Design and Analysis of Algorithms CS218M

Greedy Algorithms (2)

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

The Greedy Paradigm

- Build the solution by selecting elements (or making choices) one by one.
- A simple rule allows choice of element at each stage. Local optimality.
- Greedy choice property: The current selection cannot be removed (no backtracking/exploring alternative choices).
- The final solution must be optimal.

Sequence of locally optimalchoices gives globally optimal solution.

Examples: Picking 10 coins, Finding shortest path, Minimum Spanning Tree.

Minimum Spanning Tree (MST)

Given connected and weighted undirected graph $G = (V, E, w)$ with nodes V, Edges $E \subseteq V \times V$ and $w : E \rightarrow \Re$, find $A \subseteq E$ s.t.

Minimum Spanning Tree (MST)

Given connected and weighted undirected graph $G = (V, E, w)$ with nodes V, Edges $E \subseteq V \times V$ and $w : E \to \Re$, find $A \subseteq E$ s.t.

- A is a tree spanning V.

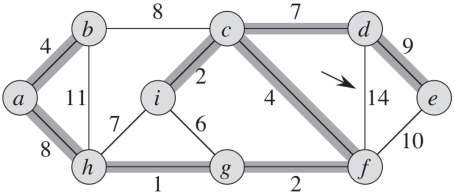
Minimum Spanning Tree (MST)

Given connected and weighted undirected graph $G = (V, E, w)$ with nodes V, Edges $E \subseteq V \times V$ and $w : E \to \Re$, find $A \subseteq E$ s.t.

- A is a tree spanning V.
- Let $wt(A) = \Sigma_{e \in A} w(e)$. Then for all $B \subseteq E$, if B is a spanning tree then $wt(B) \geq wt(A)$.

Given connected and weighted undirected graph $G = (V, E, w)$ with nodes V, Edges $E \subseteq V \times V$ and $w : E \rightarrow \Re$, find $A \subseteq E$ s.t.

- A is a tree spanning V.
- Let $wt(A) = \Sigma_{e \in A} w(e)$. Then for all $B \subseteq E$, if B is a spanning tree then $wt(B) \geq wt(A)$.



Greedy Paradigm + Data Structures

Grow A adding one edge at a time.

Grow A adding one edge at a time.

Kruskal

Add lowest weight edge which does not form a cycle to current A.

Grow A adding one edge at a time.

Kruskal

Add lowest weight edge which does not form a cycle to current A.

Prim

Extend current set of edges A having vertices U_A with a minimum weight edge going out of U_A.

Generic MST Algorithm

- Grow A one edge at a time.
- Invariant: Current set of edges A is a subset of some MST.
- An edge which can be added to A maintaining the invariant is called a safe edge.

