The Design and Analysis of Algorithm

Greedy Algorithm

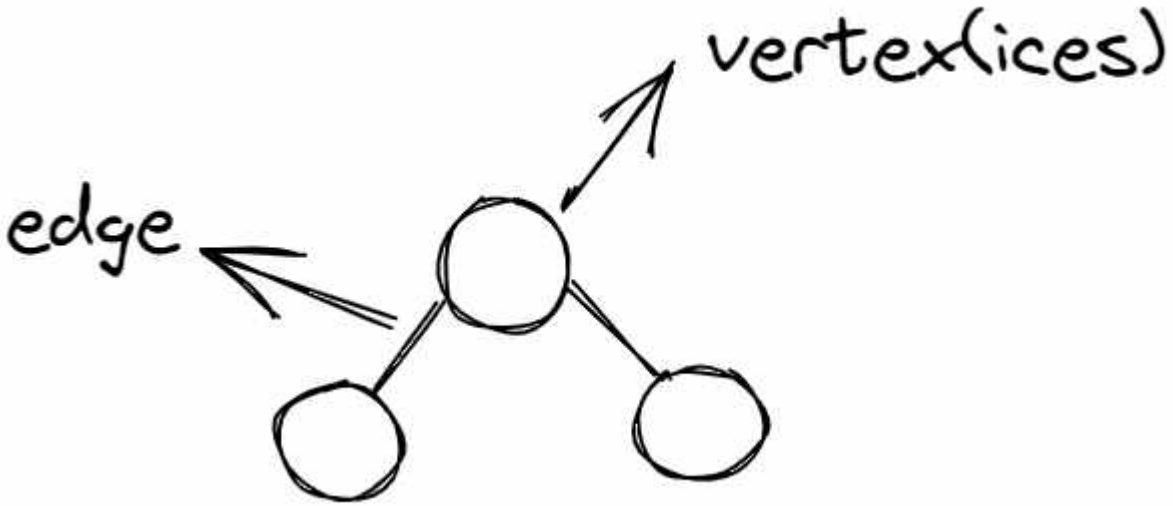
Instructor: Sadullah Karimi
M.Sc. in CSE

Greedy Technique

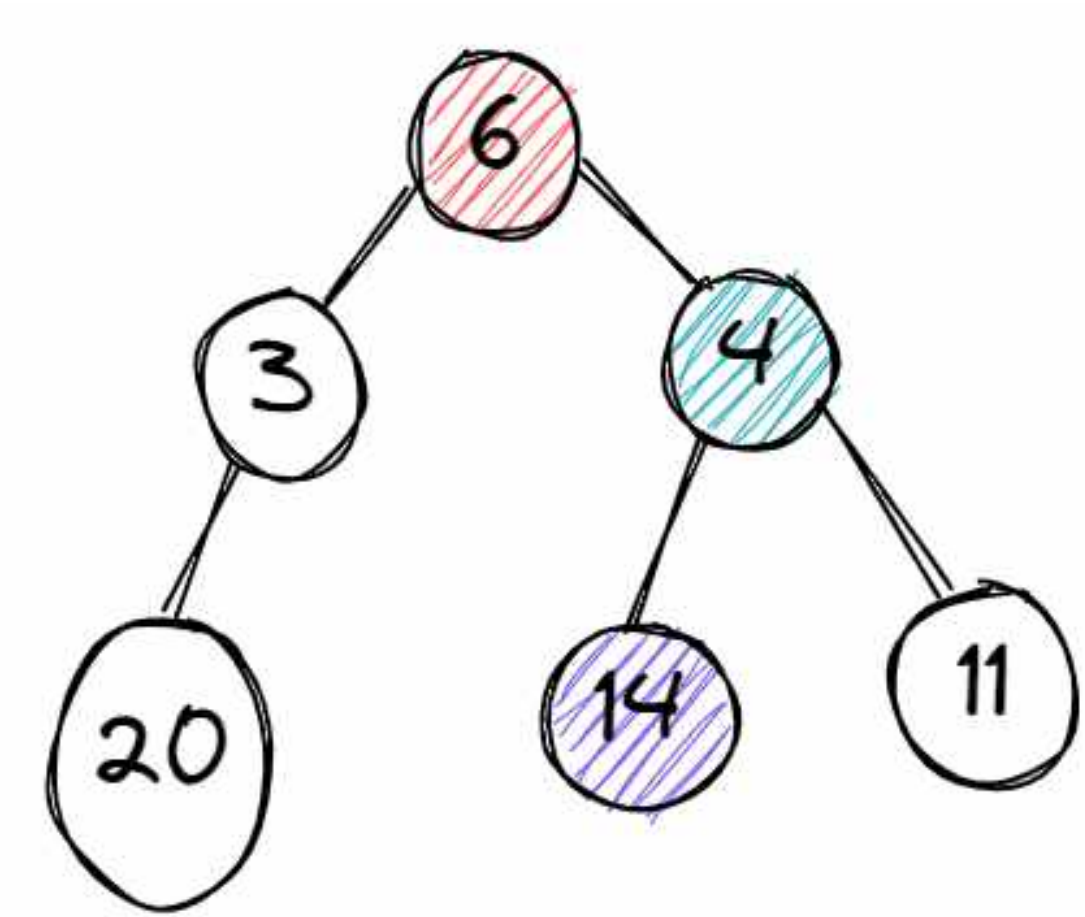
- The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.
- Greedy algorithms are a class of algorithms that make locally optimal choices at each step with the hope of finding a global optimum solution.
- On each step—and this is the central point of this technique—the choice made must be:
  - feasible, i.e., it has to satisfy the problem's constraints
  - locally optimal, i.e., it has to be the best local choice among all feasible choices available on that step
  - irrevocable, i.e., once made, it cannot be changed on subsequent steps of the algorithm

