;
14              $\pi[v] = u$;
15   **end**
16   **end**
17 **end**

- Lines 1-4 takes $O(|V|)$ time
- While loop runs for $|V|$ times
  - For loop runs for $d(u)$ times
  - time for decrease key i.e., to update new value of $d[v]$

Therefore, total time=

$\Theta(|V| \cdot T_{\textsc{Extract-Min}} + |E| \cdot T_{\textsc{extract-key}})$

Analysis

```
1  for each v ∈ V do
2      d[v] = ∞;
3      π[v] = NIL;
4  end
5  d[s] = 0;
6  S = ∅;
7  Q = V;
8  while Q ≠ ∅ do
9      u = EXTRACT-MIN(Q);
10     S = S ∪ {u};
11     for each v ∈ Adj[u] do
12         if (d[v] > d[u] + w(u, v)) then
13             d[v] = d[u] + w(u, v);
14             π[v] = u;
15         end
16     end
17 end
```

- Lines 1-4 takes $O(|V|)$ time
- While loop runs for $|V|$ times
  - For loop runs for $d(u)$ times
  - time for decrease key i.e., to update new value of $d[v]$

Therefore, total time=

$$\Theta(|V| \cdot T_{\text{EXTRACT-MIN}} + |E| \cdot T_{\text{EXTRACT-KEY}})$$

Similar to Prim's algorithm.........

Analysis

So total time $= \Theta(|V|) \cdot T_{\textsc{Extract-Min}} + \Theta(|E|) \cdot T_{\textsc{decrease-Key}}$

Analysis

So total time $= \Theta(|V|) \cdot T_{\text{Extract-Min}} + \Theta(|E|) \cdot T_{\text{Decrease-Key}}$

- Array: .

Analysis

So total time $= \Theta(|V|) \cdot T_{\textsc{Extract-Min}} + \Theta(|E|) \cdot T_{\textsc{Decrease-Key}}$

- Array: $T_{\textsc{Extract-Min}} = \Theta(|V|)$ and $T_{\textsc{Decrease-Key}} = \Theta(1)$.

Analysis

So total time $= \Theta(|V|) \cdot T_{\textsc{Extract-Min}} + \Theta(|E|) \cdot T_{\textsc{decrease-Key}}$

- Array: $T_{\textsc{Extract-Min}} = \Theta(|V|)$ and $T_{\textsc{Decrease-Key}} = \Theta(1)$. So total time $=\Theta(|V|^2)$

Analysis

So total time $= \Theta(|V|) \cdot T_{\text{EXTRACT-MIN}} + \Theta(|E|) \cdot T_{\text{DECREASE-KEY}}$

- Array: $T_{\text{EXTRACT-MIN}} = \Theta(|V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(1)$.
  So total time $=\Theta(|V|^2)$
- Heap: .

Analysis

So total time $= \Theta(|V|) \cdot T_{\textsc{Extract-Min}} + \Theta(|E|) \cdot T_{\textsc{decrease-Key}}$

- Array: $T_{\textsc{Extract-Min}} = \Theta(|V|)$ and $T_{\textsc{Decrease-Key}} = \Theta(1)$. So total time $=\Theta(|V|^2)$
- Heap: $T_{\textsc{Extract-Min}} = \Theta(\log|V|)$ and $T_{\textsc{Decrease-Key}} = \Theta(\log|V|)$.

Analysis

So total time = $\Theta(|V|) \cdot T_{\text{EXTRACT-MIN}} + \Theta(|E|) \cdot T_{\text{DECREASE-KEY}}$

- Array: $T_{\text{EXTRACT-MIN}} = \Theta(|V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(1)$.
  So total time $=\Theta(|V|^2)$
- Heap: $T_{\text{EXTRACT-MIN}} = \Theta(\log|V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(\log|V|)$.
  So total time $=\Theta(|E| \cdot \log|V|)$

Analysis

So total time $= \Theta(|V|) \cdot T_{\text{EXTRACT-MIN}} + \Theta(|E|) \cdot T_{\text{DECREASE-KEY}}$

- Array: $T_{\text{EXTRACT-MIN}} = \Theta(|V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(1)$.
  So total time $=\Theta(|V|^2)$
- Heap: $T_{\text{EXTRACT-MIN}} = \Theta(\log |V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(\log |V|)$.
  So total time $=\Theta(|E| \cdot \log |V|)$
- Fibonnaci Heap:

Analysis

So total time $= \Theta(|V|) \cdot T_{\text{EXTRACT-MIN}} + \Theta(|E|) \cdot T_{\text{DECREASE-KEY}}$

- Array: $T_{\text{EXTRACT-MIN}} = \Theta(|V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(1)$.
  So total time $=\Theta(|V|^2)$
- Heap: $T_{\text{EXTRACT-MIN}} = \Theta(\log |V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(\log |V|)$.
  So total time $=\Theta(|E| \cdot \log |V|)$
- Fibonnaci Heap: $T_{\text{EXTRACT-MIN}} = \Theta(\log |V|)$ (Amortized) and $T_{\text{DECREASE-KEY}} = \Theta(1)$ (Amortized)

Analysis

So total time $= \Theta(|V|) \cdot T_{\text{EXTRACT-MIN}} + \Theta(|E|) \cdot T_{\text{DECREASE-KEY}}$

- Array: $T_{\text{EXTRACT-MIN}} = \Theta(|V|)$ and $T_{\text{DECREASE-KEY}} = \Theta(1)$.
  So total time $=\Theta(|V|^2)$
- Heap: $T_{\text{EXTRACT-MIN}} = \Theta(\log|V|)$ and
  $T_{\text{DECREASE-KEY}} = \Theta(\log|V|)$.
  So total time $=\Theta(|E| \cdot \log|V|)$
- Fibonnaci Heap: $T_{\text{EXTRACT-MIN}} = \Theta(\log|V|)$ (Amortized)
  and $T_{\text{DECREASE-KEY}} = \Theta(1)$ (Amortized)
  So total time $=\Theta(|E| + |V| \cdot \log|V|)$

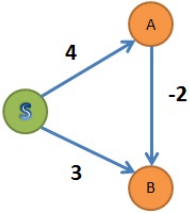
Drawback of Dijkstra's algorithm

Drawback of Dijkstra's algorithm

- Presence of negative weights on edges

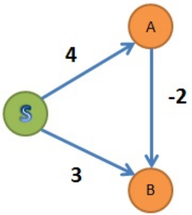
Drawback of Dijkstra's algorithm

- Presence of negative weights on edges



Drawback of Dijkstra's algorithm

- Presence of negative weights on edges



Whether a DP will work ?