

# Chapter 4

## Greedy Approach

### LEARNING OBJECTIVES

- ☞ Greedy approach
- ☞ Knapsack problem
- ☞ Fractional knapsack problem
- ☞ Spanning trees
- ☞ Prim's algorithm
- ☞ Kruskal's algorithm
- ☞ Tree and graph traversals
- ☞ Back tracking
- ☞ Graph traversal
- ☞ Breadth first traversal
- ☞ Depth first search
- ☞ Huffman codes
- ☞ Task-scheduling problem
- ☞ Sorting and order statistics
- ☞ Simultaneous minimum and maximum
- ☞ Graph algorithms

### GREEDY APPROACH

In a greedy method, we attempt to construct an optimal solution in stages.

- At each stage we make a decision that appears to be the best (under some criterion) at the time.
- A decision made at one stage is not changed in a later stage, so each decision should assure feasibility.
- Some problems that use greedy approach are:
  1. Knapsack problem
  2. Minimum spanning tree
  3. Prim's algorithm
  4. Kruskal's algorithm

### KNAPSACK PROBLEM

The knapsack problem is a problem in combinatorial optimization: given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. We have  $n$  kinds of items, 1 through  $n$ . Each kind of item  $i$  has a value  $V_i$  and a weight  $W_i$ , usually assume that all values and weights are non-negative. The maximum weight that we can carry in the bag is  $W$ .

### Solved Examples

#### Example 1: (Making change)

Problem:	Accept $n$ dollars, to return a collection of coins with a total value of $n$ dollars.
Configuration:	A collection of coins with a total value of $n$ .
Objective function:	Minimize number of coins returned.
Greedy solution:	Always return the largest coin you can.

- Coins are valued \$.30, \$.020, \$.05, \$.01 use a greedy choice property and make \$.40 by using 3 coins.

**Solution:**  $\$0.30 + \$0.05 + \$0.05 = \$0.40$

### Fractional Knapsack Problem

Given: A set  $S$  of  $n$  items, with each item  $i$  having

- $b_i$  – a positive benefit
- $w_i$  – a positive weight

Goal: Choose items with maximum total benefit but with weight at most  $W$ .






If we are allowed to take fractional amounts, then this is the fractional knapsack problem.


- In this case, let  $x_i$  denote the amount we take of item  $i$ .

- Objective: Maximize  $\sum_{i \in S} b_i(x_i/w_i)$

- Constraint:  $\sum_{i \in S} x_i \leq W$

**Example 2:**

Items					
Weight	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit	\$12	\$32	\$40	\$30	\$50
Value (\$ per ml)	3	4	20	5	50

  
 10 ml

**Solution:** 1 ml of 5, 2 ml of 3, 6 ml of 4, 1 ml of 2

- Greedy choice: Keep taking item with highest value (benefit to weight ratio).
- Correctness: suppose there is a better solution, there is an item  $i$  with higher value than a chosen item  $j$ . (i.e.,  $v_j < v_i$ ). If we replace some  $j$  with  $i$ , we get a better solution.

Thus there is no better solution than the greedy one.

$N = 3, m = 20$

$(P_1, P_2, P_3) = (25, 24, 15)$

$(W_1, W_2, W_3) = (18, 15, 10)$

**Example 3:**

	$X_1$	$X_2$	$X_3$	$\Sigma W_i X_i$	$\Sigma P_i X_i$
1.	1/2	1/3	1/4	$9 + 5 + 2.5 = 16.5$	$12.5 + 8 + 3.75 = 24.25$
2.	1	2/15	0	$18 + 2 + 0 = 20$	$25 + 3.2 + 0 = 28.2$
3.	0	2/3	1	$0 + 10 + 10 = 20$	$0 + 16 + 15 = 31$
4.	0	1	1/2	$0 + 15 + 5 = 20$	$0 + 24 + 7.5 = 31.5$

(1), (2), (3), (4) are feasible ones but (4) is the optimum solution.

**SPANNING TREES**

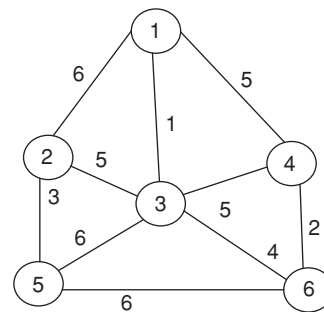
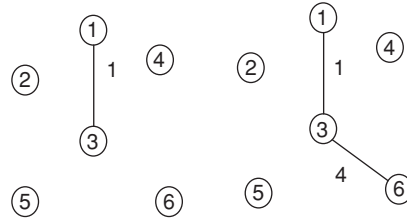
A spanning tree of a graph is just a sub graph that contains all the vertices and is a tree. A graph may have many spanning trees.

- A sub graph that spans (reaches out to) all vertices of a graph is called a spanning sub graph.
- A sub graph that is a tree and that spans all vertices of the original graph is called a spanning tree.
- Among all the spanning trees of a weighted and connected graph, the one (possibly more) with the least total weight is called a Minimum Spanning Tree (MST).

**PRIM'S ALGORITHM**

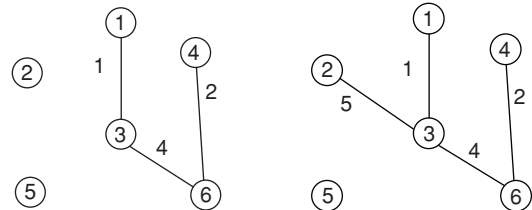
Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm continuously increases the size, of a tree, one edge at a time starting with a tree consisting of a single vertex, until it spans all vertices.

- Using a simple binary heap data structure and an adjacency list representation, prim's algorithm can be shown to run in time  $O(E \log V)$  where  $E$  is the number of edges and  $V$  is the number of vertices.

**Example:****Start:**

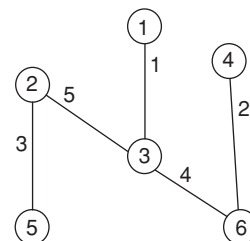
Iteration 1:  $U = \{1, 3\}$

Iteration 2:  $U = \{1, 3, 6\}$



Iteration 3:  $U = \{1, 3, 6, 4\}$

Iteration 4:  $U = \{1, 3, 6, 4, 2\}$



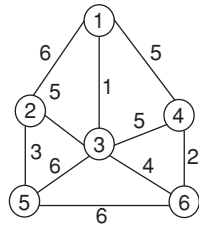
Iteration 5:  $U = \{1, 3, 6, 4, 2, 5\}$

**Figure 1** An example graph for illustrating prim's algorithm.

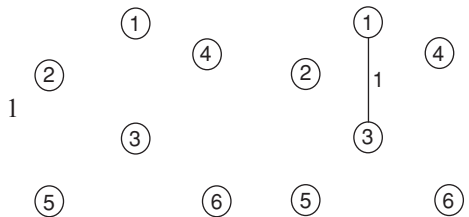
## KRUSKAL'S ALGORITHM

Like prim's algorithm, Kruskal's algorithm also constructs the minimum spanning tree of a graph by adding edges to the spanning tree one-by-one. At all points, during its execution the set of edges selected by prim's algorithm forms exactly one tree. On the other hand the set of edges selected by Kruskal's algorithm forms a forest of trees. Kruskal's algorithm is conceptually simple. The edges are selected and added to the spanning tree in increasing order of their weights. An edge is added to the tree only if it does not create a cycle.

**Example:**

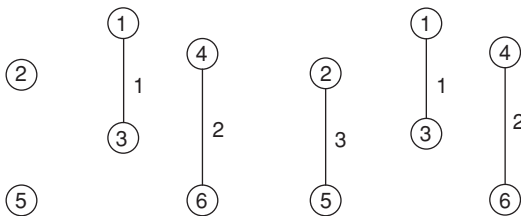


**Start:**



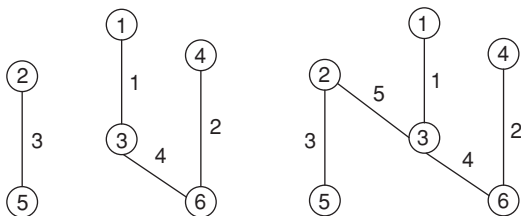
Initial configuration

Setp 1: choose (1, 3)



Setp 2: choose (4, 6)

Setp 3: choose (2, 5)



Setp 4: choose (3, 6)

Setp 6: choose (4, 3)

## TREE AND GRAPH TRAVERSALS

### Back Tracking

Backtracking is a general algorithm technique that considers searching every possible combination in order to solve an optimization problem.

Backtracking is also known as depth first search (or) branch and bound. Backtracking is an important tool for solving constraint satisfaction problems, such as crosswords, verbal arithmetic, sudoku and many other puzzles. It is often the more convenient technique for parsing, for the knapsack problem and other combinatorial optimization problems.

- The advantage of backtracking algorithm is that they are complete, that is they are guaranteed to find every solution to every possible puzzle.

### Graph Traversal

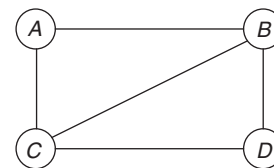
To traverse a graph is to process every node in the graph exactly once, because there are many paths leading from one node to another, the hardest part about traversing a graph is making sure that you do not process some node twice. There are general solutions to this difficulty.

- When you first encounter a node, mark it as REACHED. When you visit a node, check if it is marked REACHED, if it is, just ignore it. This is the method our algorithms will use.
- When you process a node, delete it from the graph. Deleting the node causes the deletion of all the arcs that lead to the node, so it will be impossible to reach it more than once.

### General traversal strategy

- Mark all nodes in the graph as NOT REACHED,
- Pick a starting node, mark it as REACHED, and place it on the READY list.
- Pick a node on the READY list. Process it remove it from READY. Find all its neighbors, those that are NOT REACHED should be marked as REACHED and added to READY.
- Repeat 3 until READY is empty.