Greedy

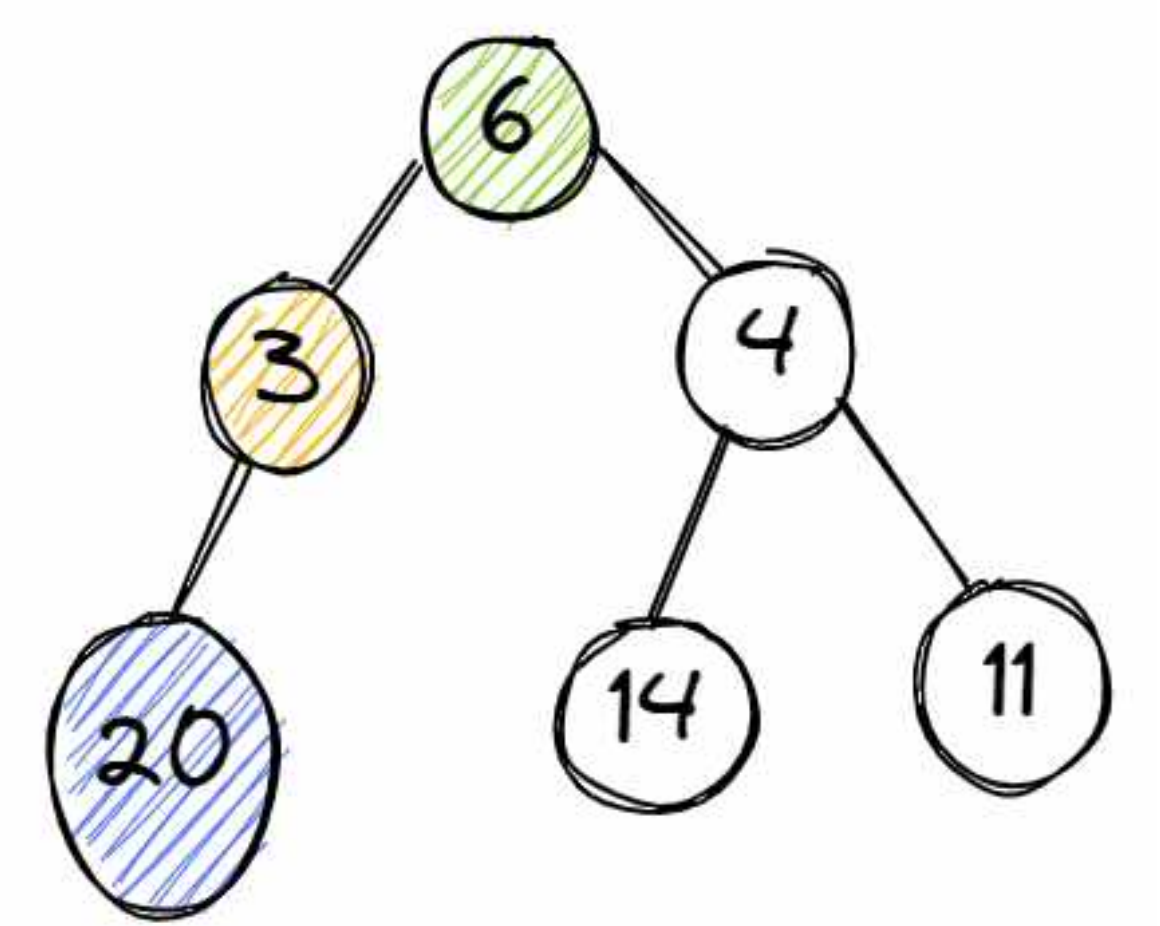
- According to the Oxford English Dictionary, "greedy" means having excessive desire for something without considering the effect or damage done
- Edsger Dijkstra, a computer scientist and mathematician who wanted to calculate a minimum spanning tree, introduced the term "Greedy algorithm". Prim and Kruskal came up with optimization techniques for minimizing cost of graphs.



