

Greedy Algorithms

V. Balasubramanian
SSN College of Engineering



Introduction

- Charles Dickens' - Ebenezer Scrooge greedy person ever, fictional or real.
- Each day his only drive was to greedily grab as much gold as he could.
- Christmas Carol



ssn

Contd...

- They often lead to very efficient and simple solutions.
- Like dynamic programming, greedy algorithms are often used to solve optimization problems.
- The greedy approach is more straightforward.
- DP, a recursive property is used to divide an instance into smaller instances.

Contd...

- In the *greedy approach*, there is no division into smaller instances. A ***greedy algorithm*** arrives at a solution by making a sequence of choices, each of which simply looks the best at the moment. That is, each choice is locally optimal. The hope is that a globally optimal solution will be obtained, but this is not always the case.



Greedy Strategy

```
while (there are more coins and the instance is not solved){  
    grab the largest remaining coin;           // selection procedure  
    if (adding the coin makes the change exceed  
        the amount owed)                       // feasibility check  
        reject the coin;  
    else  
        add the coin to the change;  
    if (the total value of the change equals the  
        amount owed)                           // solution check  
        the instance is solved;  
}
```

Example

- penny, nickel, dime, quarter, half dollar
- $D1 = 25$ (quarter), $d2 = 10$ (dime),
- $d3 = 5$ (nickel), and $d4 = 1$ (penny).
- Change for 36,48

Coins



Amount owed: 36 cents

Step

Total Change

1. Grab quarter



2. Grab first dime



3. Reject second dime



4. Reject nickel



5. Grab penny



Example

- dime, nickel, and penny and 12 paise coin
- Change for 16
- Example:

e.g., $d_1 = 25$, $d_2 = 10$, $d_3 = 1$ and $n = 30$.

Coins



Amount owed: 16 cents

Step

Total Change

1. Grab 12-cent coin



2. Reject dime



3. Reject nickel



4. Grab four pennies



Example

- Coins 1,3,4
- Change for 6.

Approach

- A ***selection procedure*** chooses the next item to add to the set. The selection is performed according to a greedy criterion that satisfies some locally optimal consideration at the time.
- • A ***feasibility check*** determines if the new set is feasible by checking whether it is possible to complete this set in such a way as to give a solution to the instance.
- • A ***solution check*** determines whether the new set constitutes a solution to the instance.

Greedy Technique

- Constructs a solution to an optimization problem piece by piece through a sequence of choices that are:
 - feasible
 - locally optimal
 - irrevocable
- For some problems, yields an optimal solution for every instance.
- For most, does not but can be useful for fast approximations.



Applications

- Optimal solutions:
 - change making for “normal” coin denominations
 - minimum spanning tree (MST)
 - single-source shortest paths
 - simple scheduling problems
 - Huffman codes



Coin Change Problem

Given unlimited amounts of coins of denominations $d_1 > \dots > d_m$,
give change for amount n with the least number of coins

Example: $d_1 = 25c$, $d_2 = 10c$, $d_3 = 5c$, $d_4 = 1c$ and $n = 48c$

Greedy solution:

Greedy solution is

- optimal for any amount and “normal” set of denominations
- may not be optimal for arbitrary coin denominations



Minimum Spanning Tree (MST)

- Spanning tree of a connected graph G : a connected acyclic subgraph of G that includes all of G 's vertices
- Minimum spanning tree of a weighted, connected graph G : a spanning tree of G of minimum total weight

Example

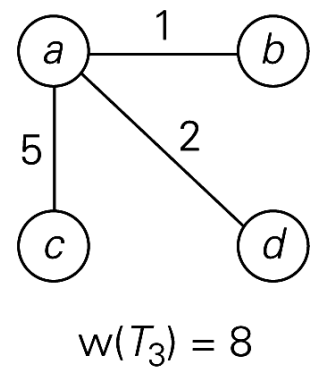
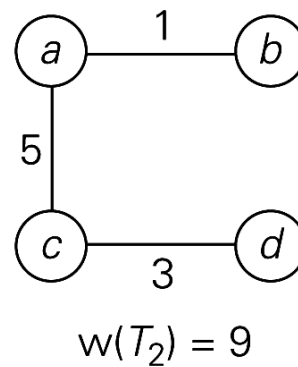
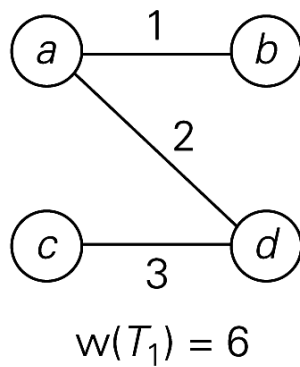
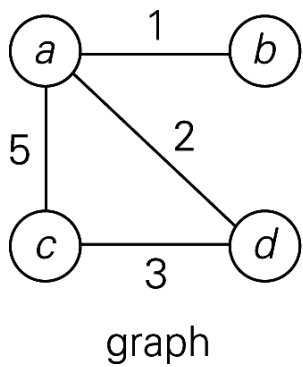


FIGURE 9.1 Graph and its spanning trees; T_1 is the minimum spanning tree

Prims Algorithm

- Suppose an urban planner wants to connect certain cities with roads so that it is possible for someone to drive from any city to any other city.
- Budgetary Constraint

- Robert Clay Prim (born September 25, 1921 in Sweetwater, Texas) is an American mathematician and computer scientist.
- Robert Prim along with coworker Joseph Kruskal
- Prim's algorithm, was originally discovered in 1930 by Czech mathematician Vojtěch Jarník and later independently by Prim in 1957.

Informal Goal: Connect a bunch of points together as cheaply as possible.

Applications: Clustering (more later), networking. Servers, web pages or documents etc

Blazingly Fast Greedy Algorithms:

- Prim's Algorithm [1957; also Dijkstra 1959, Jarnik 1930]
- Kruskal's algorithm [1956]

⇒ $O(m \log n)$ time (using suitable data structures)

of edges

of vertices

Problem

vertices

edges

Input: Undirected graph $G = (V, E)$ and a cost c_e for each edge $e \in E$.

- Assume adjacency list representation (see Part I for details)
- OK if edge costs are negative

Output: minimum cost tree $T \subseteq E$ that spans all vertices.

i.e., sum of edge costs

i.e.: (1) T has no cycles, (2) the subgraph (V, T) is connected (i.e., contains path between each pair of vertices).

Assumption

Assumption #1: Input graph G is connected.

- Else no spanning trees.
- Easy to check in preprocessing (e.g., depth-first search).

Assumption #2: Edge costs are distinct.

- Prim + Kruskal remain correct with ties (which can be broken arbitrarily).