GENERIC-MST(G, w)

```
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

Terminology

Given connected, undirected, weighted graph $G = (V, E, w)$, and $A \subseteq E$, define

- Pair $(S, V - S)$ is a cut.

Terminology

Given connected, undirected, weighted graph $G = (V, E, w)$, and $A \subseteq E$, define

- Pair $(S, V - S)$ is a cut.
- Edge (u, v) crosses the cut $(S, V - S)$ if $u \in S$ and $v \notin S$ or vice verse.

Terminology

Given connected, undirected, weighted graph $G = (V, E, w)$, and $A \subseteq E$, define

- Pair $(S, V - S)$ is a cut.
- Edge (u, v) crosses the cut $(S, V - S)$ if $u \in S$ and $v \notin S$ or vice verse.
- Cut $(S, V - S)$ respects A if no edge of A is a crossing edge.

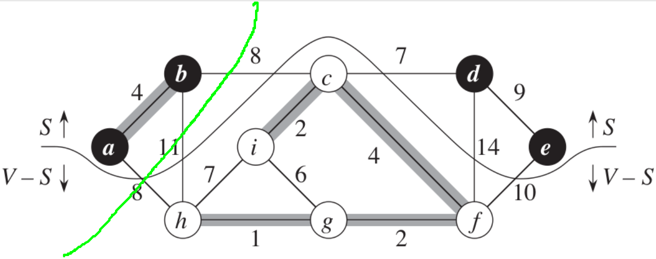
Terminology

Given connected, undirected, weighted graph $G = (V, E, w)$, and $A \subseteq E$, define

- Pair $(S, V - S)$ is a cut.
- Edge (u, v) crosses the cut $(S, V - S)$ if $u \in S$ and $v \notin S$ or vice verse.
- Cut $(S, V - S)$ respects A if no edge of A is a crossing edge.
- An edge (u, v) is a light edge if it is of minimum weight amongst all edges crossing the cut.

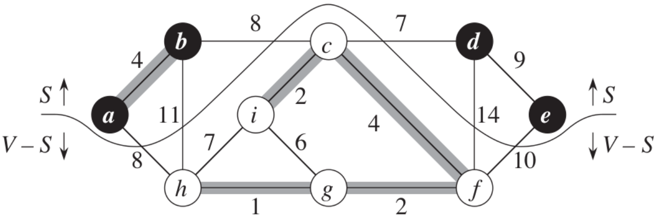
Theorem

Let A subset of some MST. Let cut $(S, V-S)$ respect A and let (u, v) be a light edge. Then, (u, v) is a safe edge.



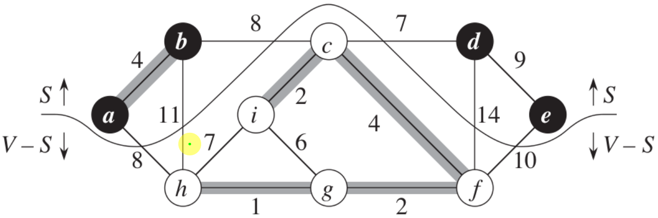
Theorem

Let A subset of some MST. Let cut $(S, V-S)$ respect A and let (u, v) be a light edge. Then, (u, v) is a safe edge.



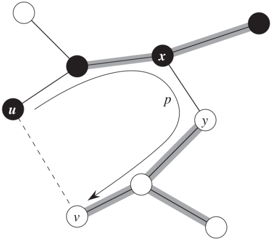
Theorem

Let A subset of some MST. Let cut $(S, V - S)$ respect A and let (u, v) be a light edge. Then, (u, v) is a safe edge.



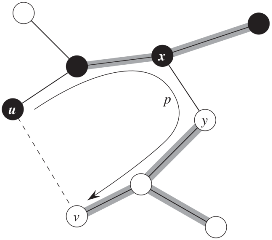
Proof

- A gray edges, MST T. Let $(u, v) \notin T$ be a light-edge.



Proof

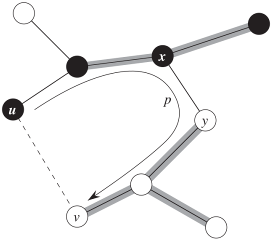
- A gray edges, MST T. Let $(u, v) \notin T$ be a light-edge.



- Let $(x, y) \in T$ be crossing edge. (Must exist).

Proof

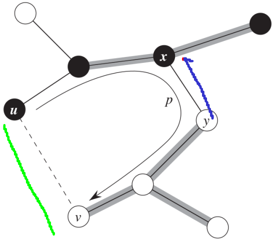
- A gray edges, MST T. Let $(u, v) \notin T$ be a light-edge.



- Let $(x, y) \in T$ be crossing edge. (Must exist).
- Hence $w(u, v) \leq w(x, y)$. (why?)

Proof

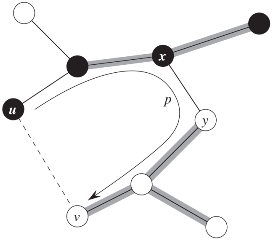
- A gray edges, MST T. Let $(u, v) \notin T$ be a light-edge.



- Let $(x, y) \in T$ be crossing edge. (Must exist).
- Hence $w(u, v) \leq w(x, y)$. (why?)
- Let $T' = {\color{blue}T - \{(x, y)\}} \cup {\color{red}\{(u, v)\}}$. Then T' is a spanning tree.

Proof

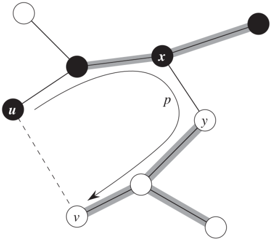
- A gray edges, MST T. Let $(u, v) \notin T$ be a light-edge.



- Let $(x, y) \in T$ be crossing edge. (Must exist).
- Hence $w(u, v) \leq w(x, y)$. (why?)
- Let $T' = T - \{(x, y)\} \cup \{(u, v)\}$. Then T' is a spanning tree.
- $wt(T') = wt(T) - w(x, y) + w(u, v)$.
  Hence, $wt(T') \leq wt(T)$.

Proof

- A gray edges, MST T. Let $(u, v) \notin T$ be a light-edge.



- Let $(x, y) \in T$ be crossing edge. (Must exist).
- Hence $w(u, v) \leq w(x, y)$. (why?)