Greedy



Not Greedy



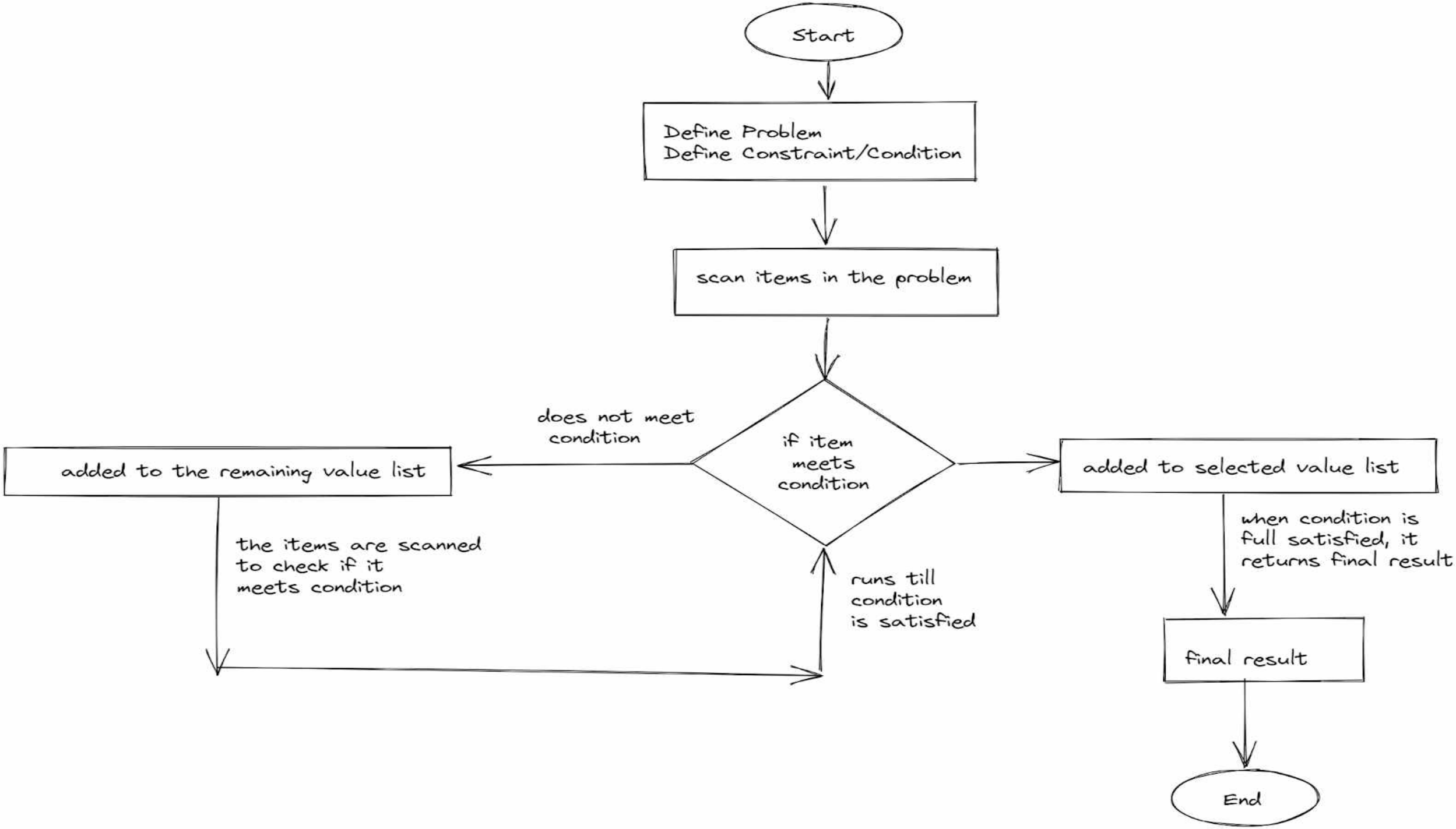
Characteristics of a Greedy Algorithm

- The algorithm solves its problem by finding an optimal solution. This solution can be a maximum or minimum value. It makes choices based on the best option available.

- The algorithm is fast and efficient with time complexity of O(n log n) or O(n). Therefore applied in solving large-scale problems.

- The search for optimal solution is done without repetition – the algorithm runs once.

- It is straightforward and easy to implement.


Start
Define Problem
Define Constraint/Condition
scan items in the problem
if item meets condition
does not meet condition
added to the remaining value list
the items are scanned to check if it meets condition
runs till condition is satisfied
added to selected value list
when condition is full satisfied, it returns final result
final result
End

Advantages of Using a Greedy Algorithm

- Greedy algorithms are quite straight forward to implement and easy to understand. They are also very efficient and have a lower complexity time of O(N * logN).

- They're useful in solving optimization problems, returning a maximum or minimum value.

Disadvantages/Limitations of Using a Greedy Algorithm

- Even though greedy algorithms are straightforward and helpful in optimization problems, they don't offer the best solutions at all times.

- Also greedy algos only run once, so they don't check the correctness of the result produced.

