Greedy Technique

The General Method
Definition:
Greedy technique is a general algorithm design strategy, built on following elements:
• **configurations**: different choices, values to find
• **objective function**: some configurations to be either maximized or minimized.

The method:

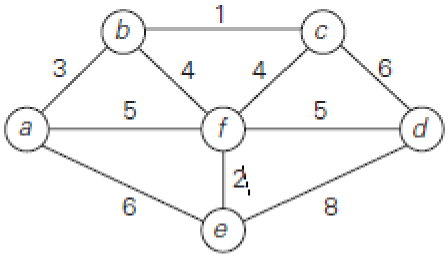
• Applicable to **optimization problems** ONLY
1. • Constructs a solution through a sequence of steps
2. • Each step expands a partially constructed solution so far, until a complete solution to the problem is reached.
3. **On each step, the choice made must be**
   • **Feasible:** it has to satisfy the problem‘s constraints
   • **Locally optimal:** it has to be the best local choice among all feasible choices available on that step
   • **Irrevocable:** Once made, it cannot be changed on subsequent steps of the algorithm.

1. Prim’s Algorithm

DEFINITION A***spanning tree*** of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph. If such a graph has weights assigned to its edges, a ***minimum spanning tree*** is its spanning tree of the smallest weight, where the ***weight*** of a tree is defined as the sum of the weights on all its edges. The ***minimum spanning tree problem*** is the problem of finding a minimum spanning tree for a given weighted connected graph.

ALGORITHM *Prim(G)*
//Prim’s algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E\rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
  find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
  such that v is in V_T and u is in $V - V_T$
  $V_T \leftarrow V_T \cup \{u^*\}$
  $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

Example: Apply the Prim's algorithm to the following graph.



Sol^n

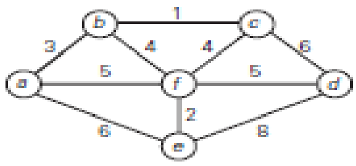
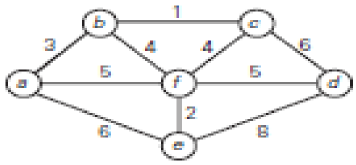
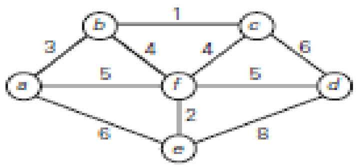
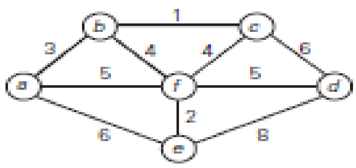
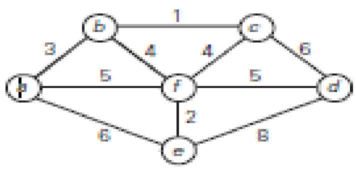
Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	
c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**	
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

FIGURE 9.3 Application of Prim's algorithm. The parenthesized labels of a vertex in the middle column indicate the nearest tree vertex and edge weight; selected vertices and edges are shown in bold.

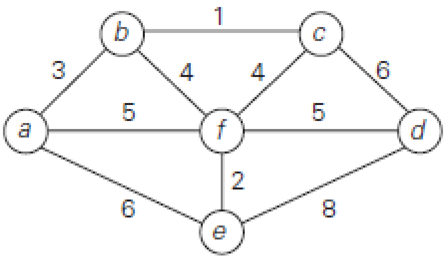
2 Kruskal's Algorithm

Kruskal's algorithm after Joseph Kruskal, who discovered this algorithm when he was a second-year graduate student [Kru56]. Kruskal's algorithm looks at a minimum spanning tree of a weighted connected graph G = _V, E_ as an acyclic subgraph with $|V| - 1$ edges for which the sum of the edge weights is the smallest. (It is not difficult to prove that such a subgraph must be a tree.) Consequently, the algorithm constructs a minimum spanning tree as an expanding sequence of subgraphs that are always acyclic but are not necessarily connected on the intermediate stages of the algorithm.

ALGORITHM *Kruskal*(G)

//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \le \cdots \le w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $\quad ecounter \leftarrow 0$ $\qquad$ //initialize the set of tree edges and its size
$k \leftarrow 0$ $\qquad$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
$\quad k \leftarrow k + 1$
$\quad$ **if** $E_T \cup \{e_{i_k}\}$ is acyclic
$\qquad E_T \leftarrow E_T \cup \{e_{i_k}\}$; $\quad ecounter \leftarrow ecounter + 1$
return E_T

Example: Apply Kruskal's Algorithm for the following graph.



Solution:

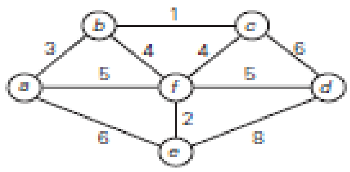
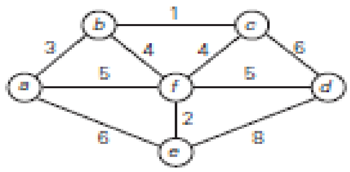
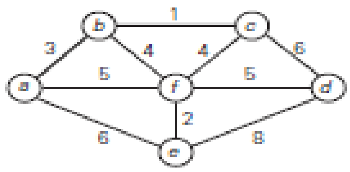
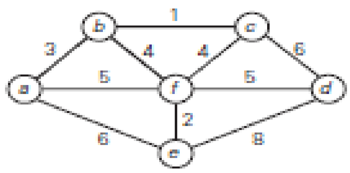
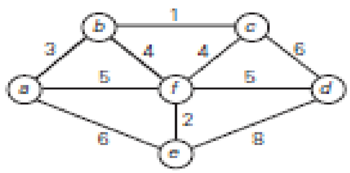
Tree edges	Sorted list of edges	Illustration
	bc 1, ef 2, ab 3, bf 4, cf 4, af 5, df 5, ae 6, cd 6, de 8	
bc 1	bc 1, **ef** 2, ab 3, bf 4, cf 4, af 5, df 5, ae 6, cd 6, de 8	
ef 2	bc 1, ef 2, **ab** 3, bf 4, cf 4, af 5, df 5, ae 6, cd 6, de 8	
ab 3	bc 1, ef 2, ab 3, **bf** 4, cf 4, af 5, df 5, ae 6, cd 6, de 8	
bf 4	bc 1, ef 2, ab 3, bf 4, cf 4, af 5, **df** 5, ae 6, cd 6, de 8	
df 5		

FIGURE 9.5 Application of Kruskal's algorithm. Selected edges are shown in bold.

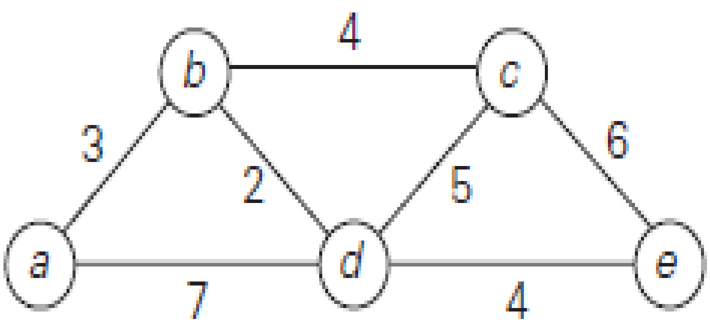
3.Dijkstra’s Algorithm

Dijkstra’s algorithm finds the shortest paths to a graph’s vertices in order of their distance from a given source. ***single-source shortest-paths problem***: for a given vertex called the ***source*** in a weighted connected graph, find shortest paths to all its other vertices.

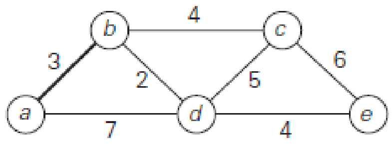
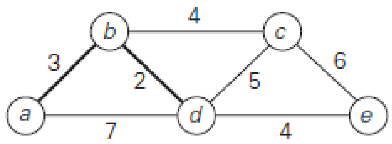
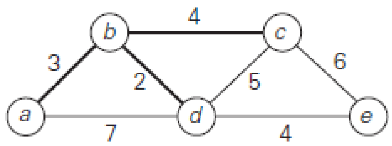
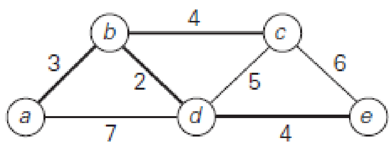
ALGORITHM *Dijkstra*(G, s)

//Dijkstra’s algorithm for single-source shortest paths
//Input: A weighted connected graph $G = \langle V, E \rangle$ with nonnegative weights
// and its vertex s
//Output: The length d_v of a shortest path from s to v
// and its penultimate vertex p_v for every vertex v in V
Initialize(Q) //initialize priority queue to empty
for every vertex v in V
    $d_v \leftarrow \infty$; $p_v \leftarrow$ **null**
    Insert(Q, v, d_v) //initialize vertex priority in the priority queue
$d_s \leftarrow 0$; *Decrease*(Q, s, d_s) //update priority of s with d_s
$V_T \leftarrow \varnothing$
for $i \leftarrow 0$ **to** $|V| - 1$ **do**
    $u^* \leftarrow$ *DeleteMin*(Q) //delete the minimum priority element
    $V_T \leftarrow V_T \cup \{u^*\}$
    for every vertex u in $V - V_T$ that is adjacent to u^* **do**
        if $d_{u^*} + w(u^*, u) < d_u$
            $d_u \leftarrow d_{u^*} + w(u^*, u)$; $p_u \leftarrow u^*$
            Decrease(Q, u, d_u)

Exmaple: Apply Dijkstra Algorithm for the following graph.



Solution:

Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are as follows:

from a to b : $a - b$ of length 3

from a to d : $a - b - d$ of length 5

from a to c : $a - b - c$ of length 7

from a to e : $a - b - d - e$ of length 9