- Let $T' = T - \{(x, y)\} \cup \{(u, v)\}$. Then T' is a spanning tree.
- $wt(T') = wt(T) - w(x, y) + w(u, v)$.
  Hence, $wt(T') \leq wt(T)$.
- Hence, T' is MST containing (u, v).

Kruskal Algorithm and Correctness

At each iteration.

- Add edges from E to A in order of increasing weights.

Kruskal Algorithm and Correctness

At each iteration.

- Add edges from E to A in order of increasing weights.
- A gives rise to a set of disjoint trees.

At each iteration.

- Add edges from E to A in order of increasing weights.
- A gives rise to a set of disjoint trees.
- Kruskal iteration extends A by minimum weight edge (u, v) which does not form a cycle. Thus, it connects two trees T_1 and T_2 (and merges these).

Kruskal Algorithm and Correctness

At each iteration.

- Add edges from E to A in order of increasing weights.
- A gives rise to a set of disjoint trees.
- Kruskal iteration extends A by minimum weight edge (u, v) which does not form a cycle. Thus, it connects two trees T_1 and T_2 (and merges these).
- Choose cut respecting A as $(T_1, S - T_1)$. Clearly, (u, v) is safe edge. Theorem applies.

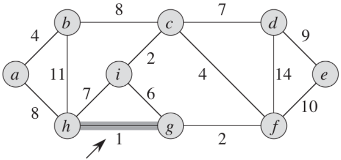
Kruskal Algorithm and Correctness

At each iteration.

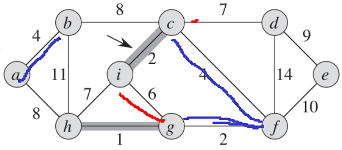
- Add edges from E to A in order of increasing weights.
- A gives rise to a set of disjoint trees.
- Kruskal iteration extends A by minimum weight edge (u, v) which does not form a cycle. Thus, it connects two trees T_1 and T_2 (and merges these).
- Choose cut respecting A as $(T_1, S - T_1)$. Clearly, (u, v) is safe edge. Theorem applies.
- Adding it using UNION gives A as set of trees represented as disjoint sets.

Kruskal Algorithm: Example

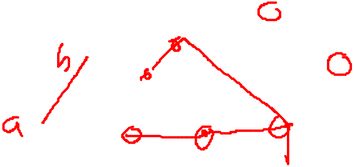
$A = \phi$



(b)



$A =$



Maintain $\mathcal{S} = (S_1, \ldots, S_k)$ with $u_i \in S_i$ as unique representative.

Maintain $\mathcal{S} = (S_1, \ldots, S_k)$ with $u_i \in S_i$ as unique representative.

- MAKESET(u)

Maintain $\mathcal{S} = (S_1, \ldots, S_k)$ with $u_i \in S_i$ as unique representative.

- MAKESET(u)
- FINDSET(u)

Disjoint Set Data Structure

Maintain $\mathcal{S} = (S_1, \ldots, S_k)$ with $u_i \in S_i$ as unique representative.

- MAKESET(u)
- FINDSET(u)
- UNION(u,v)

Disjoint Set Data Structure

Maintain $\mathcal{S} = (S_1, \ldots, S_k)$ with $u_i \in S_i$ as unique representative.

- MAKESET(u)
- FINDSET(u)
- UNION(u,v)
- Implemeted using union by rank and path compression (CLRS 21.3, 21.4). For m operations over n element set, $O(m \cdot \alpha(n))$ where $\alpha(n)$ is very slowly growing (almost constant!).

Kruskal Algorithm for MST

MST-KRUSKAL(G, w)

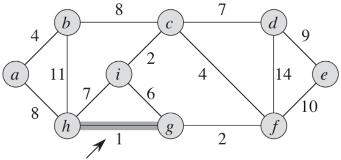
$A = \emptyset$
for each vertex $v \in G.V$ $\quad O(V)$
  MAKE-SET(v)
sort the edges of $G.E$ into nondecreasing order by weight w $\quad O(E \lg E)$
for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
  if FIND-SET$(u) \neq$ FIND-SET(v)
    $A = A \cup \{(u, v)\}$
    UNION(u, v) $\quad O(E)$
return A