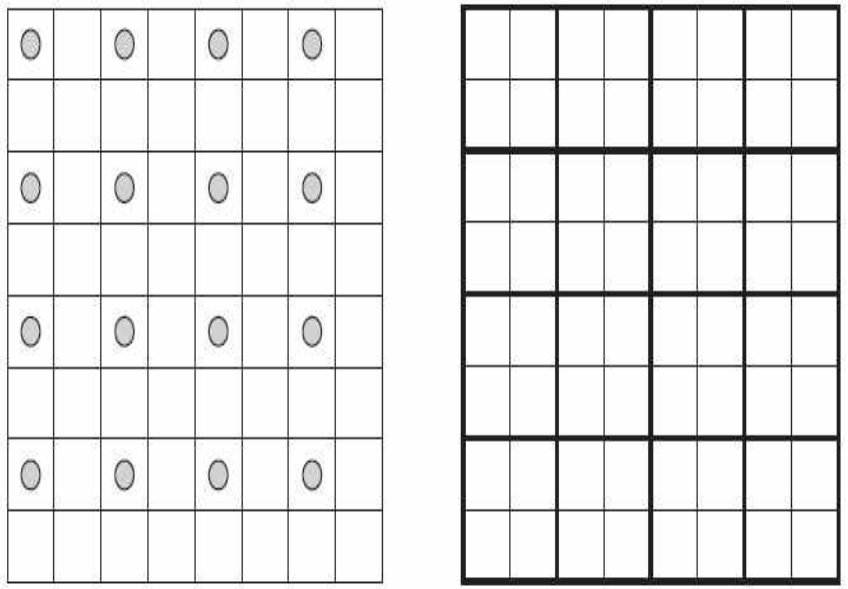


FIGURE 9.1 (a) Placement of 16 chips on non-adjacent squares. (b) Partition of the board proving impossibility of placing more than 16 chips.

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.

Figure 9.2 presents a simple example illustrating these notions.

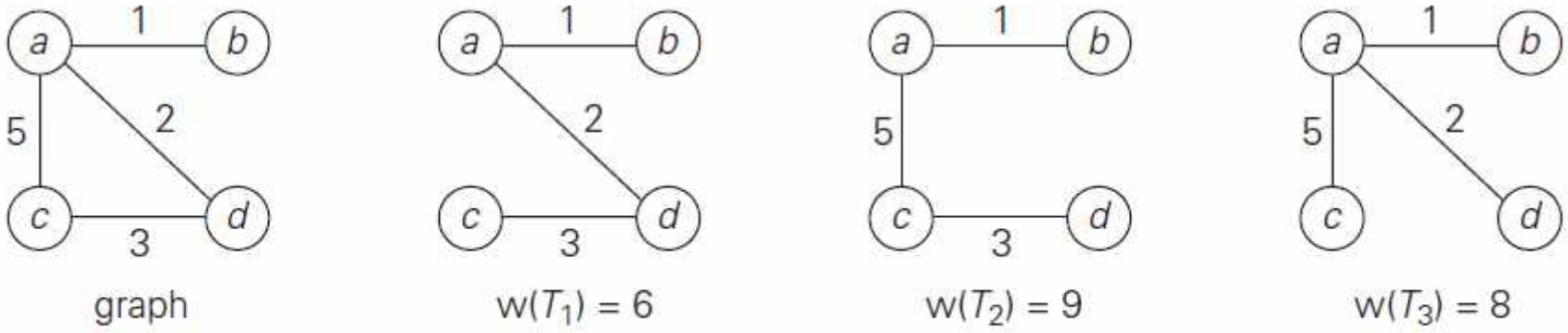
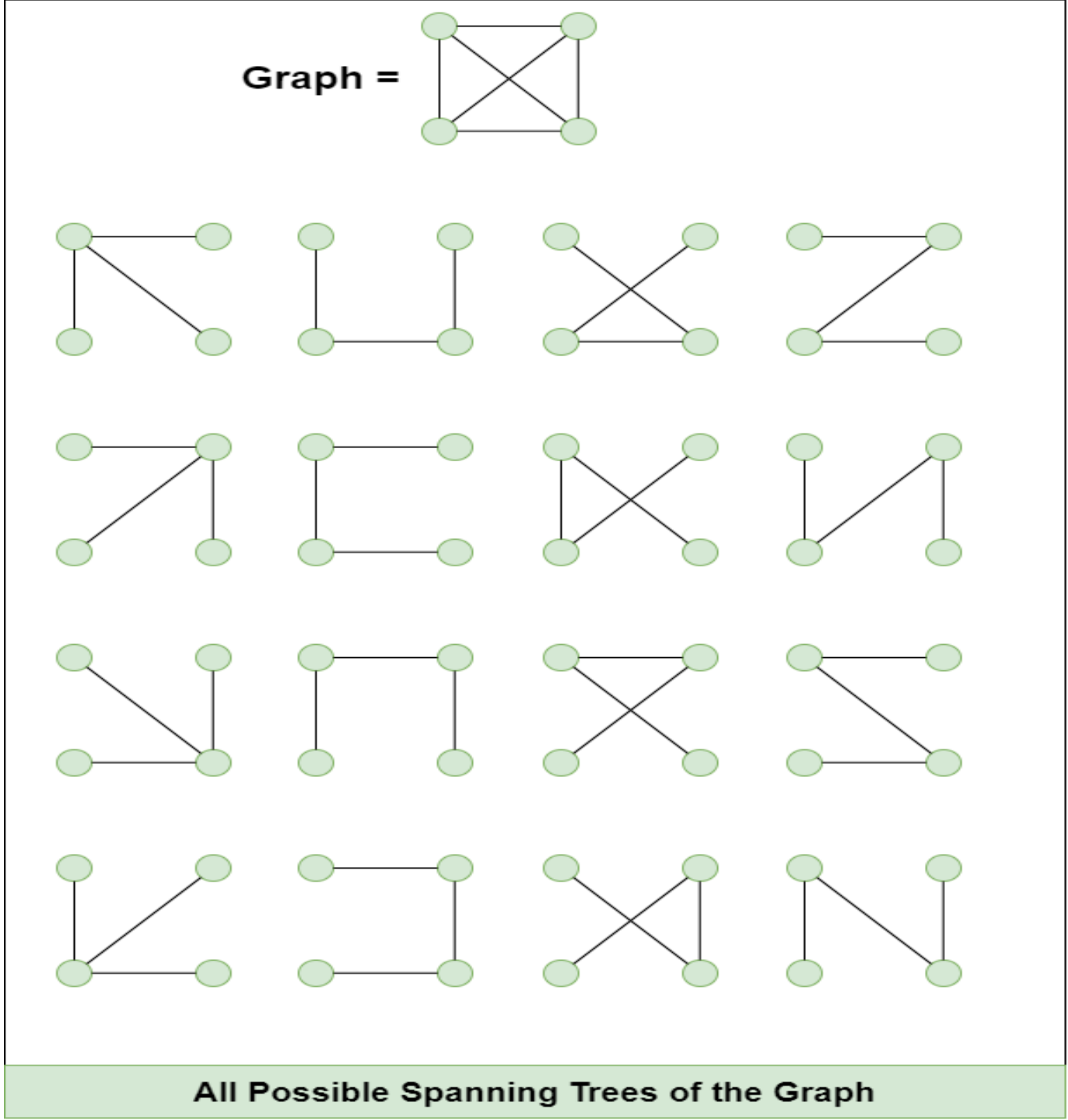