**Example:**



**Step I:**  $A = B = C = D = \text{NOT REACHED}$

**Step II:**  $\text{READY} = \{A\}$ .  $A = \text{REACHED}$

**Step III:** Process A.  $\text{READY} = \{B, C\}$ .  
 $B = C = \text{REACHED}$

**Step IV:** Process C.  $\text{READY} = \{B, D\}$ .  
 $D = \text{REACHED}$

**Step V:** Process B.  $\text{READY} = \{D\}$

**Step VI:** Process D.  $\text{READY} = \{\}$

The two most common traversal patterns are

- Breadth first traversal
- Depth first traversal

### Breadth First Traversal

In breadth first traversal, READY is a QUEUE, not an arbitrary list. Nodes are processed in the order they are reached (FIFO), this has the effect of processing nodes according to their distance from the initial node. First, the initial node is processed. Then all its neighbors are processed. Then all of the neighbors etc.

- Since a graph has no root, we must specify the vertex at which to start the traversal.
- Breadth first tree traversal first visits all the nodes at depth zero (i.e., the root) then all the nodes at depth 1, and so on.

### Procedure

First, the starting vertex is enqueued. Then, the following steps are repeated until the queue is empty.

1. Remove the vertex at the head of the queue and call it vertex.
2. Visit vertex
3. Follow each edge emanating from vertex to find the adjacent vertex and call it ' $t_o$ '. If ' $t_o$ ' has not already been put into the queue, enqueue it.

Notice, that a vertex can be put into the queue at most once. Therefore, the algorithm must somehow keep track of the vertices that have been enqueued.

### Procedure for BFS for undirected graph $G(V, E)$

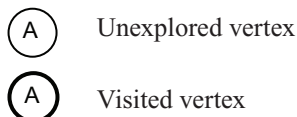
To perform BFS over a graph, the data structures required are queue ( $Q$ ) and the visited set (Visited), ' $V$ ' is the starting vertex.

### Procedure for BFS( $V$ )

Steps

1. Visit the vertex ' $V$ '
2. Enqueue the vertex  $V$
3. while ( $Q$  is not Empty)
  - (i)  $V = \text{dequeue}()$ ;
  - (ii) for all vertices  $\vartheta$  adjacent to  $V$ 
    - (a) if not visited ( $\vartheta$ )
      - Enqueue ( $\vartheta$ )
      - Visit the vertex ' $\vartheta$ '
      - end if.
    - end for
- end while
- Stop

Example:



— Unexplored edge  
 —→ Discovery edge  
 - - - → Cross edge

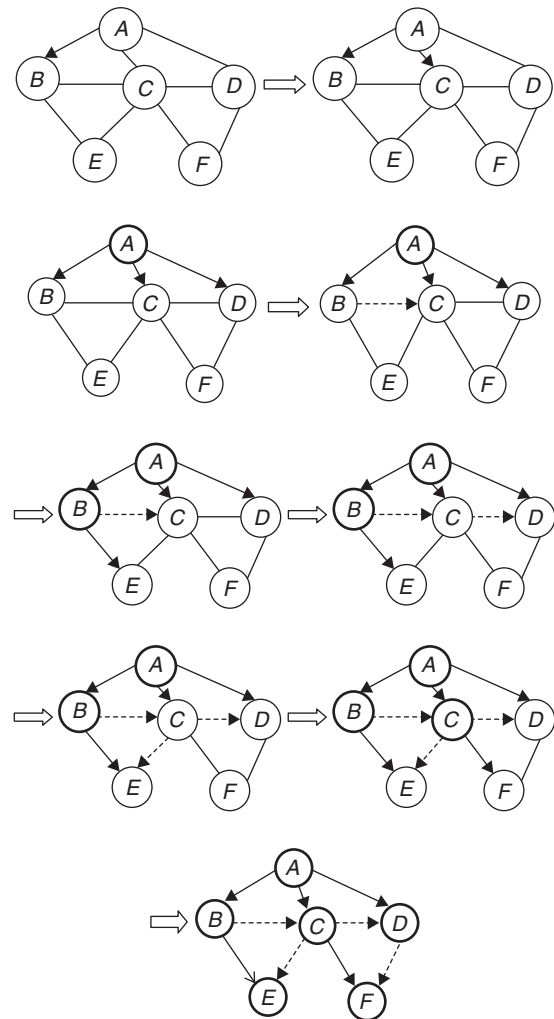


Figure 2 Breadth-first search

### Depth First Search

A depth first traversal of a tree always starts at the root of the tree. Since a graph has no root, when we do a depth first traversal, we must specify the vertex at which to begin. A depth first traversal of a tree visits a node and then recursively visits the sub trees of that node similarly, depth first traversal of a graph visits a vertex and then recursively visits all the vertices adjacent to that node. A graph may contain cycles, but the traversal must visit every vertex at most once.

The solution to the problem is to keep track of the nodes that have been visited.

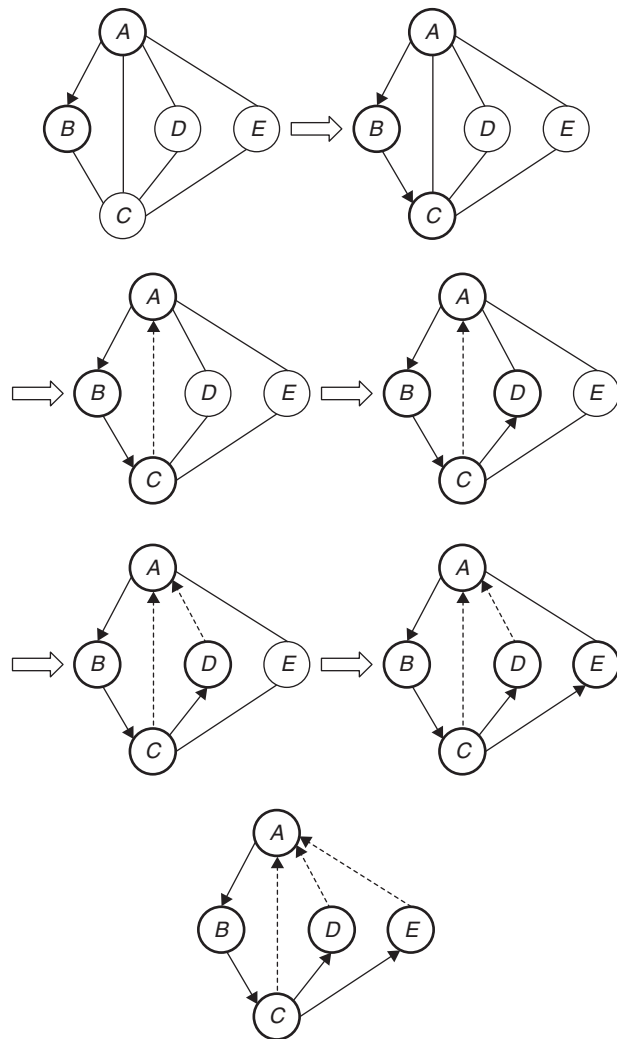
### Procedure for DFS for undirected graph $G(V, E)$

To perform DFS over a graph, the data structures required are stack ( $S$ ) and the list (visited), ' $V$ ' is the start vertex.

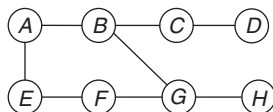
**Procedure for DFS(*V*)**

Steps

1. push the start vertex '*V*' into the stack *S*
2. while (*S* is not empty)
  - (i) pop a vertex *V*
  - (ii) if '*V*' is not visited
    - (a) visit the vertex
    - (b) Store '*V*' in visited
    - (c) push all the adjacent vertices of '*V*' in to visited
  - (iii) End if
3. End while
4. Stop.

**Example:****Figure 3** Depth first search

- Let us compare two traversal orders on the following graph:



Initial steps:

READY = [*A*]. process A. READY = [*B*, *E*]. process *B*.

It is at this point that two traversal strategies differ. Breadth first adds B's neighbors to the back of READY, depth first adds them to the front.

**Breadth first**

- READY = [*E*, *C*, *G*]
- Process *E*. READY = [*C*, *G*, *F*]
- Process *C*. READY = [*G*, *F*, *D*]
- Process *G*. READY = [*F*, *D*, *H*]
- Process *F*. READY = [*D*, *H*]
- Process *D*. READY = [*H*]
- Process *H*. READY = [ ]

**Depth First**

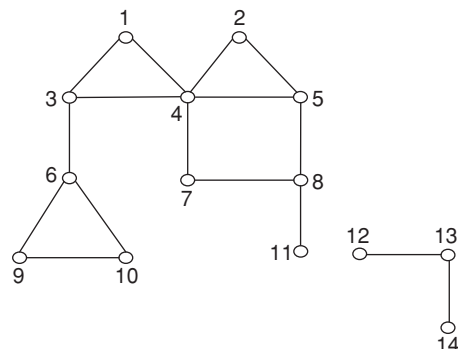
- READY = [*C*, *G*, *E*]
- Process *C*. READY = [*D*, *G*, *E*]
- Process *D*. READY = [*G*, *E*]
- Process *G*. READY = [*H*, *F*, *E*]
- Process *H*. READY = [*F*, *E*]
- Process *F*. READY = [*E*]
- Process *E*. READY = [ ]

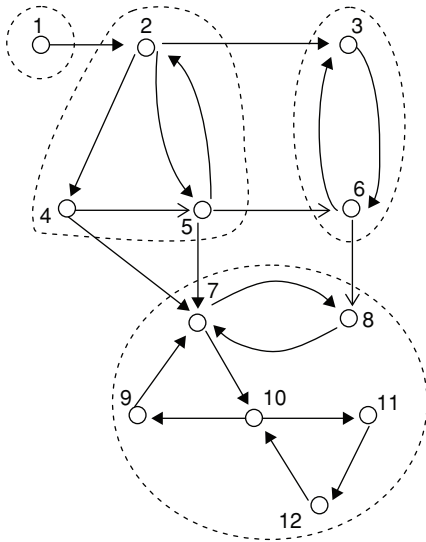
**CONNECTED COMPONENTS**

A graph is said to be connected if every pair of vertices in the graph are connected. A connected component is a maximal connected sub graph of '*G*'. Each vertex belongs to exactly one connected component as does each edge.