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

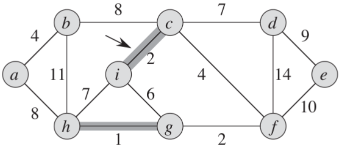
Running Time

$E \cdot lg(E)$ for sorting edges. Also, $O(V)$ of MAKE-SET and $O(E)$ of FIND-SET+UNION operations. Hence, $E \cdot lg(E) + \underline{(E + V)}\alpha(V)$. Simplifies to $O(E \cdot lg(E))$. $O(E \lg(V))$

Kruskal Algorithm: Example



(b)



Prim Algorithm for MST

- Maintain A as a single tree with set of vertices U_A. Let $Q = S - U_A$.

Prim Algorithm for MST

- Maintain A as a single tree with set of vertices U_A. Let $Q = S - U_A$.
- In each iteration, we choose edge e with minimum weight amongst $\{(u, v) \mid u \in U_A \wedge v \notin U_A\}$. Clearly, this is safe edge.

Prim Algorithm for MST

- Maintain A as a single tree with set of vertices U_A. Let $Q = S - U_A$.
- In each iteration, we choose edge e with minimum weight amongst $\{(u, v) \mid u \in U_A \wedge v \notin U_A\}$. Clearly, this is safe edge.
- For each vertex $v \in Q$, priority $v.key$ is weight of minimum weight edge between (any vertex in) A and v. If no such edge $key = \infty$.

Prim Algorithm for MST

- Maintain A as a single tree with set of vertices U_A. Let $Q = S - U_A$.
- In each iteration, we choose edge e with minimum weight amongst $\{(u, v) \mid u \in U_A \wedge v \notin U_A\}$. Clearly, this is safe edge.
- For each vertex $v \in Q$, priority $v.key$ is weight of minimum weight edge between (any vertex in) A and v. If no such edge $key = \infty$.
- Maintain Q as a priority queue using the heap data structure. Choose v by EXTRACT_MIN(Q).

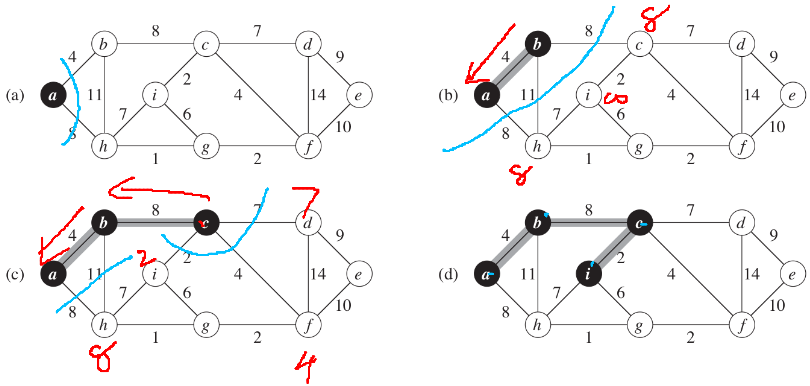
Prim Algorithm for MST

- Maintain A as a single tree with set of vertices U_A. Let $Q = S - U_A$.
- In each iteration, we choose edge e with minimum weight amongst $\{(u, v) \mid u \in U_A \land v \notin U_A\}$. Clearly, this is safe edge.
- For each vertex $v \in Q$, priority $v.key$ is weight of minimum weight edge between (any vertex in) A and v. If no such edge $key = \infty$.
- Maintain Q as a priority queue using the heap data structure. Choose v by EXTRACT_MIN(Q).
- After adding v, update key of all vertices adjecent to v which are in Q.

Prim Algorithm

$S = \{a\}$

$S = \{a, b\}$



Prim Algorithm

A u_A $Q = V - u_A$

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

$O(V)$

$O(1)$

$O(V \log(V))$

$O(E)$

$O(E \log(V))$

Complexity of Prim Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Complexity of Prim Algorithm (2)

- (loop at line 1) executes $O(V)$ iterations.

Complexity of Prim Algorithm (2)

- (loop at line 1) executes $O(V)$ iterations.
- (line 5) $O(V)$ for forming MIN-priority queue of V.

Complexity of Prim Algorithm (2)

- (loop at line 1) executes $O(V)$ iterations.
- (line 5) $O(V)$ for forming MIN-priority queue of V.
- (loop at line 6) iterates V times and takes $O(lg(V)$ for each EXTRACT-MIN. Hence $V \cdot lg(V)$.

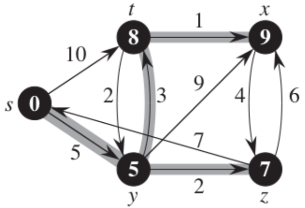
Complexity of Prim Algorithm (2)

- (loop at line 1) executes $O(V)$ iterations.