FIGURE 9.2 Graph and its spanning trees, with T_1 being the minimum spanning tree.

Spanning Tree

- What is a Spanning Tree?
- A spanning tree is a subset of Graph G, such that all the vertices are connected using minimum possible number of edges. Hence, a spanning tree does not have cycles and a graph may have more than one spanning tree.

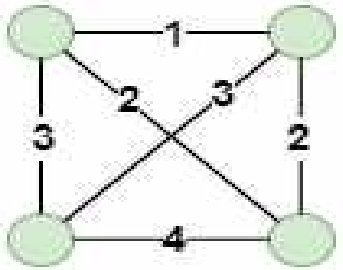


Properties of a Spanning Tree:

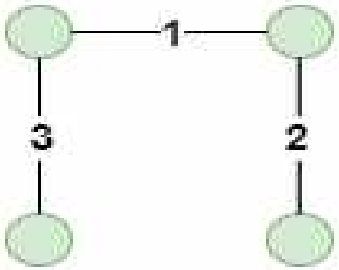
- A Spanning tree does not exist for a disconnected graph.
- For a connected graph having N vertices then the number of edges in the spanning tree for that graph will be N-1.
- A Spanning tree does not have any cycle.
- We can construct a spanning tree for a complete graph by removing E-N+1 edges, where E is the number of Edges and N is the number of vertices.
- Cayley's Formula: It states that the number of spanning trees in a complete graph with N vertices is N^{N-2}
  - For example: N=4, then maximum number of spanning tree possible =4^{4-2} = 16 (shown in the above image).

Minimum Spanning Tree of a Graph may not be Unique:

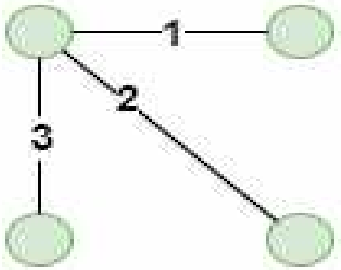
Graph(V,E) =



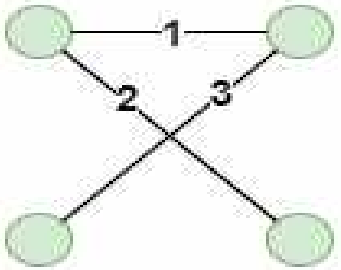
All Possible MST's of the above Graph



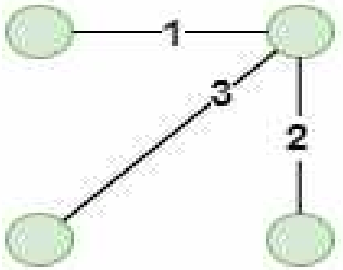
MST Cost =6



MST Cost =6



MST Cost =6



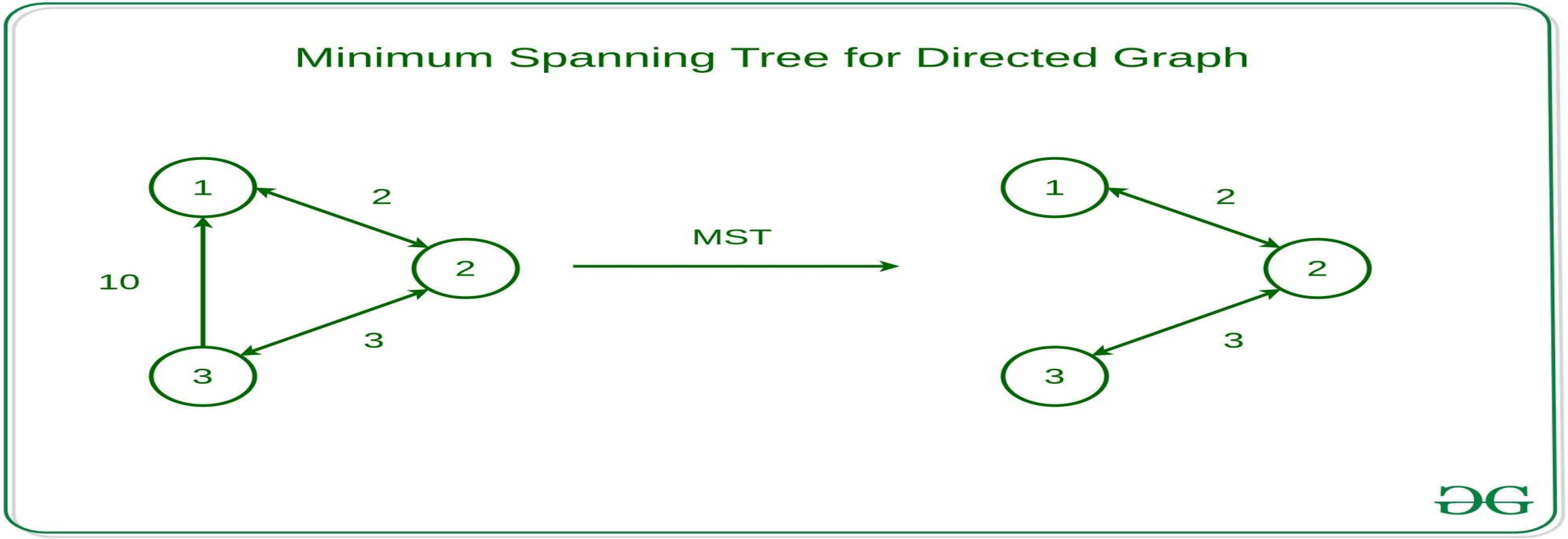
MST Cost =6

Real World Applications of A Spanning Tree:

- Several path finding algorithms, such as Dijkstra's algorithm and A* search algorithm, internally build a spanning tree as an intermediate step.
- Building Telecommunication Network.
- Image Segmentation to break an image into distinguishable components.
- Computer Network Routing Protocol
- Network design: Spanning trees can be used in network design to find the minimum number of connections required to connect all nodes. Minimum spanning trees, in particular, can help minimize the cost of the connections by selecting the cheapest edges.
- Image processing: Spanning trees can be used in image processing to identify regions of similar intensity or color, which can be useful for segmentation and classification tasks.
- Social network analysis: Spanning trees and minimum spanning trees can be used in social network analysis to identify important connections and relationships among individuals or groups.

Minimum Spanning Tree(MST):

- The weight of a spanning tree is determined by the sum of weight of all the edge involved in it.
- A minimum spanning tree (MST) is defined as a spanning tree that has the minimum weight among all the possible spanning trees.



Properties of Minimum Spanning Tree:

- A minimum spanning tree connects all the vertices in the graph, ensuring that there is a path between any pair of nodes.
- An MST is acyclic, meaning it contains no cycles. This property ensures that it remains a tree and not a graph with loops.
- An MST with V vertices (where V is the number of vertices in the original graph) will have exactly V - 1 edges, where V is the number of vertices.
- An MST is optimal for minimizing the total edge weight, but it may not necessarily be unique.
- The cut property states that if you take any cut (a partition of the vertices into two sets) in the original graph and consider the minimum-weight edge that crosses the cut, that edge is part of the MST.

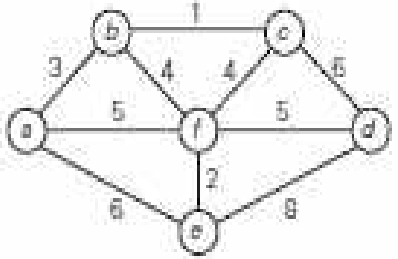
Kruskal's Algorithm

- Remarkably, there is another greedy algorithm for the minimum spanning tree problem that also always yields an optimal solution.
- It is named Kruskal's algorithm after Joseph Kruskal, who discovered this algorithm when he was a second-year graduate student.
- 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.

Pseudocode of 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 $w(e_{i_1}) \leq \cdots \leq w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $ecounter \leftarrow 0$ //initialize the set of tree edges and its size
$k \leftarrow 0$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
    $k \leftarrow k + 1$
    if $E_T \cup \{e_{i_k}\}$ is acyclic
        $E_T \leftarrow E_T \cup \{e_{i_k}\}$; $ecounter \leftarrow ecounter + 1$
return E_T