- A graph that is not connected is naturally and obviously decomposed into several connected components (Figure 4). Depth first search does this handily. Each restart of the algorithm marks a new connected component.
- The directed graph in (Figure 5) is "Connected" Part of it can be "Pulled apart" (so to speak, without "breaking" any edges).
- Meaningful way to define connectivity in directed graph is:

'Two nodes *U* and *V* of a directed graph  $G = (V, E)$  connected if there is a path from *U* to *V*', and one from *V* to *U*. This relation between nodes is reflective, symmetric and transitive. As such, it partitions *V* into disjoint sets, called the strongly connected components of the graph. In the directed graph of figure 2 there are four strongly connected components.

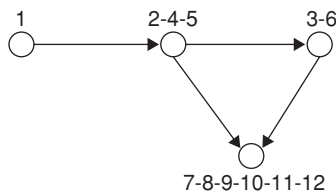
**Figure 4** Undirected graph.



**Figure 5** A directed graph and its strongly connected components

If we shrink each of these strongly connected components down to a single node and draw an edge between two of them if there is an edge from some node in the first to some node in the second, the resulting directed graph has to be a directed acyclic graph (DAG) – it has no cycles (figure 6). The reason is simple.

A cycle containing several strongly connected components would merge them all to a single strongly connected component.



Every directed graph is a DAG of its strongly connected components.

## HUFFMAN CODES

For compressing data, a very effective and widely used technique is Huffman coding. We consider the data to be a sequence of characters. Huffman's greedy algorithm uses a table of the frequencies of occurrence of the characters to build up an optimal way of representing each character as a binary string.

**Example:** Suppose we have a 1,00,000 – character data file, that we wish to store compactly. The characters in the file occur with the frequencies given below:

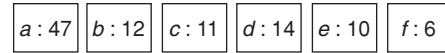
Character	a	b	c	d	e	f
Frequency	47	12	11	14	10	6

**Solution:** Two methods are used for compression of data are:

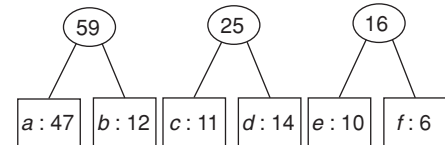
## Fixed Length Coding

- Arrange all the characters in sequence (no particular order is followed)
- $a = 47, b = 12, c = 11, d = 14, e = 10, f = 6$

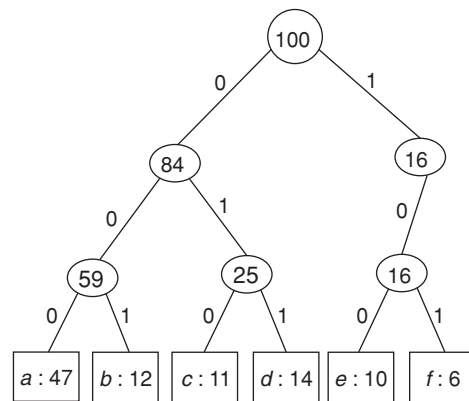
**Step I:**



**Step II:**



**Step III:**



We interpret the binary code word for a character as the path from the root to that character where '0' means 'go to the left child', and 1 means 'go to the right child'.

The above tree is not binary search tree, since the leaves need not appear in sorted order.

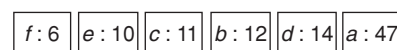
## Constructing Huffman Code

This algorithm builds the tree  $T$  corresponding to the optimal code in a bottom-up manner. It begins with set of  $|C|$  leaves and performs a sequence of  $|C|-1$  'merging' operations to create the final tree.

- A min-priority queue  $Q$ , keyed on  $f$ , is used to identify the 2 least – frequent objects to merge together. The result of the merger of 2 objects is a new object whose frequency is the sum of the frequencies of the 2 objects that were merged.
- In the given example, there are 6 alphabets the initial queue size is  $n = 6$ , and 5 merge steps are required to build the tree. The final tree represents the optimal prefix code. The code word for a letter is the sequence of edge labels on the path from the root to the letter.

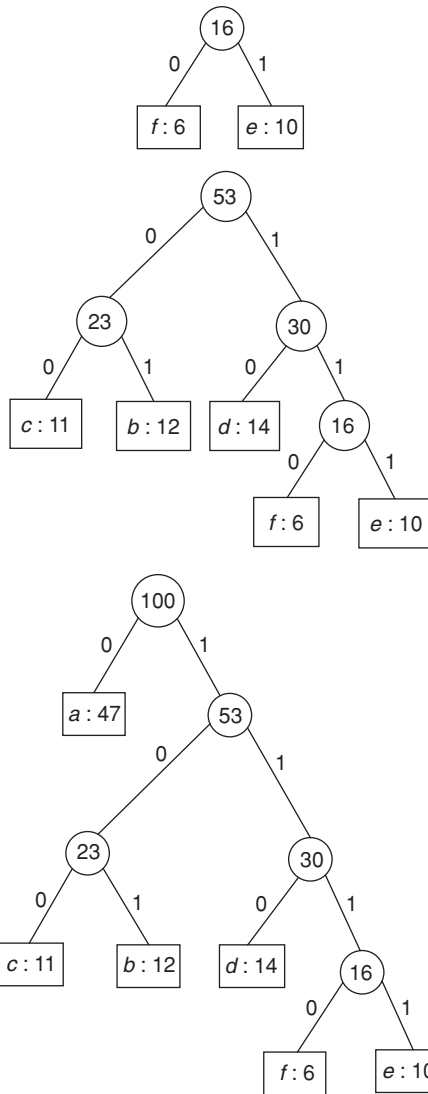
$a = 47, b = 12, c = 11, d = 14, e = 10, f = 6$

**Step I:** Arrange the characters in non-decreasing order according to their frequencies





Let  $x$  and  $y$  be 2 characters in  $C$  having the lowest frequencies. Then there exists an optimal prefix code for  $C$  in which the code words for  $x$  and  $y$  have the same length and differ only in the last bit



**Analysis:** The analysis of the running time of Huffman's algorithm assumes that  $Q$  is implemented as a binary min-heap for a set  $C$  of ' $n$ ' characters, the initialization of  $Q$  can be performed in  $O(n)$  time using the BUILD – MIN HEAP procedure.

Each heap operation requires  $O(\log n)$  time, and this will be performed exactly  $(n - 1)$  times, it contributes to  $O(n \log n)$  running time. Thus the total running time of HUFFMAN on a set of ' $n$ ' characters is  $O(n \log n)$ .

## TASK-SCHEDULING PROBLEM

This is the problem of optimally scheduling unit – time tasks on a single processor, where each task has a deadline, along with a penalty that must be paid if the deadline is missed. The problem looks complicated, but it can be solved in simple manner using a greedy algorithm.

- A unit – time task is a job, such as a program to be run on a computer, that requires exactly one unit of time to complete.
- Given a finite set  $S$  of unit – time tasks, a schedule for  $S$  is a permutation of  $S$  specifying the order in which these tasks are to be performed.
- The first task in the schedule begins at time '0' and finishes at time 1, the second task begins at time 1 and finishes at time 2, and so on
- The problem of scheduling unit – time tasks with deadlines and penalties for a single processor has the following inputs:

1. A set  $S = \{a_1, a_2, \dots, a_n\}$  of  $n$  unit – time tasks:
2. A set of  $n$  integer deadlines  $d_1, d_2, \dots, d_n$  such that each  $d_i$  satisfies  $1 \leq d_i \leq n$  and task  $a_i$  is supposed to finish by time  $d_i$ .
3. A set of  $n$  non-negative weights or penalties  $w_1, w_2, \dots, w_n$ , such that we incur a penalty of  $w_i$  if task  $a_i$  is not finished by time  $d_i$  and we incur no penalty if a task finishes by its deadline.

**Example:** Consider the following 7 tasks,  $T_1, T_2, T_3, T_4, T_5, T_6, T_7$ . Every task is associated with profit and deadline.

Tasks	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$
Deadline	4	2	4	3	1	4	5
Profit	75	65	55	45	40	35	30

45	65	55	75	30			
$T_4$	$T_2$	$T_3$	$T_1$	$T_7$			
0	1	2	3	4	5	6	7

$T_1$  has highest profit, so it will be executed first and the deadline of  $T_1$ , is '4' so  $T_1$  has to be executed within 4 slots of time, same procedure is applied to other tasks also.

The tasks which are not executed by CPU are  $T_5$  and  $T_6$ .

**Profit:** sum up the profits made by executing the tasks. Profit =  $45 + 65 + 55 + 75 + 30 = 270$

**Analysis:** We can use a greedy algorithm to find a maximum weight independent set of tasks. We can then create an optimal schedule having the tasks in A as its early tasks.

This method is an efficient algorithm for scheduling unit – time tasks with deadlines and penalties for a single processor. The running time is  $O(n^2)$  using GREEDY METHOD, since each of the  $O(n)$  independent checks made by that algorithm takes time  $O(n)$ .

## SORTING AND ORDER STATISTICS

### Minimum and Maximum

This algorithm determines, how many comparisons are necessary to find minimum or maximum of a set of ' $n$ ' elements. Usually we can obtain maximum or minimum, by performing

$(n - 1)$  comparisons; examine each element of the set in turn and keep track of the smallest element seen so far.

Consider the following procedure.

Assume that the set of elements reside in an array  $A$  where  $\text{length}[A] = n$

MINIMUM ( $A$ )

Min  $\leftarrow A[1]$

For  $i \leftarrow 2$  to  $\text{length}[A]$

Do if  $\text{min} > A[i]$

Then  $\text{min} \leftarrow A[i]$

Return min.

### Simultaneous minimum and maximum

In some applications, we must find both the minimum and the maximum of a set of ' $n$ ' elements.

We can find the minimum and maximum independently using  $(n - 1)$  comparisons for each, for a total of  $(2n - 2)$  comparisons.

- In fact, atmost  $3\lfloor n/2 \rfloor$  comparisons are sufficient to find both the minimum and the maximum.
- The strategy is to maintain the minimum and maximum elements seen so far.
- Rather than processing each element of the input by comparing it against the current minimum and maximum at a cost of 2 comparisons per element, we process elements in pairs.
- Compare pairs of elements from the input with each other, and then we compare smaller to the current minimum and the larger to the current maximum, at a cost of 3 comparisons for every 2 elements.
- Setting up initial values for the current minimum and maximum depends on whether ' $n$ ' is odd or even. If ' $n$ ' is odd, we set both the minimum and maximum to the value of the first element, and then we process the rest of the elements in pairs.
- If ' $n$ ' is even, we perform 1 comparison on the first 2 elements to determine the initial values of the minimum and maximum and then process the rest of the elements in pairs.