No cycles



(disallowed)

Greedy Algorithm

```
 $F = \emptyset$  // Initialize set of  
// edges to empty.  
while (the instance is not solved){  
    select an edge according to some locally  
    optimal consideration; // selection procedure  
    if (adding the edge to  $F$  does not create a cycle)  
        add it; // feasibility check  
    if ( $T = (V, F)$  is a spanning tree) // solution check  
        the instance is solved;  
}
```


Prims Algorithm

```
 $F = \emptyset;$  // Initialize set of edges  
 $Y = \{v_1\};$  // to empty.  
// Initialize set of vertices to  
// contain only the first one.  
while (the instance is not solved){  
    select a vertex in  $V - Y$  that is // selection procedure and  
    nearest to  $Y;$  // feasibility check  
  
    add the vertex to  $Y;$   
    add the edge to  $F;$   
  
    if ( $Y == V$ ) // solution check  
        the instance is solved;  
}
```

Algorithm

- Initialize $X = \{s\}$ [$s \in V$ chosen arbitrarily]
- $T = \emptyset$ [invariant: X = vertices spanned by tree-so-far T]
- While $X \neq V$
 - Let $e = (u, v)$ be the cheapest edge of G with $u \in X, v \notin X$.
 - Add e to T
 - Add v to X .

While loop: Increase # of spanned vertices in cheapest way possible.



Prims Algorithm

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

$E_T \leftarrow \emptyset$

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 $V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

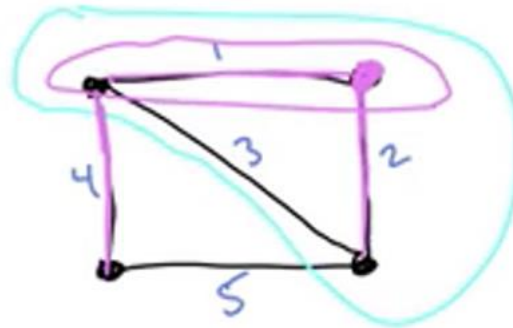
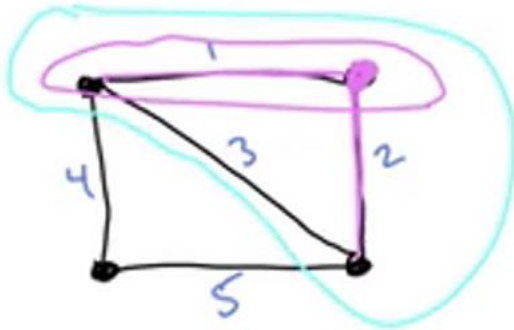
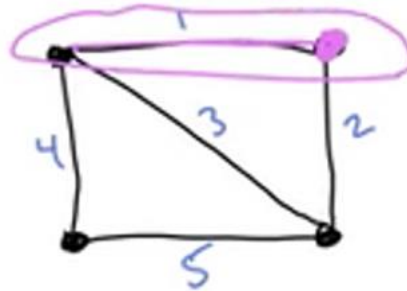
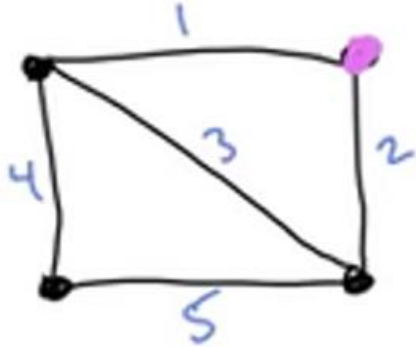
return E_T



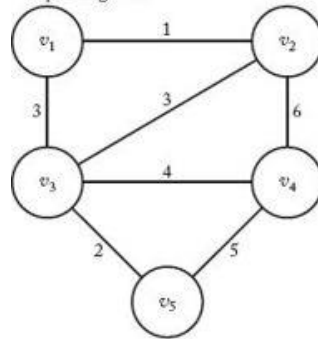
Algorithm

- Start with tree T_1 consisting of one (any) vertex and “grow” tree one vertex at a time to produce MST through a series of expanding subtrees T_1, T_2, \dots, T_n
- On each iteration, construct T_{i+1} from T_i by adding vertex not in T_i that is closest to those already in T_i (this is a “greedy” step!)
- Stop when all vertices are included

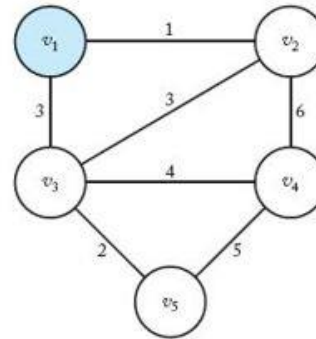
Example



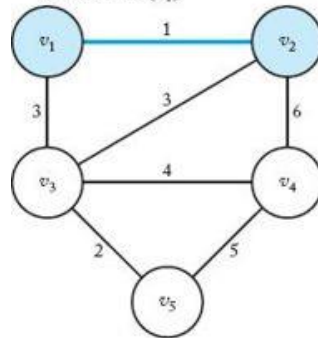
Determine a minimum spanning tree.



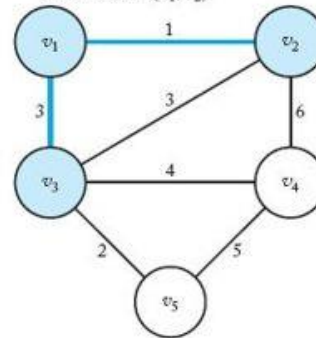
1. Vertex v_1 is selected first.



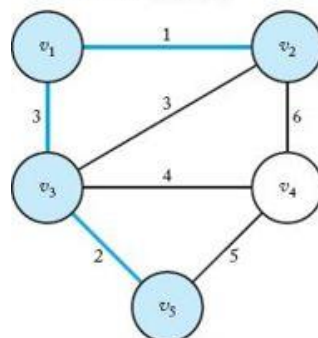
2. Vertex v_2 is selected because it is nearest to $\{v_1\}$.



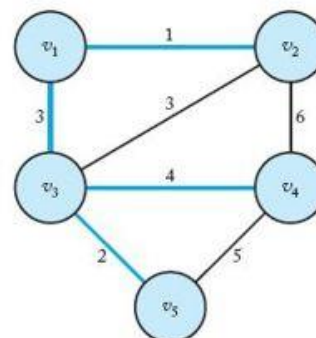
3. Vertex v_3 is selected because it is nearest to $\{v_1, v_2\}$.

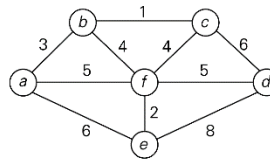


4. Vertex v_5 is selected because it is nearest to $\{v_1, v_2, v_3\}$.



5. Vertex v_4 is selected.





Tree vertices	Remaining vertices	Illustration
$a(-, -)$	$b(a, 3)$ $c(-, \infty)$ $d(-, \infty)$ $e(a, 6)$ $f(a, 5)$	
$b(a, 3)$	$c(b, 1)$ $d(-, \infty)$ $e(a, 6)$ $f(b, 4)$	
$c(b, 1)$	$d(c, 6)$ $e(a, 6)$ $f(b, 4)$	
$f(b, 4)$	$d(f, 5)$ $e(f, 2)$	
$e(f, 2)$	$d(f, 5)$	
$d(f, 5)$		

FIGURE 9.2 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.