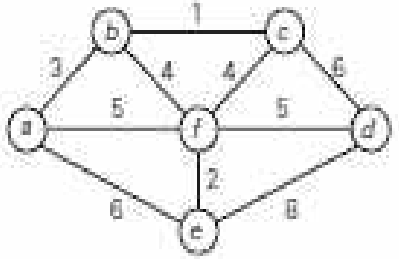
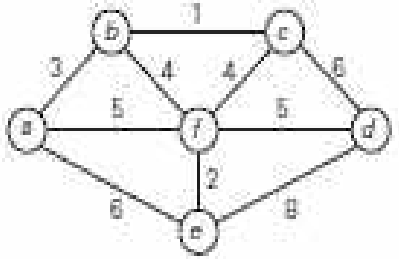
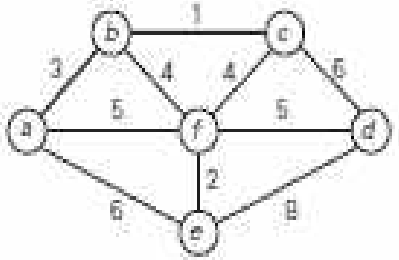
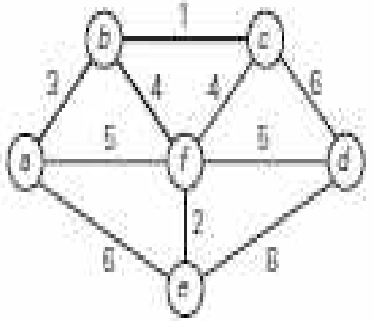
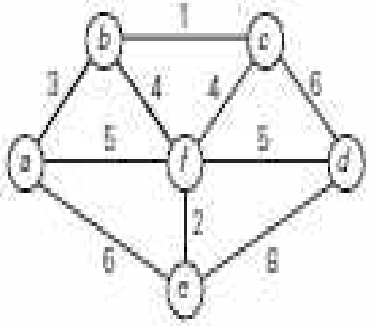
Tree edges	Sorted list of edges	Illustration
	bc ef ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bc 1	bc **ef** ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ef 2	bc ef **ab** bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ab 3	bc ef ab **bf** cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bf 4	bc ef ab bf cf af **df**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

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

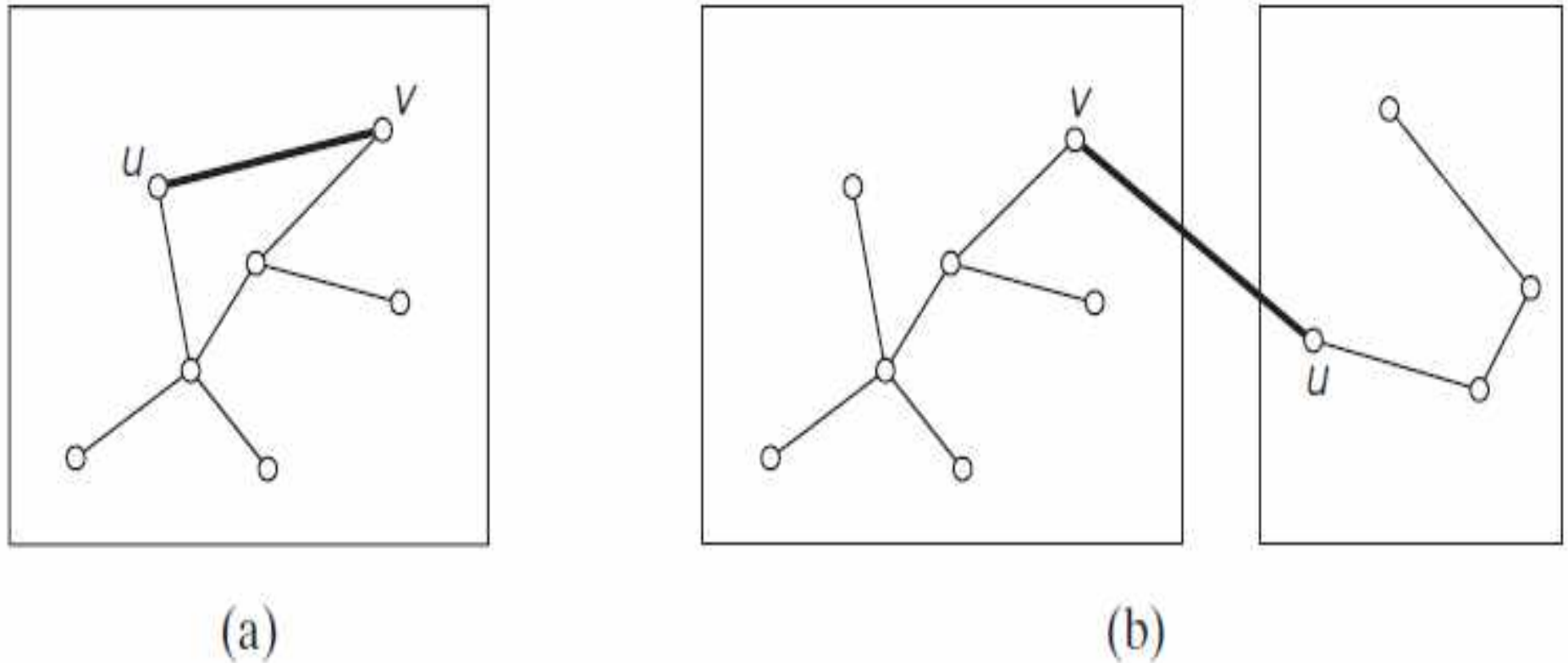


FIGURE 9.6 New edge connecting two vertices may (a) or may not (b) create a cycle.