**Analysis:** If ' $n$ ' is odd the total number of comparisons would be  $3\lfloor n/2 \rfloor$ .

If ' $n$ ' is even, we need 1 initial comparison followed by  $\frac{3(n-2)}{2}$  comparisons, for a total of  $\frac{3n}{2} - 2$ .

$\therefore$  The total number of comparisons is atmost  $3\lfloor n/2 \rfloor$

## GRAPH ALGORITHMS

### Single Source Shortest Path

In a shortest-path problem, we are given a weighted directed graph  $G = (V, E)$  with weight function  $W: E \rightarrow R$  mapping edges to real-valued weights. The weight of path  $P = \langle V_o, V_1 \dots V_k \rangle$  is the sum of the weights of its constituent edges. Shortest-path weight from  $U$  to  $V$  is defined by

$$\delta(U, V) = \begin{cases} \min\{W(P) : u \rightarrow v\} & \text{if there is a path from 'U' to 'V' otherwise} \\ \infty & \end{cases}$$

Edge weights can be interpreted as metrics other than distances. They are often used to represent time, cost, penalties, loss, or any other quantity.

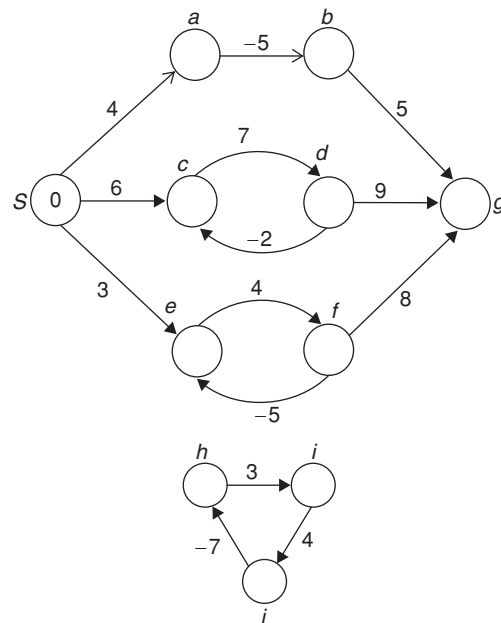
- The breadth first search algorithm is a shortest-path algorithm that works on un weighted graphs, that is, graphs in which each edge can be considered to have unit weight.

### Negative-weight edges

Some of the instances of the single-source-shortest-paths problem, there may be edges whose weights are negative.

- If the graph  $G = (V, E)$  contains no negative weight cycles reachable from source  $S$ , then for all  $v \in V$ , the shortest – path weight  $d(S, V)$  remains well defined, even if it has a negative value.
- If there is a negative-weight cycle reachable from  $S$ , shortest-path weights are not well defined.
- No path from ' $S$ ' to a vertex on the cycle can be a shortest path - a lesser weight path can always be found that follows the proposed 'shortest' path and then traverses the negative-weight cycle.
- If there is a negative-weight cycle on some path form ' $S$ ' to ' $V$ ', we define  $\delta(S, V) = -\infty$ .

**Example:** Consider the following graph, calculate the shortest distance to all vertices from sources ' $S$ '.



**Solution:**

- Shortest path from  $S$  to  $a$  is  $\delta(S, a) = W(S, a) = 4$  (because there is only one path from ' $S$ ' to ' $a$ ') )
- Similarly, there is only one path from ' $s$ ' to ' $b$ '  
 $\delta(S, a) = W(S, a) + W(a, b) = 4 + (-5) = -1$
- Shortest-path from ' $s$ ' to ' $c$ '



There are infinitely many paths from 'S' to 'c'

1.  $\langle S, c \rangle$
2.  $\langle S, c, d, c \rangle$
3.  $\langle S, c, d, c, d, c \rangle$  and so on  
 $\delta \langle S, c \rangle = 6$   
 $\delta \langle S, d, d, c \rangle = 6 + 7 - 2 = 11$   
 $\delta \langle S, c, d, c, d, c \rangle = 6 + 7 - 2 + 7 - 2 = 16$   
 $\delta \langle S, c, d, c, d, c, d, c \rangle$   
 $= 6 + 7 - 2 + 7 - 2 + 7 - 2 = 21$

The cycle  $\langle c, d, c \rangle$  has weight  $= 7 + (-2) = 5 > 0$

The shortest path from 'S' to 'c' is  $\langle s, c \rangle$  with weight  $\delta(S, c) = 6$  similarly, the shortest-path from 'S' to 'd' is  $\langle s, c, d \rangle$ , with weight  $\delta(S, d) = w(S, c) + W(c, d) = 13$

May there are infinitely paths from 'S' to 'e'

1.  $\langle s, e \rangle$
2.  $\langle s, e, f, e \rangle$
3.  $\langle s, e, f, e, f, e \rangle$  and so on

Since the cycle  $\langle e, f, e \rangle$  has weight  $4 + (-5) = -1 < 0$ . However, there is no shortest path from 'S' to 'e' by traversing the negative-weight cycle  $\langle e, f, e \rangle$  arbitrarily many times, we can find paths from 's' to 'e' with arbitrarily large negative weights,

So  $\delta(S, e) = -\infty$

Similarly,  $\delta(S, f) = -\infty$

- The shortest path from 'S' to 'g':  
'g' can be reachable from 'f'; we can also find paths with arbitrarily large negative weights from 's' to 'g' and  $\delta(s, g) = -\infty$
- Vertices 'h', 'i' and 'j' also form a negative - weight cycle. They are not reachable from 'S' so,  $\delta(S, h) = \delta(S, i) = \delta(S, j) = \infty$

### Dijkstra's Algorithm

Dijkstra's algorithm solves the single-source shortest-path problem on a weighted, directed graph  $G = (V, E)$ , for the case in which all edge weights are non-negative.

- The running time of Dijkstra's algorithm is lower than that of the Bellman-Ford algorithm.
- Dijkstra's algorithm maintains a set 's' of vertices whose final shortest-path weights from the source 'S' have already been determined.
- The algorithm repeatedly selects the vertex  $u \in (V - S)$  with the minimum shortest-path estimate, adds 'u' to 'S'

DIJKSTRA (G, W, S)

INITIALIZE - SINGLE - SOURCE (G, S)

$S \leftarrow \emptyset$

$S \leftarrow V[G]$

While  $Q \neq \emptyset$

do  $u \leftarrow \text{EXTRACT} - \text{MIN}(Q)$

$S \leftarrow S \cup \{u\}$

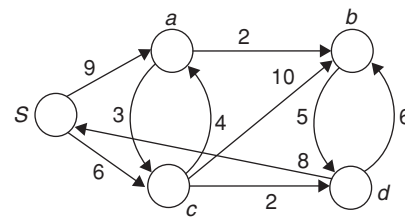
For each vertex  $v \in \text{Adj}[u]$

do RELAX (u, v, w)

The algorithm maintains the invariant that  $Q = V - S$  at the start of each iteration of the while loop. Initially the min - priority queue  $Q$  contains all the vertices in  $V$ . ( $\because S = \emptyset$ ). Each time through the while loop, a vertex 'u' is extracted from  $Q = V - S$  and added to set  $S$ .

- Each vertex is extracted from  $Q$  and added to  $S$  exactly once, so the contents of while loop will be executed exactly  $|V|$  times.
- Dijkstra's algorithm always chooses the 'closest' or 'lightest' vertex in  $(V - S)$  to add to set  $S$ , we say that it uses a greedy strategy.

**Example:** Consider the following graph, what is the shortest path?



**Solution:**

S	V - S
S	a b c d
S c	a b d
S c d	a b
S c d a	b
S c d a b	$\emptyset$

Distance from S to all vertices of  $(V - S)$

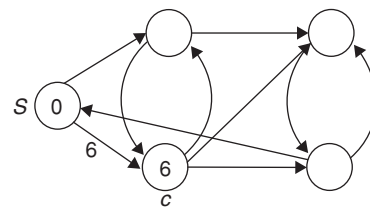
$$d[a] = 9$$

$$d[b] = \infty$$

$$d[c] = 6$$

$$d[d] = \infty$$

9,  $\infty$ , 6,  $\infty$  values are given to MIN-PRIORITY Queue 'Q', '6' is returned.

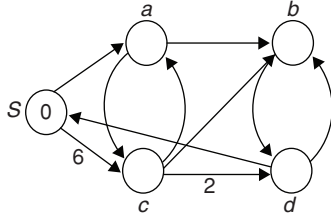


Distance from [Sc] to all vertices of  $(V - S)$

$$d[b] = (S - c - b) = 6 + 10 = 16$$

$$d[a] = \min\{(S - a) = 9, (s - c - a) = 10\} = 9$$

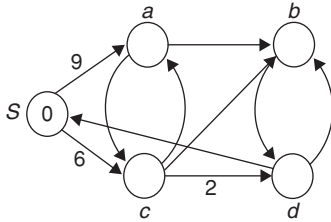
$$d[d] = \min\{\infty, (S - c - d) = 6 + 2 = 8\} = 8$$



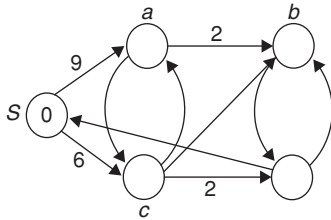
Distance from  $[s \ c \ d]$  to  $[ab]$

$$d[a] = \min\{9, (S - c - d - S - a) = 25\} = 9$$

$$d[b] = \min\{16, (S - c - d - b) = 14\} = 14$$



$$d[a] = \min\{14, (s - a - b) = 9 + 2 = 11\} = 11$$



**Analysis:** It maintains the min-priority queue 'Q' by calling three priority-queue operations: INSERT, EXTRACT-MIN, and DECREASE-KEY. We maintain the min-priority queue by taking the vertices being numbered 1 to  $|V|$ . We store  $d[v]$  in the  $v$ th entry of an array. Each INSERT and DECREASE-KEY operation takes  $O(1)$  time, and each EXTRACT-MIN operation takes  $O(v)$  time ( $\therefore$  we have to search through the entire array) for a total time of  $O(v^2 + E) = O(v^2)$ .