- (line 5) $O(V)$ for forming MIN-priority queue of V.
- (loop at line 6) iterates V times and takes $O(lg(V)$ for each EXTRACT-MIN. Hence $V \cdot lg(V)$.
- (loop at line 8) iterates $2 \cdot E$ times. Each iteration takes $O(lg(V))$ for change key. Hence $O(E \cdot lg(V))$.

Complexity of Prim Algorithm (2)

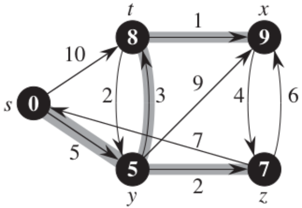
- (loop at line 1) executes $O(V)$ iterations.
- (line 5) $O(V)$ for forming MIN-priority queue of V.
- (loop at line 6) iterates V times and takes $O(lg(V)$ for each EXTRACT-MIN. Hence $V \cdot lg(V)$.
- (loop at line 8) iterates $2 \cdot E$ times. Each iteration takes $O(lg(V))$ for change key. Hence $O(E \cdot lg(V))$.
- Hence, overall complexity $O(E \cdot lg(V))$.

Given directed and weighted graph $G = (V, E, l)$ with nodes V, Edges $E \subseteq V \times V$ and $l : E \rightarrow \Re$, and start node $s \in V$, for every node t find smallest weight path $v_0, v_1, \ldots, v_k$ where $v_0 = s$ and $v_k = t$ and its weight $d(t)$.



Given directed and weighted graph $G = (V, E, l)$ with nodes V, Edges $E \subseteq V \times V$ and $l : E \to \Re$, and start node $s \in V$, for every node t find smallest weight path $v_0, v_1, \ldots, v_k$ where $v_0 = s$ and $v_k = t$ and its weight $d(t)$.

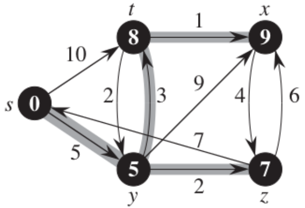
- Weight of a path $v_0, v_1, \ldots, v_k$ is $\Sigma_{i=0}^{k-1} \; l(v_i, v_{i+1})$.



Single Source Shortest Paths

Given directed and weighted graph $G = (V, E, l)$ with nodes V, Edges $E \subseteq V \times V$ and $l : E \rightarrow \Re$, and start node $s \in V$, for every node t find smallest weight path $v_0, v_1, \ldots, v_k$ where $v_0 = s$ and $v_k = t$ and its weight $d(t)$.

- Weight of a path $v_0, v_1, \ldots, v_k$ is $\Sigma_{i=0}^{k-1} \; l(v_i, v_{i+1})$.
- Shortest Path Tree as node attribute π: Let $w.\pi = v$ give the predecessor of w on the shortest path from s to w as v.



Dijkstra's SSP Algorithm

- We assume that $l_e \geq 0$ for all $e \in E$. No negative edge weights.
- We maintain $S \subseteq V$ for which shortest paths are found.

Dijkstra's SSP Algorithm

- We assume that $l_e \geq 0$ for all $e \in E$. No negative edge weights.
- We maintain $S \subseteq V$ for which shortest paths are found.

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

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

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

While $S \neq V$

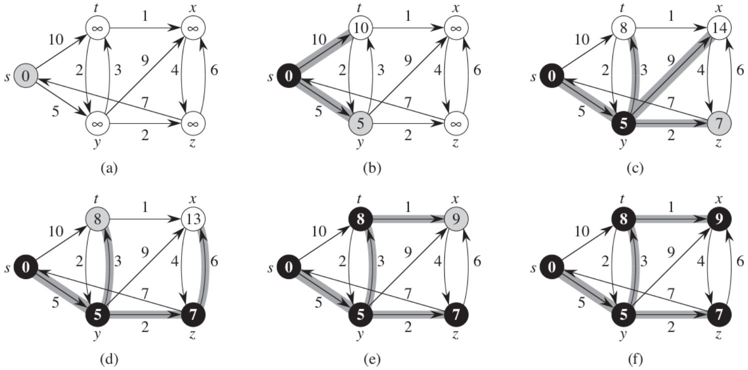
    Select a node $v \notin S$ with at least one edge from S for which

        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

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

EndWhile

Dijkstra's SSP Algorithm: Example



Invariant

- For all $u \in S$, the $d(u)$ gives the length of the shortest path from s to u andπ gives the shortest path to u.

Invariant

- For all $u \in S$, the $d(u)$ gives the length of the shortest path from s to u and π gives the shortest path to u.
