Minimum Spanning Tree

- **Kruskal’s Algorithm**
  - Finds a minimum spanning tree for a connected weighted graph
  - Create a set of trees, where each vertex in the graph is a separate tree
  - Create set $S$ containing all edges in the graph
  - While $S$ not empty
    - Remove edge with minimum weight from $S$
    - If that edge connects two different trees, then add it to the forest, combining two trees into a single tree
    - Otherwise discard that edge

**Step 1**
Minimum Spanning Tree

- Prim’s Algorithm
  - Finds a subset of the edges (that form a tree) including every vertex and the total weight of all the edges in tree is minimized
    - Choose starting vertex
    - Create the Fringe Set
    - Loop until the MST contains all the vertices in the graph
      - Remove edge with minimum weight from Fringe Set
      - Add the edge to MST
      - Update the Fringe Set

Prim – Initialization

- Pick any vertex \(x\) as the starting vertex
- Place \(x\) in the Minimum Spanning Tree (MST)
- For each vertex \(y\) in the graph that is adjacent to \(x\)
  - Add \(y\) to the Fringe Set
- For each vertex \(y\) in the Fringe Set
  - Set weight of \(y\) to weight of the edge connecting \(y\) to \(x\)
  - Set \(x\) to be parent of \(y\)
Prim – Body

While number of vertices in MST < vertices in the graph

Find vertex $y$ with minimum weight in the Fringe Set
Add vertex and the edge $x,y$ to the MST
Remove $y$ from the Fringe Set
For all vertices $z$ adjacent to $y$ that are not in MST
   If $z$ is not in the Fringe Set
      Add $z$ to the Fringe Set
      Set parent to $y$
      Set weight of $z$ to weight of the edge connecting $z$ to $y$
   Else
      If Weight($y,z$) < Weight($z$) then
         Set parent to $y$
         Set weight of $z$ to weight of the edge connecting $z$ to $y$
Inserting into Ordered Binary Tree

-- insert procedure

procedure Insert (Root : in out Nodeptr;
   Element : in Integer ) is
   New_Node : Nodeptr;
begin
   if Root = null then
      New_Node := new Node;
      New_Node.Element := Element;
      Root := New_Node;
   else
      if Root.Element < Element then
         Insert(Root.Right_Child, Element);
      else
         Insert(Root.Left_Child, Element);
      end if;
   end if;
end Insert;

Inserting into a Binary node

• Insert 10, 11, 9, 7, 8, 12
• Insert 10

• Insert 11

• Insert 9

• Insert 7
Inserting into Ordered Binary Tree

- Insert 8

```
    10
   /  
  9    11
 /     /
7      8
```

- Insert 12

```
    10
   /  
  9    11
 /     /
7      8
    /  
   12
```

Shortest Path Problems

- **Dijkstra’s algorithm**
  - Finds shortest path for a directed and connected graph $G(V,E)$ which has non-negative weights.
  - Applications:
    - Internet routing
    - Road generation within a geographic region
    - ...

Dijkstra’s Algorithm

- **Dijkstra(G,w,s)**
  - $Init\_Source(G,s)$
  - $S := empty\ set$
  - $Q := set\ of\ all\ vertices$

  ```
  while $Q$ is not an empty set loop
  
  $u := Extract\_Min(Q)$
  
  $S := S union \{u\}$
  
  for each vertex $v$ which is a neighbor of $u$ loop
  Relax($u,v,w$)
  ```
Dijkstra’s Algorithm

- **Init_Source(G, s)**
  
  for each vertex \( v \) in \( V[G] \) loop
  
  \( d[v] := \text{infinite} \)
  
  previous\([v]\) := 0
  
  \( d[s] := 0 \)

- \( v = \text{Extract_Min}(Q) \) searches for the vertex \( v \) in the vertex set \( Q \) that has the least \( d[v] \) value. That vertex is removed from the set \( Q \) and then returned.

- **Relax(u, v, w)**

  if \( d[v] > d[u] + w(u, v) \) then
  
  \( d[v] := d[u] + w(u, v) \)
  
  previous\([v]\) := u

Example:

- **V** = \{a, b, c, d, s\}
- **E** = \{(s,c), (c,d), (d,b), (b,d), (c,b), (a,c), (c,a), (a,b), (s,a)\}

- **S** = \{\( \emptyset \)\}
- **Q** = \{s, a, b, c, d\}

\[
\begin{pmatrix}
0 & \infty & \infty & \infty & \infty \\
10 & 0 & \infty & \infty & \infty \\
2 & 1 & 0 & \infty & \infty \\
3 & \infty & \infty & 0 & \infty \\
9 & 4 & 6 & \infty & 0 \\
\end{pmatrix}
\]

Extract_Min(Q) \( \rightarrow \) s
Neighbors of s = a, c

Relax (s,c,5)
Relax (s,a,10)
Dijkstra’s Algorithm

S = \{s, c, d\}
Q = \{a, b\}

Extract Min (Q) \rightarrow d
Neighbors of d = b
Relax (d, b, 6)

prev = \[
\begin{pmatrix}
0 & c \\
8 & c \\
5 & d \\
7 & s \\
\end{pmatrix}
\]

Dijkstra’s Algorithm

S = \{s, c, d, a\}
Q = \{\}

Extract Min (Q) \rightarrow b
Neighbors of b = d
Relax (b, d, 4)

prev = \[
\begin{pmatrix}
0 & c \\
8 & a \\
9 & s \\
4 & c \\
\end{pmatrix}
\]

Dijkstra’s Algorithm

S = \{s, c, d, a\}
Q = \{b\}

Extract Min (Q) \rightarrow a
Neighbors of a = b, c
Relax (a, b, 1)
Relax (a, c, 3)

prev = \[
\begin{pmatrix}
0 & c \\
8 & d \\
5 & s \\
7 & c \\
\end{pmatrix}
\]

Dijkstra’s Algorithm

S = \{s, c, d, a, b\}
Q = \{\}

Extract Min (Q) \rightarrow b
Neighbors of b = d
Relax (b, d, 4)