- If we implement the min - priority queue with a binary min-heap. Each EXTRACT-MIN operation takes time  $O(\log V)$ , there are  $|V|$  such operations.
- The time to build the binary min-heap is  $O(v)$ . Each DECREASE-KEY operation takes time  $O(\log V)$ , and there are still atmost  $|E|$  such operations. The total running time is  $O((V + E) \log V)$ , which is  $O(E \log V)$  if all vertices are reachable from the source.
- We can achieve a running time of  $O(V \log V + E)$  by implementing the min-priority queue with a Fibonacci heap.

### Bellman–Ford Algorithm

Bellman–Ford algorithm solves the single-source shortest-path problems in the case in which edge weights may be negative.

- When negative edge lengths are permitted, we require that the graph have no cycles of negative length. This is

necessary to ensure that shortest-paths consist of a finite number of edges.

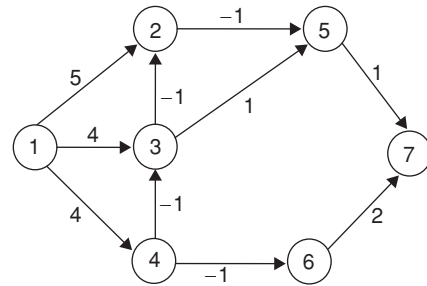
- When there are no cycles of negative length, there is a shortest-path between any two vertices of an  $n$ -vertex graph that has atmost  $(n - 1)$  edges on it.
- A path that has more than  $(n - 1)$  edges must repeat atleast one vertex and hence must contain a cycle
- Let  $\text{dist}^k[u]$  be the length of a shortest-path from the source vertex 'v' to vertex 'u' under the constraint that the shortest-path contains atmost 'x' edges. Then  $\text{dist}^k[u] = \text{cost}[v, u]$   $1 \leq u \leq n$  when there are no cycles of negative length we can limit our search for shortest-paths to paths with at most  $(n - 1)$  edges. Hence,  $\text{dist}^{n-1}[u]$  is the length of an unrestricted shortest-path from 'v' to 'u'.

The Recurrence Relation for  $\text{dist}$  is:

$$\text{Dist}^k[u] = \min\{\text{dist}^{k-1}[u], \min\{\text{dist}^{k-1}[i] + \text{cost}[i, u]\}\}$$

This recurrence can be used to compute  $\text{dist}^k$  from  $\text{dist}^{k-1}$ , for  $k = 2, 3, \dots, n - 1$ .

**Example:** Consider the given directed graph



Find the shortest path from vertex '1' to all other vertices using Bellman–Ford algorithm?

**Solution:** Source vertex is '1' the distance from '1' to '1' in all '6' iterations will be zero. Since the graph has '7' vertices, the shortest-path can have atmost '6' edges. The following figure illustrates the implementation of Bellman–Ford algorithm:

	1	2	3	4	5	6	7
1	0	5	4	4	$\infty$	$\infty$	$\infty$
2	0	3	3	4	4	3	$\infty$
3	0	2	3	4	2	3	5
4	0	2	3	4	1	3	3
5	0	2	3	4	1	3	2
6	0	2	3	4	1	3	2
7	0	2	3	4	1	3	2

### Analysis

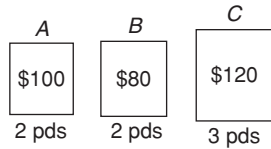
- Each iteration takes  $O(n^2)$  time if adjacency matrices are used and  $O(e)$  time if adjacency lists are used. Here 'e' is the number of edges in the graph.
- The time complexity is  $O(n^3)$  when adjacency matrices are used and  $O(N * E)$  when adjacency lists are used.

## EXERCISES

## Practice Problems I

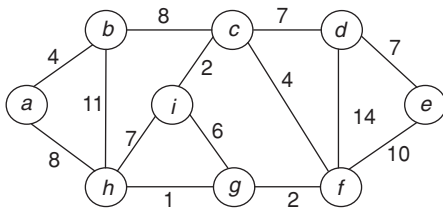
**Directions for questions 1 to 14:** Select the correct alternative from the given choices.

1. A thief enters a store and sees the following:



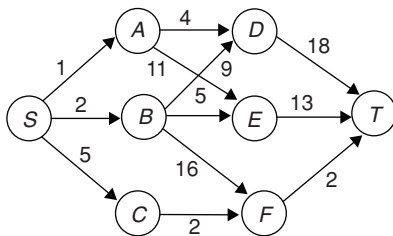
His knapsack can hold 4 pounds, what should he steal to maximize profit? (Use 0–1 Knapsack).

- (A) A and B (B) A and C  
(C) B and C (D) A, B and C
2. By using fractional Knapsack, calculate the maximum profit, for the data given in the above question?  
(A) 180 (B) 170  
(C) 160 (D) 150
3. Consider the below figure:



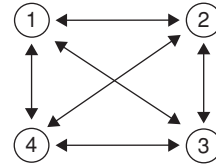
What is the weight of the minimum spanning tree using Kruskal's algorithm?

- (A) 34 (B) 35  
(C) 36 (D) 38
4. Construct a minimum spanning tree for the figure given in the above question, using prim's algorithm. What are the first three nodes, added to the solution set respectively (consider 'a' as starting node).  
(A) b, c, i (B) h, b, c  
(C) c, i, b (D) h, c, b
5. Consider the below graph, calculate the shortest distance from 'S' to 'T'?



- (A) 23 (B) 9  
(C) 20 (D) 22
6. Solve the travelling salesman problem, with the given distances in the form of matrix of graph, which of the following gives optimal solution?

C	1	2	3	4
1	0	15	20	25
2	10	0	14	15
3	11	18	0	17
4	13	13	14	0



- (A) 1 – 2 – 4 – 3 – 1 (B) 2 – 3 – 4 – 1 – 2  
(C) 1 – 4 – 2 – 3 – 1 (D) 2 – 4 – 3 – 1 – 2

7. Calculate the maximum profit using greedy strategy, knapsack capacity is 50. The data is given below:

$$n = 3$$

$$(w_1, w_2, w_3) = (10, 20, 30)$$

$$(p_1, p_2, p_3) = (60, 100, 120) \text{ (dollars)? (0/1 knapsack)}$$

- (A) 180 (B) 220  
(C) 240 (D) 260

**Common data for questions 8 and 9:** Given that

	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Fixed length code word	000	001	010	011	100	101

8. Using Huffman code, find the path length of internal nodes.

- (A) 8 (B) 100  
(C)  $100 \times 8$  (D)  $100/8$

9. Using above answer, external path length will be

- (A) 18 (B) 108  
(C) 8 (D) None of these

**Common data for questions 10 and 11:**

10.



Using 0–1 knapsack select a subset of the three items shown, whose weight must not exceed 50 kg. What is the value?

- (A) 2220 (B) 2100  
(C) 2600 (D) 2180

11. Which of the following gives maximum profit, using fractional knapsack?

- (A)  $x_1 = 1, x_2 = 1, x_3 = 0$  (B)  $x_1 = 1, x_2 = 1, x_3 = 2/3$   
(C)  $x_1 = 1, x_2 = 0, x_3 = 1$  (D)  $x_1 = 1, x_2 = 1, x_3 = 1/3$

12. Using dynamic programming find the longest common subsequence (LCS) in the given 2 sub sequences:

$x[1, \dots, m]$

$y[1, \dots, n]$

$x : A B C B D A B$

$y : B D C A B A$

Find longest sequence sets common to both.

- (A) (BDAB, BCAB, BCBA)  
 (B) (BADB, BCAB, BCBA)  
 (C) (BDAB, BACB, BCBA)  
 (D) (BDAB, BCAB, BBBA)
13. Let  $C_1, C_2, C_3, C_4$  represent coins.  
 $C_1 = 25$  paisa

$C_2 = 10$  paisa

$C_3 = 5$  paisa

$C_4 = 1$  paisa

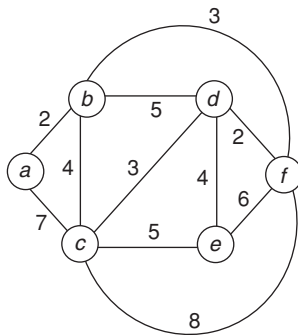
To represent 48 paisa, what is the minimum number of coins used, using greedy approach?

- (A) 6 (B) 7  
 (C) 8 (D) 9
14. Worst-case analysis of hashing occurs when
- (A) All the keys are distributed  
 (B) Every key hash to the same slot  
 (C) Key values with even number, hashes to slots with even number  
 (D) Key values with odd number hashes to slots with odd number.

## Practice Problems 2

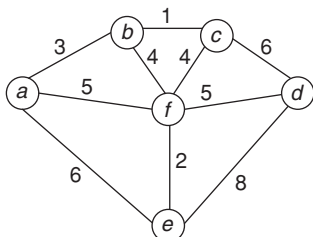
**Directions for questions 1 to 15:** Select the correct alternative from the given choices.

1. Consider the given graph:

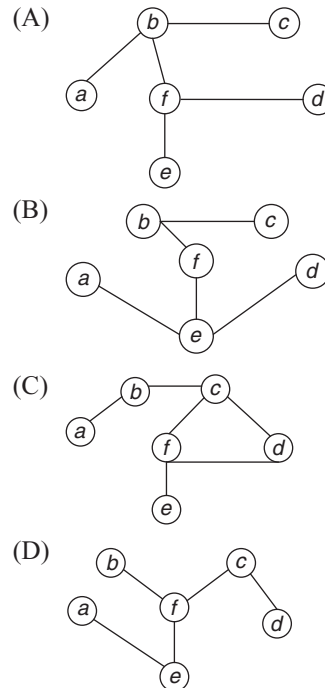


Which one of the following cannot be the sequence of edges added, in that order, to a minimum spanning tree using Kruskal's algorithm?