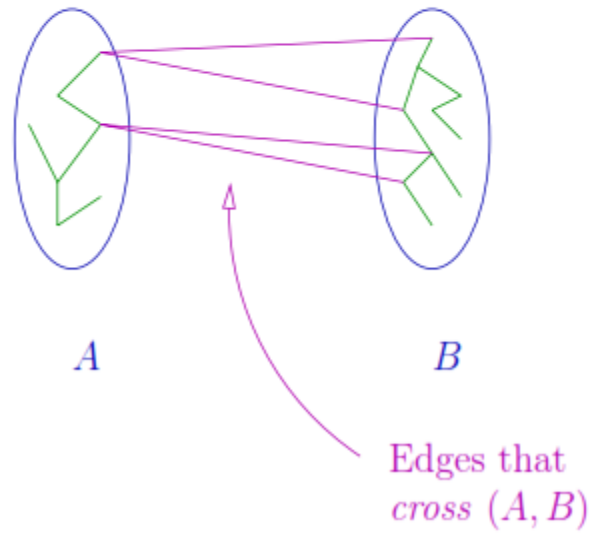
Proof of Correctness

- Mathematical Induction

Correctness

Claim: Prim's algorithm outputs a spanning tree.

Definition: A cut of a graph $G = (V, E)$ is a partition of V into 2 non-empty sets.



- Roughly how many cuts does a graph with n vertices have?
- $2^n - 2$

Contd...

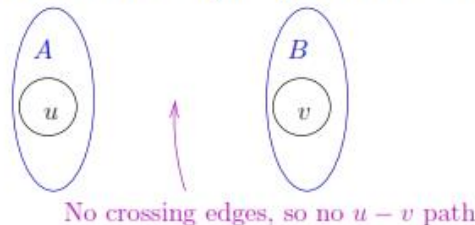
- PQRS
- 0000 A : { P,Q,R,S } B : { } // ignore, B is empty
- 0001 A : { P,Q,R } B : { S }
- 0010 A : { P,Q,S } B : { R }
- 0011 A : { P,Q } B : { R,S }
- 0100 A : { P,R,S } B : { Q }
- 0101 A : { P,R } B : { Q,S }
- 0110 A : { P,S } B : { Q,R }
- 0111 A : { P } B : { Q,R,S }
- 1000 A : { Q,R,S } B : { P }
- 1001 A : { Q,R }, B : { P,S }
- 1010 A : { Q,S } B : { P,R }
- 1011 A : { Q } B : { P,R,S }
- 1100 A : { R,S } B : { P,Q }
- 1101 A : { R } B : { P,Q,S }
- 1110 A : { S } B : { P,Q,R }
- 1111 A : { } B : { P,Q,R,S } // ignore, A empty



Empty Cut Lemma

Empty Cut Lemma: A graph is not connected $\iff \exists \text{ cut } (A, B)$ with no crossing edges.

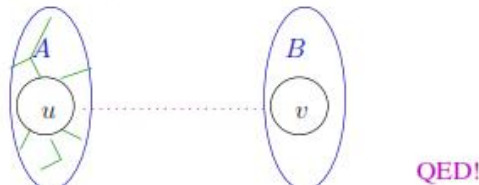
Proof: (\Leftarrow) Assume the RHS. Pick any $u \in A$ and $v \in B$. Since no edges cross (A, B) there is no u, v path in G . $\Rightarrow G$ not connected.



(\Rightarrow) Assume the LHS. Suppose G has no $u - v$ path. Define $A = \{\text{Vertices reachable from } u \text{ in } G\}$ (u 's connected component)

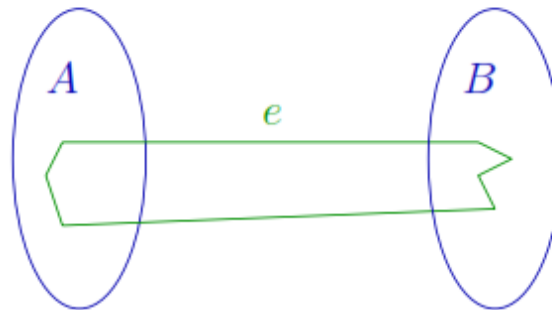
$B = \{\text{All other vertices}\}$ (all other connected components)

Note: No edges cross cut (A, B) (otherwise A would be bigger!)



Lemma 2

Double-Crossing Lemma: Suppose the cycle $C \subseteq E$ has an edge crossing the cut (A, B) : then so does some other edge of C .



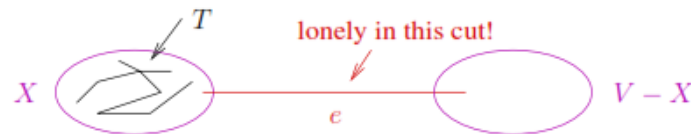
Lonely Cut Corollary: If e is the only edge crossing some cut (A, B) , then it is not in any cycle. [If it were in a cycle, some other edge would have to cross the cut!]

Proof contd...

Claim: Prim's algorithm outputs a spanning tree.

[Not claiming MST yet]

Proof: (1) Algorithm maintains invariant that T spans X
[straightforward induction - you check]



(2) Can't get stuck with $X \neq V$

[otherwise the cut $(X, V - X)$ must be empty; by Empty Cut Lemma input graph G is disconnected]

(3) No cycles ever get created in T . Why? Consider any iteration, with current sets X and T . Suppose e gets added.

Key point: e is the first edge crossing $(X, V - X)$ that gets added to $T \Rightarrow$ its addition can't create a cycle in T (by Lonely Cut Corollary). **QED!**



Analysis

- Initialize $X = \{s\}$ [$s \in V$ chosen arbitrarily]
- $T = \emptyset$ [invariant: X = vertices spanned by tree-so-far T]
- While $X \neq V$
 - Let $e = (u, v)$ be the cheapest edge of G with $u \in X, v \notin X$.
 - Add e to T , add v to X .

Running time of straightforward implementation:

- $O(n)$ iterations [where $n = \#$ of vertices]
 - $O(m)$ time per iteration [where $m = \#$ of edges]
- $\Rightarrow O(mn)$ time



Contd...

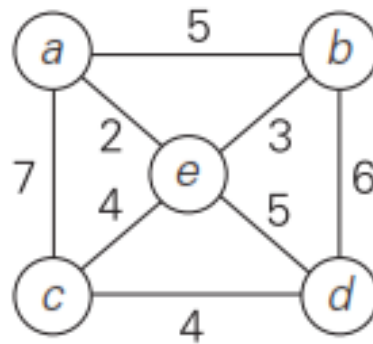
- Proof by induction that this construction actually yields MST
- Needs priority queue for locating closest fringe vertex

Efficiency

- $O(n^2)$ for weight matrix representation of graph and array implementation of priority queue
- $O(m \log n)$ for adjacency list representation of graph with n vertices and m edges and min-heap implementation of priority queue

Exercise

Apply Prim's algorithm to the following graph. Include in the priority queue all the vertices not already in the tree.

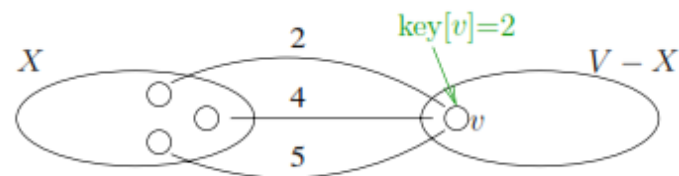


Tree vertices	Priority queue of remaining vertices
a(-,-)	b(a,5) c(a,7) d(a, ∞) e(a,2)
e(a,2)	b(e,3) c(e,4) d(e,5)
b(e,3)	c(e,4) d(e,5)
c(e,4)	d(c,4)
d(c,4)	

The minimum spanning tree found by the algorithm comprises the edges *ae*, *eb*, *ec*, and *cd*.

Invariant #1: Elements in heap = vertices of $V - X$.

Invariant #2: For $v \in V - X$, $\text{key}[v]$ = cheapest edge (u, v) with $u \in X$ (or $+\infty$ if no such edges exist).

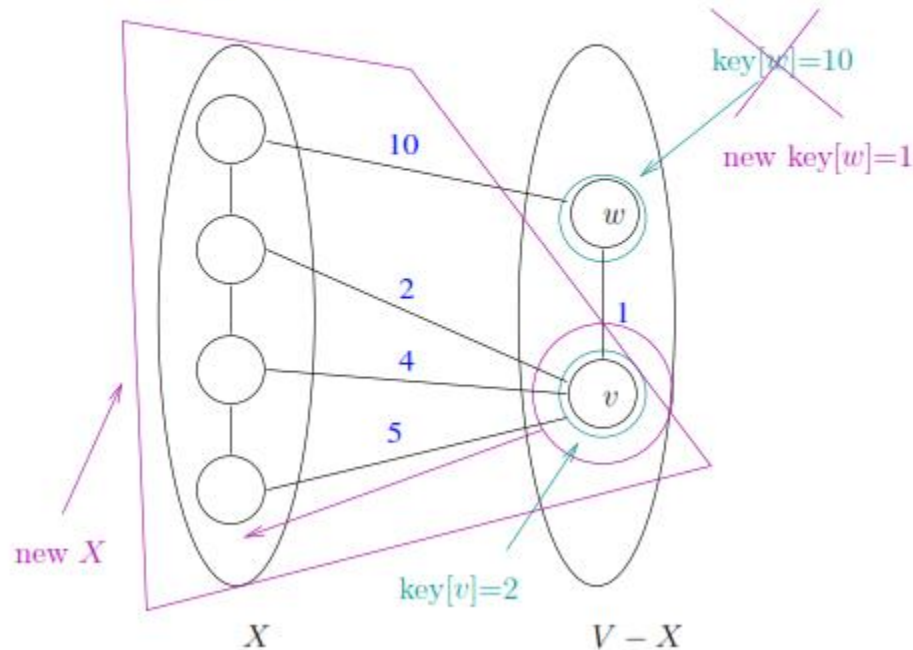


Check: Can initialize heap with $O(m + n \log n) = O(m \log n)$ preprocessing.

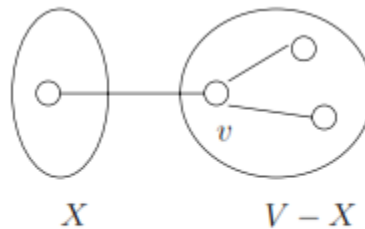
To compare keys $n - 1$ Inserts $m \geq n - 1$ since G connected

2,10,1

Question: What is: (i) current value of $\text{key}[v]$ (ii) current value of $\text{key}[w]$ (iii) value of $\text{key}[w]$ after one more iteration of Prim's algorithm?



When v is added to X



Pseudocode: When v added to X :

- For each edge $(v, w) \in E$:
 - If $w \in V - X \rightarrow$ The only whose key might have changed
(Update key if needed):
 - Delete w from heap
 - Recompute $\text{key}[w] := \min\{\text{key}[w], c_{vw}\}$
 - Re-Insert into heap

$O(|E| \log |V|)$

Kruskal's Algorithm

- Sort the edges in nondecreasing order of lengths
- “Grow” tree one edge at a time to produce MST through a series of expanding forests F_1, F_2, \dots, F_{n-1}
- On each iteration, add the next edge on the sorted list unless this would create a cycle. (If it would, skip the edge.)



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 $w(e_{i_1}) \leq \dots \leq w(e_{i_{|E|}})$

$E_T \leftarrow \emptyset$; $ecounter \leftarrow 0$ //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

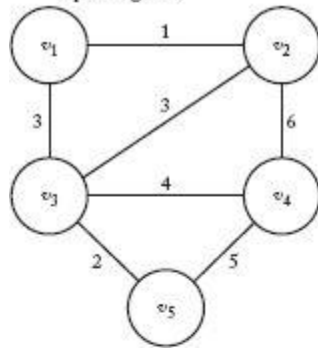


```

void kruskal (int n, int m,
              set_of_edges E,
              set_of_edges& F)
{
    index i, j;
    set_pointer p, q;
    edge e;
    Sort the m edges in E by weight in nondecreasing order;
    F =  $\emptyset$ ;
    initial(n); // Initialize n disjoint subsets.
    while (number of edges in F is less than n - 1){
        e = edge with least weight not yet considered;
        i, j = indices of vertices connected by e;
        p = find(i);
        q = find(j);
        if (! equal(p, q)){
            merge(p, q);
            add e to F;
        }
    }
}

```

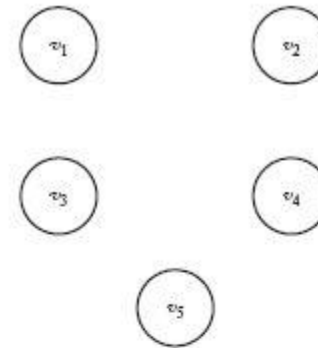

Determine a minimum spanning tree.



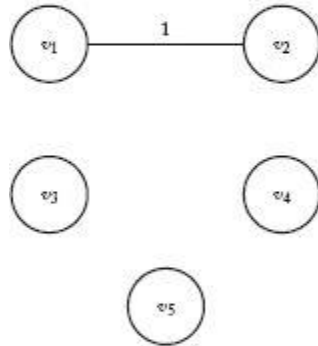
1. Edges are sorted by weight.

- (v_1, v_2) 1
- (v_3, v_5) 2
- (v_1, v_3) 3
- (v_2, v_3) 4
- (v_4, v_5) 5
- (v_2, v_4) 6

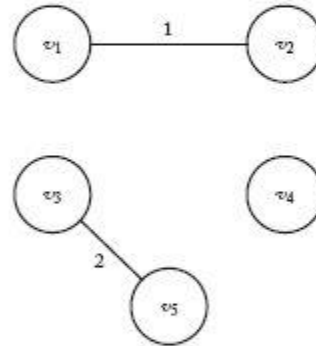
2. Disjoint set are created.



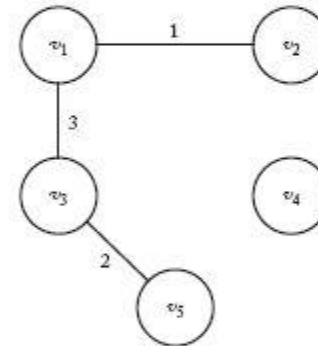
3. Edge (v_1, v_2) is selected.