- For all $v \notin S$ define $d'(v) \; = \; min_{(u,v):u \in S} \;\; d(u) + l(u,v)$.

Invariant

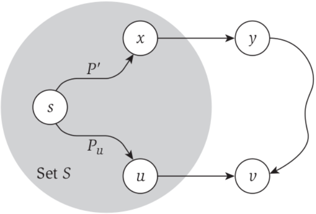
- For all $u \in S$, the $d(u)$ gives the length of the shortest path from s to u andπ gives the shortest path to u.
- For all $v \notin S$ define $d'(v) \; = \; min_{(u,v):u \in S} \;\; d(u) + l(u,v)$.

Maintaining Invariant: Greedy Choice

For extending S, choose $v \notin S$ with minimum $d'(v)$ and set $d(v) = d'(v)$.

Correctness of Greedy Choice

- If (u, v) is edge with $u \in S$ and $v \notin S$ giving minimum $d(u) + l(u, v)$then $d(v) = d(u) + l(u, v)$.



Complexity of Dijkstra SSP Algorithm

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
      d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Complexity of Dijkstra SSP Algorithm

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

- while loop iterates V times.

Complexity of Dijkstra SSP Algorithm

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
      d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

- while loop iterates V times.
- In each iteration, we scan all E edges to find the minimum $d'(v)$.

Complexity of Dijkstra SSP Algorithm

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile

- while loop iterates V times.
- In each iteration, we scan all E edges to find the minimum $d'(v)$.
- Total time $O(V \cdot E)$.

Complexity of Dijkstra SSP Algorithm

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

- while loop iterates V times.
- In each iteration, we scan all E edges to find the minimum $d'(v)$.
- Total time $O(V \cdot E)$.
- If we compute and store $d'(v)$ in an array and update it only for required edges, complexity becomes $O(V^2 + E)$ which simplifies to $O(V^2)$.

Priority Queue based SSP Algorithm

Priority Queue based SSP Algorithm

- We can improve the performance by storing $V - S$ nodes in MIN-PRIORITY QUEUE by the key $d'(v)$.

Priority Queue based SSP Algorithm

- We can improve the performance by storing $V - S$ nodes in MIN-PRIORITY QUEUE by the key $d'(v)$.

Priority Queue based SSP Algorithm

- We can improve the performance by storing $V - S$ nodes in MIN-PRIORITY QUEUE by the key $d'(v)$.

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Complexity

Complexity

- Initializing priority queue $O(V)$.

Complexity

- Initializing priority queue $O(V)$.
- Loop (line 4) iterates $O(V)$ times. Hence EXTRACT-MIN executes $O(V)$ times giving $O(V \cdot lg(V))$.

Complexity

- Initializing priority queue $O(V)$.
- Loop (line 4) iterates $O(V)$ times. Hence EXTRACT-MIN executes $O(V)$ times giving $O(V \cdot lg(V))$.
- For loop (line 7) iterates $O(E)$ with 1 CHANGE-KEY operation each. Gives $O(E \cdot lg(V))$.

Complexity

- Initializing priority queue $O(V)$.
- Loop (line 4) iterates $O(V)$ times. Hence EXTRACT-MIN executes $O(V)$ times giving $O(V \cdot lg(V))$.
- For loop (line 7) iterates $O(E)$ with 1 CHANGE-KEY operation each. Gives $O(E \cdot lg(V))$.
- Overall Complexity is $O(E \cdot lg(V))$. Good for sparse graphs.