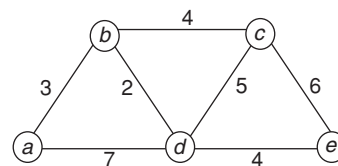
- (A)  $(a-b), (d-f), (b-f), (d-c), (d-e)$   
 (B)  $(a-b), (d-f), (d-c), (b-f), (d-e)$   
 (C)  $(d-f), (a-b), (d-c), (b-f), (d-e)$   
 (D)  $(d-f), (a-b), (b-f), (d-e), (d-c)$
2. The worst case height analysis of B-tree is
- (A)  $O(n)$   
 (B)  $O(n^2)$   
 (C)  $O(\log n)$   
 (D)  $O(n \log n)$
3. Consider the given graph:



Which of the following is the minimum spanning tree. (If we apply Kruskal algorithm).



4. Consider the following graph:



Find the shortest path using Dijkstra's algorithm.

- (A)  $a-b-d-e$  (B)  $a-b-c-d$   
 (C)  $a-c-d-e$  (D)  $a-b-c-e$
5. Which statement is true about Kruskal's algorithm?
- (A) It is a greedy algorithm for the minimum spanning tree problem.  
 (B) It constructs spanning tree by selecting edges in increasing order of their weights.  
 (C) It does not accept creation of cycles in spanning tree.  
 (D) All the above

6. Dijkstra's algorithm bears similarity to which of the following for computing minimum spanning trees?  
 (A) Breadth first search (B) Prim's algorithm  
 (C) Both (A) and (B) (D) None of these
7. Which of the following algorithm always yields a correct solution for a graph with non-negative weights to compute shortest paths?  
 (A) Prim's algorithm (B) Kruskal's algorithm  
 (C) Dijkstra's algorithm (D) Huffman tree
8. Let the load factor of the hash table is number of keys is  $n$ , cells of the hash table is  $m$  then  
 (A)  $\infty = n/m$  (B)  $\infty = m/n$   
 (C)  $\propto \frac{m+1}{n}$  (D)  $\propto \frac{n+1}{m}$
9. To implement Dijkstra's shortest path algorithm on unweighted graphs so that it runs in linear time, the data structure to be used is:  
 (A) Queue  
 (B) Stack  
 (C) Heap  
 (D) B-tree
10. The development of a dynamic-programming algorithm can be broken into a sequence of four steps, which are given below randomly.  
 I. Construct an optimal solution from computed information.  
 II. Compute the value of an optimal solution in a bottom-up fashion.  
 III. Characterize the structure of an optimal solution.  
 IV. Recursively defines the value of an optimal solution.  
 The correct sequence of the above steps is  
 (A) I, II, III, IV (B) IV, III, I, II  
 (C) IV, II, I, III (D) III, IV, II, I
11. Let  $V$  stands for vertex,  $E$  stands for edges.  
 For both directed and undirected graphs, the adjacency list representation has the desirable property that the amount of memory required is  
 (A)  $\theta(V)$  (B)  $\theta(E)$   
 (C)  $\theta(V + E)$  (D)  $\theta(V - E)$
12. Which of the following is false?  
 (A) Adjacency-matrix representation of a graph permits faster edge look up.  
 (B) The adjacency matrix of a graph requires  $\theta(v^2)$  memory, independent of the number of edges in the graph.  
 (C) Adjacency-matrix representation can be used for weighted graphs.  
 (D) All the above
13. Dynamic programming is a technique for solving problems with  
 (A) Overlapped sub problems  
 (B) Huge size sub problems  
 (C) Small size sub problems  
 (D) None of these
14. The way a card game player arranges his cards, as he picks them up one by one is an example of \_\_\_\_\_.  
 (A) Bubble sort (B) Selection sort  
 (C) Insertion sort (D) None of the above
15. You want to check whether a given set of items is sorted. Which method will be the most efficient if it is already in sorted order?  
 (A) Heap sort (B) Bubble sort  
 (C) Merge sort (D) Insertion sort

### PREVIOUS YEARS' QUESTIONS

**Data for question 1:** We are given 9 tasks  $T_1, T_2 \dots T_9$ . The execution of each task requires one unit of time. We can execute one task at a time. Each task  $T_i$  has a profit  $P_i$  and a deadline  $D_i$ . Profit  $P_i$  is earned if the task is completed before the end of the  $D_i$ th unit of time.

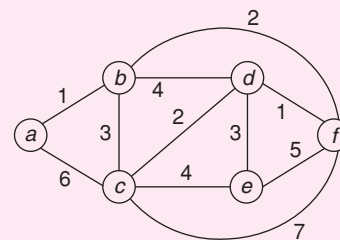
Task	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_8$	$T_9$
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

1. What is the maximum profit earned? [2005]  
 (A) 147 (B) 165  
 (C) 167 (D) 175
2. Consider a weighted complete graph  $G$  on the vertex set  $\{v_1, v_2, \dots, v_n\}$  such that the weight of the edge  $(v_i, v_j)$  is  $2|i - j|$ . The weight of the minimum spanning tree is: [2006]  
 (A)  $n - 1$  (B)  $2n - 2$   
 (C)  $\binom{n}{2}$  (D)  $n^2$

3. To implement Dijkstra's shortest path algorithm on unweighted graphs so that it runs in linear time, the data structure to be used is: [2006]

- (A) Queue  
 (B) Stack  
 (C) Heap  
 (D) B-Tree

4. Consider the following graph: [2006]



Which one of the following cannot be the sequence of edges added, in that order, to a minimum spanning tree using Kruskal's algorithm?

- (A)  $(a - b), (d - f), (b - f), (d - c), (d - e)$   
 (B)  $(a - b), (d - f), (d - c), (b - f), (d - e)$   
 (C)  $(d - f), (a - b), (d - c), (b - f), (d - e)$   
 (D)  $(d - f), (a - b), (b - f), (d - e), (d - c)$

**Common data for questions 5 and 6:** A 3-ary max-heap is like a binary max-heap, but instead of 2 children, nodes have 3 children. A 3-ary heap can be represented by an array as follows: The root is stored in the first location,  $a[0]$ , nodes in the next level, from left to right, is stored from  $a[1]$  to  $a[3]$ . The nodes from the second level of the tree from left to right are stored from  $a[4]$  location onward. An item  $x$  can be inserted into a 3-ary heap containing  $n$  items by placing  $x$  in the location  $a[n]$  and pushing it up the tree to satisfy the heap property.

5. Which one of the following is a valid sequence of elements in an array representing 3-ary max-heap? [2006]

- (A) 1, 3, 5, 6, 8, 9                      (B) 9, 6, 3, 1, 8, 5  
 (C) 9, 3, 6, 8, 5, 1                      (D) 9, 5, 6, 8, 3, 1

6. Suppose the elements 7, 2, 10 and 4 are inserted, in that order, into the valid 3-ary max-heap found in the above question, Q-76. Which one of the following is the sequence of items in the array representing the resultant heap? [2006]

- (A) 10, 7, 9, 8, 3, 1, 5, 2, 6, 4  
 (B) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1  
 (C) 10, 9, 4, 5, 7, 6, 8, 2, 1, 3  
 (D) 10, 8, 6, 9, 7, 2, 3, 4, 1, 5

7. In an unweighted, undirected connected graph, the shortest path from a node  $S$  to every other node is computed most efficiently, in terms of *time complexity*, by [2007]

- (A) Dijkstra's algorithm starting from  $S$ .  
 (B) Warshall's algorithm  
 (C) Performing a DFS starting from  $S$ .  
 (D) Performing a BFS starting from  $S$ .

8. A complete  $n$ -ary tree is a tree in which each node has  $n$  children or no children. Let  $I$  be the number of internal nodes and  $L$  be the number of leaves in a complete  $n$ -ary tree. If  $L = 41$ , and  $I = 10$ , what is the value of  $n$ ? [2007]

- (A) 3    (B) 4  
 (C) 5    (D) 6

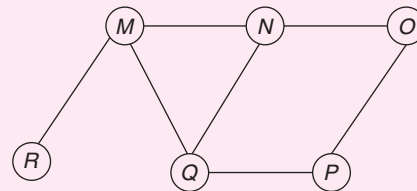
9. Consider the following C program segment where CellNode represents a node in a binary tree: [2007]

```
struct CellNode {
    struct CellNode *leftChild;
    int element;
    struct CellNode *rightChild;
};
```

```
int GetValue (struct CellNode *ptr) {
    int value = 0;
    if (ptr != NULL) {
        if ((ptr->leftChild == NULL) &&
            (ptr->rightChild == NULL))
            value = 1;
        else
            value = value + GetValue(ptr->leftChild)
                + GetValue(ptr->rightChild);
    }
    return (value);
}
```

The value returned by GetValue when a pointer to the root of a binary tree is passed as its argument is:

- (A) The number of nodes in the tree  
 (B) The number of internal nodes in the tree  
 (C) The number of leaf nodes in the tree  
 (D) The height of the tree
10. Let  $w$  be the minimum weight among all edge weights in an undirected connected graph. Let  $e$  be a specific edge of weight  $w$ . Which of the following is FALSE? [2007]
- (A) There is a minimum spanning tree containing  $e$ .  
 (B) If  $e$  is not in a minimum spanning tree  $T$ , then in the cycle formed by adding  $e$  to  $T$ , all edges have the same weight.  
 (C) Every minimum spanning tree has an edge of weight  $w$ .  
 (D)  $e$  is present in every minimum spanning tree.
11. The Breadth first search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is [2008]

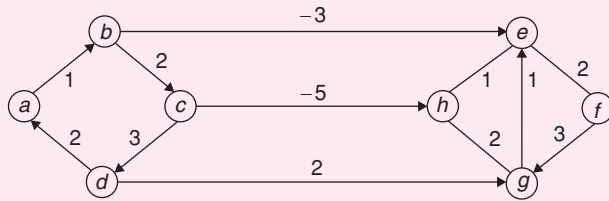


- (A) MNOPQR                                      (B) NQMPOR  
 (C) QMNPOR                                      (D) QMNPOR
12.  $G$  is a graph on  $n$  vertices and  $2n - 2$  edges. The edges of  $G$  can be partitioned into two edge-disjoint spanning trees. Which of the following is NOT true for  $G$ ? [2008]

- (A) For every subset of  $k$  vertices, the induced subgraph has at most  $2k - 2$  edges  
 (B) The minimum cut in  $G$  has at least two edges  
 (C) There are two edge-disjoint paths between every pair of vertices  
 (D) There are two vertex-disjoint paths between every pair of vertices



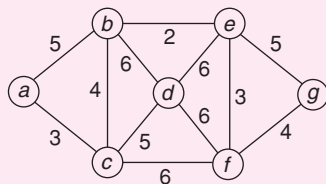
13.



Dijkstra's single source shortest path algorithm when run from vertex  $a$  in the above graph, computes the correct shortest path distance to

[2008]

- (A) Only vertex  $a$   
 (B) Only vertices  $a, e, f, g, h$   
 (C) Only vertices  $a, b, c, d$   
 (D) all the vertices
14. You are given the post-order traversal,  $P$ , of a binary search tree on the  $n$  elements  $1, 2, \dots, n$ . You have to determine the unique binary search tree that has  $P$  as its post-order traversal. What is the time complexity of the most efficient algorithm for doing this? [2008]  
 (A)  $\Theta(\log n)$   
 (B)  $\Theta(n)$   
 (C)  $\Theta(n \log n)$   
 (D) None of the above, as the tree cannot be uniquely determined
15. Which of the following statement(s) is/are correct regarding Bellman–Ford shortest path algorithm? [2009]  
 P. Always finds a negative weighted cycle, if one exists.  
 Q. Finds whether any negative weighted cycle is reachable from the source.  
 (A) P only (B) Q only  
 (C) Both P and Q (D) Neither P nor Q
16. Consider the following graph: [2009]



Which one of the following is NOT the sequence of edges added to the minimum spanning tree using Kruskal's algorithm? [2009]

- (A)  $(b, e) (e, f) (a, c) (b, c) (f, g) (c, d)$   
 (B)  $(b, e) (e, f) (a, c) (f, g) (b, c) (c, d)$   
 (C)  $(b, e) (a, c) (e, f) (b, c) (f, g) (c, d)$   
 (D)  $(b, e) (e, f) (b, c) (a, c) (f, g) (c, d)$
- Common data for questions 17 and 18:** Consider a binary max-heap implemented using an array.
17. Which one of the following array represents a binary max-heap? [2009]

- (A)  $\{25, 12, 16, 13, 10, 8, 14\}$   
 (B)  $\{25, 14, 13, 16, 10, 8, 12\}$   
 (C)  $\{25, 14, 16, 13, 10, 8, 12\}$   
 (D)  $\{25, 14, 12, 13, 10, 8, 16\}$

18. What is the content of the array after two delete operations on the correct answer to the previous question?

[2009]

- (A)  $\{14, 13, 12, 10, 8\}$  (B)  $\{14, 12, 13, 8, 10\}$   
 (C)  $\{14, 13, 8, 12, 10\}$  (D)  $\{14, 13, 12, 8, 10\}$

**Common data for questions 19 and 20:** Consider a complete undirected graph with vertex set  $\{0, 1, 2, 3, 4\}$ . Entry  $W_{ij}$  in the matrix  $W$  below is the weight of the edge  $\{i, j\}$ .

$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$

19. What is the minimum possible weight of a spanning tree  $T$  in this graph such that vertex 0 is a leaf node in the tree  $T$ ? [2010]  
 (A) 7 (B) 8  
 (C) 9 (D) 10
20. What is the minimum possible weight of a path  $P$  from vertex 1 to vertex 2 in this graph such that  $P$  contains at most 3 edges? [2010]  
 (A) 7 (B) 8  
 (C) 9 (D) 10

**Common data for questions 21 and 22:** A hash table of length 10 uses open addressing with hash function  $h(k) = k \bmod 10$ , and linear probing. After inserting 6 values into an empty hash table, the table is as shown below:

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

21. Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

[2010]

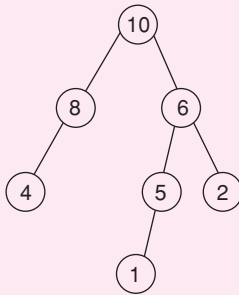
- (A) 46, 42, 34, 52, 23, 33  
 (B) 34, 42, 23, 52, 33, 46  
 (C) 46, 34, 42, 23, 52, 33  
 (D) 42, 46, 33, 23, 34, 52

22. How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table shown above? [2010]

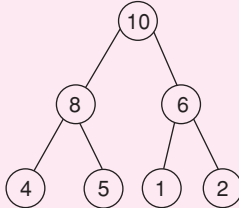
(A) 10 (B) 20  
(C) 30 (D) 40

23. A max-heap is a heap where the value of each parent is greater than or equal to the value of its children. Which of the following is a max-heap? [2011]

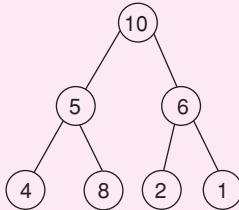
(A)



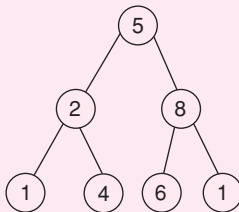
(B)



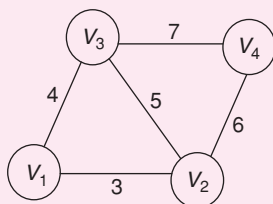
(C)



(D)



**Common data for questions 24 and 25:** An undirected graph  $G(V, E)$  contains  $n(n > 2)$  nodes named  $V_1, V_2, \dots, V_n$ . Two nodes  $V_i, V_j$  are connected if and only if  $0 < |i - j| \leq 2$ . Each edge  $(V_i, V_j)$  is assigned a weight  $i + j$ . A sample graph with  $n = 4$  is shown below.



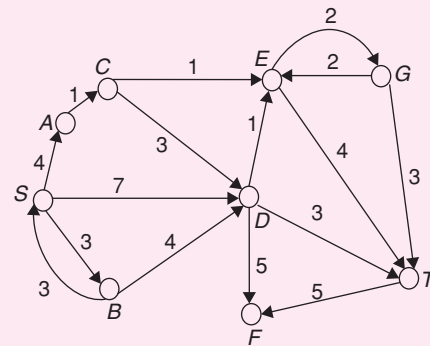
24. What will be the cost of the minimum spanning tree (MST) of such a graph with  $n$  nodes? [2011]

(A)  $\frac{1}{12}(11n^2 - 5n)$  (B)  $n^2 - n + 1$   
(C)  $6n - 11$  (D)  $2n + 1$

25. The length of the path from  $V_5$  to  $V_6$  in the MST of previous question with  $n = 10$  is [2011]

(A) 11 (B) 25  
(C) 31 (D) 41

26. Consider the directed graph shown in the figure below. There are multiple shortest paths between vertices  $S$  and  $T$ . Which one will be reported by Dijkstra's shortest path algorithm? Assume that, in any iteration, the shortest path to a vertex  $v$  is updated only when a strictly shorter path to  $v$  is discovered. [2012]



(A) SDT (B) SBDT  
(C) SACDT (D) SACET

27. Let  $G$  be a weighted graph with edge weights greater than one and  $G^1$  be the graph constructed by squaring the weights of edges in  $G$ . Let  $T$  and  $T^1$  be the minimum spanning trees of  $G$  and  $G^1$ , respectively, with total weights  $t$  and  $t^1$ . Which of the following statements is **TRUE**? [2012]

(A)  $T^1 = T$  with total weight  $t^1 = t^2$   
(B)  $T^1 = T$  with total weight  $t^1 < t^2$   
(C)  $T^1 \neq T$  but total weight  $t^1 = t^2$   
(D) None of the above

28. What is the time complexity of Bellman-Ford single-source shortest path algorithm on a complete graph of  $n$  vertices? [2013]

(A)  $\Theta(n^2)$  (B)  $\Theta(n^2 \log n)$   
(C)  $\Theta(n^3)$  (D)  $\Theta(n^3 \log n)$

29. Consider the following operation along with Enqueue and Dequeue operations on queues, where  $k$  is a global parameter.

```

MultiDequeue(Q) {
    m = k

```

```

while (Q is not empty) and (m > 0) {
    Dequeue (Q)
    m = m - 1
}

```

What is the worst case time complexity of a sequence of  $n$  queue operations on an initially empty queue?

[2013]

- (A)  $\Theta(n)$  (B)  $\Theta(n+k)$   
 (C)  $\Theta(nk)$  (D)  $\Theta(n^2)$

30. The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the post order traversal sequence of the same tree?

[2013]

- (A) 10, 20, 15, 23, 25, 35, 42, 39, 30  
 (B) 15, 10, 25, 23, 20, 42, 35, 39, 30  
 (C) 15, 20, 10, 23, 25, 42, 35, 39, 30  
 (D) 15, 10, 23, 25, 20, 35, 42, 39, 30

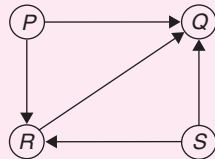
31. Let  $G$  be a graph with  $n$  vertices and  $m$  edges. What is the tightest upper bound on the running time of depth first search on  $G$ , when  $G$  is represented as an adjacency matrix?

[2014]

- (A)  $\theta(n)$  (B)  $\theta(n+m)$   
 (C)  $\theta(n^2)$  (D)  $\theta(m^2)$

32. Consider the directed graph given below.

[2014]



Which one of the following is TRUE?

- (A) The graph does not have any topological ordering.  
 (B) Both PQRS and SRQP are topological orderings.  
 (C) Both PSRQ and SPRQ are topological orderings.  
 (D) PSRQ is the only topological ordering.
33. There are 5 bags labeled 1 to 5. All the coins in a given bag have the same weight. Some bags have coins of weight 10 gm. Others have coins of weight 11 gm. I pick 1, 2, 4, 8, 16 coins respectively from bags 1 to 5. Their total weight comes out to 323 gm. Then the product of the labels of the bags having 11 gm coins is \_\_\_\_
34. A priority queue is implemented as a max-heap. Initially it has 5 elements. The level-order traversal of the heap is : 10, 8, 5, 3, 2. Two new elements 1 and 7 are inserted into the heap in that order. The level-order traversal of the heap after the insertion of the elements is

[2014]

- (A) 10, 8, 7, 3, 2, 1, 5 (B) 10, 8, 7, 2, 3, 1, 5  
 (C) 10, 8, 7, 1, 2, 3, 5 (D) 10, 8, 7, 5, 3, 2, 1

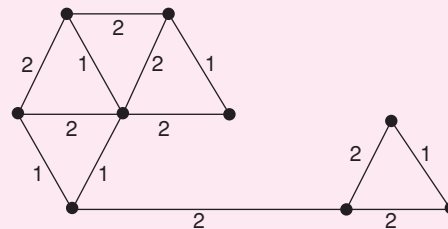
35. Consider the tree arcs of a BFS traversal from a source node  $W$  in an unweighted, connected, undirected graph. The tree  $T$  formed by the tree arcs is a data structure for computing

[2014]

- (A) The shortest path between every pair of vertices  
 (B) The shortest path from  $W$  to every vertex in the graph  
 (C) The shortest paths from  $W$  to only those nodes that are leaves of  $T$ .  
 (D) The longest path in the graph

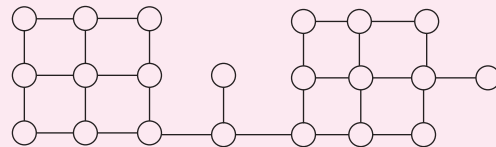
36. The number of distinct minimum spanning trees for the weighted graph below is \_\_\_\_.

[2014]



37. Suppose depth first search is executed on the graph below starting at some unknown vertex. Assume that a recursive call to visit a vertex is made only after first checking that the vertex has not been visited earlier. Then the maximum possible recursion depth (Including the initial call) is \_\_\_\_.

[2014]

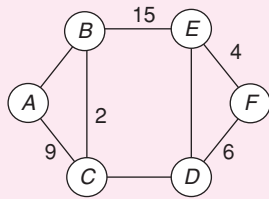


38. Suppose we have a balanced binary search tree  $T$  holding  $n$  numbers. We are given two numbers  $L$  and  $H$  and wish to sum up all the numbers in  $T$  that lie between  $L$  and  $H$ . suppose there are  $m$  such numbers in  $T$ . If the tightest upper bound on the time to compute the sum is  $O(n^a \log^b n + m^c \log^d n)$ , the value of  $a + 10b + 100c + 1000d$  is \_\_\_\_.

[2014]

39. The graph shown below has 8 edges with distinct integer edge weights. The minimum spanning tree (MST) is of weight 36 and contains the edges:  $\{(A, C), (B, C), (B, E), (E, F), (D, F)\}$ . The edge weights of only those edges which are in the MST are given in the figure shown below. The minimum possible sum of weights of all 8 edges of this graph is \_\_\_\_

[2015]



40. Consider two decision problems  $Q_1$ ,  $Q_2$  such that  $Q_1$  reduces in polynomial time to 3-SAT and 3-SAT reduces in polynomial time to  $Q_2$ . Then which one of the following is consistent with the above statement? [2015]

(A)  $Q_1$  is in NP,  $Q_2$  is NP hard.  
 (B)  $Q_2$  is in NP,  $Q_1$  is NP hard.  
 (C) Both  $Q_1$  and  $Q_2$  are in NP.  
 (D) Both  $Q_1$  and  $Q_2$  are NP hard.

41. Given below are some algorithms, and some algorithm design paradigms.

1. Dijkstra's Shortest Path	i. Divide and Conquer
2. Floyd-Warshall algorithm to compute all pairs shortest path	ii. Dynamic Programming
3. Binary search on a sorted array	iii. Greedy design
4. Backtracking search on a graph	iv. Depth-first search
	v. Breadth-first search

Match the above algorithms on the left to the corresponding design paradigm they follow.

(A) 1-i, 2-iii, 3-i, 4-v (B) 1-iii, 2-iii, 3-i, 4-v  
 (C) 1-iii, 2-ii, 3-i, 4-iv (D) 1-iii, 2-ii, 3-i, 4-v

42. A Young tableau is a 2D array of integers increasing from left to right and from top to bottom. Any unfilled entries are marked with  $\infty$ , and hence there cannot be any entry to the right of, or below a  $\infty$ . The following Young tableau consists of unique entries.

1	2	5	14
3	4	6	23
10	12	18	25
31	$\infty$	$\infty$	$\infty$

When an element is removed from a Young tableau, other elements should be moved into its place so that the resulting table is still a Young tableau (unfilled entries maybe filled in with a  $\infty$ ). The minimum number of entries (other than 1) to be shifted, to remove 1 from the given Young tableau is \_\_\_\_\_. [2015]

43. Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets

numbered 0 to 9 for  $i$  ranging from 0 to 2020?

[2015]

(A)  $h(i) = i^2 \bmod 10$   
 (B)  $h(i) = i^3 \bmod 10$   
 (C)  $h(i) = (11 * i^2) \bmod 10$   
 (D)  $h(i) = (12 * i) \bmod 10$

44. Let  $G$  be a weighted connected undirected graph with distinct positive edge weights. If every edge weight is increased by the same value, then which of the following statements is/are TRUE? [2016]

$P$  : Minimum spanning tree of  $G$  does not change.

$Q$  : Shortest path between any pair of vertices does not change.

(A)  $P$  only (B)  $Q$  only  
 (C) Neither  $P$  nor  $Q$  (D) Both  $P$  and  $Q$

45. Let  $G$  be a complete undirected graph on 4 vertices, having 6 edges with weights being 1,2,3,4,5, and 6. The maximum possible weight that a minimum weight spanning tree of  $G$  can have is \_\_\_\_\_. [2016]

46.  $G = (V, E)$  is an undirected simple graph in which each edge has a distinct weight, and  $e$  is a particular edge of  $G$ . Which of the following statements about the minimum spanning trees (MSTs) of  $G$  is/are TRUE? [2016]

I. If  $e$  is the lightest edge of some cycle in  $G$ , then every MST of  $G$  includes  $e$

II. If  $e$  is the heaviest edge of some cycle in  $G$ , then every MST of  $G$  excludes  $e$

(A) I only (B) II only  
 (C) both I and II (D) neither I nor II

47. Breadth First Search (BFS) is started on a binary tree beginning from the root vertex. There is a vertex  $t$  at a distance four from the root. If  $t$  is the  $n$ -th vertex in this BFS traversal, then the maximum possible value of  $n$  is \_\_\_\_\_. [2016]

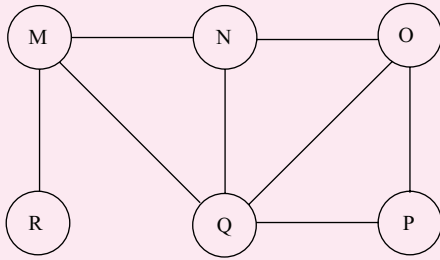
48. Let  $G = (V, E)$  be any connected undirected edge-weighted graph. The weights of the edges in  $E$  are positive and distinct. Consider the following statements:  
 (I) Minimum spanning Tree of  $G$  is always unique.  
 (II) Shortest path between any two vertices of  $G$  is always unique.

Which of the above statements is/are necessarily true?

[2017]

(A) (I) only  
 (B) (II) only  
 (C) both (I) and (II)  
 (D) neither (I) nor (II)

49. The Breadth First Search (BFS) algorithm has been implemented using the queue data structure. Which one of the following is a possible order of visiting the nodes in the graph below? [2017]



- (A) MNOPQR  
(B) NQMPOR  
(C) QMNROP  
(D) POQNMR

50. A message is made up entirely of characters from the set  $X = \{P, Q, R, S, T\}$ . The table of probabilities for each of the characters is shown below:

Character	Probability
P	0.22
Q	0.34
R	0.17
S	0.19
T	0.08
Total	1.00

If a message of 100 characters over  $X$  is encoded using Huffman coding, then the expected length of the encoded message in bits is \_\_\_\_\_. [2017]

51. Let  $G$  be a simple undirected graph. Let  $T_D$  be a depth first search tree of  $G$ . Let  $T_B$  be a breadth first search tree of  $G$ .

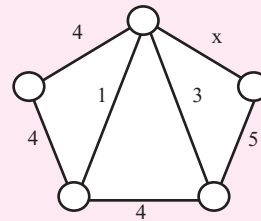
Consider the following statements.

- (I) No edge of  $G$  is a cross edge with respect to  $T_D$ .  
(A cross edge in  $G$  is between two nodes neither of which is an ancestor of the other in  $T_D$ .)  
(II) For every edge  $(u, v)$  of  $G$ , if  $u$  is at depth  $i$  and  $v$  is at depth  $j$  in  $T_B$ , then  $|i - j| = 1$ .

Which of the statements above must necessarily be true? [2018]

- (A) I only (B) II only  
(C) Both I and II (D) Neither I nor II

52. Consider the following undirected graph  $G$ :



Choose a value for  $x$  that will maximize the number of minimum weight spanning trees (MWSTs) of  $G$ . The number of MWSTs of  $G$  for this value of  $x$  is \_\_\_\_\_. [2018]

## ANSWER KEYS

### EXERCISES

#### Practice Problems 1

1. A    2. A    3. B    4. A    5. B    6. A    7. B    8. A    9. A    10. C  
11. B    12. A    13. A    14. B

#### Practice Problems 2

1. D    2. C    3. A    4. A    5. D    6. C    7. C    8. A    9. A    10. D  
11. C    12. C    13. A    14. C    15. D

#### Previous Years' Questions

1. A    2. B    3. C    4. D    5. D    6. A    7. D    8. C    9. C    10. B  
11. C    12. D    13. D    14. B    15. B    16. D    17. C    18. D    19. D    20. B  
21. C    22. C    23. B    24. B    25. C    26. D    27.    28. C    29. A    30. D  
31. C    32. C    33. 12 to 12    34. A    35. B    36. 6 to 6    37. 19    38. 110    39. 69  
40. A    41. C    42. 5    43. B    44. A    45. 7    46. B    47. 31    48. A    49. D  
50. 225    51. A    52. 4