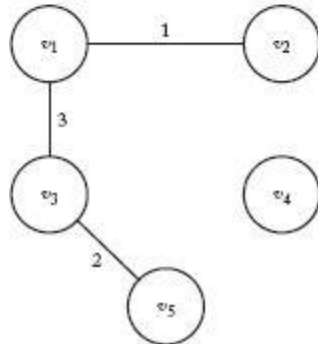
4. Edge (v_3, v_5) is selected.



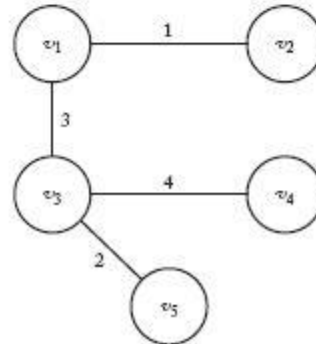
5. Edge (v_1, v_3) is selected.

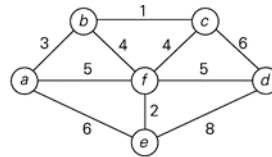


6. Edge (v_2, v_3) is selected.



7. Edge (v_3, v_4) is selected.





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

FIGURE 9.4 Application of Kruskal's algorithm. Selected edges are shown in bold.

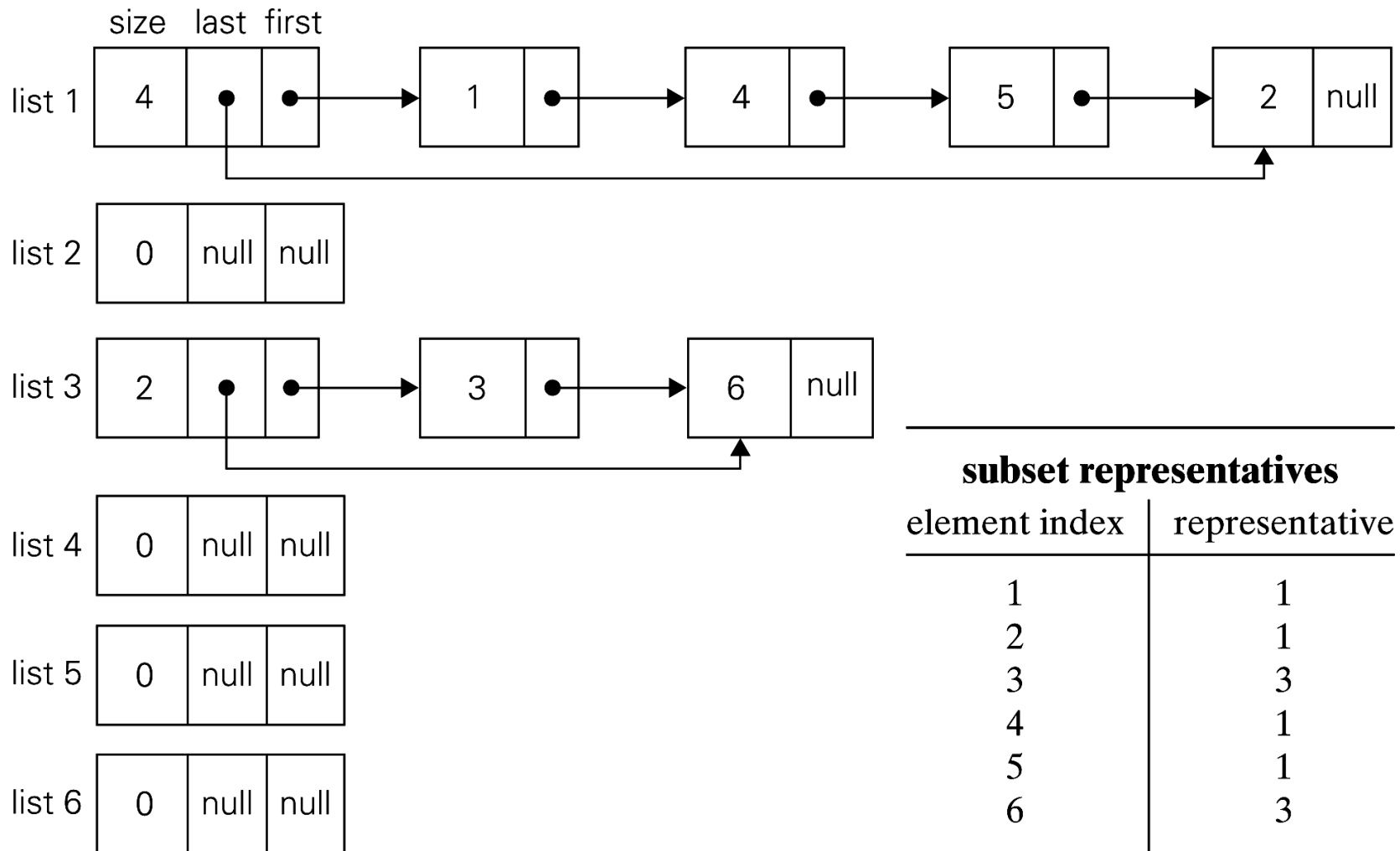


FIGURE 9.6 Linked-list representation of subsets {1, 4, 5, 2} and {3, 6} obtained by quick find after performing $union(1, 4)$, $union(5, 2)$, $union(4, 5)$, and $union(3, 6)$. The lists of size 0 are considered deleted from the collection.

Union, find

makeset(x) creates a one-element set $\{x\}$. It is assumed that this operation can be applied to each of the elements of set S only once.

find(x) returns a subset containing x .

union(x, y) constructs the union of the disjoint subsets S_x and S_y containing x and y , respectively, and adds it to the collection to replace S_x and S_y , which are deleted from it.

For example, let $S = \{1, 2, 3, 4, 5, 6\}$. Then *makeset*(i) creates the set $\{i\}$ and applying this operation six times initializes the structure to the collection of six singleton sets:

$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}$.

Performing *union*(1, 4) and *union*(5, 2) yields

$\{1, 4\}, \{5, 2\}, \{3\}, \{6\}$,

and, if followed by *union*(4, 5) and then by *union*(3, 6), we end up with the disjoint subsets

$\{1, 4, 5, 2\}, \{3, 6\}$.



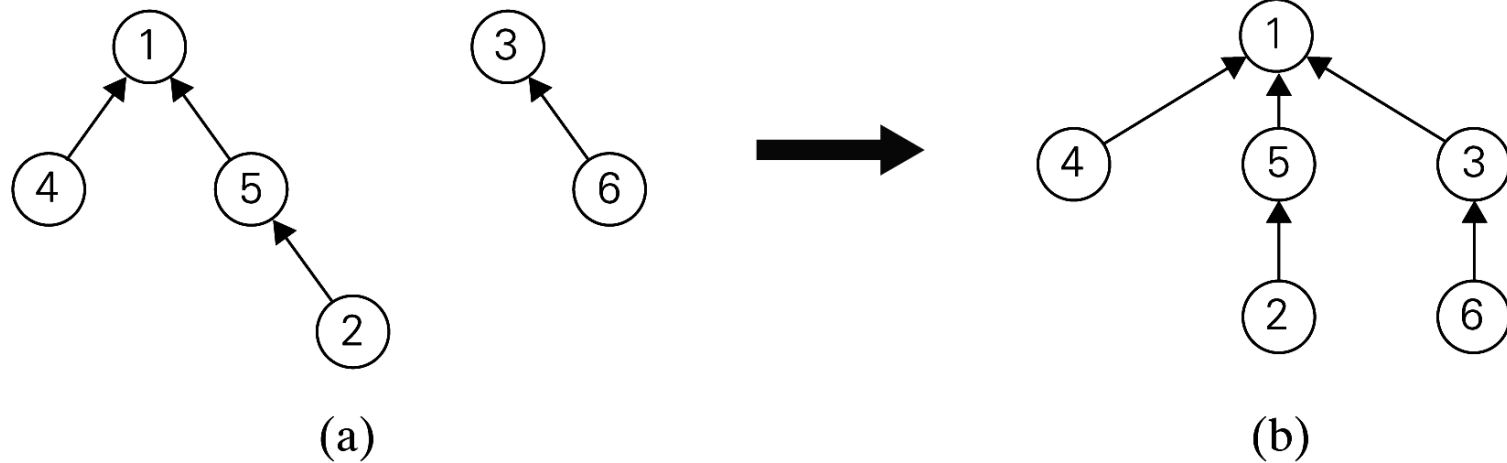


FIGURE 9.7 (a) Forest representation of subsets $\{1, 4, 5, 2\}$ and $\{3, 6\}$ used by quick union. (b) Result of $\text{union}(5, 6)$.

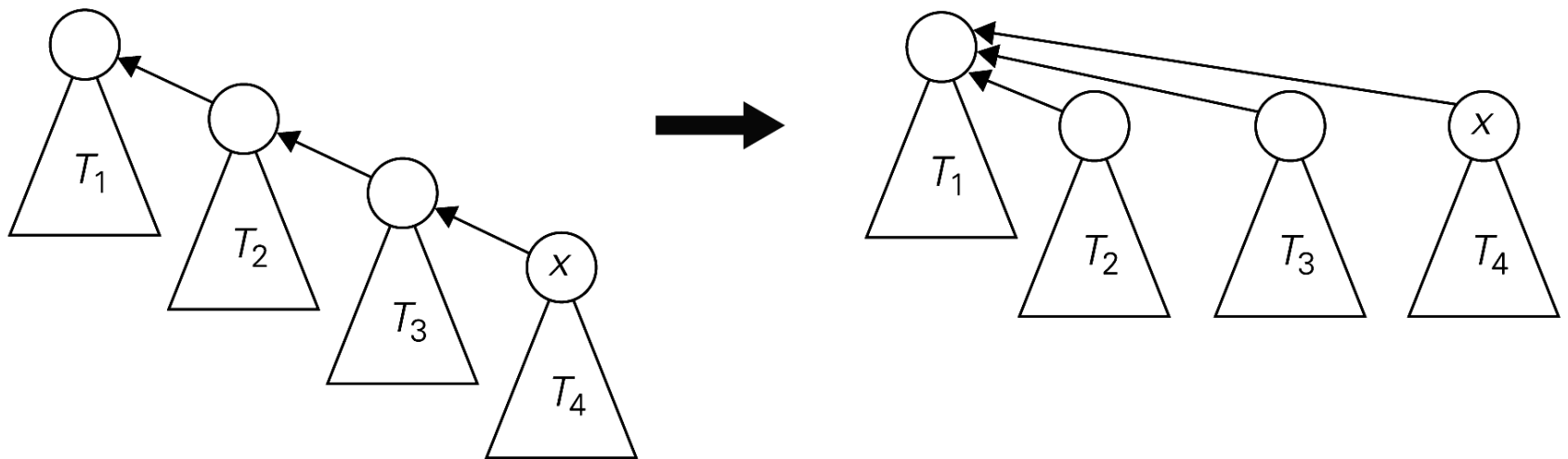
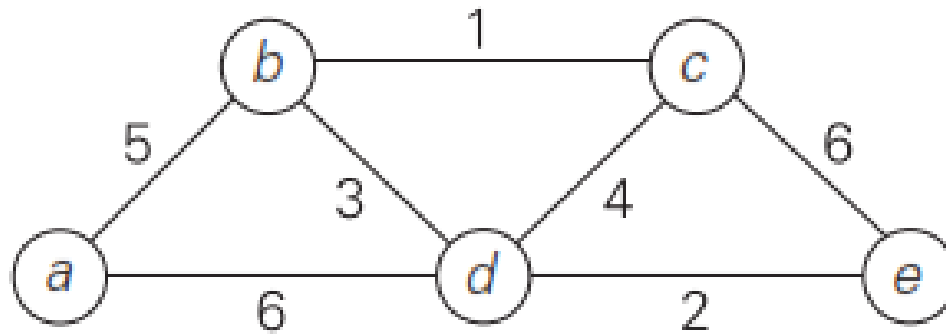


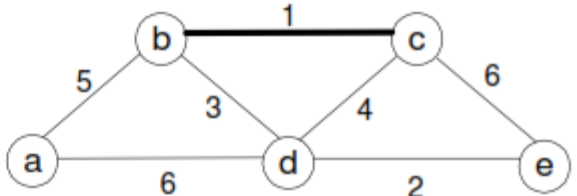
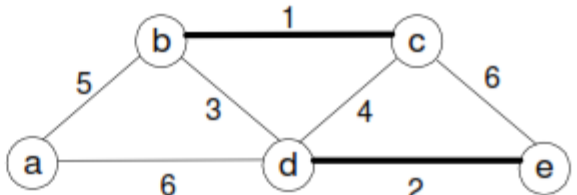
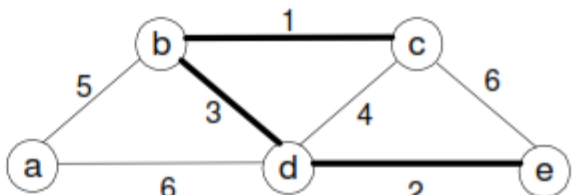
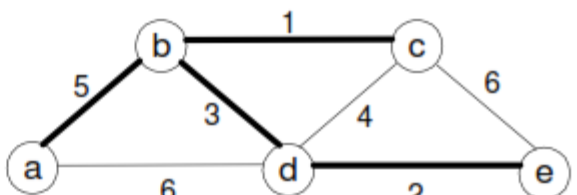
FIGURE 9.8 Path compression

Complexity analysis

- Sorting the edges $m \log m$ – merge sort
- Disjoint set:
- $O(n)$ – to initialise disjoint data set
- $O(m \log m)$ – find, equal, merge

Example



Tree edges	Sorted list of edges (selected edges are shown in bold)							Illustration
	bc ₁	de ₂	bd ₃	cd ₄	ab ₅	ad ₆	ce ₆	
bc ₁	bc ₁	de ₂	bd ₃	cd ₄	ab ₅	ad ₆	ce ₆	
de ₂	bc ₁	de ₂	bd ₃	cd ₄	ab ₅	ad ₆	ce ₆	
bd ₃	bc ₁	de ₂	bd ₃	cd ₄	ab ₅	ad ₆	ce ₆	
ab ₅								

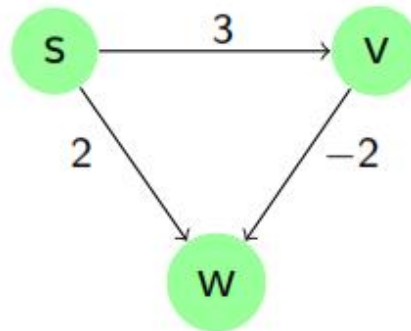
- a.** If e is a minimum-weight edge in a connected weighted graph, it must be among edges of at least one minimum spanning tree of the graph.
- b.** If e is a minimum-weight edge in a connected weighted graph, it must be among edges of each minimum spanning tree of the graph.
- c.** If edge weights of a connected weighted graph are all distinct, the graph must have exactly one minimum spanning tree.
- d.** If edge weights of a connected weighted graph are not all distinct, the graph must have more than one minimum spanning tree.

Exercise

- Does Kruskal's algorithm work correctly on graphs that have negative edge weights?

Dijkstras Assumption

Example: Dijkstra's algorithm with negative edge lengths. What does the algorithm compute as the length of a shortest s - w path, and what is the correct answer?



- Edsger W. Dijkstra (1930–2002), a noted Dutch pioneer of the science and industry of computing, discovered this algorithm in the mid-1950s. Dijkstra said about his algorithm: “This was the first graph problem I ever posed myself and solved. The amazing thing was that I didn’t publish it. It was not amazing at the time. At the time, algorithms were hardly considered a scientific topic.”
- Turing award -1972

Assumption

- This algorithm is applicable to undirected and directed graphs with nonnegative weights only.

- Single Source Shortest Paths Problem: Given a weighted
- connected graph G , find shortest paths from source vertex s
- to each of the other vertices
- Dijkstra's algorithm: Similar to Prim's MST algorithm, with
- a different way of computing numerical labels: Among vertices
- not already in the tree, it finds vertex u with the smallest sum
- $d_v + w(v,u)$
- where v is a vertex for which shortest path has been already found
- on preceding iterations (such vertices form a tree)
- d_v is the length of the shortest path from source to v
- $w(v,u)$ is the length (weight) of edge from v to u

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 vertex priority queue to empty

for every vertex v in V **do**

$d_v \leftarrow \infty$; $p_v \leftarrow \text{null}$

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

$V_T \leftarrow \emptyset$

for $i \leftarrow 0$ **to** $|V| - 1$ **do**

$u^* \leftarrow \text{DeleteMin}(Q)$ //delete the minimum priority element

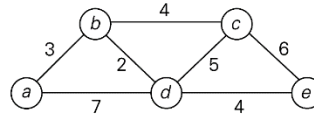
$V_T \leftarrow V_T \cup \{u^*\}$

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)



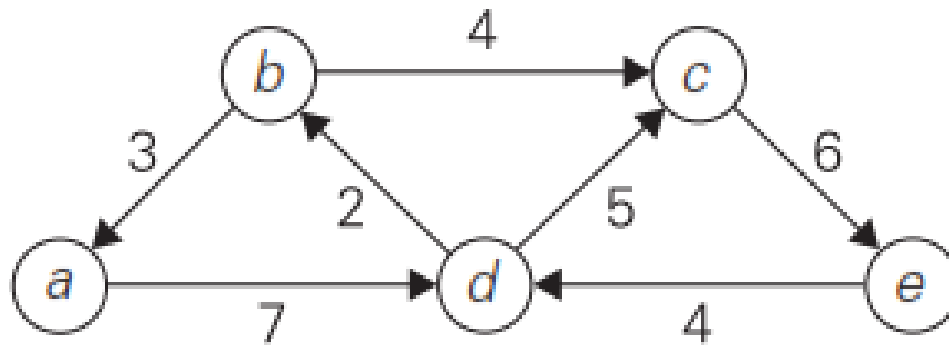
Tree vertices	Remaining vertices	Illustration
$a(-, 0)$	$b(a, 3)$ $c(-, \infty)$ $d(a, 7)$ $e(-, \infty)$	
$b(a, 3)$	$c(b, 3 + 4)$ $d(b, 3 + 2)$ $e(-, \infty)$	
$d(b, 5)$	$c(b, 7)$ $e(d, 5 + 4)$	
$c(b, 7)$	$e(d, 9)$	
$e(d, 9)$		

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

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

Example



solution

Tree vertices	Remaining vertices			
a(-,0)	b(-,∞)	c(-,∞)	d(a,7)	e(-,∞)
d(a,7)	b(d,7+2)	c(d,7+5)	e(-,∞)	
b(d,9)	c(d,12)	e(-,∞)		
c(d,12)	e(c,12+6)			
e(c,18)				

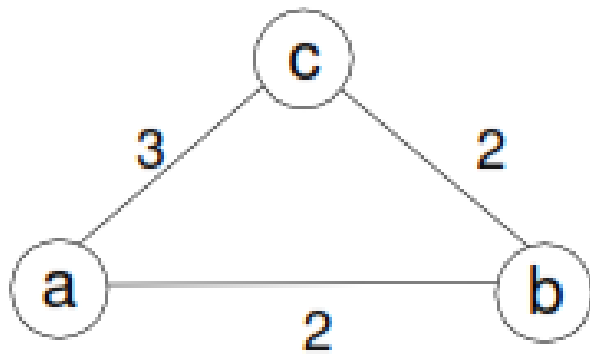
The shortest paths (identified by following nonnumeric labels backwards from a destination vertex to the source) and their lengths are:

from a to d : $a - d$ of length 7
 from a to b : $a - d - b$ of length 9
 from a to c : $a - d - c$ of length 12
 from a to e : $a - d - c - e$ of length 18

Example

- Let T be a tree constructed by Dijkstra's algorithm in the process of solving the single-source shortest-paths problem for a weighted connected graph G .
- **a. True or false: T is a spanning tree of G ?**
- **b. True or false: T is a minimum spanning tree of G ?**





Huffman code

EXAMPLE Consider the five-symbol alphabet {A, B, C, D, _} with the following occurrence frequencies in a text made up of these symbols:

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

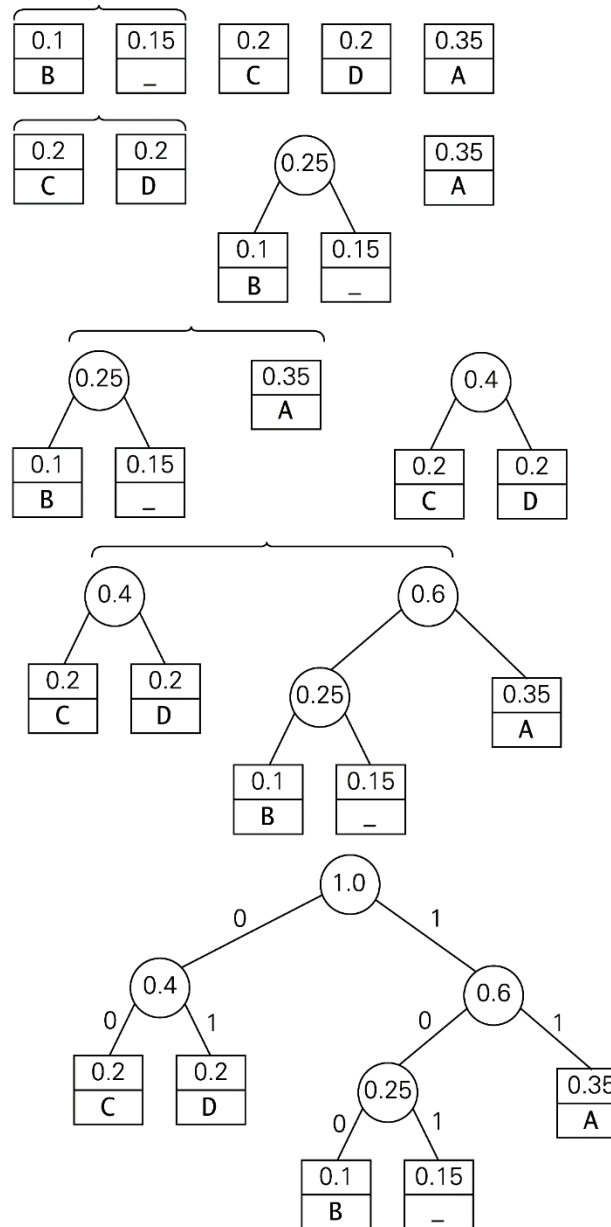


FIGURE 9.11 Example of constructing a Huffman coding tree

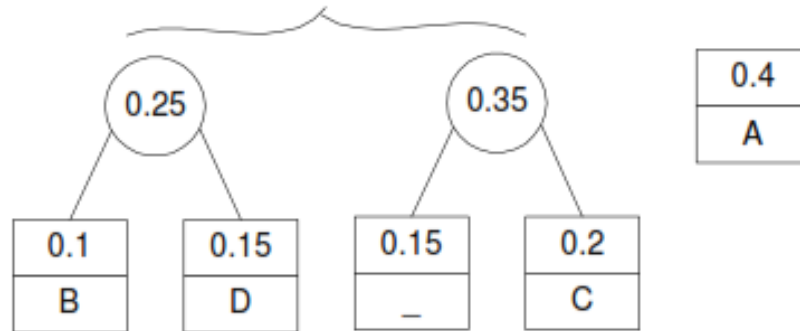
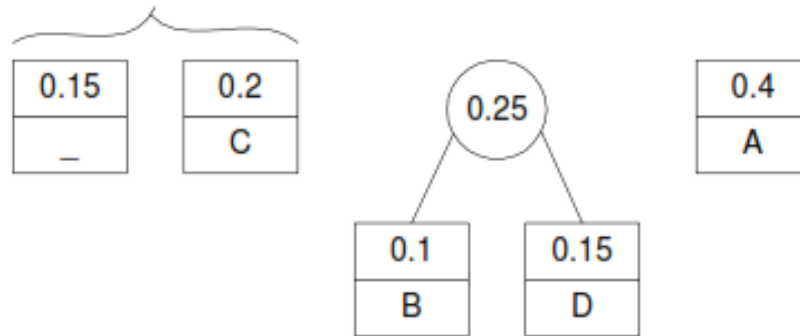
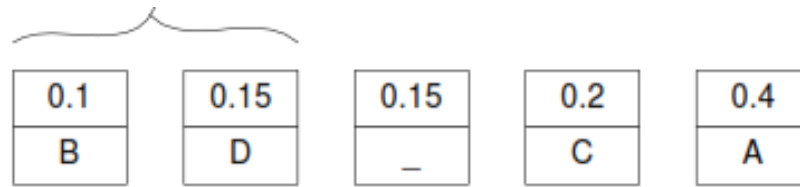
The resulting codewords are as follows:

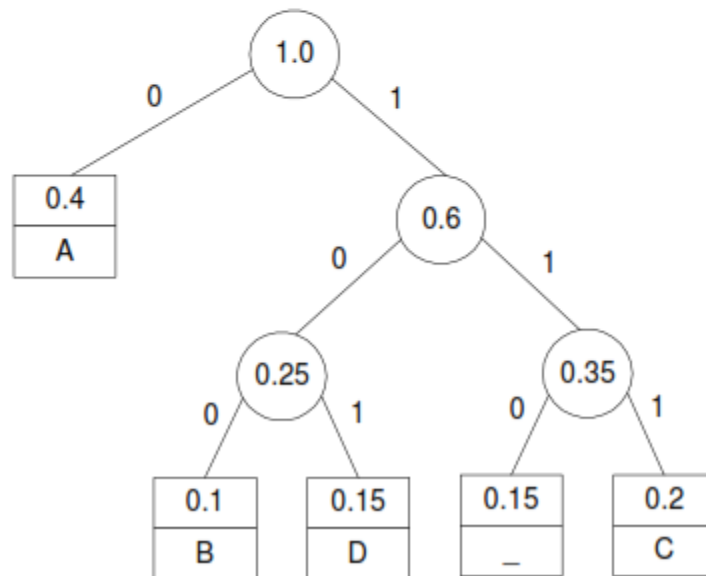
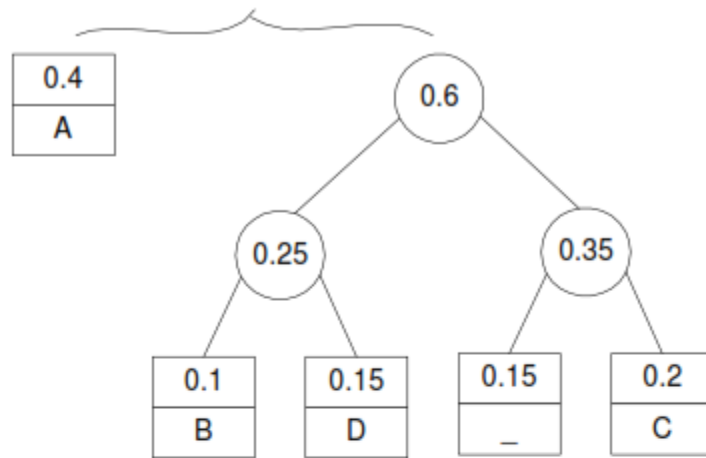
symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

Exercise

symbol	A	B	C	D	_
frequency	0.4	0.1	0.2	0.15	0.15

- b.** Encode ABACABAD using the code of question (a).
- c.** Decode 100010111001010 using the code of question (a).





solution

character	A	B	C	D	—
probability	0.4	0.1	0.2	0.15	0.15
codeword	0	100	111	101	110

b. The text ABACABAD will be encoded as 0100011101000101.

c. With the code of part a, 100010111001010 will be decoded as

100|0|101|110|0|101|0
_{B A D _ A D A}

Exercise

- What is the maximal length of a codeword possible in a Huffman encoding of an alphabet of n symbols?