Dijkstra's Algorithm

- A variety of practical applications of the shortest-paths problem have made
- the problem a very popular object of study:
  - Transportation Planning
  - Packet Routing In Communication Networks, Including The Internet.
  - Include Finding Shortest Paths In Social Networks,
  - Speech Recognition,
  - Document Formatting,
  - Robotics,
  - Compilers,
  - Airline Crew Scheduling.
  - Pathfinding In Video Games
  - Finding Best Solutions To Puzzles Using Their State-space Graphs
- This algorithm is applicable to undirected and directed graphs with nonnegative weights only

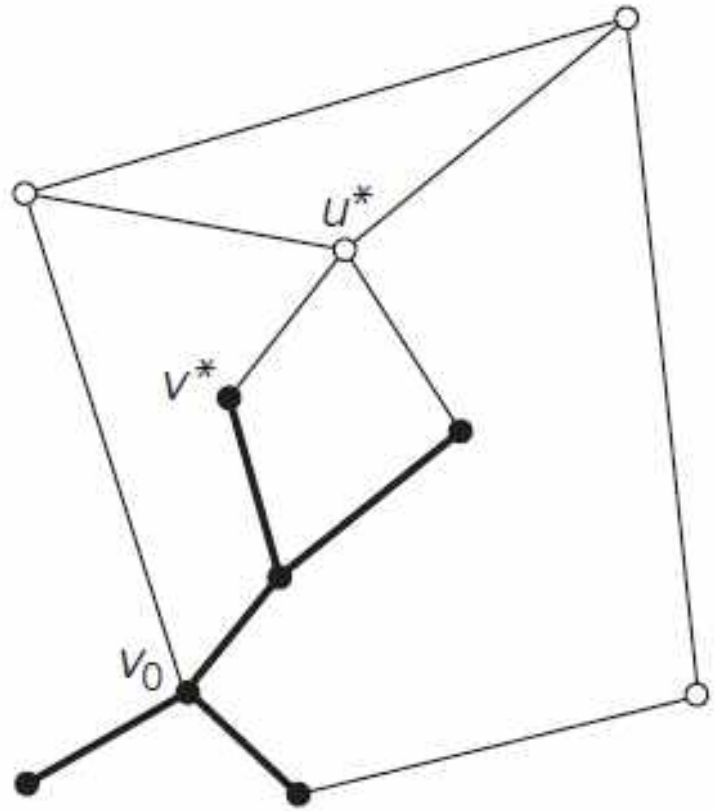
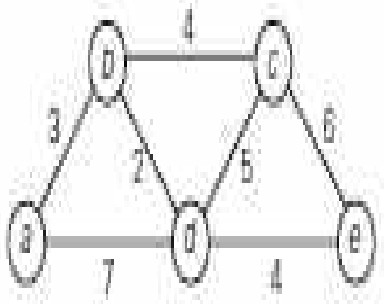


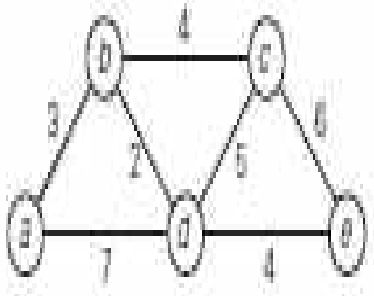
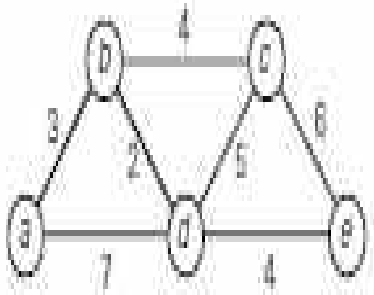
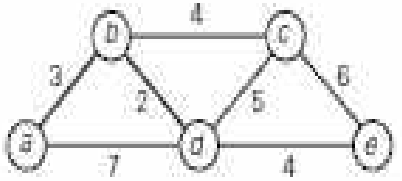
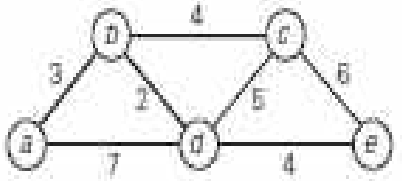
FIGURE 9.10 Idea of Dijkstra's algorithm. The subtree of the shortest paths already found is shown in bold. The next nearest to the source v_0 vertex, u^*, is selected by comparing the lengths of the subtree's paths increased by the distances to vertices adjacent to the subtree's vertices.

Pseudocode

ALGORITHM *Dijkstra*(G, s)

```
//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph G = ⟨V, E⟩ 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 ← ∞;  p_v ← null
    Insert(Q, v, d_v)  //initialize vertex priority in the priority queue
d_s ← 0;  Decrease(Q, s, d_s)  //update priority of s with d_s
V_T ← ∅
for i ← 0 to |V| − 1 do
    u* ← DeleteMin(Q)  //delete the minimum priority element
    V_T ← V_T ∪ {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 ← d_u* + w(u*, u);  p_u ← u*
            Decrease(Q, u, d_u)
```



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

FIGURE 9.11 Application of Dijkstra's algorithm. The next closest vertex is shown in bold.

Questions: