

Algorithm Analysis and Design

DECAP538

Edited by
Ajay Kumar Bansal



LOVELY
PROFESSIONAL
UNIVERSITY



L O V E L Y
P R O F E S S I O N A L
U N I V E R S I T Y

Algorithm Analysis and Design

Edited By:
Ajay Kumar Bansal

Title: ALGORITHM DESIGN AND ANALYSIS

Author's Name: Dr. Divya

Published By : Lovely Professional University

Publisher Address: Lovely Professional University, Jalandhar Delhi GT road, Phagwara - 144411

Printer Detail: Lovely Professional University

Edition Detail: (I)

ISBN: 978-81-19334-37-7



Copyrights@ Lovely Professional University

Content

Unit 1:	Introduction	1
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 2:	Divide and Conquer	29
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 3:	Greedy Method	58
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 4:	Dynamic Programming	87
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 5:	Dynamic Programming	109
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 6:	Backtracking	140
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 7:	Branch and Bound	167
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 8:	Pattern Matching	197
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 9:	Huffman Encoding	226
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 10:	Lower Bound Theory	244
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 11:	More on Lower Bounds	254
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 12:	Learning and its Types	263
	<i>Dr. V Devendran, Lovely Professional University</i>	
Unit 13:	Interactable Problems	274
	<i>Dr. Divya, Lovely Professional University</i>	
Unit 14:	Interactable Problems	286
	<i>Dr. Divya, Lovely Professional University</i>	

Unit 01: Introduction

CONTENTS

- Objectives
- Introduction
- 1.1 Stacks
- 1.2 Queues
- 1.3 Trees
- 1.4 Dictionaries
- 1.5 Graphs
- 1.6 Computational Model
- 1.7 Algorithms
- 1.8 Finding the Time Complexity
- 1.9 Asymptotic Analysis
- 1.10 Recursion
- 1.11 Divide-and-Conquer Strategy
- 1.12 Computational Problem
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the basic data structures and computational models
- Understand the Asymptotic notations
- Understand what recursion is and how to trace the recursive function
- Write a recurrence relation
- Various strategies for solving problems

Introduction

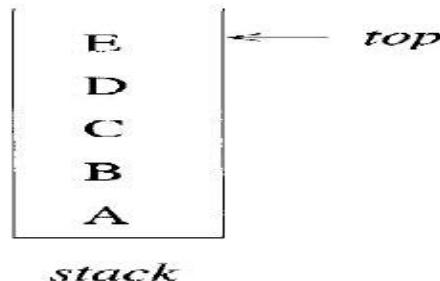
A data structure is a particular way of organizing data in a computer so that it can be used effectively. For example, we can store a list of items having the same datatype using the array data structure. There are various data structures available: stacks, queues, trees, dictionaries, and graphs.

1.1 Stacks

A stack is an ordered list in which all Insertions and deletions are made at one end, called the top. Stack follows the LIFO operations. The stack can be represented by using arrays and lists. These

Algorithm Analysis and Design

stacks can be represented by using a 1-D array. The representation is stack [0 : n-1], where n is the maximum number of allowable entries. The first or bottom element in the stack is stored at stack [0], the second at stack[1], and the i th at stack[i-1]. Associated with the array is a variable, typically called top, which points to the top element in the stack. To test whether the stack is empty, we ask "if (top < 0)". If not, the topmost element is at stack [top]. Checking whether the stack is full by "if (top > n-1)".



We can perform two types of operations on stacks. These are: push and pop. Push operation is the addition of elements in the stack. The pop operation is the removal of element from the stack. In both above cases, the top of the stack is updated.

Algorithm Add (item)

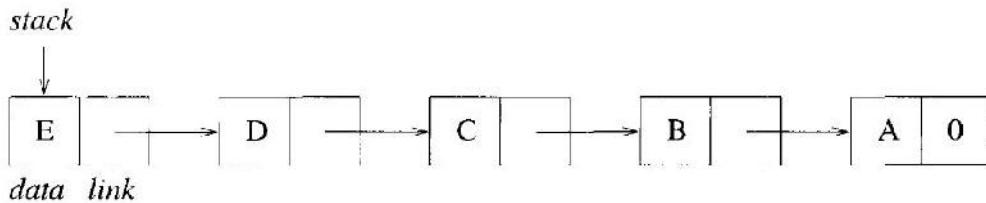
```
// Push an element on to the stack. Return true if successful. Else return false, item is used as an input.
{
    if (top ≥ n - 1) then
    {
        write ("Stack is full!\n");
        return false;
    }
    else
    {
        top := top + 1;
        stack[top] := item;
        return true;
    }
}
```

Algorithm Delete(item)

```
// Pop the top element from the stack. Return true if successful. Else return false, item is used as an output.
{
    if (top < 0) then
    {
        write ("Stack is empty!\n");
        return false;
    }
    else
    {
        item := stack[top];
        top := top - 1;
        return true;
    }
}
```

When we represent the stack using links, we make use of a pointer. In links, we have nodes. The node is a collection of data and link information. A stack can be represented by using nodes with two fields, possibly called data and link. These two things are explained as:

- **Data Field:** The data field of each node contains an item in the stack and the corresponding link field points to the node containing the next item in the stack.
- **Link Field:** The link field of the last node is zero, for we assume that all nodes have an address greater than zero.



The data can be represented as

node=record

```
{
Type data;
node *link;
}
```

Algorithm Add (item)

```
{
// Get a new node.
temp:=new node;
if (temp ≠ 0) then
{
  (temp → data) :=item;(temp → link) :=top;
  top :=temp; return true;
}
else
{
  write ("Out of space!");
  return false;
}
}
```

Algorithm Delete (item)

```
{
if (top = 0) then
{
  write (\ "Stack is empty!");
  return false;
}
else
{
  item:=(top → data); temp:= top;
  top :=(top → link);
}
```

Algorithm Analysis and Design

```

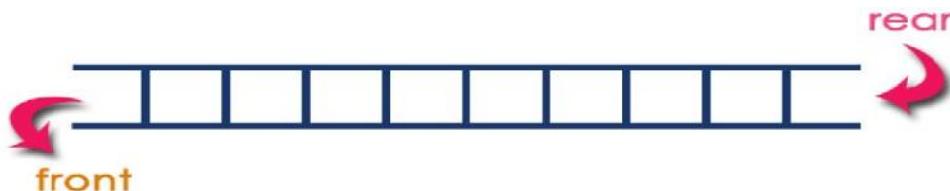
        delete temp;
        return true;
    }
}

```

The variable top points to the topmost node (the last item inserted) in the list. The empty stack is represented by setting top := 0. Because of the way the links are pointing, insertion and deletion are easy to accomplish. In the case of Add, the statement temp := new node; assigns to the variable temp the address of an available node. If no more nodes exist, it returns(). If a node exists, we store appropriate values into the two fields of the node. Then the variable top is updated to point to the new top element of the list. Finally, true is returned. If no more space exists, it prints an error message and returns false. Referring to Delete, if the stack is empty, then trying to delete an item produces the error message "Stack is empty!" and false is returned. Otherwise, the top element is stored as the value of the variable item, a pointer to the first node is saved, and top is updated to point to the next node. The deleted node is returned for future use and finally true is returned. The use of links to represent a stack requires more storage than the sequential array stack [0: n - 1] does. However, there is greater flexibility when using links, for many structures can simultaneously use the same pool of available space. Most importantly the times for insertion and deletion using either representation is independent of the size of the stack.

1.2 Queues

A queue is an ordered list in which all insertions take place at one end, the rear, whereas all deletions take place at the other end, the front. It follows the FIFO operations.



An efficient queue representation can be obtained by taking an array q[0 : n - 1] and treating it as if it were circular. Elements are inserted by increasing the variable rear to the next free position. When rear = n - 1, the next element is entered at q[0] in case that spot is free. The variable front always points one position counterclockwise from the first element in the queue. The variable front= rear if and only if the queue is empty and we initially set front:=rear := 0. Another way to represent a queue is by using links. As with the linked stack example, each node of the queue is composed of the two fields data and link. A queue is pointed at by two variables, front and rear. Deletions are made from the front, and insertions at the rear. Variable front= 0 signals an empty queue. To insert an element, it is necessary to move rear one position clockwise. This can be done using the code

```

if (rear= n - 1) then rear :=0;
else rear := rear + 1;

```

Algorithm Add (item)

```

{
    rear :=(rear+ 1) mod n; // Advance rear clockwise.
    if (front= rear) then
    {
        write ("Queue is full!");
        if (front=0) then rear := n-1;
        else rear :=rear - 1;
        // Move rear one position counterclockwise.
        Return false;
    }
}

```

```

}
else
{
q[rear] := item; // Insert new item.
return true;
}
}
}

```

Algorithm Delete Q (item)

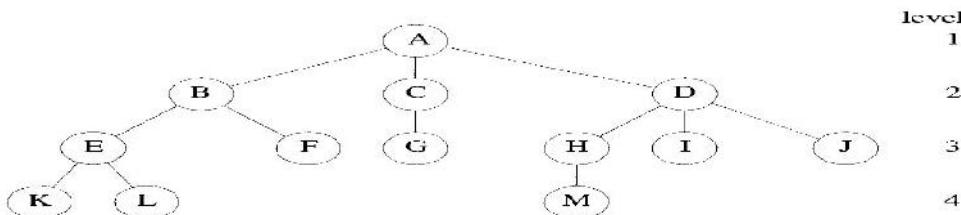
```

{
if (front= rear) then
{
write ("Queue is empty!");
return false;
}
else
{
front:=(front+ 1) mod n; // Advance front clockwise.
item:=q[front]; //Set item to front of queue.
return true;
}}

```

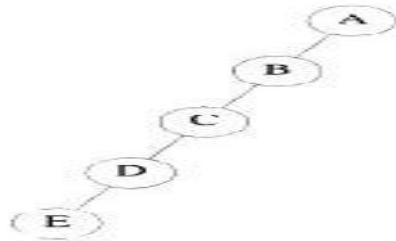
1.3 Trees

A tree is a finite set of one or more nodes such that there is a specially designated node called the root and the remaining nodes are partitioned into $n > 0$ disjoint sets T_1, \dots, T_n , where each of these sets is a tree. The sets T_1, \dots, T_n are called the subtrees of the root. The below figure represents the tree which has 13 nodes. Here the root node contains A. The number of subtrees of a node is called its degree. The degree of A is 3, of C is 1, and of F is 0. Nodes that have degree zero are called leaf or terminal nodes. The set $\{K, L, F, G, M, I, J\}$ is the set of leaf node. The other nodes are referred to as non-terminals. Thus, the children of D are H, I, and J, and the parent of D is A. The children of the same parent are said to be siblings. For example, H, I, and J are siblings. The degree of a tree is the maximum degree of the nodes in the tree. The ancestors of a node are all the nodes along the path from the root to that node. The ancestors of M are A, D, and H. The level of a node is defined by initially letting the root be at level one. If a node is at level p, then its children are at level $p + 1$. The height or depth of a tree is defined to be the maximum level of any node in the tree. A forest is a set of $n > 0$ disjoint trees. The notion of a forest is very close to that of a tree because if we remove the root of a tree, we get a forest.

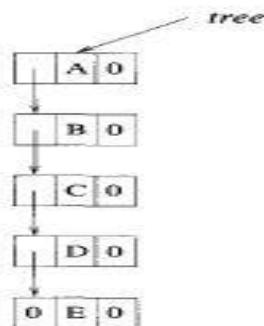
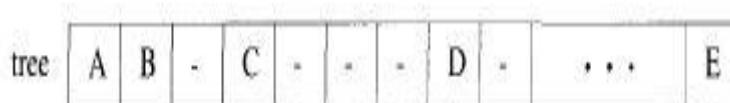


Binary Trees

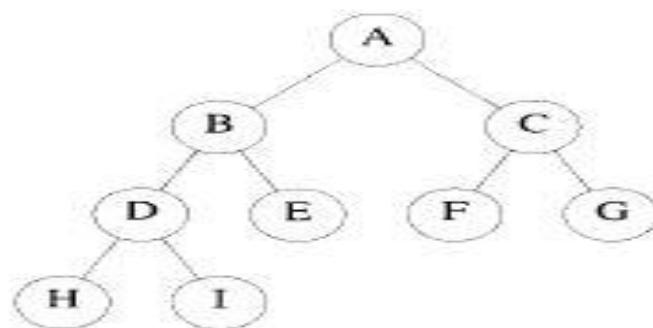
A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left and right subtrees. In a binary tree any node can have at most two children; that is, there is no node with degree greater than two. For binary trees we distinguish between the subtree on the left and that on the right. Furthermore, a binary tree is allowed to have zero nodes whereas any other tree must have at least one node. Thus, a binary tree is really a different kind of object than any other tree. The binary trees can be of type left skewed and right skewed. One example of left skewed is shown in below figure. If we follow the concept of sequential placing of elements, then a block will be left empty in place of missing element.

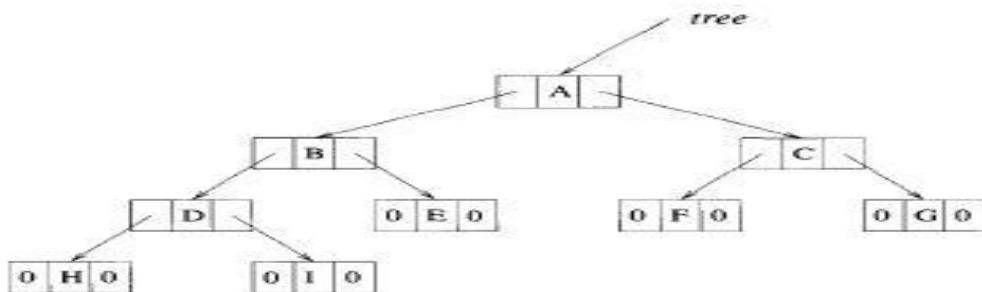
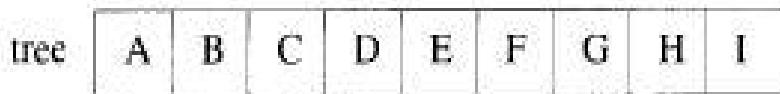


First A is placed, then B is placed. As there is no right child of A, then this place will be left empty. Then C is placed. Following the same manner, we will keep on placing the elements and leaving the empty spaces for missing elements.



This example shows the complete binary tree. A complete binary tree is a binary tree in which all the levels are filled except possibly the lowest one, which is filled from the left. A complete binary tree is just like a full binary tree, but with two major differences. All the leaf elements must lean towards the left. The same way is used to place the elements in arrays and lists.





Binary Search Tree

A binary search tree is a binary tree. It may be empty. If it is not empty, then it satisfies the following properties:

1. Every element has a key and no two elements have the same key (i.e., the keys are distinct).
2. The keys (if any) in the left subtree are smaller than the key in the root.
3. The keys (if any) in the right subtree are larger than the key in the root.
4. The left and right subtrees are also binary search trees.

A binary search tree can support the operation search, insert, and delete among others.

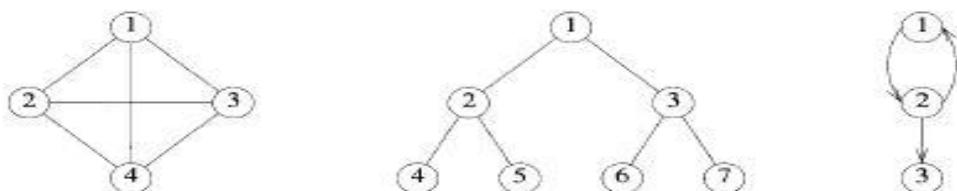


1.4 Dictionaries

A dictionary is a general-purpose data structure for storing a group of objects. A dictionary has a set of keys, and each key has a single associated value. When presented with a key, the dictionary will return the associated value. An abstract data type that supports the operations insert, delete, and search is called a dictionary. Dictionaries have found application in the design of numerous algorithms. **Example:** Consider the database of books maintained in a library system. When a user wants to check whether a particular book is available, a **search**, operation is called for. If the book is available and is issued to the user, a **delete** operation can be performed to remove this book from the set of available books. When the user returns the book, it can be **inserted** back into the set.

1.5 Graphs

A graph G consists of two sets V and E . The set V is a finite, nonempty set of vertices. The set E is a set of pairs of vertices; these pairs are called edges. The notations $V(G)$ and $E(G)$ represent the sets of vertices and edges, respectively, of graph G . We also write $G = (V, E)$ to represent a graph. In an undirected graph the pair of vertices representing any edge is unordered. Thus, the pairs (u, v) and (v, u) represent the same edge. In a directed graph each edge is represented by a directed pair (u, v) ; u is the tail and v the head of the edge. Therefore, (v, u) and (u, v) represent two different edges. Few examples of graphs are shown below.



Adjacency matrices of graphs: We can also form the adjacency matrices of graphs.

	1	2	3
1	0	1	0
2	1	0	1
3	0	0	0

	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

1.6 Computational Model

A computational model is a mathematical model in computational science that requires extensive computational resources to study the behavior of a complex system by computer simulation. The goal of computational modeling is to use precise mathematical models to make better sense of data. The common basis of programming language and computer architecture is known as computational model. It provides higher level of abstraction than the programming language and the architecture. Computational model is the combination of the above two things.

1.7 Algorithms

It is a step-by-step procedure for solving a computational problem. Algorithms are most important and durable part of computer science because they can be studied in a language- and machine-independent way. This means that we need techniques that enable us to compare the efficiency of algorithms without implementing them. The two most important tools are: The RAM model of computation and, The asymptotic analysis of worst-case complexity.

RAM Model

RAM stands for Random Access Machine. It is not R.A. Memory. An idealized notion of how the computer works, each "simple" operation (+, -, =, if) takes exactly 1 step. Each memory access takes exactly 1 step. Loops and method calls are not simple operations but depend upon the size of the data and the contents of the method. It measures the run time of an algorithm by counting the number of steps. The program for RAM is not stored in memory. Thus, we are assuming that the program does not modify itself. A Random-Access Machine (RAM) consists of: a fixed program, an unbounded memory, a read-only input tape, a write-only output tape, each memory register can hold an arbitrary integer (*) and each tape cell can hold a single symbol from a finite alphabet s.

Space Complexity

The amount of memory required by an algorithm to run to completion.

- **Fixed part:** The size required to store certain data/variables, that is independent of the size of the problem: Such as int a (2 bytes, float b (4 bytes) etc.
- **Variable part:** Space needed by variables, whose size is dependent on the size of the problem: Dynamic array a[].

$$S(P) = c + S(\text{instance characteristics})$$

c = constant. Constant Space: one for n, one for a [passed by reference!], one for s, one for I , constant space=c=4

Running Time of Algorithms

The running time of an algorithm depends on input size n . The bits in the binary representation of the input vertices and edges in a graph gives the running time of an algorithm. It is based upon the number of primitive operations performed. A primitive operation is the unit of operation that can be identified in the pseudo-code. There is a predefined procedure for determining the time complexity:

Step-1. Determine how you will measure input size. Example: N items in a list $N \times M$ table (with N rows and M columns). Then there will be two numbers of length N .

Step-2. Choose the type of operation (or perhaps two operations) Comparisons, Swaps, Copies, Additions.

Step-3. Decide whether you wish to count operations in the

- Best case? - the fewest possible operations.
- Worst case? - the most possible operations.
- Average case? - This is harder as it is not always clear what is meant by an "average case".

Normally calculating this case requires some higher mathematics such as probability theory.

Step-4. For the algorithm and the chosen case (best, worst, average), express the count as a function of the input size of the problem. Some examples of primitive operations in an algorithm are given as:

- Assign a value to a variable (i.e., $a=5$)
- Call a method (i.e., `method()`)
- Arithmetic operation (i.e., $a*b$, $a-b*c$)
- Comparing two numbers (i.e., $a \leq b$, $a > b$ & $\&a > c$)
- Indexing into an array (i.e., $a[0]=5$)
- Following an object reference (i.e., `Test obj`) Returning from a method (i.e., `return I`)

The running time of a for loop is at most the running time of the statements inside the for loop (including tests) times the number of iterations.

- Example: `for(i=0;i<=n-1;i++)`

$A[i]=0$; Let running time of basic operations is constant C and loop is iterated n times then Total Running time will be $n*C$. Note: (number of basic steps in loop body) * (number of iterations)

RASP Machine or stored program model

It is same as RAM but allow program to change itself as it is now stored in the memory. It has same power as RAM. The example is Von Neumann architecture. Around 1944–1945, John von Neumann proposed that, since program and data are logically the same, programs should also be stored in the memory of a computer. Computers built on the von Neumann model divide the computer hardware into four subsystems: memory, arithmetic logic unit, control unit, and input/output. The von Neumann model states that the program must be stored in memory. This is totally different from the architecture of early computers in which only the data was stored in memory: the programs for their task were implemented by manipulating a set of switches or by changing the wiring system. The memory of modern computers hosts both a program and its corresponding data. This implies that both the data and programs should have the same format, because they are stored in memory. In fact, they are stored as binary patterns in memory – a sequence of 0s and 1s.

Sequential execution of instructions

A program in the von Neumann model is made of a finite number of instructions. In this model, the control unit fetches one instruction from memory, decodes it, then executes it. In other words, the instructions are executed one after another. Of course, one instruction may request the control unit to jump to some previous or following instruction, but this does not mean that the instructions are not executed sequentially. Sequential execution of a program was the initial requirement of a

Algorithm Analysis and Design

computer based on the von Neumann model. Today's computers execute programs in the order that is the most efficient.

Difference between random access machine and random-access memory

The Random-Access Machine is a simplified model of a modern computer used in theoretical computer science to analyze algorithms. Random Access Memory is the primary memory device in which the CPU stores data in a real-life computer. As already discussed, an algorithm is a step-by-step procedure for solving a computational problem and a program is also a step-by-step procedure for solving a computational problem. Although there are few differences between an algorithm and a program:

Algorithm	Program
It is written at the design time.	It is written at the implementation time.
No need to write proper syntax, write in simple English sentences.	Always be in proper syntax.
For writing an algorithm, a person must have domain knowledge. Any language can be used for writing algorithms.	The program is written only using programming language.
Not dependent upon hardware and OS.	Dependent upon hardware and OS.
We analyse the algorithm.	We test the programs.
Priori analysis is done over the algorithms by studying it into greater details knowing how it is working and we get some results.	Posteriori testing is done on the programs, so we run and test how much time it is taking to execute and the amount of memory it is consuming in terms of bytes.
We may find the time and space consumed by an algorithm.	We may find time taken for execution of program.

Characteristics of algorithm

- 1) Input – It may take 0 or more inputs.
- 2) Output – It must generate atleast 1 output.
- 3) Definiteness – Every statement must have one single and exact meaning.
- 4) Finiteness – Algorithm must have finite set of statements. It must terminate at some point.
- 5) Effectiveness – Statements must serve some purpose.

Algorithm writing

A simple algorithm of swapping can be written as:

```
Algorithm swap (a, b)
{
    temp = a;
    a = b;
    b = temp;
}
```

There are few points that must be seen in an algorithm are:

- 1) Data types are not decided at the time of algorithm making.
- 2) Data declaration is also not done at the time of algorithm making.
- 3) For assignment, we can also use `:=`, `<-` in place of `=`.
- 4) To show the beginning and end, we can also use the keywords, begin and end respectively.
- 5) Instead of brackets, we can also use keywords like begin and end.

As discussed, these algorithms are the procedures for solving a problem, whether you are solving it manually on a paper or making a program so that your computer can solve it. Either way we are going to analyse it. There are few criteria for analysis: time, space, data transfer, network consumption, power consumption and CPU register consumption. Out of all, the time and space taken are two important points for analysis. Because the data transfer, network consumption etc. are point of concern when it involves the communication through network. So, the main consideration will be time and space taken.

Analysis of time taken: As discussed, every simple statement takes 1 unit of time. So here in the algorithm of swapping, the time taken is 3 which is a constant. So, $F(n) = 3$. The constant is represented as $O(1)$.

Analysis of space taken: The variables taken in the algorithm are `a`, `b`, and `temp`. The space taken is 3 words which is a constant. Constant is represented as $O(1)$. The analysis can be done by using the frequency count method. The few examples can be seen below:



Example 1:

```
algorithm sum (a, n)
{
    s=0;                                1
    for (i=0; i<n; i++)                n + 1
    {
        s = s + a[ i ];                  n
    }
    return s;                            1
}                                         F (n) = 2n + 3
```

$F(n) = 2n + 3$, so degree of polynomial = 1. Then the time complexity = $O(n)$. Space complexity: Variables used = `A`, `n`, `S`, `i`. `A:n` words, `n:1` word, `S:1` word, `i:1` word. $S(n) = n + 3$ words, so space complexity = $O(n)$



Example 2:

```
Algorithm Add (A, B, n)
{
    for (i=0; i<n; i++)                N + 1
    {
        for(j=0; j<n; j++)            N * (N+1)
        {
            C[i, j] = A[i, j] + B[i, j];   N * N
        }
    }
}                                         F(n) = 2n^2 + 2n + 1
```

$F(n) = 2n^2 + 2n + 1$, here the highest degree is 2. So, the time complexity is $O(n^2)$. Variables: `A`, `B`, `C`, `n`, `i`, `j`. `A: n^2`, `B: n^2`, `C: n^2`, `n: 1`, `i: 1` and `j: 1` word. Space complexity $S(n)$ is $3n^2 + 3$. Degree is $O(n^2)$.

Algorithm Analysis and Design**1.8 Finding the Time Complexity**

Below few examples are given for finding the time complexity.



Example 1:

```
for ( i = 0; i < n; i++) _____ N+1
{
  A line of code or a statement _____ N
}
```

$O(n)$

The for loop will run for $N+1$ number of times. This will not affect the degree of code. We can skip this. We need to check for how many times the statement inside the loop will run. So, the degree will depend upon that.



Example 2:

```
for ( i = n; i > 0; i--)
{
  A line of code or a statement _____ N
}
```

$O(n)$



Example 3:

```
for ( i = 1; i < n; i = i+2)
{
  A line of code or a statement _____ N / 2
}
```

$O(n)$



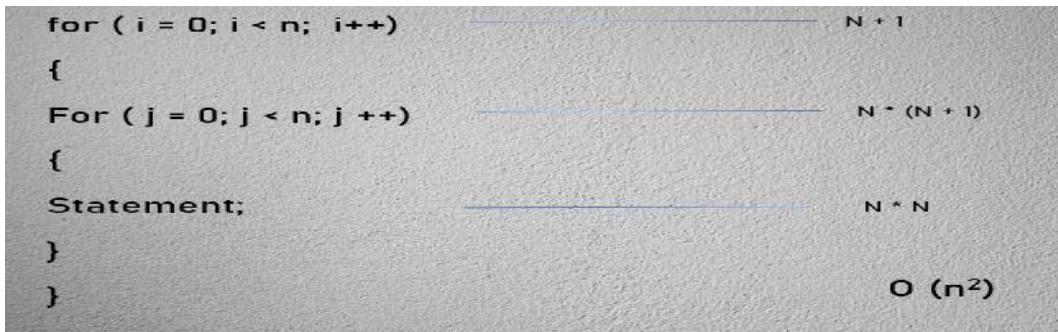
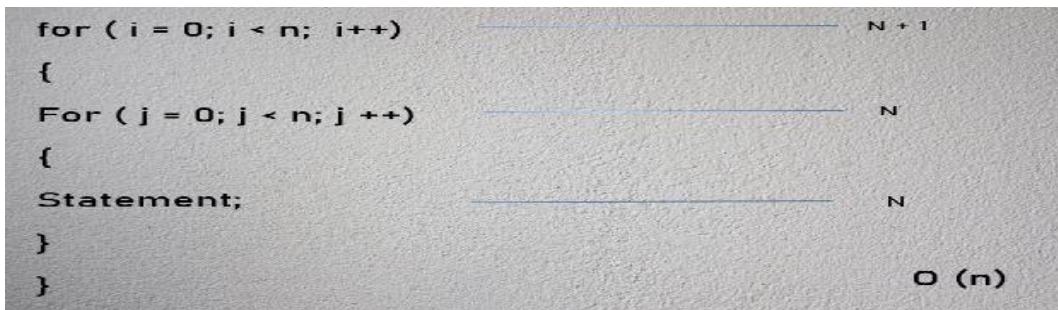
Example 4:

```
for ( i = 1; i < n; i = i+20)
{
  A line of code or a statement _____ N / 20
}
```

$O(n)$



Example 5:



Types of Time Functions

1. $O(1)$ → Constant
2. $O(\log n)$ → Logarithmic
3. $O(n)$ → Linear
4. $O(n^2)$ → Quadratic
5. $O(n^3)$ → Cubic
6. $O(2^n), O(3^n), O(n^n)$ → Exponential

Comparison of Classes of Functions

So, the classes of functions can be written and compared as: $1 < \log n < (n)^{1/2} < n < n \log n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$. If we have two algorithms and we want to see which one is taking less time. Then one naive way of doing this is – implement both the algorithms and run the two programs on your computer for different inputs and see which one takes less time. There are many problems with this approach for analysis of algorithms.

- 1) It might be possible that for some inputs, first algorithm performs better than the second. And for some inputs second performs better.
- 2) It might also be possible that for some inputs, first algorithm performs better on one machine and the second works better on other machine for some other inputs. The solution to the above problem is the asymptotic analysis.

1.9 Asymptotic Analysis

Asymptotic Analysis is the big idea that handles above issues in analyzing algorithms. In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time). We calculate, how the time (or space) taken by an algorithm increases with the input size. Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis. These notations are used for representing the symbol form or the more communicable form of a function. The notations used are:

- 1) O (Big - Oh notation): Works as an upper bound of a function.
- 2) Ω (Big - Omega notation): Works as a lower bound of a function.
- 3) Θ (Theta notation): Works as an average bound of a function.

Algorithm Analysis and Design

If we see the comparison of classes, $1 < \log n < (n)^{1/2} < n < n \log n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$, then Out of these, Θ is most useful. O provides the upper bound and Ω provides the lower bound.

Big Oh notation

The function $f(n) = O(g(n))$ iff \exists +ve constants C and n_0 . such that $f(n) \leq C * g(n) \forall n_0$. Example

- $F(n) = 2n + 3$ $f(n) \leq C * g(n) \forall n_0$
- So, $2n + 3 \leq \dots$.

Here $2n + 3$ must be less than or equal to \dots . We can write anything which fulfils the condition.

Can it be $10n$??

- $2n + 3 \leq 10n$ $f(n) \leq C * g(n) \forall n_0$
 - For $n = 0$, $2(0) + 3 \leq 10(0) \Rightarrow 3 \leq 0$ FALSE
 - For $n = 1$, $2(1) + 3 \leq 10(1) \Rightarrow 5 \leq 10$ TRUE
 - For $n = 2$, $2(2) + 3 \leq 10(2) \Rightarrow 7 \leq 20$ TRUE
 - For $n = 3$, $2(3) + 3 \leq 10(3) \Rightarrow 9 \leq 30$ TRUE
- So, $2n + 3 \leq 10n$ for $n \geq 1$
- So, according to the function $f(n) \leq C * g(n)$,
 - $F(n) = 2n + 3$, $C = 10$ and $g(n) = n$.
 - Therefore, $f(n) = 2n + 3 = O(n)$.

Can it be $7n$??

- $2n + 3 \leq 7n$ $f(n) \leq C * g(n) \forall n_0$
 - For $n = 0$, $2(0) + 3 \leq 7(0) \Rightarrow 3 \leq 0$ FALSE
 - For $n = 1$, $2(1) + 3 \leq 7(1) \Rightarrow 5 \leq 7$ TRUE
 - For $n = 2$, $2(2) + 3 \leq 7(2) \Rightarrow 7 \leq 14$ TRUE
 - For $n = 3$, $2(3) + 3 \leq 7(3) \Rightarrow 9 \leq 21$ TRUE
- So, $2n + 3 \leq 7n$ for $n \geq 1$
- So, according to the function $f(n) \leq C * g(n)$,
 - $F(n) = 2n + 3$, $C = 7$ and $g(n) = n$.
 - Therefore, $f(n) = 2n + 3 = O(n)$.

So, what it can be????

It can be anything on R.H.S. which fulfils the condition. Instead of struggling for the value at the R.H.S, we can write it in the terms of n .

So, can it be in terms of n ??

- $2n + 3 \leq 2n + 3n$ $f(n) \leq C * g(n) \forall n_0$
- $2n + 3 \leq 5n$
- For $n = 0$, $2(0) + 3 \leq 5(0) \Rightarrow 3 \leq 5$ FALSE
- For $n = 1$, $2(1) + 3 \leq 5(1) \Rightarrow 5 \leq 5$ TRUE
- For $n = 2$, $2(2) + 3 \leq 5(2) \Rightarrow 7 \leq 10$ TRUE
- For $n = 3$, $2(3) + 3 \leq 5(3) \Rightarrow 9 \leq 15$ TRUE

So, $2n + 3 \leq 5n$ for $n \geq 0$

So, according to the function $f(n) \leq C * g(n)$,

Unit 01: Introduction

- $F(n) = 2n + 3$, $C = 5$ and $g(n) = n$.
- Therefore, $f(n) = 2n + 3 = O(n)$.

Can it be in terms of n^2 ??

- $2n + 3 \leq 2n^2 + 3n^2$ $f(n) \leq C * g(n) \forall n_0$
- $2n + 3 \leq 5n^2$
- For $n = 0$, $2(0) + 3 \leq 5(0)^2 \Rightarrow 3 \leq 5$ FALSE
- For $n = 1$, $2(1) + 3 \leq 5(1)^2 \Rightarrow 5 \leq 5$ TRUE
- For $n = 2$, $2(2) + 3 \leq 5(2)^2 \Rightarrow 7 \leq 10$ TRUE
- For $n = 3$, $2(3) + 3 \leq 5(3)^2 \Rightarrow 9 \leq 15$ TRUE

So, $2n + 3 \leq 5n$ for $n \geq 0$

So, according to the function $f(n) \leq C * g(n)$,

- $F(n) = 2n + 3$, $C = 5$ and $g(n) = n^2$.

Therefore, $f(n) = 2n + 3 = O(n^2)$.

$1 < \log n < (n)^{1/2} < n < n \log n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$

So, all the functions greater than or equal, belongs to the class of upper bound.

Big Omega notation

- The function $f(n) = \Omega(g(n))$ iff \exists +ve constants C and n_0 . such that $f(n) \geq C * g(n) \forall n_0$.
- $F(n) = 2n + 3$
- $2n + 3 \geq \dots$
- Here $2n + 3$ must be greater than or equal to \dots . We can write anything which fulfils the condition.

Can it be n ??

- $2n + 3 \geq 1 * n$
- For $n = 1$, $2(1) + 3 \geq 1, 5 \geq 1$ TRUE
- For $n = 2$, $2(2) + 3 \geq 2 * 2, 7 \geq 4$ TRUE

So, $2n + 3 \geq 1 * n$, for $n \geq 1$

So, according to the function $f(n) \geq C * g(n)$,

- $F(n) = 2n + 3$, $C = 1$ and $g(n) = n$.

Therefore, $f(n) = 2n + 3 = \Omega(n)$.

Can it be n^2 ??

- $2n + 3 \geq 1 * n^2$
- For $n = 1$, $2(1) + 3 \geq (1)^2, 5 \geq 1$ TRUE
- For $n = 2$, $2(2) + 3 \geq 2 * 2, 7 \geq 4$ TRUE
- For $n = 3$, $2(3) + 3 \geq 3 * 3, 9 \geq 9$ TRUE
- For $n = 4$, $2(4) + 3 \geq 4 * 4, 11 \geq 16$ FALSE

So, it can not be n^2 .

- $1 < \log n < (n)^{1/2} < n < n \log n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$

So, all the functions less than or equal, belongs to the class of upper bound.

Big Theta notation

The function $f(n) = \Theta(g(n))$ iff \exists +ve constants C_1 and C_2 and n_0 . such that $C_1 * g(n) \leq f(n) \leq C_2 * g(n) \forall n_0$.

Example

- $F(n) = 2n + 3$
 - $1 * n \leq 2n + 3 \leq 5 * n$
 - $C_1 * g(n) \leq f(n) \leq C_2 * g(n)$
- So, $C_1 = 1$, $C_2 = 5$, $g(n) = n$ and $f(n) = 2n + 3$.
- $1 < \log n < (n)^{1/2} < n < n \log n < n^2 < n^3 < \dots < 2^n < 3^n < \dots < n^n$

N defines the average bound of the function.



Example - 1: $F(n) = 2n^2 + 3n + 4$; $2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$; $2n^2 + 3n + 4 \leq 9n^2$,
 $n \geq 1$, $C = 9$, $g(n) = n^2$, So, $f(n) = O(n^2)$; $F(n) = 2n^2 + 3n + 4$; $2n^2 + 3n + 4 \geq 1 * n^2$;
 $2n^2 + 3n + 4 \geq 1 * n^2$, $n \geq 1$; $C = 1$, $g(n) = n^2$, So, $f(n) = \Omega(n^2)$; $F(n) = 2n^2 + 3n + 4$;
 $* n^2 \leq 2n^2 + 3n + 4 \geq 9 * n^2$; $1 * n^2 \leq 2n^2 + 3n + 4 \geq 9 * n^2$, $n \geq 1$; $C_1 = 1$, $C_2 = 9$, $g(n) = n^2$, So, $f(n) = \Theta(n^2)$



Example - 2: $F(n) = n!$, $n! = n * n-1 * n-2 * n-3 * \dots * 3 * 2 * 1$; $n! = 1 * 2 * 3 * \dots * n-3 * n-2 * n-1 * n$;
 $1 * 1 * 1 * \dots * 1 \leq 1$; $2 * 3 * \dots * n \leq n * n * n * \dots * n$; $1 \leq n! \leq n^n$. On both sides, things are different. So, we cannot write in terms of Θ . So, it is $\Omega(1)$ and $O(n^n)$. $1 < \log n < (n)^{1/2} < n < n \log n < n^2 < n^3 < \dots < 2^n < 3^n < \dots < n^n$. We cannot find the exact place of $n!$. We can only find the upper and lower bounds.



Example - 3: $F(n) = \log n!$; $\log(1 * 1 * 1 * \dots * 1) \leq \log(1 * 2 * 3 * \dots * n) \leq \log(n * n * n * \dots * n)$; $1 \leq \log n! \leq \log n^n$. $\Omega(1)$ and $O(n \log n)$.

Factorial function

We cannot define the average or tight bound for factorial functions. We can find upper and lower bounds for these.

Comparisons of functions

Suppose the functions that wish to compare are n^2 and n^3 .

Method 1: Sample some values: There are n number of values which can be sampled. So, identifying a set is quite difficult.

Method 2: Apply log on both sides. So, the log formulas for that are:

- 1) $\log ab = \log a + \log b$
- 2) $\log a/b = \log a - \log b$
- 3) $\log a^b = b \log a$
- 4) $a^{\log_b c} = b^{\log_a c}$
- 5) $a^b = n$ then $b = \log_n a$



Example 1:

$F(n) = n^2 \log n$ and $g(n) = n (\log n)^{10}$. Compare these two functions and find out which one is greater. Use the method of applying log on both sides.

$$= \log(n^2 \log n) \quad \log(n (\log n)^{10})$$

Unit 01: Introduction

$$\begin{array}{ll}
 = \text{Log } n^2 + \text{log}(\text{log } n) & \text{log } n + \text{log} (\text{log } n)^{10} \\
 = 2 \text{log } n + \text{log}(\text{log } n) & \text{log } n + 10 \text{log} (\text{log } n) \\
 \text{=2 log n & log n: So, 2 log n is bigger} & \\
 = 2 \text{log } n + \text{log}(\text{log } n) & \text{log } n + 10 \text{log} (\text{log } n) \\
 \text{=2 log n & log n: So, 2 log n is bigger} & \\
 \text{So, } F(n) = n^2 \text{ log } n \text{ is bigger.} &
 \end{array}$$



Example 2:

$$F(n) = n^{\log n} \text{ and } g(n) = 2 \sqrt{n}$$

Apply log on both sides

$$\begin{array}{ll}
 = \text{Log } n^{\log n} & \text{log } 2 \sqrt{n} \\
 = \text{Log } n * \log n & \sqrt{n} * \log^2 2 \\
 = \text{Log}^2 n & \sqrt{n} * 1 \\
 = \text{Log}^2 n & \sqrt{n} \\
 = \text{Log}^2 n & \sqrt{n}
 \end{array}$$

Again apply log on both sides

$$\begin{array}{ll}
 = \text{Log} (\text{Log}^2 n) & \log(\sqrt{n}) \\
 = 2 \text{log}(\text{log } n) & \frac{1}{2} \log n
 \end{array}$$

So, $f(n)$ is smaller than $g(n)$.

Best, worst, and average case analysis.

Linear search

92	87	53	10	15	23	67
0	1	2	3	4	5	6

In this, there can be successful search (if the element is found in the list) and unsuccessful search (if the element is not found in the list). So, here we need to find the best case, average case and worst case.

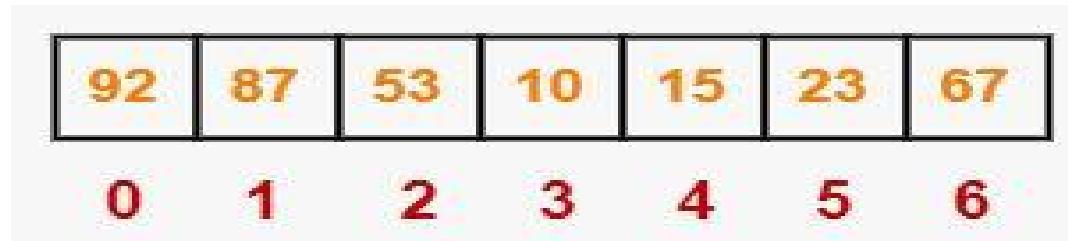
Best case

In case, the element which you are searching is present at first index. Time taken in this case: 1 (Constant) and it can be written as $O(1)$; $B(n) = O(1)$.

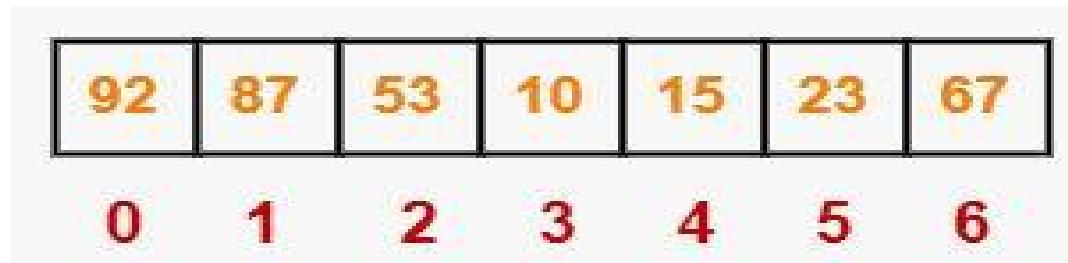
92	87	53	10	15	23	67
0	1	2	3	4	5	6

Algorithm Analysis and Design**Worst case**

If it takes maximum time means the element which we are searching, is present at last. Time taken is n . $W(n) = O(n)$.

**Average case**

- Average case = (All possible case time / no of cases).
- Average case time = $1 + 2 + 3 + \dots + n / n$.
- Average case time = $n(n+1)/2/n = (n + 1) / 2$.
- Average case analysis may not be possible always.



- $B(n) = 1; B(n) = O(1); B(n) = \Omega(1); B(n) = \Theta(1)$.
- $W(n) = n; W(n) = O(1); W(n) = \Omega(1); W(n) = \Theta(1)$.

1.10 Recursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc. Recursive thinking is important in programming. It helps you break down big problems into smaller ones. Often, the recursive solution can be simpler to read than the iterative one. Let us consider a problem that a programmer has to determine the sum of first n natural numbers, there are several ways of doing that but the simplest approach is simply add the numbers starting from 1 to n .

- **Approach(1)** – Simply adding one by one

$$f(n) = 1 + 2 + 3 + \dots + n$$
- **Approach(2)** – Recursive adding

$$\begin{array}{ll} f(n) = 1 & n=1 \\ f(n) = n + f(n-1) & n>1 \end{array}$$

There is a simple difference between the approach (1) and approach(2) and that is in approach(2) the function “ $f()$ ” itself is being called inside the function.

What is base condition in recursion?

In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
```

```

{
    if (n <= 1) // base case
        return 1;
    else
        return n*fact(n-1);
}

```

Here, base case for $n \leq 1$ is defined and larger value of number can be solved by converting to smaller one till base case is reached. If the base case is not reached or not defined, then the stack overflow problem may arise.

```

int fact(int n)
{
    // wrong base case (it may cause stack overflow).
    if (n == 100)
        return 1;
    else
        return n*fact(n-1);
}

```

If $\text{fact}(10)$ is called, it will call $\text{fact}(9)$, $\text{fact}(8)$, $\text{fact}(7)$ and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error.

When any function is called from $\text{main}()$, the memory is allocated to it on the stack. A recursive function calls itself, the memory for a called function is allocated on top of memory allocated to calling function and different copy of local variables is created for each function call. When the base case is reached, the function returns its value to the function by whom it is called, and memory is de-allocated and the process continues.

```

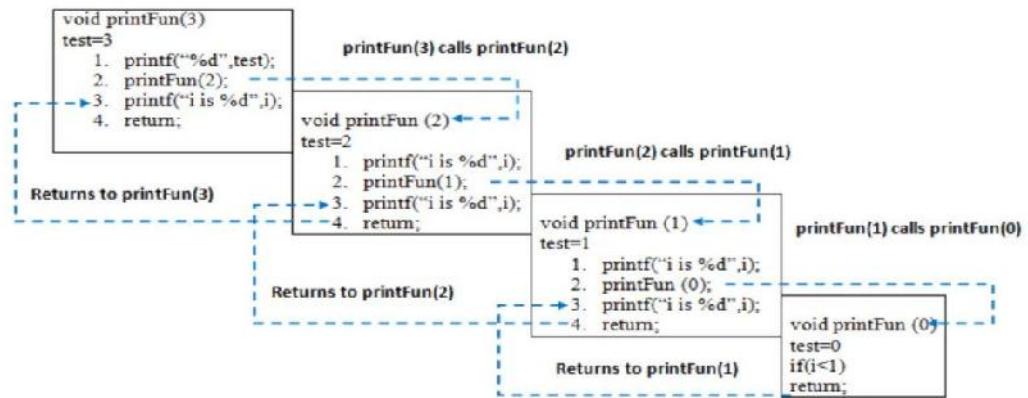
void printFun(int test)
{
    if (test < 1)
        return;
    else {
        cout << test << " ";
        printFun(test - 1); // statement 2
        cout << test << " ";
        return;
    }
}

// Driver Code
int main()
{
    int test = 3;
    printFun(test);
}

```

Output : 3 2 1 1 2 3

The memory stacks



Advantages and disadvantages of recursion

Recursion reduces the program size and makes it compact. It avoids redundancy of code. As a result, the code is easier to maintain. Reduce unnecessary calling of function. Through recursion one can solve problems in easy way.

Recursive solution is always logical, and it is very difficult to trace.(debug and understand). In recursive we must have an if statement somewhere to force the function to return without the recursive call being executed, otherwise the function will never return. Recursion takes a lot of stack space, usually not considerable when the program is small and running on a PC. Recursion uses more processor time.

1.11 Divide-and-Conquer Strategy

This technique can be divided into the following three parts:

1. **Divide:** This involves dividing the problem into smaller sub-problems.
2. **Conquer:** Solve sub-problems by calling recursively until solved.
3. **Combine:** Combine the sub-problems to get the final solution of the whole problem.

The algorithm for divide and conquer strategy is:

Algorithm DAC(P)

```

{
  if Small( $P$ ) then return S( $P$ );
  else
  {
    divide  $P$  into smaller instances  $P_1, P_2, \dots, P_k$ ,  $k \geq 1$ ;
    Apply DAC to each of these subproblems;
    return Combine(DAC( $P_1$ ), DAC( $P_2$ ), ..., DAC( $P_k$ ));
  }
}
  
```

The important thing which relates to any algorithm is the recurrence relation. Below few examples are given related to decreasing and dividing functions.



Example - 1

```
void test (int n)
```

```
{
```

```
if (n > 0)
```

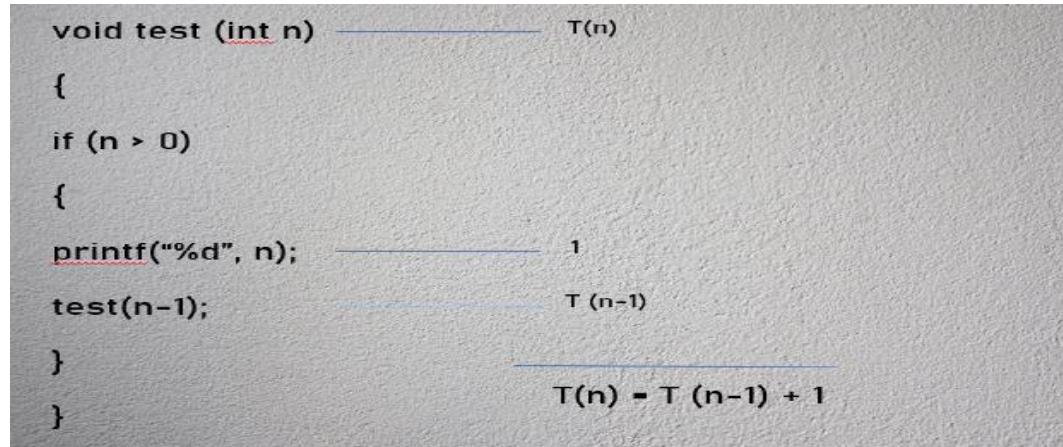
```
{
```

```

printf("%d", n);
test(n-1);
}
}
n = 4

```

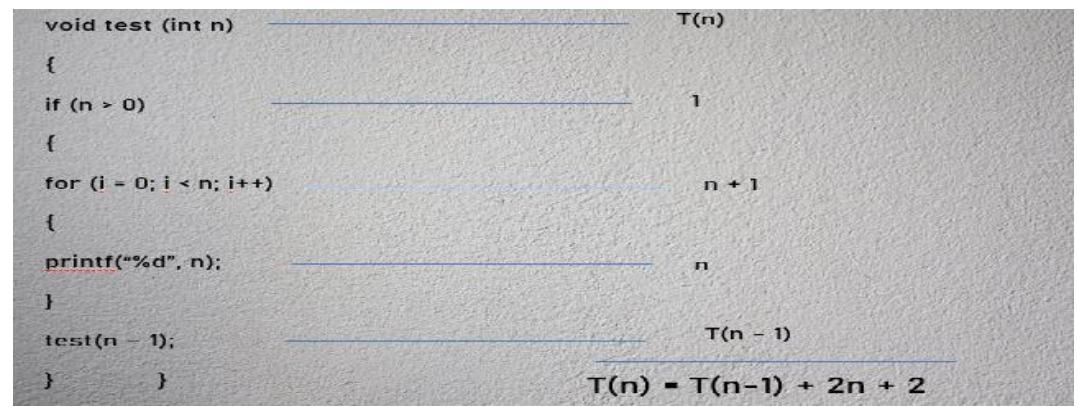
In the example, major work done is just printing of values. Here, 4 number of times it is printing the values and 5 number of times the function is calling itself. Time taken for printing: each one of this will take 1 unit of time. Time taken for making calls: Each one for making calls will take 1 unit of time. `printf` is executing for n number of times and the calls to the function are made for $n+1$ number of times. It depends upon the number of calls, the time taken is $n+1$, time complexity is $O(n)$. The recurrence relation for this is:



- $T(n) = \{ 1 \quad n = 0$
 $\quad \quad \quad \{ T(n-1) + 1 \quad n > 0$
- $T(n) = T(n-1) + 1$
- $T(n-1) = T(n-2) + 1$
- Substitute $T(n-1)$; $T(n) = [T(n-2) + 1] + 1$
- $T(n) = T(n-2) + 2$
- $T(n) = [T(n-3) + 1] + 2$
- $T(n) = T(n - 3) + 3$
- $T(n) = T(n - 3) + 3$
- If we continue this for k (number of steps) times,
- then, $T(n) = T(n-k) + k$
- We know that for $n=0$, $T(n) = 1$.
- So, we assume that we have reached at 0, $n-k = 0$
- $n-k = 0; n = k$
- $T(n) = T(n - n) + n; T(n) = T(0) + n; T(n) = 1 + n = O(n)$



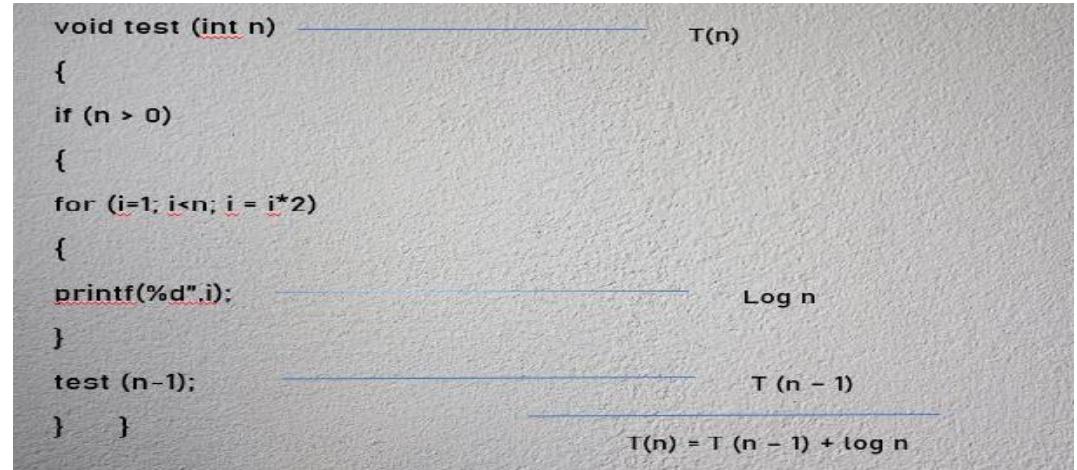
Example - 2

Algorithm Analysis and Design

- $T(n) = T(n-1) + 2n + 2$
- $T(n) = T(n-1) + n$ (After rounding off)
- $T(n) = \{ 1 \quad n = 0$
 $\quad \{ T(n-1) + n \quad n > 0$
- $0 + 1 + 2 + 3 + \dots + n-2 + n-1 + n = n(n+1) / 2$
- $T(n) = n(n+1) / 2$
- Time complexity is $O(n^2)$.



Example – 3



- $T(n) = \{ 1 \quad n = 0$
- { $T(n-1) + \log n \quad n > 0$

$$\begin{aligned}
 &= \log n + \log(n-1) + \log(n-2) + \dots + \log 2 + \log 1 \\
 &= \log(n * (n-1) * \dots * 2 * 1) \\
 &= \log(1 * 2 * \dots * (n-1) * n) \\
 &= \log(n!)
 \end{aligned}$$

There is no average bound for $n!$. There is an upper bound: $O(n \log n)$. Time taken is $n \log n$.

From the above given three examples,

- Example 1: $T(n) = T(n-1) + 1 \dots O(n)$
- Example 2: $T(n) = T(n-1) + n \dots O(n^2)$
- Example 3: $T(n) = T(n-1) + \log n \dots O(n \log n)$

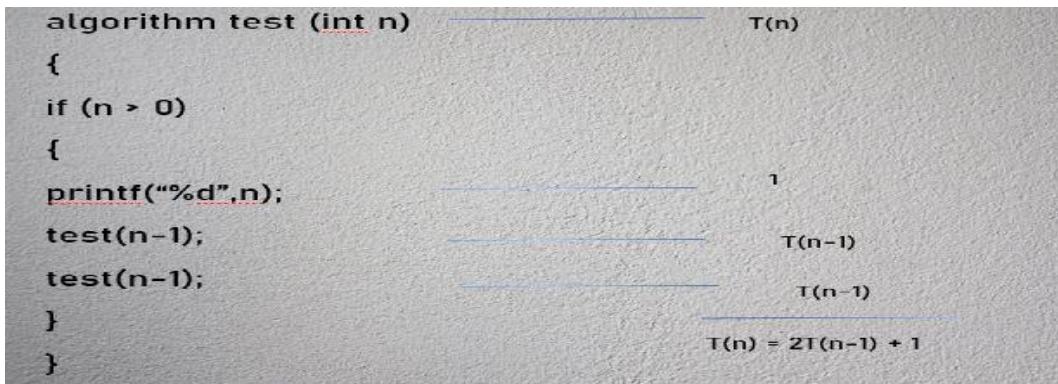
we can conclude:

Unit 01: Introduction

- If $T(n) = T(n-1) + n^2 \dots O(n^3)$



Example - 4



- $T(n) = \{ 1 \quad n = 0$
 $\quad \quad \quad = \{ 2T(n-1) + 1 \quad n > 0$
- $1, 2, 4, 8, \dots$
 $= 1, 2^1, 2^2, 2^3, \dots 2^k$
 $= \text{Total time will be } 1 + 2 + 2^2 + 2^3 + \dots + 2^k$
 $= \text{This is the sum of GP series.}$
 $= \text{So the time taken is } 2^{k+1} - 1.$
- Assume $n-k=0$
 $\text{So, } n=k$
 $\text{So, the time taken will be } 2^{n+1} - 1$
 $\text{Hence, Time complexity is } O(2^n)$

Master theorem

Some of the decreasing functions are:

- Example 1: $T(n) = T(n-1) + 1 \dots O(n)$
- Example 2: $T(n) = T(n-1) + n \dots O(n^2)$
- Example 3: $T(n) = T(n-1) + \log n \dots O(n \log n)$

The general form of these recurrence relations:

- $T(n) = aT(n-b) + f(n), a>0, b>0, f(n) = O(n^k) \text{ where } k \geq 0$

Case 1: $a=1$

$$T(n) = T(n-1) + 1 \dots O(n)$$

$$T(n) = T(n-1) + n \dots O(n^2)$$

$$T(n) = T(n-1) + \log n \dots O(n \log n)$$

Here $a = 1$, then the time complexity is in multiplication of n . So, it will be $O(n * f(n))$.

Case 2: $a > 1$

$$T(n) = 2T(n-1) + 1 \dots O(2^n)$$

$$T(n) = 3T(n-1) + 1 \dots O(3^n)$$

$$T(n) = 2T(n-1) + n \dots O(n 2^n)$$

The time complexity is $O(n^k a^n)$

Case 3: $a < 1$

- $O(n^k)$
- $O(f(n))$

1.12 Computational Problem

A computational problem is a problem that a computer might be able to solve or a question that a computer may be able to answer. The types of problems are decision problem, search problem, counting problem, optimization problem and function problem.

Decision problem: A decision problem is a computational problem where the answer for every instance is either yes or no.

Search problem: In a search problem, the answers can be arbitrary strings.

Counting problem: A counting problem asks for the number of solutions to a given search problem.

Optimization problem: An optimization problem asks for finding a "best possible" solution among the set of all possible solutions to a search problem

Function problem: In a function problem a single output (of a total function) is expected for every input, but the output is more complex than that of a decision problem, that is, it isn't just "yes" or "no".

Various strategies

- 1) Divide-and-Conquer strategy
- 2) Greedy method
- 3) Dynamic programming
- 4) Backtracking
- 5) Branch and Bound

Divide-and-Conquer strategy

Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, $1 < k < n$, yielding k sub-problems. These sub-problems must be solved, and then a method must be found to combine sub-solutions into a solution of the whole. Various problems which can be solved using divide-and-conquer strategies: binary search, merge sort, quick sort, heap sort and arithmetic with large numbers.

Greedy method

It is the most straightforward method. The problems have 'n' inputs and require us to obtain a subset that satisfies some constraints. Various problems which can be solved using greedy method are: knapsack problem, minimal spanning tree, single source shortest path

Dynamic programming

It is used to solve the optimization problem. Optimization problem results in either minimum or maximum results. Dynamic programming is used where we have problems, which can be divided into similar sub-problems so that their results can be re-used. Some problems where dynamic programming can be implemented are: chained matrix multiplication, all pair shortest path, optimal binary search tree, single source shortest path and reliability design.

Backtracking

It follows the Brute force approach. Brute Force approach says that for any given problem you should try all possible solutions and pick up the desired one. This is not for optimization problems. Backtracking is used when you have multiple solutions, and you want all of them. Some problems which can be solved using backtracking approach are: sum of subset problem, N Queens problem, graph colouring problem and Hamiltonian Cycle

Branch and Bound

Unit 01: Introduction

It follows Breadth First Search. It is similar to backtracking because it also uses state space tree for solving the problem. But it is useful for solving optimization problems, only minimization problems, not maximization problems. Branch and bound follows BFS whereas backtracking follow DFS. The problems are: job sequencing with deadlines, 0/1 Knapsack problem and travelling salesperson problem.

Other problems are pattern matching, Huffman encoding, approximation problem and Interactable problems

Summary

- Stack follows the **LIFO** operations.
- A stack can be represented by using nodes with two fields, possibly called data and link.
- Algorithms are not dependent upon hardware and OS.
- Recursion reduces the program size, makes it compact and avoids redundancy of code.
- Dynamic programming is used to solve the optimization problem.

Keywords

- **Tree:** A tree is a finite set of one or more nodes such that there is a specially designated node called the root and the remaining nodes are partitioned into $n > 0$ disjoint sets T_1, \dots, T_n , where each of these sets is a tree.
- **Binary Tree:** A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left and right subtrees.
- **Dictionaries:** An abstract data type that supports the operations insert, delete, and search is called a dictionary.
- **Algorithm:** It a step-by-step procedure for solving a computational problem.
- **Asymptotic analysis:** In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time).
- **Asymptotic notations:** Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

Self Assessment

1. In a queue, the deletions take place at _____ end.
 - A. Rear
 - B. Top
 - C. Front
 - D. None of the above

2. In a binary search tree, the elements in the right sub-tree are always _____ than the root node.
 - A. Smaller
 - B. Greater
 - C. Both of the above
 - D. None of the above

3. In a stack, the insertions and deletions are made at _____ end.
 - A. One

Algorithm Analysis and Design

- B. Two
 - C. Three
 - D. None of the above
4. One simple operation of mathematics like addition/subtraction/multiplication takes _____ step(s).
- A. One
 - B. Two
 - C. Three
 - D. Four
5. Which of these is a primitive operation?
- A. Calling a method
 - B. Comparing two numbers
 - C. Indexing into an array
 - D. All of the above
6. The RAM model of computation stands for
- A. Random Access Memory
 - B. Random Access Machine
 - C. Realtime Access Machine
 - D. None of the above
7. The RAM model of computation consists of
- A. A read-only input tape
 - B. A write-only input tape
 - C. A fixed program
 - D. All of the above
8. The function $f(n) = 2n^2 + 3n + 1$ has the time complexity as
- A. $O(n^3)$
 - B. $O(n^2)$
 - C. $O(n)$
 - D. None of the above
9. Which of these notations is upper bound of a function?
- A. Theta notation
 - B. Big Oh notation
 - C. Big Omega notation
 - D. None of the above
10. An algorithm
- A. Can be written in English sentences

- B. Not dependent upon hardware of computer
 - C. Not dependent upon operating system
 - D. All of above
11. An algorithm must have the characteristics as
- A. Effectiveness
 - B. Definiteness
 - C. Finiteness
 - D. All of the above
12. In which situation, the stack overflow occurs?
- A. Base case is not defined
 - B. Base case is not reached
 - C. Either of the above
 - D. None of the above
13. Recursion helps in
- A. Avoiding the redundancy in code
 - B. Easy maintenance of code
 - C. Compacting the code
 - D. All of the above
14. The time complexity of decreasing function is $O(n^k a^n)$ when
- A. $a = 1$
 - B. $a < 1$
 - C. $a > 1$
 - D. None of the above
15. Which of these is the most straightforward way of solving the problems?
- A. Divide and Conquer
 - B. Greedy method
 - C. Backtracking
 - D. Branch and Bound

Answers for Self Assessment

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

Review Questions

1. What are basic data structures? Explain any five of them.
2. What is a computational model? What is an algorithm? Explain two tools of analysis.
3. What is time and space complexity? Write the steps to determine the time complexity.
4. Write the detailed difference between a program and an algorithm.
5. What are the characteristics and criteria of analysis of an algorithm?
6. What is asymptotic analysis? What are asymptotic notations? Explain.
7. What is best, average, and worst case? Explain with help of an example.
8. What is recursion? What is its need? Why stack overflow occurs in recursion?



Further Readings

<https://www.geeksforgeeks.org/data-structures/>

<https://www.geeksforgeeks.org/recursion/>

<https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/>

Unit 02: Divide and Conquer

CONTENTS

- Objectives
- Introduction
- 2.1 Algorithm of divide-and-Conquer
- 2.2 Binary Search
- 2.3 Merging
- 2.4 Merge Sort
- 2.5 Quick sort
- 2.6 Arithmetic with Large Numbers
- 2.7 Binary Tree
- 2.8 Heap
- 2.9 Heap Sort
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the general method of divide-and-conquer strategy and binary search technique
- Understand the merging, merge sort process and designing of merge sort algorithm
- understand the quick sort, design, and analysis of quick sort algorithm
- understand the arithmetic with large numbers
- Understand the heap sort

Introduction

Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, $1 < k < n$, yielding k sub-problems. These sub-problems must be solved, and then a method must be found to combine sub-solutions into a solution of the whole.

2.1 Algorithm of divide-and-Conquer

Algorithm DAC(P)

```
{
if Small( $P$ ) then return  $S(P)$ ;
else
{
```

Algorithm Design and Analysis

divide P into smaller instances P_1, P_2, \dots, P_k , $k \geq 1$;

Apply DAC to each of these subproblems.

return Combine(DAC(P_1),DAC(P_2),...,DAC(P_k));

}

}

1) Whatever the problem is, the sub-problem will be same as that problem.

2) You cannot opt this strategy, if there is no method of combining the sub-solutions to make the whole solution.

Various problems which can be solved using divide-and-conquer strategy are: binary search, merge sort, quick sort and arithmetic with large numbers

2.2 Binary Search

It is a very efficient algorithm for searching of a number in a sorted array. It follows divide-and-conquer strategy.

2	3	5	8	11	13	17	19	21	25	28	33	39	47	56
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Index pointers are taken here low (l) and high (h). The calculation of mid is done by using the formula:

Mid = floor[(l + h)/2]. Key is the element to be searched.



Example 1:

2	3	5	8	11	13	17	19	21	25	28	33	39	47	56
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Here key element is 28, l=1 and h=15. Here mid is 8 after calculation. The element which you are searching for is greater than the mid-point. So, we need to look in the right list.

1	H	mid
1	15	(1+15)/2=8

In this step, the list on the left side will be freeze and we will look for element in the list at right. Here l=9 and h=15.

2	3	5	8	11	13	17	19	21	25	28	33	39	47	56
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Key = 28

1	H	mid
---	---	-----

Unit 02: Divide and Conquer

1	15	$(1+15)/2=8$
9	15	$(9+15)/2=12$

Mid =12

2	3	5	8	11	13	17	19	21	25	28	33	39	47	56
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Key = 28

1	h	mid												
1	15	$(1+15)/2=8$												
9	15	$(9+15)/2=12$												
9	11	$(9+11)/2=10$												
2	3	5	8	11	13	17	19	21	25	28	33	39	47	56
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Key = 28

1	h	mid
1	15	$(1+15)/2=8$
9	15	$(9+15)/2=12$
9	11	$(9+11)/2=10$
11	11	$(11+11)/2=11$

Total number of comparisons = 4. It shows that the binary search is faster than the linear search.

Key = 8

1	h	Mid												
1	15	$(1+15)/2=8$												
2	3	5	8	11	13	17	19	21	25	28	33	39	47	56

Algorithm Design and Analysis

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

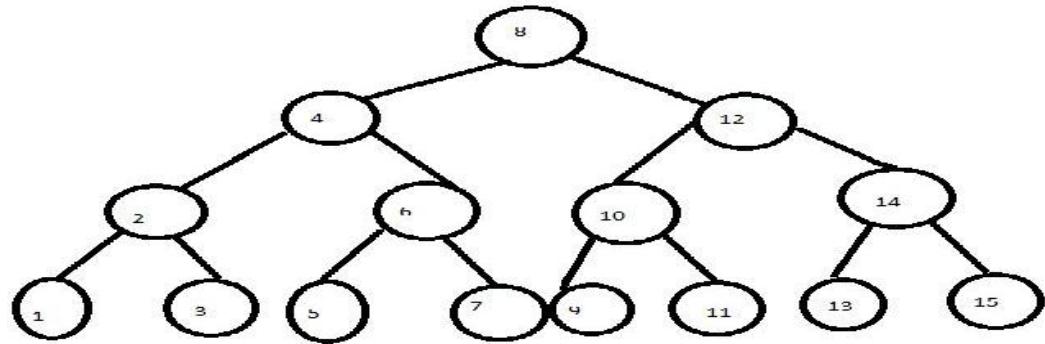
Key = 8

1	h	Mid
1	15	(1+15)/2=8
1	7	(1+7)/2=4

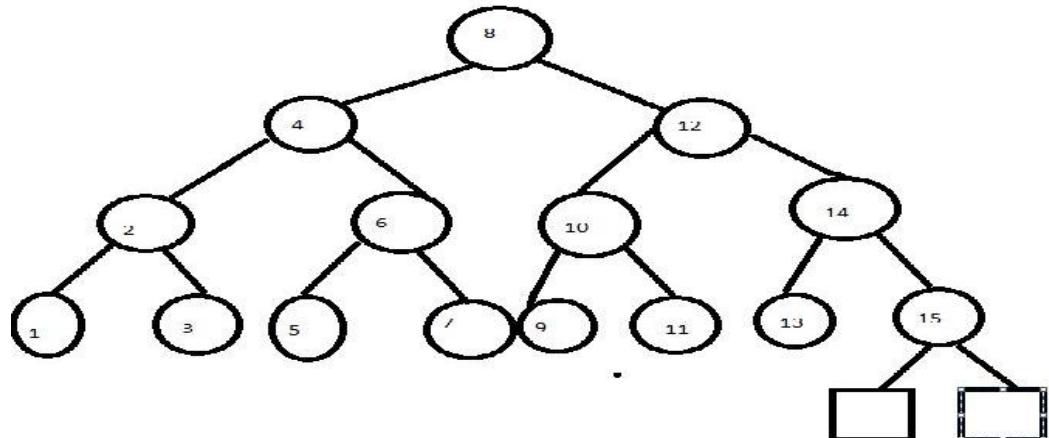
Total number of comparisons = 2.

Tracing as a tree

2	3	5	8	11	13	17	19	21	25	28	33	39	47	56
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

**Analysis**

Maximum number of comparisons depends upon the height of a tree. Height of a tree = $\log n$, Time taken by binary search = $\log n$

What if the element is not present?

If the element is not present in the list. Then again, in case of unsuccessful search, 4 comparisons are required.

Unit 02: Divide and Conquer

Minimum number of comparisons required = 1.So, minimum time = O(1). Maximum number of comparisons required = log n.So, maximum time = O(log n).

- **Best case:** The key element is located in the middle of the list and the time required is O(1).
- **Worst case:** The key element is located in the leaf of tree and the time required is O(log n).

2.3 Merging

It is a process of combining two sorted lists into a single sorted list. Two sorted lists are given, if we combine these two lists, the resulted list should be sorted. In the below example, two lists A and B are given. To manage these two lists, we need two pointers i and j. And for the merged list C we need one more pointer, i.e., k. We need to increment i and j. The process of merging is shown below.

General process of merging

List A contains the elements 2, 8, 15 and 18. List B contains 5, 9, 12 and 17. Three pointers i, j and k are taken. I and j will point towards the first element of A and B respectively.

A	B	C
2----i	5----j	----k
8	9	
15	12	
18	17	

As out of these two, 2 is the smallest one. So, 2 will come to the list C. Pointer i will points towards the next element of A and B will point towards the 5. Pointer k will also move to the next.

A	B	C
2	5---j	2
8----i	9	----k
15	12	
18	17	

According to the rule, 5 is added to C.

A	B	C
2	5	2
8----i	9----j	5----k

Algorithm Design and Analysis

15	12	
18	17	

A	B	C
2	5	2
8	9---j	5
15----i	12	8----k
18	17	

A	B	C
2	5	2
8	9	5
15----i	12---j	8
18	17	9---k

A	B	C
2	5	2
8	9	5
15----i	12	8
18	17---j	9
		12---k

Unit 02: Divide and Conquer

A	B	C
2	5	2
8	9	5
15	12	8
18----i	17---j	9
		12
		15----k

A	B	C
2	5	2
8	9	5
15	12	8
18----i	17	9
----j		12
		15
		17----k

A	B	C
2	5	2
8	9	5

Algorithm Design and Analysis

15	12	8
18	17	9
---i	---j	12
		15
		17
		18----k

Number of elements in list A = m, number of elements in list B = n. So, the time taken by list C for merging of elements of A and B are : $\Theta(m+n)$

Algorithm

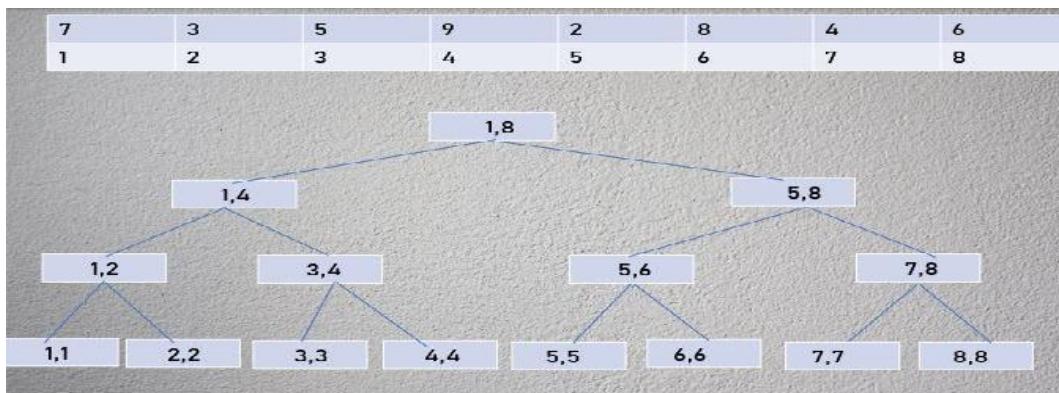
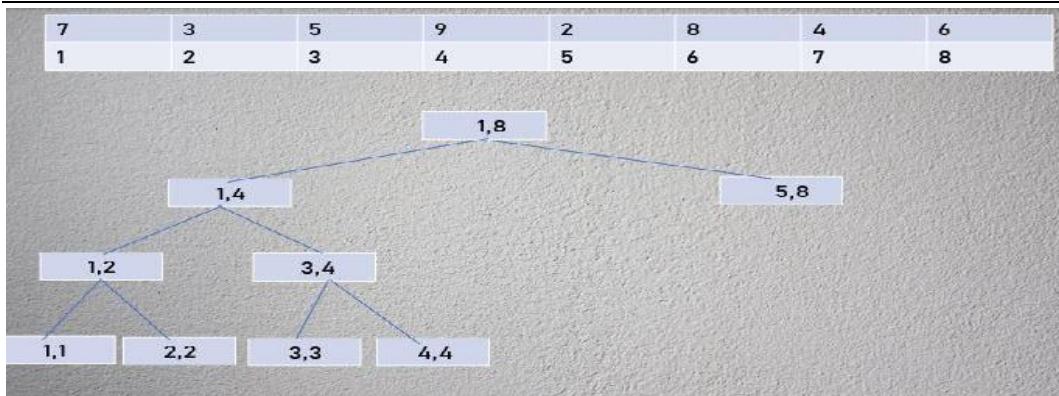
```
Algorithm merge(A,B,m,n)
{
    i=1,j=1,k=1;
    while(i<=m && j<=n)
    {
        if(A[i]<B[j])
            C[k++]=A[i++];
        else
            C[k++]=B[j++];
    }
    for(;i<=m;i++)
        C[k++]=A[i];
    for(;j<=n;j++)
        C[k++]=B[j];
}
```

2.4 Merge Sort

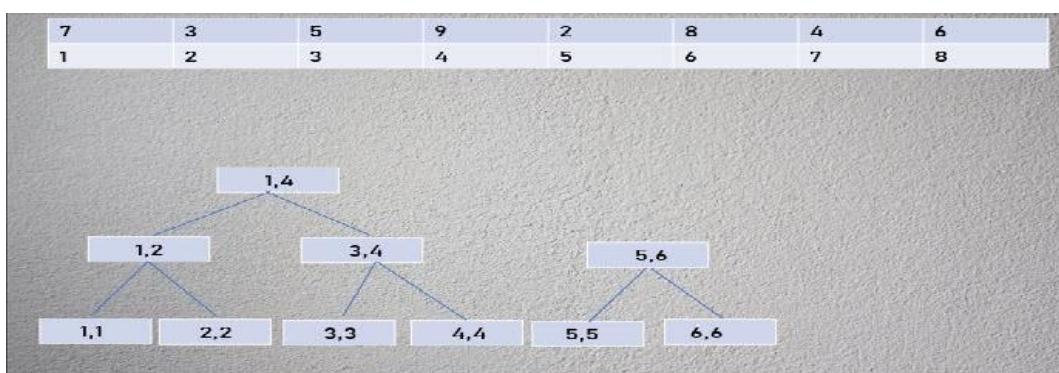
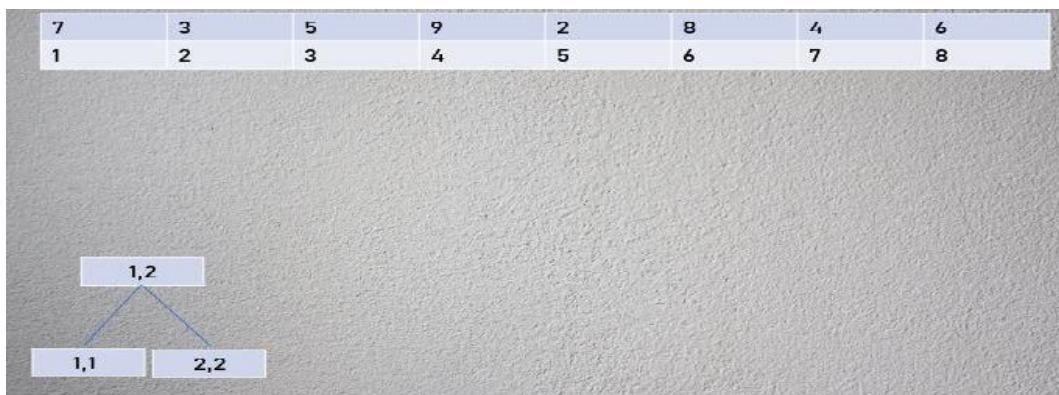
Merge sort is a recursive procedure. It is based upon the divide-and-conquer strategy. Divide-and-conquer strategy divides a large problem into small sub-problems. Small subproblem is meant if there is a single element in the list.

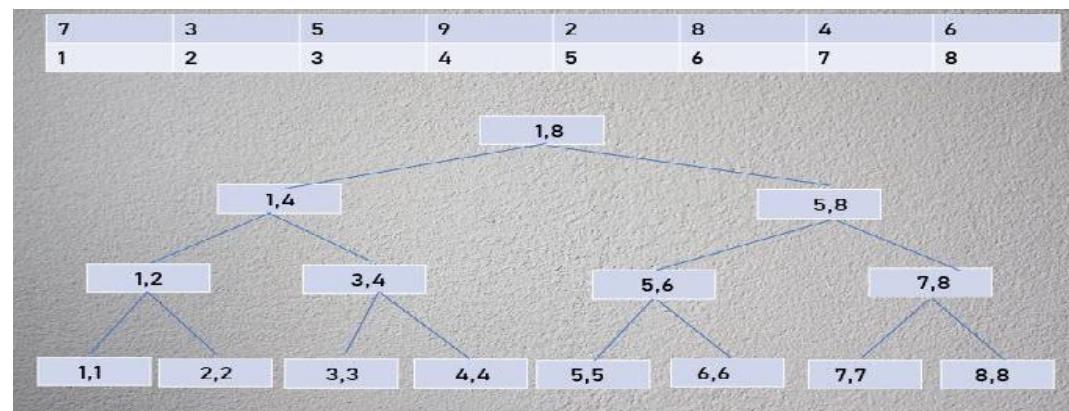
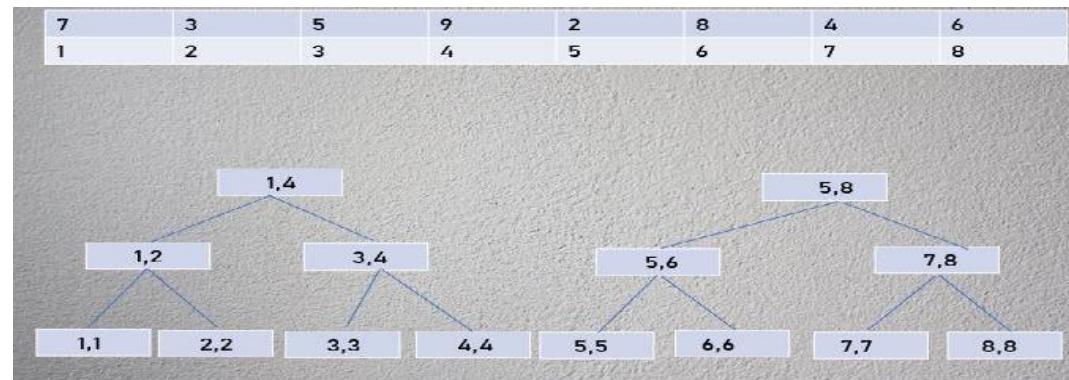
Tracing

The whole list is divided into two lists. Then each list is taken and again divided. The process continues until there is only one element in each list. Below shown is the process of merge sort.

Unit 02: Divide and Conquer

The list is now sorted. Now we need to merge the list.





Algorithm MergeSort(l,h)

```
{
if(l<h)
{
mid=(l+h)/2;
MergeSort(l,mid);
MergeSort(mid+1,h);
Merge(l,mid,h);
}
}
```

Time taken by Merge Sort

There are three levels in which the work of merging is done and at each level n elements are merged. So, for 8 elements, we got 3 levels. ($\log_2 8 = 3$). Hence the time taken by Merge sort is $\Theta(n \log n)$. The traversing is done in post order.

Time complexity using recurrence relation

Algorithm MergeSort(l,h)

```
{
if(l<h)                                just a condition
{
mid=(l+h)/2;                            1
MergeSort(l,mid);                      T(n/2)
}
```

Unit 02: Divide and Conquer

```

MergeSort(mid+1,h);           T(n/2)
Merge(l,mid,h);               n
}
}


$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$


```

Using Master's theorem

- $a=2, b=2, f(n)=n.$
- $\log_b a = \log_2 2 = 1$
- $n^k = n^1$
- $\log_2 2 = k$
- So, it will be $\Theta(n \log n).$

2.5 Quick sort

Quick sort works on the idea that an element is in the sorted position, if all the elements on the left side of that element are smaller than that and all the elements on the right side of that element are greater than that. Rest of the elements may or may not be sorted. It follows the divide-and-conquer strategy. It is recursive in nature.

Process of Quick sort

We recursively perform three steps:

- 1) Bring the pivot to its appropriate position such that on left side there are smaller elements than that and on the right side there are greater elements than that.
- 2) Quick sort the left part.
- 3) Quick sort the right part.

Recursively apply the divide-and-conquer strategy on this until all the elements are in sorted position.

Algorithm

The algorithm for quick sorting is given below.

```

QuickSort(l,h)
{
  If(l<h)
  {
    j=partition(l,h);
    QuickSort(l,j);
    QuickSort(j+1,h);
  }
}

```

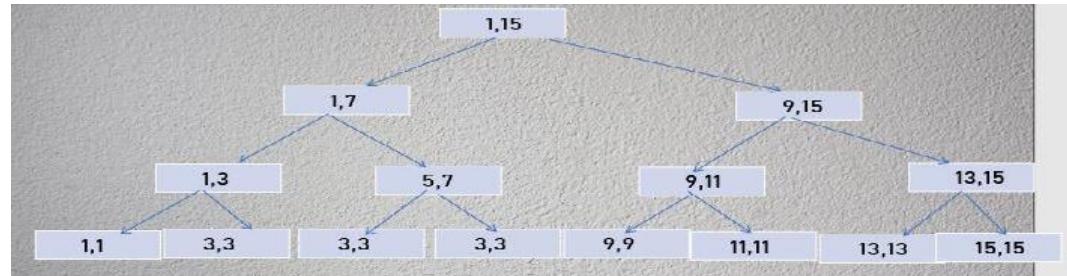
The partitioning algorithm is given below.

Algorithm Design and Analysis**Partition(l,h)**

```

{
pivot = A[l];
i=l,j=h;
while(i<j)
{
do
{
i++;
} while (A[i]<= pivot);
do
{
j--;
} while (A[j] > pivot);
if(i<j)
swap(A[i],A[j]);
}
swap(A[l], A[j]);
return j;
}

```

Tracing**Analysis**

This is the case when the partition is happening in the middle. Time taken at each level is n . So, in total it will be $(n \log n)$. Time complexity = $O(n \log n)$. This is the best case, the time complexity is $O(n \log n)$. Middle element in the sorted list. In best case, we are trying to select the median as the middle element which is not possible every time. So, achieving best case in quick sort is not possible every time. Randomly it can happen. Suppose we are trying to sort a list which is already sorted. Every time we are partitioning in the beginning.

Problems in quick sort

Achieving best case is not possible every time. In worst case, we are sorting again the sorted list. So, these are the problems in quick sort. So, the solution is to don't always select the first element. Select the middle element, then partition. So, this is how we move from $O(n^2)$ to $O(n \log n)$.

As the algorithm is recursive in nature, so no external space is required. We use the stack for this. In worst case, it may take upto n spaces.

2.6 Arithmetic with Large Numbers

Suppose x and y are two n -bit integers and assume for convenience that n is a power of 2. As a first step toward multiplying x and y , split each of them into their halves, which are $n/2$ bits long:

$$x = \boxed{x_L} \boxed{x_R} = 2^{n/2}x_L + x_R$$

$$y = \boxed{y_L} \boxed{y_R} = 2^{n/2}y_L + y_R.$$

For instance, if $x = 10110110$ then $x_L = 1011$, $x_R = 0110$, and $x = 1011 \times 2^4 + 0110$.

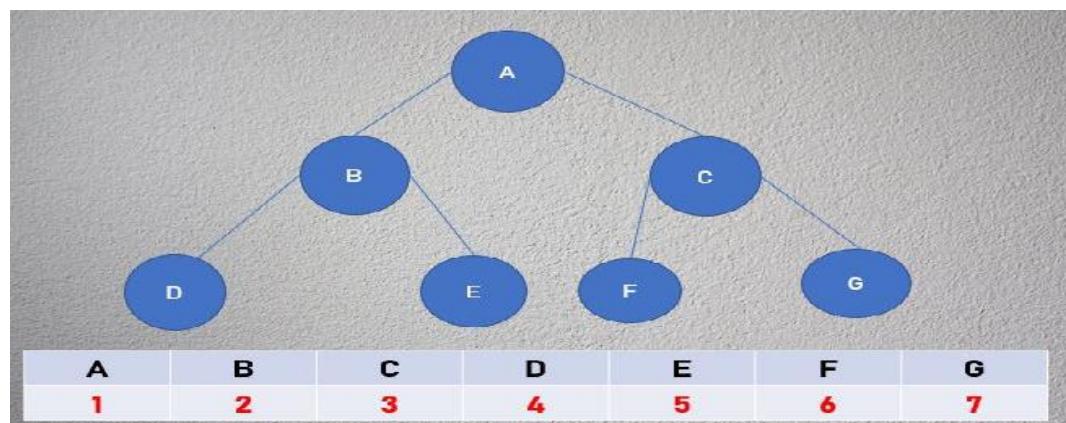
The product of x and y can then be rewritten as

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^{n/2}(x_Ly_R + x_Ry_L) + x_Ry_R.$$

2.7 Binary Tree

A **binary tree** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. See the example given below.

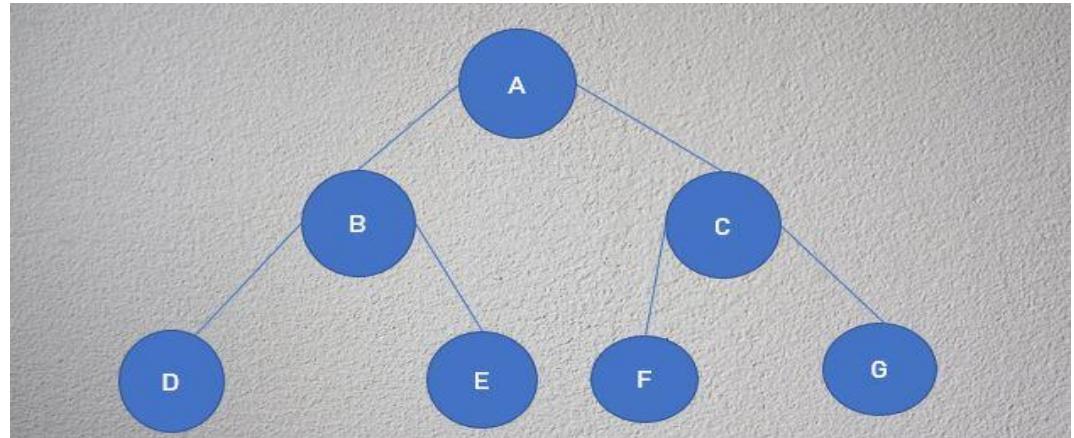
Representation of binary tree in array



Two points must be considered for storing the elements all the elements must be stored and the relationship between them must be preserved.

Relationship between elements in the array: If the node is at index i , then the left child will be at $-2*i$. and the right child will be at $-2*i + 1$. The parent will be at $(\text{floor}(i / 2))$.

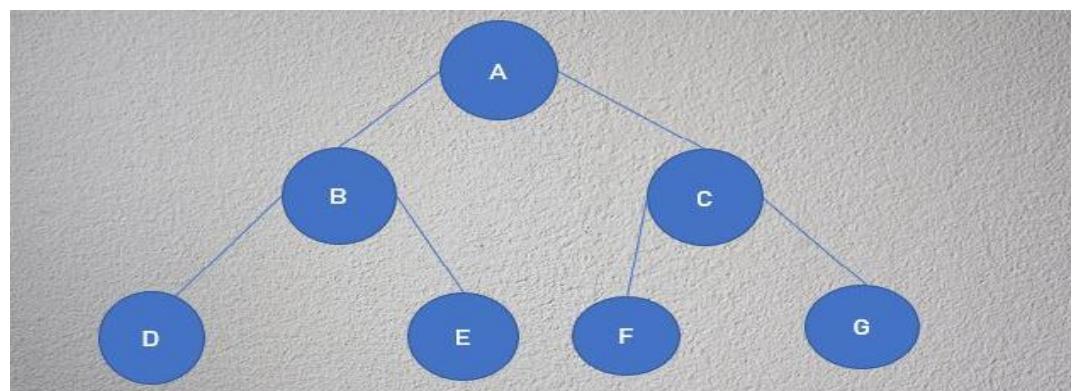
Full Binary Tree



Total height of the tree is 2. A full binary tree is a tree in which every node other than the leaves has two children. As the height of the tree is 2, in that height it is having maximum number of nodes. If we try to add any node, then the height will be increased. If height is h , then the full binary tree will have $2^{h+1} - 1$ number of nodes.

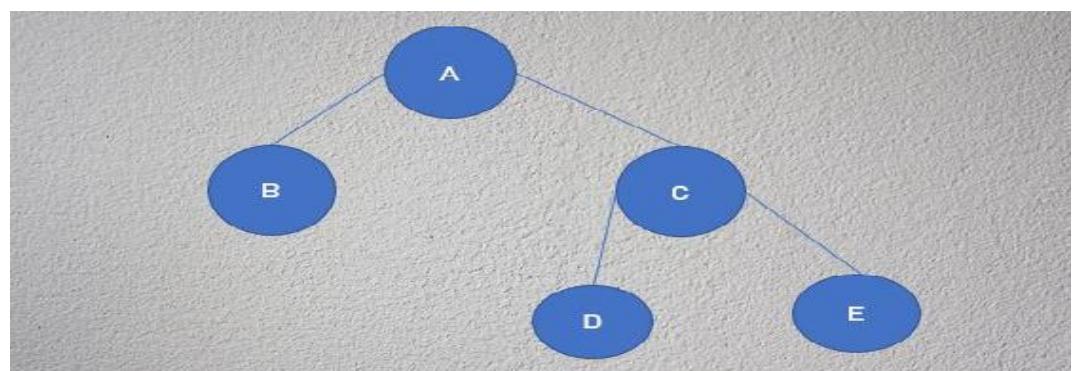
Complete Binary Tree

A binary tree in which every level (depth), except possibly the deepest, is completely filled.



A	B	C	D	E	F	G
1	2	3	4	5	6	7

Not a complete binary tree



A	B	C	--	--	D	E

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Complete and full binary trees: Every full binary tree is also a complete binary tree. If we go level by level, then there should be no missing node. A complete binary tree of height h is a full binary tree up to level h-1.

Height of complete binary tree

The height of the complete binary tree will be minimum only that is $\log n$.

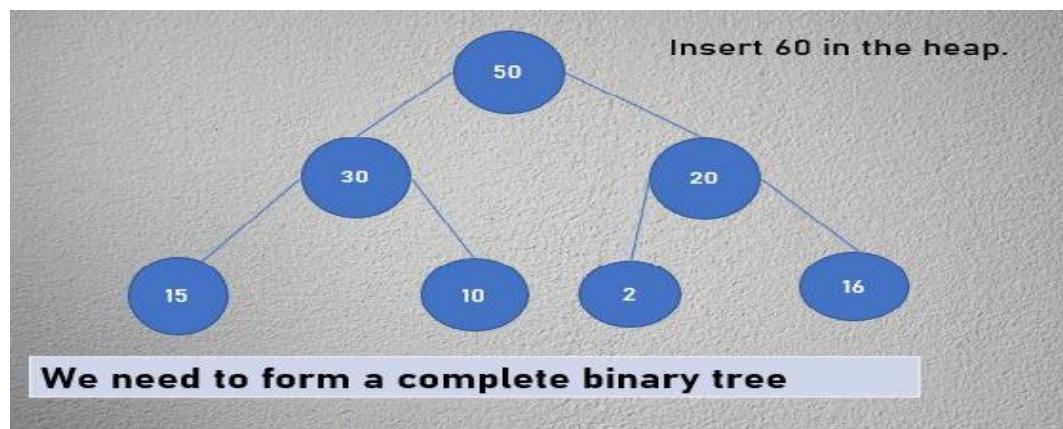
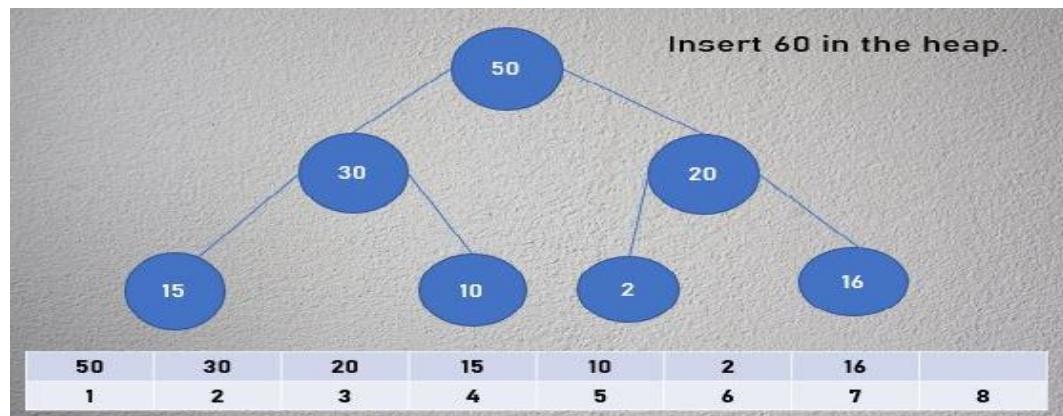
2.8 Heap

Heap is a complete binary tree.

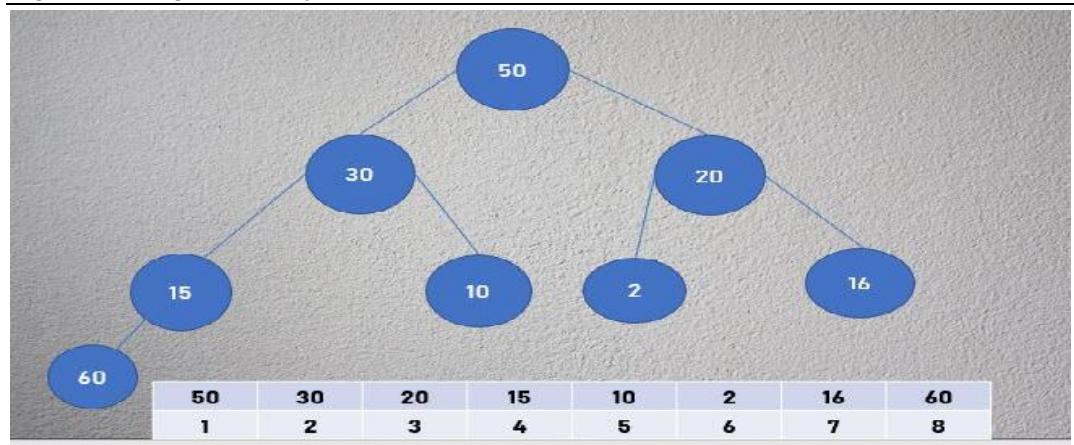
Types of heap

- **Max heap:** Max heap is a complete binary tree in which every node is having the element greater than or equal to its descendants.
- **Min heap:** Min heap is a complete binary tree in which every node is having the element smaller than or equal to its descendants.

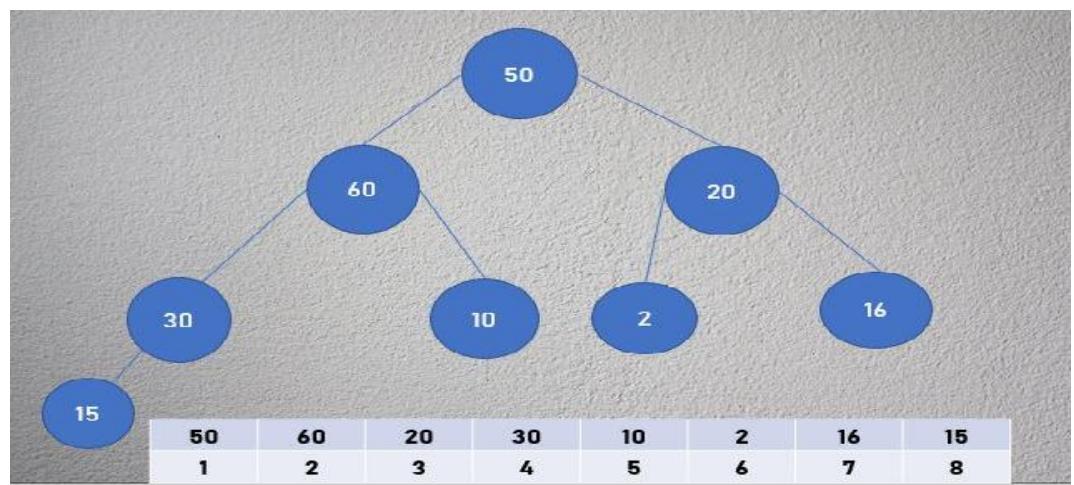
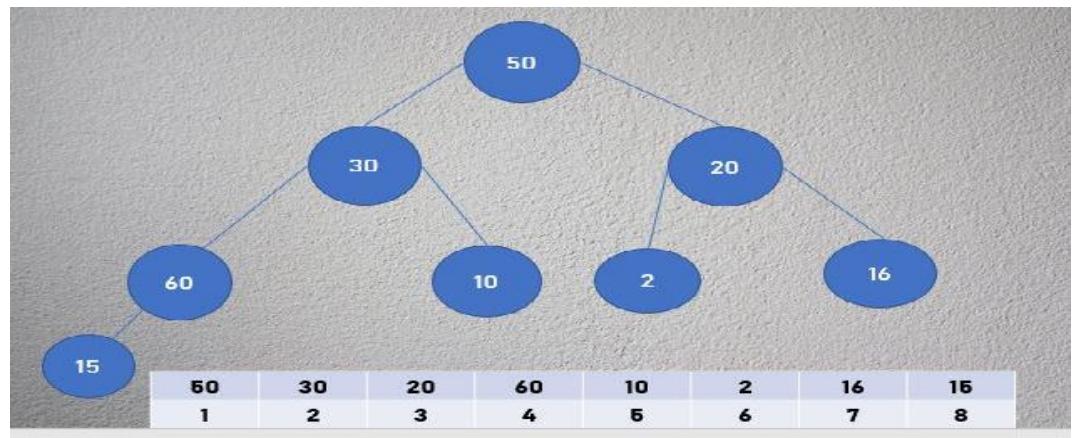
Insert operation in max heap

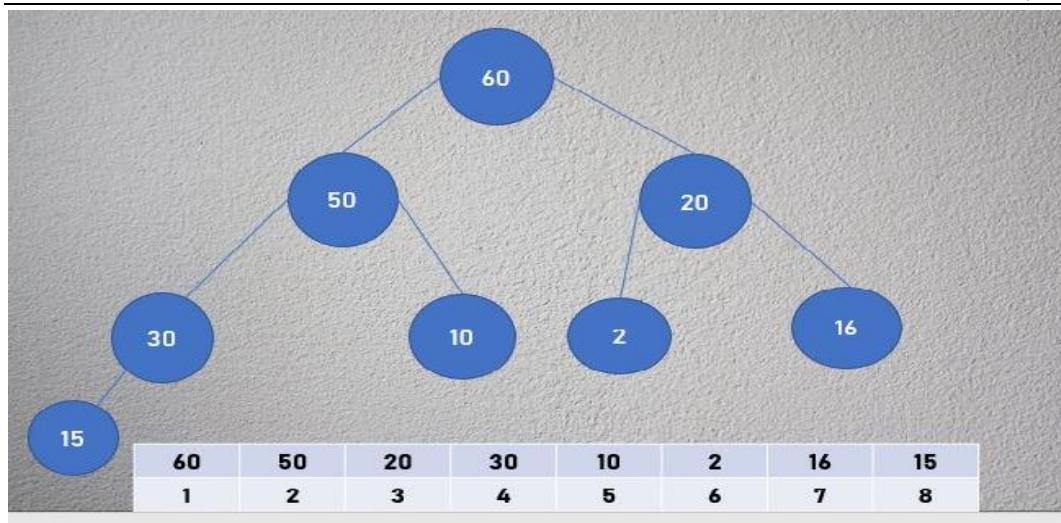


We have represented the heap in the form of an array and in the array, the empty location is at the end. So, 60 will be the left child of 15 initially.



It is not forming a heap because the condition is violated here. So, we need to adjust the nodes. Compare it with its parent (all its ancestors).



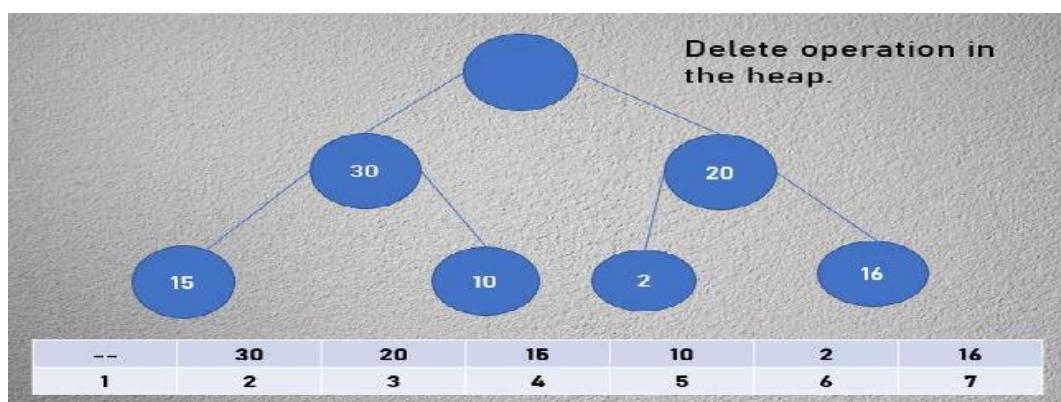
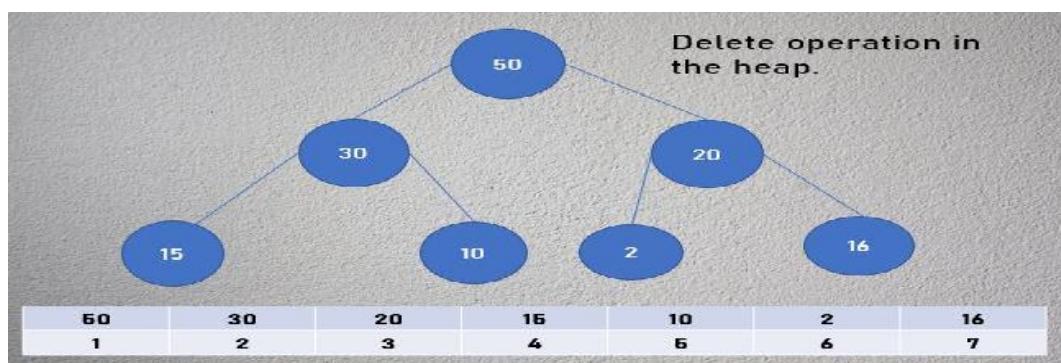


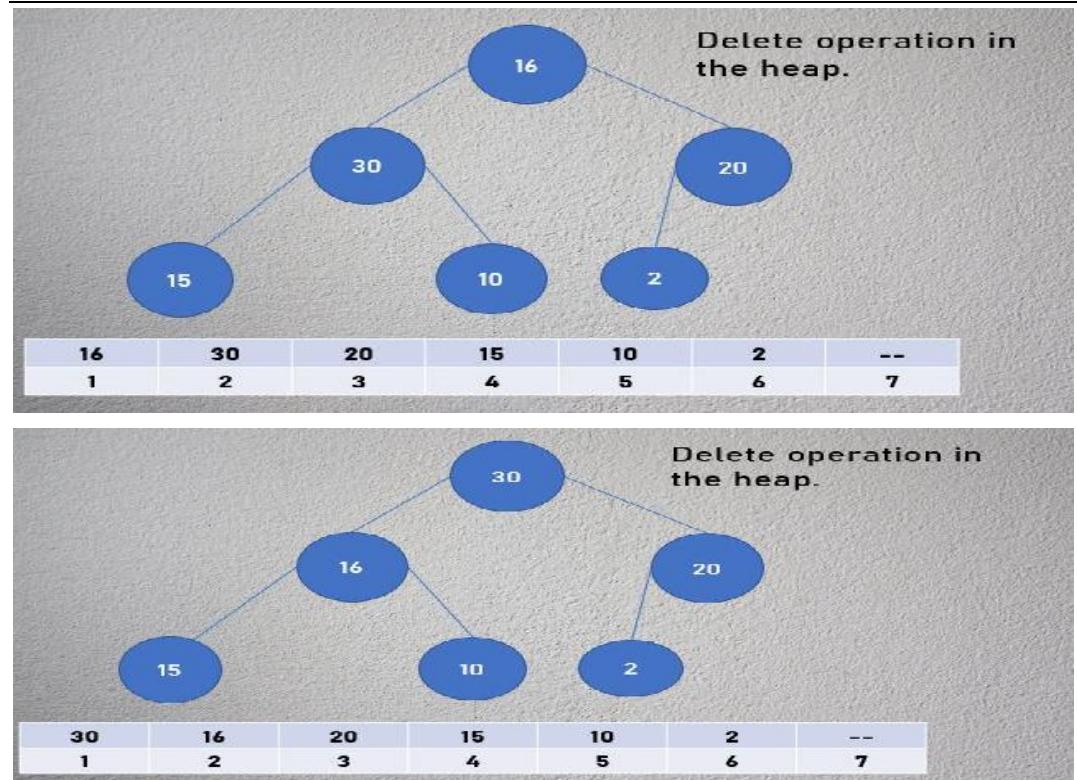
Time taken for insertion

The time taken = no of swaps. The number of swaps depends upon the height of the tree ($\log n$). So, time taken for insertion = $O(\log n)$. If this element is 6 only, then we need not to swap anything. So, the time will be $O(1)$. So, the time taken can be $O(1)$ to $O(\log n)$.

Delete operation in max heap

Only root element can be deleted from the heap. While deletion, make sure it looks like a complete binary tree. When the root element is deleted, the element at the last should take the place of root element and then the adjustment will be made.





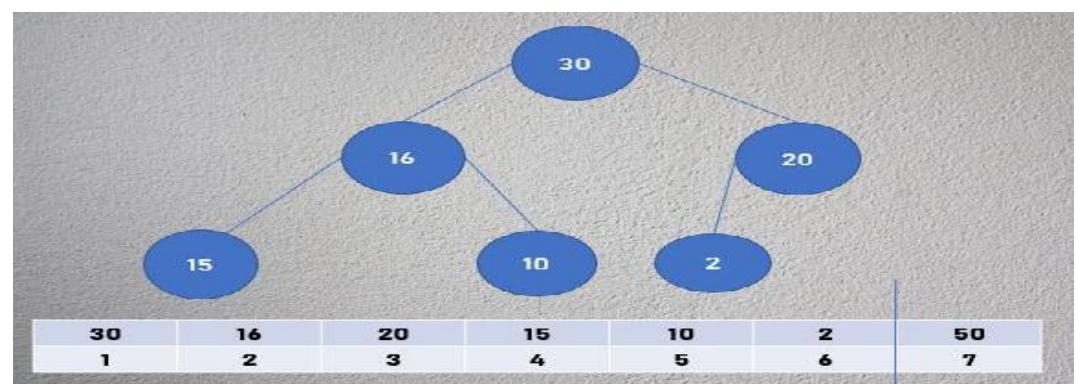
Time taken for deletion

- Time taken is depends upon the height of tree. Maximum time taken is $\log n$.

Insert and deletion operation

- In insertion operation, the adjustment was done from leaf to root.
- In deletion operation, the adjustment was done from root to leaf.

A free space at the end is used for preserving a copy of the deleted item.



Unit 02: Divide and Conquer

Next element deleted will be 30.

30	50
6	7

Next element deleted will be 30.

20	30	50
5	6	7

Next element deleted will be 30.

16	20	30	50
4	5	6	7

Next element deleted will be 30.

15	16	20	30	50
3	4	5	6	7

Next element deleted will be 30.

10	15	16	20	30	50
2	3	4	5	6	7

Next element deleted will be 30.

2	10	15	16	20	30	50
1	2	3	4	5	6	7

The idea behind the heap sort is that delete the elements and put them in the empty locations, the elements will be sorted.

2.9 Heap Sort

It contains two steps:

- 1) For given set of elements, create a heap by inserting all elements one by one.
- 2) Once the elements are inserted, delete all the elements from the heap one by one.

Insertion of elements in the heap

The set of elements given are: 10, 20, 15, 30 and 40.

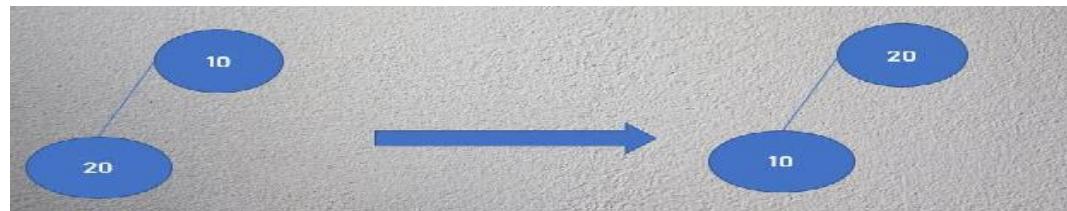
- **STEP 1:** Create a heap. Insert element 10 in the heap.

10				
1	2	3	4	5



We have only element 10 in the heap. So, we can call this as min-heap or max-heap. Insert 20 in the heap. Adjust it by comparing it to its ancestors.

20	10			
1	2	3	4	5

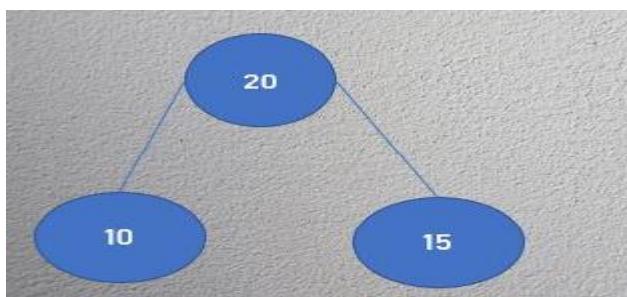


Insert 15 in the heap. Adjust it by comparing it to its ancestors.

20	10	15		
----	----	----	--	--

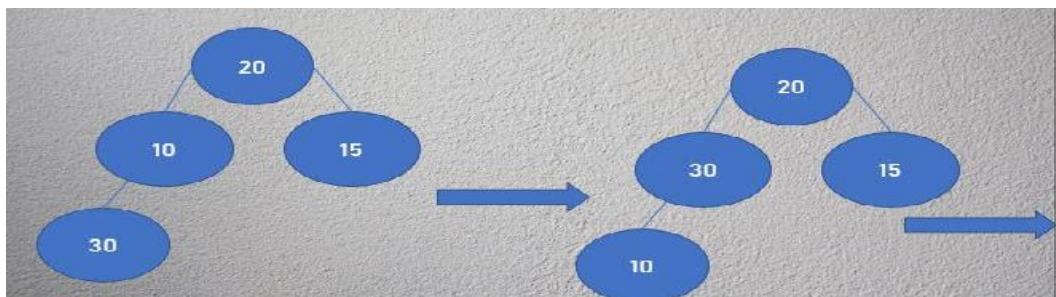
Unit 02: Divide and Conquer

1	2	3	4	5
---	---	---	---	---



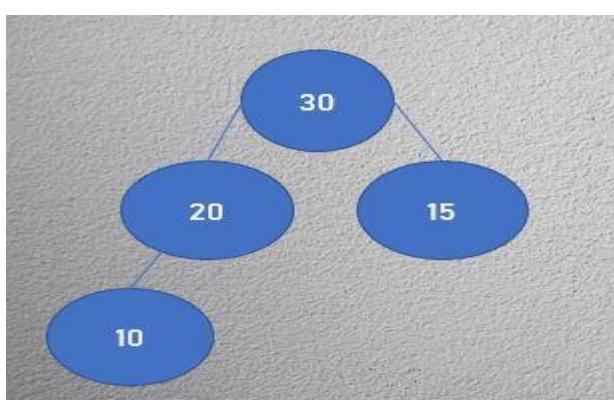
Insert 30 in the heap and adjust it by comparing it to its ancestors.

20	30	15	10	
1	2	3	4	5



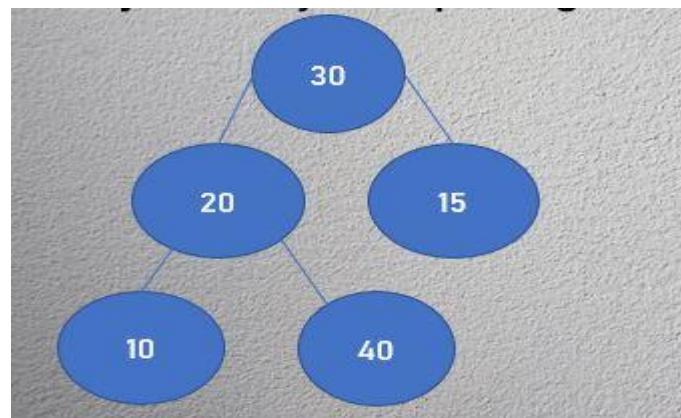
30	20	15	10	
1	2	3	4	5

Adjust it by comparing it to its ancestors



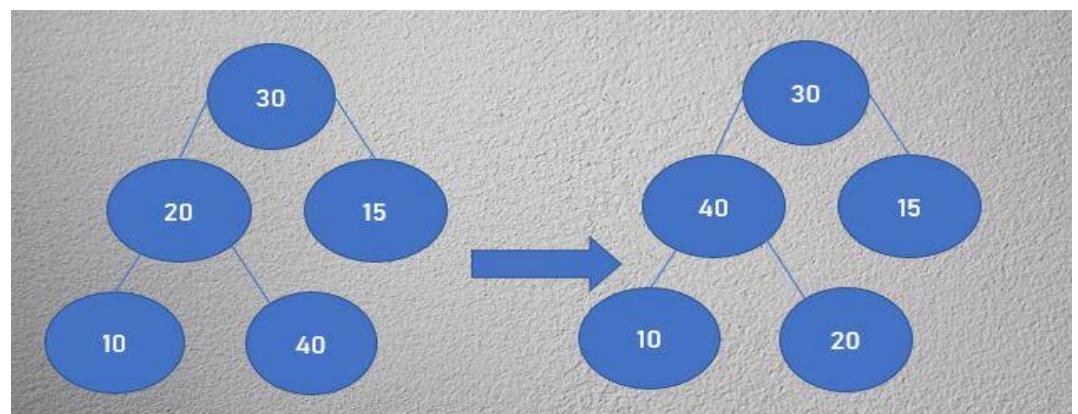
Insert 40 in the heap. Adjust it by comparing it to its ancestors.

30	20	15	10	40
1	2	3	4	5

Algorithm Design and Analysis

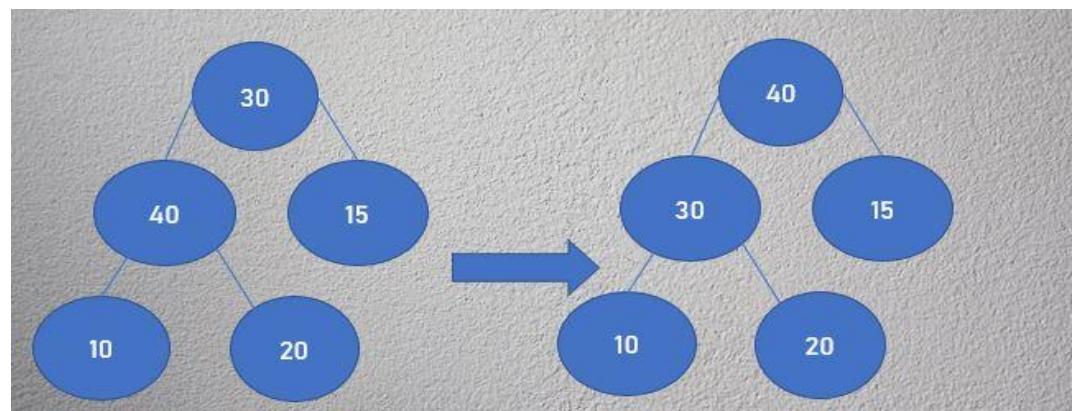
30	40	15	10	20
1	2	3	4	5

Adjust it by comparing it to its ancestors



40	30	15	10	20
1	2	3	4	5

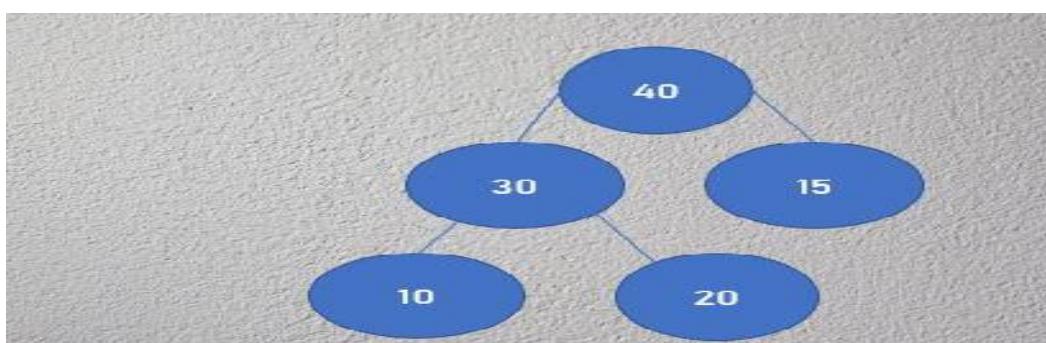
Adjust it by comparing it to its ancestors



40	30	15	10	20
1	2	3	4	5

Unit 02: Divide and Conquer

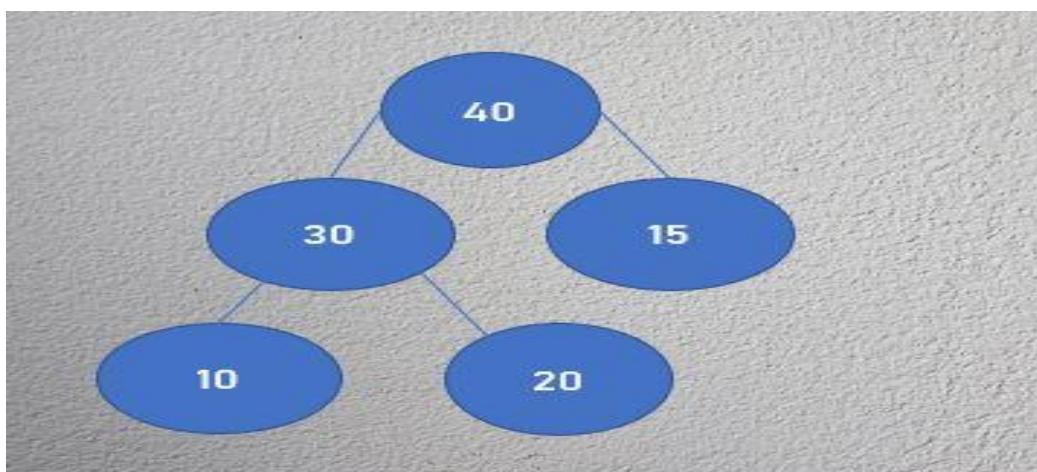
1	2	3	4	5
---	---	---	---	---

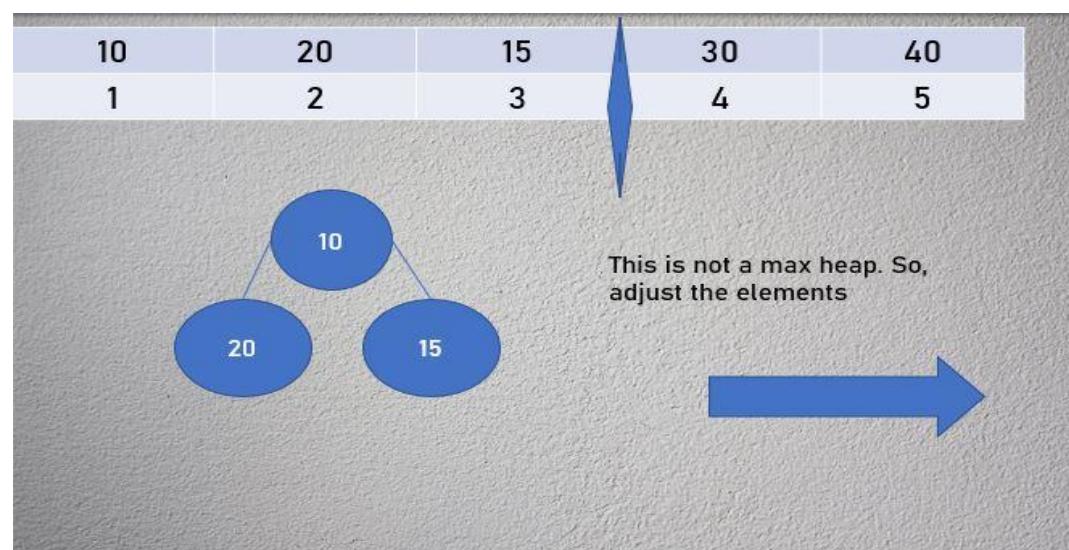
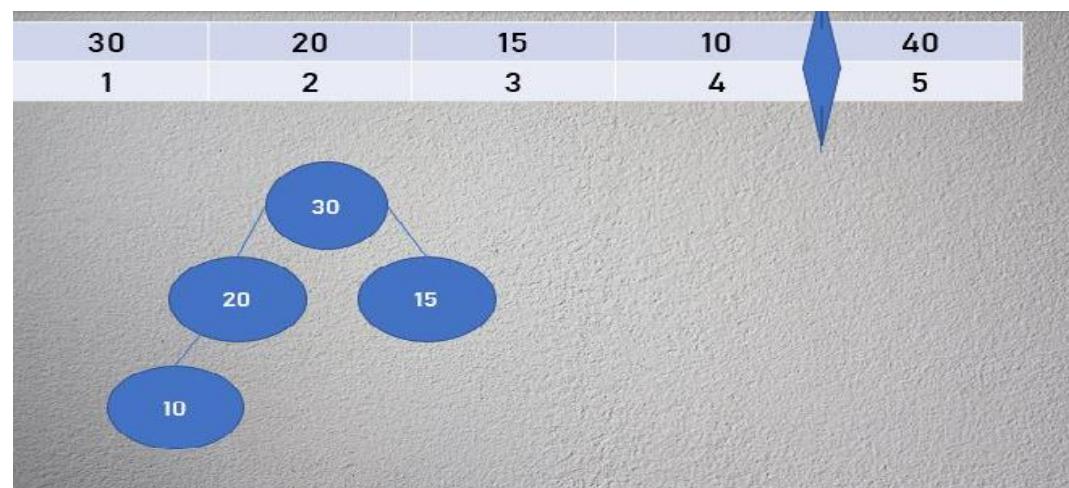
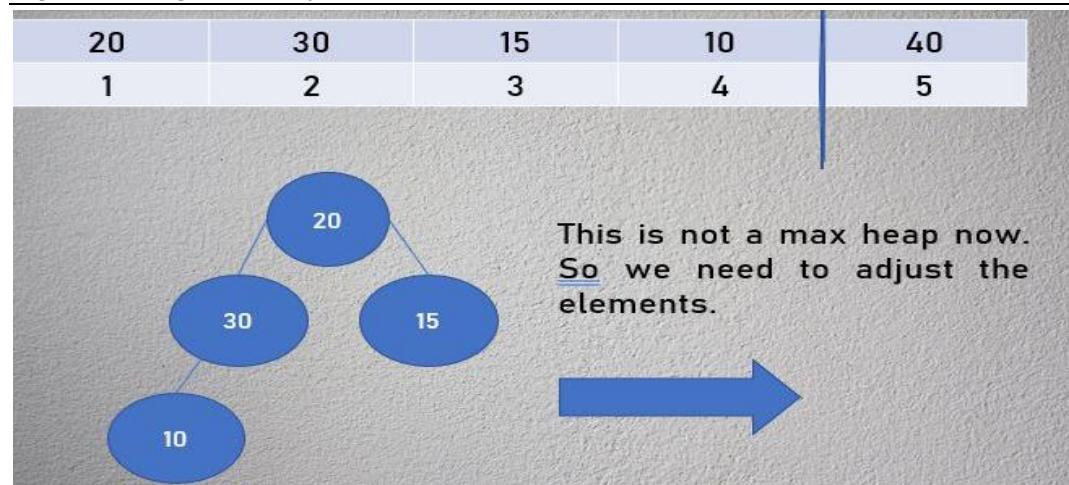
**Time taken in insertion**

Total n elements are inserted. Time taken is dependent upon the height of a tree which is $\log n$. So, the time taken in insertion is $n \log n$.

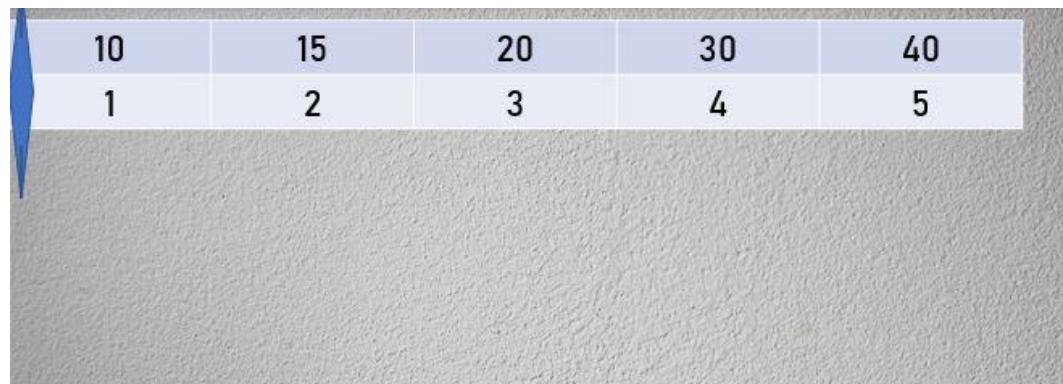
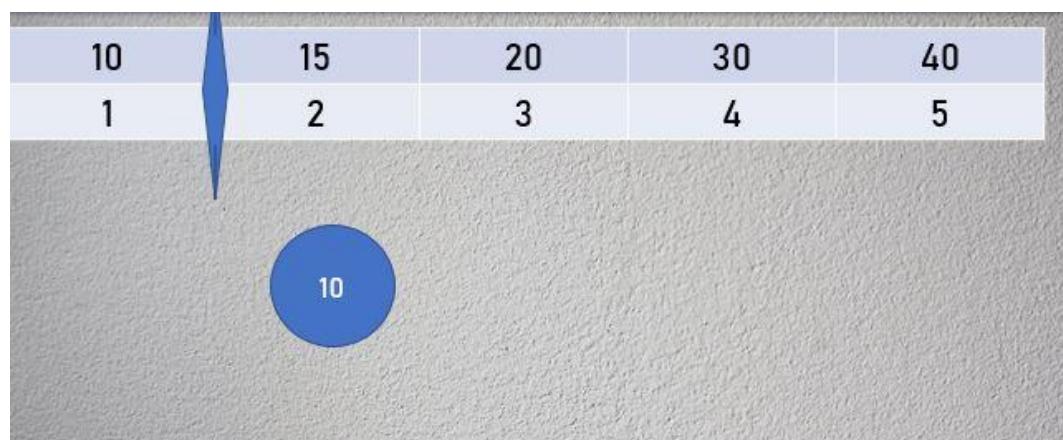
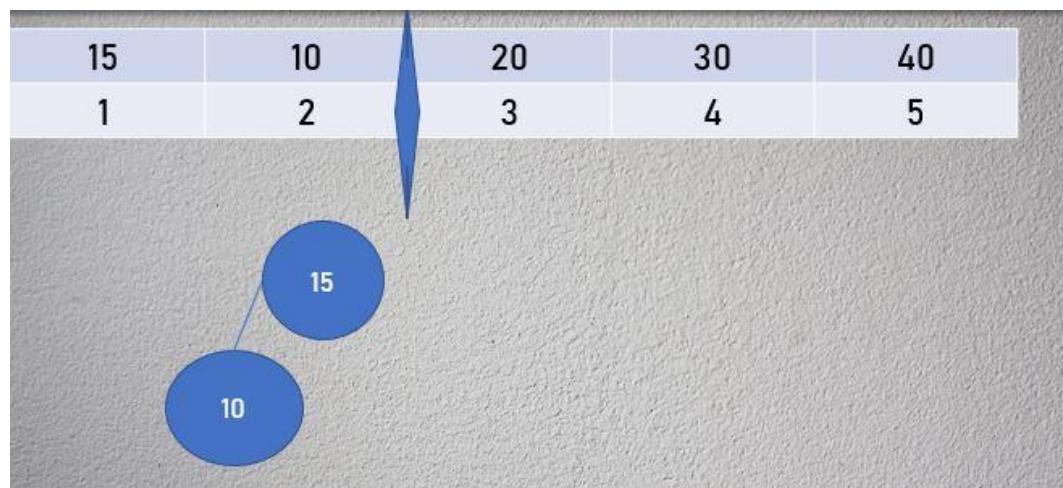
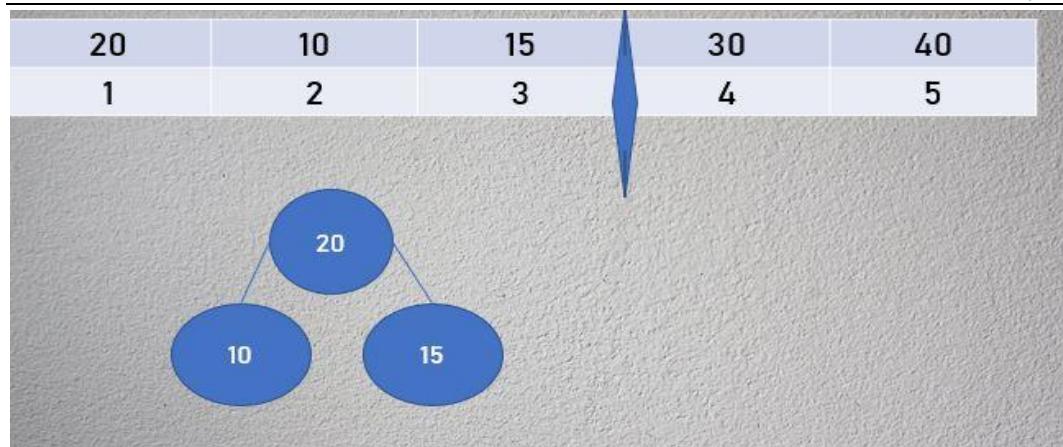
Deletion of elements from heap

40	30	15	10	20
1	2	3	4	5



Algorithm Design and Analysis

Unit 02: Divide and Conquer



Algorithm Design and Analysis

Time taken in deletion of elements from the heap

There are n elements which we are deleting. Each element taking $\log n$ time. So, total time in deletion is $n \log n$.

Total time taken in heap sort:

The time taken in insertion is $n \log n$. The time taken in deletion is $n \log n$. Total time taken is $2n \log n$. The time complexity is $O(n \log n)$.

Summary

- The reapplication of divide-and-conquer principle is expressed by recursive algorithm.
- The sub-problems are of the same type as of original problem in divide and conquer strategy.
- The time taken in merging of elements of lists A and B are : $\Theta(m+n)$.
- Time taken by Merge sort is $\Theta(n \log n)$.
- Quick sort follows the divide-and-conquer strategy.
- A complete binary tree of height h is a full binary tree up to level h-1.

Keywords

- **Binary search:** It is a very efficient algorithm for searching of a number in a sorted array. It follows divide-and-conquer strategy.
- **Merging:** It is a process of combining two sorted lists into a single sorted list.
- **Merge sort:** It is a recursive procedure. It is based upon the divide-and-conquer strategy.
- **Complete binary tree:** A binary tree in which every level (depth), except possibly the deepest, is completely filled.
- **Max heap:** Max heap is a complete binary tree in which every node is having the element greater than or equal to its descendants.
- **Min heap:** Min heap is a complete binary tree in which every node is having the element smaller than or equal to its descendants.

Self Assessment

1. What can be done using divide and conquer strategy?
 - A. Merge sort
 - B. Binary search
 - C. Quick sort
 - D. All of the above

2. How do we find the mid in binary search?
 - A. Mid = ceil [(l + h)/ 2]
 - B. Mid = floor [(l + h)/ 2]
 - C. Mid = square [(l + h)/ 2]
 - D. None of the above

3. The best case time complexity of binary search is
 - A. $O(1)$
 - B. $O(\log n)$

Unit 02: Divide and Conquer

- C. $O(n)$
 - D. $O(n \log n)$
4. The time complexity of merge sort is _____.
- A. $O(n)$
 - B. $O(1)$
 - C. $O(\log n)$
 - D. $O(n \log n)$
5. Traversing of elements in merge sort is done in _____.
- A. Preorder
 - B. Postorder
 - C. Inorder
 - D. None of the above
6. In merge sort, the list is considered as small, when it is having ____ element.
- A. 0
 - B. 1
 - C. 2
 - D. 3
7. The time complexity of quick sort is
- A. $O(n)$
 - B. $O(\log n)$
 - C. $O(n \log n)$
 - D. None of the above
8. Which data structure is used for sorting of elements in quick sort?
- A. Stack
 - B. Queue
 - C. Tree
 - D. Graph
9. In quick sort, if we are sorting a sorted list, then it is _____.
- A. Best case
 - B. Average case
 - C. Worst case
 - D. None of the above
10. The idea behind the quick sort is, the elements on the left side of pivot must be _____ and the elements on the right side of pivot must be _____.
- A. Greater, smaller
 - B. Smaller, greater

Algorithm Design and Analysis

- C. Smaller, smaller
- D. Greater, greater

11. How many index pointers are required for quick sorting?
 - A. 0
 - B. 1
 - C. 2
 - D. 3
12. In a binary tree, if the node is at index i , then the parent of that node will be at
 - A. 2^*i
 - B. 2^*i+1
 - C. Flr ($i/2$)
 - D. Ceil ($i/2$)
13. The number of nodes in a binary tree of height h will be
 - A. $2^{h+1} - 1$
 - B. $2^h - 1$
 - C. $2^{h-1} - 1$
 - D. None of the above
14. In best case, the time taken for insertion of element in heapsort is
 - A. $O(1)$
 - B. $O(\log n)$
 - C. $O(n)$
 - D. $O(2)$
15. In a binary tree, if the node is at index i , then the left child will be at
 - A. 2^*i
 - B. 2^*i+1
 - C. 3^*i
 - D. 3^*i+1

Answers for Self Assessment

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

Review Questions

1. What is dividing-and conquer strategy? Explain how this strategy is applied in binary search?
2. What is merging and merge sort? Write the analysis part.
3. What is the idea behind quick sorting? Explain its process.
4. Write the algorithm for quick sort and its partitioning. Explain it with the help of an example.
5. What is arithmetic with large number? Explain with example.
6. What is a binary tree? Explain difference between full and complete binary trees.
7. What is a heap? Explain its types with diagrams.
8. Explain the insert and delete operations in maxheap and minheap.



Further Readings

<https://www.javatpoint.com/divide-and-conquer-introduction>

<https://www.programiz.com/dsa/merge-sort>

<https://www.javatpoint.com/quick-sort>

<https://www.interviewbit.com/tutorial/heap-sort-algorithm/>

Unit 03: Greedy Method

CONTENTS

- Objectives
- Introduction
- 3.1 Knapsack Problem
- 3.2 Graph
- 3.3 Spanning Tree
- 3.4 Prim's Algorithm
- 3.5 Kruskal's Algorithm
- 3.6 Single Source Shortest Path
- 3.7 Dijkstra Algorithm
- 3.8 Optimal Storage on Tapes
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- understand the greedy method and Knapsack problem
- Understand the minimal spanning tree
- Understand the Prim's and Kruskal's algorithms
- Understand the single source shortest algorithm
- Understand the optimal storage on tapes problem

Introduction

There are various approaches for solving the problems. In previous unit, we discussed about the divide and conquer strategies. This strategy works on dividing the problem into subproblems, then each of the subproblem and combining all the sub solutions to form the solution of the bigger problem. Apart from divide and conquer strategy, we have some other strategies which works differently. The next approach which is used to solve the problems is greedy approach. It is the most straightforward method. For getting the solution of a problem, we have various ways but in greedy method we just consider that subset of solution which satisfies some constraints. Any subset that satisfies those constraints is called a feasible solution. We need to find a feasible solution that either maximizes or minimizes a given objective function. A feasible solution that does this is called an optimal solution.

Algorithm of greedy method

The greedy method suggests that one can devise an algorithm that works in stages, considering one input at a time. The algorithm for greedy method is:

Algorithm Design and Analysis

```

AlgorithmGreedy(a,n)
// a[1:n] contains the n inputs.
{
solution:=0;// Initialize the solution.
for i :=1 to n do
{
x :=Select(a);
if Feasible(solution, x) then
solution:=Union (solution,x);
}
return solution;
}

```

Problems to be solved

There are various problems which can be solved using the greedy approach. These are:

1. Knapsack problem
2. Minimal spanning tree
3. Single source shortest path
4. Optimal storage on tapes

3.1 Knapsack Problem

The Knapsack problem is stated as: Several objects are given, and every object has some profit and weight associated with it. This knapsack is a bag or container with some defined capacity. In this problem, we must fill this bag with these objects, and we will carry this bag to some other place and sell the objects. Based upon that we will get the profit. Now if we think, there can be different ways by which we can fill the knapsack. But in greedy method, we need to satisfy some constraint. Only those solutions will be taken which satisfies this constraint. In Knapsack problem, constraint here is that the total weight should be less than or equal to some predefined value. But our objective is to take those values which maximizes the results. So, in the given problem, we have few objects and the associated weights and profits.



Example 1:

Objects (O)	1	2	3	4	5	6	7
Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1

So, first we need to find P/W first. This P/W states what profit is associated with each unit of item.

Objects (O)	1	2	3	4	5	6	7

Unit 03: Greedy Method

Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3

$0 \leq x \leq 1$. The value of x can be between 0 and 1. It means we can take any fraction of item. It is not mandatory to take the whole item.

If we see in the above table. It is clearly stating that the maximum P/W is given by object 5. So, this will be included first.

Objects (O)	1	2	3	4	5	6	7
Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
					1		

After this the remaining capacity is $15-1 = 14$.

The next greater P/W is given by object 1. We have 2 units of object 2.

Objects (O)	1	2	3	4	5	6	7
Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
	1				1		

After this the remaining capacity is $14-2=12$.

The next greater P/W is given by object 6. We have 4 units of object 6.

Objects (O)	1	2	3	4	5	6	7
Profit (P)	10	5	15	7	6	18	3

Algorithm Design and Analysis

Weights (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
	1				1	1	

After this the remaining capacity is $12-4=8$.

The next greater P/W is given by objects 3. We have 5 units of object 3.

Objects (O)	1	2	3	4	5	6	7
Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
	1		1		1	1	

After this the remaining capacity is $8-5=3$.

The same P/W is given by object 7. We have 1 unit of object 7.

Objects (O)	1	2	3	4	5	6	7
Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
	1		1		1	1	1

After this the remaining capacity is $3-1 = 2$.

The next higher P/W is given by object 2. We have 3 unit of object 2. We can't take the whole object2 as it will exceed the capacity of the bag. We will just take 2 units out of 3.

Objects (O)	1	2	3	4	5	6	7
-------------	---	---	---	---	---	---	---

Unit 03: Greedy Method

Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
	1	2/3	1		1	1	1

After this the remaining capacity is $2-2 = 0$. The capacity of the bag is over now. We can't include object 4 in the bag. So it will be 0.

Objects (O)	1	2	3	4	5	6	7
Profit (P)	10	5	15	7	6	18	3
Weights (W)	2	3	5	7	1	4	1
P/W	5	1.6	3	1	6	4.5	3
x	1	2/3	1	0	1	1	1

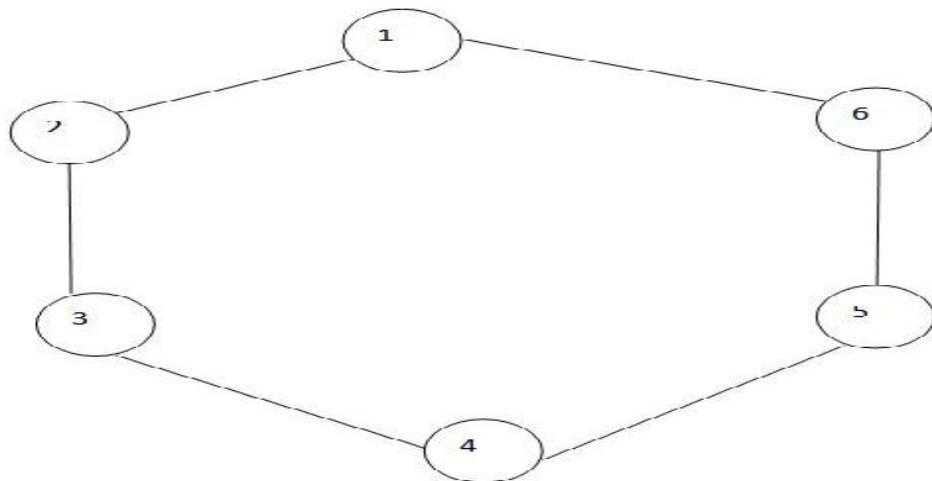
$$\begin{aligned}\Sigma x_i W_i &= 1*2 + 3*2/3 + 5*1 + 7*0 + 1*1 + 4*1 + 1*1 \\ &= 2+2+5+1+4+1 \\ &= 15\end{aligned}$$

The profit is calculated as:

$$\begin{aligned}\Sigma x_i P_i &= 1*10 + 2/3*5 + 1*15 + 0*7 + 1*6 + 1*18 + 1*3 \\ &= 10 + 2.6 + 15 + 6 + 18 + 3 \\ &= 54.6\end{aligned}$$

3.2 Graph

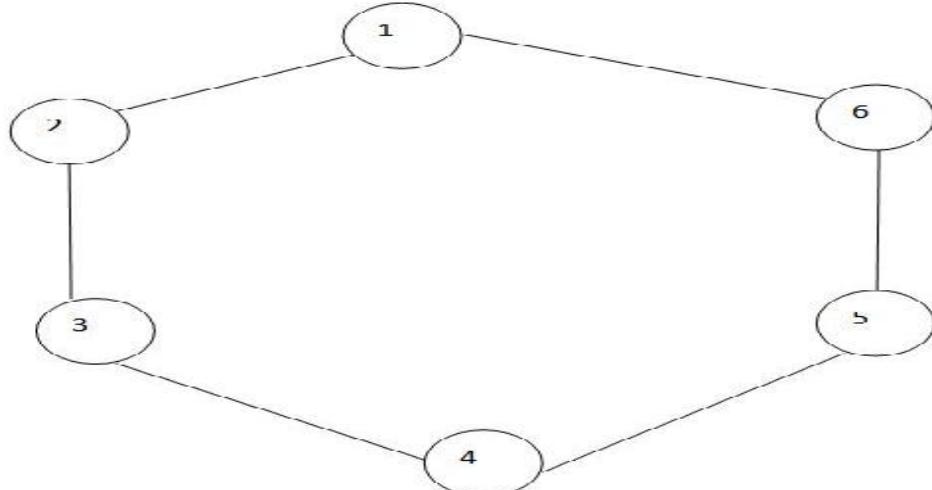
Graph is a mathematical representation of a network, and it describes the relationship between vertices and edges. A graph consists of some vertices and edges between them. The length of the edges and position of the vertices do not matter. Given below one example of a graph.



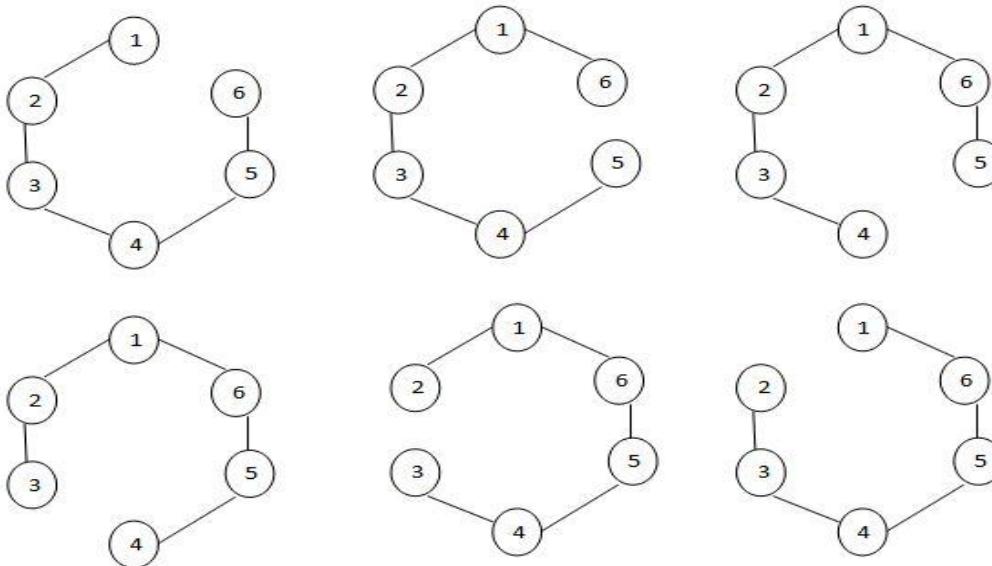
So, a graph is given with two things, i.e., $G = (V, E)$. Here in the example graph. $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1,2), (2,3), (3,4), (4,5), (5,6), (6,1)\}$. From a graph, we need to find the minimal spanning tree.

3.3 Spanning Tree

A spanning tree is a subset of Graph G , which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected. By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A spanning tree is sub-graph of a graph which has all the vertices; the number of edges will be less one than the number of vertices. If the given graph is:



Then the spanning trees will be



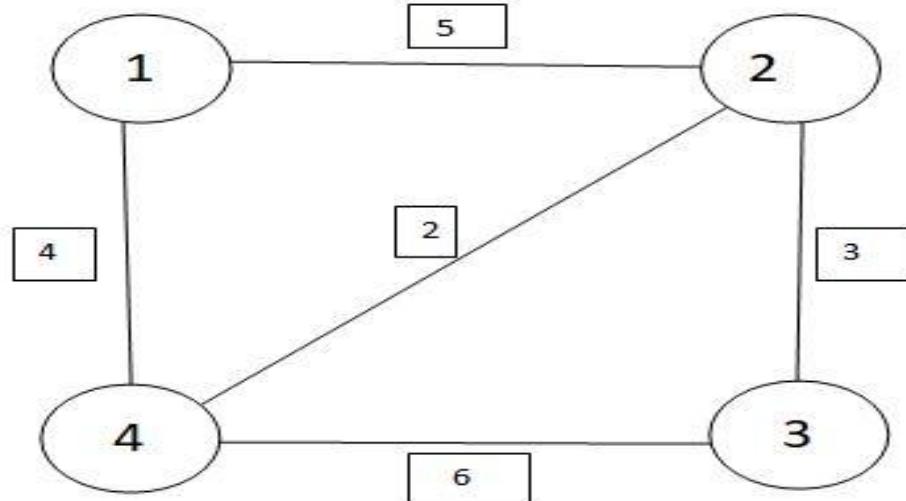
So, spanning tree is defined as:

- $S \subseteq G$. It represents that spanning tree is a subset of the graph.
- $S = (V', E')$. It contains a subset of vertices and edges.
- $V' = V$. The number of vertices will be same.
- $E' = |V| - 1$. The number of edges will be less one than the number of vertices.

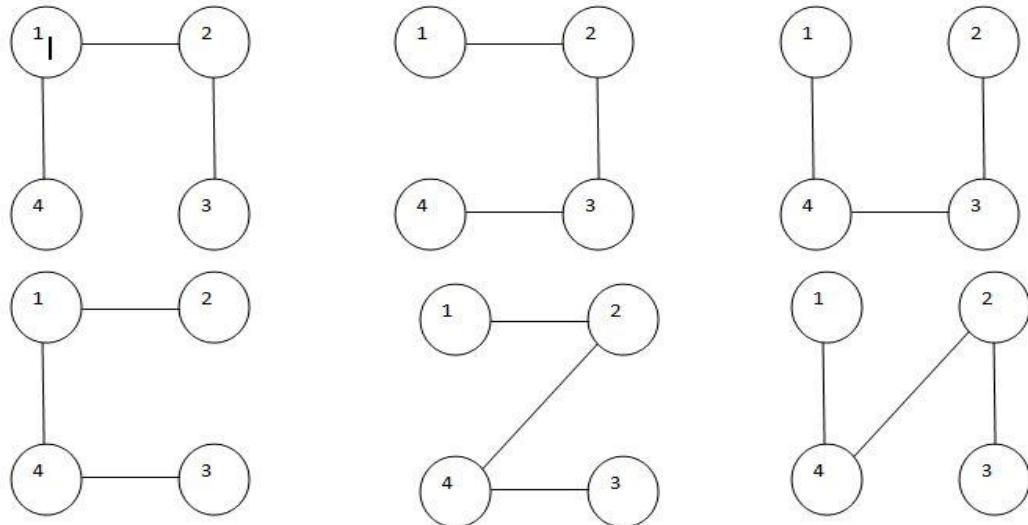
So, for a graph there can be various spanning trees. As in the example, number of edges in the graph are 6. So, the total number of edges in the spanning tree will be $6-1=5$. Then the number of spanning trees possible will be ${}^6C_5 = 6$.



Example: A weighted graph is given here.



Out of this weighted graph, some of the possible spanning trees are



Out of a graph, we can have various spanning tree. But here our objective is to find the minimal spanning tree. A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. For finding the minimal spanning tree, there are greedy methods:

- 1) Prim's Algorithm
- 2) Kruskal's Algorithm

3.4 Prim's Algorithm

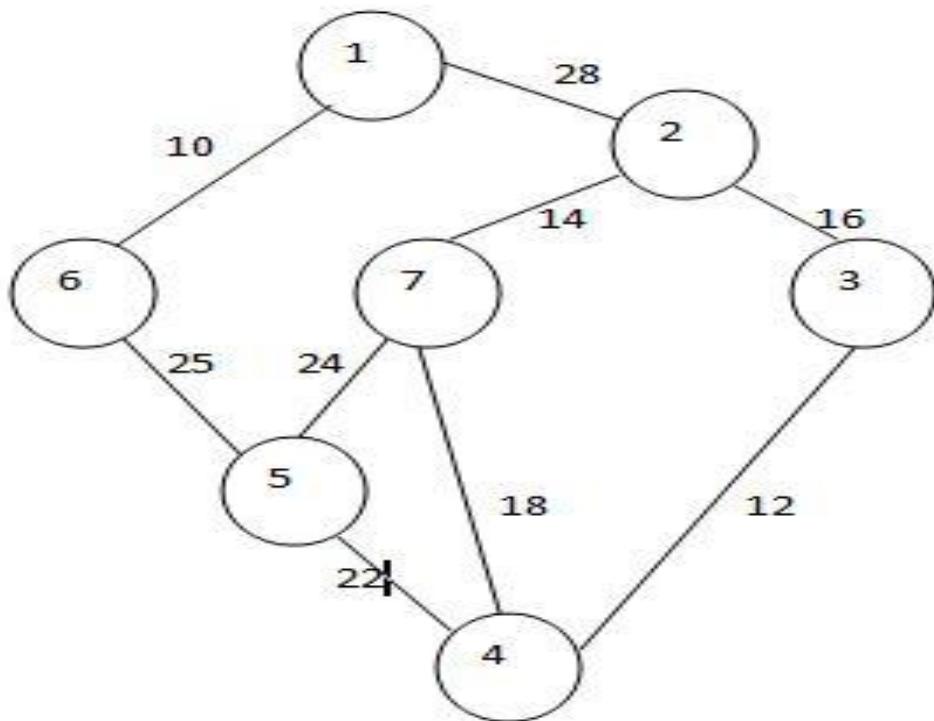
In computer science, Prim's algorithm (also known as Jarník's algorithm) is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Algorithm

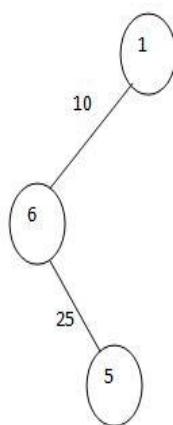
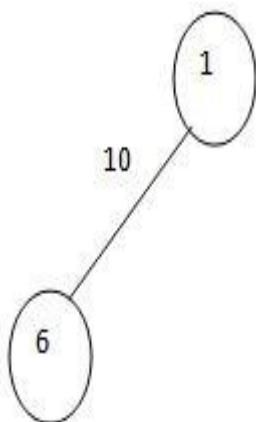
- 1) Select the minimum cost edge.
- 2) Select the next connected minimum cost edge (it should not form a cycle).
- 3) Form a spanning graph.
- 4) Calculate the total cost.

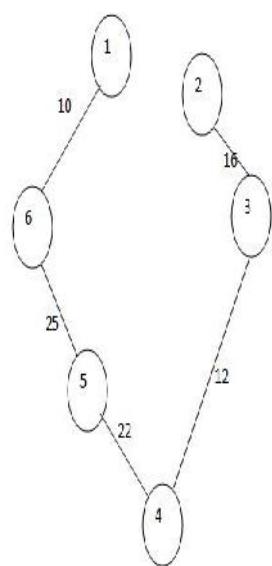
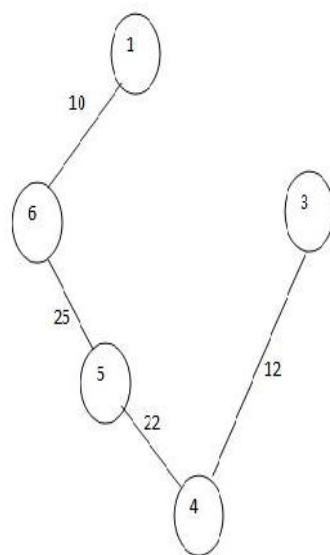
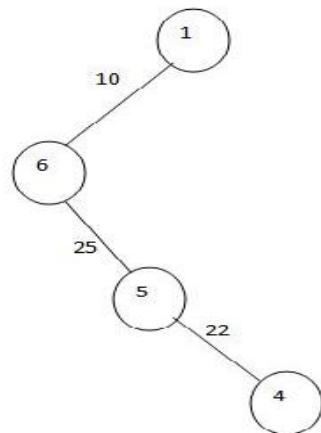


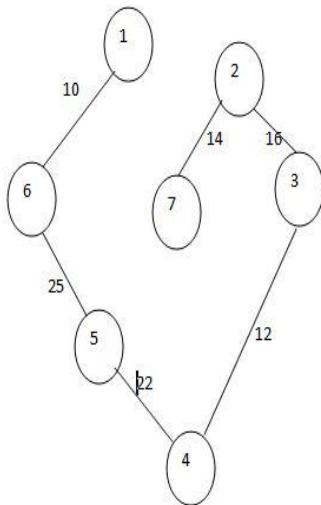
Example: Given a weighted graph G, find the minimal spanning tree using Prim's algorithm



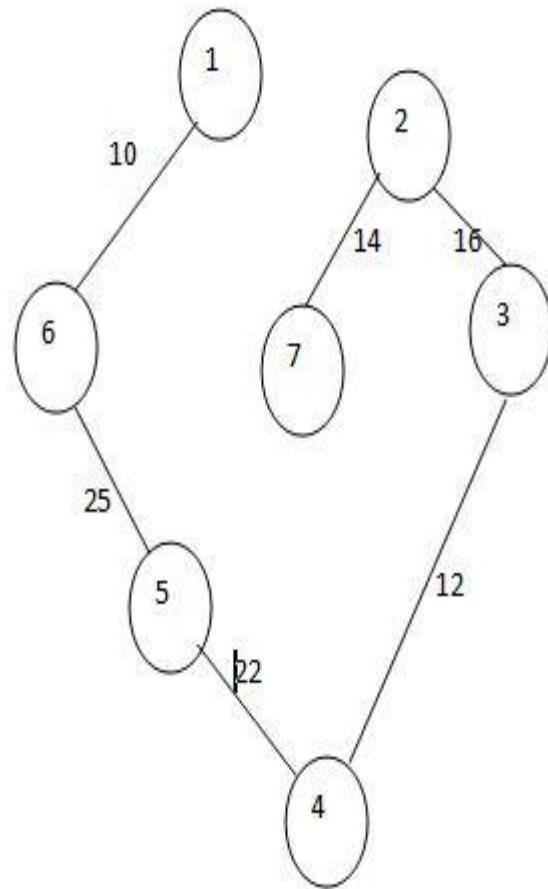
Solution:



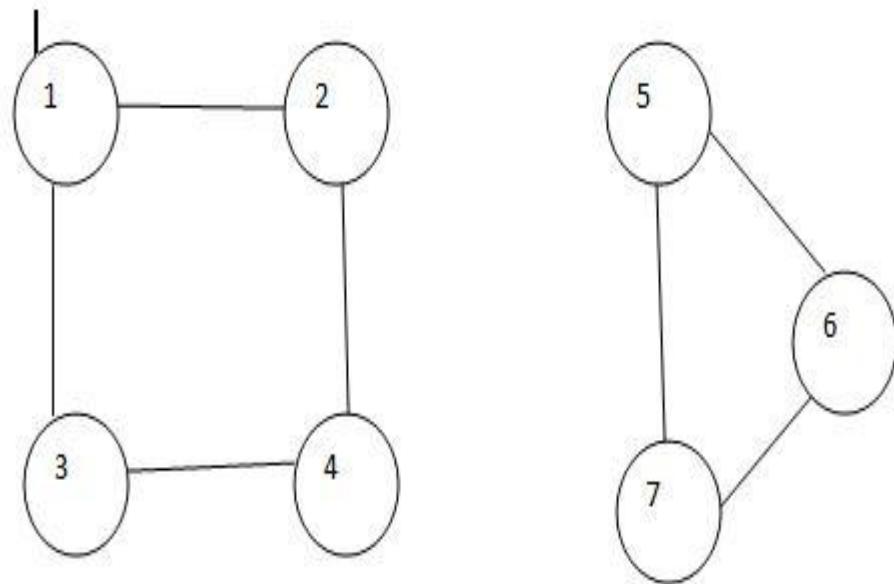




The minimal spanning tree is:



$$\text{COST} = 10 + 25 + 22 + 12 + 16 + 14 = 99$$

Case of non-connected graph:

In case of non-connected graph, we can't find the spanning tree.

3.5 Kruskal's Algorithm

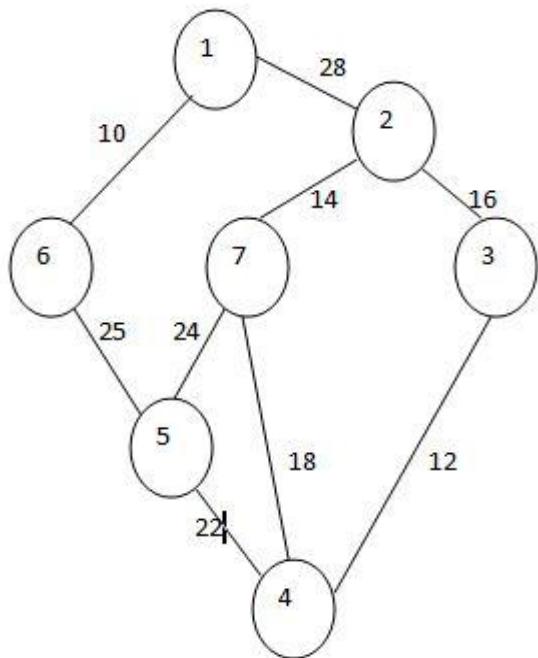
The Kruskal's algorithm follows the greedy approach. It says that always select a minimum cost edge (but it should not form a cycle). Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which, we can traverse every vertex of the graph. Kruskal's algorithm follows greedy approach which finds an optimum solution at every stage instead of focusing on a global optimum.

Algorithm

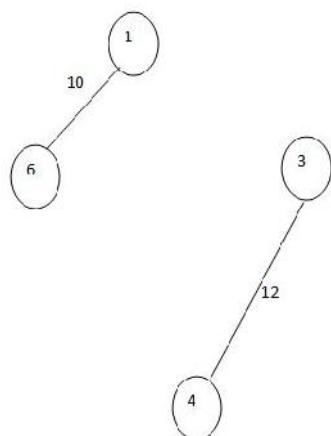
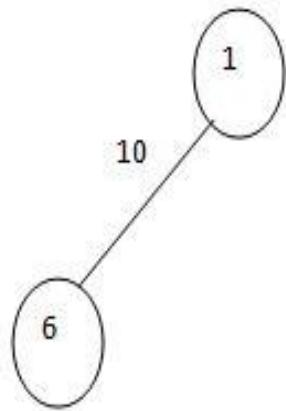
- Step 1: Create a forest in such a way that each graph is a separate tree.
- Step 2: Create a priority queue Q that contains all the edges of the graph.
- Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
- Step 4: Remove an edge from Q
- Step 5: IF the edge obtained in Step 4 connects two different trees, then Add it to the forest
(for combining two trees into one tree).ELSE Discard the edge
- Step 6: END

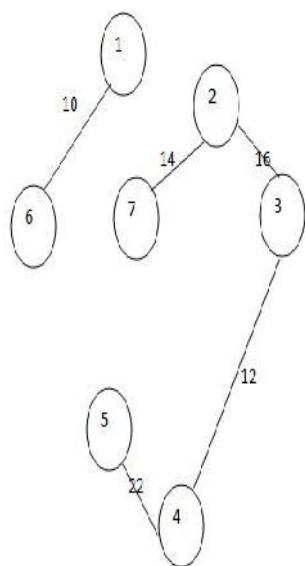
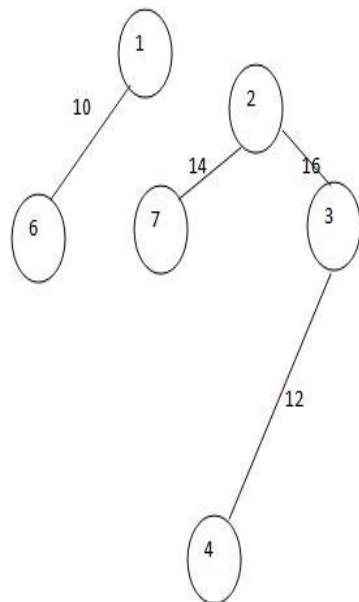
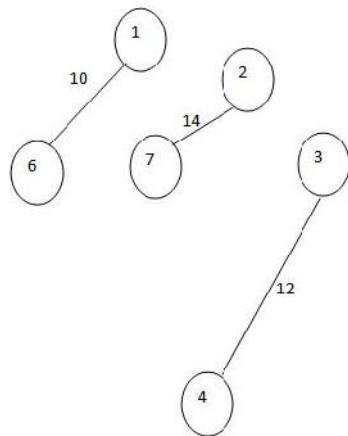


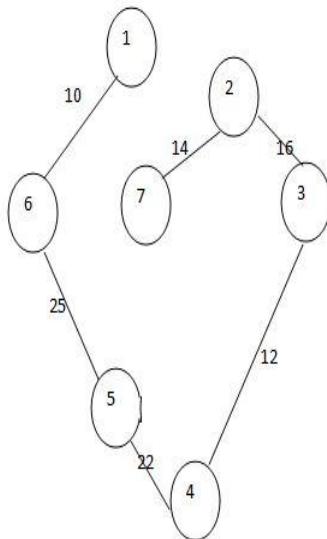
Example: Given a weighted graph, find out the minimal spanning tree using Kruskal's algorithm.



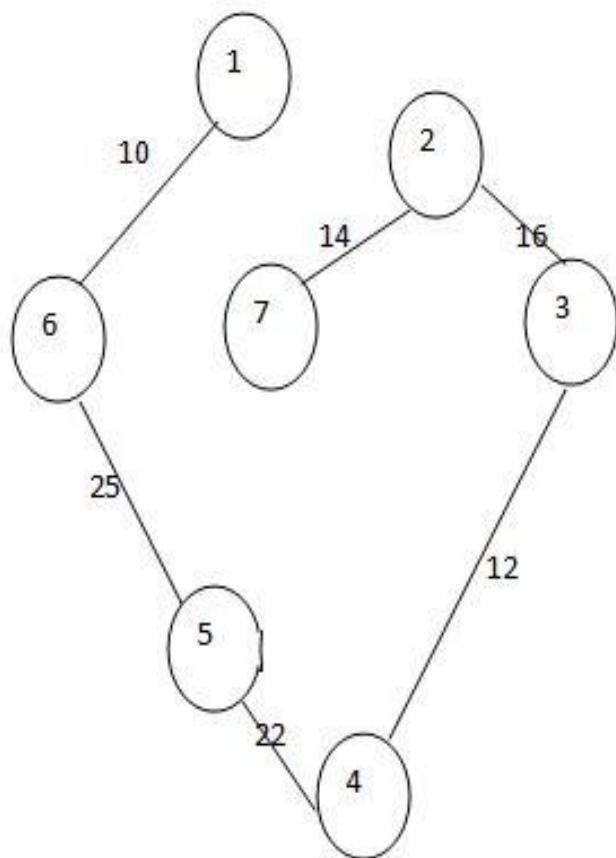
Solution:







The minimal spanning tree is:

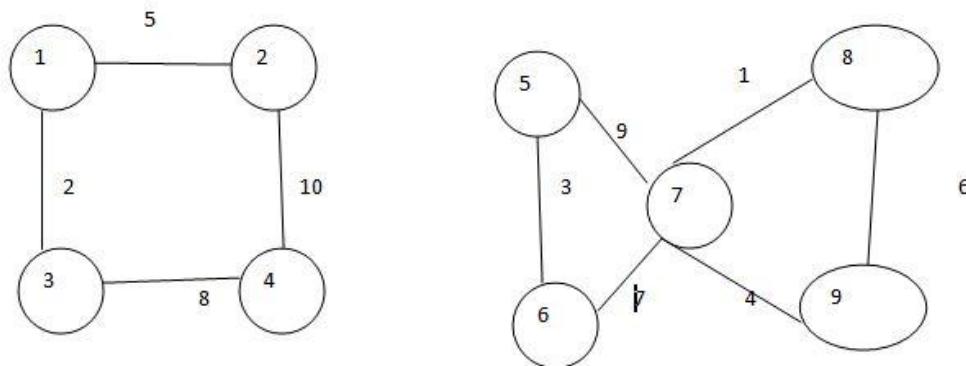


$$\text{COST} = 10 + 25 + 22 + 12 + 14 + 16$$

$$= 99$$

Time Complexity

The time complexity is $\Theta(|V| |E|) = \Theta(n.e) = \Theta(n^2)$. The Time taken by Kruskal's algorithm can be improved by taking Min Heap. A min-heap is a binary tree such that - the data contained in each node is less than (or equal to) the data in that node's children. When you use a Min heap, you need not to search. It will give the minimum value. The time taken for finding a minimum cost edge = Log n. The number of times, the procedure is repeated = n. So, the time complexity is $\Theta(n \log n)$.

Case of non-connected graph

In case of non-connected graph, it may find the spanning graph for different components, but it is not able to find the spanning graph for the complete one.

3.6 Single Source Shortest Path

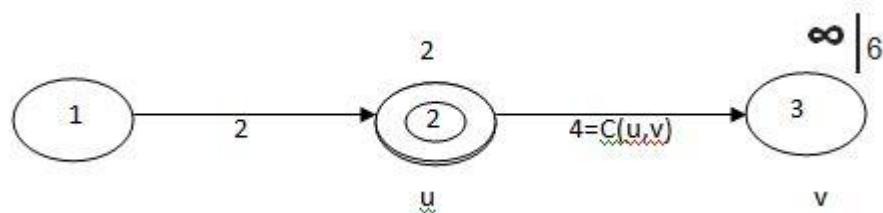
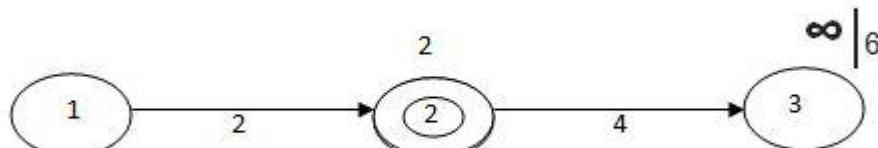
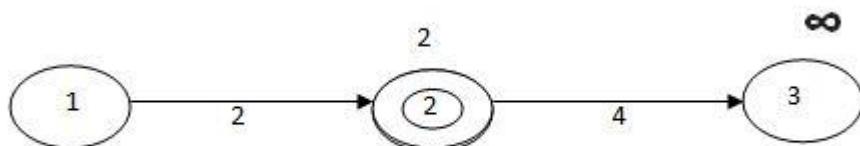
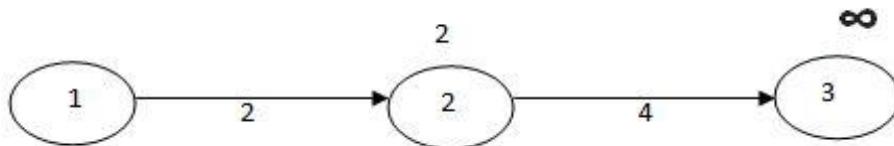
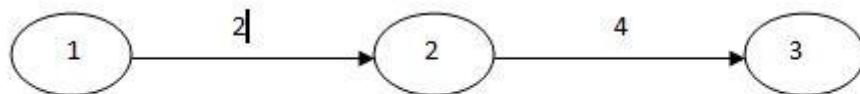
The Single-Source Shortest Path (SSSP) problem consists of finding the shortest paths between a given vertex v and all other vertices in the graph. Algorithm such as Dijkstra solve this problem. This problem is related to finding the shortest path from the starting vertex to all the other vertices (May be a direct path or via other vertices). Any vertex can be selected as a source vertex. So, greedy algorithm can be applied here. It can be solved in stages by taking only one input at a time. So, for this we will apply the Dijkstra algorithm. This algorithm can work on directed as well as non-directed graph.

3.7 Dijkstra Algorithm

Dijkstra's algorithm always searches for the shortest path. So, it selects a vertex with the shortest path and then find the shortest path to the vertices.

Algorithm

- 1) Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
- 2) Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
- 3) For the current node, consider all of its unvisited neighbors and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.
- 4) When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
- 5) If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
- 6) Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

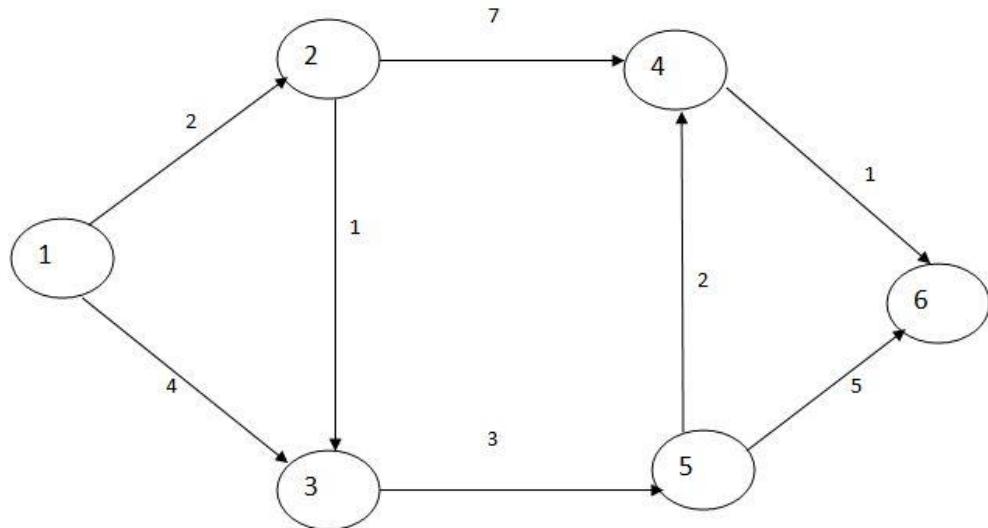
Relaxation

if ($d[u] + c(u, v) < d[v]$)

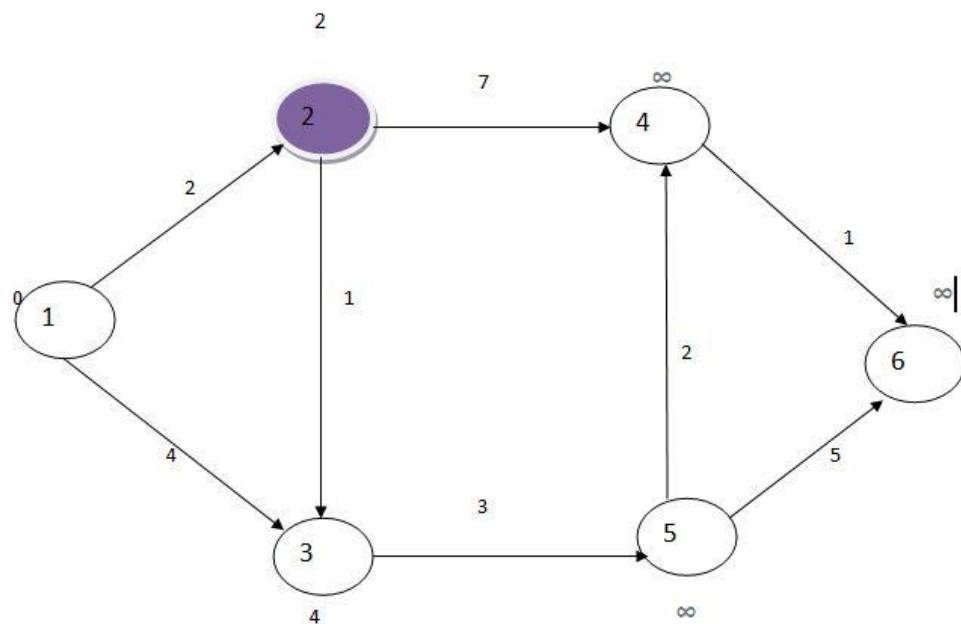
$d[v] = d[u] + c(u, v)$

Procedure of Dijkstra algorithm

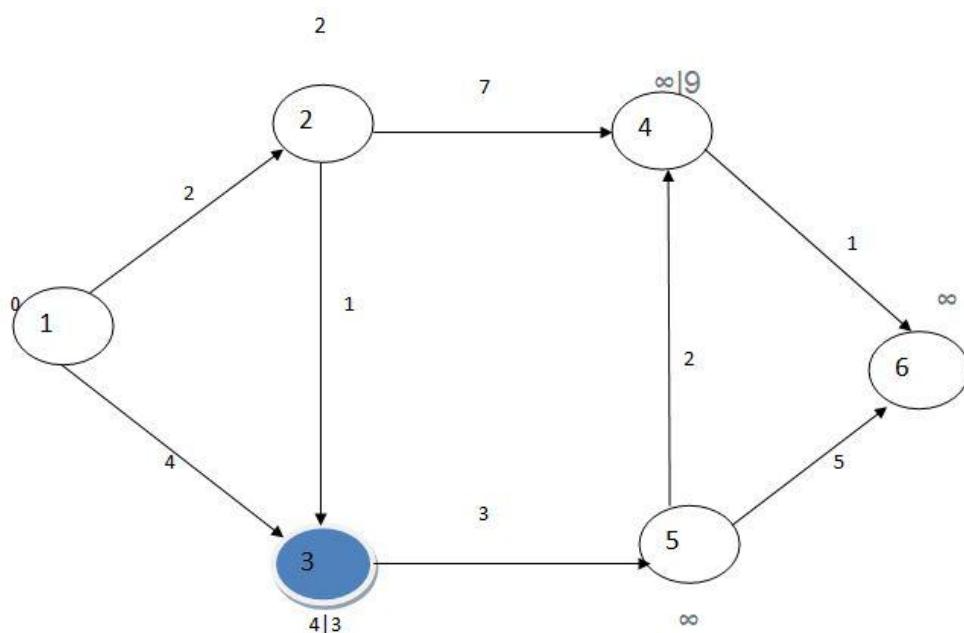
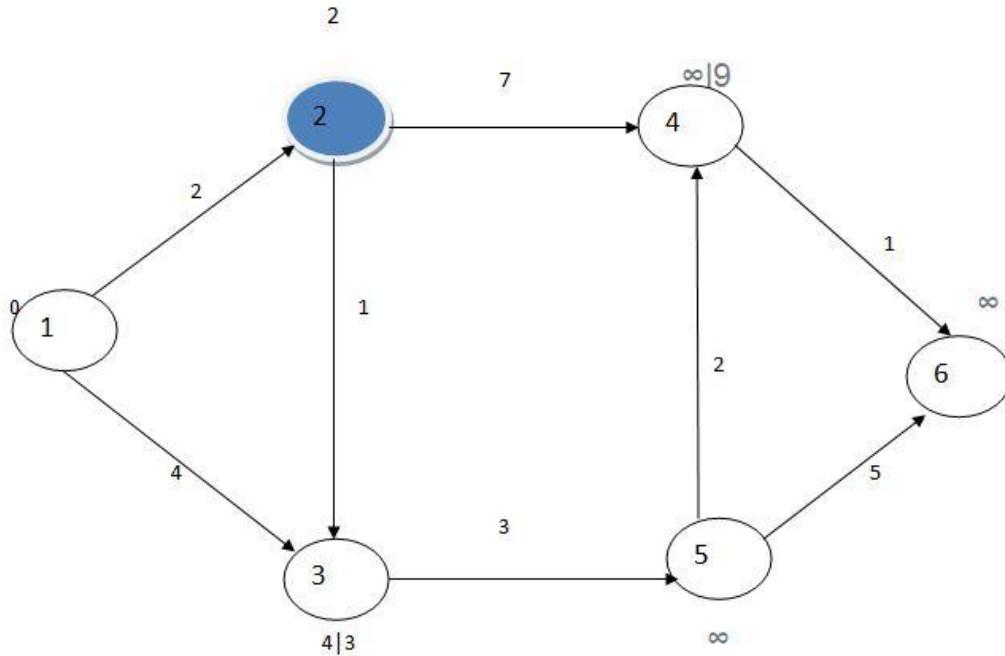
A weighted graph is given. We need to find the single source shortest path. The starting vertex given is 1.

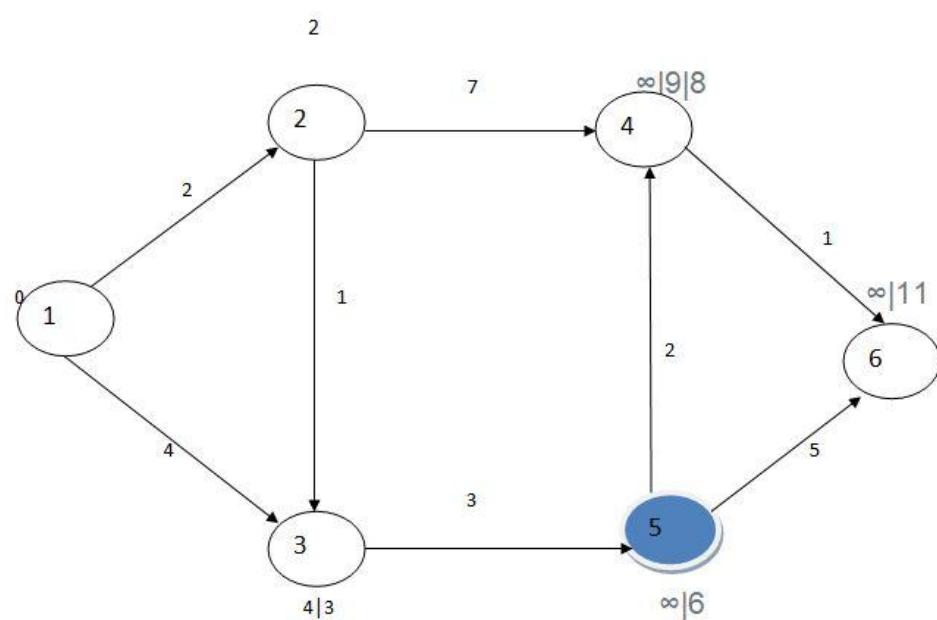
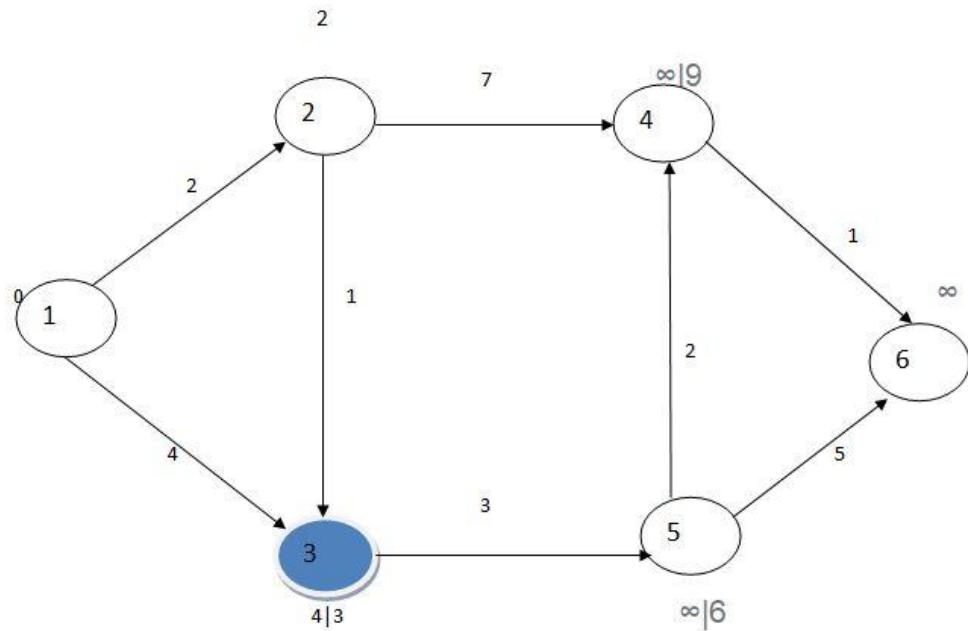


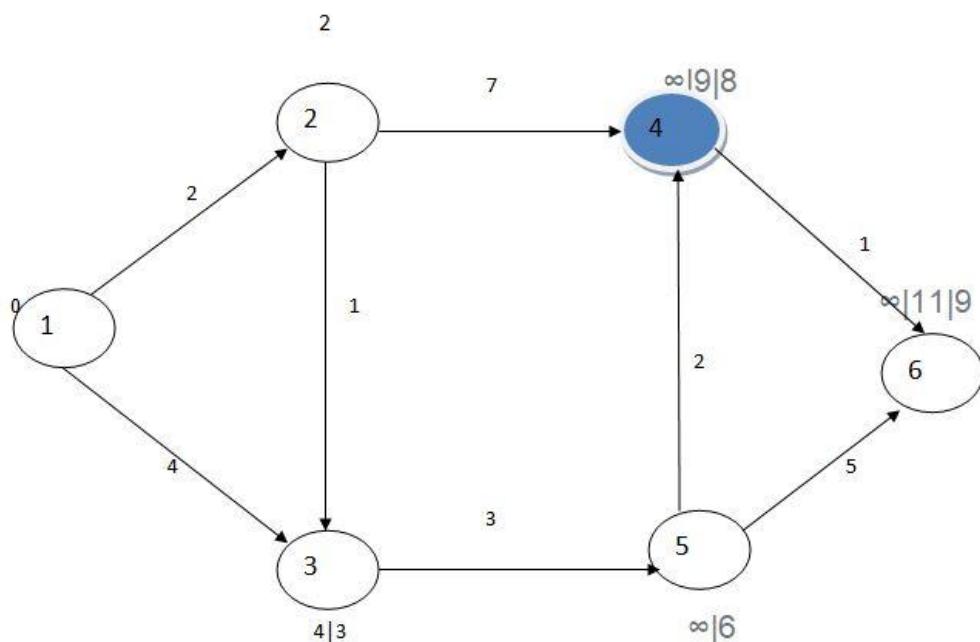
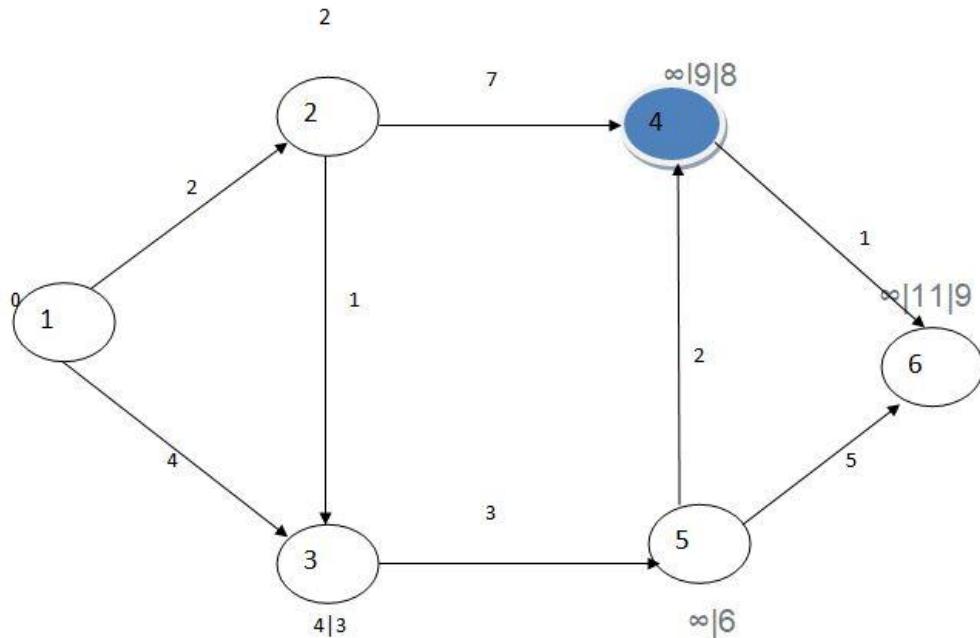
We see that the distance from vertex 1 to 1 is 0. Rest all the distances are updated, if there is a path (direct or indirect) path between them. If there is no path, then we have assigned infinity to that vertex.

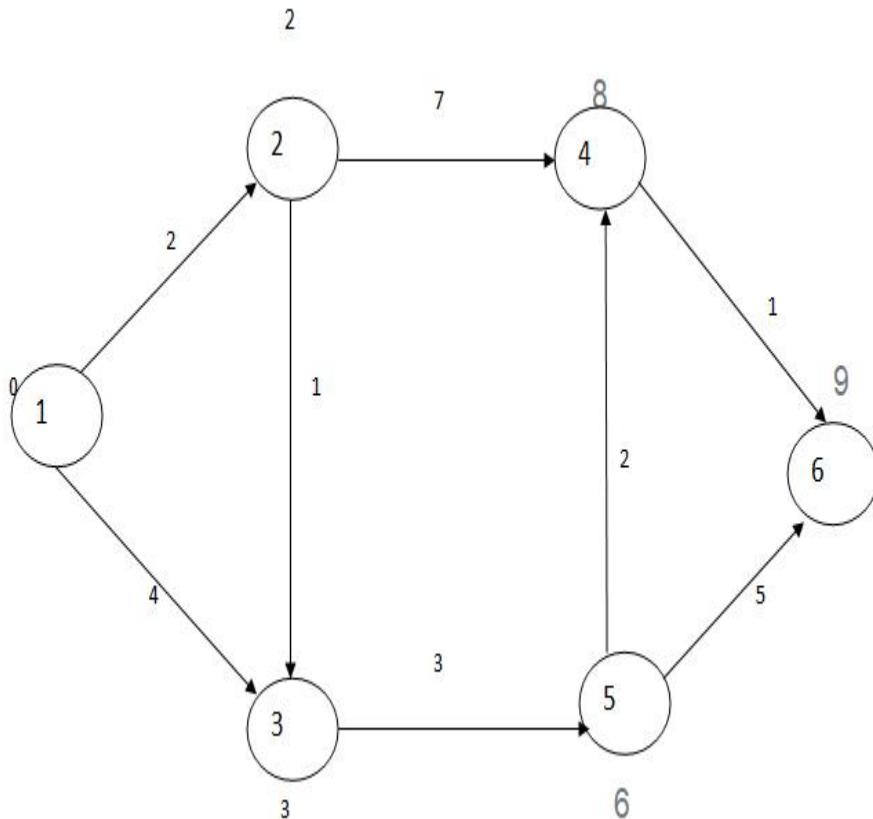


Vertices 3 and 4 are connected.







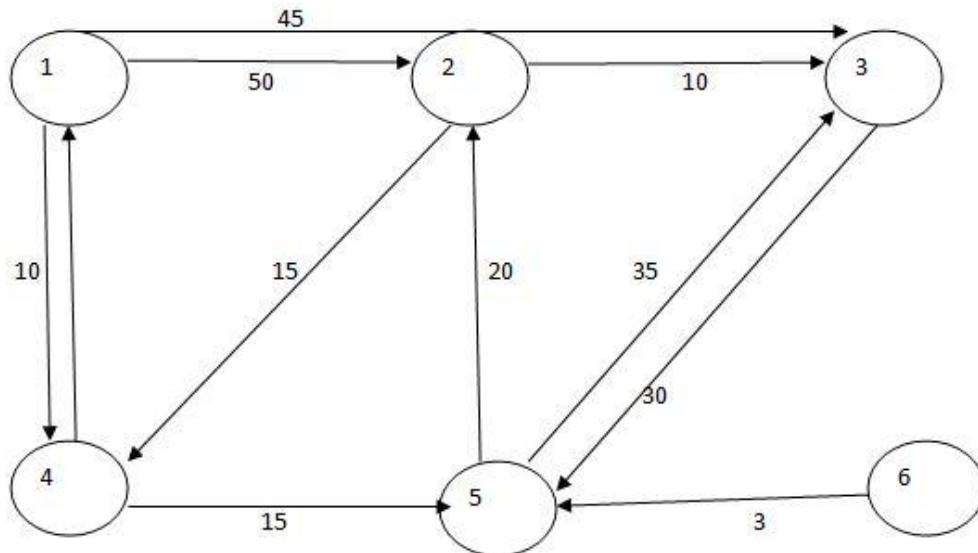


vertices	distances
2	2
3	3
4	8
5	6
6	9

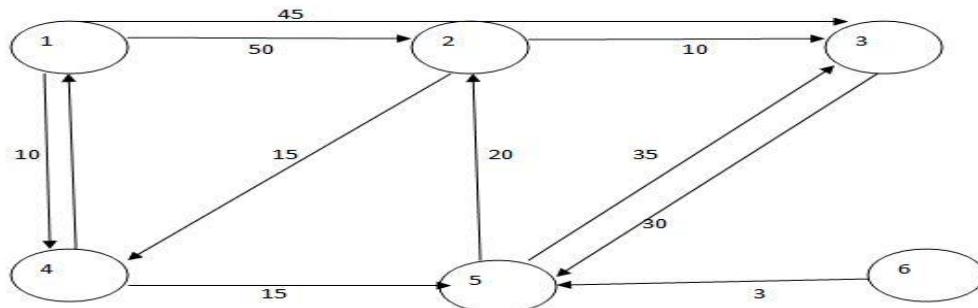


Example:

Given a weighted graph, find out the single source shortest path using Dijkstra algorithm. Starting vertex given is 1.



Solution:



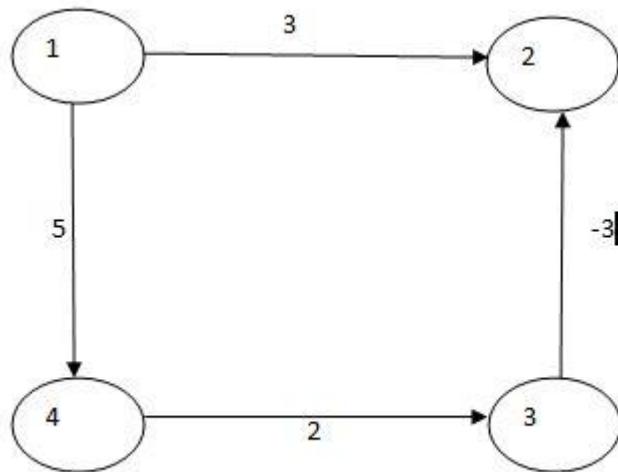
Selected Vertex	2	3	4	5	6
4	50	45	10	∞	∞
5	50	45	10	25	∞
2	45	45	10	25	∞
3	45	45	10	25	∞
6	45	45	10	25	∞

Time complexity

- Number of vertices, $|V| = n$
- It is relaxing = Atmost n vertices
- Time taken = $n \cdot n = n^2$

- Worst case time = $\Theta(n^2)$

Drawback of Dijkstra algorithm



In case of negative edges, it does not work.

Conclusion

- Dijkstra algorithm may or may not work on negative edges.
- It is not trying all the possibilities. It is just seeing the minimum one and selecting it. So, it is a greedy approach. So, greedy approach is failed here when we have the negative edges.

3.8 Optimal Storage on Tapes

There are n programs that are to be stored on a computer tape of length l . Associated with each program i is a length l_i , $1 \leq i \leq n$. Clearly, all programs can be stored on the tape if and only if the sum of the lengths of the programs is at most l . We assume that whenever a program is to be retrieved from this tape, the tape is initially positioned at the front. Hence, if the programs are stored in the order $I = i_1, i_2, i_3, \dots, i_n$, the time t_i needed to retrieve program i_j is proportional to $\sum_{1 \leq k \leq j} l_{i_k}$. If all programs are retrieved equally often, then the expected or mean retrieval time (MRT) is $(1/n) \sum_{1 \leq j \leq n} t_i$.

Requirement

In the optimal storage on tape problem, we are required to find a permutation for the n programs so that when they are stored on the tape in this order the MRT is minimized. This problem fits the ordering paradigm. Minimizing the MRT is equivalent to minimizing $d(l) = \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq j} l_{i_k}$.

Example

Given three tapes and their associated lengths. We need to find optimal ordering.

Solution: Let $n = 3$ and $(l_1, l_2, l_3) = (5, 10, 3)$. There are $n! = 6$ possible orderings. These orderings and their respective d values are:

• Ordering I	$d(l)$
1, 2, 3	$5+5+10+5+10+3 = 38$
1, 3, 2	$5+5+3+5+3+10 = 31$

Unit 03: Greedy Method

2, 1, 3	$10+10+5+10+5+3 = 43$
2, 3, 1	$10+10+3+10+3+5 = 41$
3, 1, 2	$3+3+5+3+5+10 = 29$
3, 2, 1	$3+3+10+3+10+5 = 34$

The optimal ordering is 3, 1, 2.

A greedy approach to building the required permutation would choose the next program on the basis of some optimization measure. One possible measure would be the d value of the permutation constructed so far. The next program to be stored on the tape would be one that minimizes the increase in d . If we have already constructed the permutation $i_1, i_2, i_3, \dots, i_r$, then appending program j gives the permutation $i_1, i_2, i_3, \dots, i_r, i_{r+1} = j$. This increases the d value by $\sum_{1 \leq k \leq r} l_{ik} + l_j$. Since $\sum_{1 \leq k \leq r} l_{ik}$ is fixed and independent of j , we trivially observe that the increase in d is minimized if the next program chosen is the one with the least length from among the remaining programs. The greedy method simply requires us to store the programs in non-decreasing order of their lengths. This ordering can be carried out in $O(n \log n)$ time.

Algorithm

Algorithm Store(n, m)

```
// n is the number of programs and m the number of tapes.
{
j := 0; // Next tape to store on
for i := 1 to n do
{
    write ("append program", i, "to permutation for tape", j);
    j := (j + 1) mod m;
}
}
```

The tape storage problem can be extended to several tapes. If there are $m > 1$ tapes, T_0, \dots, T_{m-1} , then the programs are to be distributed over these tapes. For each tape a storage permutation is to be provided. If L is the storage permutation for the subset of programs on tape j , then $d(L_j)$ is as defined earlier. The total retrieval time (TD) is $\sum_{0 \leq j \leq m-1} d(L_j)$. The objective is to store the programs in such a way as to minimize TD. It has a computing time of $\Theta(n)$ and does not need to know the program lengths.

Summary

- In greedy method, the problems have n inputs and require us to obtain a subset that satisfies some constraints.
- The greedy method suggests that one can devise an algorithm that works in stages, considering one input at a time.
- A spanning tree does not have cycles and it cannot be disconnected. A spanning tree is a sub-graph of a graph which has all the vertices; the number of edges will be less than one than the number of vertices.
- There are greedy methods for finding the minimal cost spanning trees: Prim's Algorithm and Kruskal's Algorithm.
- Kruskal's algorithm says that always select a minimum cost edge (but it should not form a cycle).

Algorithm Design and Analysis

- Dijkstra algorithm may or may not work on negative edges.
- We assume that whenever a program is to be retrieved from this tape, the tape is initially positioned at the front.

Keywords

- **Feasible solution:** Any subset that satisfies the constraints is called a **feasible solution**.
- **Optimal solution:** We need to find a feasible solution that either maximizes or minimizes a given objective function. A feasible solution that does this is called an optimal solution.
- **Knapsack:** A bag or container with some defined capacity.
- **Spanning Tree:** A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges.
- **Min heap:** A min-heap is a binary tree such that - the data contained in each node is less than (or equal to) the data in that node's children.
- **Single source shortest path:** Finding the shortest path from the starting vertex to all the other vertices (May be a direct path or via other vertices).
- **Dijkstra algorithm:** The Dijkstra algorithm can work on directed as well as non-directed graph. It always searches for the shortest path. So, it selects a vertex with the shortest path and then find the shortest path to the vertices.
- **Optimal storage on tapes problem:** In the optimal storage on tape problem, we are required to find a permutation for the n programs so that when they are stored on the tape in this order the MRT is minimized.

Self Assessment

1. Choose the odd one out from the following.
 - Merge sort
 - Binary search
 - Arithmetic with large numbers
 - 0/1Knapsack
2. The sum of all programs must be _____ of total length of tape
 - Sqrt
 - Less than
 - Greater than
 - None of the above
3. Which of these approaches is used for solving optimal storage on tapes problem?
 - Greedy approach
 - Divide and Conquer
 - Backtracking
 - Branch and Bound
4. What is the computing time of tapes?
 - $O(1)$
 - $O(n)$
 - $O(\log n)$
 - None of the above

5. The ordering in tapes can be carried out in ____ time.
 - A. $O(1)$
 - B. $O(n)$
 - C. $O(\log n)$
 - D. $O(n \log n)$

6. Out of these, the Dijkstra algorithm cannot work on ____.
 - A. Directed edges
 - B. Undirected edges
 - C. Negative edges
 - D. None of the above

7. The Dijkstra algorithm follows ____.
 - A. Dynamic programming
 - B. Greedy method
 - C. Divide and Conquer
 - D. Quick sort

8. The Dijkstra algorithm can work on
 - A. Directed graph
 - B. Undirected graph
 - C. Both of the above
 - D. None of the above

9. The Dijkstra algorithm always searches for ____ path.
 - A. Shortest
 - B. Longest
 - C. Equal
 - D. None of the above

10. Which data structure can be used to improve the time taken by minheap?
 - A. Stack
 - B. Queue
 - C. Minheap
 - D. Maxheap

11. The time complexity when minheap is employed is
 - A. $O(n)$
 - B. $O(n^2)$
 - C. $O(\log n)$
 - D. $O(n \log n)$

12. A spanning tree

- A. Does not have cycles
 B. Cannot be disconnected
 C. Both of the above
 D. None of the above
13. What are the greedy methods for finding the minimal spanning tree?
 A. Prim's algorithm
 B. Kruskal's algorithm
 C. Both of the above
 D. None of the above
14. In Knapsack problem, the constraint is that the total weight of items must be _____ the capacity of Knapsack.
 A. Greater than or equal to
 B. Less than or equal to
 C. Can be both
 D. None of the above
15. Knapsack problem where we can't take fractions is also known as _____
 A. 0/1 Knapsack problem
 B. Continuous knapsack problem
 C. Divisible knapsack problem
 D. Non continuous knapsack problem

Answers for Self Assessment

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

Review Questions

1. What is greedy method? Define various kinds of problems which can be solved using greedy method.
2. Mary wants to carry some fruits in her knapsack and maximize the profit she makes. She should pick them such that she **minimizes** weight (\leq bag's \leq bag's capacity) and **maximizes** value. Here are the **weights** and **profits** associated with the different fruits: **Items:** {Apple, Orange, Banana, Melon}, **Weights:** {2, 3, 1, 4} and **Profits:** { 4, 5, 3, 7 }. **Knapsack Capacity:** 5.
3. What is a graph? Explain its components. What is a spanning tree? How can we find out the number of possible spanning trees possible out of a graph?
4. Explain Prim's algorithm with the help of an example. Write its algorithm and drawbacks.

Unit 03: Greedy Method

5. Explain Kruskal's algorithm with the help of an example. Write its algorithm and drawbacks.
6. What is single source shortest path problem? Write its algorithm.
7. Explain the process of Dijkstra algorithm with the help of an example.
8. What is optimal storage on tapes problem? What is its requirement? How greedy approach is applied on it?



Further Readings

<https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>
https://www.tutorialspoint.com/data_structures_algorithms/greedy_algorithms.htm
<https://www.sciencedirect.com/topics/computer-science/shortest-path-problem#:~:text=The%20Single%2DSource%20Shortest%20Path,%5B1%5D%20solve%20this%20problem.>

Unit 04: Dynamic Programming

CONTENTS

- Objectives
- Introduction
- 4.1 Methods of Use
- 4.2 Tabulation Method
- 4.3 Matrix Multiplication Problem
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions:
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the concept of dynamic programming
- Understand the applicability of dynamic programming
- Understand the method of matrix multiplication
- Understand the chained matrix multiplication
- How to apply the dynamic programming on it

Introduction

There are various strategies to solve the problems. In previous units we studied about divide-and-conquer and greedy methods. Both methods apply on different kinds of problems. Next strategy we have is dynamic programming. This strategy is used to solve the optimization problems. Optimization problem results in either minimum or maximum results. It is usually based on a recurrent formula and one (or some) starting states. A sub-solution of the problem is constructed from previously found ones. DP solutions have a polynomial complexity which assures a much faster running time than other techniques like backtracking, brute-force etc.

Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems.

Greedy method vs dynamic programming

1. In greedy method, we try to follow the predefined procedure to get the optimal results. The procedure is known to be optimal, and we follow that procedure to get the best results. For example: Kruskal's algorithm for finding the minimal spanning tree and Dijkstra's algorithm for finding the shortest path vertex and continue relaxing the vertex.
- In dynamic programming, we explore all possible solutions and pick up the best one. So, it is more time consuming as compared to the greedy method. For a problem, there may be

Algorithm Analysis and Design

many solutions which are feasible and from those, we pick up the best one. The problems are solved using the recursive formulas.

2. In greedy method, the decision is taken only once, and we follow the procedure based upon that decision.
- In dynamic programming, in every stage we take the decision.

We will not use recursion of programming; we use the recursion formula. Dynamic programming follows the principle of optimality. The principle of optimality says that a problem can be solved by using the sequence of decisions. An algorithm is given which works recursively for finding out the Fibonacci of a number.

```
int fib(int n)
{
    if(n <= 1)
        return n;
    return fib(n-2) + fib(n-1);
}
```

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n - 2) + fib(n - 1) & \text{if } n > 1 \end{cases}$$

In the question given, a fibonacci series is given, we need to find out fib(5).

0	1	1	2	3	5	8	13	21	34
0	1	2	3	4	5	6	7	8	9

The procedure is

- fib(5)
- fib(3) + fib(4)
- {fib(1) + fib(2)} + {fib(2) + fib(3)}
- {fib(1) + fib(0) + fib(1)} + {fib(0) + fib(1) + fib(1) + fib(2)}
- {fib(1) + fib(0) + fib(1)} + fib(0) + fib(1) + fib(1) + fib(0) + fib(1)}

So, it is clear that for finding out fib(5), we need to have fib(0) and fib(1) and for finding just fib(5), it is making total 15 calls. The recurrence relation for this is $T(n) = 2T(n-1) + 1$. By master's theorem, for decreasing function, it is $O(2^n)$. Hence, the time taken by this function = $O(2^n)$.

As for just finding out fib(5), it is making 15 calls. It means a lot of time is consumed here. For reducing the time taken, we can use two methods which are discussed next.

4.1 Methods of Use

So, in dynamic programming there are two methods which can be used:

Unit 04: Dynamic Programming

- 1) Memorization method
- 2) Tabulation method

Memorization method:

According to the example given, initially when we don't have the answer for fib(0), fib(1) or fib(2), then in that case we are putting -1 in front of that. So, initially everything will be -1.

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

Now we want to find out fib(5). For that we need to call fib(3) and fib(4). Till now we don't have the solution for it. So, it will be again -1.

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

Now we want to find out fib(3). For that we need to call fib(2) and fib(1). As for fib(1), we have the value as 1. We will put that value.

-1	1	-1	-1	-1	-1
0	1	2	3	4	5

Now we want to find out fib(2). For that we need to call fib(0) and fib(1). As for fib(0), we have the value as 0. We will put that value.

0	1	-1	-1	-1	-1
0	1	2	3	4	5

The value of fib(0) and fib(1) will be used for the calculation of fib(2). So we will fill the value in the table.

0	1	1	-1	-1	-1
0	1	2	3	4	5

The same procedure will be followed for fib(3)

Algorithm Analysis and Design

0	1	1	2	-1	-1
0	1	2	3	4	5

For fib(4)

0	1	1	2	3	-1
0	1	2	3	4	5

For fib(5)

0	1	1	2	3	4
0	1	2	3	4	5

Now according to the memorization method

- fib(5)
- fib(3) + fib(4)
- {fib(1) + fib(2)} + {fib(2) + fib(3)}
- {fib(1) + fib(0) + fib(1)} + {fib(0) + fib(1) + fib(1) + fib(2)}
- {fib(1) + fib(0) + fib(1)} + fib(0) + fib(1) + fib(1) + fib(0) + fib(1)}

Total number of calls to the function is 6 if we use the memorization method. So, for fib(5), we have 6 calls and it is equal to $n+1$. Time complexity will be $\Theta(n)$. It shows that it follows top down approach.

4.2 Tabulation Method

In tabulation method, we need to write the iterative function using the loop. The iterative function is:

```
int fib(int n)
{
    if(n<=1)
        return n;
    f[0]=0, f[1]=1;
    for(int i =2; i<=n; i++)
    {
        f[i] = f[i-2] + f[i-1];
    }
    return f[n];
```

Unit 04: Dynamic Programming

 }

As the value of fib(0) and fib(1) are known, we need to fill the values. So, if we make a call to fib(2), that we can find out by calling fib(0) and fib(1). We just need to add these two values.

0	1	1			
0	1	2(i)	3	4	5

Following the same process, we will find out the value of fib(3)

0	1	1	2		
0	1	2	3(i)	4	5

The value of fib(4) is:

0	1	1	2	3	
0	1	2	3	4(i)	5

The value of fib(5) is:

0	1	1	2	3	5
0	1	2	3	4	5(i)

It follows the bottom up approach, as for finding out fib(5), it has first found out fib(0) then fib(1) and so on. The tabulation method is widely used in dynamic programming.

Advantages and disadvantages of dynamic programming

- Dynamic Programming is not recursive.
- DP solves the sub problems only once and then stores it in the table.
- In DP the sub-problems are not independent.
- It takes a lot of memory to store the calculated result of every sub-problem without ensuring if the stored value will be utilized or not.
- Many times, output value gets stored and never gets utilized in the next sub-problems while execution. It leads to unnecessary memory utilization.
- In DP, functions are called recursively. Stack memory keeps increasing.

Conclusion

- Dynamic Programming is mainly an optimization over plain recursion.

Algorithm Analysis and Design

- Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming.
- The idea is to simply store the results of subproblems, so that we do not have to recompute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

4.3 Matrix Multiplication Problem

In mathematics, a matrix (plural matrices) is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns, which is used to represent a mathematical object or a property of such an object. The condition for multiplying two matrices is the columns of first matrix must be equal to the column of second matrix. For example: A[5*4] * B[4*3]. But if we reverse the order, that will not work. For example: B[4*3] * A[5*4].

If you are multiplying two matrices, A[5*4] and B[4*3], then you get the result as C[5*3], i.e., A[5*4] * B[4*3] = C[5*3]. In C[5*3], there will be a total of 15 elements. In total 4 multiplications are done for each single element. SO the cost here is in total $15 \times 4 = 60$ multiplications are required to fill the elements in C[5*3].



For example: A₁ [5*4], A₂ [4*6], A₃ [6*2] and A₄[2*7]. In matrix multiplication problem, we will multiply all four matrices, A₁ * A₂ * A₃ * A₄. All of these matrices can't be multiplied together, we have to take a pair at a time for multiplication. The challenge here is to select the pair of matrices for multiplication first, so that we have the total minimum cost for multiplications. This problem is known as **matrix chain multiplication problem**.

- The First way for doing this:** A₁ * A₂ * A₃ * A₄

$$\begin{aligned}
 &= (A_1 * A_2) * A_3 * A_4 \\
 &= \{(A_1 * A_2) * A_3\} * A_4 \\
 &= [(A_1 * A_2) * A_3] * A_4
 \end{aligned}$$

- Second way:** A₁ * A₂ * A₃ * A₄

$$\begin{aligned}
 &= A_1 * (A_2 * A_3) * A_4 \\
 &= \{A_1 * (A_2 * A_3)\} * A_4 \\
 &= [\{A_1 * (A_2 * A_3)\} * A_4]
 \end{aligned}$$

- Third way:** A₁ * A₂ * A₃ * A₄

$$\begin{aligned}
 &= A_1 * A_2 * (A_3 * A_4) \\
 &= (A_1 * A_2) * (A_3 * A_4)
 \end{aligned}$$

Like this, there are various ways to do so. If we make a tree out of the possible ways, there will be 3 nodes corresponding to each multiplication. Total number of possible ways, T(n) = $2nC_n/(n+1)$. Hence, T(3) = 5.

Dynamic programming says that we must try all the possibilities and pick out the best one. In dynamic programming, the tabulation method follows the bottom-up approach. So, we will start from the minimum value, i.e., A₁.

- A₁ * A₂ * A₃ * A₄
- [5*4] [4*6] [6*2] [2*7]
- A₁=m[1,1]
- Not multiplied with anyone. So, cost of this one is zero.

Unit 04: Dynamic Programming

M	1	2	3	4
1	0			
2				
3				
4				

S	1	2	3	4
1				
2				
3				
4				

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] [4*6] [6*2] [2*7]$
- $A_2 = m[2,2]$
- Not multiplied with anyone, so, cost of this one is zero.

M	1	2	3	4
1	0			
2		0		
3				
4				
S	1	2	3	4

1					
2					
3					
4					

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] [4*6] [6*2] [2*7]$
- $A_3 = m[3,3]$
- Not multiplied with anyone, so, cost of this one is zero.

M	1	2	3	4
1	0			
2		0		
3			0	
4				

S	1	2	3	4
1				
2				
3				
4				

- $A_1 * A_2 * A_3 * A_4$

Unit 04: Dynamic Programming

- $[5*4] \quad [4*6] \quad [6*2] \quad [2*7]$
- $A_4 = m[4,4]$
- Not multiplied with anyone.
- So, cost of this one is zero.

M	1	2	3	4
1	0			
2		0		
3			0	
4				0

S	1	2	3	4
1				
2				
3				
4				

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] \quad [4*6] \quad [6*2] \quad [2*7]$
- $M[1,2] = A_1 * A_2$
- A_1 dim are $5*4$, A_2 dim are $4*6$.
- Total number of mul = $5*4*6 = 120$

M	1	2	3	4
1	0	120		

Algorithm Analysis and Design

2		0		
3			0	
4				0

S	1	2	3	4
1		1		
2				
3				
4				

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] [4*6] [6*2] [2*7]$
- $M[2,3] = A_2 * A_3$
- A_2 dim are $4*6$, A_3 dim are $6*2$.
- Total number of mul = $4*6*2 = 48$.

M	1	2	3	4
1	0	120		
2		0	48	
3			0	
4				0

S	1	2	3	4

Unit 04: Dynamic Programming

1		1		
2			2	
3				
4				

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] [4*6] [6*2] [2*7]$
- $M[3,4] = A_3 * A_4$
- A_3 dim are $6*2$, A_4 dim are $2*7$.
- Total number of mul = $6*2*7=84$.

M	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

S	1	2	3	4
1		1		
2			2	
3				3
4				

- $A_1 * A_2 * A_3 * A_4$

Algorithm Analysis and Design

- $[5*4] \quad [4*6] \quad [6*2] \quad [2*7]$
- $M[1,3] = A1 * A2 * A3$
- A1 dim are $5*4$, A2 dim are $4*6$ and A3 dim are $6*2$.
- **There are two possibilities:** First way: $A1*(A2 * A3)$ and Second way: $(A1 * A2)*A3$

M	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

S	1	2	3	4
1		1		
2			2	
3				3
4				

- **First way:** $A1*(A2 * A3) = 5*4 \quad 4*6 \quad 6*2$
- Cost of A1 = $M[1,1] = 0$
- Cost of $A2 * A3 = M[2,3] = 48$
- Total cost = $0 + 48 + 5*4*2$
- Total cost = 88

M	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

Unit 04: Dynamic Programming

S	1	2	3	4
1		1		
2				
3				
4				

S	1	2	3	4
1		1		
2			2	
3				3
4				

M	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

Using first way, the total cost is 88 and using second way, the total cost is 180. So, the minimum out of these two is 88.

Algorithm Analysis and Design

M	1	2	3	4
1	0	120	88	
2		0	48	
3			0	84
4				0

S	1	2	3	4
1		1	1	
2			2	
3				3
4				

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] [4*6] [6*2] [2*7]$
- $M[2,4] = A_2 * A_3 * A_4$
- A_2 dim are $4*6$, A_3 dim are $6*2$ and A_4 dim are $2*7$.
- **There are two possibilities:**
- First way: $A_2 * (A_3 * A_4)$ and second way: $(A_2 * A_3) * A_4$

M	1	2	3	4
1	0	120	88	
2		0	48	
3			0	84

Unit 04: Dynamic Programming

4				0
---	--	--	--	---

S	1	2	3	4
1		1	1	
2			2	
3				3
4				

- First way: $A_2 * (A_3 * A_4) = 4 * 6 \quad 6 * 2 \quad 2 * 7$
- $M[2,2] + M[3,4] + 4 * 6 * 7 = 0 + 84 + 168 = 252$

M	1	2	3	4
1	0	120	88	
2		0	48	
3			0	84
4				0

S	1	2	3	4
1		1	1	
2			2	
3				3

4				
---	--	--	--	--

- **Second way:** $(A_2 * A_3) * A_4 = 4 * 6 \quad 6 * 2 \quad 2 * 7$
- $M[2,3] + M[4,4] + 4 * 2 * 7 = 48 + 0 + 56 = 104.$
- Minimum is 104.

M	1	2	3	4
1	0	120	88	
2		0	48	104
3			0	84
4				0

S	1	2	3	4
1		1	1	
2			2	3
3				3
4				

- $A_1 \quad * \quad A_2 \quad * \quad A_3 \quad * \quad A_4$
- $[5*4] \quad [4*6] \quad [6*2] \quad [2*7]$

M	1	2	3	4
1	0	120	88	

Unit 04: Dynamic Programming

2		0	48	104
3			0	84
4				0
S	1	2	3	4
1		1	1	
2			2	3
3				3
4				

- $M[1,4] = \min\{ M[1,1] + M[2,4] + 5*4*7, M[1,2] + M[3,4] + 5*6*7, M[1,3] + M[4,4] + 5*2*7 \}$
- $M[1,4] = \min\{0+104+140, 120+84+210, 88+0+70\}$
- $M[1,4] = 158$
- Smaller result is given by the value 3.

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] [4*6] [6*2] [2*7]$

M	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

S	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

- $A_1 * A_2 * A_3 * A_4$
- $[5*4] [4*6] [6*2] [2*7]$
- $d_0=5, d_1=4, d_2=6, d_3=2, d_4=7$
- $M[i,j] = \min\{M[i,k] + M[k+1,j] + d_{i-1} * d_k * d_j\}$

Time complexity

- We are not generating the full table, we are just generating almost half of the table. $n(n-1)/2 = n^2$
- For each element we are calculating all and finding the minimum, so the time taken is n .
- So, the time taken is $n^2 * n = n^3$. So, the time complexity is $\Theta(n^3)$.

Algorithm

```

main()
{
int n=5;
int p[ ] = {5,4,6,2,7};
int m [5][5] = {0};
int j, min, i;
for(int d=1; d<n-1; d++)
{
for(int i=1; i<n-d; i++)
{
j=i+d;
min=32767;
for (int k=1, k<=j-1; k++)
{
q= m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j];
if(q<min)

```

```

{
min=q;
s[i][j] = k;
}
}
m[i][j] = min;
}
}

cout << m [1][n-1];
}

```

Summary

- Dynamic programming is used to solve the optimization problem.
- In dynamic programming, we explore all possible solutions and pick up the best one.
- Dynamic programming is more time consuming as compared to the greedy method.
- Dynamic programming follows the principle of optimality which says that a problem can be solved by using the sequence of decisions.
- In greedy method, the decision is taken only once, and we follow the procedure based upon that decision. In dynamic programming, in every stage we take the decision.
- There are two methods which can be used in dynamic programming: Memorization method and tabulation method.
- In dynamic programming, the tabulation method follows the bottom-up approach and memorization method follows the top down approach.

Keywords

- **Dynamic programming:** It is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used.
- **Matrix chain multiplication problem:** To select the pair of matrices for multiplication first, so that we have the total minimum cost for multiplications. This problem is known as **matrix chain multiplication problem**.
- **Top-down approach:** This is the direct fall-out of the recursive formulation of any problem. If the solution to any problem can be formulated recursively using the solution to its sub-problems, and if its sub-problems are overlapping, then one can easily memoize or store the solutions to the sub-problems in a table. Whenever we attempt to solve a new sub-problem, we first check the table to see if it is already solved. If a solution has been recorded, we can use it directly, otherwise we solve the sub-problem and add its solution to the table.
- **Bottom-up approach:** Once we formulate the solution to a problem recursively as in terms of its sub-problems, we can try reformulating the problem in a bottom-up fashion: try solving the sub-problems first and use their solutions to build-on and arrive at solutions to bigger sub-problems. This is also usually done in a tabular form by iteratively generating solutions to bigger and bigger sub-problems by using the solutions to small sub-problems.

Self Assessment

1. Which of these follows the top-down approach?
 - A. Tabulation method

- B. Memorization method
 - C. Both of the above
 - D. None of the above
2. Which of these follows the bottom-up approach?
- A. Tabulation method
 - B. Memorization method
 - C. Both of the above
 - D. None of the above
3. Which data structure is used for solving problems of dynamic programming?
- A. Queue
 - B. Stack
 - C. Tree
 - D. Graph
4. If four matrices, i.e., A₁, A₂, A₃ and A₄, are to multiplied using tabulation method of dynamic programming, then which one will be considered first?
- A. A₁
 - B. A₂
 - C. A₃
 - D. A₄
5. What is the time complexity of matrix chained multiplication method?
- A. O(n)
 - B. O(1)
 - C. O(n²)
 - D. O(n³)
6. The tabulation method of dynamic programming follows _____
- A. Left to right approach
 - B. Right to left approach
 - C. Top-down approach
 - D. Bottom -up approach
7. The ordering in tapes can be carried out in _____ time.
- A. O (1)
 - B. O (n)
 - C. O (log n)
 - D. O (n log n)
8. As per greedy method, the programs are stored in _____ order.
- A. Decreasing

Unit 04: Dynamic Programming

- B. Non-decreasing
 - C. Left to right
 - D. Right to left
9. The sum of all programs must be _____ of total length of tape
- A. Sqrt
 - B. Less than
 - C. Greater than
 - D. None of the above
10. The dynamic programming deals with
- A. Realization problems
 - B. Optimization problems
 - C. Neutralization problems
 - D. None of the above
11. Which of these methods can be used in dynamic programming?
- A. Tabulation method
 - B. Memorization method
 - C. Both of the above
 - D. None of the above
12. For multiplication of two matrices,
- A. The number of columns of first matrix must be equal to number of rows of second matrix.
 - B. The number of rows of first matrix must be equal to number of rows of second matrix.
 - C. The number of rows of first matrix must be equal to number of columns of second matrix.
 - D. The number of columns of first matrix must be equal to number of columns of second matrix.
13. In a matrix of dimension A [5*3], how many elements will be there?
- A. 3
 - B. 5
 - C. 15
 - D. 2
14. Which of these approaches is used for solving optimal storage on tapes problem?
- A. Greedy approach
 - B. Divide and Conquer
 - C. Backtracking
 - D. Branch and Bound
15. What is the computing time of tapes?
- A. $O(1)$
 - B. $O(n)$

- C. $O(\log n)$
- D. None of the above

Answers for Self Assessment

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

Review Questions

1. What is dynamic programming? Explain its difference between dynamic programming and greedy approach.
2. What are two methods of use in dynamic programming? Explain the difference between both.
3. What are advantages and disadvantages of dynamic programming? Explain its time complexity.
4. What is matrix chained multiplication? How dynamic programming is used to solve the problem.
5. Explain the algorithm and time complexity of matrix chained multiplication.



Further Readings

<http://www.topcoder.com/thrive/articles/Dynamic%20Programming:%20From%20Novice%20to%20Advanced>

<https://www.geeksforgeeks.org/matrix-chain-multiplication-dp-8/>

Unit 05: Dynamic Programming

CONTENTS

- Objectives
- Introduction
- 5.1 All-Pair Shortest Path
- 5.2 Optimal Binary Search Tree Problem
- 5.3 Bellman Ford Algorithm
- 5.4 Reliability Design
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

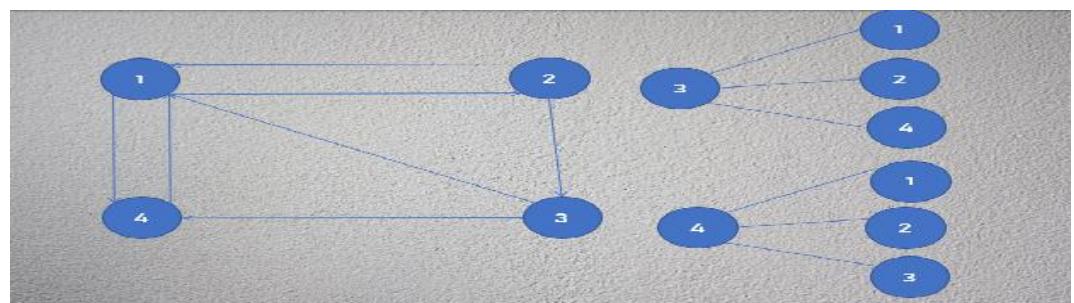
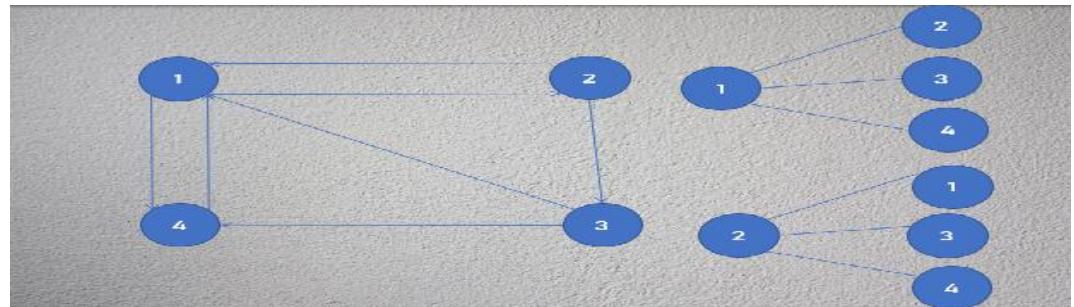
- Understand the concept of dynamic programming
- Understand the all-pair shortest path problem
- Understand the finding of optimal binary search tree problem
- Understand the Bellman Ford algorithm
- understand the reliability design

Introduction

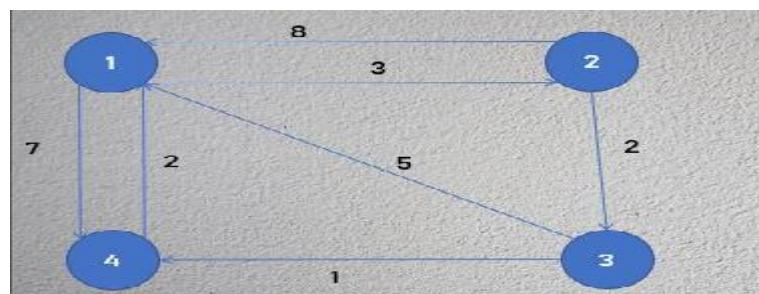
In dynamic programming, we find the solution of a subproblem, that is used in solving the big problem. In previous unit, we studied about two problems which can be efficiently solved using dynamic programming. Now further we are going to discuss more about dynamic programming.

5.1 All-Pair Shortest Path

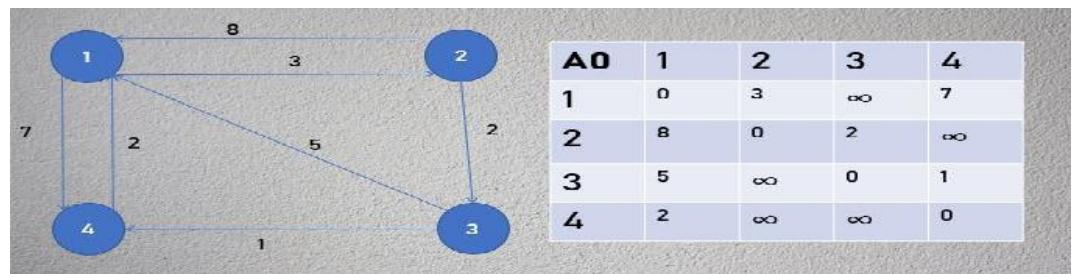
Given a weighted graph, $G(V, E)$: V is a set of vertices and E is set of edges which joins these vertices. Every edge joining two vertices has some associated weight. The problem is to find the shortest path between every pair of vertices. This problem can be solved using dynamic programming. Below shown is a graph which has four vertices and seven edges. Here in the graph if we see, some of the vertices are directly connected and some are not directly connected. The paths in the graph can be from vertex 1 to vertex 2, vertex 3 and vertex 4, then from vertex 2 to vertex 1, vertex 3 and vertex 4 and from vertex 3 to vertex 1, vertex 2 and vertex 4 and vertex 4 to vertex 1, vertex 2 and vertex 3 as shown in below two figures.



This seems like the process of Dijkstra's algorithm. So, if we see the similarity with Dijkstra's algorithm. There are few points to consider: the problem looks like the single source shortest path, i.e., Dijkstra's algorithm which finds the shortest path from one of the source vertexes and it takes $O(n^2)$ time and the Dijkstra's algorithm needs to be run for n number of times for n vertexes one by one. So, the total time will be $O(n^3)$. So, this works in the following manner. Given a weighted graph, we need to find the all-pair shortest path.

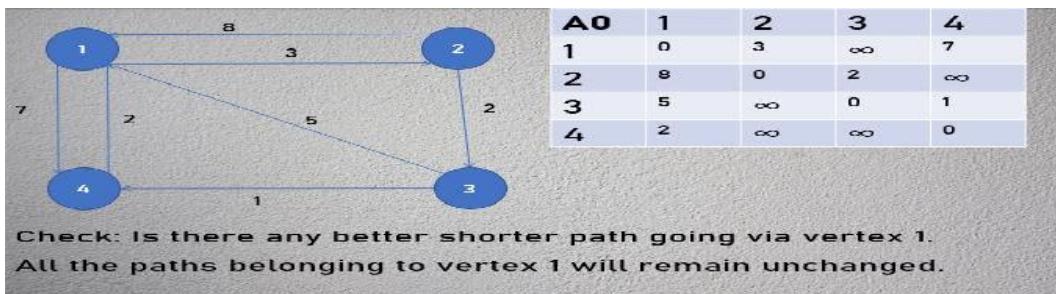


Initially by seeing the weights, we need to fill the values in the matrix. As from vertex 1 to vertex 1, there is no path, so it will be 0. The same thing follows. If there is a path, then we fill the value. If there is no path, then we will fill infinity in place. Like this we will fill all the values.

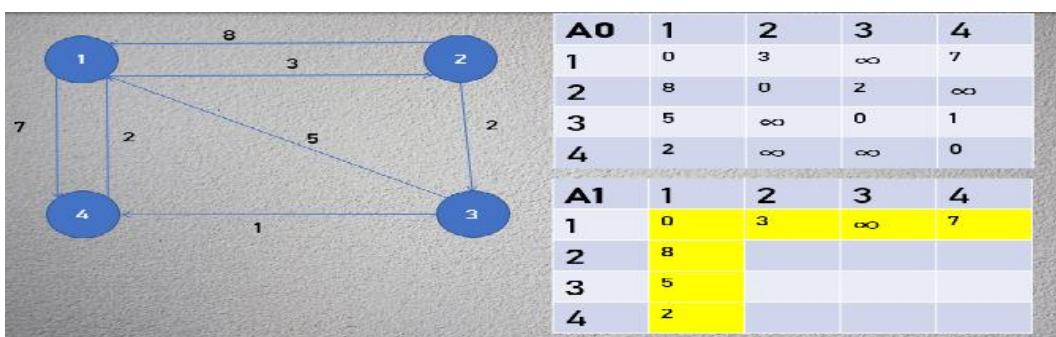


Unit 05: Dynamic Programming

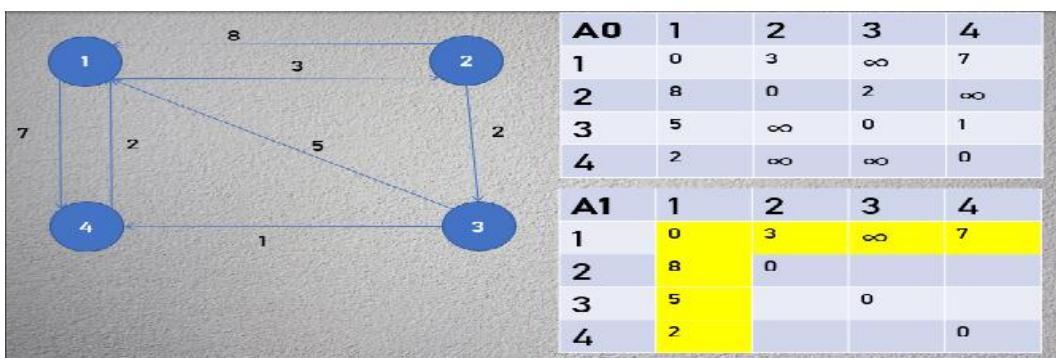
First, we will take the intermediate vertex as vertex 1. Now we will check for the shorter path going via, if that exist, then will take the shorter value out of both. But all the paths belonging to vertex 1 will remain unchanged.



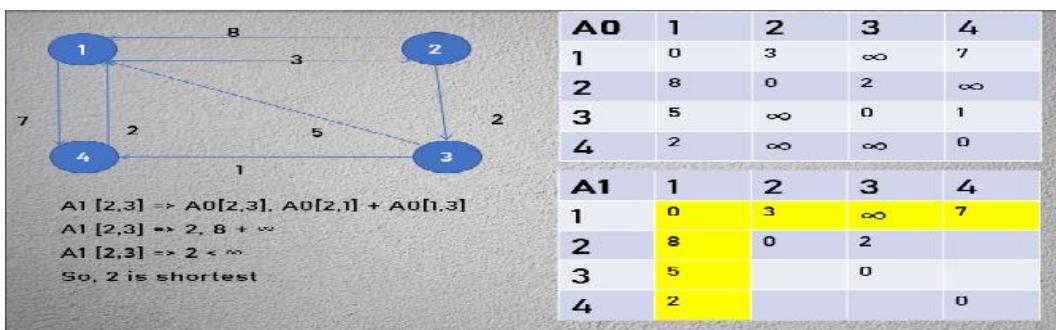
All the paths belonging to vertex 1 will remain unchanged. So that is highlighted in yellow colour.



No self-loops here so 0 will be put in place.

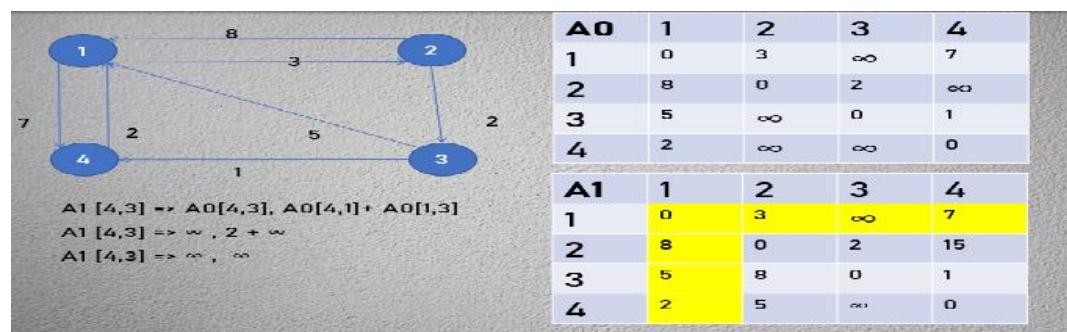
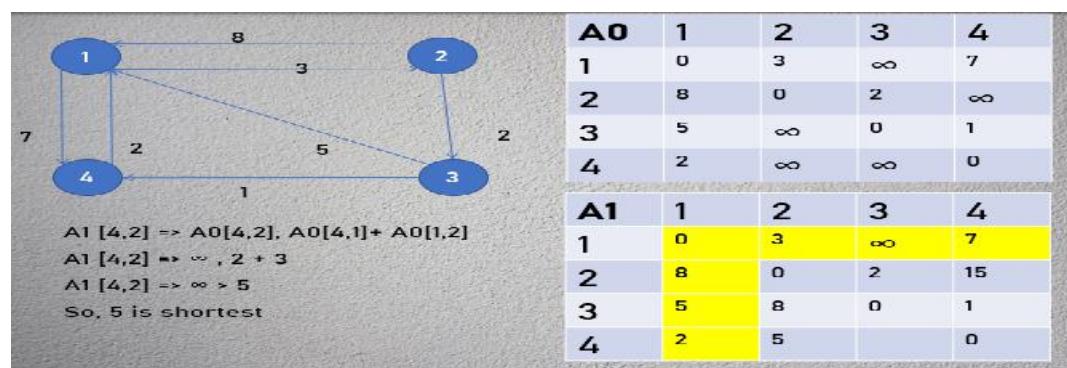
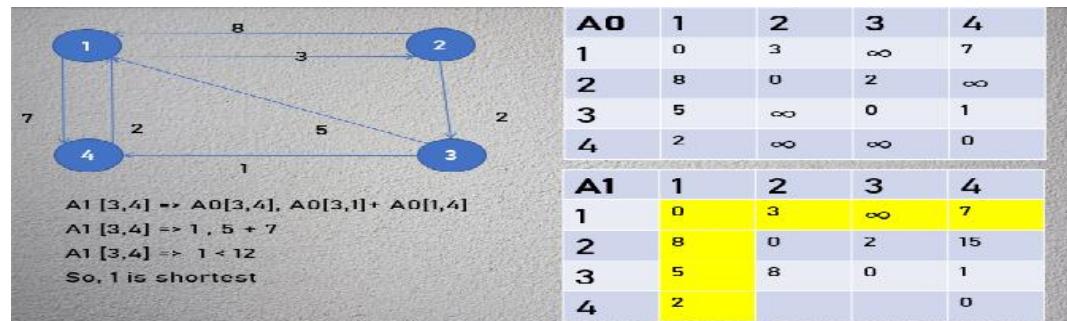
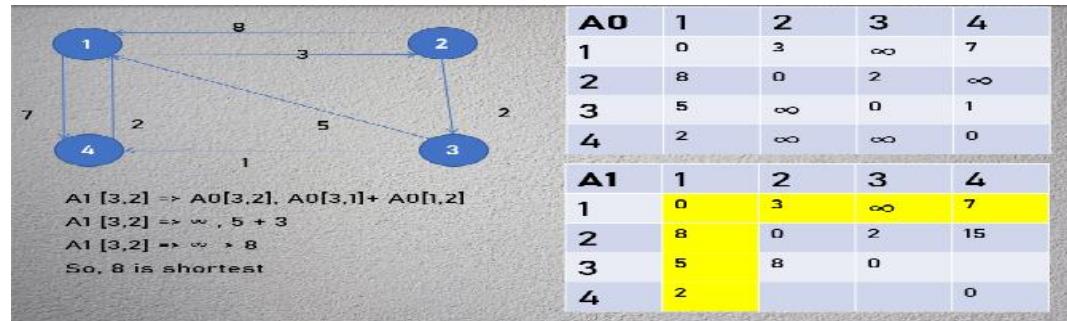
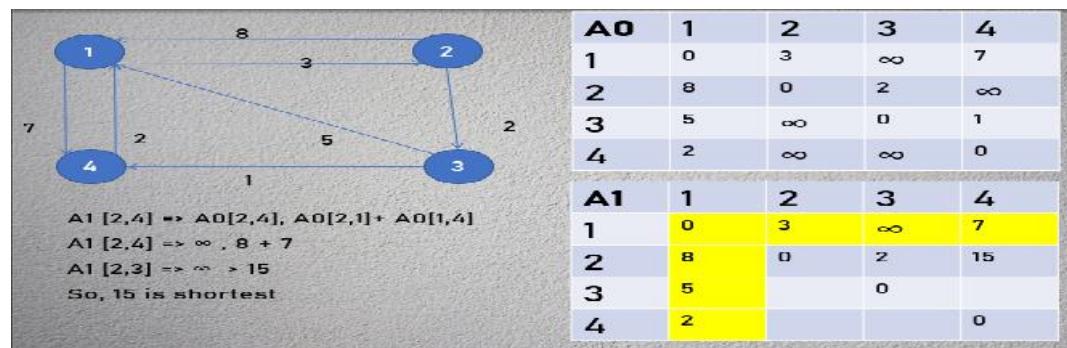


We must take the minimum value out of 2. From vertex 2 to vertex 3, a direct path is there which has weight 2 and if we follow the indirect path, it is giving us weight as 8+infinity which is equal to infinity. So, out of these, 2 is shortest. So 2 is placed here.

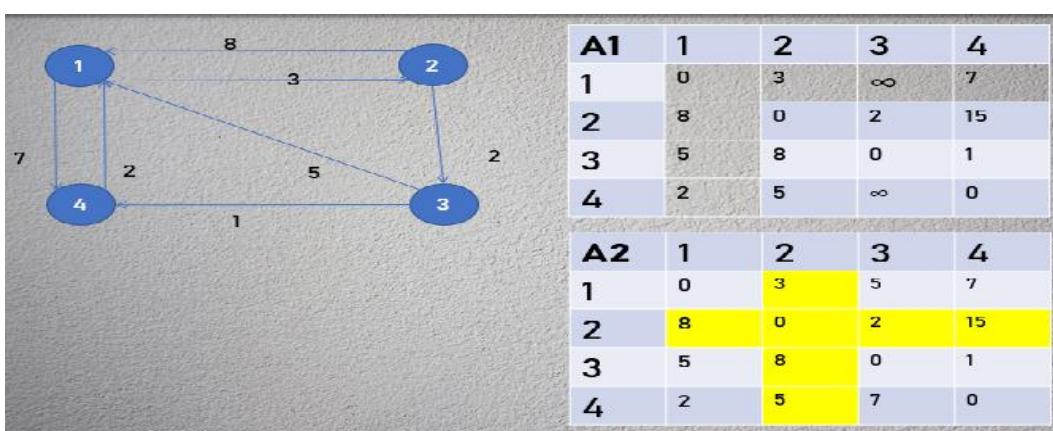
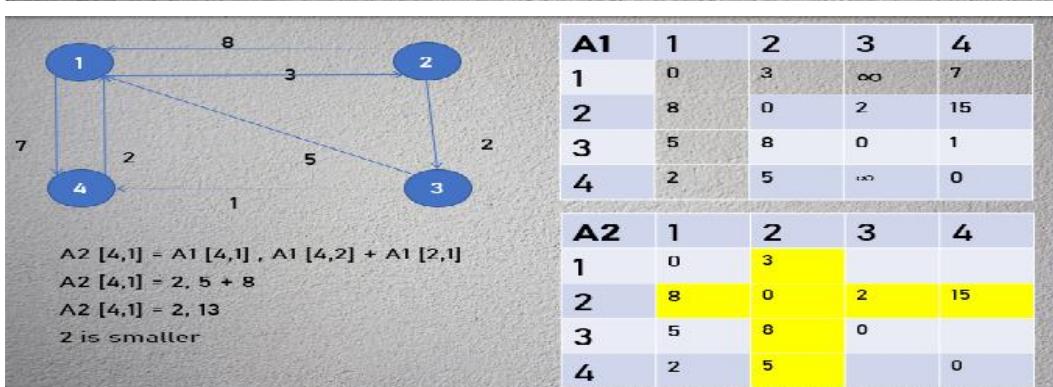
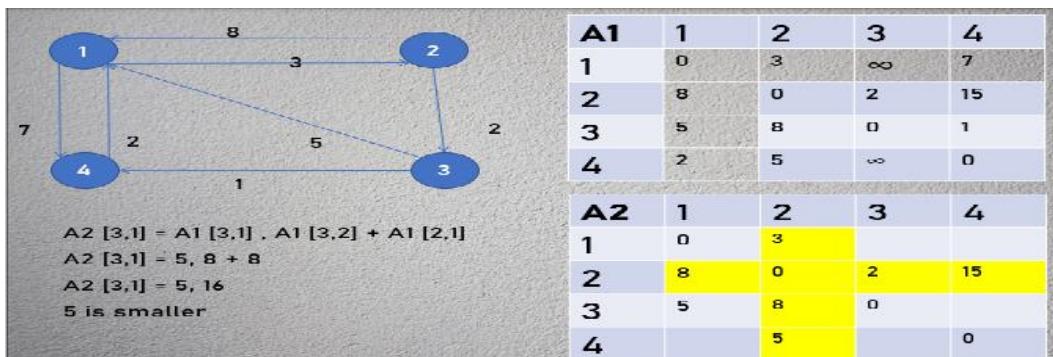
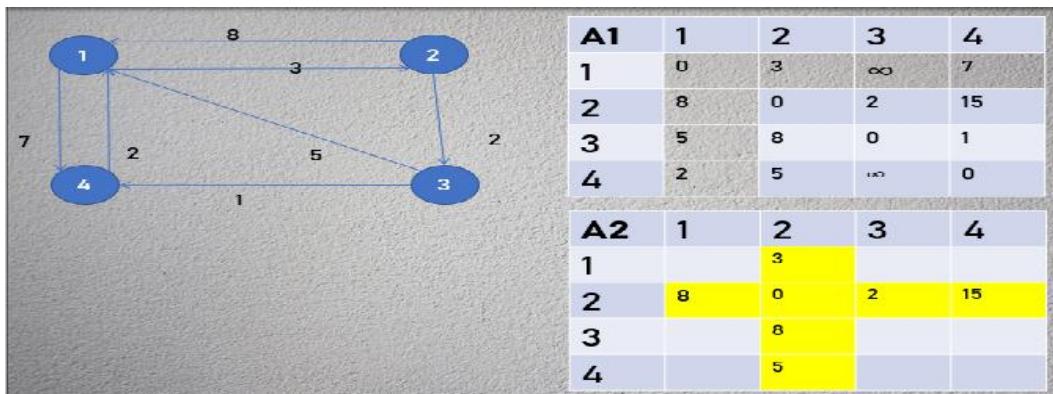


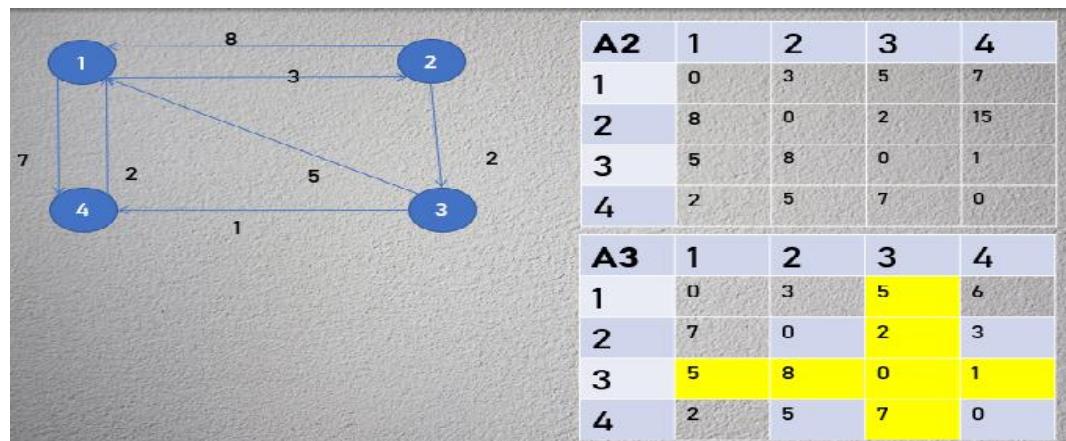
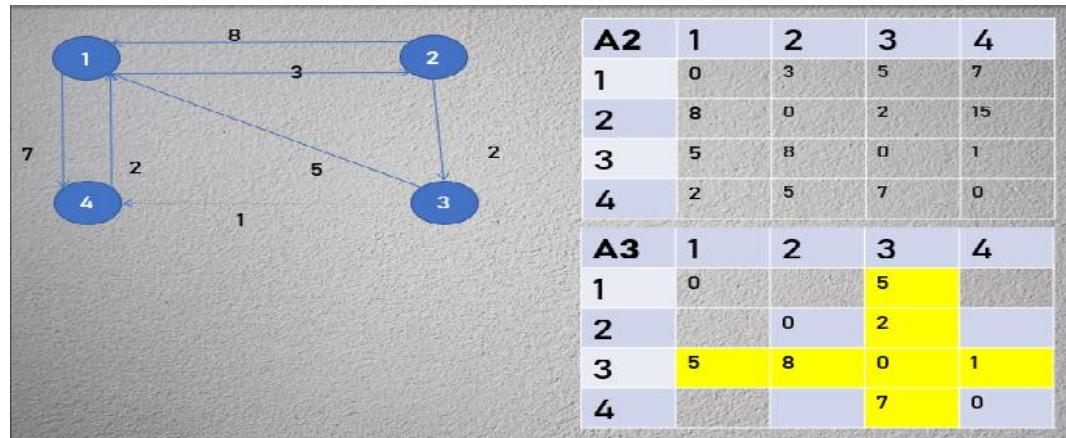
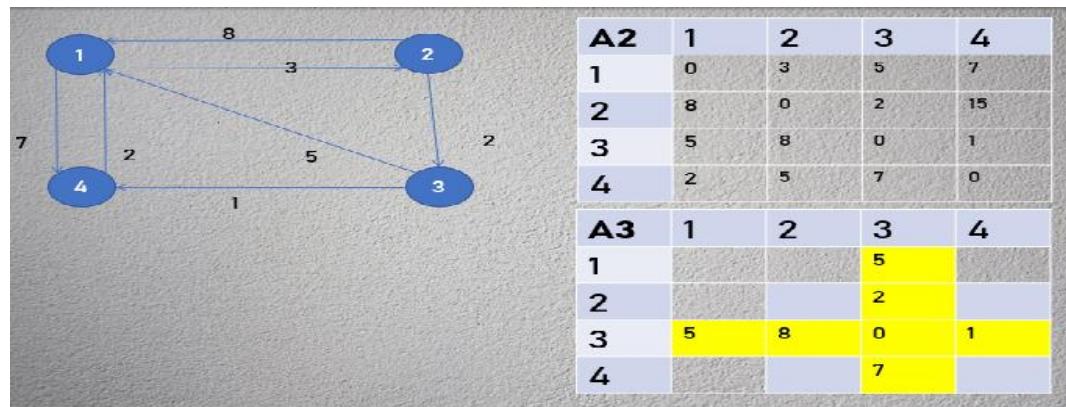
Like this we are going to find out the shortest between all the directed paths and indirect paths.

Algorithm Design and Analysis

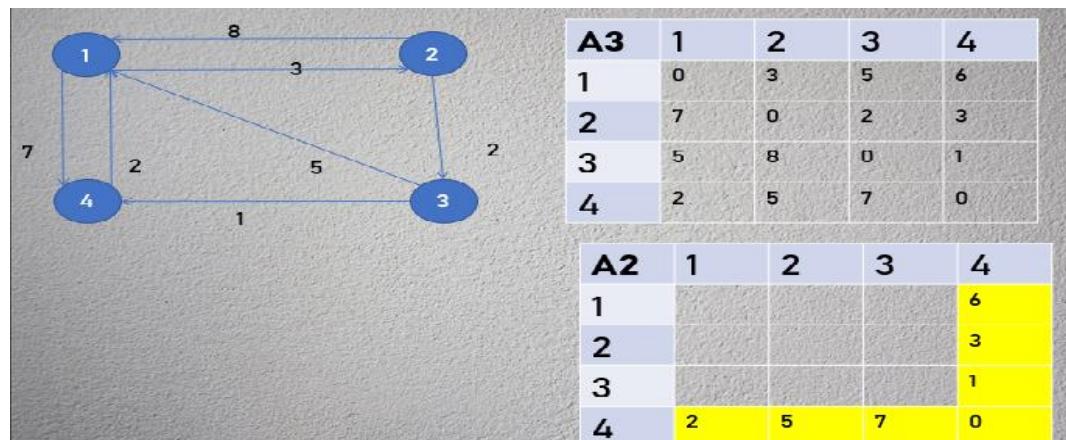


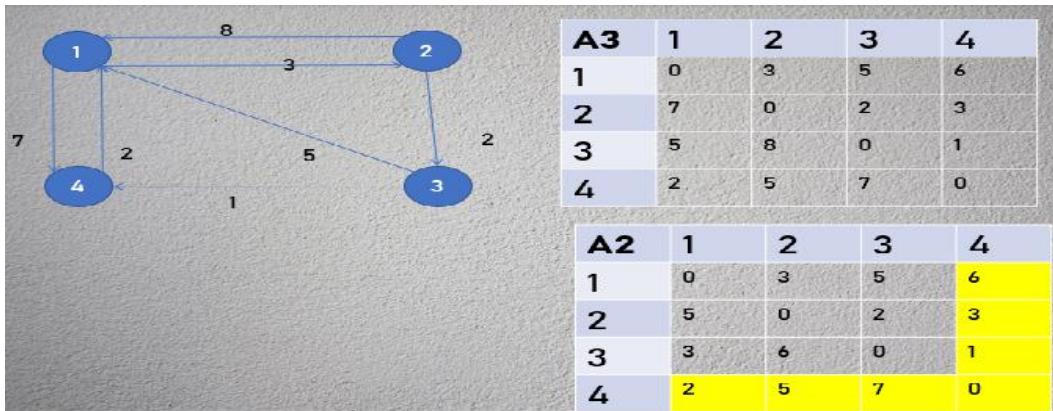
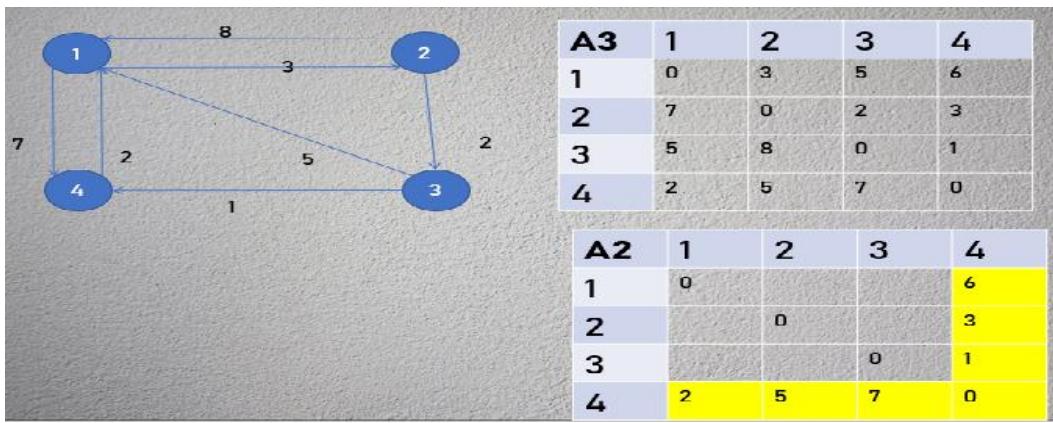
Now same process is followed by taking vertex 2 as an intermediate vertex.

Unit 05: Dynamic Programming

Algorithm Design and Analysis

Same process is followed by taking vertex 4 as an intermediate vertex.





So, out of all the comparisons and taking the values, we can generate the formula as:

$$A^k[i,j] = \min\{ A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j] \}$$

Algorithm:

The algorithm for this can be written as:

```
for (k=1; k<=n; k++)
{
    for (i=1, i<=n; i++)
    {
        for (j=1; j<=n; j++)
        {
            a[i,j] = min(a[i,j], a[i,k] + a[k,j]);
        }
    }
}
```

Time taken

The time taken by the algorithm to find the all-pair shortest path is $\Theta(n^3)$.

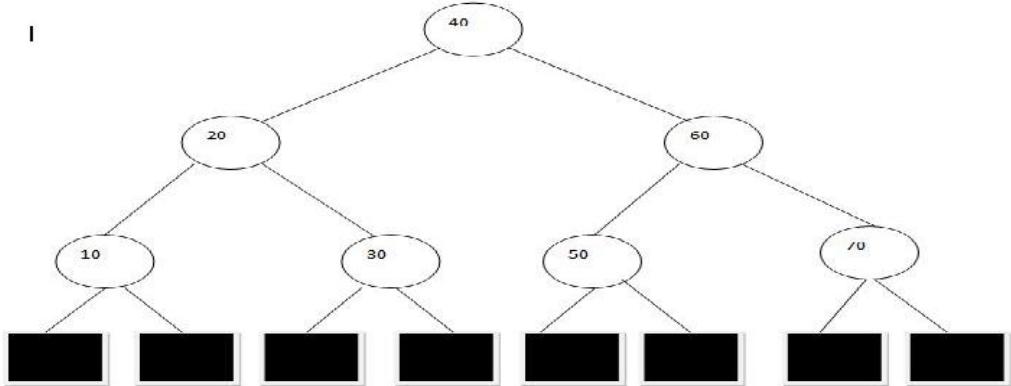
5.2 Optimal Binary Search Tree Problem

The binary search tree is a node-based binary tree data structure which has the following properties:

Algorithm Design and Analysis

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

Below figure shows a binary search tree. Here in the tree try searching for any element and find the number of comparisons.

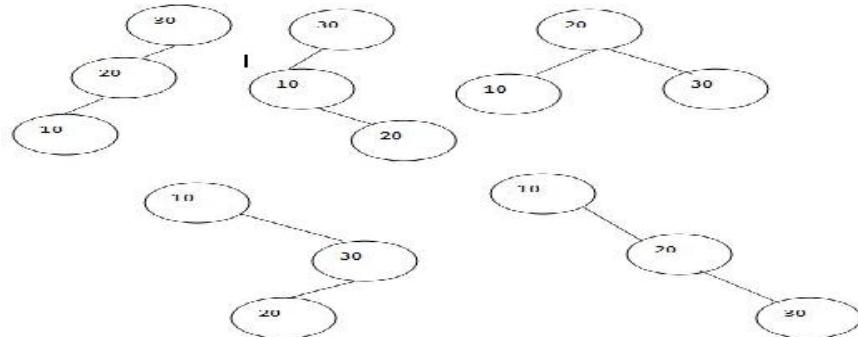


The possible searches can be of two types. There may be successful and unsuccessful searches.

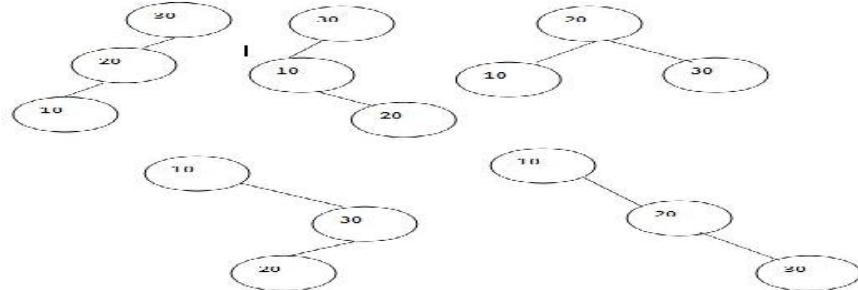
- **Successful search:** The key element is found.
- **Unsuccessful search:** The key element is not found, and it is represented by dummy nodes.

If we talk about the number of comparisons, then the maximum comparisons are 3 which is the height of binary search tree. The cost will be number of comparisons.

Number of possible binary search trees: $T(n) = 2^n C_n / (n+1)$ where n is the number of key elements. For example: Keys: 10, 20, 30. Then $T(3) = 2^3 C_3 / (3+1) = 6 C_3 / (3+1) = 6 \cdot 3! / 4 = 5$. The five possible binary search trees out of three keys 10, 20 and 30 are given below.



Cost of searching



The cost of searching in third binary search tree is 2 and in the rest of trees, the cost of 3. SO, we prefer tree no 2. The conclusion on this is if the height of binary search tree is less then the number

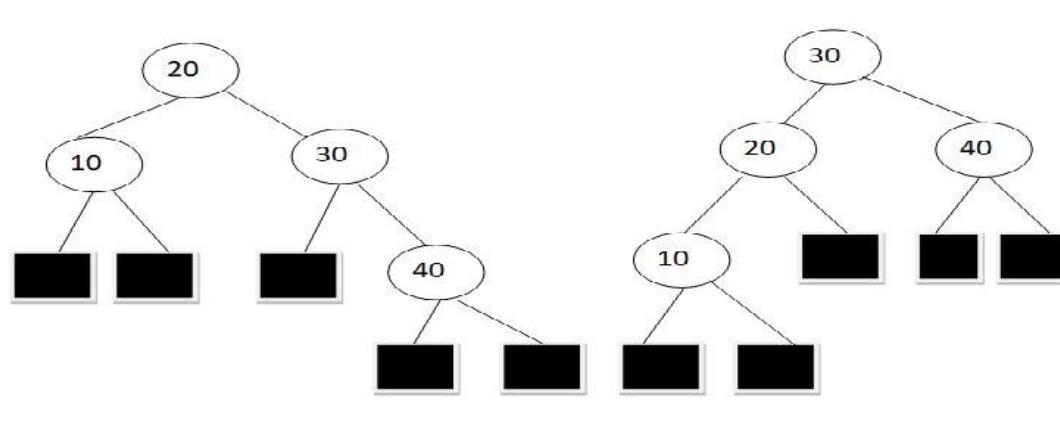
Unit 05: Dynamic Programming

of comparisons will be less. So, the height of binary search tree should be minimum so that the cost is less.

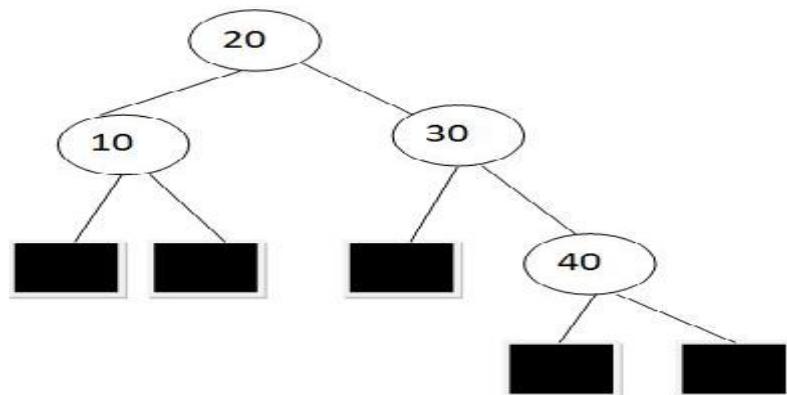
Probability of searching

In this successful and unsuccessful search probability will be counted.

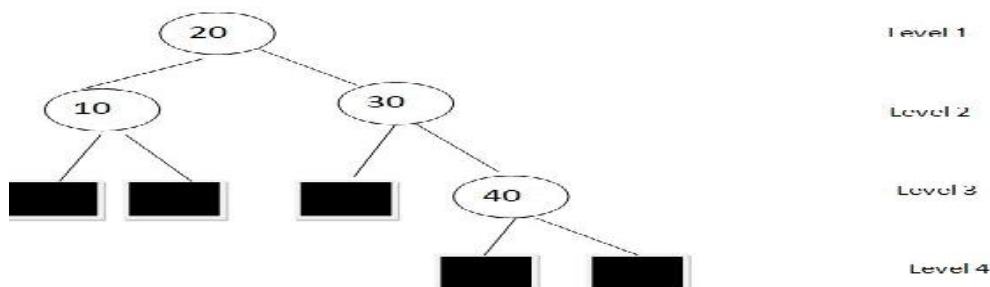
- **Keys:** 10 20 30 40
- **P_i:** .1 .2 .1 .2
- **Q_i:** .1 .05 .15 .05 .05
- **P_i:** Successful search probability
- **Q_i:** Unsuccessful search probability
- Total will be 1.

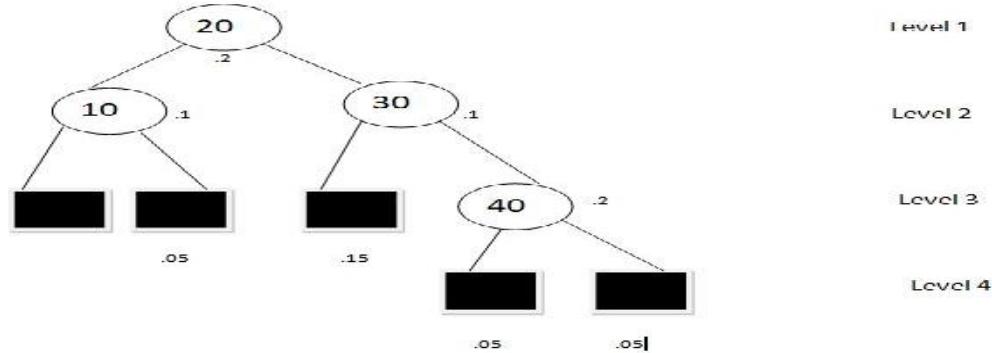


For 4 keys, 14 different binary search trees are possible. Out of these 14 different binary searches one binary search tree is shown below.



If there are n keys, there will be n+1 square nodes as shown. The cost of searching in binary search tree is calculated as:





So, cost of searching will be: $1*0.2 + 2*0.1 + 2*0.1 + 3*0.2 + 2*0.1 + 2*0.05 + 2*0.15 + 3*0.05 + 3*0.05 = 2.1$

So, the formula for cost of existing binary search tree can be written as:

- Cost $[0,n] = \sum_{1 \leq i \leq n} P_i * \text{level}(a_i) + \sum_{0 \leq i \leq n} Q_i * (\text{level}(E_i) - 1)$

So, from above discussion, we can conclude that the optimal binary search tree is:

- The cost of searching is minimum.
- Cost is dependent upon the height of BST.
- Out of 4 keys, 14 BST are possible, and we want the best one which gives us the minimum cost of searching.

But few problems in BST are if the keys and probabilities are given then we have to generate a BST such that the cost is minimum. One solution is to draw all the 14 trees and find out the cost and select the tree which gives us the minimum cost. Other solution is to apply the dynamic programming. Here the dynamic programming tries you a faster method for trying all possible BST and picking up the best one without trying all of them.

- $C[i, j] = \min_{i < k < j} \{ C[i, k-1] + C[k, j] \} + W[i, j]$
- $C[0,3] = \min_{0 < k < 3} \{ C[0,0] + C[1,3], C[0,1] + C[2,3], C[0,2] + C[3,3] \} + W[0,3]$
- Now again apply on $C[1,3]$ and so on. To know the answer of big value, we should know the answer of smaller values.
- As per the DP approach, we should know the smaller value from that we can find the larger values.
- $C[0,3] = \min_{0 < k < 3} \{ C[0,0] + C[1,3], C[0,1] + C[2,3], C[0,2] + C[3,3] \} + W[0,3]$
- For calculation of $C[0,3]$ we should know:
 - $C[0,0], C[1,1], C[2,2], C[3,3]$
 - $C[0,1], C[1,2], C[2,3]$
 - $C[0,2], C[1,3]$
 - $C[0,3]$
 - $C[0,0], C[1,1], C[2,2], C[3,3]$ $j-i=0$
 - $C[0,1], C[1,2], C[2,3]$ $j-i=1$
 - $C[0,2], C[1,3]$ $j-i=2$
 - $C[0,3]$ $j-i=3$
- In this problem, we will not use formula, but we will use the table which is generated from that formula.
- We also must consider which k is giving us the minimum cost.
- We also must consider the weight differences:
- $W[0,2] = q_0 + p_1 + q_1 + p_2 + q_2$

Unit 05: Dynamic Programming

- $W[0,3] = q_0 + p_1 + q_1 + p_2 + q_2 + p_3 + q_3$
- So, $W[0,3] = W[0,2] + p_3 + q_3$
- So, $W[i,j] = W[i,j-1] + p_j + q_j$



Example:

- Keys = { 10, 20, 30, 40 }
- $P_i = \{ 3, 3, 1, 1 \}$
- $Q_i = \{ 2, 3, 1, 1, 1 \}$
- These probabilities are in decimal point. These must be divided by 16 so that we can get the total probability as 1.

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} =$ $C_{00} =$ $R_{00} =$	$W_{11} =$ $C_{11} =$ $R_{11} =$	$W_{22} =$ $C_{22} =$ $R_{22} =$	$W_{33} =$ $C_{33} =$ $R_{33} =$	$W_{44} =$ $C_{44} =$ $R_{44} =$
j-i=1	$W_{01} =$ $C_{01} =$ $R_{01} =$	$W_{12} =$ $C_{12} =$ $R_{12} =$	$W_{23} =$ $C_{23} =$ $R_{23} =$	$W_{34} =$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

$$W[0,0]=Q_0=2, W[1,1]=Q_1=3, W[2,2]=Q_2=1, W[3,3]=Q_3=1, W[4,4]=1$$

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} =$ $R_{00} =$	$W_{11} = 3$ $C_{11} =$ $R_{11} =$	$W_{22} = 1$ $C_{22} =$ $R_{22} =$	$W_{33} = 1$ $C_{33} =$ $R_{33} =$	$W_{44} = 1$ $C_{44} =$ $R_{44} =$

Algorithm Design and Analysis

j-i=1	$W_{01} =$ $C_{01} =$ $R_{01} =$	$W_{12} =$ $C_{12} =$ $R_{12} =$	$W_{23} =$ $C_{23} =$ $R_{23} =$	$W_{34} =$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

$$C[0,0] = C[1,1] = C[2,2] = C[3,3] = C[4,4] = 0$$

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} =$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} =$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} =$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} =$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} =$
j-i=1	$W_{01} =$ $C_{01} =$ $R_{01} =$	$W_{12} =$ $C_{12} =$ $R_{12} =$	$W_{23} =$ $C_{23} =$ $R_{23} =$	$W_{34} =$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

$$R[0,0] = R[1,1] = R[2,2] = R[3,3] = R[4,4] = 0$$

	j=0	j=1	j=2	j=3	j=4
--	-----	-----	-----	-----	-----

Unit 05: Dynamic Programming

j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} =$ $C_{01} =$ $R_{01} =$	$W_{12} =$ $C_{12} =$ $R_{12} =$	$W_{23} =$ $C_{23} =$ $R_{23} =$	$W_{34} =$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				



Example:

$$W[i,j] = W[i,j-1] + p_j + q_j$$

$$W[0,1] = W[0,0] + P1 + Q1 = 2 + 3 + 3 = 8$$

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} =$ $R_{01} =$	$W_{12} =$ $C_{12} =$ $R_{12} =$	$W_{23} =$ $C_{23} =$ $R_{23} =$	$W_{34} =$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			

Algorithm Design and Analysis

j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				
-------	--	--	--	--	--

$$W[1,2] = W[1,1] + P2 + Q2 = 3 + 3 + 1 = 7$$

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} =$ $R_{01} =$	$W_{12} = 7$ $C_{12} =$ $R_{12} =$	$W_{23} =$ $C_{23} =$ $R_{23} =$	$W_{34} =$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

$$W[2,3] = W[2,2] + P3 + Q3 = 1 + 1 + 1 = 3$$

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} =$ $R_{01} =$	$W_{12} = 7$ $C_{12} =$ $R_{12} =$	$W_{23} = 3$ $C_{23} =$ $R_{23} =$	$W_{34} =$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		

Unit 05: Dynamic Programming

j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} =$ $R_{01} =$	$W_{12} = 7$ $C_{12} =$ $R_{12} =$	$W_{23} = 3$ $C_{23} =$ $R_{23} =$	$W_{34} = 3$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} =$ $C_{02} =$ $R_{02} =$	$W_{13} =$ $C_{13} =$ $R_{13} =$	$W_{24} =$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} =$ $R_{01} =$	$W_{12} = 7$ $C_{12} =$ $R_{12} =$	$W_{23} = 3$ $C_{23} =$ $R_{23} =$	$W_{34} = 3$ $C_{34} =$ $R_{34} =$	

Algorithm Design and Analysis

j-i=2	$W_{02} = 12$ $C_{02} =$ $R_{02} =$	$W_{13} = 9$ $C_{13} =$ $R_{13} =$	$W_{24} = 5$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} =$ $C_{03} =$ $R_{03} =$	$W_{14} =$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} =$ $R_{01} =$	$W_{12} = 7$ $C_{12} =$ $R_{12} =$	$W_{23} = 3$ $C_{23} =$ $R_{23} =$	$W_{34} = 3$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} = 12$ $C_{02} =$ $R_{02} =$	$W_{13} = 9$ $C_{13} =$ $R_{13} =$	$W_{24} = 5$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} = 14$ $C_{03} =$ $R_{03} =$	$W_{14} = 11$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} =$ $C_{04} =$ $R_{04} =$				

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$

Unit 05: Dynamic Programming

j-i=1	$W_{01} = 8$ $C_{01} =$ $R_{01} =$	$W_{12} = 7$ $C_{12} =$ $R_{12} =$	$W_{23} = 3$ $C_{23} =$ $R_{23} =$	$W_{34} = 3$ $C_{34} =$ $R_{34} =$	
j-i=2	$W_{02} = 12$ $C_{02} =$ $R_{02} =$	$W_{13} = 9$ $C_{13} =$ $R_{13} =$	$W_{24} = 5$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} = 14$ $C_{03} =$ $R_{03} =$	$W_{14} = 11$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} = 16$ $C_{04} =$ $R_{04} =$				



Example: $C[i, j] = \min_{i < k \leq j} \{C[i, k-1] + C[k, j]\} + W[i, j]$

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} = 8$ $R_{01} =$	$W_{12} = 7$ $C_{12} = 7$ $R_{12} =$	$W_{23} = 3$ $C_{23} = 3$ $R_{23} =$	$W_{34} = 3$ $C_{34} = 3$ $R_{34} =$	
j-i=2	$W_{02} = 12$ $C_{02} =$ $R_{02} =$	$W_{13} = 9$ $C_{13} =$ $R_{13} =$	$W_{24} = 5$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} = 14$ $C_{03} =$ $R_{03} =$	$W_{14} = 11$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} = 16$ $C_{04} =$ $R_{04} =$				

Algorithm Design and Analysis

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} = 8$ $R_{01} = 1$	$W_{12} = 7$ $C_{12} = 7$ $R_{12} = 2$	$W_{23} = 3$ $C_{23} = 3$ $R_{23} = 3$	$W_{34} = 3$ $C_{34} = 3$ $R_{34} = 4$	
j-i=2	$W_{02} = 12$ $C_{02} =$ $R_{02} =$	$W_{13} = 9$ $C_{13} =$ $R_{13} =$	$W_{24} = 5$ $C_{24} =$ $R_{24} =$		
j-i=3	$W_{03} = 14$ $C_{03} =$ $R_{03} =$	$W_{14} = 11$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} = 16$ $C_{04} =$ $R_{04} =$				

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} = 8$ $R_{01} = 1$	$W_{12} = 7$ $C_{12} = 7$ $R_{12} = 2$	$W_{23} = 3$ $C_{23} = 3$ $R_{23} = 3$	$W_{34} = 3$ $C_{34} = 3$ $R_{34} = 4$	
j-i=2	$W_{02} = 12$ $C_{02} = 19$ $R_{02} =$	$W_{13} = 9$ $C_{13} = 12$ $R_{13} =$	$W_{24} = 5$ $C_{24} = 8$ $R_{24} =$		
j-i=3	$W_{03} = 14$ $C_{03} =$ $R_{03} =$	$W_{14} = 11$ $C_{14} =$ $R_{14} =$			

Unit 05: Dynamic Programming

j-i=4	$W_{04} = 16$ $C_{04} =$ $R_{04} =$				
-------	---	--	--	--	--

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} = 8$ $R_{01} = 1$	$W_{12} = 7$ $C_{12} = 7$ $R_{12} = 2$	$W_{23} = 3$ $C_{23} = 3$ $R_{23} = 3$	$W_{34} = 3$ $C_{34} = 3$ $R_{34} = 4$	
j-i=2	$W_{02} = 12$ $C_{02} = 19$ $R_{02} = 1$	$W_{13} = 9$ $C_{13} = 12$ $R_{13} = 2$	$W_{24} = 5$ $C_{24} = 8$ $R_{24} = 3$		
j-i=3	$W_{03} = 14$ $C_{03} =$ $R_{03} =$	$W_{14} = 11$ $C_{14} =$ $R_{14} =$			
j-i=4	$W_{04} = 16$ $C_{04} =$ $R_{04} =$				

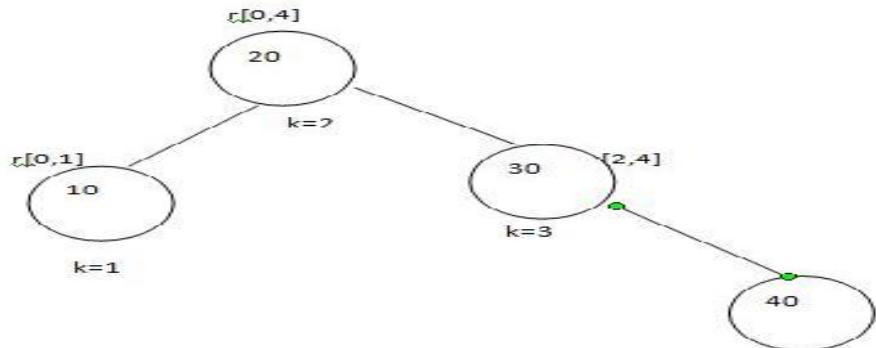
	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} = 8$ $R_{01} = 1$	$W_{12} = 7$ $C_{12} = 7$ $R_{12} = 2$	$W_{23} = 3$ $C_{23} = 3$ $R_{23} = 3$	$W_{34} = 3$ $C_{34} = 3$ $R_{34} = 4$	
j-i=2	$W_{02} = 12$ $C_{02} = 19$ $R_{02} = 1$	$W_{13} = 9$ $C_{13} = 12$ $R_{13} = 2$	$W_{24} = 5$ $C_{24} = 8$ $R_{24} = 3$		

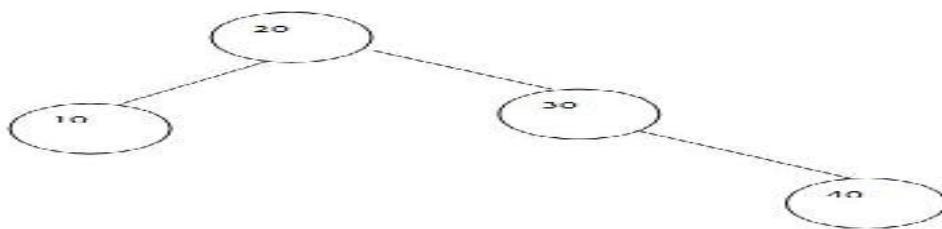
Algorithm Design and Analysis

j-i=3	$W_{03} = 14$ $C_{03} = 25$ $R_{03} = 2$	$W_{14} = 11$ $C_{14} = 19$ $R_{14} = 2$			
j-i=4	$W_{04} = 16$ $C_{04} =$ $R_{04} =$				

	j=0	j=1	j=2	j=3	j=4
j-i=0	$W_{00} = 2$ $C_{00} = 0$ $R_{00} = 0$	$W_{11} = 3$ $C_{11} = 0$ $R_{11} = 0$	$W_{22} = 1$ $C_{22} = 0$ $R_{22} = 0$	$W_{33} = 1$ $C_{33} = 0$ $R_{33} = 0$	$W_{44} = 1$ $C_{44} = 0$ $R_{44} = 0$
j-i=1	$W_{01} = 8$ $C_{01} = 8$ $R_{01} = 1$	$W_{12} = 7$ $C_{12} = 7$ $R_{12} = 2$	$W_{23} = 3$ $C_{23} = 3$ $R_{23} = 3$	$W_{34} = 3$ $C_{34} = 3$ $R_{34} = 4$	
j-i=2	$W_{02} = 12$ $C_{02} = 19$ $R_{02} = 1$	$W_{13} = 9$ $C_{13} = 12$ $R_{13} = 2$	$W_{24} = 5$ $C_{24} = 8$ $R_{24} = 3$		
j-i=3	$W_{03} = 14$ $C_{03} = 25$ $R_{03} = 2$	$W_{14} = 11$ $C_{14} = 19$ $R_{14} = 2$			
j-i=4	$W_{04} = 16$ $C_{04} = 32$ $R_{04} = 2$				

So, the optimal Binary Search Tree



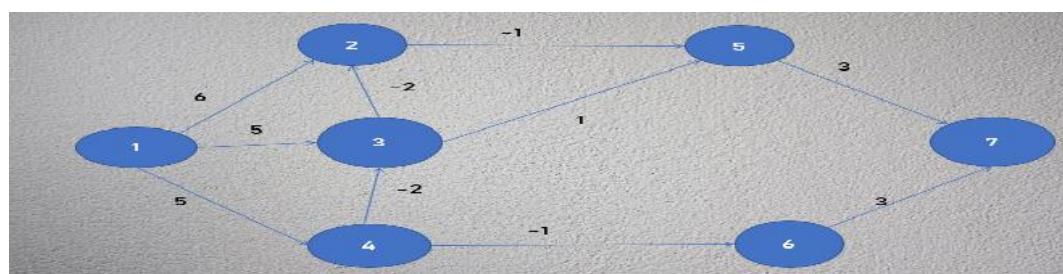


5.3 Bellman Ford Algorithm

In single source shortest path problem, given a weighted directed graph, we must start from the starting vertex and find out the shortest path to all the other vertices from that vertex. The few considerations of Dijkstra algorithm are: this algorithm does the same task, it will not work properly if the edges have negative weights and it may or may not give accurate results.

The Bellman Ford algorithm is used for this as: this algorithm gives the accurate results, it follows dynamic programming approach and in DP, we try out all possible solutions and pick up the best one. The concept behind the algorithm is we go on repeatedly relaxing all the edges for $(N-1)$ times. Here, N is the number of vertices.

Example

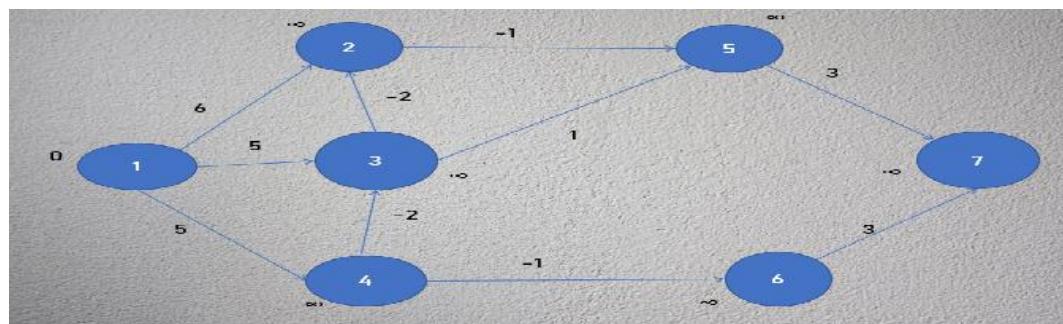


The Relaxation formula is: if $(d[u] + C[u,v] < d[v])$ then

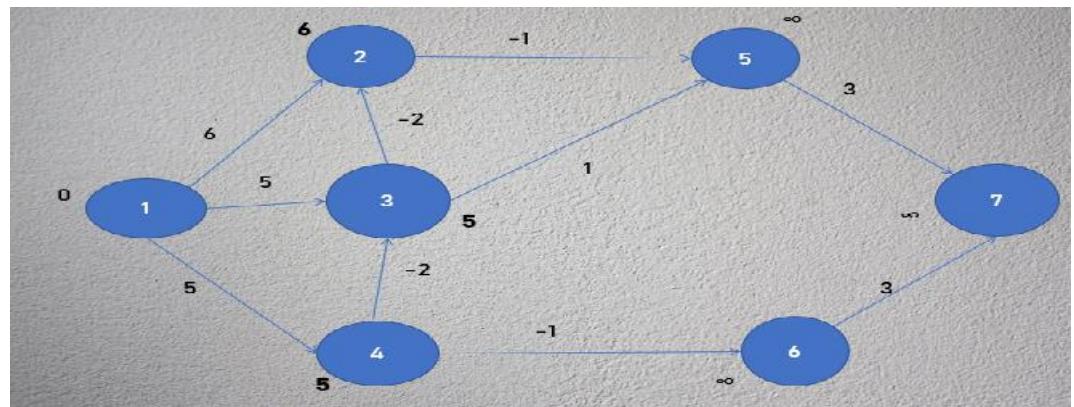
$$d[v] = d[u] + C[u, v]$$

Prepare a list of all edges: List of edges: (1,2), (1,3), (1,4), (2,5), (3,2), (3,5), (4,3), (4,6), (5,7), (6,7). We have to relax all these edges 6 times as $7 - 1 = 6$.

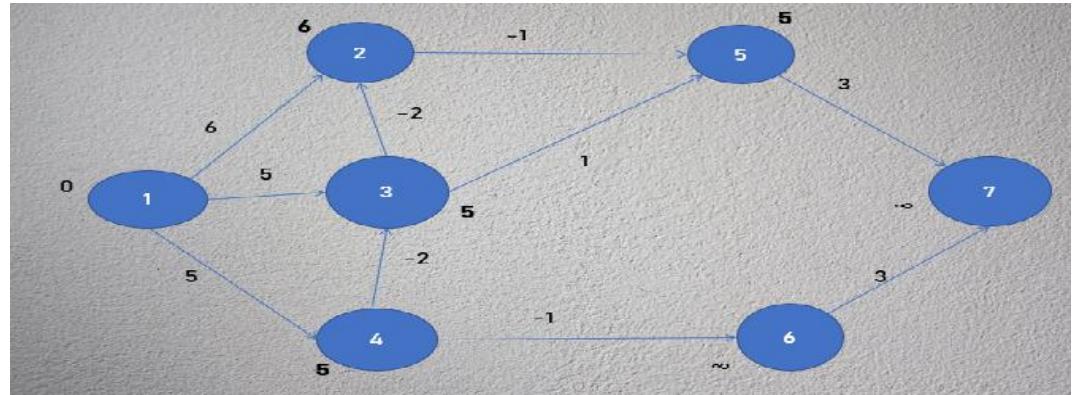
1st iteration:



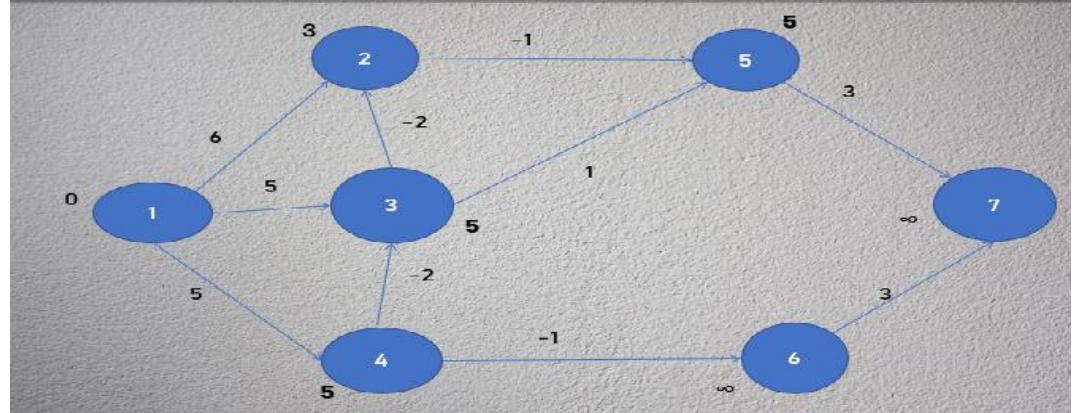
$(1, 2) \rightarrow 0 + 6 < \infty$, $6 < \infty$, So change ∞ as 6. $(1, 3) \rightarrow 0 + 5 < \infty$, $5 < \infty$, So change ∞ as 5. $(1, 4) \rightarrow 0 + 5 < \infty$, $5 < \infty$, So change ∞ as 5.



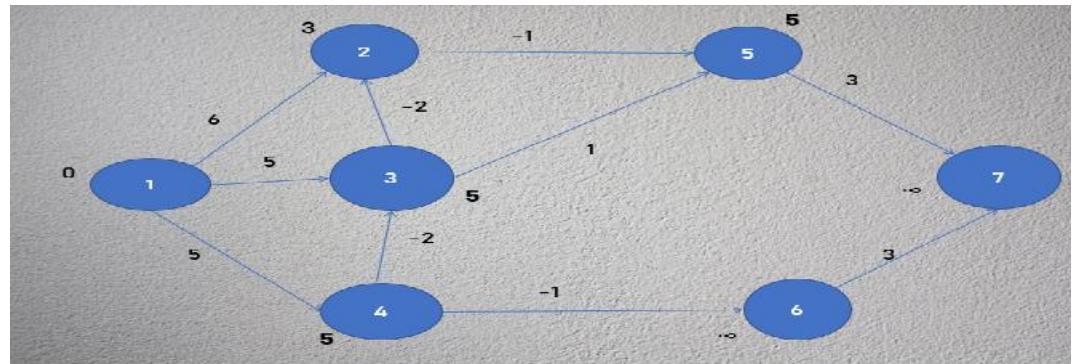
- Take the next edge from the list, i.e., $(2, 5)$. $6 - 1 = 5$



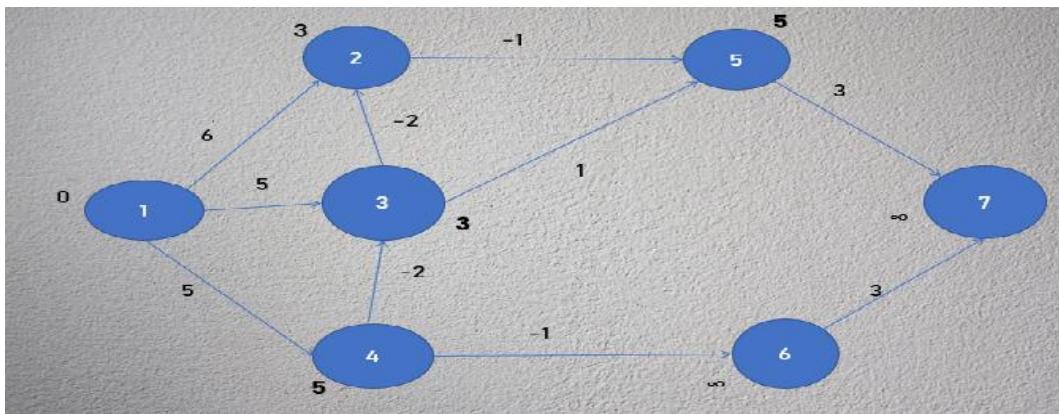
- Take the next edge from the list, i.e., $(3, 2)$. $5 - 2 = 3$



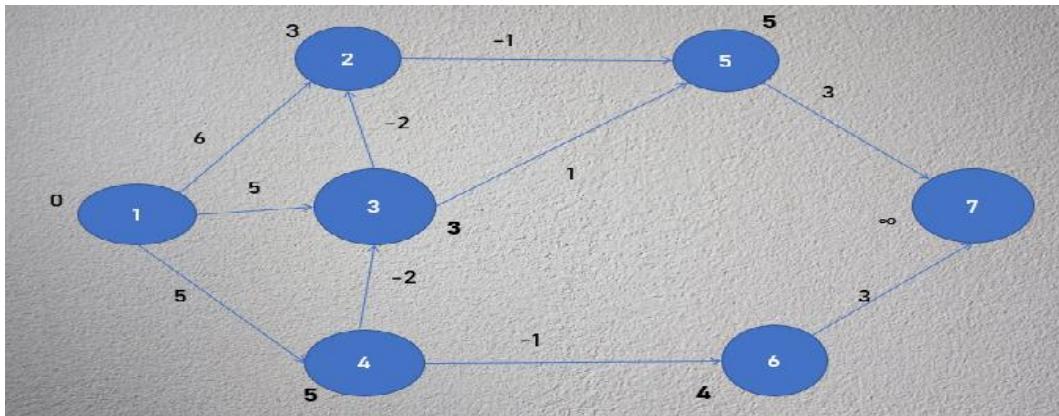
- Take the next edge from the list, i.e., $(3, 5)$. $5 + 1 = 6$. $6 > 5$. So, don't modify this.



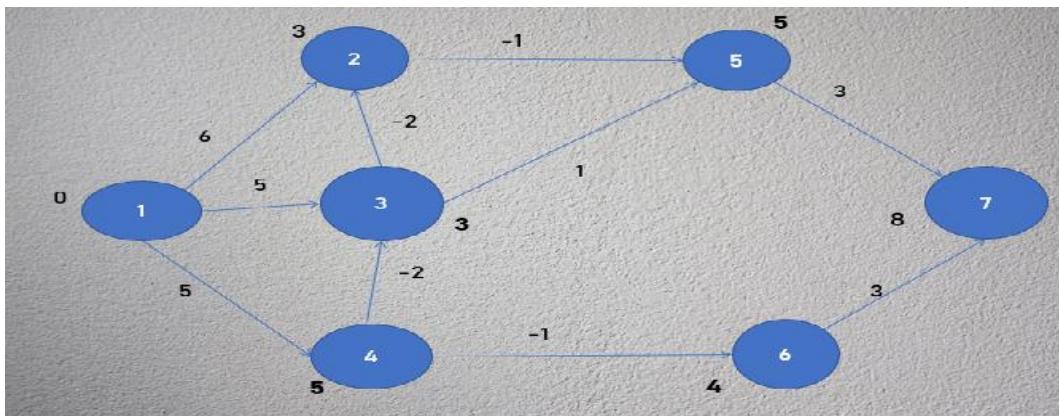
- Take the next edge from the list, i.e., $(4, 3)$. $5 - 2 = 3$. $3 < 5$. So, modify this.

Unit 05: Dynamic Programming

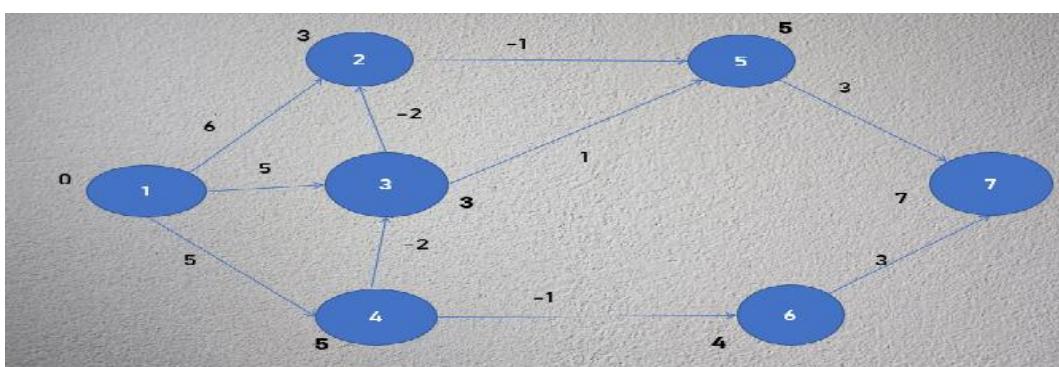
- Take the next edge from the list, i.e., $(4, 6)$. $5 - 1 = 4.4 < \infty$. So, modify this.



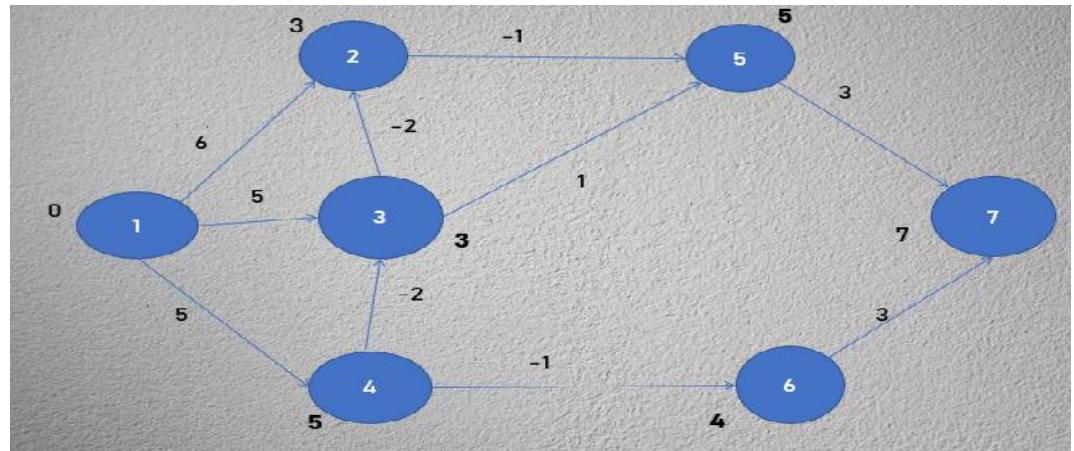
- Take the next edge from the list, i.e., $(5, 7)$. $5 + 3 = 8.8 < \infty$. So, modify this.



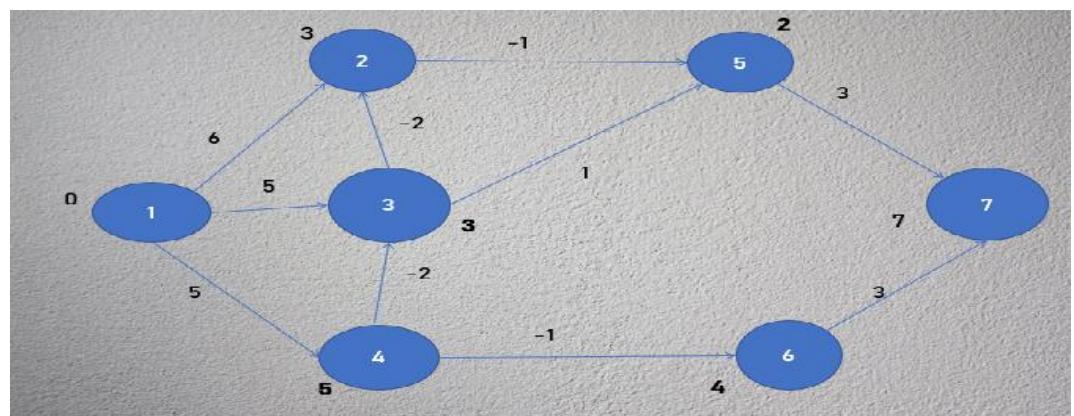
- Take the next edge from the list, i.e., $(6, 7)$. $4 + 3 = 7.7 < 8$. So, modify this.



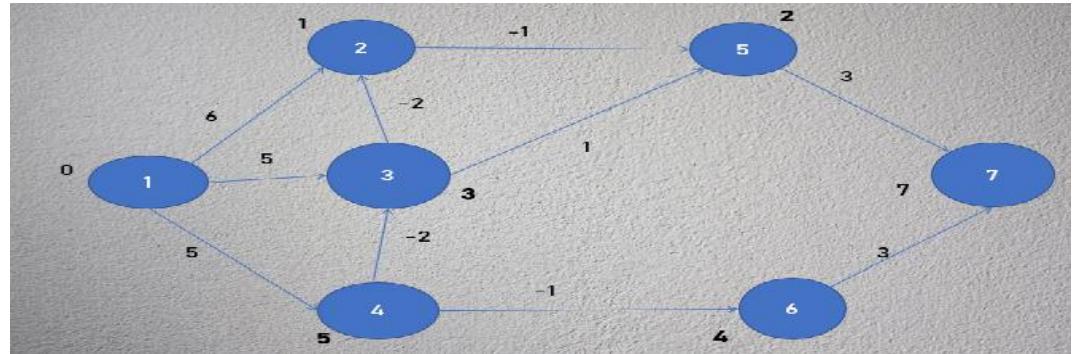
- Till here, all the edges are relaxed for 1 time. Continue it for $n-1 = 7-1 = 6$ times.

Algorithm Design and Analysis2nd iteration

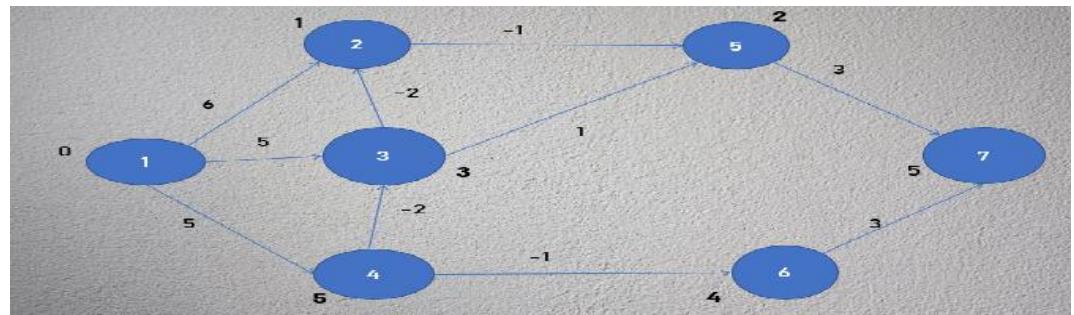
- Relaxation of edges: $(1, 2) \rightarrow 3, (1, 3) \rightarrow 3, (1, 4) \rightarrow 5, (2, 5) \rightarrow 3 - 1 = 2 < 5 \rightarrow 2$



- $(3,2) \rightarrow 3 - 2 = 1 < 3 \rightarrow 1$



- $(3,5) \rightarrow 2, (4, 3) \rightarrow 3, (4,6) \rightarrow 4, (5, 7) \rightarrow 2 + 3 = 5 < 7 \rightarrow 5, (6, 7) \rightarrow 5.$

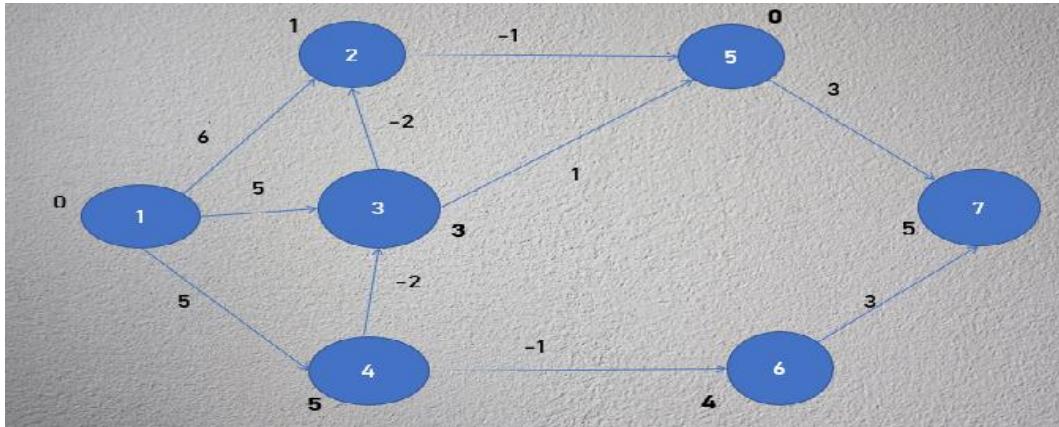


- The procedure is done for 2 times. Still we need to repeat it for 4 more times.

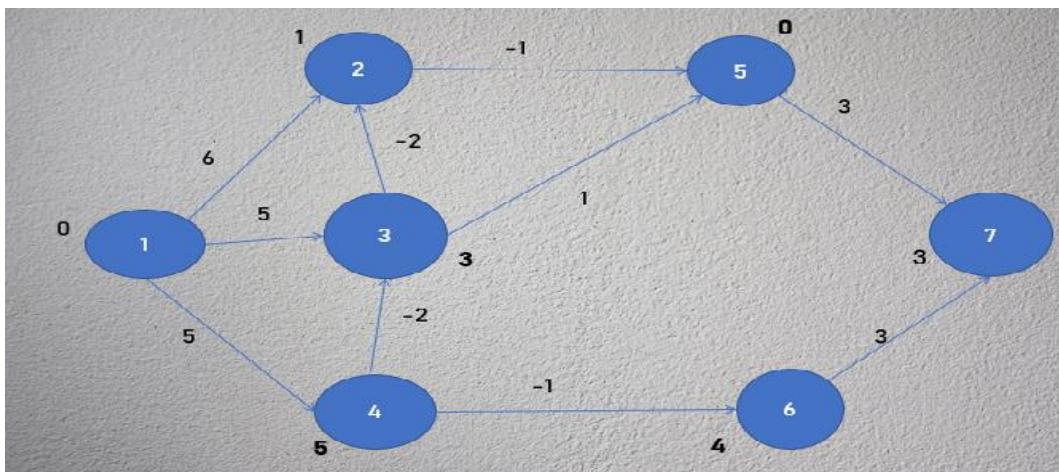
3rd iteration

Unit 05: Dynamic Programming

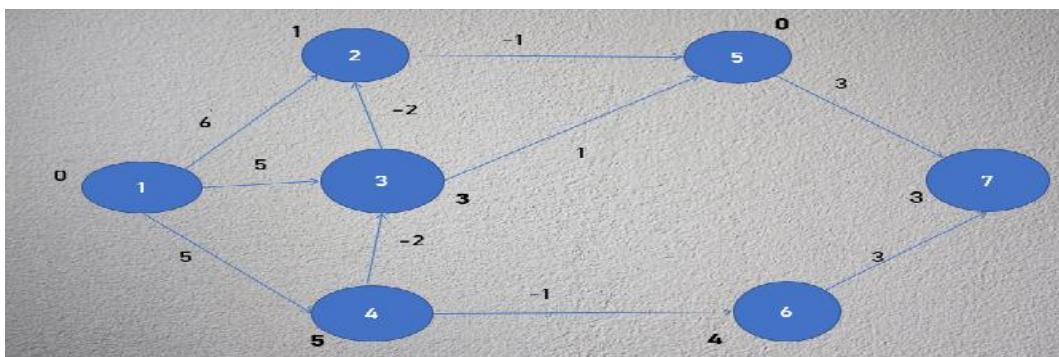
- $(1,2) \rightarrow 1, (1,3) \rightarrow 3, (1,4) \rightarrow 5, (2, 5) \rightarrow 1-1 = 0$



- $(3, 2) \rightarrow 1, (3, 5) \rightarrow 0, (4, 3) \rightarrow 1, (4, 6) \rightarrow 4, (5, 7) \rightarrow 0+3=3 < 5 \rightarrow 3$



- $(6, 7) \rightarrow 3$



4th iteration:

- $(1,2) \rightarrow 1, (1,3) \rightarrow 1, (1,4) \rightarrow 5, (2,5) \rightarrow 0, (3,2) \rightarrow 1, (3,5) \rightarrow 0, (4,3) \rightarrow 1, (4,6) \rightarrow 3, (5,7) \rightarrow 3, (6,7) \rightarrow 3$. Now the values have stopped changing. If we continue doing it for 2 more times, the values will not change.

The values at each vertex are:

- $1 \rightarrow 0, 2 \rightarrow 1, 3 \rightarrow 3, 4 \rightarrow 5, 5 \rightarrow 0, 6 \rightarrow 4, 7 \rightarrow 3$

Time complexity

There are total e edges. $|V| - 1$ no of times the algorithm is relaxing the edges. $O(|V| \cdot |E|)$. If V and E are n and n . Then the time complexity will be $O(n^2)$. For a complete graph, the number of edges will be $n*(n-1)/2$. Time complexity will be $O(n*(n*(n-1)/2)) = O(n^3)$.

Failure of Bellman Ford algorithm

If there are negative cycles (reachable from the source), Bellman-Ford **can** be considered to fail.

5.4 Reliability Design

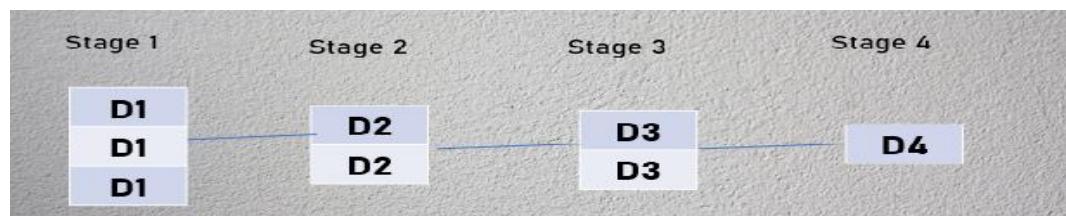
Reliability design or design of reliability is a process that ensures a product, or system, performs a specified function within a given environment over the expected lifetime. Design of reliability often occurs at the design stage – before physical prototyping – and is often part of an overall design for excellence strategy. The complexities of today's technologies make design of reliability more significant – and valuable – than ever before. Some of these reasons include:

- 1) **Product differentiation:** As electronic technologies reach maturity, there are fewer opportunities to set products apart from the competition through traditional metrics – like price and performance.
- 2) **Reliability assurance:** Advanced circuitry, sophisticated power requirements, new components, new material technologies and less robust parts make ensuring reliability increasingly difficult.
- 3) **Cost control:** 70% of a project's budget is allocated to design.
- 4) **Preserving profits:** Products get to market earlier, preventing erosion of sales and market share.

The problematic thing is: we have to set up a system which requires some devices, i.e., D1, D2, D3, D4. These devices are connected in series. Each device is associated with some cost, i.e., C1, C2, C3, C4. These devices are having some reliability, i.e., R1, R2, R3, R4. The reliability of device may vary. Suppose the reliability of each device is 0.9. The reliability of whole system will be the product of all the reliabilities.

D1	D2	D3	D4
C1	C2	C3	C4
r1	r2	r3	r4
0.9	0.9	0.9	0.9

Total reliability of the system = $(0.9)^4 = 0.656$. There are 35% chances that system may fail. We need to improve our system such that the reliability is maximum. We need to have more than one copy of devices so that the whole system can work efficiently. For increasing the reliability, we need to connect the devices in parallel.



Reliability of stage 1

- $R1 = 0.9, 1 - R1 = 1 - 0.9 = 0.1, (1 - R1)^3 = (0.1)^3 = 0.001, 1 - (1 - R1)^3 = 0.999$

In cost C, we have to set up a system after buying these devices and connect them. The question here is, that how many copies of each device we should buy such that the reliability of the system must be maximized.



Example

Given a system which requires 3 devices, i.e., D1, D2 and D3. The cost associated with the devices are 30, 15 and 20 respectively. The reliability of the devices are also given, i.e, 0.9, 0.8 and 0.5

Unit 05: Dynamic Programming

respectively. The total cost given is 105. How many copies of each device should be taken so that the reliability is maximized in this cost.

Di	Ci	Ri	Ui
D1	30	0.9	
D2	15	0.8	
D3	20	0.5	

Here we are trying to find the upper bound: we are trying to setup a system by taking multiple copies of the devices. Minimum one copy is required of each device, so the cost is $30 + 15 + 20 = 65$. Remaining amount = $105 - 65 = 40$.

The formula for calculation of upper bound will be:

- For 1st device=Flr (C - $\sum Ci / Ci$) + 1 = $40/30 + 1 = 1 + 1 = 2$
- For 2nd device=Flr(C - $\sum Ci / Ci$) + 1 = $40/15+1 = 2 + 1 = 3$
- For 3rd device=Flr(C - $\sum Ci / Ci$) + 1 = $40/20 + 1 = 2+1 = 3$

Di	Ci	Ri	Ui
D1	30	0.9	2
D2	15	0.8	3
D3	20	0.5	3

The objective here is to maximize the reliability.

For 1st device:

(R, C); S0 = {(1,0)}; Consider D1 = S1 = {(0.9, 30)} (if 1 copy is taken)

We can take two copies of it, $1-(1-r1)^2 = 1-(1-0.9)^2 = 1-(0.1)^2 = 1-0.01 = 0.99$ S2 = {(0.99, 60)}

So, for set1, the possibilities are {(0.9,30),(0.99,60)}.

For 2nd device:

S2 = {(0.72, 45),(0.792,75)} (For 1 copy)

(For 2nd copy) Reliability = $1-(1-r2)^2 = 1- (1-0.8)^2 = 1-0.04 = 0.96$

S3 = {(0.864, 60), (0.9504, 90)}

If I take two copies of 2nd device, then the cost of that is 90. if 90 are spent on 2nd device, the remaining cost is 15. the cost for 3rd device is 20. So, I cant buy third device, so the ordered pair is infeasible. S2 = {(0.864, 60)}

For 3rd copy, $1-(1-r2)^3 = 1- (1-0.8)^3 = 1-0.008 = 0.992$

S3 = {(0.8928, 75)}

So, for set2, the possibilities {(0.72,45), (0.792,75), (0.864,60), (0.8928,75)}. Check the reliability and associated costs: Apply Dominance rule, Cut the first ordered pair with the highest cost. So, for set2, the possibilities {(0.72,45), (0.864,60), (0.8928,75)}.

For 3rd device:

S3 = {(0.36,65), (0.432,80), (0.4464,95)} (for 1 copy)

$$R3 = 0.5$$

$$1 - (1 - r3)^2$$

$$1 - (0.5)^2 = 0.75$$

$$S^3_2 = \{(0.54, 85), (0.648, 100), (\quad, 115)\}$$

$$\text{For } S^3_3 = 1 - (1 - r3)^3 = 0.875$$

$$S^3_3 = \{(0.63, 105)\}$$

For set 3, the possibilities are : $\{(0.36, 65), (0.432, 80), (0.4464, 95), (0.648, 100)\}$. The maximum reliability which we are getting is 100.

$(0.648, 100).D3 = 2, D2 = 2, D1 = 1$. Reliability = 0.648. Cost = 100

Summary

- Dynamic programming is used to solve all-pair shortest path problem, reliability design and finding of optimal binary tree problem.
- The time taken by all-pair shortest path problem is $\Theta(n^3)/$
- The conclusion on this is if the height of binary search tree is less than the number of comparisons will be less.
- The cost is dependent upon the height of BST.
- DP tries you a faster method for trying all possible BST and picking up the best one without trying all of them.
- Bellman Ford algorithm follows dynamic programming approach. In DP, try out all possible solutions and pick up the best one.

Keywords:

- **Binary Search Tree** is a node-based binary tree data structure which has the following properties: the left subtree of a node contains only nodes with keys lesser than the node's key, the right subtree of a node contains only nodes with keys greater than the node's key and the left and right subtree each must also be a binary search tree.
- **Single source shortest path problem:** Given a weighted directed graph, we have to start from the starting vertex and find out the shortest path to all the other vertices from that vertex.
- **Bellman Ford algorithm:** The concept behind the Bellman Ford algorithm is go on repeatedly relaxing all the edges for $(N-1)$ times. N is the number of vertices.
- **Reliability design:** Reliability design or design of reliability is a process that ensures a product, or system, performs a specified function within a given environment over the expected lifetime.
- **Reliability assurance:** Advanced circuitry, sophisticated power requirements, new components, new material technologies and less robust parts make ensuring reliability increasingly difficult.

Self Assessment

1. In reliability design, what is considered for the devices to be used in setup?
 - A. Lower bound
 - B. Upper bound
 - C. Average bound
 - D. None of the above

Unit 05: Dynamic Programming

2. For calculation of upper bound, which function is used?
 - A. Floor
 - B. Ceil
 - C. Sqrt
 - D. None of the above

3. The design of reliability is done to
 - A. Reduce the cost
 - B. Assurance
 - C. Product Differentiation
 - D. All of the above

4. In Bellman Ford algorithm, how many times, the edges are relaxed, if the number of vertices is 7?
 - A. 5
 - B. 6
 - C. 7
 - D. 8

5. In Bellman Ford algorithm, we always look for _____ value.
 - A. Minimum
 - B. Maximum
 - C. Equal
 - D. None of the above

6. In Bellman Ford algorithm, what is the time complexity of a complete graph?
 - A. $O(n)$
 - B. $O(n^2)$
 - C. $O(n^3)$
 - D. $O(n^4)$

7. Out of these techniques, which one provides the fastest method for generating optimal binary search tree?
 - A. Greedy method
 - B. Dynamic programming
 - C. Branch and bound
 - D. Divide and Conquer

8. In binary search tree, the left subtree always contains the elements ____ the root node.
 - A. Lesser than
 - B. Greater than
 - C. Equal to
 - D. None of the above

Algorithm Design and Analysis

9. If there are 3 key elements, then the number of possible binary search trees will be
 - A. 3
 - B. 4
 - C. 5
 - D. 6

10. In all pair shortest path problem, we try to find out
 - A. The shortest path from one vertex to all other vertices
 - B. The shortest path between every pair of vertices
 - C. The longest path from one vertex to all other vertices
 - D. The longest path between every pair of vertices

11. While modifying the values, the value chosen should be
 - A. Minimum
 - B. Maximum
 - C. Equal
 - D. None of the above

12. The Dijkstra algorithm is used to find
 - A. The shortest path from one vertex to all other vertices
 - B. The shortest path between every pair of vertices
 - C. The longest path from one vertex to all other vertices
 - D. The longest path between every pair of vertices

13. The reliability of whole system is the _____ of all the reliabilities.
 - A. Addition
 - B. Subtraction
 - C. Multiplication
 - D. Division

14. In binary search tree, the right subtree always contains the elements _____ the root node.
 - A. Lesser than
 - B. Greater than
 - C. Equal to
 - D. None of the above.

15. Which technique is used for solving all pair shortest path problem?
 - A. Greedy method
 - B. Dynamic programming
 - C. Divide-and-conquer strategy
 - D. None of the above

Answers for Self Assessment

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

Review Questions

1. What is all-pair shortest path problem? Is it advisable to use Dijkstra's algorithm for this? If no, then why?
2. Explain the process of solving all-pair shortest path problem with the help of an example. Write its time complexity.
3. What is a binary search tree? How can we find out the number of possible binary search trees?
4. Explain the time complexity and failure point of Bellman Ford algorithm.
5. What is reliability design? When is it made? Is it critical?
6. Given a system which requires 3 devices, i.e., D1, D2 and D3. The cost associated with the devices are 30, 15 and 20 respectively. The reliability of the devices are also given, i.e, 0.9, 0.8 and 0.5 respectively. The total cost given is 105. How many copies of each device should be taken so that the reliability is maximized in this cost.

**Further Readings**

<https://www.reliasoft.com/resources/resource-center/design-for-reliability-overview-of-the-process-and-applicable-techniques>

<https://www.tutorialspoint.com/all-pairs-shortest-paths#:~:text=The%20all%20pair%20shortest%20path,other%20nodes%20in%20the%20graph.>

<https://www.geeksforgeeks.org/optimal-binary-search-tree-dp-24/>

Unit 06: Backtracking

CONTENTS

- Objectives
- Introduction
- 6.1 Sum of Subsets Problem
- 6.2 N Queens problem
- 6.3 Graph Coloring Problem
- 6.4 Hamiltonian Cycle Problem
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the backtracking algorithm
- Understand the N Queens Problem
- Understand the graph colouring problem
- Understand the Hamiltonian cycle problem

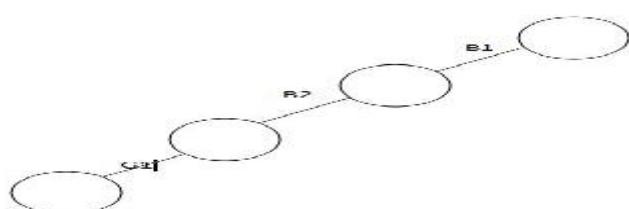
Introduction

The backtracking algorithm follows the Brute force approach. Brute Force approach says that for any given problem you should try all possible solutions and pick up the desired one. This is not for optimization problems. Backtracking is used when you have multiple solutions, and you want all of them. Given one example we have three students, 2 boys and 1 girl and we have three chairs for them. The students are represented as: B1 B2 G1. The chairs are represented as:

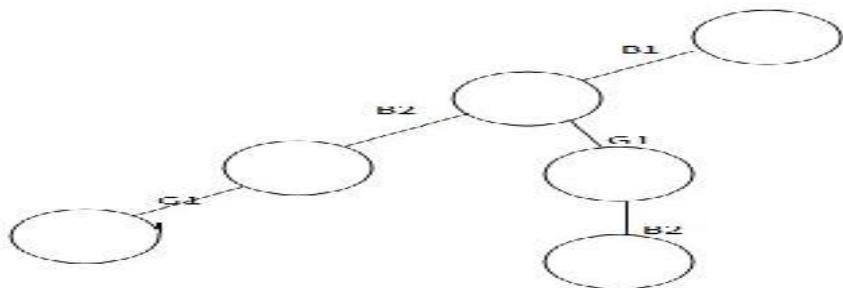


So, for this there are $3! = 6$ Ways by which we can arrange them. The arrangements can be found, and we can see them in the form of a tree which we call as a solution tree or simply the state space tree. Below this, 6 arrangements are shown in the form of solution 1 to solution 6.

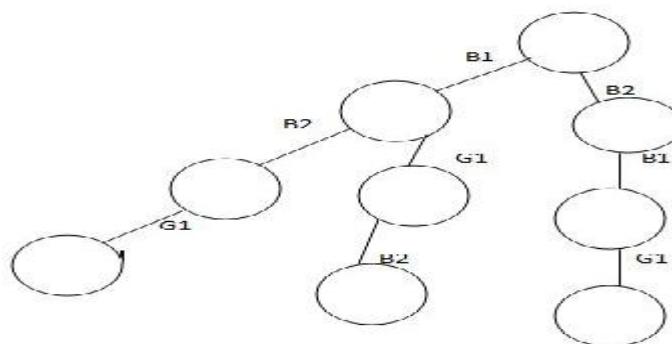
Solution 1



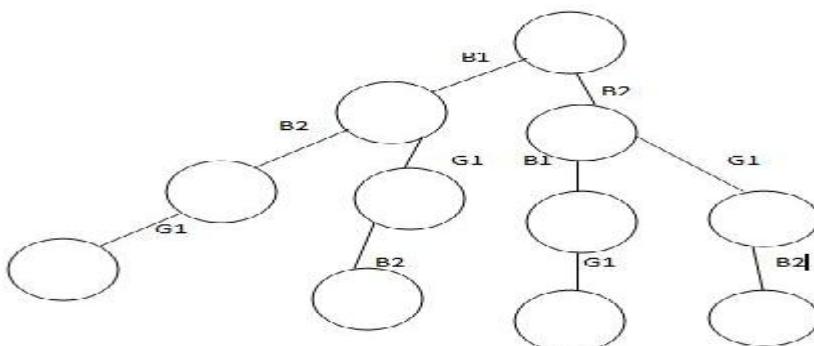
B1	B2	G1
----	----	----

Solution 2

B1	G1	B2
----	----	----

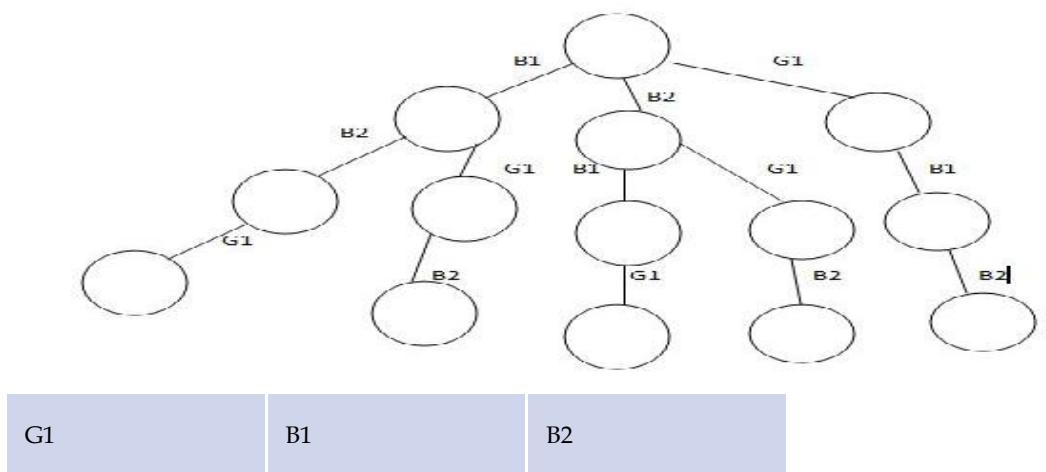
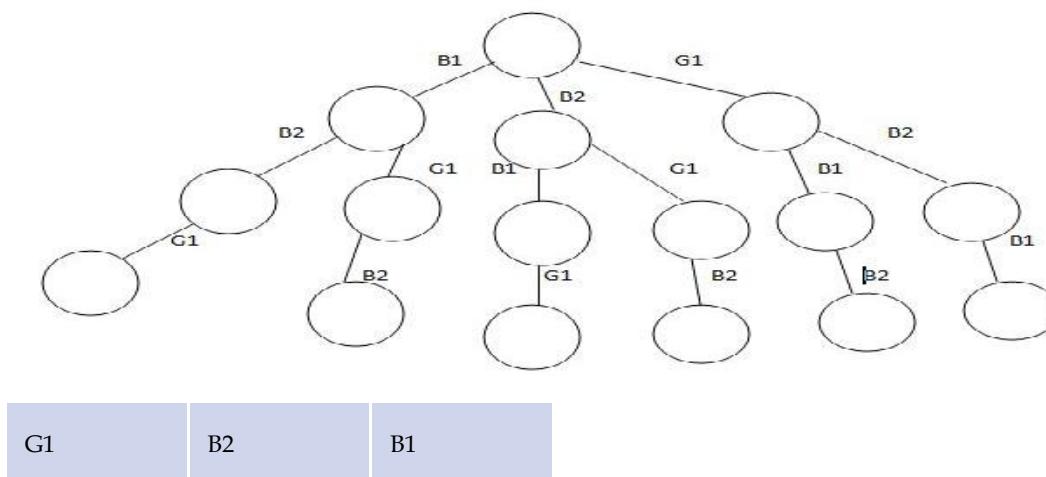
Solution 3

B2	B1	G1
----	----	----

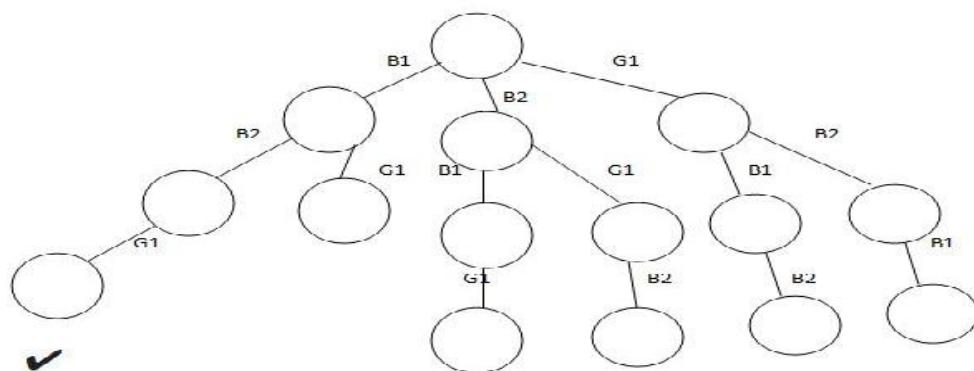
Solution 4

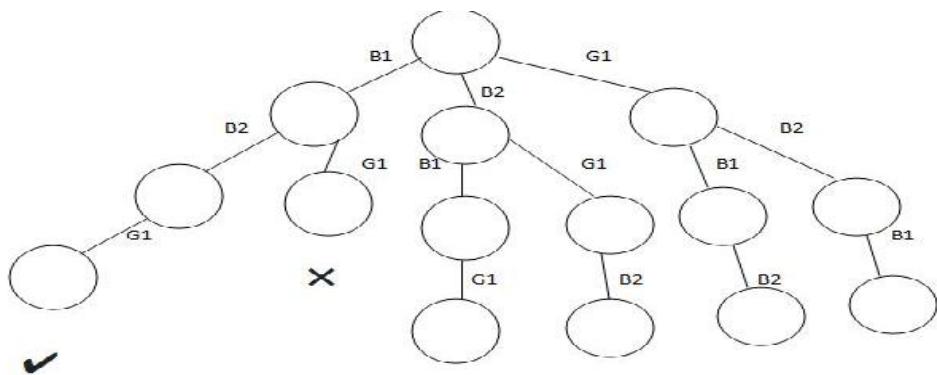
B2	G1	B2
----	----	----

Solution 5

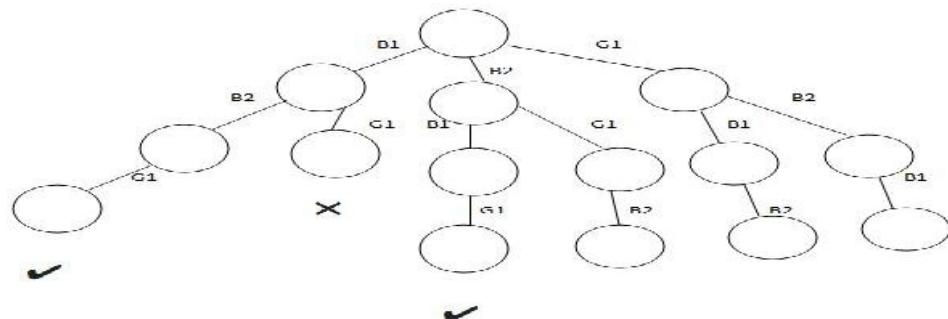
**Solution 6**

The different arrangements are shown as shown in the form of solution 1 to solution 6. This is an example of just 3 girls and 3 chairs. For this we have a total of 6 arrangements. What if we have 10,000 students? Usually when we study backtracking algorithm on any problem, it has some constraint associated with that. So we will consider only those solutions which satisfies those constraints. The constraint here is girl should not sit in middle. So, there are various possible arrangements which are shown in the form of state space tree.

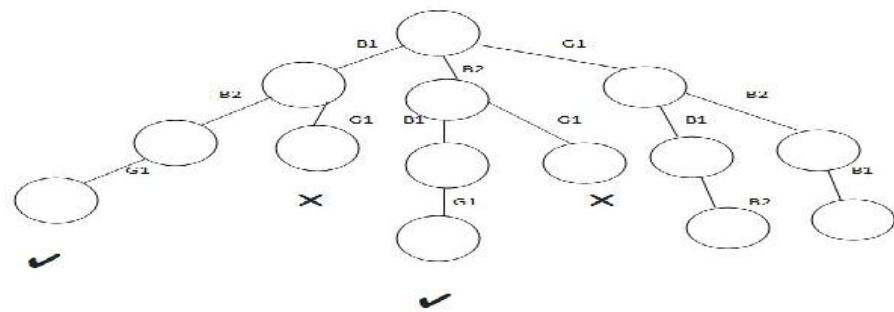
Solution 1**Solution 2**



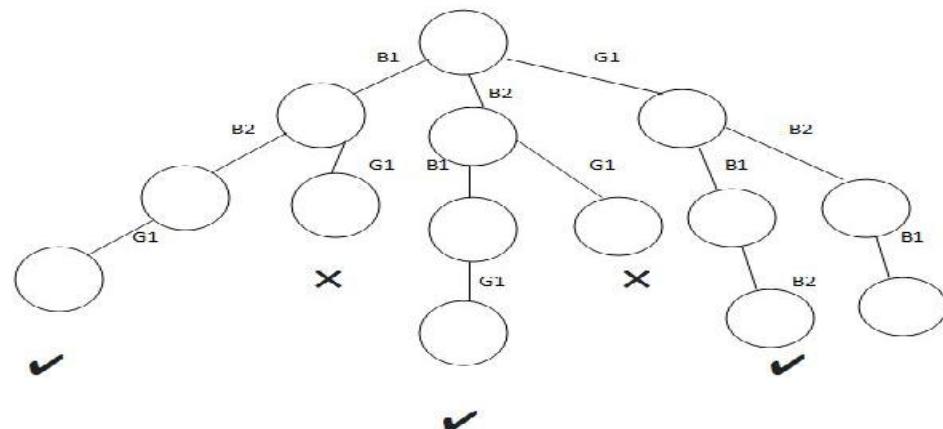
Solution 3



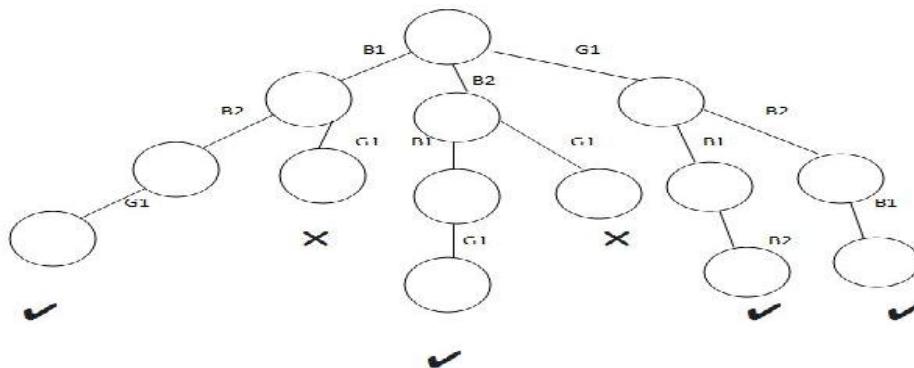
Solution 4



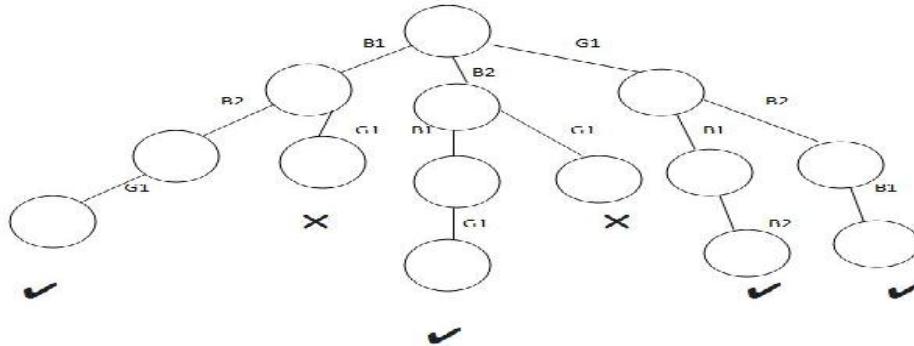
Solution 5



Solution 6



The bounding function here is the node where we have the girl in the middle, that is killed. That node is killed because we are imposing a condition which is called as Bounding function. So if we look carefully, the backtracking algorithm follows DFS.



Algorithm

Step 1 – if current_position is goal, return success

Step 2 – else,

Step 3 – if current_position is an end point, return failed.

Step 4 - else, if current_position is not end point, explore and repeat above steps.

Various Problems

Apart from the arrangement problem, there are various problems which can be solved using backtracking. These are:

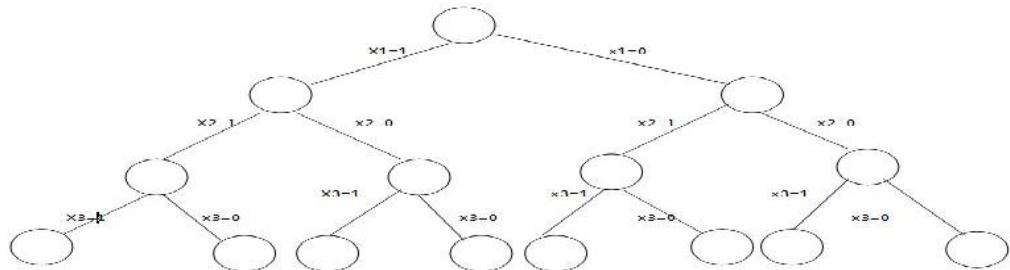
- Sum of Subsets problem
- N Queens Problem
- Graph Colouring Problem
- Hamiltonian Cycle

6.1 Sum of Subsets Problem

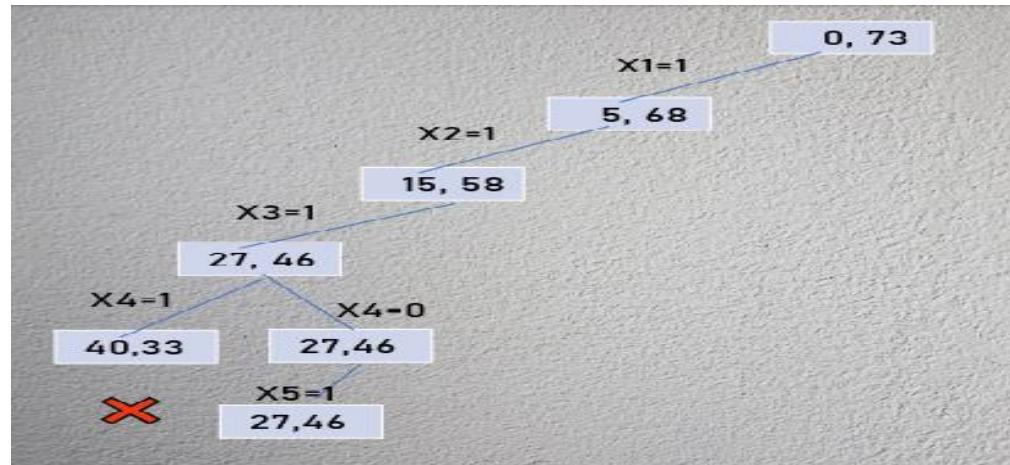
Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number N. We are considering the set contains non-negative values. It is assumed that the input set is unique (no duplicates are presented). So here N is given as 6 and the set is given as 5, 10, 12, 13, 15, 18. So, out of this set we have to select a subset of numbers whose sum is equal to 30.

- $W[1,6] = \{5, 10, 12, 13, 15, 18\}; N=6, m=30$.

If we create a state space tree out of these, it will look like



So, the time consumed is related to total paths, i.e., $2^n = 2^{10}$. So when we follow the constraint we need to backtrack here and find the solution. The state space tree will look like this.



So, the conclusion is backtracking is a general algorithm for finding all solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons a candidate as soon as it determines that the candidate cannot possibly be completed to a valid solution. So, there are three keys in backtracking:

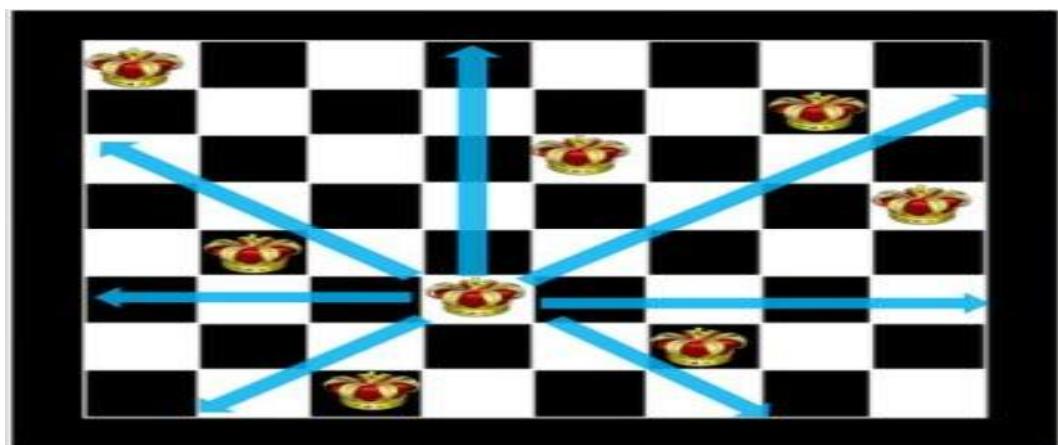
- 1) Choice
- 2) Constraint
- 3) Goal

6.2 N Queens problem

In N Queens problem, the problem is to place n queens on an $n \times n$ chessboard, so that no two queens are attacking each other.



If we have 4×4 chessboard, then the total number of arrangements will be $16C_4 = 1820$. Given an example of 4×4 queens chessboard, the 4 queens are i.e, Q1, Q2, Q3 and Q4. The constraint here is no two queens should lie in the same Row: Q1 can come in row 1, Q2 can come in row 2, Q3 can come in row 3 and Q4 can come in row 4. No two queens should lie in same column. Apart from this no two queens should lie in same diagonal. The clear picture is shown in below diagram. It states that queens should be placed such as no two queens should intersect in row, column and diagonal.

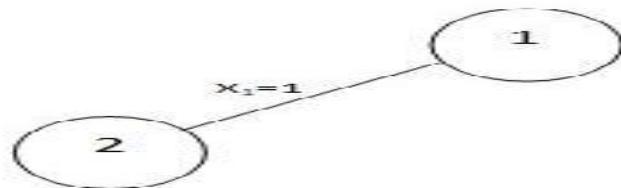


This problem of $N \times N$ queens can be solved by using backtracking. A backtracking algorithm tries to construct a solution to a computational problem incrementally, one small piece at a time. Whenever the algorithm needs to decide between multiple alternatives to the next component of the solution, it recursively evaluates every alternative and then chooses the best one.

4 Queens problem

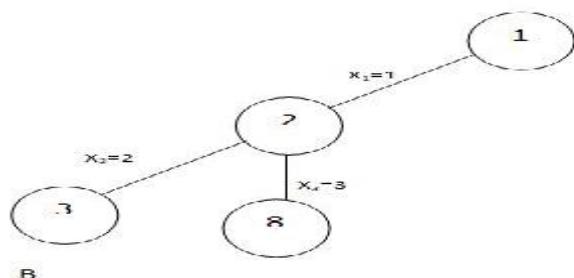
In this Q1 is placed in place [1,1].

1			



As already Q1 is place in [1,1]. So Q2 can't be placed in same row, same column, and same diagonal. So, it will be placed in [2,3]. Here B represents the bounding function.

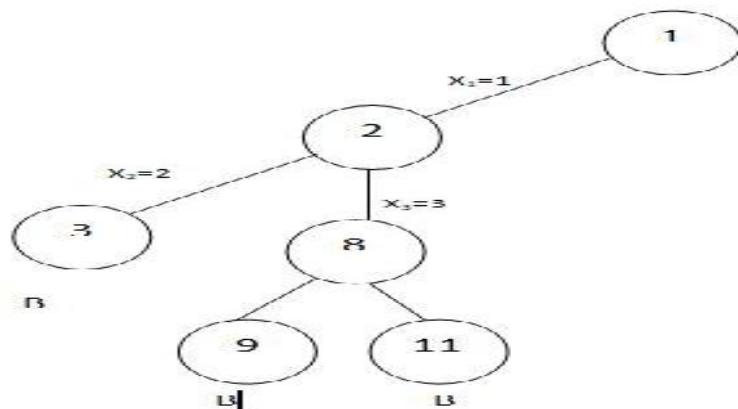
1			
		2	



Algorithm Design and Analysis

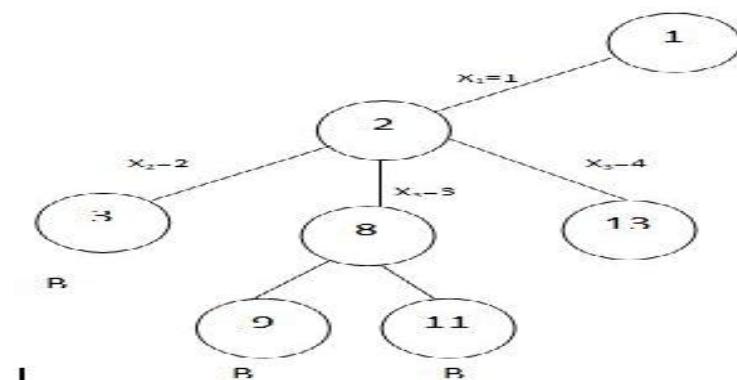
If we place Q2 in [2,3], there is no way we can place Q3 and Q4. So we have to move it.

1			
		2	



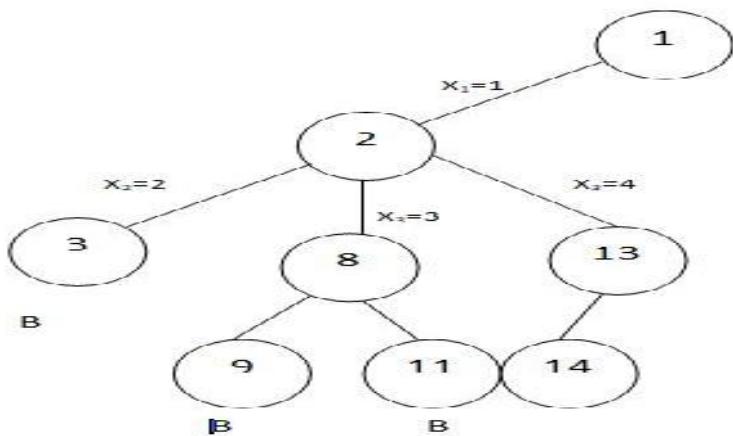
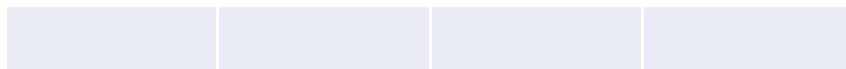
Q2 can be placed in [2,4].

1			
			2



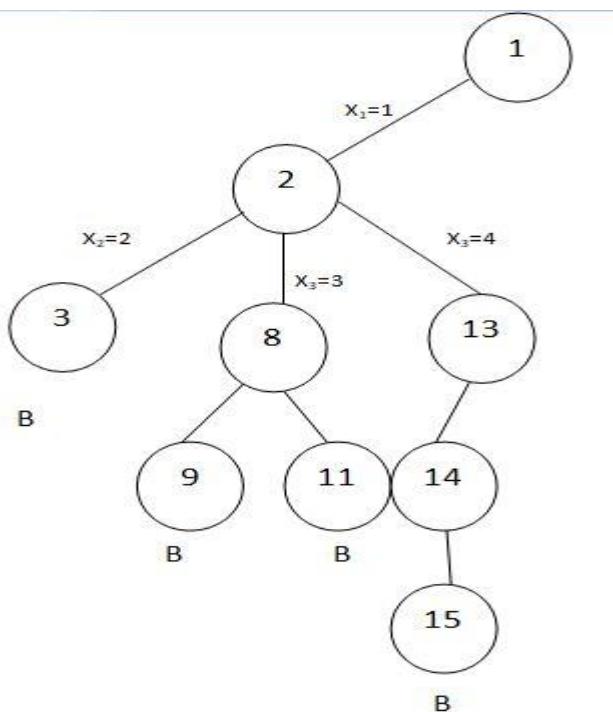
After this Q3 can be placed in [3,2].

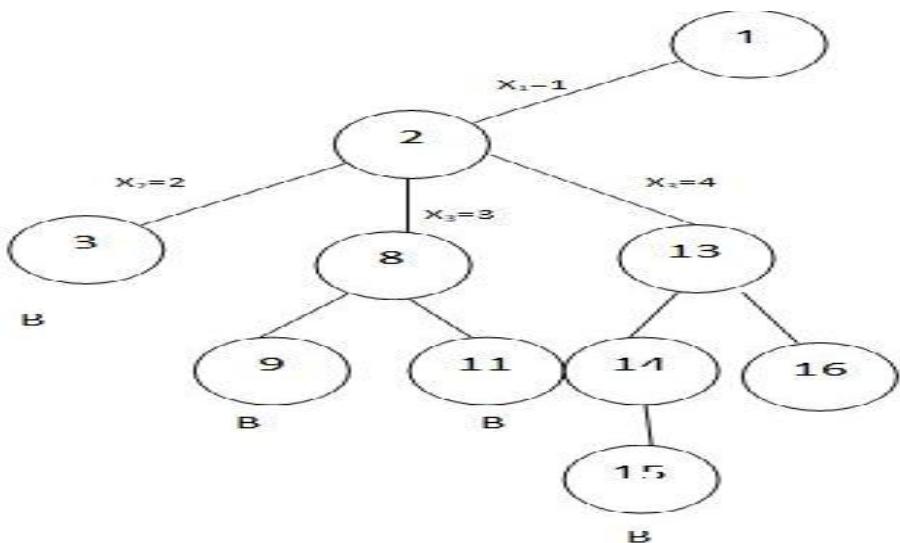
1			
			2
	3		



Now after placing Q1, Q2 and Q3, there is no way we can place Q4. So, the arrangement is not correct.

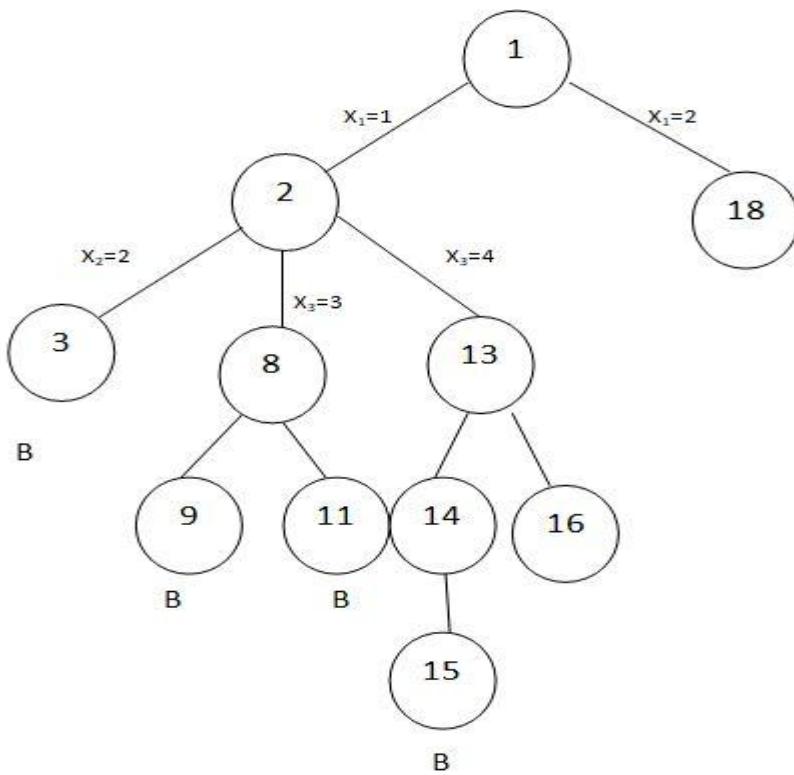
1			
			2
	3		
-	-	-	-





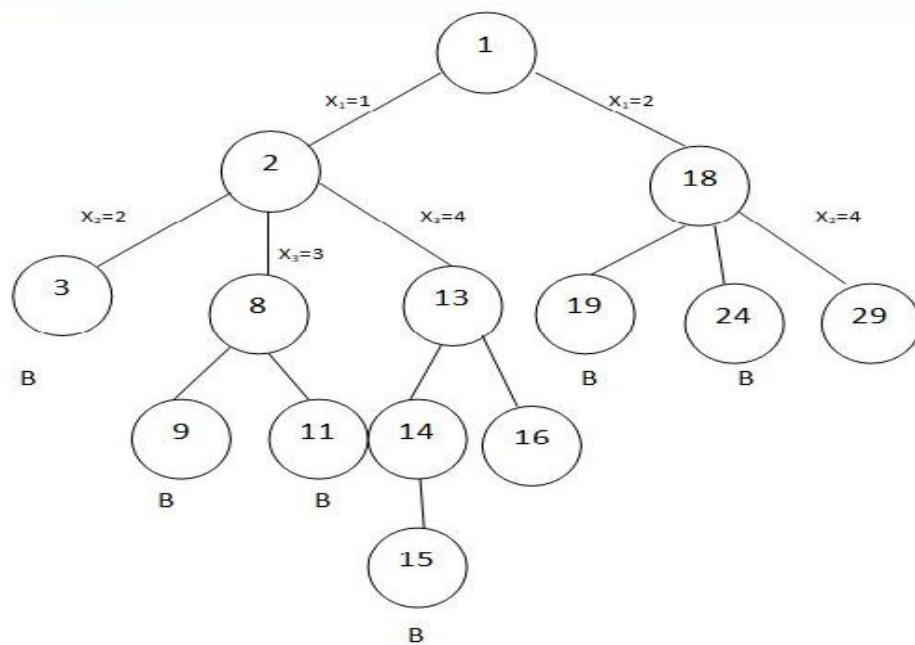
So we need to move Q1 to [1,2].

	1		



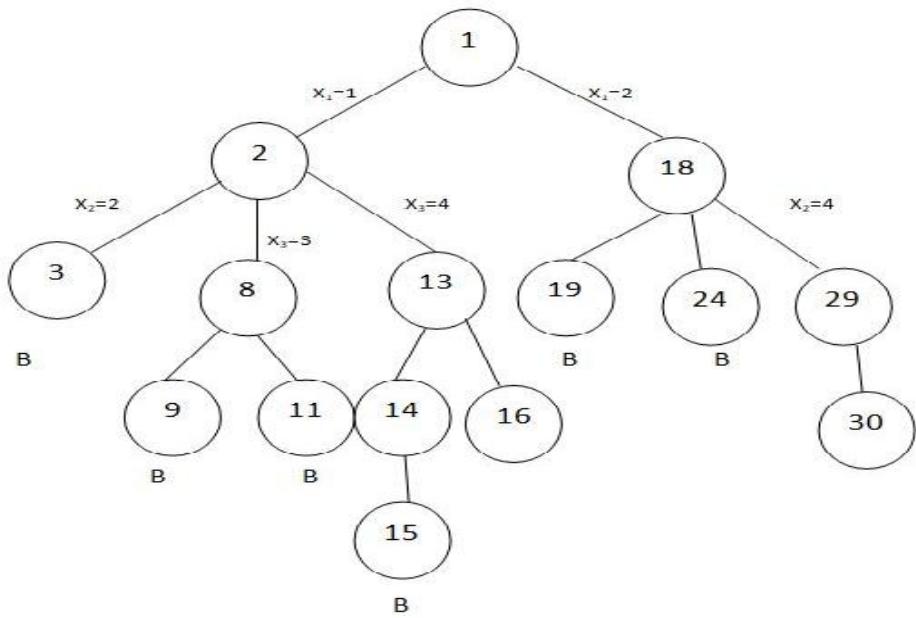
After placing Q1 in [1,2], we can place Q2 in [2,4].

	1		
			2



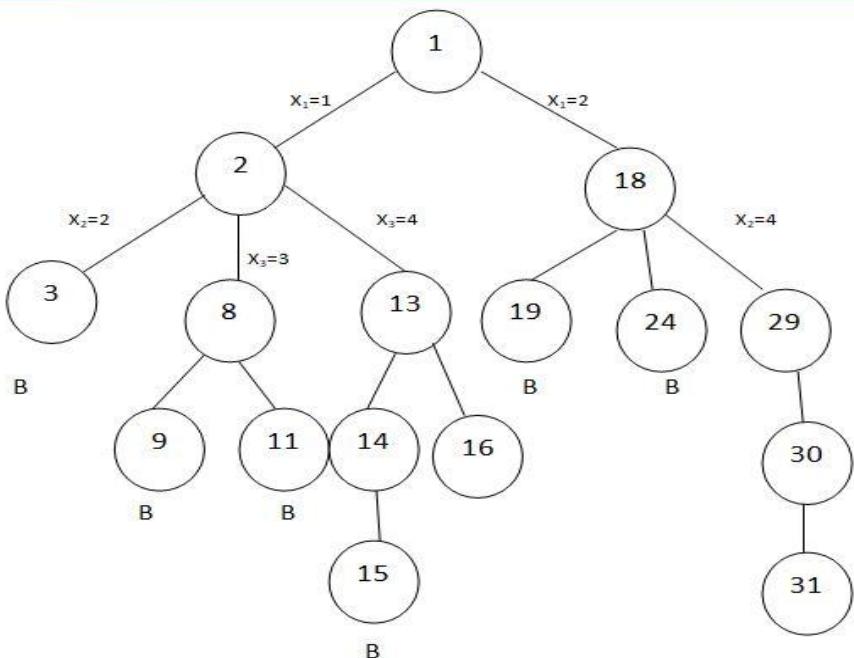
After this Q3 can be placed in [3,1].

	1		
			2
3			

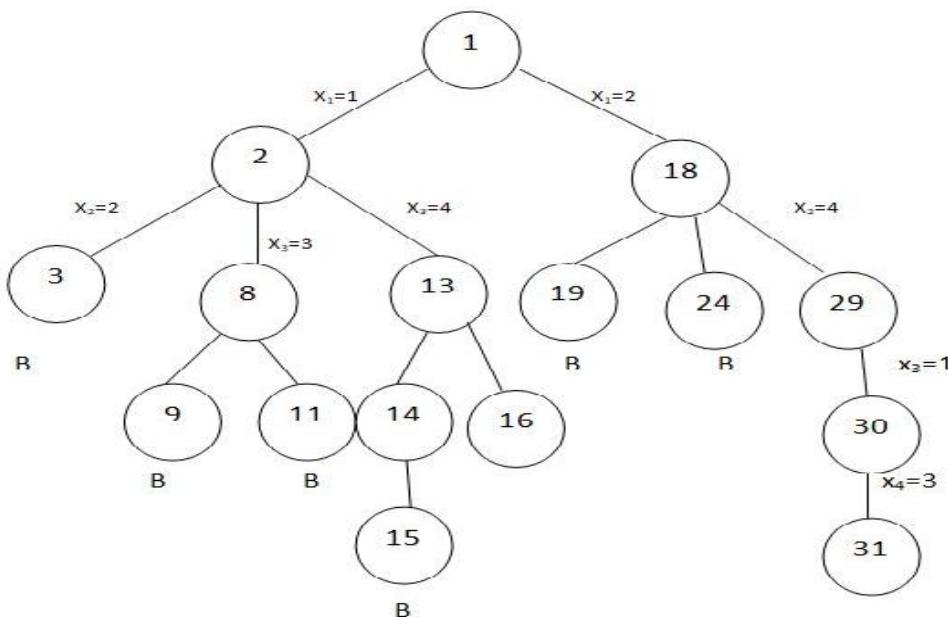


Q4 can be placed in [4,3]. As we can see the one of the possible arrangements of four queens is Q1[1,2], Q2[2,4], Q3[3,1] and Q4[4,3].

	1		
			2
3			
		4	



So, the state space tree for 4 queens problem is



Another Solution for 4 queens problem is

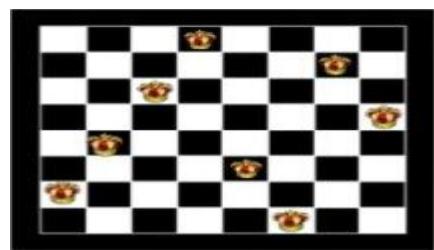
		1	
2			
		3	
	4		

Algorithm

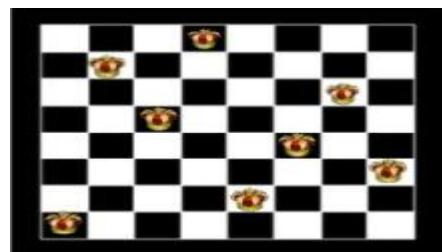
- 1) Start in the leftmost column
- 2) If all queens are placed return true
- 3) Try all rows in the current column. Do following for every tried row.
 - a) If the queen can be placed safely in this row, then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
 - b) If placing the queen in [row, column] leads to a solution then return true.
 - c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
- 4) If all rows have been tried and nothing worked, return false to trigger backtracking.

For 8 queens' problem, we have 12 unique solutions. Few of the possible unique solutions are:

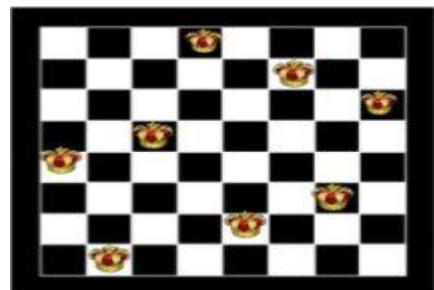
Solution 1



Solution 2

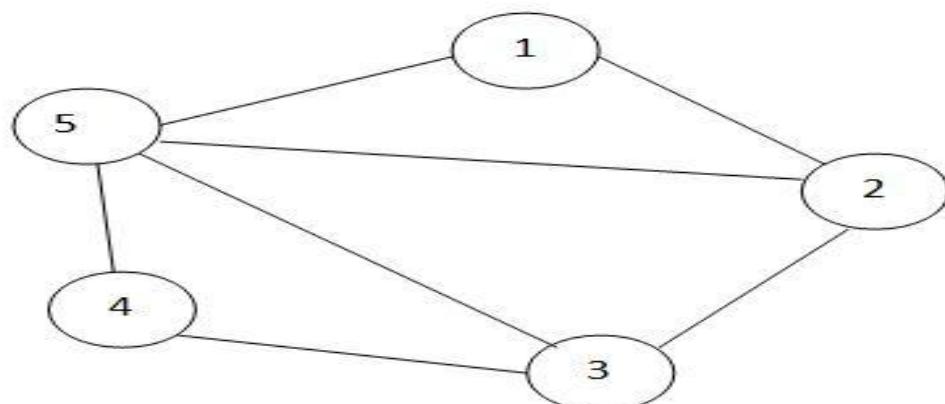


Solution 3

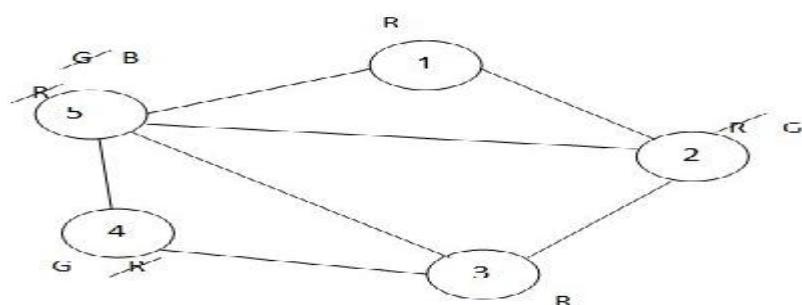
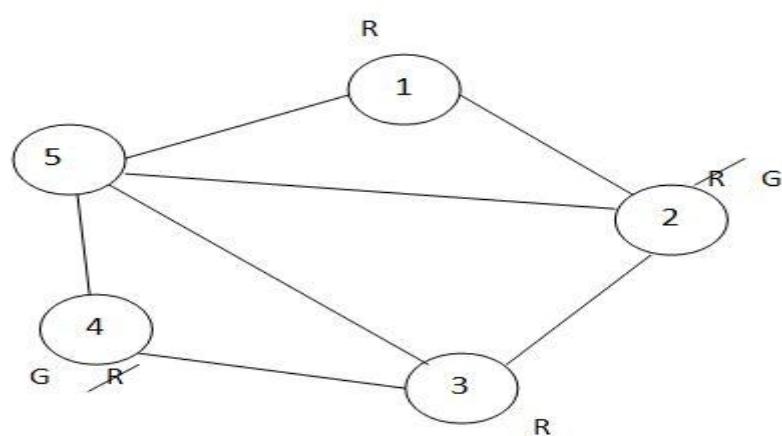
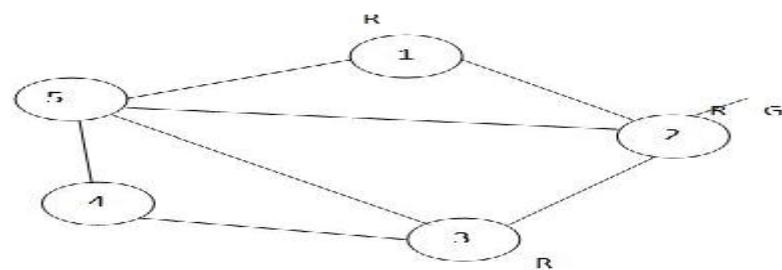
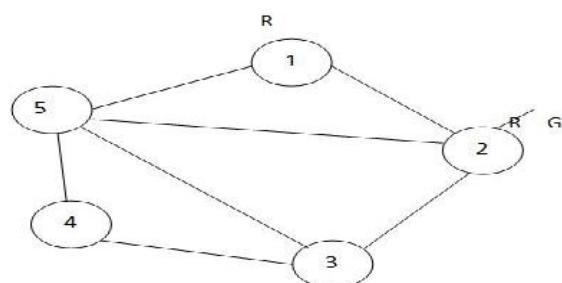
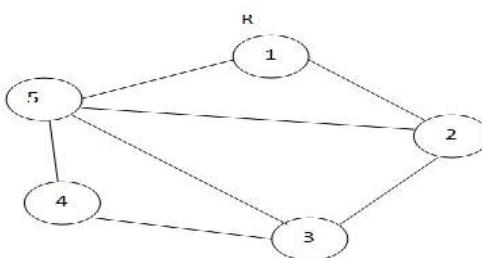


6.3 Graph Coloring Problem

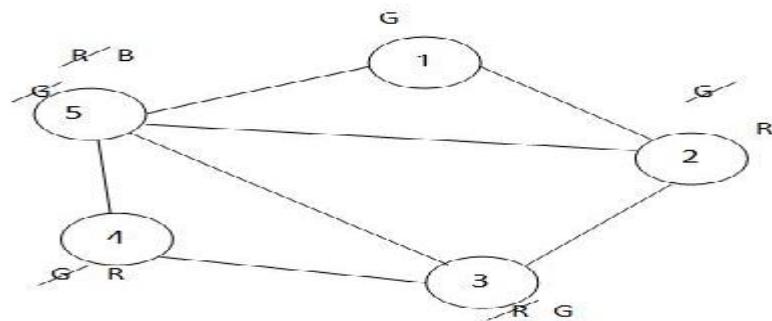
Graph coloring is the procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same color. The smallest number of colors required to color a graph G is called its **chromatic number** of that graph. The constraint here is no adjacent/neighboring vertices get same color. Given a graph in which we have 5 vertices and the total number of colors given is 3 which are R, G and B.



Solution 1



Solution 2

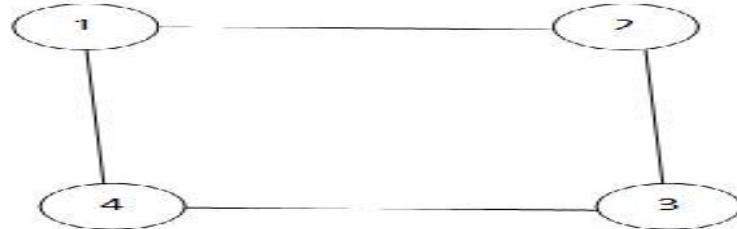


In this we can have two types of problems:

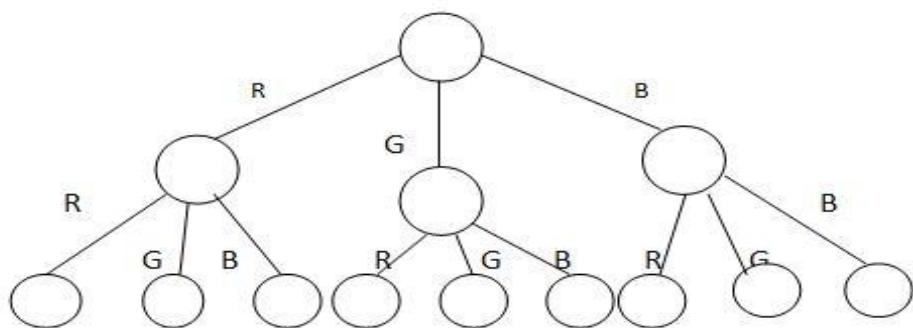
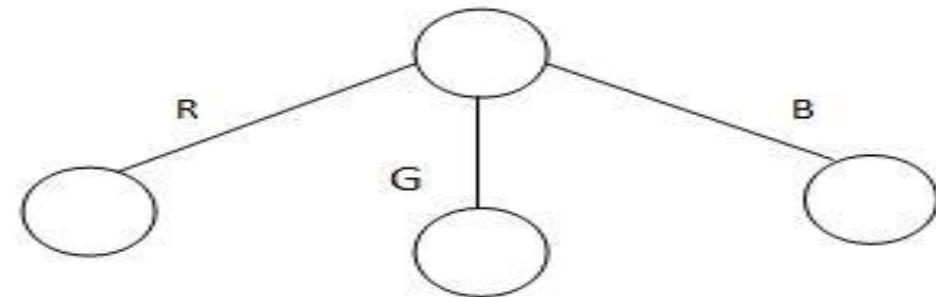
- 1) mColouring Decision Problem: Given a graph and a set of colours, from that we can identify whether with a given set of colours we can colour the graph or not.
- 2) mColouring Optimization Problem: Given a graph, we want to know the minimum number of colours required to colour the graph.

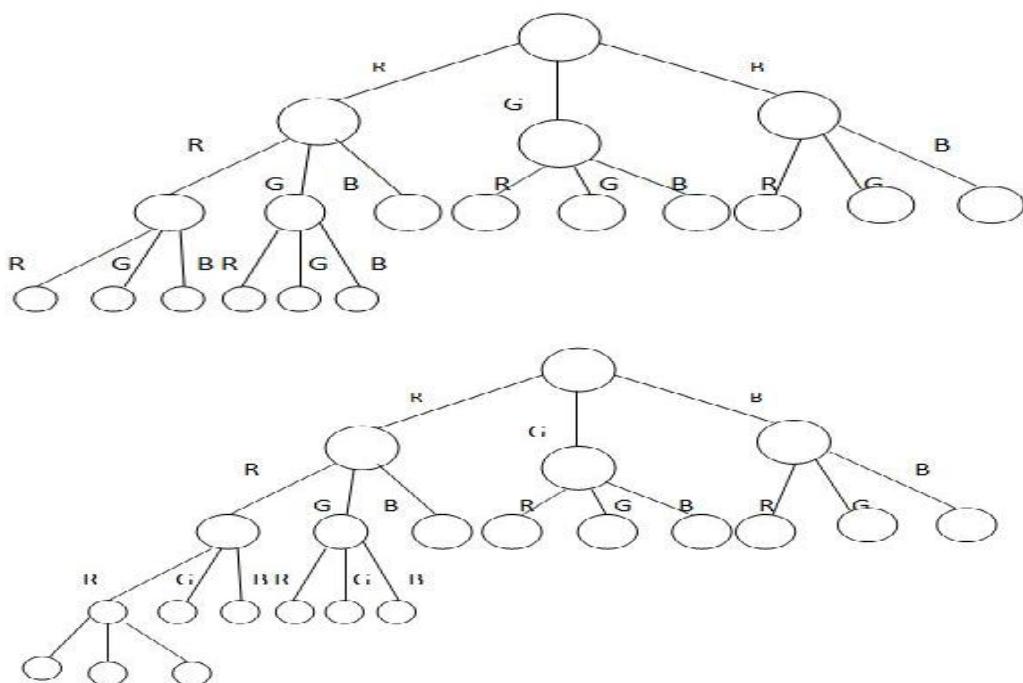


Example

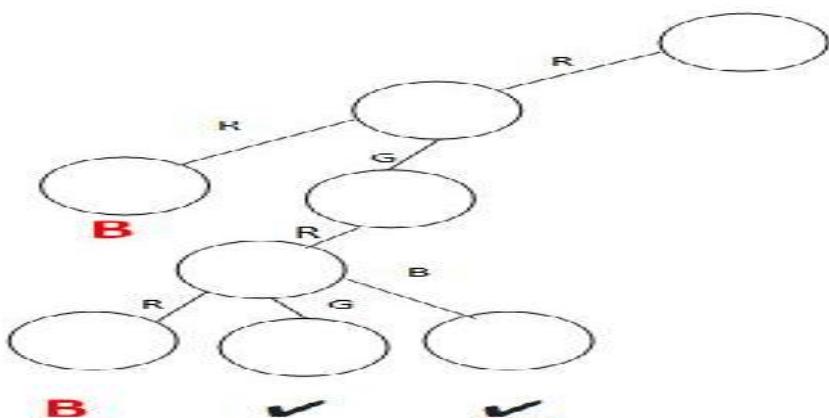


$M = 3$ and the colors are: {R, G, B}. This can be colored if there is not any constraint.





So, the total number of nodes = $1 + 3 + 3*3 + 3*3*3 + 3*3*3*3 = 1 + 3 + 3^2 + 3^3 + 3^4 = (3^{4+1} - 1) / (3-1) = (3^{4+1} - 1) / 2$. If we apply the constraint that no two adjacent vertices should get the same color, then it can be painted as: B in red letters is for the bounding function.



So, if we follow this constraint then some of the possible solutions are:

- {R, G, R, G}; {R, G, R, B}; {R, G, B, G}; {R, B, R, G}.

Algorithm

1. Different colors:

- Confirm whether it is valid to color the current vertex with the current color (by checking whether any of its adjacent vertices are colored with the same color).
 - If yes then color it and otherwise try a different color.
 - Check if all vertices are colored or not.
 - If not then move to the next adjacent uncolored vertex.
2. If no other color is available then backtrack (i.e. un-color last colored vertex).

Time complexity

If bounding function is applied, some of the nodes are getting killed. So, it is going to be 3^n . Time complexity = $O(3^n)$.

6.4 Hamiltonian Cycle Problem

Given a graph, we start from some starting vertex and visit all the vertices exactly once and return to the starting vertex, so that it forms a cycle. In this, we check that is there any Hamiltonian cycle possible in a graph. So, the obvious questions that comes in my mind are:

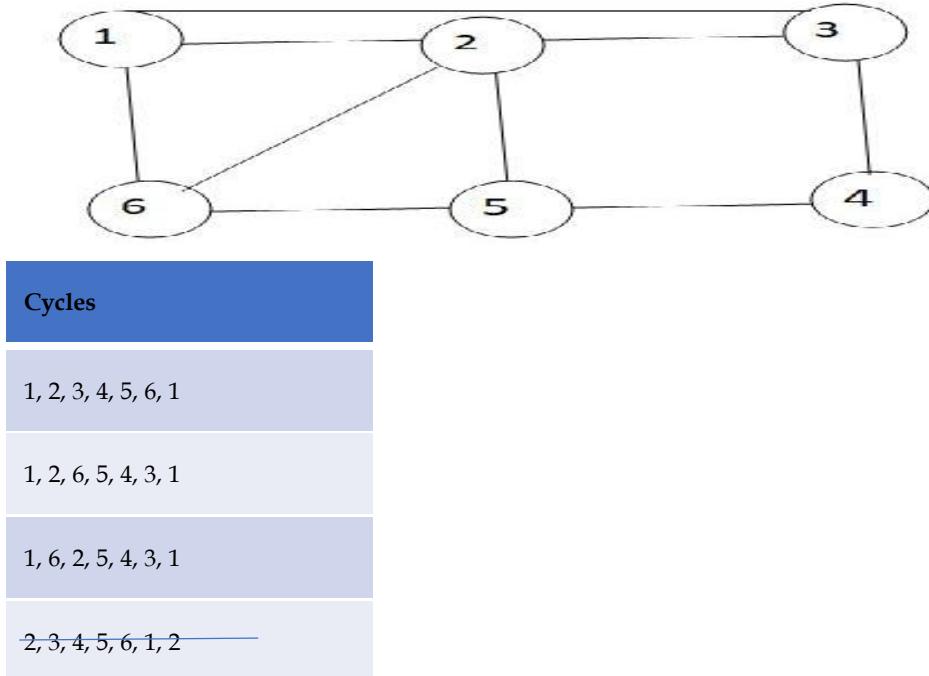
- **Question 1:** We need to check whether there is any cycle or not.
- **Question 2:** How many cycles are there in the graph?

Important points that need to be considered here are:

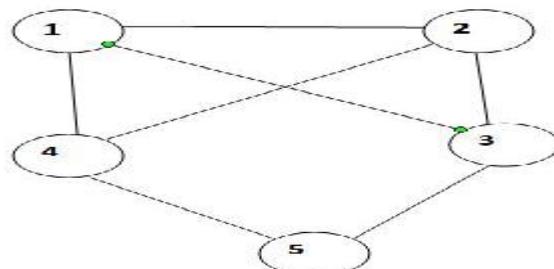
- 1) The graph given may be directed or undirected graph.
- 2) It must be connected so that it can form a cycle.
- 3) The problem is NP hard, it takes exponential time.

Example

Given a graph, we must find out the possible cycles.



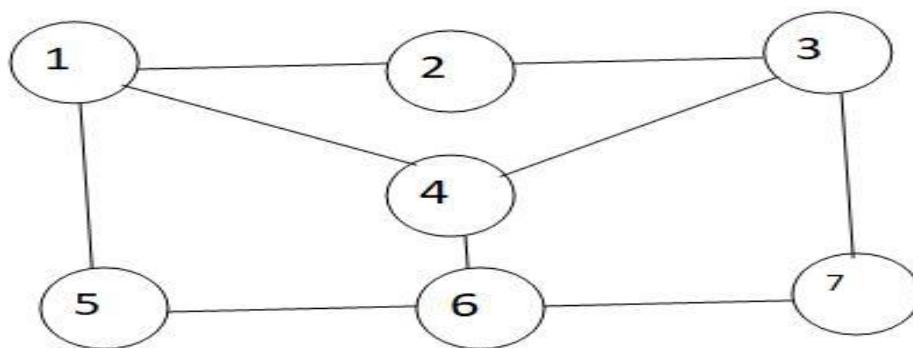
The last one is not a cycle.



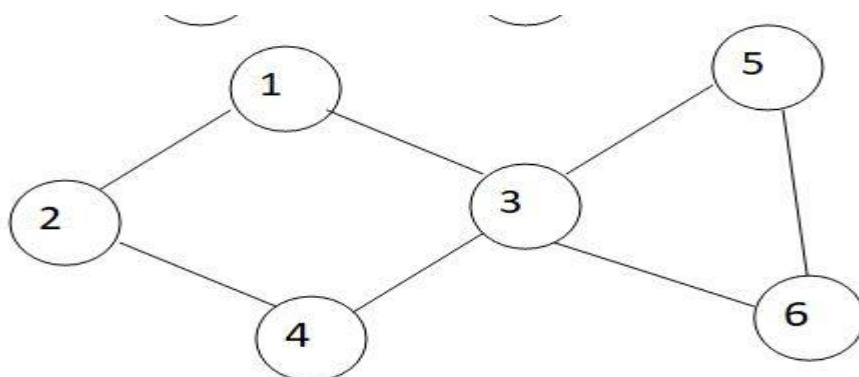
Cycles

1, 2, 3, 5, 4, 1

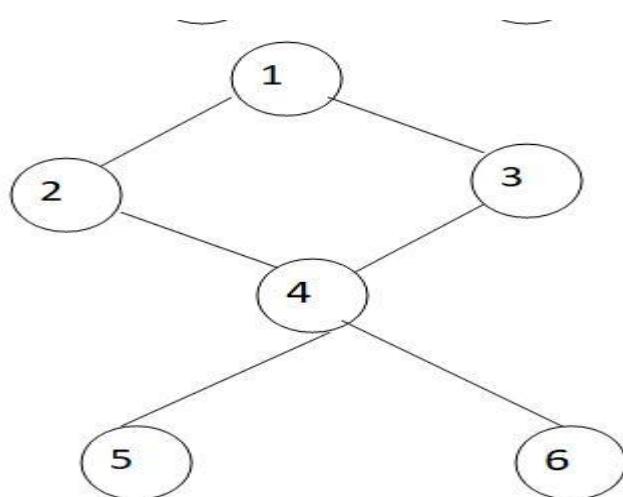
1, 2, 4, 5, 3, 1



Here in the above graph, no hamiltonian cycle is possible



Vertex 3 is the junction for the above graph, and it is known as articulation point.



In the above graph, the Hamiltonian cycle is not possible as vertex 5 and 6 are known as pendant vertices. So, the conclusion is a Hamiltonian cycle is not possible because of pendant vertices and articulation point.

Algorithm

```
Algorithm Hamiltonian(k)
```

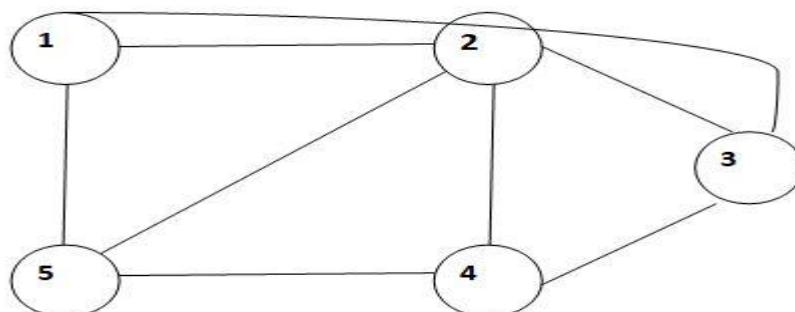
```
{
do
{
NextVertex(k);
if(x[k]==0)
return;
If(k==n)
Print(x,[1:n]);
Else
Hamiltonian (k+1);
} while(true);
}
```

```
Algorithm NextVertex(k)
```

```
{
do
{
X[k] = (x[k]+1)mod (n+1);
If(x[k] == 0) return;
If(G[x[k-1],x[k] ]≠ 0)
{
For j=1 to k-1 do if (x[j] == x[k])
If(j==k)
If(k<n of (k==n) && G[x[n], x[1] ≠ 0
Return;
} while (true);
}
```

Example

Given a graph below, find the Hamiltonian cycle



1	2	3	4	5

If two vertices are directly connected then it is marked as 1, otherwise it is marked as 0.

G	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

0	0	0	0	0
1	2	3	4	5

Starting Vertex is 1

1	1 2	0	0	0
1	2	3	4	5

1	2	0	0	0
1	2	3	4	5

1	2	1 2 3	0	0
1	2	3	4	5

1	2	3	-1 2 3 - 4	0
1	2	3	4	5

1	2	3	4	1 2 3 - 4 5
1	2	3	4	5

1	2	3	4	5
1	2	3	4	5

1	2	3	4	0
1	2	3	4	5

1	2	3	-5 -	0
1	2	3	4	5

1	2	3	0	0
1	2	3	4	5

1	2	4	0	0
1	2	3	4	5

Unit 06: Backtracking

1	2	4	<u>1 2 3</u>	0
1	2	3	4	5

1	2	4	5	0
1	2	3	4	5

1	2	4	5	3
1	2	3	4	5

1	2	5	0	0
1	2	3	4	5

1	2	5	<u>-3 4</u>	0
1	2	3	4	5

1	2	5	4	3
1	2	3	4	5

Answers from vertex 1

- 1, 2, 3, 4, 5, 1 and 1, 2, 5, 4, 3, 1

So, there are various points to consider in bounding function

- 1) It should not take duplicate.
- 2) Whenever you take a vertex, there should be an edge with the previous vertex.
- 3) If you are on the last vertex, then there should be an edge to the first vertex.

Complexity

- $n!$ vertices to explore if we keep 1 starting vertex.
- $n! = O(n^n)$

Summary

- Backtracking follows the brute force approach.
- Backtracking is not implemented on optimization problems.
- Backtracking is used when you have multiple solutions, and you want all of them.
- Usually the backtracking has some constraints, so we will consider only those solutions which satisfies those constraints.
- There are three keys in backtracking: Choice, Constraint and Goal.
- The N Queens problem can be solved by using backtracking.
- Hamiltonian cycle is not possible because of pendant vertices and articulation point.

Keywords

- **Brute Force approach:** It says that for any given problem you should try all possible solutions and pick up the desired one.
- **State space tree:** The arrangements can be seen in the form of a tree which we call as a solution tree or simply the state space tree.
- **Backtracking:** Backtracking is a general algorithm for finding all solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons a candidate as soon as it determines that the candidate cannot possibly be completed to a valid solution.
- **N Queens Problem:** In N Queens problem, the problem is to place n queens on an $n \times n$ chessboard, so that no two queens are attacking each other.
- **Graph coloring:** It is the procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same color.
- **mColouring Decision Problem:** Given a graph and a set of colours, from that we can identify whether with a given set of colours we can colour the graph or not.
- **mColouring Optimization Problem:** Given a graph, we want to know the minimum number of colours required to colour the graph.
- **Hamiltonian Cycle:** Given a graph, we start from some starting vertex and visit all the vertices exactly once and return back to the starting vertex, so that it forms a cycle. In this, we check that is there any Hamiltonian cycle possible in a graph.

Self Assessment

1. Backtracking approach is applied on
 - A. Approximation problems
 - B. Optimization problems
 - C. Constraint satisfaction problems
 - D. None of the above

2. Which of these is used in backtracking approach?
 - A. Binary tree
 - B. Binary search

- C. AVL tree
 - D. State space tree
3. Find the odd one out.
- A. Sum of Subsets
 - B. N Queens
 - C. Hamiltonian Cycle
 - D. Merge sort
4. N Queens problem can be efficiently solved using
- A. Divide and Conquer
 - B. Backtracking
 - C. Dynamic Programming
 - D. Algebraic Manipulation
5. How many unique solutions are there for 8 Queens Problem?
- A. 8
 - B. 4
 - C. 12
 - D. 16
6. In how many directions do queens attack each other?
- A. 1
 - B. 2
 - C. 3
 - D. 4
7. The time complexity of graph coloring problem is
- A. $O(n)$
 - B. $O(n^2)$
 - C. $O(n^3)$
 - D. $O(3^n)$
8. The smallest number of colors required to color a graph is called
- A. Chromatic number
 - B. Positive number
 - C. Natural number
 - D. Real number
9. The graph coloring problem can be efficiently solved using
- A. Backtracking
 - B. Divide and Conquer
 - C. Branch and Bound

D. Dynamic Programming

10. In which case, it is not possible to have Hamiltonian Cycle?

- A. Pendant vertices
- B. Articulation points
- C. Disconnected graph
- D. All of the above

11. The complexity of Hamiltonian Cycle problem is

- A. $O(1)$
- B. $O(n)$
- C. $O(\log n)$
- D. $O(n^n)$

12. The Hamiltonian Cycle problem is

- A. P Type
- B. NP Type
- C. NP Hard
- D. NP Complete

13. The backtracking approach follows

- A. BFS
- B. DFS
- C. Both of the above
- D. None of the above

14. For 4*4 chessboard, the total number of arrangements will be

- A. 2018
- B. 1820
- C. 1620
- D. 2016

15. In the Hamiltonian Cycle problem, the given graph may be

- A. Directed
- B. Undirected
- C. Either of the above
- D. None of the above

Answers for Self Assessment

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

Review Questions

1. What is backtracking? Which approach is followed by backtracking? Give few examples what kind of problems are solved by backtracking approach.
2. We have three students, 2 boys and 1 girl and we have three chairs for them. So, find some good sitting arrangements for them so that the girl should not sit in the middle. Show the state space tree.
3. What is N queens' problem? What is the constraint here?
4. Solve 4 queens problem using backtracking.
5. Write the algorithm for 4 queens' problem.
6. What is graph coloring problem? Explain it using a graph.
7. Write the algorithm for graph coloring problem.
8. What is Hamiltonian cycle problem? In which cases the Hamiltonian cycles are not possible?
9. Write the algorithm for Hamiltonian cycle problem?

**Further Readings**

- <https://www.geeksforgeeks.org/subset-sum-backtracking-4/>
- <https://leetcode.com/problems/n-queens/>
- <https://www.geeksforgeeks.org/graph-coloring-applications/>
- <https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/>

Unit 07: Branch and Bound

CONTENTS

- Objectives
- Introduction
- 7.1 Job Sequencing with Deadlines
- 7.2 0/1 Knapsack Problem
- 7.3 Travelling Salesman Problem
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the branch and bound method
- Understand the job sequencing with deadlines problem
- Understand the 0/1 knapsack problem
- Understand the travelling salesman problem

Introduction

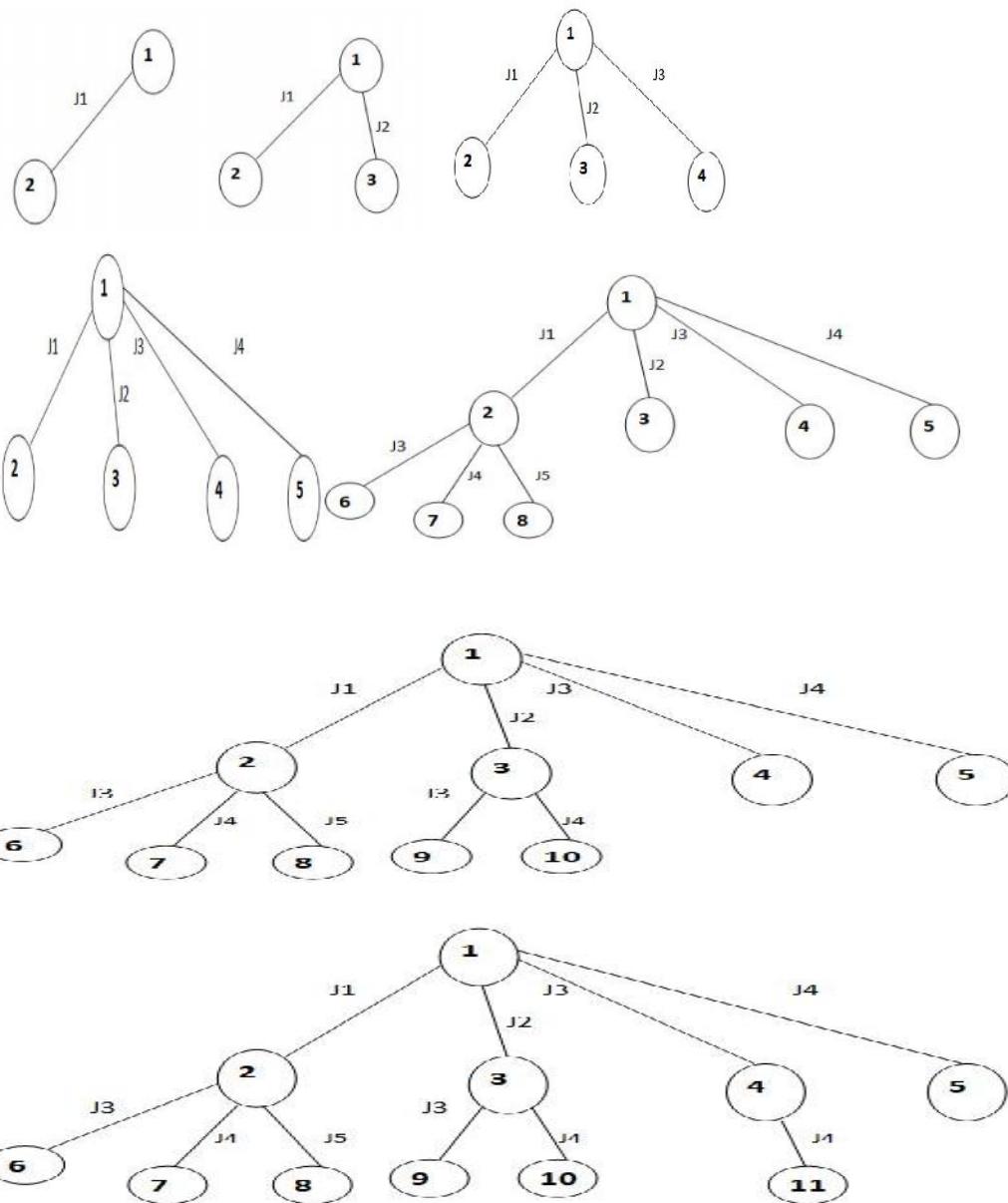
Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The Branch and Bound Algorithm technique solves these problems relatively quickly. It follows Breadth First Search. It is like backtracking because it also uses state space tree for solving the problem. But it is useful for solving optimization problem, only minimization problem, not maximization problem. Branch and bound follows BFS and backtracking follow DFS.

7.1 Job Sequencing with Deadlines

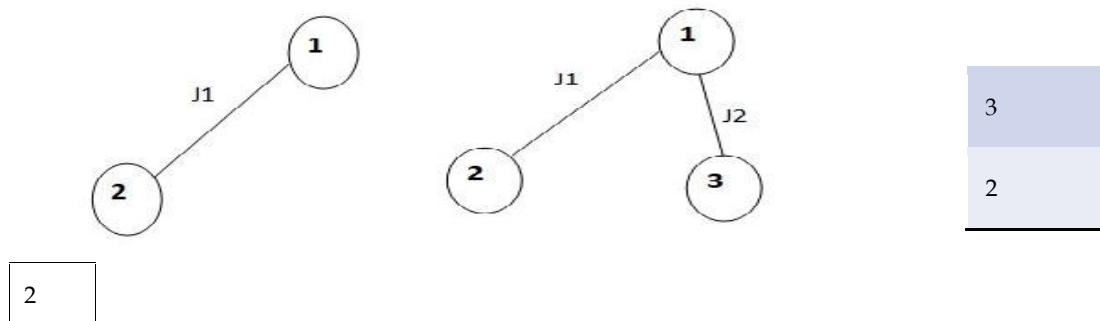
We are given with various jobs = {J1, J2, J3, J4}, P = {10, 5, 8, 3} and d = {1, 2, 1, 2}. Suppose I am doing two jobs, i.e., J1 and J4. So, there are two methods for generating a tree:
First Method: variable size jobs: {J1, J4} and **Second Method:** Fixed size jobs: {1, 0, 0, 1}

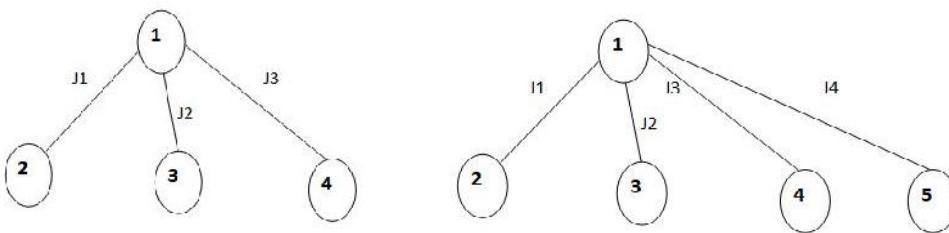
Variable size solution:

In variable size solution we write only those jobs which are active, or which are in running stage. At different times, different jobs can be active or inactive. Based upon that the bracket size will vary. That is why it is called variable size solution.

**Fixed Size Solution:**

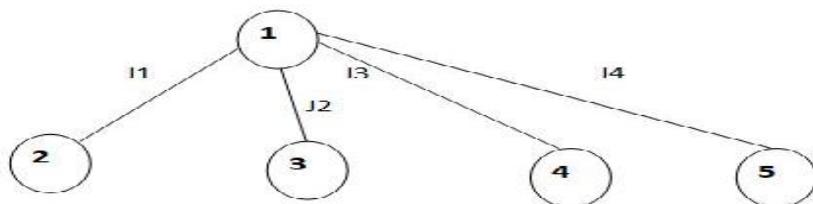
In fixed size solution, the bracket size will remain fixed. All the jobs will be included in that. The active jobs will be assigned value 1 and the inactive jobs will be assigned value 0. As the bracket size is fixed, that is why this kind is known as fixed size solution.



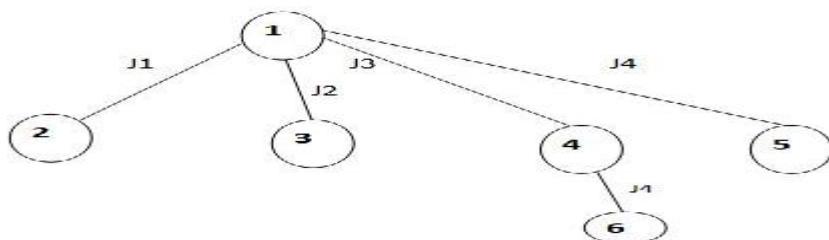


4
3
2

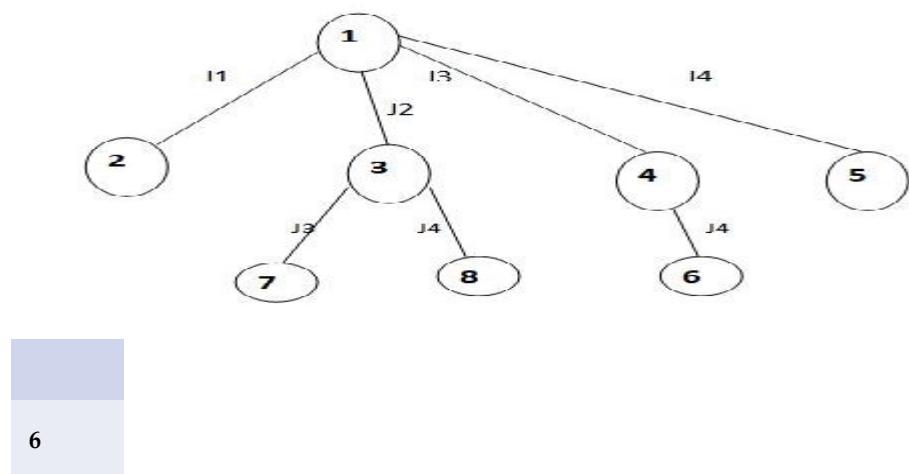
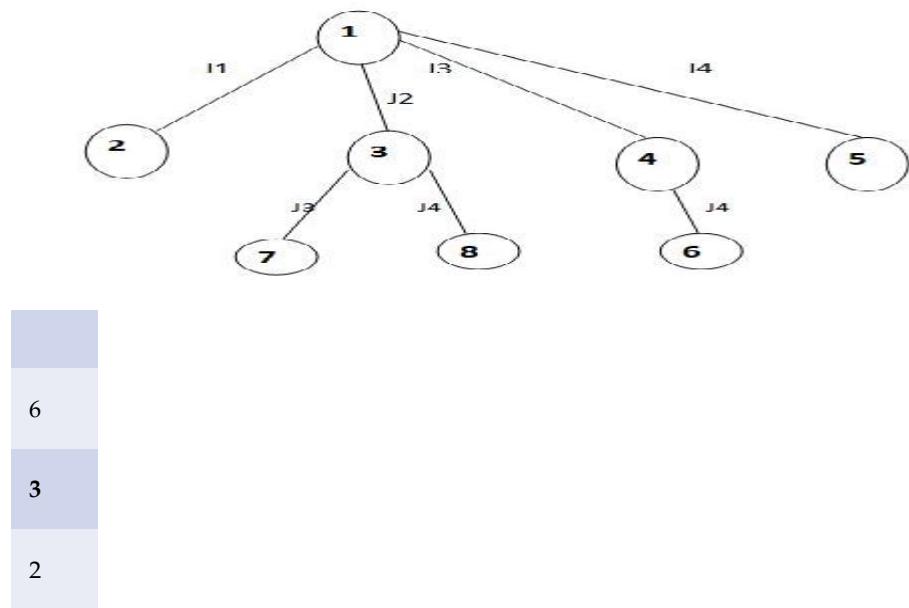
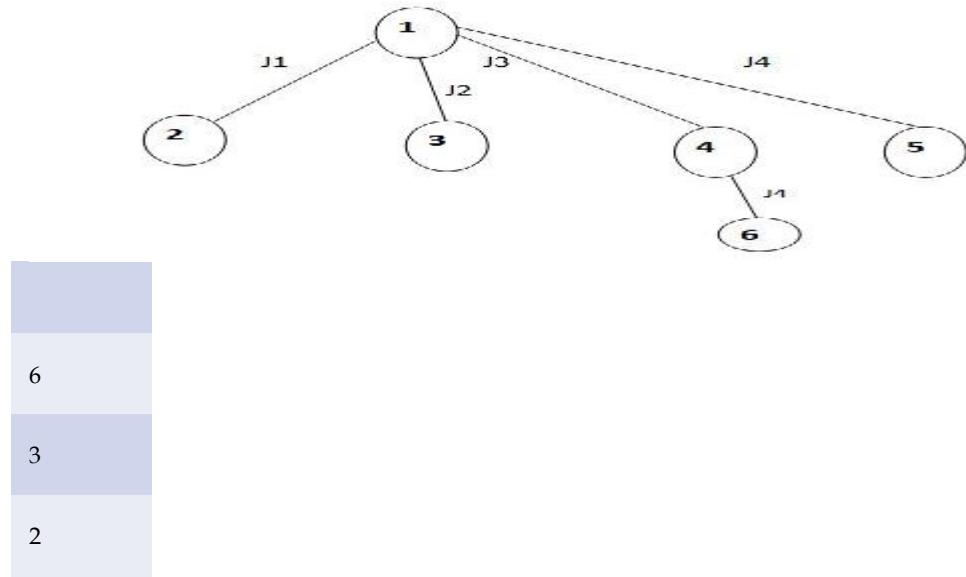
5
4
3
2



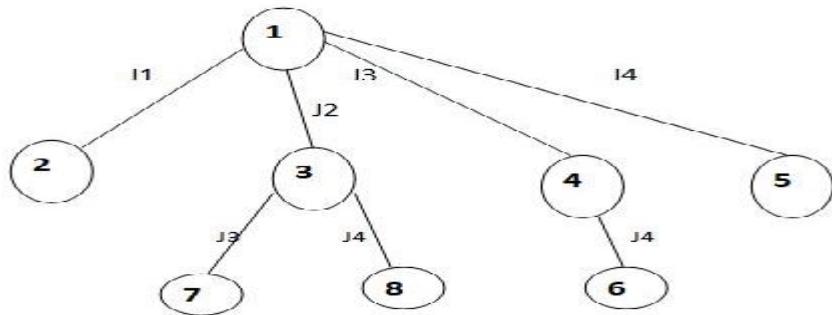
5
4
3
2



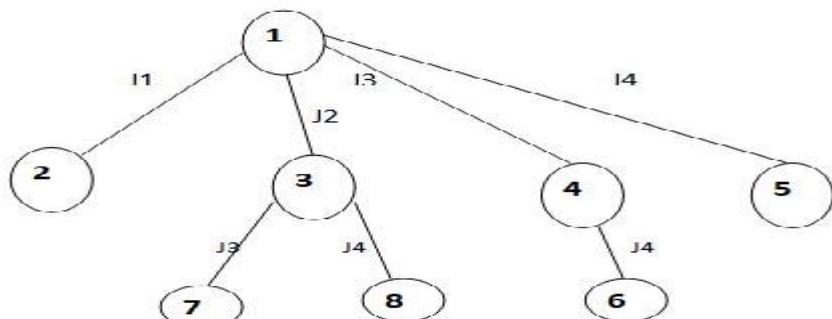
4
3
2



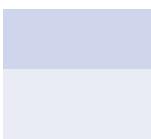
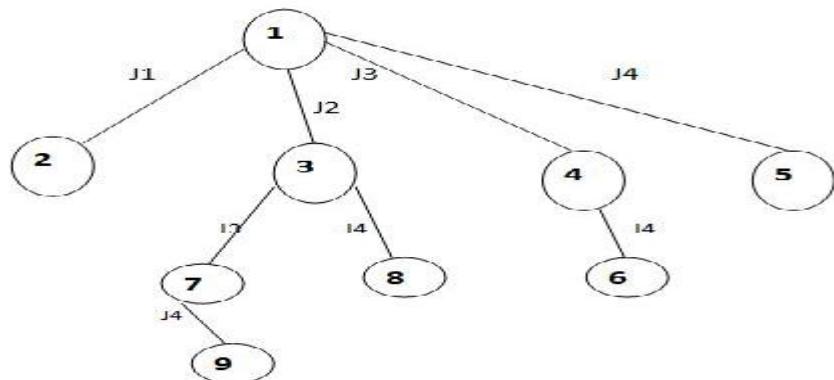
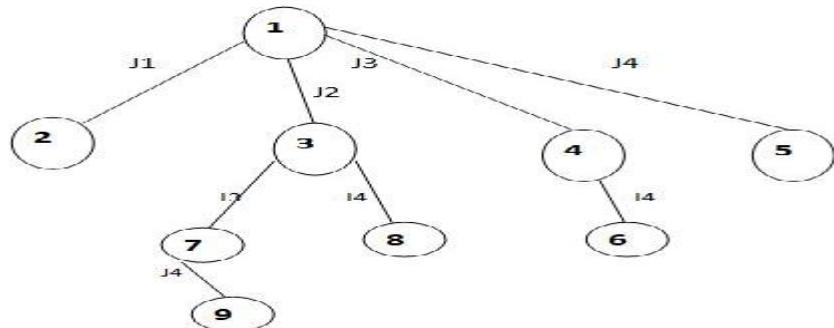
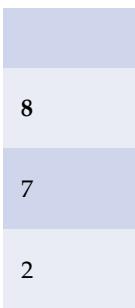
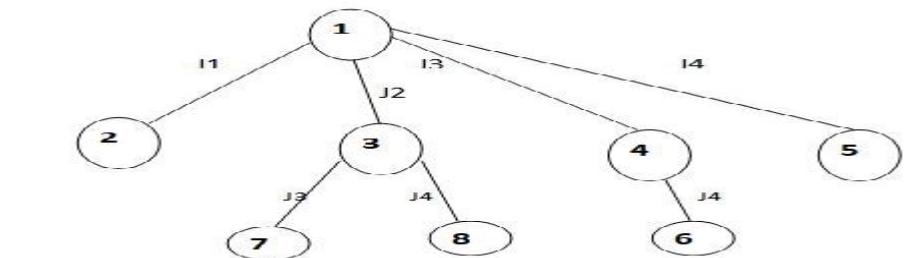
3
2



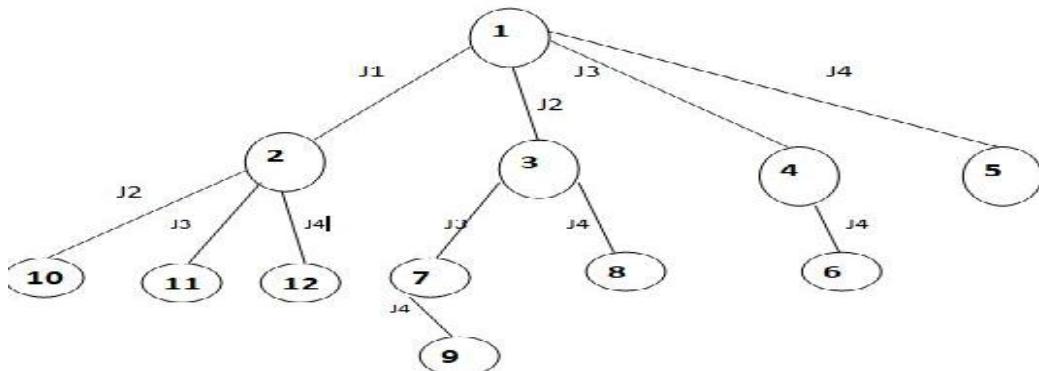
3
2



8
7
2



9
2



12
11
10

Difference between two methods

So based upon that the difference between these two methods are:

- In variable size solution, we use queue for next node exploration and in fixed size, we use stack for next node exploration.
- If queue is used, then it is called as FIFO branch and bound and if stack is used, then it is called as LIFO branch and bound.

Least Cost Branch and Bound

- Both FIFO and LIFO branch and bound are slower than least cost branch and bound.

Applications

So, the applications of branch and bound are: Job Sequencing with deadlines, 0/1 Knapsack problem and travelling salesman problem

7.2 0/1 Knapsack Problem

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays $val[0..n-1]$ and $wt[0..n-1]$ which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of $val[]$ such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item or don't pick it. It is called 0/1 property.

Given four jobs $J=\{1,2,3,4\}$, the profits $P=\{10,10,12,18\}$ and associated weights $W=\{2,4,6,9\}$ and $M = 15$, $n = 4$.

Algorithm Analysis and Design

	1	2	3	4
Profit	10	10	12	18
Weight	2	4	6	9

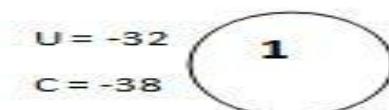
The total profit gain must be maximized. This problem 0/1 knapsack problem can also be solved using dynamic programming, greedy method. But we will solve it using branch and bound algorithm. The branch and bound is for optimization problems where we want only minimized results not maximized. We will convert the signs of profit from positive to negative and after solving the result can be converted back to positive. The rule is to explore that node whose cost is minimum, i.e., least cost branch and bound. For solving, each node in the state space tree we must find out the upper bound and the cost. Upper bound is defined as the sum of all the profits (without fraction). Cost is defined as the sum of all the profits (with fraction). The solution will be in the form of fixed size solution. Example: Suppose there are four objects; x_1, x_2, x_3 and x_4 and if only x_1 and x_3 are included then it will be $\{1, 0, 1, 0\}$.

	1	2	3	4
Profit	10	10	12	18
Weight	2	4	6	9

$M = 15, n = 4; \text{Upper bound} = \infty$

	1	2	3	4
Profit	10	10	12	18
Weight	2	4	6	9

$M = 15, n = 4; \text{Cost} = 10 + 10 + 12 + (18/9) * 3 = -38 \text{ and Upper bound} = 10 + 10 + 12 = -32$

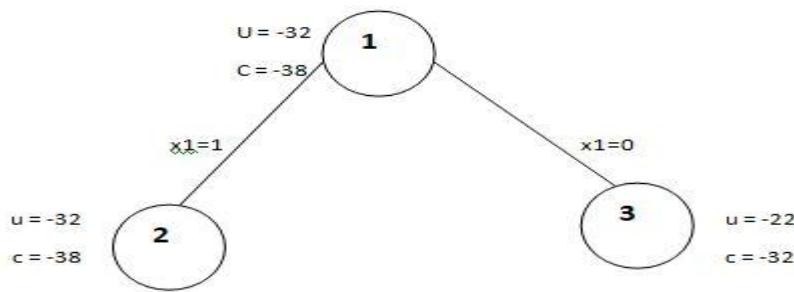


Upper bound = -32

	1	2	3	4
Profit	10	10	12	18
Weight	2	4	6	9

$M = 15, n = 4, \text{When } x_1=1, \text{Cost} = 10 + 10 + 12 + (18/9) * 3 = -38, \text{Upper bound} = 10 + 10 + 12 = -32$

When $x_1=0$, Cost = $10 + 12 + (18/9) * 5 = -32$, Upper bound = $10 + 12 = -22$



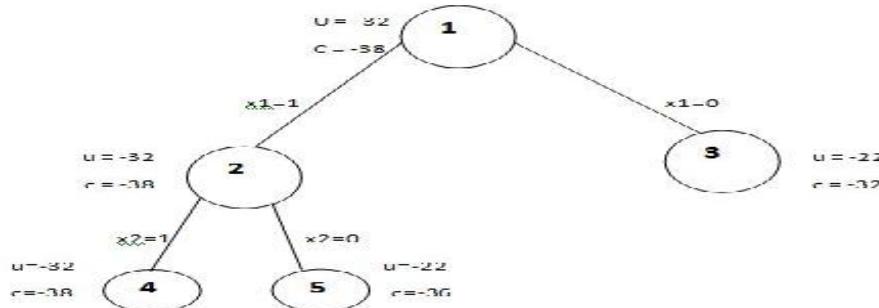
Upper bound = -32

Node 1 is done, that node will be explored next which has the minimum cost because we are following least cost branch and bound algorithm. So, in this example which will choose when $x_1 = 1$ because that is minimum.

	1	2	3	4
Profit	10	10	12	18
Weight	2	4	6	9

M = 15, n = 4, When $x_2=1$, Cost = $10 + 10 + 12 + (18/9)*3 = -38$, Upper bound = $10 + 10 + 12 = -32$

When $x_2=0$, Cost = $10 + 12 + (18/9)*7 = -36$, Upper bound = $10 + 12 = -22$



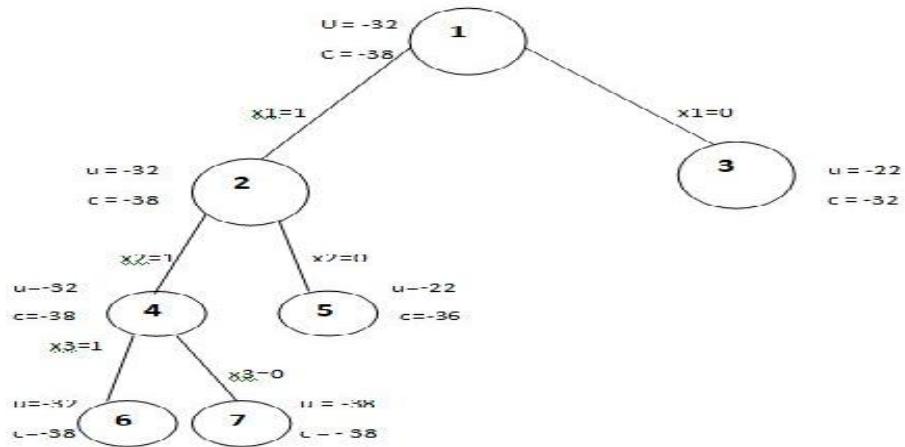
Upper bound = -32

- Next node will be 4th for exploration.

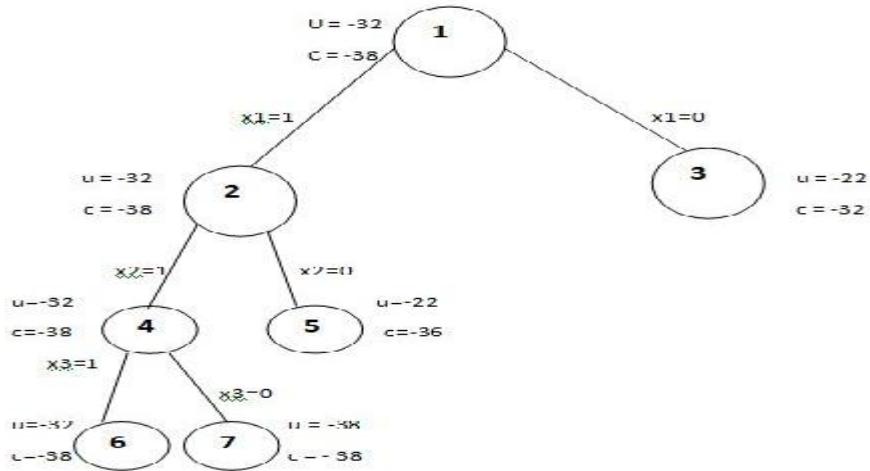
	1	2	3	4
Profit	10	10	12	18
Weight	2	4	6	9

M = 15, n = 4, When $x_3=1$, Cost = $10 + 10 + 12 + (18/9)*3 = -38$, Upper bound = $10 + 10 + 12 = -32$

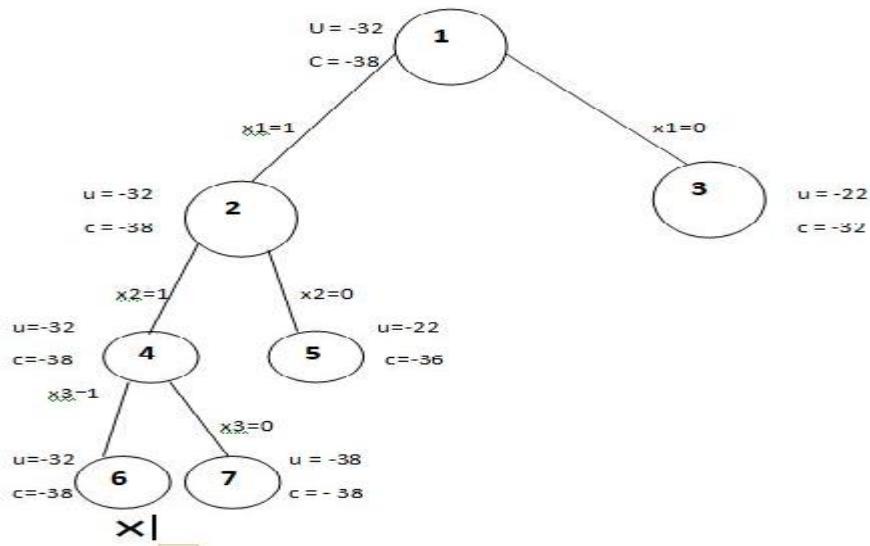
When $x_3=0$, Cost = $10 + 10 + 18 = -38$, Upper bound = $10 + 10 + 18 = -38$

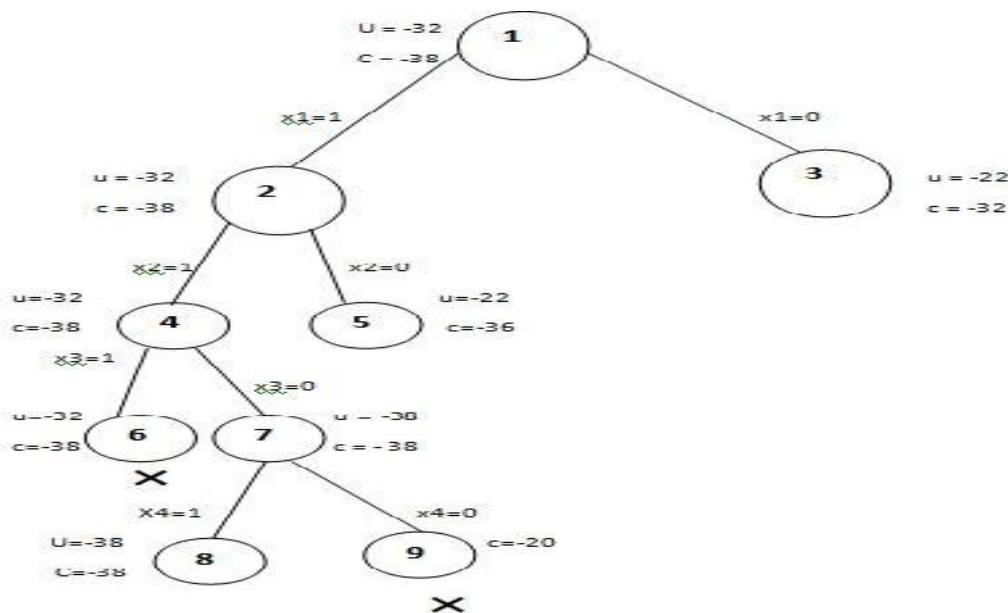


Upper bound = -38. Once we have got the new upper bound, at that time we will see all the alive nodes. Here nodes 5 and 3 are also alive nodes.



Node 5 is having the value of cost as -36 which is greater than -38, so kill the node 5. In the same way node 3 will be cancelled. Objects 1, 2, and 3 are included. Can we have object 4? No, because that will increase the capacity.





$X\{1,1,0,1\}$, Profits = $10 + 10 + 0 + 18 = 38$ and Weights = $2 + 4 + 0 + 9 = 15$

7.3 Travelling Salesman Problem

"Given a list of cities and the distances between each pair of cities, what is the **shortest** possible route that visits each city exactly once and returns to the origin city?". The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. It is a computational difficult problem, even though the problem is computationally difficult, many heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%.

Applications of TSP

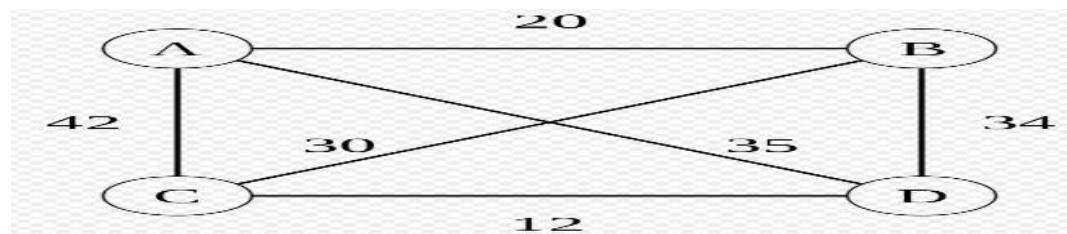
The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents travelling times or cost, or a similarity measure between DNA fragments. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources; in such problems, the TSP can be imbedded inside an optimal control problem. In many applications, additional constraints such as limited resources or time windows may be imposed.

TSP can be modeled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. Often, the model is a complete graph (i.e., each pair of vertices is connected by an edge).

Types of TSP

- **Symmetric TSP:** In the **symmetric TSP**, the distance between two cities is the same in each opposite direction, forming an undirected graph. This symmetry halves the number of possible solutions.
- **Asymmetric TSP:** In the **asymmetric TSP**, paths may not exist in both directions or the distances might be different, forming a directed graph.

The symmetric travelling salesman problem (TSP) is the problem of finding the shortest Hamiltonian cycle (or tour) in a weighted finite undirected graph without loops. The below given graph is an example of symmetric TSP.

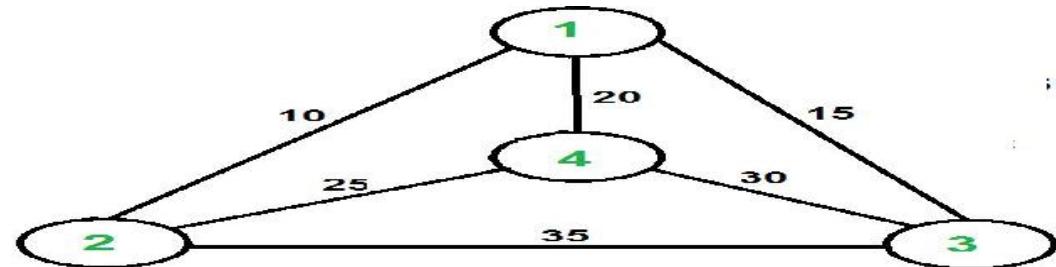


Difference between Hamiltonian cycle and TSP+

Both problems seem like each other. But these two are different. The Hamiltonian cycle problem is to find if there exist a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.



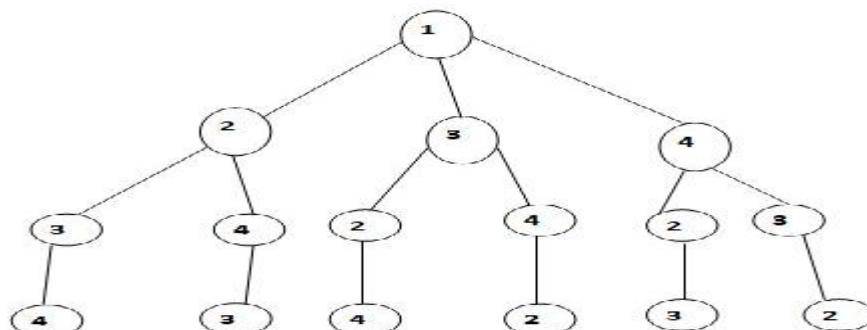
Example:



Various possible paths are:

1 - 2 - 4 - 3 - 1, 1 - 4 - 2 - 3 - 1, 1 - 3 - 4 - 2 - 1, 1 - 3 - 2 - 4 - 1 and many others.

State space tree



Difference between backtracking and branch-and-bound

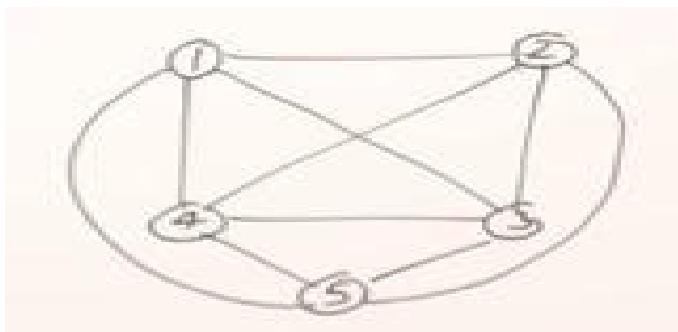
Backtracking and branch-and-bound both are similar. Both use the state space tree but the approach for solving is different. In backtracking, the tree is generated in depth first search manner or pre-order. If there are multiple solutions and for every node, it will provide a solution. Here we don't want all the solutions, we just want a solution with the minimum cost. One approach in backtracking can be added that if we add one more step of rejecting the paths which gives us the more cost, then the Travelling Salesman problem can be solved using backtracking. But it takes a lot of time and adds a lot of computation and increases the complexity of algorithm. So, for the optimization problems we avoid the approach of backtracking instead we use it for permutation problem.

Application of branch-and-bound

The branch-and-bound generates the node in this manner (breadth first search). While generating the nodes, it calculates the cost for each node. If for any node, we are getting the cost greater than upper bound, then we kill that node. So, we will generate only those nodes which are fruitful.



Example:



Cost adjacency matrix is given.

	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

We need to reduce this matrix means we have to see and write the minimum values in each row. Subtract that minimum value from all of the elements of the rows. By this way the rows will be reduced.

	1	2	3	4	5	Minimum value
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

	1	2	3	4	5	Minimum value
1	∞	20-10	30-10	10-10	11-10	10

Algorithm Analysis and Design

2	15-2	∞	16-2	4-2	2-2	2
3	3-2	5-2	∞	2-2	4-2	2
4	19-3	6-3	18-3	∞	3-3	3
5	16-4	4-4	7-4	16-4	∞	4

	1	2	3	4	5	Minimum value
1	∞	10	20	0	1	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	15	∞	0	3
5	12	0	3	12	∞	4

All the rows are reduced now, and the total cost of reduction is = $10 + 2 + 2 + 3 + 4 = 21$

	1	2	3	4	5
1	∞	10	20	0	1
2	13	∞	14	2	0
3	1	3	∞	0	2
4	16	3	15	∞	0
5	12	0	3	12	∞
Minimum Value	1	0	3	0	0

Again, we need to reduce this matrix means we have to see and write the minimum values in each column. Subtract that minimum value from all of the elements of the columns. By this way the columns will be reduced.

	1	2	3	4	5
	1	0	3	0	0

Unit 07: Branch and Bound

1	∞	10-0	20-3	0-0	1-0
2	13-1	∞	14-3	2-0	0-0
3	1-1	3-0	∞	0-0	2-0
4	16-1	3-0	15-3	∞	0-0
5	12-1	0-0	3-3	12-0	∞
Minimum Value	1	0	3	0	0

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞
Minimum Value	1	0	3	0	0

All the columns have been reduced and the cost of reduction of columns is == $1 + 0 + 3 + 0 + 0 = 4$

- Cost of reducing rows = 21, Cost of reducing columns = 4 and the total cost is $21 + 4 = 25$

Reduced Matrix for node 1 (vertex 1)

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2

Algorithm Analysis and Design

4	15	3	12	∞	0
5	11	0	0	12	∞

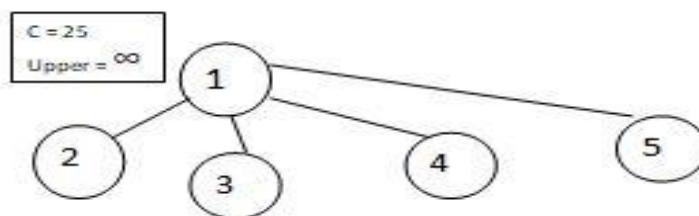
This 25 which is the cost of reducing matrix is the minimum cost. The cost can be greater than this.

State space tree

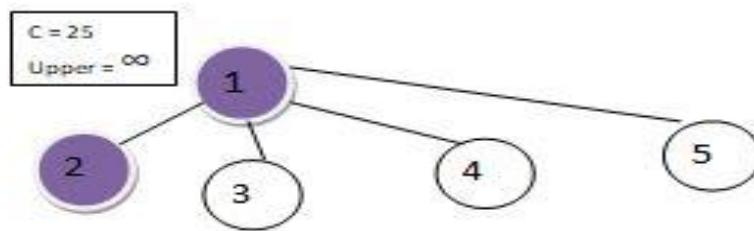
	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

(1)

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞



We have an upper bound. Initially the value is infinity. This upper bound is not updated every time. Once we will reach the leaf node, then we will update the upper. So, we will update the cost every time.



Now we will calculate the cost of 2nd matrix. That means we are going from vertex 1 to vertex 2. For this we will make elements of 1st row and 2nd columns as infinity.

From vertex 1 to vertex 2

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

Once we reach from vertex 1 to vertex 2, we are not supposed to go back to vertex 1 so, all those elements must be infinity.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

Now check whether the matrix is reduced or not. See all the rows and columns have element 0 so all are reduced. Now we will calculate the cost.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0

Algorithm Analysis and Design

3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

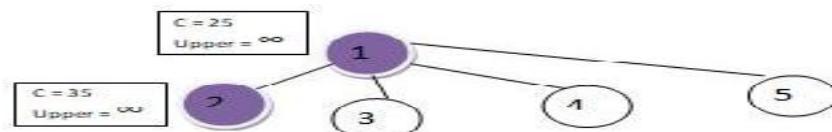
Cost of reduction = $C(1,2) + r(\text{cost of previous red}) + r^*(\text{any other reduction if you have done})$.

$C(1,2) = 10$, $r(\text{cost of main red}) = 25$, $r^* = 0$. So, cost of reduction = $10 + 25 + 0 = 35$

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

Cost of reduction = $C(1,2) + r(\text{cost of previous red}) + r^*(\text{any other reduction if you have done})$

$C(1,2) = 10$, $r(\text{cost of main red}) = 25$, $r^* = 0$. So, cost of reduction = $10 + 25 + 0 = 35$



	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

For calculation of cost from vertex 1 to vertex 3, make 1st row and 3rd column as infinity.

Cost from vertex 1 to vertex 3

	1	2	3	4	5

Unit 07: Branch and Bound

1	∞	∞	∞	∞	∞
2	12	∞	∞	2	0
3	0	3	∞	0	2
4	15	3	∞	∞	0
5	11	0	∞	12	∞

As we can't go back from vertex 3 to vertex 1. So that also need to be infinity

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	2	0
3	∞	3	∞	0	2
4	15	3	∞	∞	0
5	11	0	∞	12	∞

As we can't go back from vertex 3 to vertex 1. So that also need to be infinity. Now check whether the matrix is reduced or not

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	12	∞	∞	2	0	0
3	∞	3	∞	0	2	0
4	15	3	∞	∞	0	0
5	11	0	∞	12	∞	0

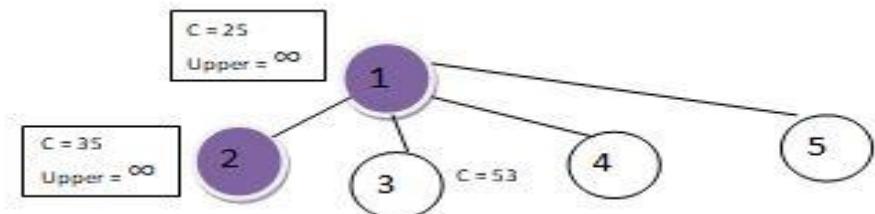
Algorithm Analysis and Design

	11	0		0	0	
--	----	---	--	---	---	--

Column 1 is not having 0, the minimum value is 11. So subtract 11.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	2	0
3	∞	3	∞	0	2
4	4	3	∞	∞	0
5	0	0	∞	12	∞

$$\text{Cost} = C(1,3) + r + r^*. \text{Cost} = 17 + 25 + 11 = 53$$



Cost from vertex 1 to vertex 4

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

Cost of reduction = 25. Cost from vertex 1 to vertex 5

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	10	∞	9	0	∞
3	0	3	∞	0	∞
4	12	0	9	∞	∞
5	∞	0	0	12	∞

Cost of reduction = 31

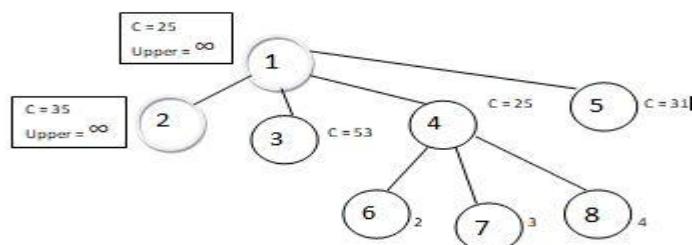
FIFO branch and bound

Next, the children of vertex 2 will be explored, then vertex 3, then vertex 4 and then vertex 5. Then it is called as FIFO branch and bound.

Least Cost Branch and Bound

Because vertex 4 is having least cost, so it will be explored first and then afterwards we will see which one will be explored.

Exploration of node 4



Take the matrix of vertex 4 and use it for further processing.

Cost from vertex 4 to vertex 2

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0

Algorithm Analysis and Design

5	11	0	0	∞	∞
---	----	---	---	----------	----------

We must see the cost from vertex 4 to vertex 2, so make the row 4 and column 2 as infinity.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

We should not go back to vertex 2 so that will also be infinity

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

See whether it is reduced or not

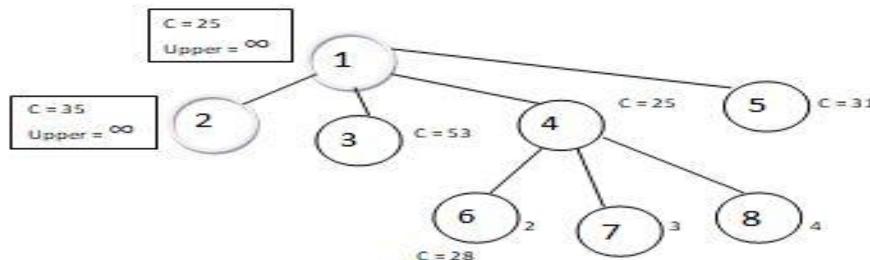
	1	2	3	4	5	Min
1	∞	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0	0
3	0	∞	∞	∞	2	0

4	∞	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞	0
Min	0	∞	0	∞	0	0

Already reduced. Need not to reduce further.

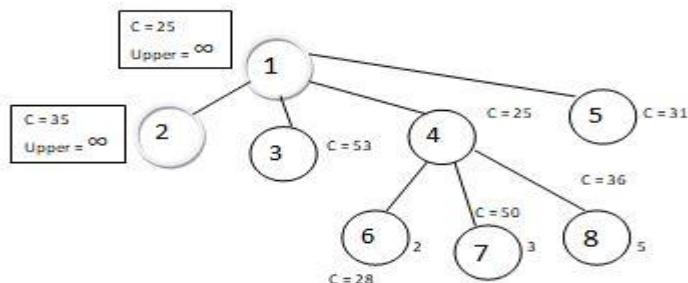
$$\text{Cost} = C(4,2) + C(\text{vertex } 4) + r^* = 3 + 25 + 0 = 28$$

Cost from vertex 4 to vertex 3

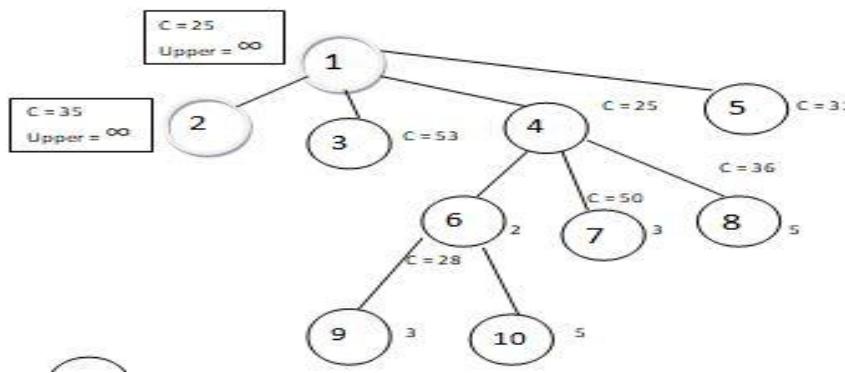


Least cost is for vertex 2(node 6). The path chosen till here is as: v1 - v4 - v2. The remaining vertices are: v3 and v5. So, these will be explored now.

Cost from vertex 4 to vertex 3 and vertex 4 to vertex 6



Next explored



Take the matrix of node 6 for node 9 (vertex 3)

	1	2	3	4	5

Algorithm Analysis and Design

1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

We need to make 2nd row and 3rd column as infinity.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	∞	∞	∞

Vertex 3 to vertex 1 will also be infinity

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	2
4	∞	∞	∞	∞	∞

Unit 07: Branch and Bound

5	11	∞	∞	∞	∞
---	----	----------	----------	----------	----------

Now check whether it is reduced or not. No, it is not reduced. So, reduce it.

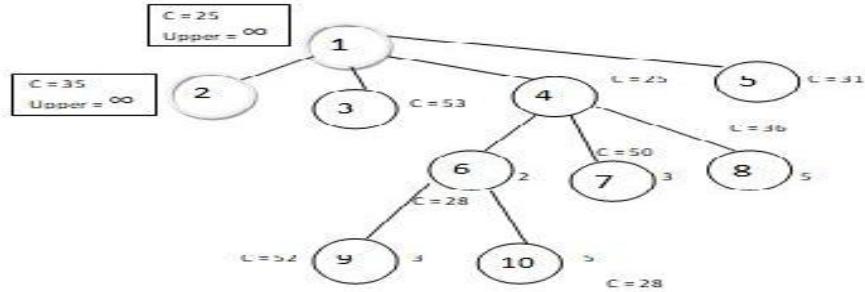
	1	2	3	4	5	Min
1	∞	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	2	2
4	∞	∞	∞	∞	∞	∞
5	11	∞	∞	∞	∞	∞
Min	11	∞	∞	∞	∞	

Subtract the values and get the matrix

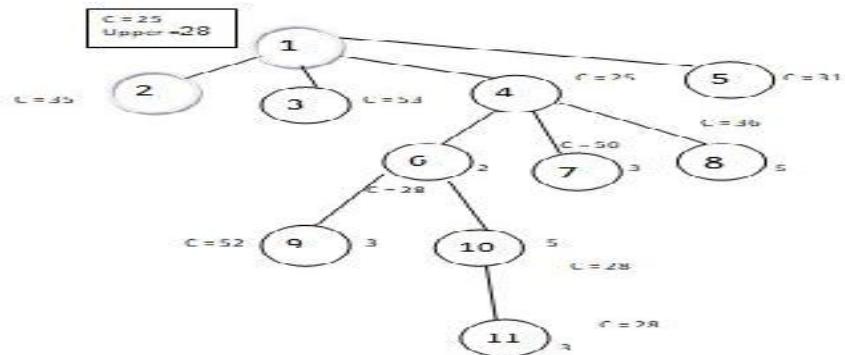
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞
5	0	∞	∞	∞	∞

Calculate the cost = $C(2,3) + C(6) + 13 = 11 + 28 + 13 = 52$

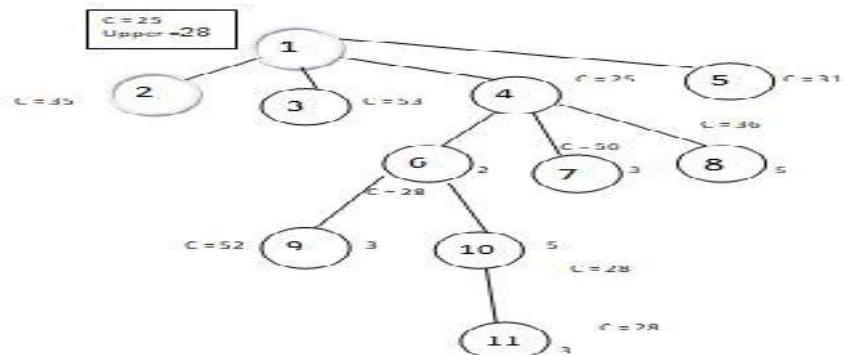
- The cost is 28.

Algorithm Analysis and Design

Next, node 10 (vertex 5) will be chosen as this is having least cost. The vertex remaining is 3 only. If we take 5th row and 3rd column as infinity. Then every element in the matrix will be infinity. Then the cost will be same. Cost = 28. Finally we reach the leaf node. Still we need to go back to vertex 1. Once we have reached to the leaf node, then update the upper bound = 28.



New upper bound is 28. Check the nodes whose cost is greater than upper bound. If any node is having cost greater than 28 then it will be killed.



Here every node's cost is greater than 28 then all the nodes will be killed except node 11 (vertex 3). So, the answer or path is 1 - 4 - 2 - 5 - 3 - 1. Cost of tour is 28.

Time Complexity

The worst-case complexity of Branch and Bound remains same as that of the Brute Force clearly because in worst case, we may never get a chance to prune a node. Whereas, in practice it performs very well depending on the different instance of the TSP. The complexity also depends on the choice of the bounding function as they are the ones deciding how many nodes to be pruned.

Summary

- Branch and Bound follows Breadth First Search.
- Branch and bound strategy are like backtracking because it also uses state space tree for solving the problem.

- Branch and bound follows BFS and backtracking follow DFS.
- If queue is used, then it is called as FIFO branch and bound and if stack is used, then it is called as LIFO branch and bound.
- Both FIFO and LIFO branch and bound are slower than least cost branch and bound.
- The branch and bound is for optimization problems where we want only minimized results not maximized.
- In branch and bound we will convert the signs of profit from positive to negative and after solving the result can be converted back to positive.

Keywords

- **Variable size solution:** In variable size solution, we use queue for next node exploration and in fixed size, we use stack for next node exploration.
- **0/1 Knapsack problem:** Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.
- **Travelling salesman problem:** "Given a list of cities and the distances between each pair of cities, what is the **shortest** possible route that visits each city exactly once and returns to the origin city?".
- **Symmetric TSP:** In the **symmetric TSP**, the distance between two cities is the same in each opposite direction, forming an undirected graph. This symmetry halves the number of possible solutions.
- **Asymmetric TSP:** In the **asymmetric TSP**, paths may not exist in both directions or the distances might be different, forming a directed graph.

Self Assessment

1. In 0/1 knapsack problem, the total profit must be
 - Minimized
 - Maximized
 - Zero
 - None of the above
2. The 0/1 Knapsack problem can be efficiently solved using
 - Dynamic programming
 - Branch and Bound
 - Backtracking
 - Greedy approach
3. Which of these variations of branch and bound is used for solving 0/1 Knapsack problem?
 - FIFO Branch and Bound
 - LIFO Branch and Bound
 - Least Cost Branch and Bound
 - Most Cost Branch and Bound
4. The upper bound in 0/1 knapsack problem considers

- A. Values without fraction
 - B. Values with fraction
 - C. Both of these
 - D. None of the above
5. The costs in 0/1 knapsack problem considers
- A. Values without fraction
 - B. Values with fraction
 - C. Both of these
 - D. None of the above
6. In travelling salesman problem, the _____ path is found out.
- A. Shortest
 - B. Longest
 - C. Both of the above
 - D. None of the above
7. In which of the variant, the number of possible solutions is less?
- A. Symmetric TSP
 - B. Asymmetric TSP
 - C. Disymmetric TSP
 - D. None of the above
8. Which of these problems seems similar to TSP?
- A. Sorting
 - B. Searching
 - C. Hamiltonian Cycle
 - D. Merging
9. Find the odd one out.
- A. Branch and bound
 - B. TSP
 - C. Optimization problem
 - D. Binary Searching
10. In TSP, if for any node, the cost is greater than upper node then
- A. That node is killed
 - B. That node is explored
 - C. That node is used
 - D. None of the above
11. The approach followed by branch and bound is
- A. Breadth First Search

Unit 07: Branch and Bound

- B. Depth First Search
- C. Left Right Search
- D. Right Left Search

- 12. The state space tree is used in
 - A. Branch and Bound
 - B. Backtracking
 - C. Both of the above
 - D. None of the above

- 13. Which of these problems is solved by using branch and bound?
 - A. Minimization problem
 - B. Maximization problem
 - C. High density problem
 - D. Low density problem

- 14. Which of these are variants of branch and bound?
 - A. FIFO branch and bound
 - B. LIFO branch and bound
 - C. Least cost branch and bound
 - D. All of the above

- 15. Which of these is faster?
 - A. FIFO branch and bound
 - B. LIFO branch and bound
 - C. Least cost branch and bound
 - D. Most cost branch and bound

Answers for Self Assessment

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

Review Questions

1. What is branch and bound method? How is it different from backtracking algorithm?
2. State the various problems which can be solved using branch and bound method? What is strategy behind using branch and bound method?
3. What is job sequencing with deadlines? State the problem using one example.
4. What is 0/1 knapsack problem? Explain using one example.
5. What is travelling salesman problem? Explain it using one example.
6. What is the difference between Knapsack problem and 0/1 Knapsack problem?



Further Readings

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_travelling_salesman_problem.htm



Web Links

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_job_sequencing_with_deadline.htm

<https://www.javatpoint.com/0-1-knapsack-problem>

Unit 08: Pattern Matching

CONTENTS

- Objectives
- Introduction
- 8.1 Various Algorithms
- 8.2 Brute Force
- 8.3 Knuth-Morris-Pratt (KMP) Algorithm
- 8.4 Boyer Moore Algorithm
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the pattern matching algorithm
- Understand the Brute Force algorithm for pattern matching
- Know the drawbacks of Brute Force pattern matching algorithm
- Understand the Knuth-Morris-Pratt (KMP) algorithm
- understand the Boyer Moore algorithm

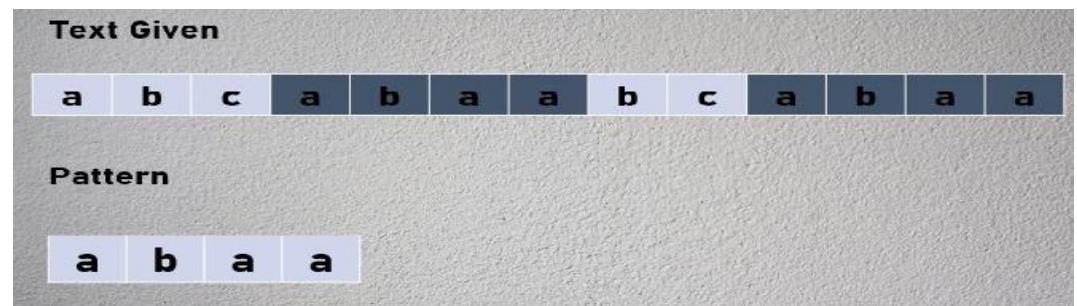
Introduction

String or pattern matching algorithms try to find a place where one or several strings (called a pattern) are found within a text. Generally, in a text file/excel file or any word file, we press the combination CTRL+F to find something in that. So when we use any kind of string or pattern matching algorithm, then the questions that comes in mind are: Does the given string exist in the text? How many times the string occurs in the text?

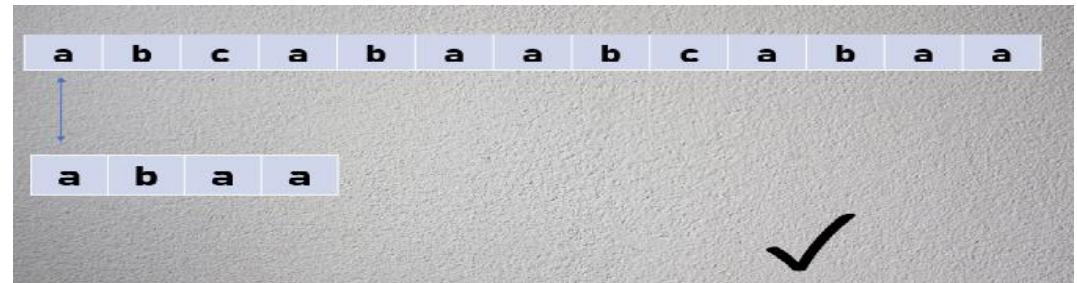
Given an element space, we have: Text array: $T[1....n]$, Pattern array: $P[1....m]$, $m \leq n$, Element of P and $T \in \Sigma$ and these can be either $\{0, 1, \dots, 9\}$ or $\{a, b, \dots, z\}$.

Basic Phenomenon

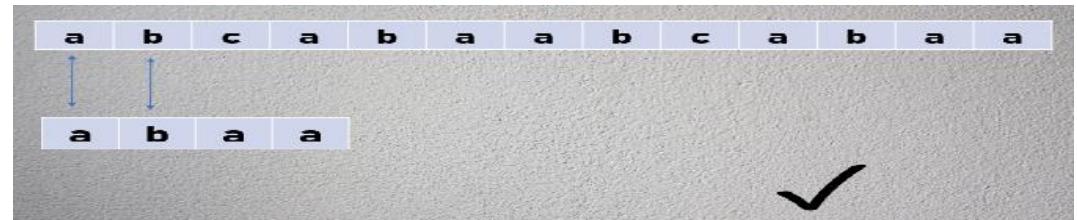
Given text is 'abcabaabcaaa' and the pattern is 'abaa'. We need to find whether the pattern exists in the text or not. If yes, then at which location? So, if look at the pattern in the text, then we can easily find out that the pattern exists in the text. Because the length of string is very less, that is why it is easy to find out. What if the text is large, then we have to apply some kind of pattern matching algorithm. The basic phenomenon of pattern matching algorithm is:



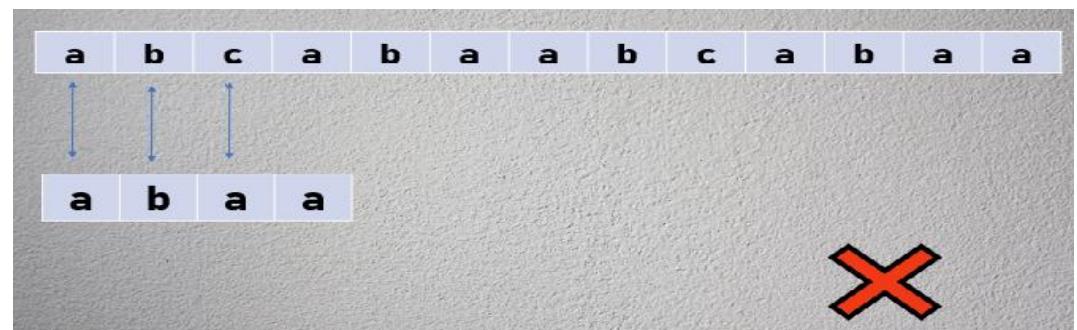
It starts with matching the first alphabet of text with first alphabet of pattern.



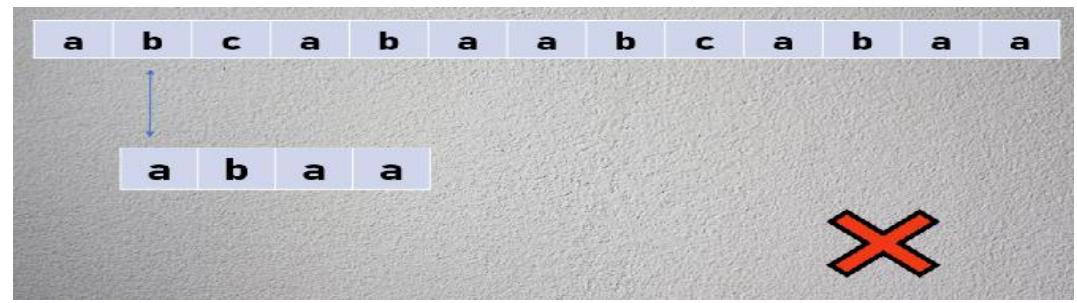
It both matches, then it matches second alphabet of text with second alphabet of pattern.



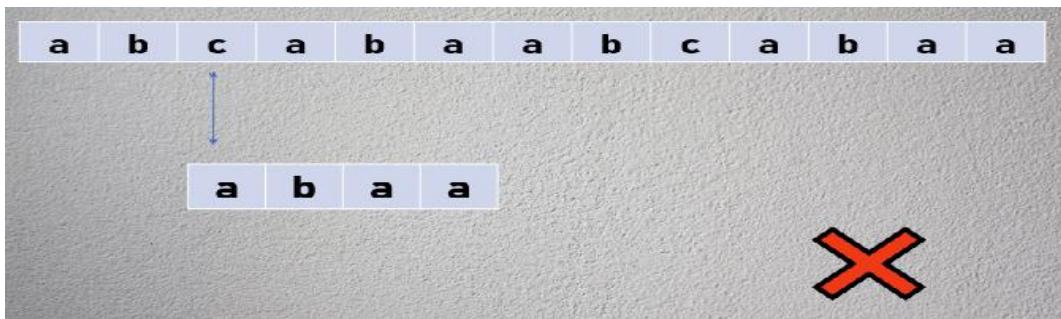
It both matches, then it matches third alphabet of text with third alphabet of pattern



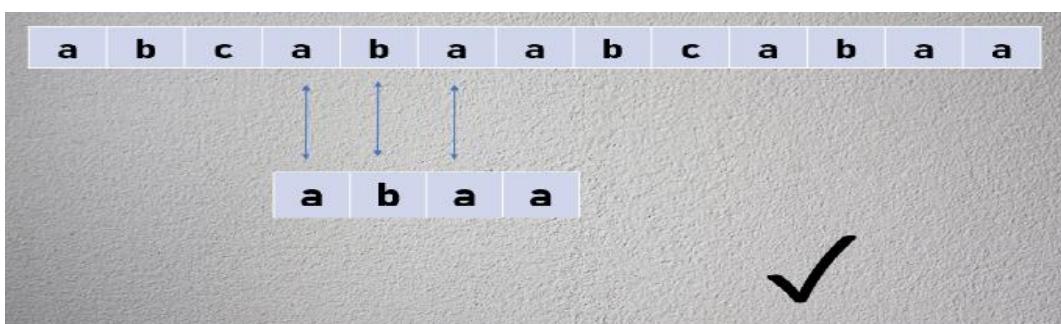
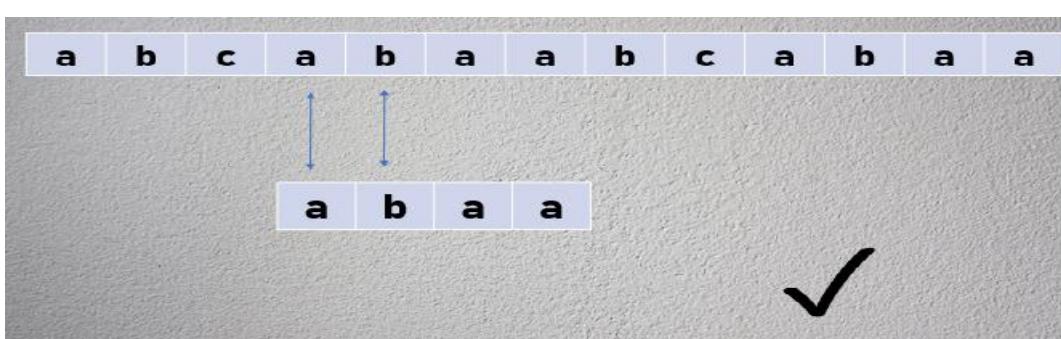
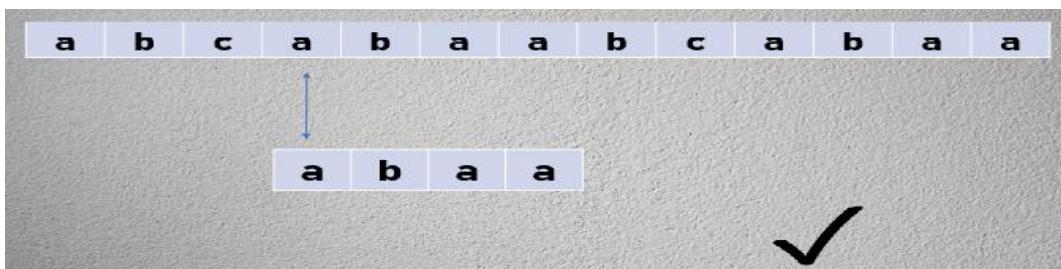
As it does not match, so the pattern will move further one position. Now the second alphabet of text will be matched with the first alphabet of pattern.



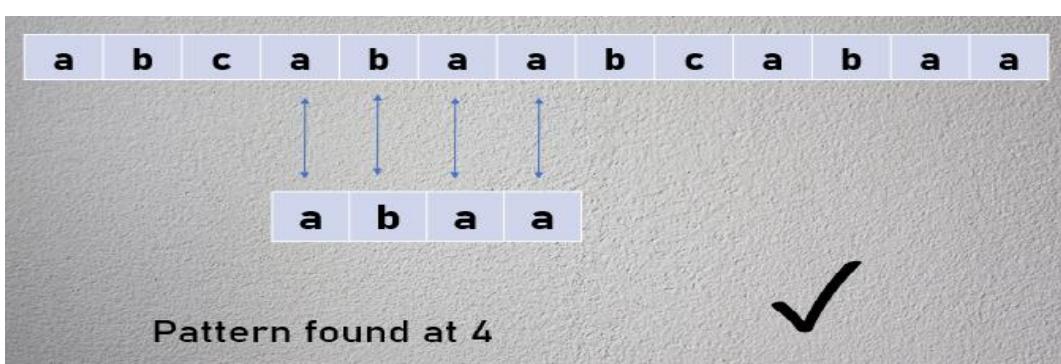
No match. So, move one position further.

Unit 08: Pattern Matching

Move one position further.

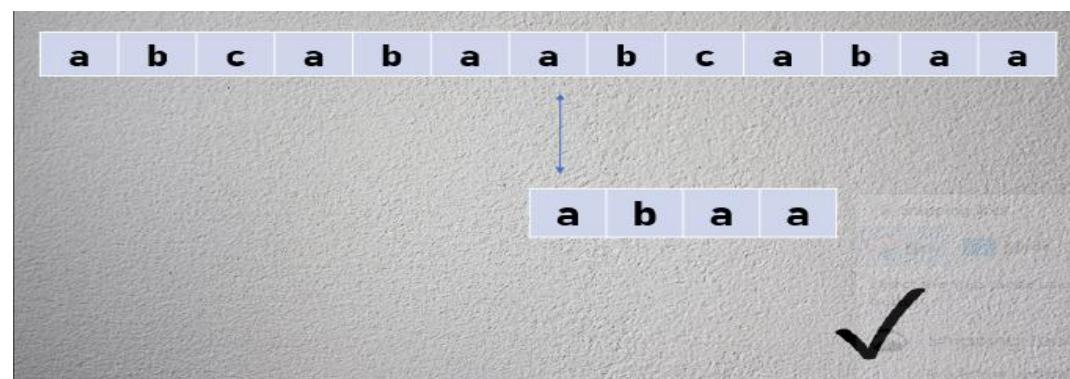
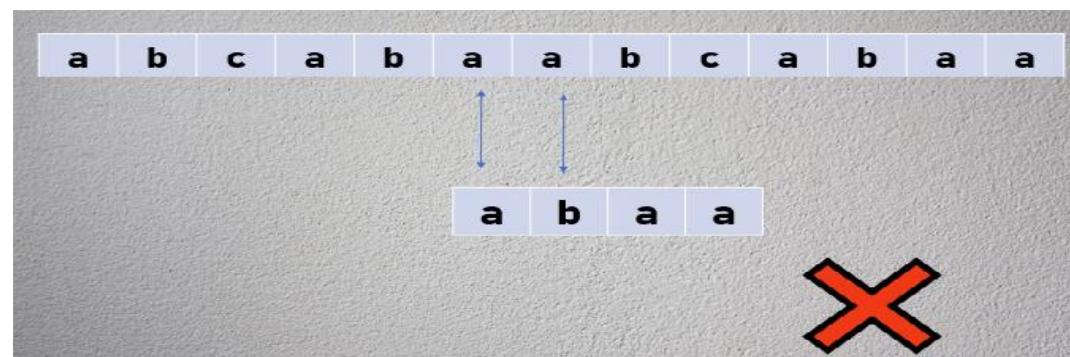
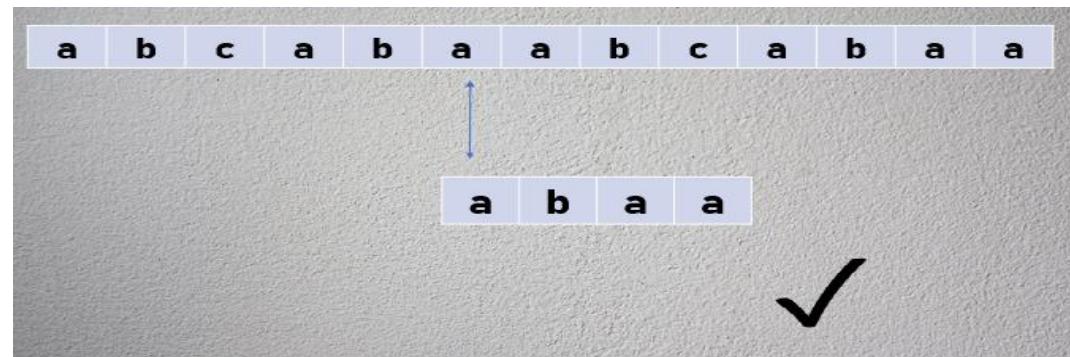
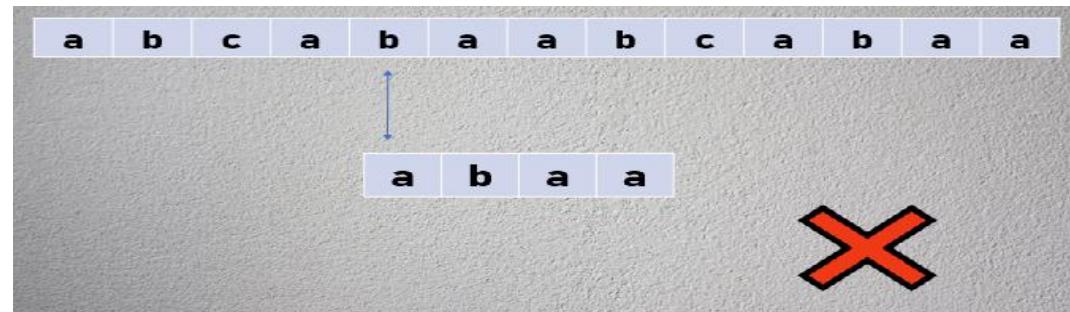


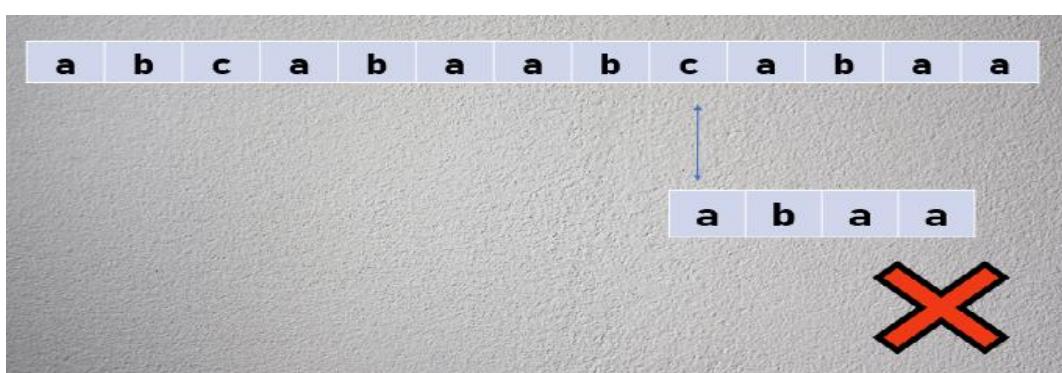
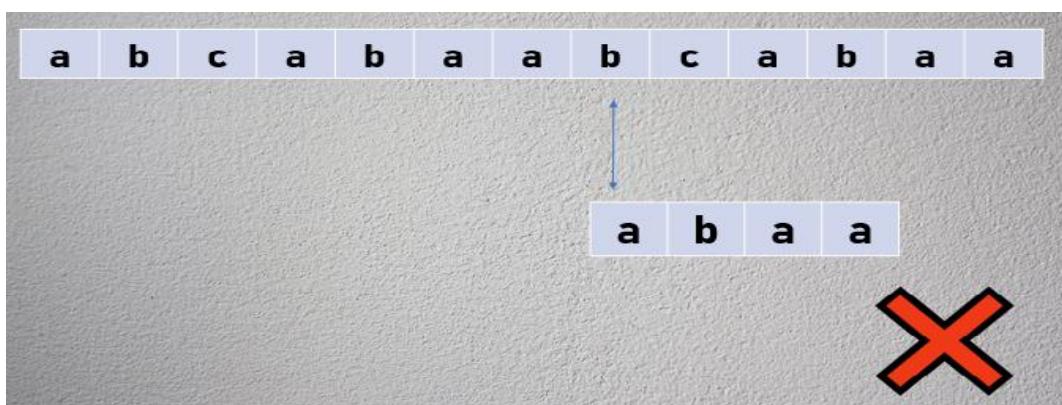
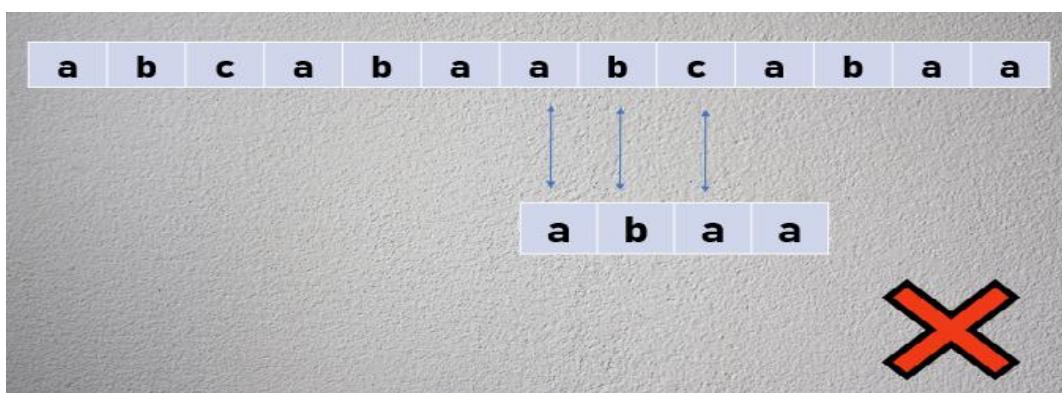
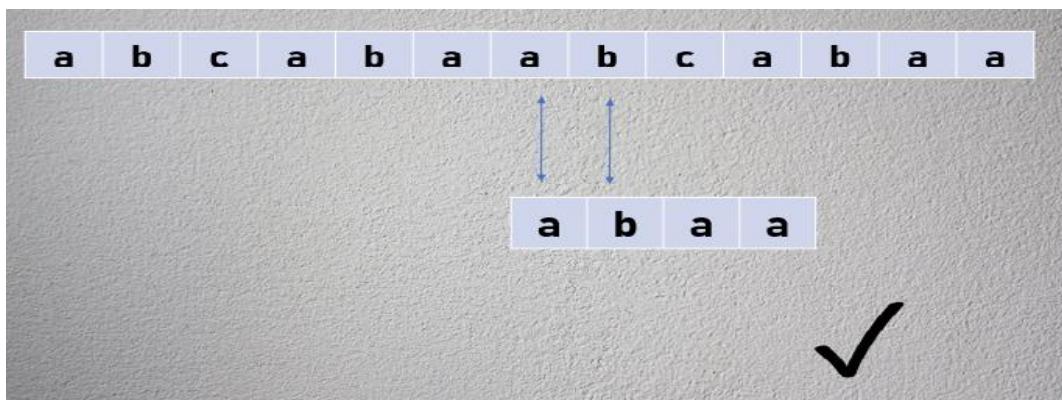
The four alphabets of pattern are matching with four alphabets of text. So, it is a hit and the pattern is found at 4.

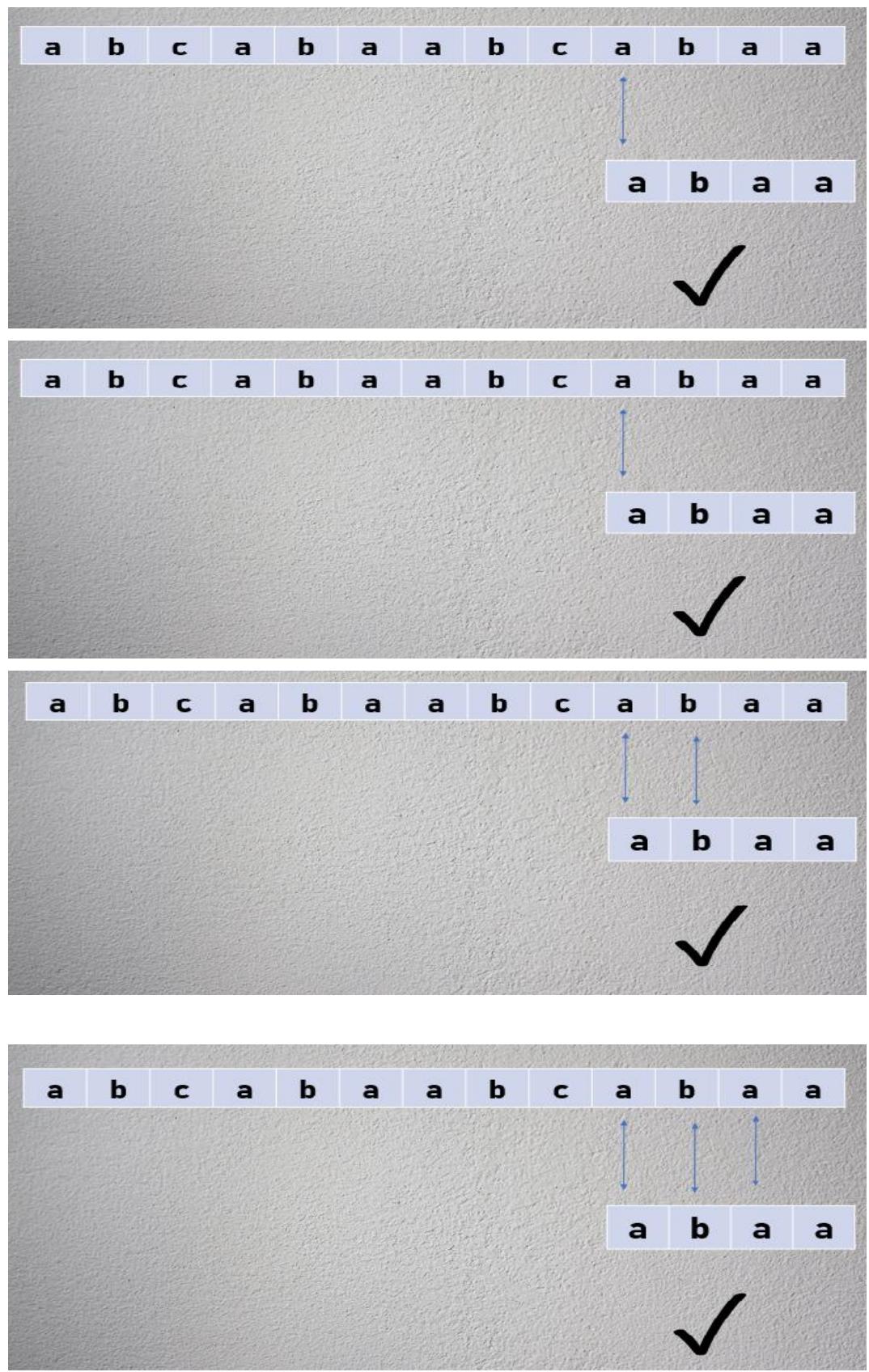


Algorithm Analysis and Design

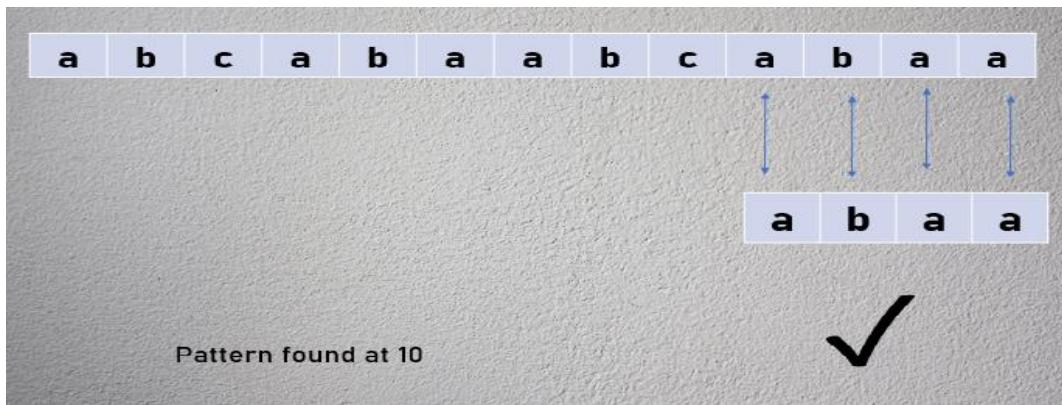
Now we will check is there any other existence of the pattern in the text. So, move one position further.



Unit 08: Pattern Matching



So, the pattern exists in the text at 10.

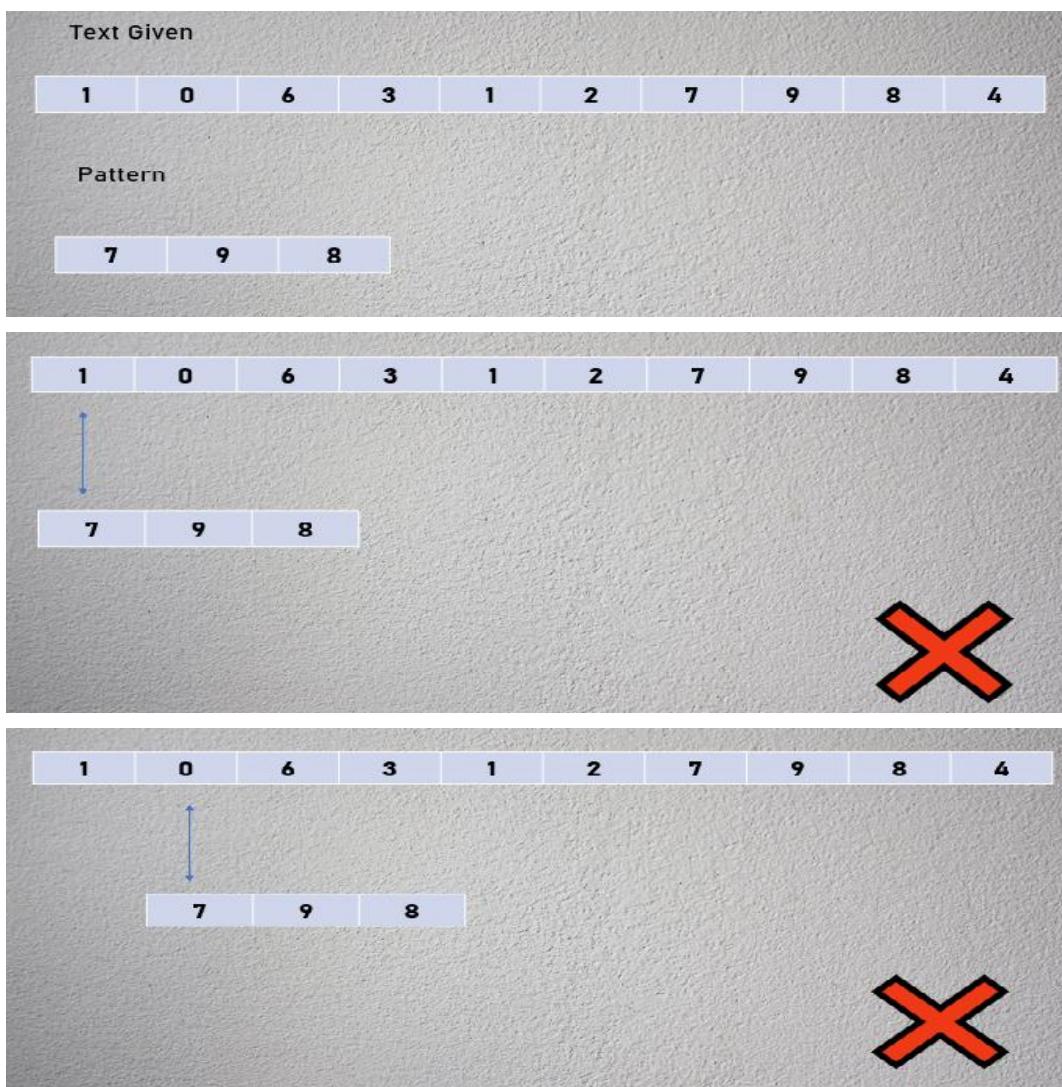
Unit 08: Pattern Matching

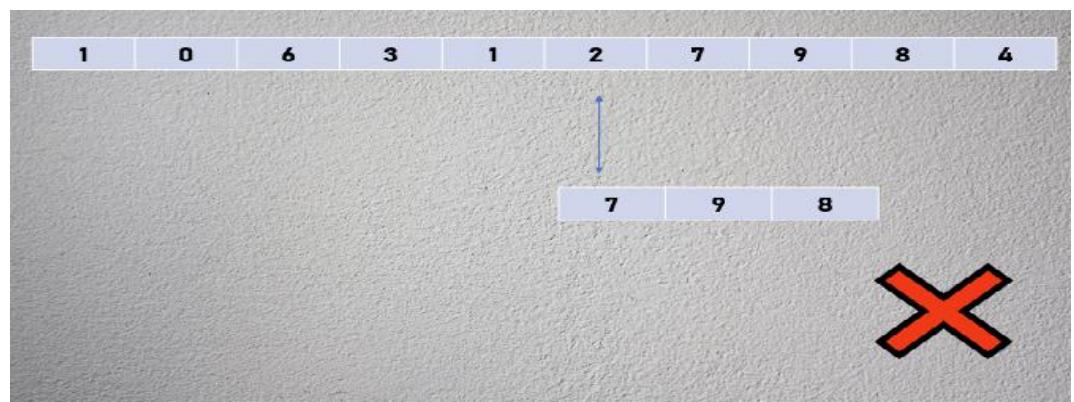
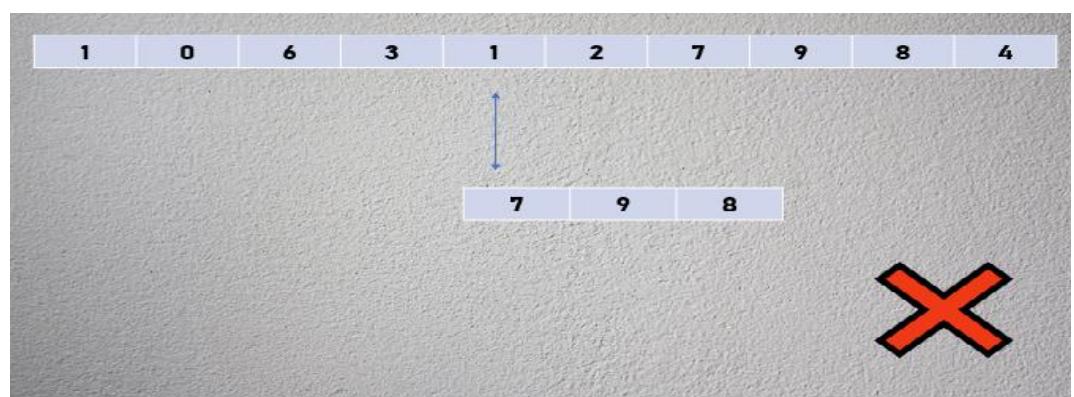
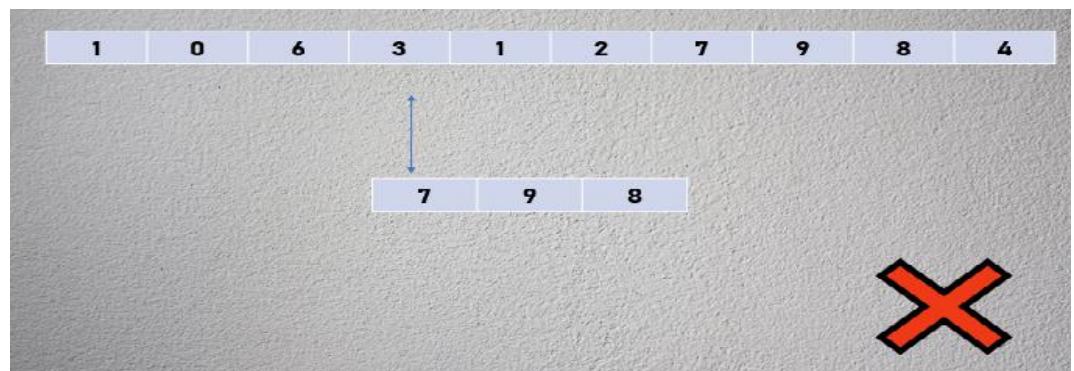
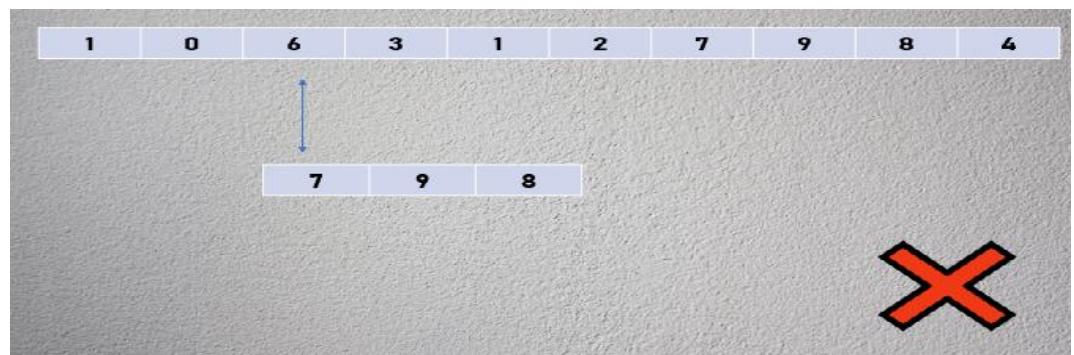
So, the answer here is to above questions are: yes, the pattern 'abaa' exists in the text. The patterns found at 4, 10.

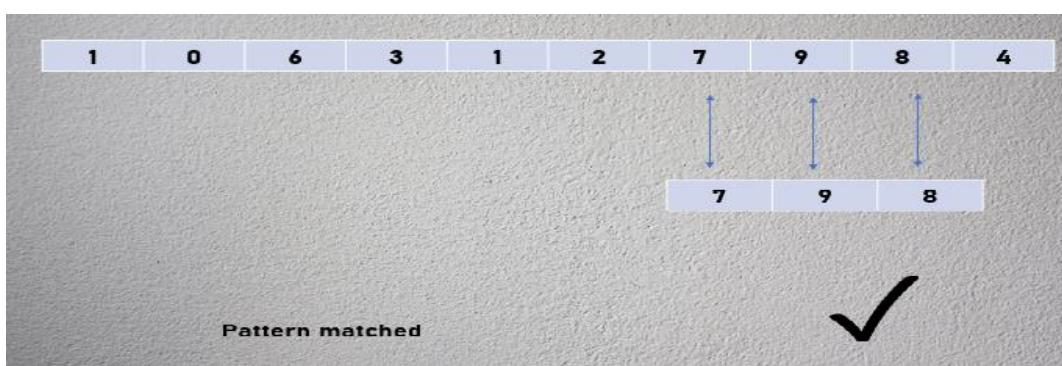
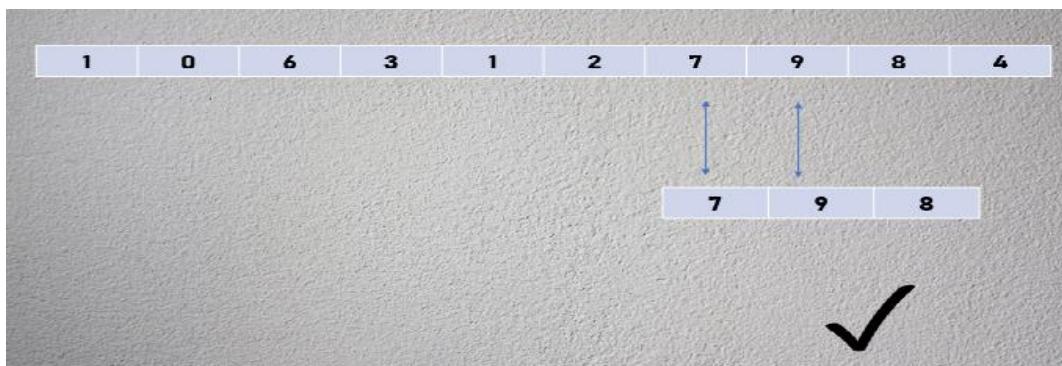
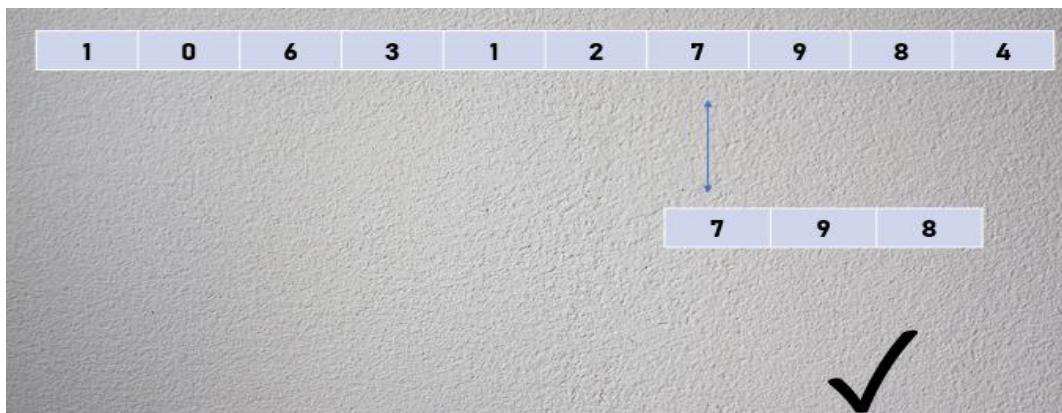
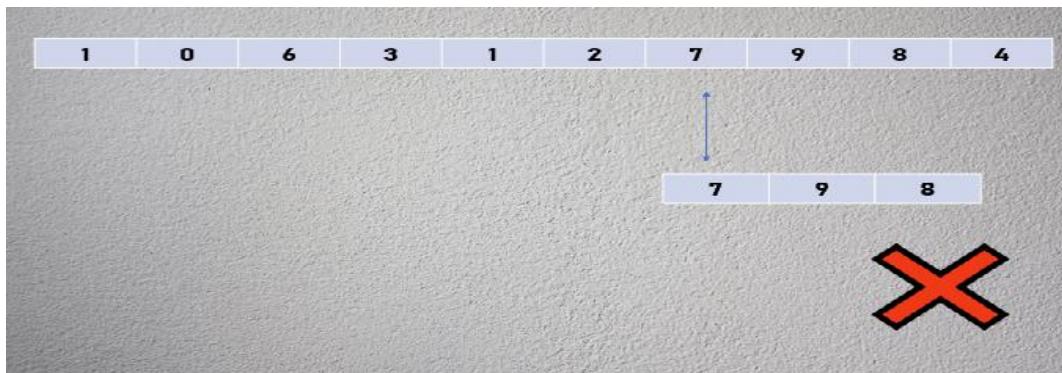


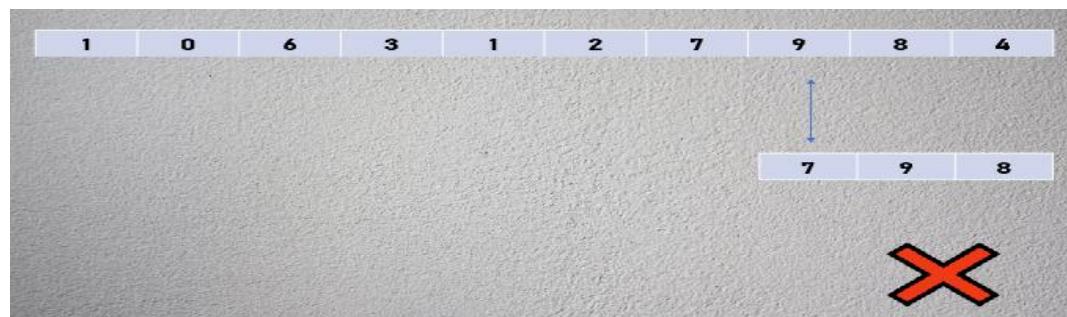
Example:

Given a text and pattern, we need to check whether the pattern exists in the text or not.



Algorithm Analysis and Design

Unit 08: Pattern Matching



Applications

So, the basic phenomenon is quite simple. These pattern matching algorithms are very simple. Though it has various applications. These are: plagiarism detection, bioinformatics and DNA sequencing, digital forensic, spelling checker, spam filters, search engines and intrusion detection system.

8.1 Various Algorithms

There are various algorithms which are used to match the pattern in the text. These are: Brute force algorithm, Robin-Karp string matching algorithm and Knuth-Morris-Pratt algorithm

8.2 Brute Force

Given a text and a pattern, we need to find out the starting index in the text where the pattern matches. It is a linear searching algorithm. It follows exhaustive search. The procedure for the same is:





In the given example, we need to find the length (T): LT = 10, length (P): LP = 4. From this, we need to find Max = $10 - 4 + 1 = 7$ (LT - LP + 1). The tracing table will be formed as:

i	1	2	3	4	For j = 1 to LP
1	1	2	3	4	
2	2	3	4	5	
3	3	4	5	6	
4	4	5	6	7	
5	5	6	7	8	
6	6	7	8	9	
7	7	8	9	10	

For $i = 1$ to max

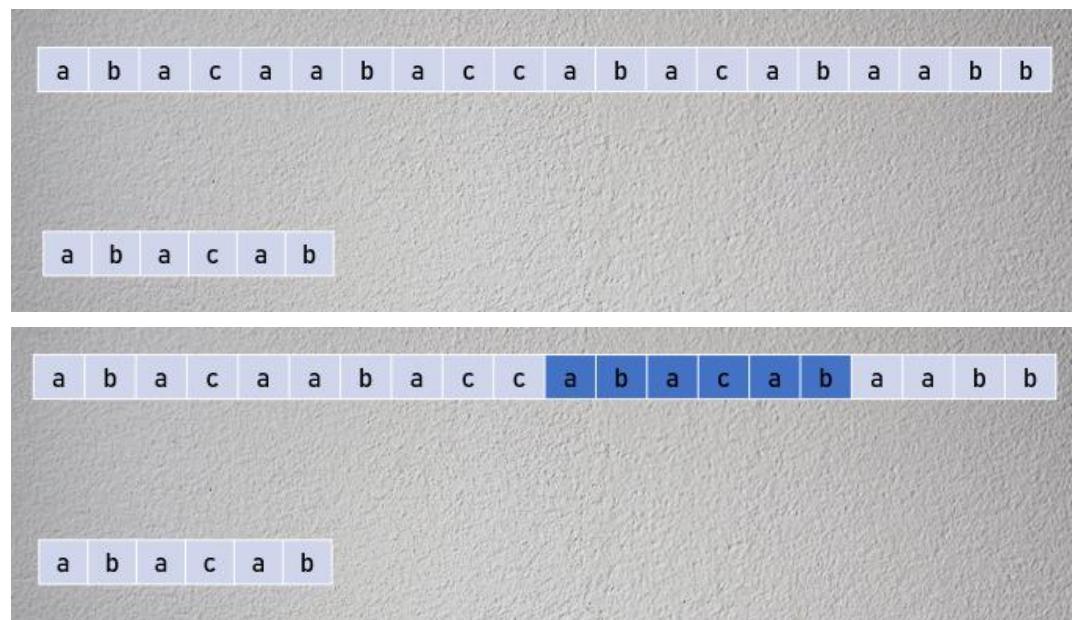
j^{th} character of the pattern will be compared with the $(j+i-1)^{\text{th}}$ character of the main string.

Algorithm

```

algorithm pattern_matching_bf(s,p): return int
{
    LT=length(T)
    LP=length (P)
    max = LT-LP+1
    for (i=1;i<=max;i++)
    {
        flag = true
        for (j=1, j<=LP&& flag == true; j++)
        {
            if (p [j] ≠ s[j+i-1])
            {
                flag = false
            }
        }
        if (flag == true)
        {
            return i;
        }
    }
    return 0
}

```



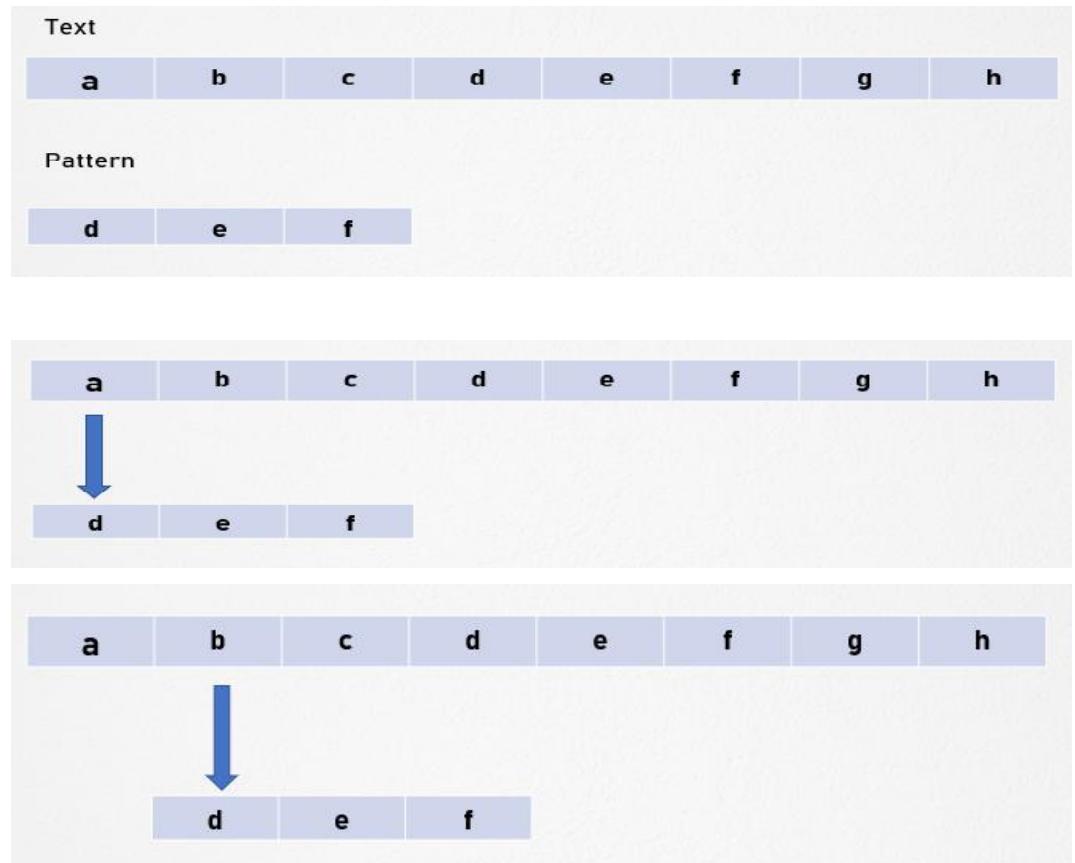
Analysis

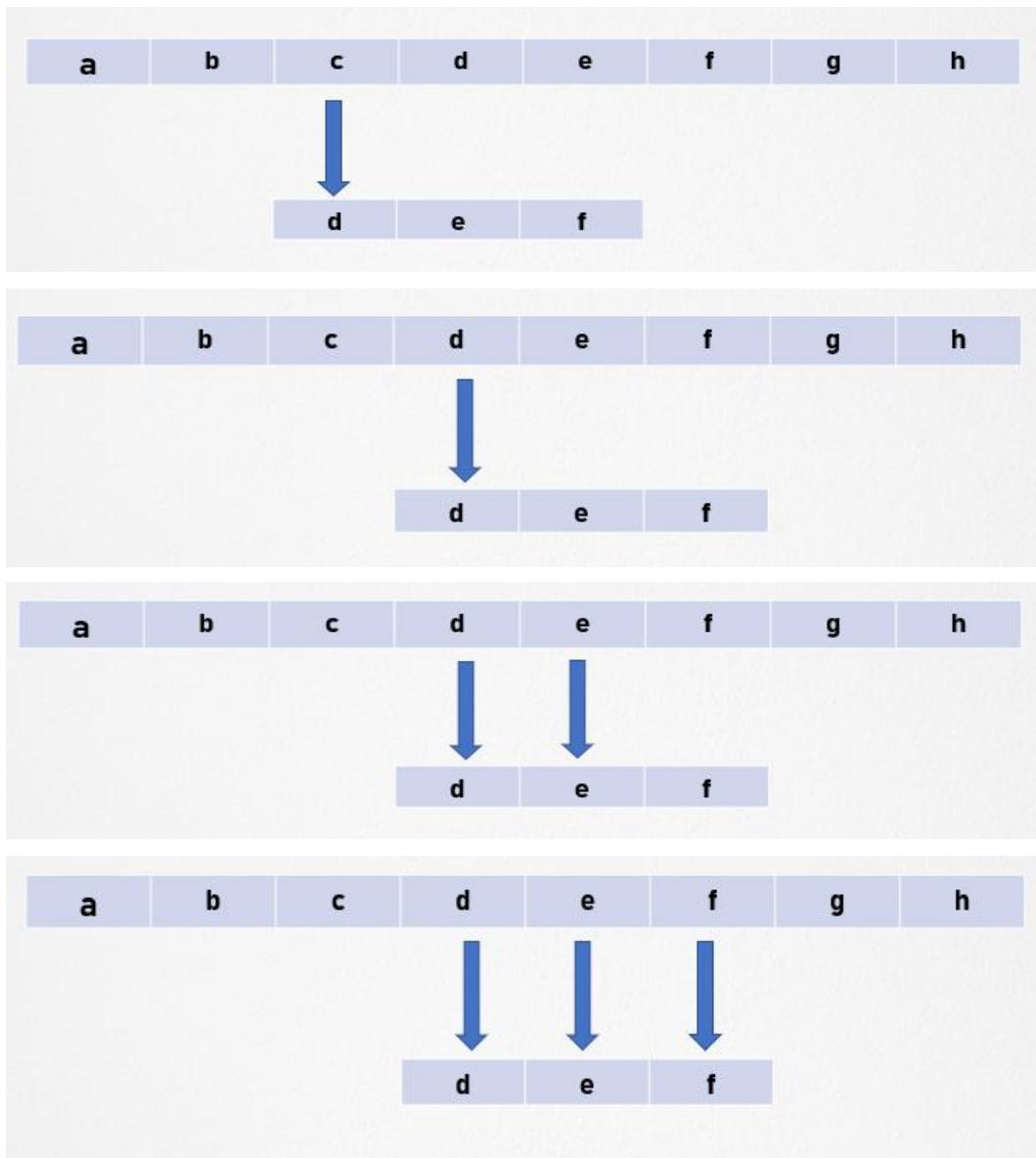
The maximum number of times, it can run is $n*m$ times. The complexity will be $O(nm)$.



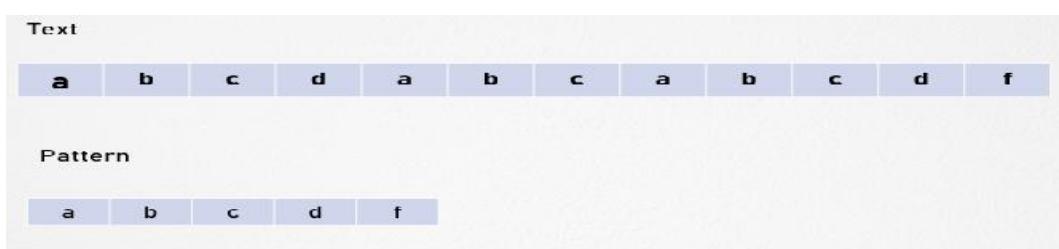
Example 1

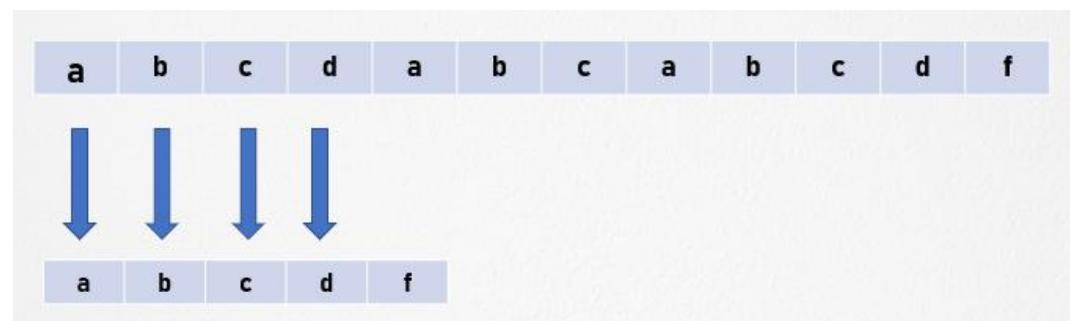
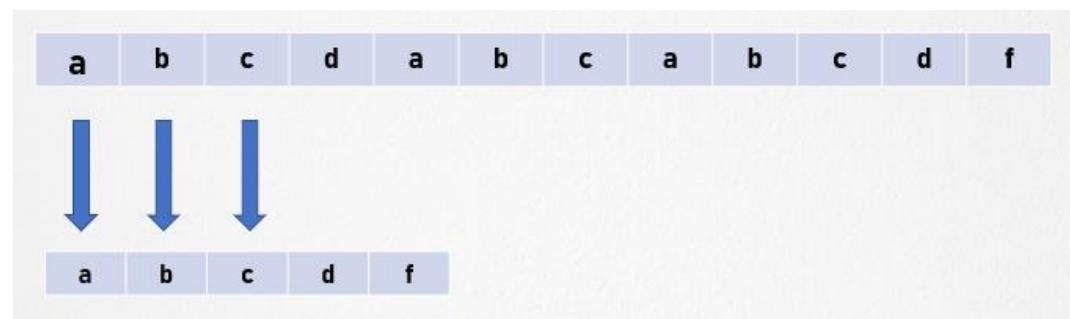
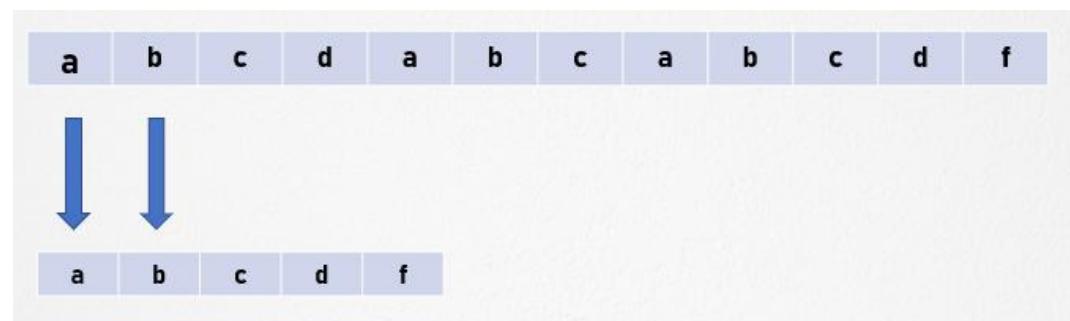
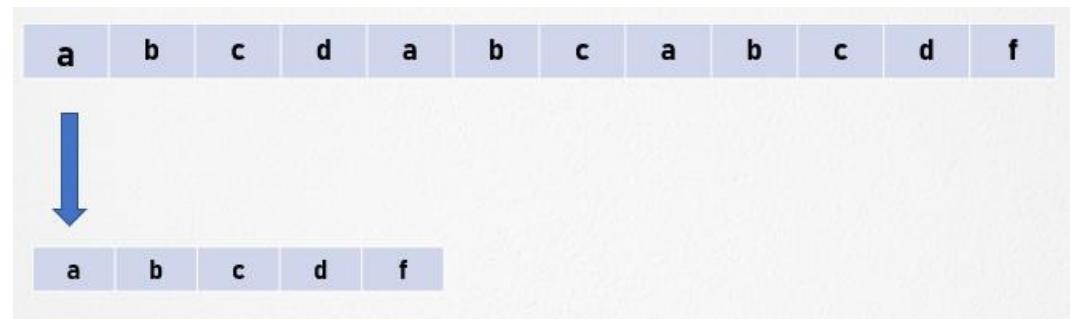
Given a text 'abcdefg' and pattern as 'def'. We need to find the existence of pattern in text.



Unit 08: Pattern Matching

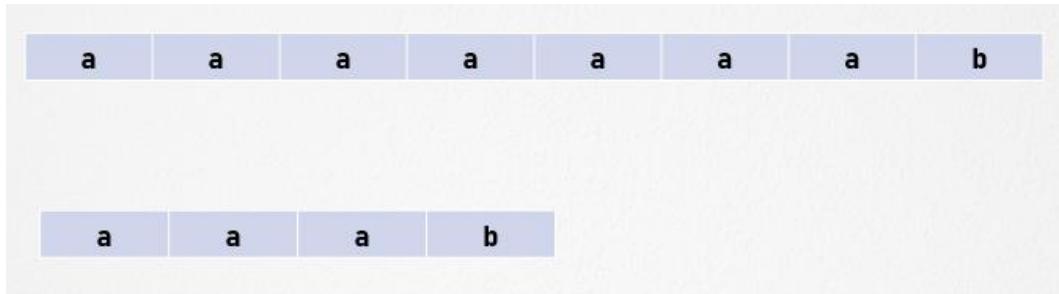
Example 2





There is a mismatch at 5th position, again j has to come back and restart.

In the previous example, i.e., example 2, we have seen the worst case of Brute Force Algorithm. It fails at the last point. So for that we need to restart.



So, in worst case, it takes $O(m*n)$ time. The solution for this is it will be better if we study the pattern before starting matching. So, based upon the experience, we have seen various drawbacks of brute force algorithm. These are: extra work needs to be done, does not study the pattern and it works blindly.

8.3 Knuth-Morris-Pratt (KMP) Algorithm

It is employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters. It is a linear time string matching algorithm. It is considered best algorithm. It uses the concept of prefix and suffix for generating π table. The π table is also known as LPS (Longest prefix which is same as some suffix) table. Here we will not backtrack the main string. Instead we will make π table.

1	2	3	4	5	6	7	8	9	10
a	b	c	d	a	b	e	a	b	f
0	0	0	0	1	2	0	1	2	0

1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	a	b	f	a	b	c
0	0	0	0	0	1	2	0	1	2	0

1	2	3	4	5	6	7	8	9
a	a	a	a	b	a	a	c	d
0	1	2	3	0	1	2	0	0

1	2	3	4	5	6	7	8	9	10
a	b	a	b	a	b	a	b	c	a
0	0	1	2	3	4	5	6	0	1



Example

TEXT GIVEN										
a	b	a	b	c	a	b	c	a	b	a

PATTERN GIVEN				
a	b	a	b	d

STEP -1 Take the indices of given text

TEXT	GIVEN															
1	a	b	c	a	b	c	a	b	a	b	a	b	a	b	d	
2	b	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
3	c	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
4	a	b	c	a	b	c	a	b	a	b	a	b	a	b	c	
5	b	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
6	c	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
7	a	b	c	a	b	c	a	b	a	b	a	b	a	b	c	
8	b	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
9	c	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
10	a	b	c	a	b	c	a	b	a	b	a	b	a	b	c	
11	b	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
12	a	b	c	a	b	c	a	b	a	b	a	b	a	b	c	
13	b	a	b	c	a	b	c	a	b	a	b	a	b	a	b	
14	a	b	c	a	b	c	a	b	a	b	a	b	a	b	c	
15	b	a	b	c	a	b	c	a	b	a	b	a	b	a	b	

STEP-2 Calculate the π table of pattern

a	b	a	b	d
1	2	3	4	5
a	b	a	b	d
0	0	1	2	0

a	b	a	b	d						
0		1		2		3		4		5
		a		b		a		b		d
		0		0		1		2		0

STEP-3 Take two variables i, j: Variable i for the text given, variable j for the pattern given. And i will start from 1 and j will start from 0.

STEP - 4 Compare T[i] with P[j+1]

If $T[i]$ and $P[j+1]$ match, then we will increment both i and j (Move i and j to the right). If $T[i]$ and $P[j+1]$ do not match, then we backtrack j and see in the LPS table the index of j .

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j	0	1	2	3	4	5									
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	
0	1	2	3	4	5										
a	b	a	b	d											
0	0	1	2	0											

If T[i] and P[j+1] match, then we increment i and j



Unit 08: Pattern Matching

		i													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

↓

		j													
0	1	2	3	4	5										
a	b	a	b	d											

✓

If $T[i]$ and $P[j+1]$ match, then we increment i and j

		i													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

↓

		j													
0	1	2	3	4	5										
a	b	a	b	d											

✓

If $T[i]$ and $P[j+1]$ match, then we increment i and j

		i													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

↓

		j													
0	1	2	3	4	5										
a	b	a	b	d											

✗

If $T[i]$ and $P[j+1]$ do not match, then we backtrack j and see in the LPS table the index of j .

		i													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

↗

		j													
0	1	2	3	4	5										
a	b	a	b	d											

✗

If $T[i]$ and $P[j+1]$ do not match, then we backtrack j and see in the LPS table the index of j .

Algorithm Analysis and Design

					i											
1	2	3	4	5	i	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	i	a	b	c	a	b	a	b	a	b	d	

↓

j																
0	1	2	3	4	5											
a	b	a	b	d												

If $T[i]$ and $P[j+1]$ do not match, then we backtrack j and see in the LPS table the index of j . At this time, we will increment j .



					i											
1	2	3	4	5	i	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	i	a	b	c	a	b	a	b	a	b	d	

↓

j																
0	1	2	3	4	5											
a	b	a	b	d												

If $T[i]$ and $P[j+1]$ match, then we increment i and j .



					i											
1	2	3	4	5	i	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	i	a	b	c	a	b	a	b	a	b	d	

↓

j																
0	1	2	3	4	5											
a	b	a	b	d												

If $T[i]$ and $P[j+1]$ match, then we increment i and j .



					i											
1	2	3	4	5	i	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	i	a	b	c	a	b	a	b	a	b	d	

↓

j																
0	1	2	3	4	5											
a	b	a	b	d												

If $T[i]$ and $P[j+1]$ do not match, then we backtrack j and see in the LPS table the index of j .



Unit 08: Pattern Matching

							i								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

j															
0	1	2	3	4	5										
a	b	a	b	d											
0	0	1	2	0											

If $T[i]$ and $P[j+1]$ do not match, then we backtrack j and see in the LPS table the index of j . At this point, we will increment i .

							i								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

j															
0	1	2	3	4	5										
a	b	a	b	d											
0	0	1	2	0											

If $T[i]$ and $P[j+1]$ match, then we increment i and j

							i								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

j															
0	1	2	3	4	5										
a	b	a	b	d											
0	0	1	2	0											

If $T[i]$ and $P[j+1]$ match, then we increment i and j

							i								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

j															
0	1	2	3	4	5										
a	b	a	b	d											
0	0	1	2	0											

If $T[i]$ and $P[j+1]$ match, then we increment i and j

													i			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d		
			j													
0	1	2	3	4	5											
a	b	a	b	d												
0	0	1	2	0												

If $T[i]$ and $P[j+1]$ match, then we increment i and j

													i			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d		
			j													
0	1	2	3	4	5											
a	b	a	b	d												
0	0	1	2	0												

If $T[i]$ and $P[j+1]$ do not match, then we backtrack j and see in the LPS table the index of j .

													i			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d		
			j													
0	1	2	3	4	5											
a	b	a	b	d												
0	0	1	2	0												

If $T[i]$ and $P[j+1]$ match, then we increment i and j

													i		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

			j												
0	1	2	3	4	5										
a	b	a	b	d											

If $T[i]$ and $P[j+1]$ match, then we increment i and j

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

			j												
0	1	2	3	4	5										
a	b	a	b	d											

i has reached to the end of text and the pattern is matched at location 11.

LPS Table

Step 1 - Define a one-dimensional array with the size equal to the length of the Pattern. (LPS[size])

Step 2 - Define variables i & j . Set $i = 0$, $j = 1$ and $LPS[0] = 0$.

Step 3 - Compare the characters at **Pattern[i]** and **Pattern[j]**.

Step 4 - If both are matched then set $LPS[j] = i+1$ and increment both i & j values by one. Goto to Step 3.

Step 5 - If both are not matched then check the value of variable ' i '. If it is '0' then set $LPS[j] = 0$ and increment ' j ' value by one, if it is not '0' then set $i = LPS[i-1]$. Goto Step 3.

Step 6- Repeat above steps until all the values of LPS[] are filled.

Time complexity

The time complexity of KMP is $O(n)$.

When we do search for a string in a notepad/word file, browser, or database, pattern searching algorithms are used to show the search results. Given a text $txt[0..n-1]$ and a pattern $pat[0..m-1]$ where n is the length of the text and m is the length of the pattern, write a function $search(char pat[], char txt[])$ that prints all occurrences of $pat[]$ in $txt[]$. The assumption here is $n > m$. Examples

- Input: $txt[] = "THIS IS A TEST TEXT"$; $pat[] = "TEST"$; Output: Pattern found at index 10
- Input: $txt[] = "AABAACAAADAABAABA"$; $pat[] = "AABA"$; Output: Pattern found at index 0, Pattern found at index 9 and Pattern found at index 12.

8.4 Boyer Moore Algorithm

The Boyer-Moore string-search algorithm is an efficient string-searching algorithm that is the standard benchmark for practical string-search literature. It was developed by Robert S. Boyer and J Strother Moore in 1977. The Boyer Moore algorithm preprocesses the pattern. The algorithm preprocesses the string being searched for (the pattern), but not the string being searched in (the text). The algorithm is thus well-suited for applications in which the pattern is much shorter than the text or where it persists across multiple searches.

Definition

T: denotes the input text to be searched. Its length is **n**.P: denotes the string to be searched for, called the pattern. Its length is **m**.**S[i]**: denotes the character at index i of string S, counting from 1.**S[i..j]**: denotes the substring of string S starting at index i and ending at j, inclusive. A prefix of S is a substring **S[1..i]** for some i in range [1, l], where l is the length of S. A suffix of S is a substring **S[i..l]** for some i in range [1, l], where l is the length of S. An alignment of P to T is an index k in T such that the last character of P is aligned with index k of T. A match or occurrence of P occurs at an alignment k if P is equivalent to **T[(k-m+1)..k]**. The Boyer-Moore algorithm searches for occurrences of P in T by performing explicit character comparisons at different alignments. Instead of a brute-force search of all alignments (of which there are $m-n+1$), Boyer-Moore uses information gained by preprocessing P to skip as many alignments as possible. If the end of the pattern is compared to the text, then jumps along the text can be made rather than checking every character of the text. The reason that this works is that in lining up the pattern against the text, the last character of the pattern is compared to the character in the text. If the characters do not match, there is no need to continue searching backwards along the text. If the character in the text does not match any of the characters in the pattern, then the next character in the text to check is located n characters farther along the text, where n is the length of the pattern. If the character in the text is in the pattern, then a partial shift of the pattern along the text is done to line up along the matching character and the process is repeated. Jumps are done along the text to make comparisons rather than checking every character in the text decreases the number of comparisons that have to be made, which is the key to the efficiency of the algorithm. More formally, the algorithm begins at alignment $k=n$, so the start of P is aligned with the start of T. The characters in P and T are then compared starting at index n in P and k in T, moving backward. The strings are matched from the end of P to the start of P. The comparisons continue until either the beginning of P is reached (which means there is a match) or a mismatch occurs upon which the alignment is shifted forward (to the right) according to the maximum value permitted by a number of rules. The comparisons are performed again at the new alignment, and the process repeats until the alignment is shifted past the end of T, which means no further matches will be found.

Combination of two approaches

A shift is calculated by applying two rules: the bad character rule and the good suffix rule. The actual shifting offset is the maximum of the shifts calculated by these rules. Boyer Moore is a combination of the following two approaches. 1) Bad Character Heuristic 2) Good Suffix Heuristic

Both above heuristics can also be used independently to search a pattern in a text.

Previous algorithms

If we look at the Naïve algorithm (Brute Force algorithm), it slides the pattern over the text one by one. KMP algorithm does preprocessing over the pattern so that the pattern can be shifted by more than one. The Boyer Moore algorithm does preprocess for the same reason.

Pre-processing

It processes the pattern and creates different arrays for each of the two heuristics. At every step, it slides the pattern by the max of the slides suggested by each of the two heuristics. So it uses **greatest offset** suggested by the two heuristics at every step. Unlike the previous pattern searching algorithms, the Boyer Moore algorithm starts matching from the last character of the pattern.

Bad Character Heuristic

The idea of bad character heuristic is simple. The character of the text which doesn't match with the current character of the pattern is called the **Bad Character**. The bad-character rule considers the

Unit 08: Pattern Matching

character in T at which the comparison process failed (assuming such a failure occurred). The next occurrence of that character to the left in P is found, and a shift which brings that occurrence in line with the mismatched occurrence in T is proposed. If the mismatched character does not occur to the left in P , a shift is proposed that moves the entirety of P past the point of mismatch.

Example

- - - - X - - K - - -
- A N P A N M A N A M -
- - N N A A M A N - - -
- - - - N N A A M A N -

Good Suffix Heuristic

The good suffix rule is markedly more complex in both concept and implementation than the bad character rule. Like the bad character rule, it also exploits the algorithm's feature of comparisons beginning at the end of the pattern and proceeding towards the pattern's start. Suppose for a given alignment of P and T , a substring t of T matches a suffix of P , but a mismatch occurs at the next comparison to the left. Then find, if it exists, the right-most copy t' of t in P such that t' is not a suffix of P and the character to the left of t' in P differs from the character to the left of t in P . Shift P to the right so that substring t' in P aligns with substring t in T . If t' does not exist, then shift the left end of P past the left end of t in T by the least amount so that a prefix of the shifted pattern matches a suffix of t in T . If no such shift is possible, then shift P by n places to the right. If an occurrence of P is found, then shift P by the least amount so that a proper prefix of the shifted P matches a suffix of the occurrence of P in T . If no such shift is possible, then shift P by n places, that is, shift P past t .

Example

- - - - X - - K - - - -
- M A N P A N A M A N A P -
- A N A M P N A M - - - -
- - - - A N A M P N A M -

The formula for constructing bad match table is **values = Length of pattern index - 1**



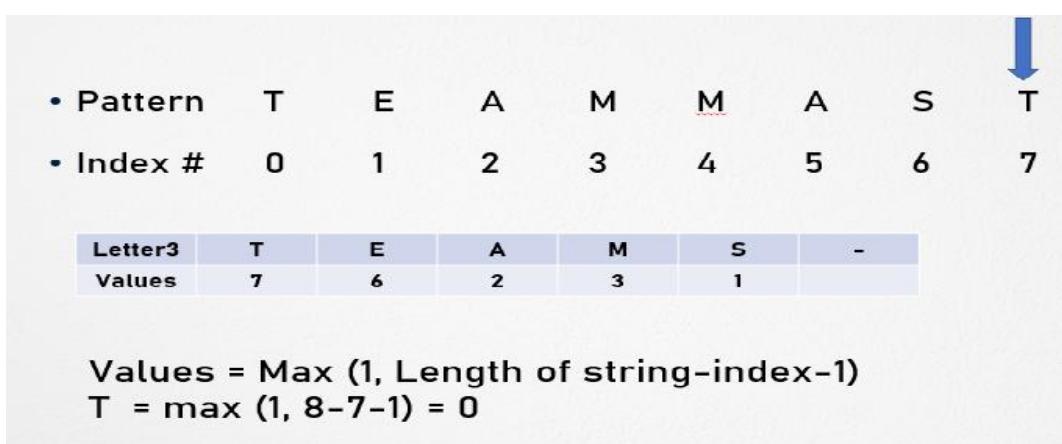
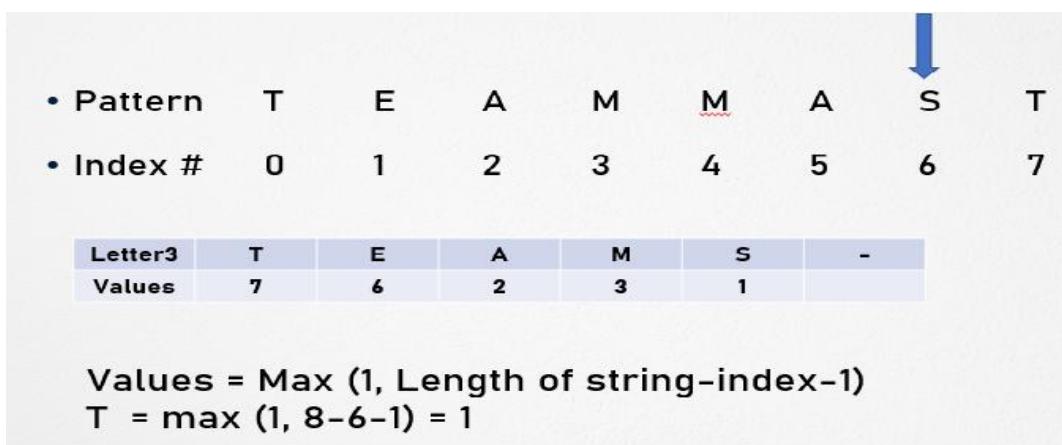
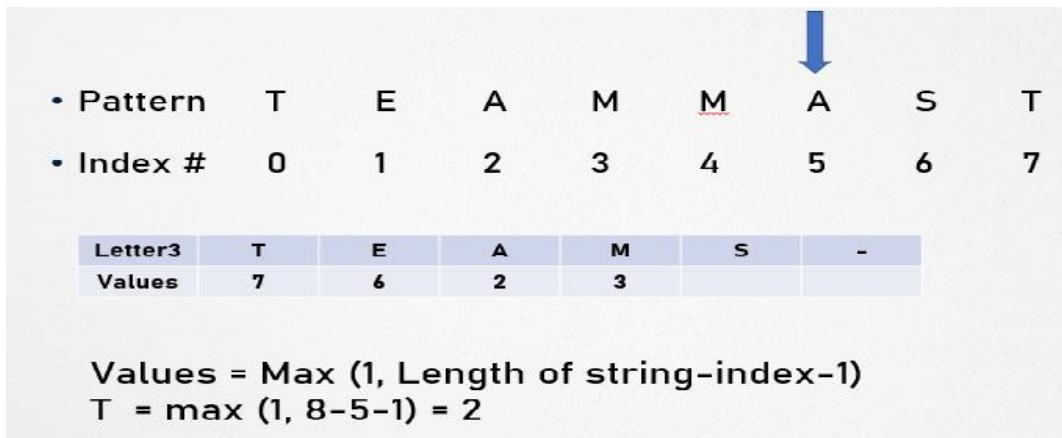
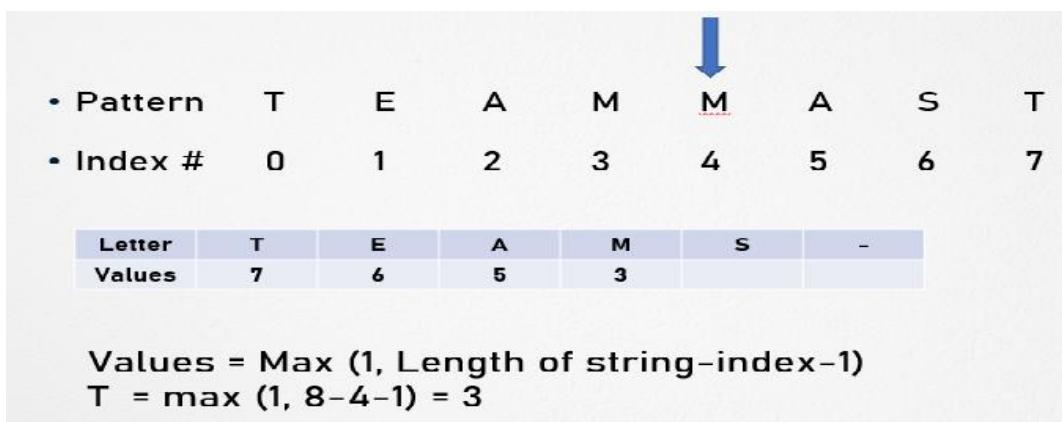
Example by Bad Match heuristic

- Text: "WELCOMETOTEAMMAST"
- Pattern: 'TEAMMAST'
- Pattern T E A M M A S T
- Index # 0 1 2 3 4 5 6 7

Letter	T	E	A	M	M	A	S	T
Values							-	

Algorithm Analysis and Design

<ul style="list-style-type: none"> • Pattern T E A M M A S T • Index # 0 1 2 3 4 5 6 7 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Letter</th><th>T</th><th>E</th><th>A</th><th>M</th><th>S</th><th>-</th></tr> </thead> <tbody> <tr> <td>Values</td><td>7</td><td></td><td></td><td></td><td></td><td></td></tr> </tbody> </table>	Letter	T	E	A	M	S	-	Values	7												
Letter	T	E	A	M	S	-															
Values	7																				
Values = Max (1, Length of string-index-1) $T = \max (1, 8-0-1) = 7$																					
<ul style="list-style-type: none"> • Pattern T E A M M A S T • Index # 0 1 2 3 4 5 6 7 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Letter</th><th>T</th><th>E</th><th>A</th><th>M</th><th>S</th><th>-</th></tr> </thead> <tbody> <tr> <td>Values</td><td>7</td><td>6</td><td></td><td></td><td></td><td></td></tr> </tbody> </table>	Letter	T	E	A	M	S	-	Values	7	6											
Letter	T	E	A	M	S	-															
Values	7	6																			
Values = Max (1, Length of string-index-1) $T = \max (1, 8-1-1) = 6$																					
<ul style="list-style-type: none"> • Pattern T E A M M A S T • Index # 0 1 2 3 4 5 6 7 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Letter</th><th>T</th><th>E</th><th>A</th><th>M</th><th>S</th><th>-</th></tr> </thead> <tbody> <tr> <td>Values</td><td>7</td><td>6</td><td>5</td><td></td><td></td><td></td></tr> </tbody> </table>	Letter	T	E	A	M	S	-	Values	7	6	5										
Letter	T	E	A	M	S	-															
Values	7	6	5																		
Values = Max (1, Length of string-index-1) $T = \max (1, 8-2-1) = 5$																					
<ul style="list-style-type: none"> • Pattern T E A M M A S T • Index # 0 1 2 3 4 5 6 7 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Letter</th><th>T</th><th>E</th><th>A</th><th>M</th><th>S</th><th>-</th></tr> </thead> <tbody> <tr> <td>Values</td><td>7</td><td>6</td><td>5</td><td>4</td><td></td><td></td></tr> </tbody> </table>	Letter	T	E	A	M	S	-	Values	7	6	5	4									
Letter	T	E	A	M	S	-															
Values	7	6	5	4																	
Values = Max (1, Length of string-index-1) $T = \max (1, 8-3-1) = 4$																					

Unit 08: Pattern Matching

The Boyer-Moore algorithm uses information gathered during the preprocess step to skip sections of the text, resulting in a lower constant factor than many other string search algorithms. In general,

Algorithm Analysis and Design

the algorithm runs faster as the pattern length increases. The key features of the algorithm are to match on the tail of the pattern rather than the head, and to skip along the text in jumps of multiple characters rather than searching every single character in the text. The worst-case running time of $O(n+m)$ only if the pattern does not appear in the text.

Variants - 1

The Boyer-Moore-Horspool algorithm is a simplification of the Boyer-Moore algorithm using only the bad character rule.

Variants-2

The Apostolico-Giancarlo algorithm speeds up the process of checking whether a match has occurred at the given alignment by skipping explicit character comparisons. This uses information gleaned during the pre-processing of the pattern in conjunction with suffix match lengths recorded at each match attempt. Storing suffix match lengths requires an additional table equal in size to the text being searched.

Variants-3

The Raita algorithm improves the performance of Boyer-Moore-Horspool algorithm. The searching pattern of particular sub-string in a given string is different from Boyer-Moore-Horspool algorithm.

Boyer-Moore algorithm is extremely fast on large alphabet (relative to the length of the pattern). The payoff is not as for binary strings or for very short patterns. For binary strings Knuth-Morris-Pratt algorithm is recommended. For the very shortest patterns, the naïve algorithm may be better.

Summary

- String or pattern matching algorithm try to find a place where one or several string (called a pattern) are found within a text.
- There are various algorithms for pattern matching are brute force algorithm, robin-karp string matching algorithm and knuth-morris-pratt algorithm.
- There are various applications of pattern matching algorithm are: plagiarism detection, bioinformatics and DNA sequencing, digital forensic, spelling checker, spam filters, search engines and intrusion detection system
- The drawbacks of brute force algorithm are it works blindly, it does not study pattern and some extra work needs to be done in this.
- Boyer Moore algorithm preprocesses the pattern. The algorithm preprocesses the string being searched for (the pattern), but not the string being searched in (the text).
- Boyer Moore is a combination of the following two approaches. Bad Character Heuristic and Good Suffix Heuristic.

Keywords

- **Pattern matching algorithm:** String or pattern matching algorithm try to find a place where one or several string (called a pattern) are found within a text.
- **Brute-force matching algorithm:** It is a linear searching algorithm. It follows exhaustive search.
- **Knuth-Morris-Pratt algorithm:** It is employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.
- **Boyer-Moore string-search algorithm:** It is an efficient string-searching algorithm that is the standard benchmark for practical string-search literature.
- **Apostolico-Giancarlo algorithm:** The Apostolico-Giancarlo algorithm speeds up the process of checking whether a match has occurred at the given alignment by skipping explicit character comparisons.

Unit 08: Pattern Matching

- **Raita algorithm:** The Raita algorithm improves the performance of Boyer-Moore-Horspool algorithm. The searching pattern of particular sub-string in a given string is different from Boyer-Moore-Horspool algorithm.

Self Assessment

1. _____ is the process of recognizing patterns by using machine learning algorithm.
 - Processed data
 - Literate statistical programming
 - Pattern recognition
 - Likelihood
2. jth character of the pattern will be compared with _____ character of main string.
 - ith
 - j-i th
 - j-i+1 th
 - j+i-1 th
3. How many index variables are required for Knuth-Morris-Pratt algorithm?
 - 1
 - 2
 - 3
 - 4
4. What is the worst-case performance of Boyer Moore algorithm?
 - $O(n)$
 - $O(m)$
 - $O(n+m)$
 - $O(nm)$
5. Which of these algorithms is used for pattern matching?
 - Brute force algorithm
 - Robin-Karp algorithm
 - Knuth Morris algorithm
 - All of the above
6. The complexity of brute force pattern matching algorithm is
 - $O(n)$
 - $O(m)$
 - $O(nm)$
 - None of the above
7. What is the time complexity of Knuth-Morris-Pratt algorithm?
 - $O(1)$

- B. $O(n)$
 - C. $O(n^2)$
 - D. $O(n^3)$
8. What is the key feature of Boyer Moore algorithm?
- A. Matching on head of pattern
 - B. Matching on tail of pattern
 - C. Matching on median of pattern
 - D. None of the above
9. In which applications, the pattern matching algorithm is applied?
- A. Digital forensic
 - B. Search engines
 - C. Plagiarism detection
 - D. All of the above
10. Brute force technique of pattern matching is a kind of
- A. Linear searching algorithm
 - B. Dynamic searching algorithm
 - C. Non-linear searching algorithm
 - D. None of the above
11. Which concept is used in Knuth-Morris-Pratt algorithm?
- A. Prefix
 - B. Suffix
 - C. π table
 - D. All of the above
12. What is the formula for constructing bad mach table?
- A. Value = length of pattern index+1
 - B. Value = length of pattern index-1
 - C. Value = length of pattern index*1
 - D. Value = length of pattern index/1
13. Given a text array [1...n] and a pattern array [1...m], what should be the condition for pattern matching?
- A. Only $m < n$
 - B. Only $m = n$
 - C. Either of the above
 - D. None of the above
14. Which of these is a drawback of Brute force pattern matching algorithm?
- A. Extra work needs to be done
 - B. Does not study the pattern
 - C. It works blindly

Unit 08: Pattern Matching

- D. All of the above
15. The Boyer Moore implements
 A. Bad Character Heuristic
 B. Good Suffix Heuristic
 C. Both of the above
 D. None of the above

Answers for Self Assessment

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

Review Questions

1. What is pattern matching algorithm? Write the basic phenomenon behind that.
2. What are different algorithms for pattern matching? What is its element space?
3. What is brute-force matching algorithm? Explain its drawbacks.
4. Explain the worst case of brute-force algorithm using two examples.
5. What is Knuth Morris Pratt algorithm? Why it is more preferred from brute force algorithm?
6. Write the steps in Knuth Morris Pratt algorithm.
7. Write the steps for creating LPS table.
8. What is Boyer Moore algorithm? Write its definition.
9. What are the key insights of Boyer Moore algorithm? Write its algorithm.
10. Explain bad match heuristic and good suffix heuristic using examples.

**Further Readings**

- <https://teachics.org/data-structures/pattern-matching-algorithms/>
- <https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/>
- <https://www.javatpoint.com/daa-boyer-moore-algorithm>

Unit 09: Huffman Encoding

CONTENTS

- Objectives
- Introduction
- 9.1 Applications of Encoding
- 9.2 Example - 1: ASCII Encoding
- 9.3 Example - 2
- 9.4 Example - 3
- 9.5 Algorithm
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the ASCII encoding
- Understand the variable size encoding
- Understand the Huffman encoding
- Understand how to calculate the cost of encoding
- Understand how to calculate the cost of decoding.

Introduction

Data transmission is the transfer of data from one digital device to another. This transfer occurs via point-to-point data streams. High data transfer rates are essential for any business. Because of the rise in mobile usage, social media, and a variety of sensors, the amount of data used annually has expanded by as much as 40 percent year over year, according to industry researchers. More than ever, high-speed data transmission infrastructure is needed by businesses in every industry to handle the ever-increasing volume of content transferred from one point to the next. Businesses are bombarded with large volumes of data every day, with increasing complexity. Content delivery networks have implemented new and improved technologies to increase data transmission rates with protocols in place to protect the original quality of the data. Data encryption and origin IP masking protect data from both known and emerging threats. Encoding is the process of converting the data or a given sequence of characters, symbols, alphabets etc., into a specified format, for the secured transmission of data. Decoding is the reverse process of encoding which is to extract the information from the converted format.

Data is technically a series of electrical charges arranged in patterns to represent information. Before data in an electronic file can be decoded and displayed on a computer screen, it must be encoded. There are various techniques available for encoding. These are used for reducing the size of data and message. One of the most used approach of encoding, i.e., Huffman encoding follows the greedy technique. Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code, and the least frequent character gets the largest code.

9.1 Applications of Encoding

- If you want to store the data in a file, you can store the data in compressed form to reduce the size of data.
- If the file is to be sent over a network, then the size of that file can be reduced by applying the data compression technique to reduce the cost of transmission.

Given a message, we can find out the cost of the message for storing/ transmission. First calculate the length of the message. This message is sent after converting into ASCII codes. The range of ASCII codes is 0-127. ASCII codes are of 8 bits. The alphabet will get converted to ASCII code and the code is converted to binary form. The ASCII Codes are given in the below table:

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	*	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 2]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B]	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	1	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	-
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

9.2 Example - 1:ASCII Encoding

B	C	C	A	B	B	D	D	A	E	C	C	B	B	A	E	D	D	C	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The characters given in the message are: A, B, C, D, and E. For each alphabet we have 8 bits in ASCII code. The message length is 20 bits. So, for this message, the total length will be $20 * 8 = 160$ bit. Therefore, the cost of sending the message with ASCII encoding technique is 160 bits.

The use of 160 bits for the message of length 20 is too high. So, we need to find the solution for this which uses less bits for encoding. As we can see, only few alphabets are used in the message, it is not necessary to use all 8 bits for a single character. The solution for the same is to use one of the given below:

- Fixed size coding
- Variable size coding

Fixed Size Coding

In fixed size encoding, we assign a fixed size code to each alphabet present in the code and the size is dependent upon the number of alphabets used in the message. As in the example given, we just have 5 alphabets.

B	C	C	A	B	B	D	D	A	E	C	C	B	B	A	E	D	D	C	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

We need to find the code for the alphabets, i.e., A, B, C, D and E.

Character	Count	Code
A	3	
B	5	
C	6	
D	4	
E	2	
	20	

We need the code for only five alphabets. If we take one bit, that can represent only two states, i.e., 0 and 1. If we take two bits, that can represent only four states, i.e., 00, 01, 10 and 11. If we take three bits, that can represent total eight states, i.e., 000, 001, 010, 011, 100, 101, 110 and 111. From this, we will choose any 5 combinations.

Character	Count	Code
A	3	000
B	5	001
C	6	010
D	4	011
E	2	100
	20	

Algorithm Design and Analysis

There are 20 characters, and each take 3 bits, so the message takes $20 * 3 = 60$ bits. When the message is sent, the codes should also be sent for the understanding of message at the receiver's side.

- For alphabets, cost of sending will be: $5 * 8 = 40$ bits
- For codes, cost of sending will be: $5 * 3 = 15$ bits
- So, the total cost (number of bits) is: $60 + 40 + 15 = 115$ bits.

The difference of cost in using these two techniques are: With ASCII encoding technique, the cost (numbers of bits) for the message was 160 bits. With fixed size coding technique, the cost (numbers of bits) for the message was 115 bits.

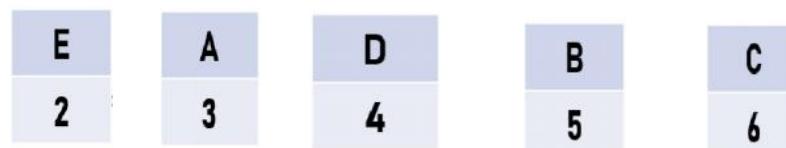
Variable size coding (Huffman Coding)

Huffman says that there is no need to take fixed size code, some characters may be appearing for few numbers of times and some may be appearing for more number of times. If we provide the smaller code to the more appearing characters, then the size of the entire message will be reduced. Huffman code follows optimal merge pattern. Given message is:

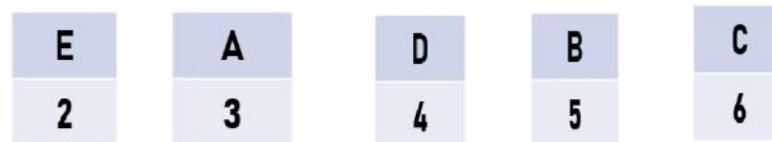
B	C	C	A	B	B	D	D	A	E	C	C	B	B	A	E	D	D	C	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Character	Count	Code
A	3	
B	5	
C	6	
D	4	
E	2	
	20	

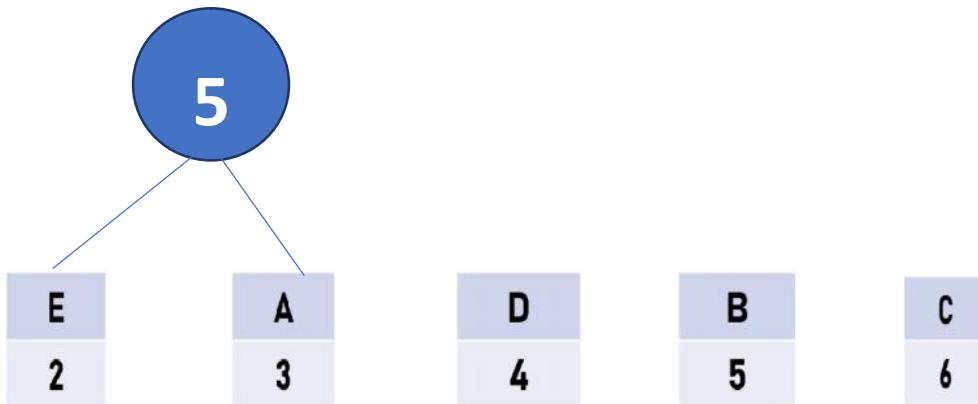
The alphabets E, A, D, B and C are used 2, 3, 4, 5 and 6 number of times.



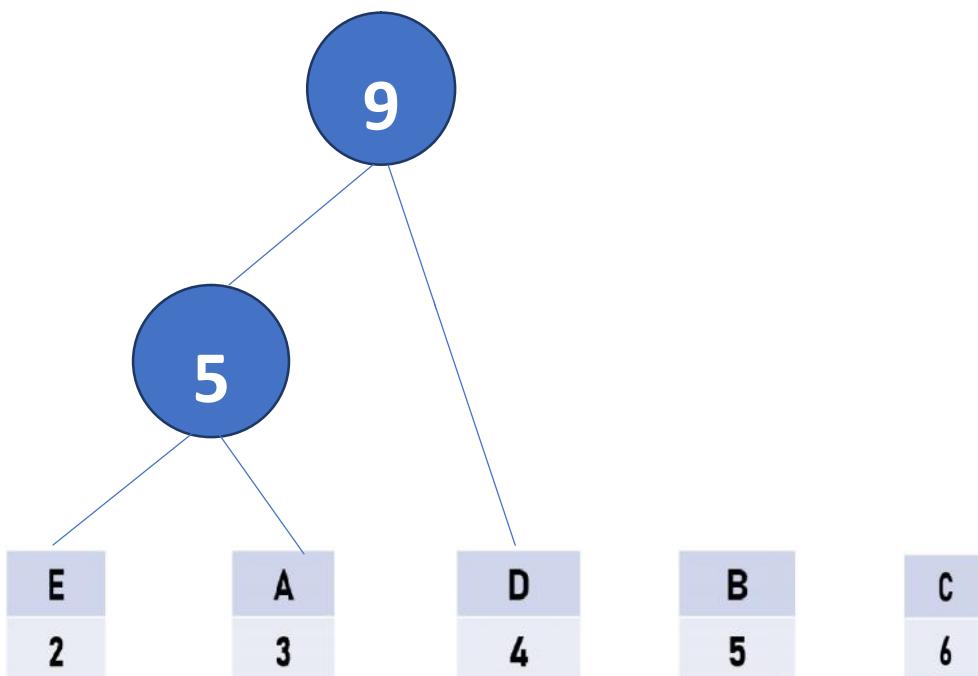
Arrange the alphabets in the increasing order of their count.



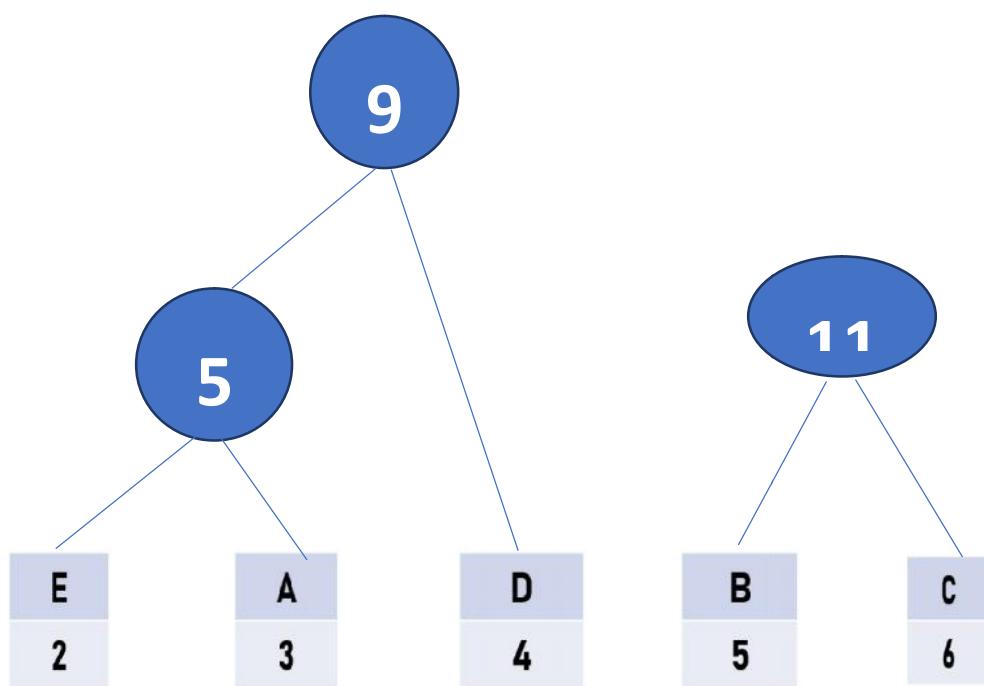
STEP - 2: Merge two smallest one.



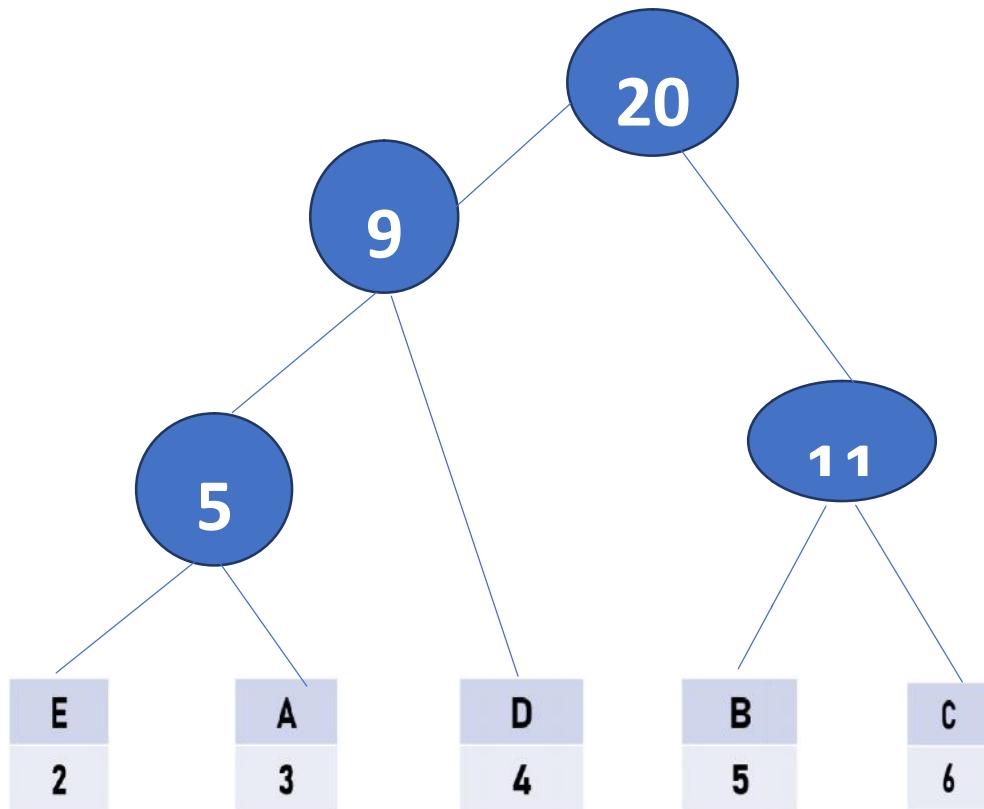
Follow the same thing.



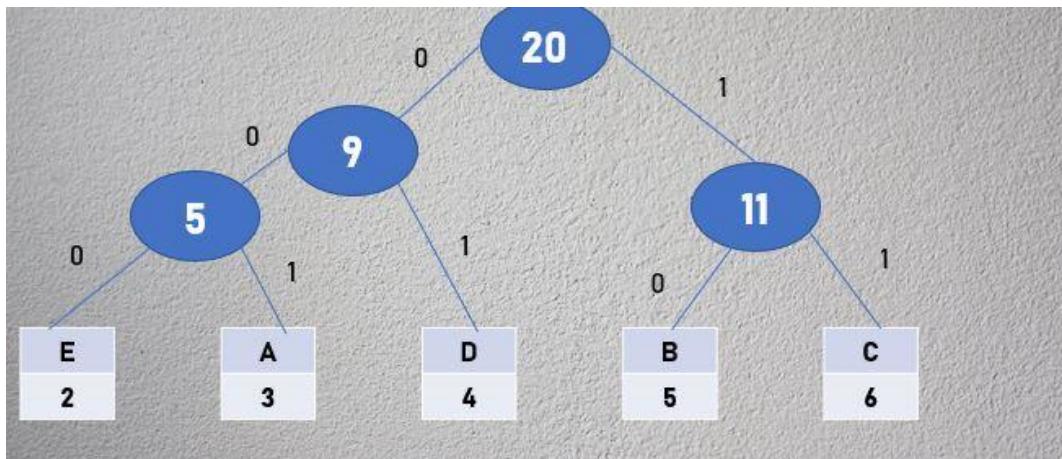
Follow the same thing.



Follow the same thing



STEP - 3: Define the codes, i.e., mark the left-hand size edges as 0 and right-hand size edges as 1.

Unit 09: Huffman Encoding**STEP - 4: Get the codes for each character**

Starting from the root, reaching till the bottom. We will get the code for each alphabet

A = 001, B = 10, C = 11, D = 01, E = 000. Put these values in the table

Character	Count	Code
A	3	001
B	5	10
C	6	11
D	4	01
E	2	000
	20	

STEP 5: Use these codes to encode the message.

Encoded message: 101111001101001010010001111101000100001011111

The Cost after encoding: A = 001, B = 10, C = 11, D = 01, E = 000. The encoded message size is = 45 bits. The difference in cost are:

ASCII Codes = $5 * 8$ = 40 bits

Variable size Codes = 12 bits

Algorithm Design and Analysis

Total number of bits = $45 + 40 + 12 = 97$ bits

The decoding at the receiver's side is done.

The encoded message is: 101111001101001010010001111101000100001011111

A = 001, B = 10, C = 11, D = 01, E = 000

B	C	C	A	B	B	D	D	A	E	C	C	B	B	A	E	D	D	C	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The Difference between three methods, the cost

- With ASCII encoding = 160 bits
- With fixed size encoding = 115 bits
- With variable size encoding = 97 bits

From this example, it is clearly shown that the Huffman encoding or variable size encoding really helps in reducing the size of message.

9.3 Example - 2

Given a message which is composed of 100 characters. The count is a = 50, b = 10, c = 30, d = 5, e = 3 and f = 2. Encode the message using fixed size and variable size coding. Show the difference between the size of message in both encodings.

Given data: a = 50, b = 10, c = 30, d = 5, e = 3, f = 2; total number of characters in the messages = 100.

ASCII Encoding

In the given message a = 50, b = 10, c = 30, d = 5, e = 3, f = 2. Total number of characters in the message = 100. The ASCII code is of 8 bits. So, the total number of bits after ASCII encoding is = $8 * 100 = 800$.

Fixed Size Encoding

A = 50, B = 10, C = 30, D = 5, E = 3, F = 2. There are 6 symbols in the messages. So, for representation of 6 symbols, we require 3 bits. Out of 000, 001, 010, 011, 100, 101, 110 and 111, we will choose 6 combinations.

Symbol	Count	Code
A	50	000
B	10	001
C	30	010
D	5	011

Unit 09: Huffman Encoding

E	3	100
F	2	101
	100	

$$\text{Message size after fixed size encoding: } = 3*50 + 10*3 + 3*30 + 5*3 + 3*3 + 2*3$$

$$= 150 + 30 + 90 + 15 + 9 + 6$$

$$= 300$$

$$\text{Cost of alphabets} = 6*8 = 48$$

$$\text{Cost of codes} = 3*16 = 48$$

$$\text{Total Cost after fixed size encoding is} = 300 + 48 + 48 = 396.$$

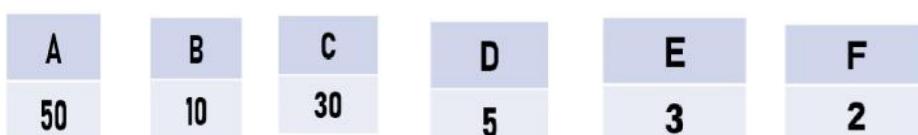
Variable Size Encoding

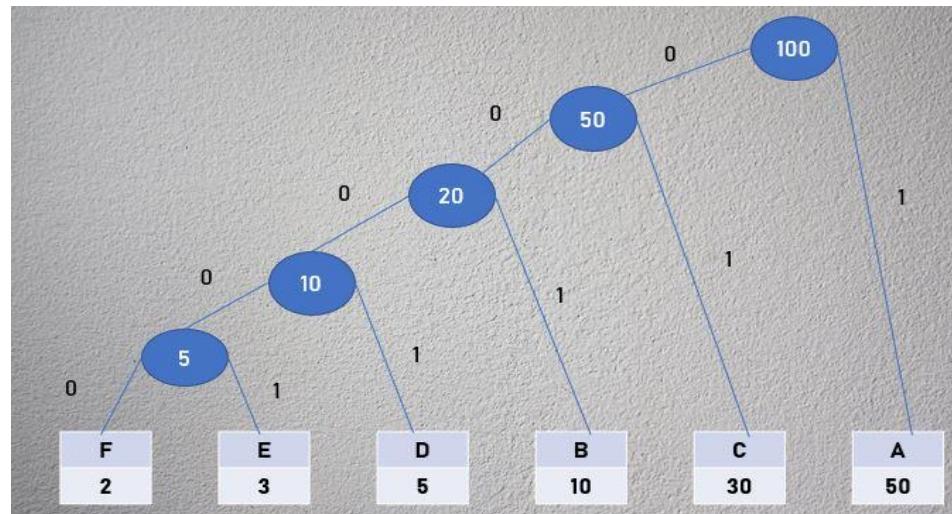
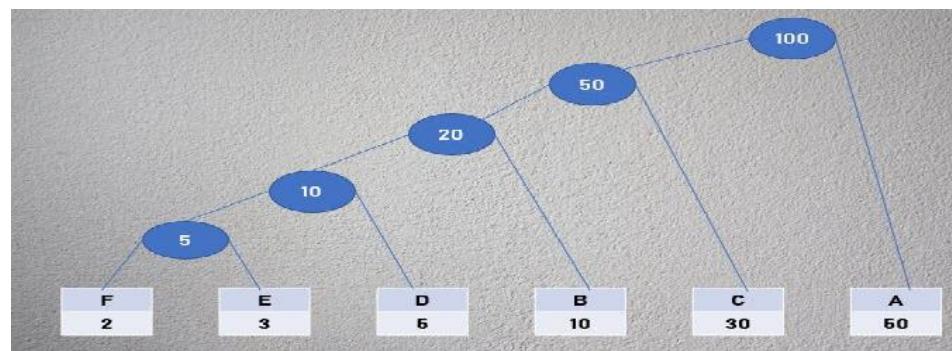
- A = 50, B = 10, C = 30, D = 5, E = 3, F = 2.

Symbol	Count
A	50
B	10
C	30
D	5
E	3
F	2

- A = 50, B = 10, C = 30, D = 5, E = 3, F = 2.

Follow the procedure discussed in first example:





- $A = 50, B = 10, C = 30, D = 5, E = 3, F = 2$.

Symbol	Count	Codes
A	50	1
B	10	001
C	30	01
D	5	0001
E	3	00001
F	2	00000

The associated cost is:

Unit 09: Huffman Encoding

Symbol	Count	Codes	Cost
A	50	1	$50*1$
B	10	001	$10*3$
C	30	01	$30*2$
D	5	0001	$5*4$
E	3	00001	$3*5$
F	2	00000	$2*5$

Number of bits in message after variable size encoding is 185 bits.

Cost of characters = $6 * 8 = 48$

Cost of variable size code = 20

So, the total cost in variable size encoding = $185 + 48 + 20 = 253$.

Hence the difference of cost using three methods are:

- With ASCII Encoding = 800
- With Fixed Size Encoding = 396
- With Variable Size Encoding = 253

9.4 Example -3

The given message is A = 5, B = 9, C = 12, D = 13, E = 16 and F = 45. The total length of message is 100.

ASCII encoding:

According to ASCII encoding technique, the message will be of $8*100 = 800$ bits.

Fixed Size encoding

For representation of 6 symbols, we require atleast combination of 3 bits. So, from 000, 001, 010, 011, 100, 101, 110 and 111, we choose 6 for representation of 6 symbols.

Symbol	Frequency	Code	Cost
A	5	000	$3*5 = 15$
B	9	001	$9*3 = 18$

C	12	010	$12*3 = 36$
D	13	011	$13*3 = 39$
E	16	100	$16*3 = 48$
F	45	101	$45*3 = 135$
	100		291

For symbols = $8 * 6 = 48$

For codes = $3 * 6 = 18$

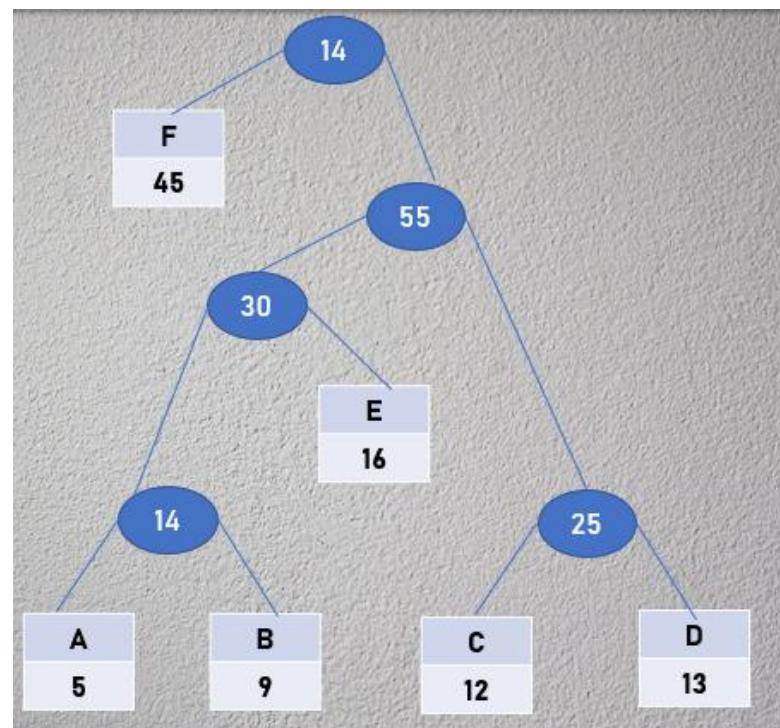
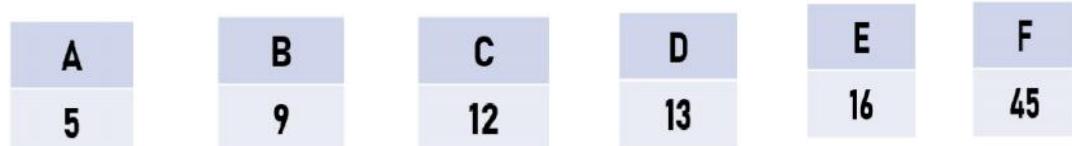
Message length = 291

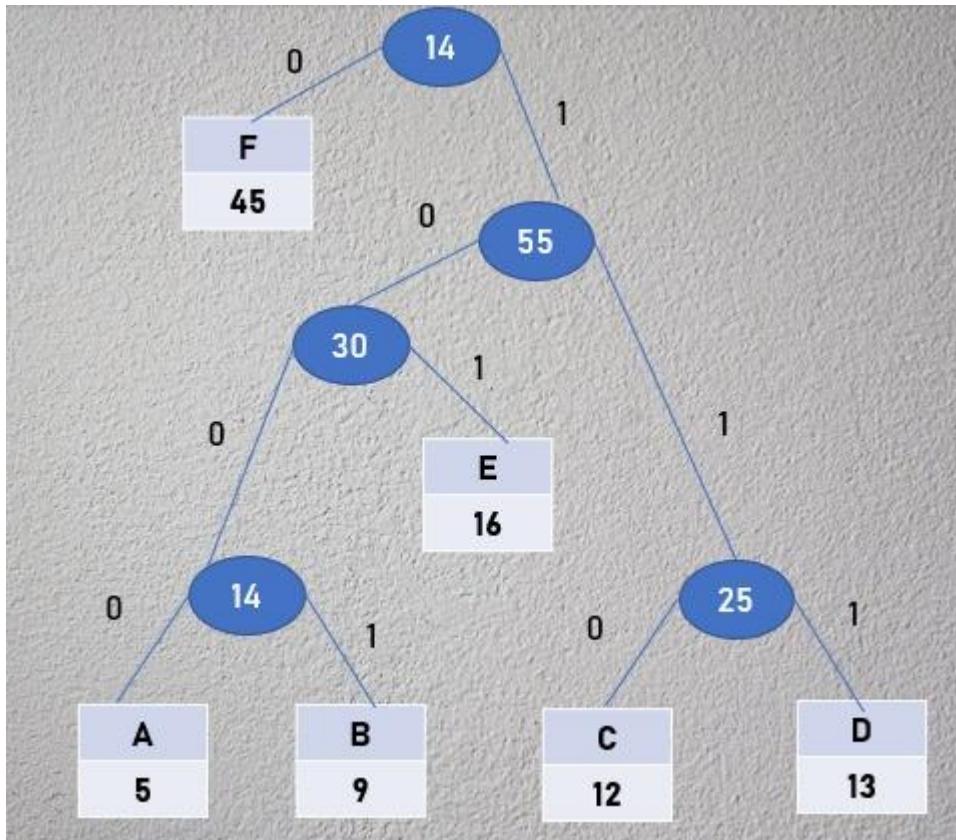
Total cost = $48 + 18 + 291 = 357$

Variable Size Encoding:

Follow the same procedure given in example 1.

- A = 5, B = 9, C = 12, D = 13, E = 16 and F = 45.





- F = 0, A = 1000, B = 1001, C = 110, D = 111, E = 101

Symbol	Frequency	Code	Cost
A	5	1000	$4*5 = 20$
B	9	1001	$9*4 = 36$
C	12	110	$12*3 = 36$
D	13	111	$13*3 = 39$
E	16	101	$16*3 = 48$
F	45	0	$45*1 = 45$
	100	18	224

For symbols = $8 * 6 = 48$

For codes = 18

Message length = 224

Algorithm Design and Analysis

Total cost = 48 + 18 + 224 = 290

Therefore, the difference of costs using these three methods are:

- ASCII encoding = 800 bits
- Fixed size encoding = 357 bits
- Variable size encoding = 290 bits

9.5 Algorithm

Input is an array of unique characters along with their frequency of occurrences and **output** is Huffman Tree.

Step 1: Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

Step 2: Extract two nodes with the minimum frequency from the min heap.

Step 3: Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

Step 4: Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node, and the tree is complete.

Time complexity

$O(n\log n)$ where n is the number of unique characters. If there are n nodes, extractMin() is called $2*(n - 1)$ times. extractMin() takes $O(\log n)$ time as it callesminHeapify(). So, overall complexity is $O(n\log n)$.

Summary

- Huffman encoding is an encoding and compression technique.
- Huffman encoding is used for reducing the size of data and message.
- Huffman encoding follows the greedy technique.
- The range of ASCII codes is 0-127.
- ASCII codes are of 8 bits.
- Huffman coding is a lossless data compression algorithm.
- Huffman code follows optimal merge pattern.
- The overall complexity in Huffman encoding is $O(n\log n)$.

Keywords

- **Encoding:** Encoding is the process of converting the data or a given sequence of characters, symbols, alphabets etc., into a specified format, for the secured transmission of data.
- **ASCII encoding:** The ASCII (American Standard Code for Information Interchange) code consists of a 7-bit binary standard used to encode a set of 128 graphic and control symbols.
- **Fixed size encoding:** To assign the fixed size code to the message. The size of the code depends upon the number of alphabets in the message.
- **Huffman encoding:** Huffman says that there is no need to take fixed size code, some characters may be appearing for few numbers of times, and some may be appearing for

Unit 09: Huffman Encoding

more number of times. If we provide the smaller code to the more appearing characters, then the size of the entire message will be reduced.

- **Huffman Tree:** Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

Self Assessment

1. The variable size encoding is
 - A greedy technique
 - An encoding technique
 - A data compression technique
 - All of the above
2. What represents the range of ASCII codes?
 - 0-110
 - 0-127
 - 0-101
 - 0-172
3. In variable length assignment, the most less occurred characters gets the _____ code.
 - Smallest
 - Longest
 - Equal
 - None of the above
4. In variable size encoding, we arrange all the alphabets in _____ order of their count.
 - Increasing
 - Decreasing
 - Either of the above
 - None of the above
5. With which approach, the Huffman encoding can be best implemented?
 - Exhaustive search
 - Greedy method
 - Brute force algorithm
 - Divide and conquer approach
6. In Huffman encoding, the data in a tree always occur?
 - Root
 - Leaves
 - Left subtree
 - Right subtree

7. The time complexity of Huffman encoding algorithm is
 - A. $O(n)$
 - B. $O(\log n)$
 - C. $O(n \log n)$
 - D. None of the above

8. In variable length assignment, the most occurred characters gets the _____ code.
 - A. Smallest
 - B. Longest
 - C. Equal
 - D. None of the above

9. Suppose the message is 'BCCABBDACC'. What will be the cost of sending the message using ASCII encoding?
 - A. 20
 - B. 8
 - C. 80
 - D. 28

10. Suppose the message is 'BCCABBDDAECCBBAEDDCC'. In The smallest code should be assigned to _____ alphabet.
 - A. A
 - B. B
 - C. C
 - D. D
 - E. E

11. Suppose the message is 'BCCABBDDAECCBBAEDDCC'. In The longest code should be assigned to _____ alphabet.
 - A. A
 - B. B
 - C. C
 - D. E

12. Out of these methods, which one will have least cost?
 - A. ASCII encoding
 - B. Fixed size encoding
 - C. Huffman encoding
 - D. All will have same cost

13. Out of these methods, which one will have most cost?
 - A. ASCII encoding
 - B. Fixed size encoding
 - C. Variable size encoding
 - D. All will have same cost

14. Which organization of data is most suitable in Huffman encoding?
- Stack
 - Graph
 - Min-heap
 - Max-heap

15. What is the time taken by min-heap for Huffman encoding?
- $O(n)$
 - $O(\log n)$
 - $O(1)$
 - None of the above

Answers for Self Assessment

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

Review Questions

- What is encoding of data? What are the applications of encoding? Explain different ways to encode the data.
- What is ASCII encoding? Explain with example.
- What is fixed size encoding? Explain with example. How is it better as compared to ASCII encoding?
- What is variable size encoding? Explain with example. How is it better as compared to ASCII and fixed size encoding? Explain the differences in cost as well.
- Given a message which contains the characters and associated frequencies as:

Character	Frequency
a	10
e	15
i	12
o	3
u	4
s	13
t	1

Construct the Huffman tree.

6. Explain the algorithm of Huffman encoding? What is its time complexity?



Further Readings

<https://www.gatevidyalay.com/huffman-coding-huffman-encoding/>

Unit 10: Lower Bound Theory

CONTENTS

- Objectives
- Introduction
- 10.1 Comparison Trees
- 10.2 Oracle and Adversary Argument
- 10.3 Majority Element Problem
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand what lower bound theory and comparison trees is
- Understand the oracles and adversary arguments

Introduction

Lower bound theory concept is based upon the calculation of minimum time that is required to execute an algorithm known as a lower bound theory or Base Bound Theory. Lower bound theory uses several methods/techniques to find out the lower bound. The main aim is to calculate a minimum number of comparisons required to execute an algorithm. There are various techniques which are used by lower bound theory:

- Comparisons Trees.
- Oracle and adversary argument
- State Space Method

10.1 Comparison Trees

In a comparison sort, we use only comparisons between elements to gain order information about an input sequence ($a_1; a_2; \dots; a_n$). Given a_i, a_j from (a_1, a_2, \dots, a_n) , We perform one of the comparisons:

- $a_i < a_j$ less than
- $a_i \leq a_j$ less than or equal to
- $a_i > a_j$ greater than
- $a_i \geq a_j$ greater than or equal to
- $a_i = a_j$ equal to

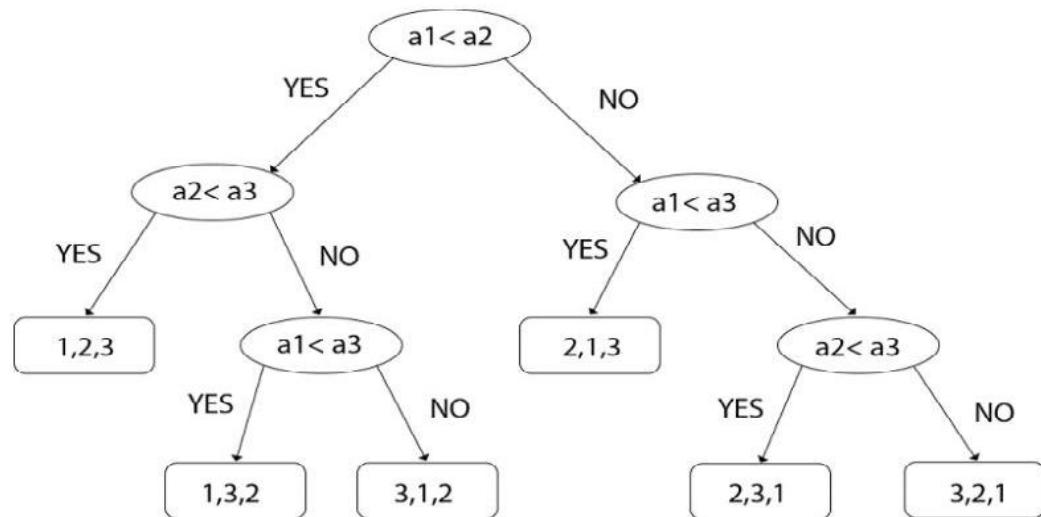
To determine their relative order, if we assume all elements are distinct, then we just need to consider $a_i \leq a_j$ ' \neq ' is excluded &, $\geq, \leq, >, <$ are equivalent. Consider sorting three numbers a_1, a_2 , and a_3 . There are $3! = 6$ possible combinations:

Algorithm Analysis and Design

- (a1, a2, a3), (a1, a3, a2),
- (a2, a1, a3), (a2, a3, a1)
- (a3, a1, a2), (a3, a2, a1)

The Comparison based algorithm defines a decision tree.

Decision Tree: A decision tree is a full binary tree that shows the comparisons between elements that are executed by an appropriate sorting algorithm operating on an input of a given size. Control, data movement, and all other conditions of the algorithm are ignored. In a decision tree, there will be an array of length n. So, total leaves will be $n!$ (i.e., total number of comparisons). If tree height is h, then surely, $n! \leq 2^n$ (tree will be binary). Taking an Example of comparing a1, a2, and a3. Left subtree will be true condition i.e., $a_i \leq a_j$. Right subtree will be false condition i.e., $a_i > a_j$.



So, from above, we got : $N! \leq 2^n$. Taking Log both sides and solve it, we get $h = n \log n$.

Comparison tree for Binary Search:

Suppose we have a list of items according to the following Position: 1,2,3,4,5,6,7,8,9,10,11,12,13,14. The mid will be calculated using the formula:

$$\text{Mid} = \text{Ceil} [(1+14)/2] = \text{Ceil} (15/2) = \text{Ceil} (7.5) = 7$$

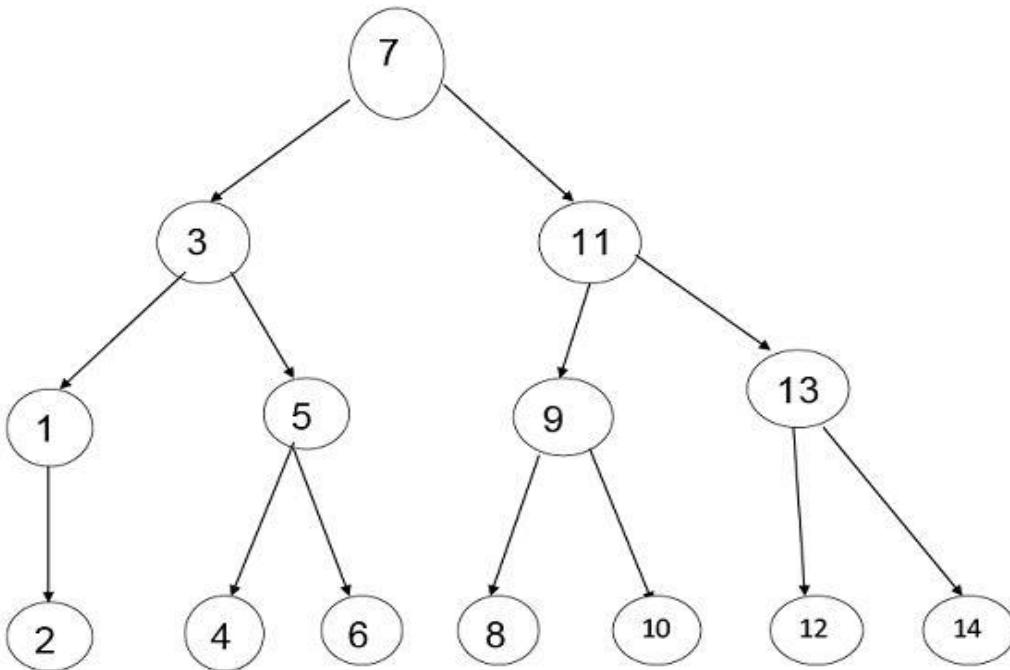
Using mid as 7, the list will be divided.

1, 2, 3, 4, 5, 6 $\text{Mid} = \left(\frac{1+6}{2} \right) = 3$	8, 9, 10, 11, 12, 13, 14 $\text{Mid} = \left(\frac{8+14}{2} \right) = 11$
--	---

In the left list, we take mid as 3 and in the right list we take mid as 11. Based upon that we divide our list.

1, 2 Mid= $(\frac{1+2}{2})=1$	4, 5, 6 Mid= $(\frac{4+6}{2})=5$	8, 9, 10 Mid= $(\frac{8+10}{2})=9$	12, 13, 14 Mid= $(\frac{12+14}{2})=13$
----------------------------------	-------------------------------------	---------------------------------------	---

And the last midpoint is: 2, 4, 6, 8, 10, 12, 14 Thus, we will consider all the midpoints and we will make a tree of it by having stepwise midpoints. According to Mid-Point, the tree will be:



Step1: Maximum number of nodes up to k level of the internal node is 2^k-1 . For Example:

- 2^k-1
- $2^3-1 = 8-1=7$
- Where k = level=3

Step2: Maximum number of internal nodes in the comparisons tree is $n!$. Here Internal Nodes are Leaves.

Step3: From Condition1 & Condition 2 we get

- $N! \leq 2^k-1$
- $14 < 15$
- Where N = Nodes

Step4: Now, $n+1 \leq 2^k$

- Here, Internal Nodes will always be less than 2^k in the Binary Search.

Step5:

- $n+1 \leq 2^k$
- $\log(n+1) = k \log 2$
- $k \geq \log(n+1)/\log 2$
- $k \geq \lceil \log_2(n+1) \rceil$

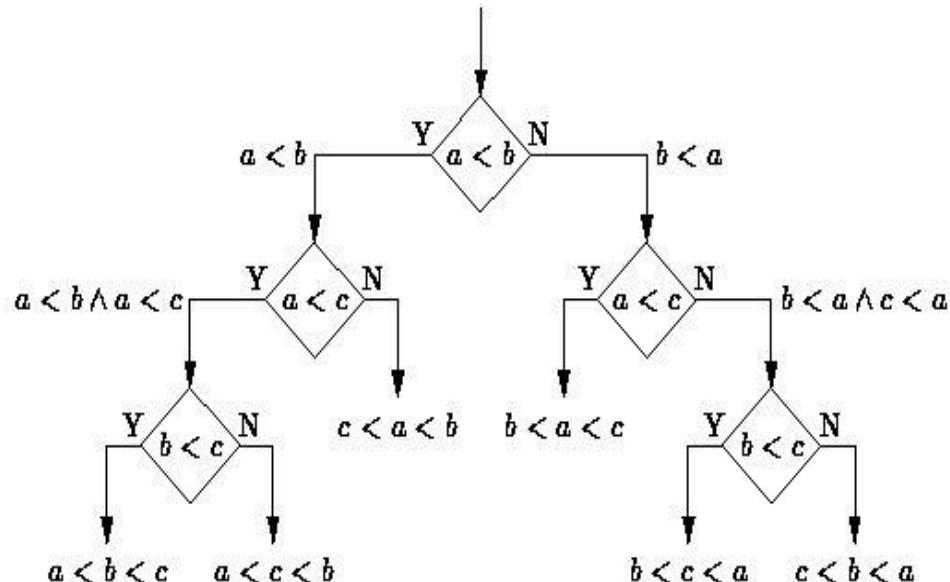
Step6: $T(n) = k$

Step7: $T(n) \geq \log_2(n+1)$

Problem of sorting

Consider the problem of sorting the sequence comprised of three distinct items. i.e., $a \neq b \wedge a \neq c \wedge b \neq c$.

Comparison sorting



10.2 Oracle and Adversary Argument

Another technique for obtaining lower bounds consists of making use of an "oracle." Given some model of estimation such as comparison trees, the oracle tells us the outcome of each comparison. To derive a good lower bound, the oracle efforts its finest to cause the algorithm to work as hard as it might. It does this by deciding as the outcome of the next analysis, the result which matters the most work to be needed to determine the final answer. And by keeping step of the work that is finished, a worst-case lower bound for the problem can be derived.

Example: (Merging Problem) given the sets A (1: m) and B (1: n), where the information in A and in B are sorted. Consider lower bounds for algorithms combining these two sets to give an individual sorted set. Consider that all the $m+n$ elements are specific and $A(1) < A(2) < \dots < A(m)$ and $B(1) < B(2) < \dots < B(n)$. Elementary combinatory tells us that there are $C((m+n), n)$ ways that the A's and B's may merge while still preserving the ordering within A and B. Thus, if we need comparison trees as our model for combining algorithms, then there will be $C((m+n), n)$ external nodes and therefore at least $\log C((m+n), n)$ comparisons are needed by any comparison-based merging algorithm. If we let MERGE(m, n) be the minimum number of comparisons used to merge m items with n items then we have the inequality as

Unit 10: Lower Bound Theory

- $\log C((m+n), m)$ MERGE (m, n) $m+n-1$.

The upper bound and lower bound can get promptly far apart as m gets much smaller than n .

Adversary

It is also one of the methods for obtaining worst case lower bounds. It is the second algorithm which intercepts access to data structures. It constructs the input data only as needed. It attempts to make original algorithm work as hard as possible. It analyzes adversary to obtain lower bound. The important restriction is although data is created dynamically, it must return consistent results.

- If it replies that $x[1] < x[2]$, it can never say later that $x[2] < x[1]$.

Adversary Lower Bound Technique

Devise a strategy to construct a worst-case input for a correct algorithm.

- The algorithm is known, i.e., Insertion sort.
- The algorithm is unknown, i.e., comparison-based sorting algorithm

Guessing Game: $Z_{100} = \{0, 1, \dots, 99\}$, Guess what number in Z_{100} I have in mind? $|L_0| = 100$, $|L_1| \geq 50$, $|L_2| \geq 25$, $|L_3| \geq 13$, $|L_4| \geq 7$, $|L_5| \geq 4$, $|L_6| \geq 2$, $|L_7| \geq 1$. The worst case lower bound: $\lceil \log_2 100 \rceil = 7$

Design against an adversary

It is a good technique for solving comparison-based problem efficiently.

- It should choose comparisons for which both answers give the same amount of information
- Keep the decision tree as balanced as possible
- Binary search, merge sort, finding both max and min, finding second largest

(1) Finding both max and min

pair up comparison: $n/2$

find largest of the winners: $n/2-1$, find smallest of the losers: $n/2-1$

at least $3n/2-2$ comparisons

(2) Finding second-largest key

finding the max of n keys: $n-1$

finding the largest of keys directly lose to max: $\lceil \lg n \rceil - 1$

at least $n + \lceil \lg n \rceil - 2$

implementation: heap

(3) Finding median

Selection (Finding median)

Divide and conquer approach

Find a "good" partition?

in finding pivot for Quick sort, we have

$T(n) = T(q) + T(n-q-1) + \Theta(n)$

(1) fixed strategy

(2) random strategy

for selection

$$T(n) = T(\max(q, n-q-1)) + \Theta(n)$$

(1) fixed strategy: $\Theta(n^2)$ in the worst case

(2) random strategy: expected $\Theta(n)$

(3) group 5 strategy: $\Theta(n)$ in the worst case

lower bound: $3n/2 - 3/2, (2n, 3n)$

Lower Bound for Comparison Sort

- Input: $n!$ different permutations.
- The adversary D maintains a list L.

Adversary Strategy:

- Initially L contains all $n!$ permutations.
- When an algorithm compares ask $a[i] < a[j]?$
 - Let L1 be the permutation in L and $a[i] < a[j]$
 - Let L2 be the permutation in L and $a[i] \geq a[j]$
 - If L1 > L2, answer yes and let L = L1.
 - Else answer no and let L = L2.
- At least half of the permutations in L remain
- The algorithm is done until L = 1.
- So, the number of comparisons is atleast = $\Omega(n \log n)$.

10.3 Majority Element Problem

A majority element in an array A of size N is an element that appears more than $N/2$ times. For example, the array 1,3,2,3,2,3,3 has a majority element 3; 1,3,2,3,2,4 has no majority element. The majority element problem is to find the majority element in an array, output -1 if it does not have one.

- Method 1: Counting the appearance times of each element. The time complexity is $O(n^2)$.
- Method 2: (1) Sorting the array in $O(n \lg n)$ time
 (2) Find the longest duplicated element in $O(n)$ time
 Thus, the complexity of the algorithm is $O(n \lg n)$
- Method 3: (linear solution)

Assume n is even, we find the candidate majority element as follows: we pair up element $A[2i-1]$ with $A[2i]$, for $i=1,2,\dots,n/2$, for each pair, if two elements are equal, put the element into array B, else discard both. B is the candidate set, where $|B| \leq n/2$.

Summary

- Lower Bound Theory uses several methods/techniques to find out the lower bound.
- The techniques which are used by lower Bound Theory are: comparisons trees, oracle and adversary argument and statespace method.

Unit 10: Lower Bound Theory

- In decision trees the control, data movement, and all other conditions of the algorithm are ignored.
- To derive a good lower bound, the oracle efforts its finest to cause the algorithm to work as hard as it might.
- Adversary is a method for obtaining worst case lower bounds.

Keywords

- **Lower bound theory:** Lower bound theory concept is based upon the calculation of minimum time that is required to execute an algorithm known as a lower bound theory or Base Bound Theory.
- **Comparison Sorting:** In a comparison sort, we use only comparisons between elements to gain order information about an input sequence ($a_1; a_2; \dots; a_n$).
- **Decision Tree:** A decision tree is a full binary tree that shows the comparisons between elements that are executed by an appropriate sorting algorithm operating on an input of a given size. Control, data movement, and all other conditions of the algorithm are ignored.
- **Oracle:** Given some model of estimation such as comparison trees, the oracle tells us the outcome of each comparison.

Self Assessment

1. Which method out of these makes the algorithm work harder by adjusting inputs?
 - Decision trees
 - Oracles
 - Adversary arguments
 - None of the above
2. In merging of arrays A (1...m) and B (1...n), how many ways are there by which A and B can merge?
 - $C((m-n), n)$
 - $C((m+n), m)$
 - $C((m+n), n)$
 - $C((m-n), m)$
3. We can design against an adversary for _____
 - Binary search
 - Merge sort
 - Finding min and max
 - All of the above
4. In finding the second largest key problem, which data structure is used for implementation?
 - Stack
 - Queue
 - Tree
 - Heap

Algorithm Analysis and Design

5. In finding median problem, which approach is applied?
 - A. Divide and Conquer
 - B. Dynamic programming
 - C. Backtracking
 - D. Branch and Bound
6. In majority element problem, if the array is of size N, then the majority element should appear for
 - A. More than N times
 - B. More than $N/2$ times
 - C. More than $N/3$ times
 - D. More than $N/4$ times
7. The concept of lower bound is based upon the calculation of _____ time required to execute an algorithm.
 - A. Minimum
 - B. Maximum
 - C. Median of time
 - D. None of the above
8. The technique used by the lower bound theory are
 - A. Comparison trees
 - B. Oracle and adversary arguments
 - C. State space methods
 - D. All of the above
9. In comparison trees, what kind of comparisons can be included to gain order information?
 - A. $a \leq b$
 - B. $a \geq b$
 - C. $a = b$
 - D. Either of the above three
10. Which of these symbols is excluded if we assume that all the elements are distinct?
 - A. $a \leq b$
 - B. $a \geq b$
 - C. $a = b$
 - D. Either of the above three
11. In a decision tree, if the array is of length n, then the total leaves will be
 - A. N
 - B. n^2
 - C. 2^n
 - D. $n!$

Unit 10: Lower Bound Theory

12. How the mid is calculated for searching an element by binary search?
 - A. $\text{mid} = l+h$
 - B. $\text{mid} = (l+h) / l$
 - C. $\text{mid} = (l+h) / h$
 - D. $\text{mid} = (l+h) / 2$
13. Which mathematical function is considered for calculation of mid in binary search?
 - A. Floor
 - B. Ceiling
 - C. Sqrt
 - D. Cubrt
14. In a decision tree, which of these represent the comparisons?
 - A. Internal nodes
 - B. Leaves
 - C. Root node
 - D. None of the above
15. In a decision tree, which of these represent the outcomes?
 - A. Internal nodes
 - B. Leaves
 - C. Root node
 - D. None of the above

Answers for Self Assessment

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

Review Questions

1. What is lower bound theory? Briefly explain the techniques used for lower bound theory concept.
2. Explain the working of comparison trees in detail. Give the appropriate example for the same.
3. Explain the comparison tree for binary search in detail.
4. What is oracle and adversary argument? Explain.
5. What is an adversary? What is important restriction? Give appropriate example for the same.



Further Readings

<https://www.coursehero.com/file/46352126/Oracles-Adversary-Argumentspdf/>



Web Links

<https://www.javatpoint.com/daa-lower-bound-theory>

<https://www.geeksforgeeks.org/lower-bound-on-comparison-based-sorting-algorithms/>

Unit 11: More on Lower Bounds

CONTENTS

- Objectives
- Introduction
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the lower bound theory through reductions.
- Understand various examples of lower bound theory

Introduction

A value that is less than or equal to every element of a set of data. Reduction is the action or fact of making something smaller or less in amount, degree, or size. Example: in {3,5,11,20,22} 3 is a lower bound. Lower bound theory is the concept based upon the calculation of minimum time required to execute an algorithm. In previous unit, the comparison trees are used for searching and sorting problem.

Now we are going to discuss about the lower bound theory through reductions. Reduction here means, suppose one problem is solved based upon that another problem can be solved easily through reduction. These reduction techniques are used for reducing the given problem to another problem for which the lower bound is already known.

The idea behind the lower bound through reduction is: If problem P is at least as hard as problem Q, then lower bound for Q is also a lower bound for P. Hence find problem Q with a known lower bound that can be reduced to problem P. Then any algorithm that solves P will also solve Q.

Let P1 reduces to P2. P1 be the selection problem and P2 is the sorting problem. Suppose we have the numbers sorted in the array, then we can find the ith smallest element from the input which can be easily obtained. The solution for P1 problem can also be obtained from the solution from P2 in time $\leq T(n)$. Thus, P1 reduces to P2 in $O(1)$ time.

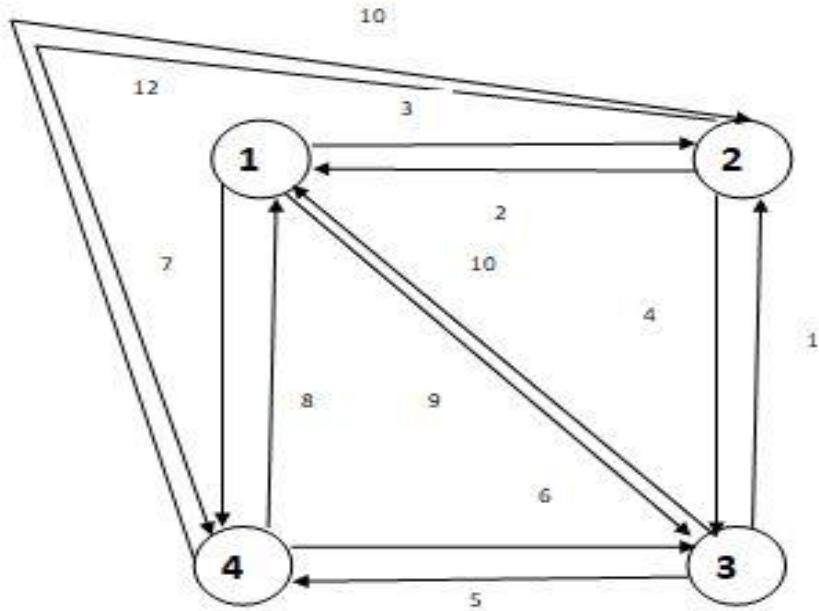


Example 1:

P1 is Hamiltonian cycle problem and P2 is Travelling salesman problem

Travelling Salesman Problem: A salesman must find a path which is optimal. He must start from a starting vertex and visit all the vertices exactly once and should return to that starting vertex after visiting.

Hamiltonian Cycle: Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path. Determine whether a given graph contains Hamiltonian Cycle or not. If it contains, then prints the path. Following are the input and output of the required function.



Starting from vertex 1, if we again come back to vertex 1 after visiting all the vertices exactly once, we calculate the cost.

- $1 - 2 - 3 - 4 - 1$
 $= 3 + 4 + 5 + 8$
 $= 20$

- $1 - 2 - 4 - 3 - 1$
 $= 3 + 12 + 6 + 10$
 $= 33$

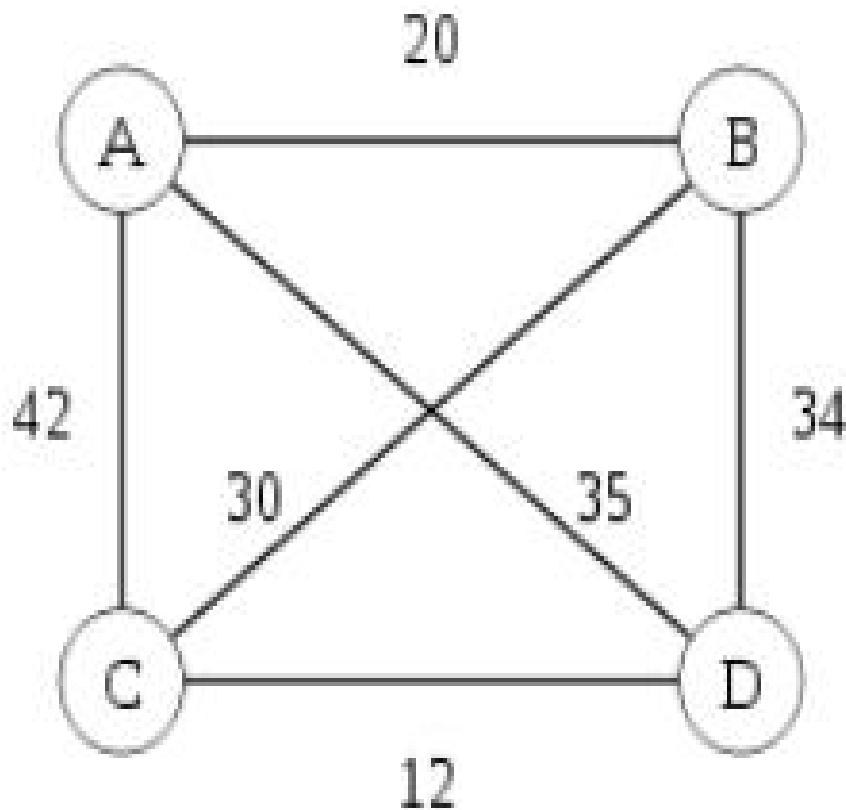
- $1 - 3 - 2 - 4 - 1$
 $= 9 + 1 + 12 + 8$
 $= 30$

- $1 - 3 - 4 - 2 - 1$
 $= 9 + 5 + 10 + 2$
 $= 26$

- $1 - 4 - 2 - 3 - 1$
 $= 7 + 10 + 5 + 10$
 $= 32$

- $1 - 4 - 3 - 2 - 1$
 $= 7 + 6 + 1 + 2$

= 16 (Minimum Cost)



Starting from vertex A, again coming back to vertex A after visiting all the vertices exactly once, we calculate the cost.

- A - B - C - D - A
= $20 + 34 + 12 + 42$
= 108.
- A - D - B - C - A
= $35 + 34 + 30 + 42$
= 141
- A - C - D - B - A
= $42 + 12 + 34 + 20$
= 108
- A - C - B - D - A
= $42 + 30 + 34 + 35$
= 141

Like this, we will calculate the costs. Now there is no need to find out Hamiltonian Cycle. Here the Hamiltonian Cycle means we need to start from some starting vertex and visit all the vertices exactly once and return to the starting vertex so that it can form a cycle. If P2 is solved, then we can easily find out the solution of P1.



Example 2:

Some matrices functions are taken for showing lower bound through reduction. An example is multiplying triangular matrices. An $n \times n$ matrix 'A' whose elements are $a_{ij} = 0$ for $i > j$, known as upper triangular matrices. An $n \times n$ matrix 'A' whose elements are $a_{ij} = 0$ for $i < j$, known as lower triangular matrices.

We are going to derive lower bounds for the problem of multiplying 2 lower triangular matrices. P1 is the problem of multiplying two full $n \times n$ matrices in time $M(n)$. P2 is the problem of multiplying two $n \times n$ lower triangular matrices in time $M_t(n)$. P1 is reducing in P2 in $O(n^2)$ time.

- $M_t(n) = \Omega(M(n))$

We have to show that P1 is reduces to P2 in $O(n^2)$ time. Note that $M(n) = O(n^2)$ since there are $2n^2$ elements in the input and n^2 in the output. Let P1 be the two matrices to be multiplied A and B of size $n \times n$ each. P2 be the two lower triangular matrices.

- $A' =$

0	0	0
0	0	0
0	A	0

- $B' =$

0	0	0
B	0	0
0	0	0

- $A', B' =$

0	0	0
0	0	0
AB	0	0

P1 was to find out $A * B$ and P2 was to find out $A' * B'$. So if we solve P2 we can easily get the solution of P1. So we can say that P1 reduces to P2 in $O(n^2)$ time. This also implies that $M(n) \leq M_t(3n) + O(n^2)$

- $M_t(n) = \Omega(M(n))$



Example 3

The lower bound through reduction is shown for inverting a lower triangular matrix. Inverting means finding the inverse of a matrix. Let A and B be the $n \times n$ matrices and I be the identity matrix. Identity matrix is a matrix where all the elements are 1. P1 is the problem of multiplying two full $n \times n$ matrices as $M(n)$. P2 is the problem of inverting a lower triangular matrix as $I_t(n)$. Finally, we need to show P1 reduces to P2 in $O(n^2)$ time. Hence, $M(n) = O(I_t(n))$ and $I_t(n) = \Omega(M(n))$.

Prove $M(n) = O(I_t(n))$. Let P1 reduces to P2 in $O(n^2)$ time. Let P1 be the two matrices to be multiplied A and B of $n \times n$ size each. Let P2 be the problem of lower triangular matrices with identity matrices. We need to find the inverse of lower triangular matrix with identity.

- $C =$

I	0	0
B	I	0
0	A	I

- Inverse of C is $C^{-1} =$

	0	0
-B		0
AB	-A	

So, we can conclude that the product of AB is obtained from the inverse of C. We get $M(n) \leq I_t(3n) + O(n^2)$. This can also be said as $M(n) = O(I_t(n))$ and $I_t(n) = \Omega(M(n))$.

Summary

- Comparison trees are used for searching and sorting problem.
- These reduction techniques are used for reducing the given problem to another problem for which the lower bound is already known.
- We can also compute the transitive closure using lower bound through reduction.
- Let P1 reduces to P2. P1 be the selection problem and P2 is the sorting problem. Suppose we have the numbers sorted in the array, then we can find the ith smallest element from the input which can be easily obtained.
- P1 is the problem of multiplying two full $n \times n$ matrices in time $M(n)$. P2 is the problem of multiplying two $n \times n$ lower triangular matrices in time $M_t(n)$. P1 is reducing in P2 in $O(n^2)$ time.

Keywords

- **Lower Bound Theory:** It is the concept based upon the calculation of minimum time required to execute an algorithm.
- **Reduction:** Reduction here means, suppose one problem is solved based upon that another problem can be solved easily through reduction.
- **Travelling Salesman Problem:** A salesman has to find a path which is optimal. He has to start from a starting vertex and visit all the vertices exactly once and should return back to that starting vertex after visiting.

Self Assessment

1. The comparison tree is used for
 - A. Searching
 - B. Sorting
 - C. Both searching and sorting
 - D. None of the above

2. The concept of lower bound theory is based upon calculation of
 - A. Minimum execution time
 - B. Maximum execution time
 - C. Both minimum and maximum execution time
 - D. None of the above

3. In travelling salesman problem, the salesman must find a path which gives the _____ cost.
 - A. Minimum
 - B. Maximum
 - C. Zero
 - D. Any of the above

4. An $n \times n$ matrix 'A' whose elements are $a_{ij} = 0$ for $i > j$, then it is a _____ triangular matrix.
 - A. Upper
 - B. Lower
 - C. Medium
 - D. None of the above

5. An $n \times n$ matrix 'A' whose elements are $a_{ij} = 0$ for $i < j$, then it is a _____ triangular matrix.
 - A. Upper
 - B. Lower
 - C. Medium
 - D. None of the above

6. For matrix multiplication, the number of _____ of first matrix must be equal to number of _____ of second matrix.
 - A. Row, row
 - B. Column, column
 - C. Row, column
 - D. Column, row

7. The two main measures for the efficiency of an algorithm are
 - A. Processor and memory
 - B. Complexity and classes
 - C. Time and space
 - D. Data and space

8. The time factor when determining the efficiency of algorithm is measured by
 - A. Counting microseconds
 - B. Counting the number of key operations
 - C. Counting the kilobytes of algorithm
 - D. None of the above

Algorithm Analysis and Design

9. Which of these cases does not exist in complexity theory?

- A. Best case
- B. Average case
- C. Worst case
- D. Null case

10. Choose the correct answer for the following statements:

I. The theory of NP-completeness provides a method of obtaining a polynomial time for NP algorithms.

II. All NP-complete problem are NP-Hard.

- A. I is FALSE and II is TRUE
- B. I is TRUE and II is FALSE
- C. Both are TRUE
- D. Both are FALSE

11. _____ is the first step in solving the problem

- A. Understanding the Problem
- B. Identify the Problem
- C. Evaluate the Solution
- D. None of these

12. _____ is the maximum number of steps that can be executed for the given parameters

- A. Average case
- B. Worst case
- C. Time complexity
- D. Best case

13. $O(1)$ means computing time is _____

- A. Constant
- B. Quadratic
- C. Linear
- D. Cubic

14. $O(n)$ means computing time is _____

- A. Constant
- B. Quadratic
- C. Linear
- D. Cubic

15. An algorithm that calls itself directly or indirectly is known as

- A. Sub algorithm
- B. Recursion
- C. Polish notation

D. Traversal algorithm

Answers for Self Assessment

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

Review Questions

1. What is lower bound theory? Explain the different methods used for it.
2. Explain the method of calculation of lower bound for searching and sorting problem.
3. If P1 is Hamiltonian cycle problem and P2 is Travelling salesman problem. Show P1 reduces to P2.
4. If P1 be the two matrices to be multiplied A and B of $n \times n$ size each and P2 be the problem of lower triangular matrices with identity matrices. Show that P1 is reducible to P2.
5. How can we compute the transitive closure using lower bound through reduction?
6. What is the idea behind lower bound theory? Explain its concept using examples.
7. Take two problems of your choice and show the concept of lower bound theory through reductions on it.



Further Readings

<https://www.cs.yale.edu/homes/aspnes/pinewiki/LowerBounds.html>

<https://cs.winona.edu/lin/cs440/ch11.pdf>

Unit 12: Learning and its Types

CONTENTS

- Objectives
- Introduction
- 12.1 Introduction to Machine Learning
- 12.2 Need for Machine Learning
- 12.3 Supervised Learning
- 12.4 Unsupervised Learning
- 12.5 Inductive Learning
- 12.6 Rule based Learning
- 12.7 Reinforcement Learning
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further readings

Objectives

1. Understanding the concepts of machine learning.
2. Understanding the difference between supervised and unsupervised learning.
3. Understanding different types of learning.
4. Understanding the concepts of rule based and inductive learning.
5. Understanding the importance of reinforcement learning.

Introduction

In this unit, the concepts of machine learning are discussed in detail. The different types of machine learning approaches are discussed such as supervised learning, unsupervised learning, inductive learning, rule based learning and reinforcement learning using examples. The preparation of the data sets is discussed along with basic data types. Particularly, the basics were given focus along with importance of each approach.

12.1 Introduction to Machine Learning

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. This is now widely used in across all the domains, starting from classification problems to humanoid robot.

Data set is the collection of data used for machine learning. Basically, the dataset is divided into three categories. They are training data, testing Data and validation Data. Here, the training data is considered for initial training purpose. Testing data is used for checking the trained machine. Validation data is used for tuning the trained machine with the help of important parameters. Training data is the data used to train an algorithm or machine learning model, is depicted in the given figure 1.

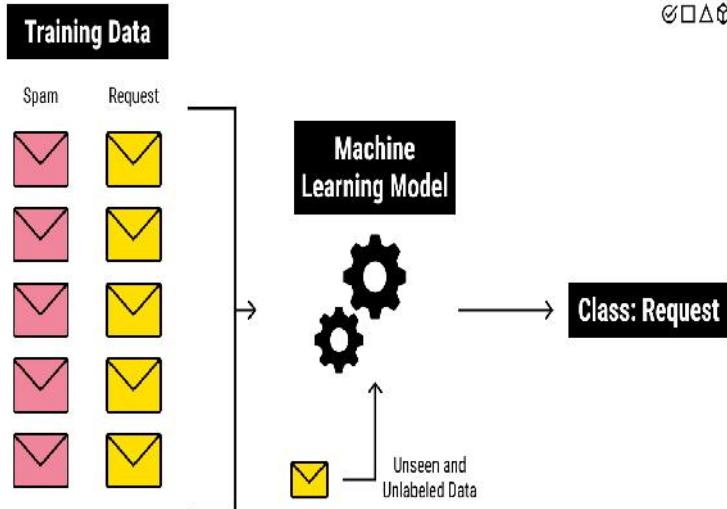


Figure 1 Machine Learning Model

Data preprocessing is the process of transforming raw data into an understandable format. So that it is used for machine learning effectively. The important steps involved in the process of preprocessing, are Data cleaning, Data integration, Data transformation and Data reduction. Data cleaning. These steps are not in our focus of this unit. But, still, let me give you in short. Data integration is the process of combining data from different sources into a single, unified view. Data transformation is the process of changing the format, structure, or values of data. Data reduction is the process of reducing the amount of capacity required to store data.

12.2 Need for Machine Learning

Today, machine learning is used in a wide range of applications. Machine learning has seen use cases ranging from predicting customer behavior to forming the operating system for self-driving cars. Machine learning as technology helps analyze large chunks of data, easing the tasks of data scientists in an automated process and is gaining a lot of prominence and recognition. Machine learning has changed the way data extraction and interpretation works by involving automatic computing algorithms that have replaced traditional statistical techniques. There are few applications to mention the importance.

- Real-time chat-bot agents
- Decision support
- Customer recommendation engines
- Customer churn modeling
- Dynamic pricing tactics
- Market research and customer segmentation
- Fraud detection
- Image classification and image recognition
- Operational efficiencies
- Information extraction

12.3 Supervised Learning

Machines are trained using well-labelled training data. The data provided for training should be very much correct without any false data as the machine solely depending on the given data for its training as in Figure 2.

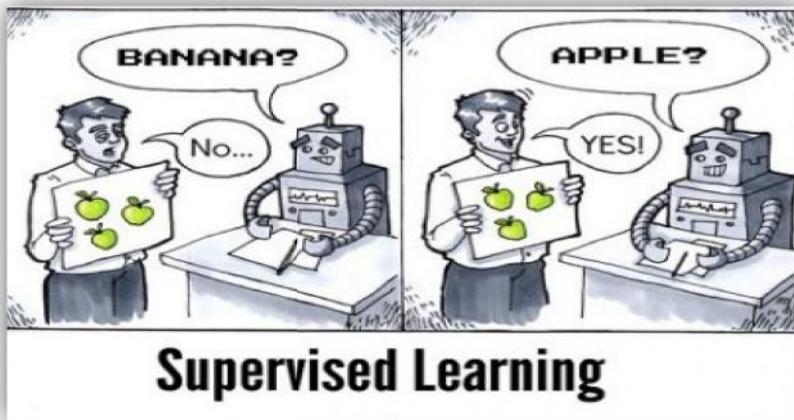


Figure 2 Supervised Machine Learning

Four types of data are explained here, as it is often handled in the process of dataset preparation or preprocessing. The data types are as given below.

- Numerical Data
- Categorical Data
- Time Series Data
- Text Data

Numerical Data

Numerical data is a datatype expressed in numbers as in Figure 3. This further classified as continuous and discontinuous data.

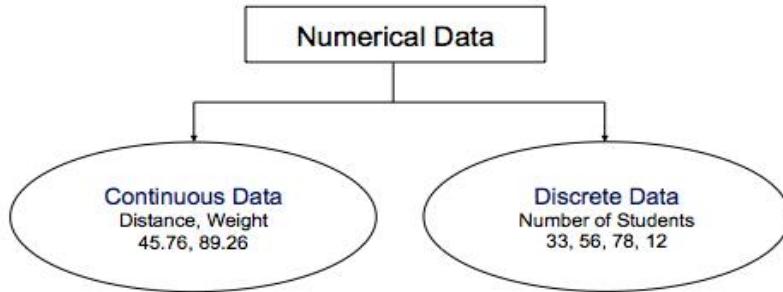


Figure 3 Numerical Data

Categorical Data

Categorical data is a collection of information that is divided into groups. We can see the "Gender" and "Remarks" as the categorical data as in the given Figure 4. They are further divided into two types such as ordinal and nominal.

Employee_ID	Gender	Remarks
0	45	Male
1	78	Female
2	56	Female
3	12	Male
4	7	Female

Figure 4 Categorical Data

Ordinal Data

Ordinal data has ranking / ordering. Ordinal features are sorted or ordered as in the figure 5.

Size of T-Shirt - S, M, L, XL.

Convert string values into integer as per order like XL > L > M > S.

Size	Encoded
S	0
XL	3
M	1
L	2

Figure 5 Ordinal Data

Nominal Data

- ❖ Nominal features are not ordered as in figure 6. Nominal data has No ranking / order.
- ❖ Color of T-Shirt : Red, Green, Blue.
- ❖ Assign numeric value to each feature.
- ❖ 0 -> Red, 1 -> Green, 2 -> Blue

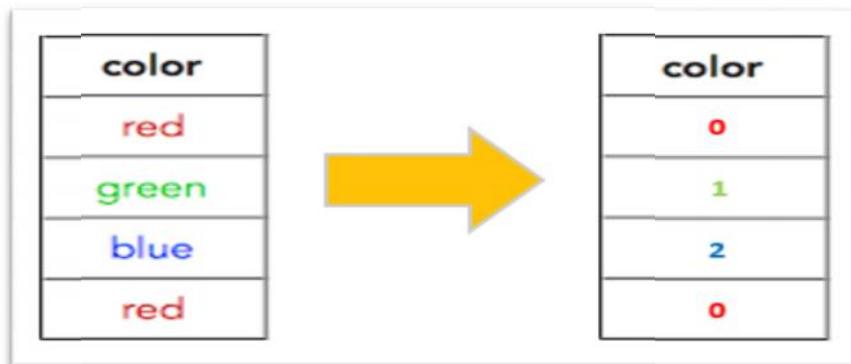


Figure 6 Nominal Data

Time Series Data

A Time Series is a sequence of data points (as in Figure 7) that occur in successive order over some period of time. The similar graphs can be seen in the stock markets, is also an example for this.



Figure 7 Time Series Data

Text Data

Text data usually consists of documents, which can represent words, sentences or even paragraphs. Figure 8 describes the text data as it can be books, material, anything that is written or typed.



Figure 8 Text Data

12.4 Unsupervised Learning

Unsupervised learning is a kind of machine learning where a model must look for patterns in a given dataset with no labels marked in them. Figure 9 depicts the how human does the unsupervised learning. This avoids supervision / teacher.

- No external guidance in performing the task.
- It will identify patterns on its own from the given data sets, which are neither classified nor labelled.
- Pattern / Distance metric is used by Machine to group or classify the given data.

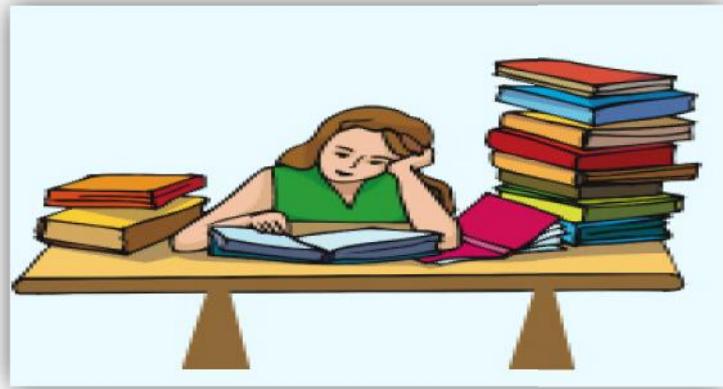


Figure 9 Unsupervised Machine Learning

12.5 Inductive Learning

Inductive learning is a process where the learner discovers rules by observing examples. The inductive learning is also known as discovery learning. The Inductive learning is different from deductive learning, where students are given rules that they then need to apply.

We often work out rules for ourselves by observing examples to see if there is a pattern or not, to see if things regularly happen in the same way. We then try applying the rule in different situations to see if it works or not.

12.6 Rule based Learning

Rule-based learning is just another type of learning which makes the class decision depending by using various "if..else" rules. These rules are easily interpretable and thus these classifiers are generally used to generate descriptive models. The condition used with "if" is called the antecedent and the predicted class of each rule is called the consequent.

Rules can be ordered, i.e. the class corresponding to the highest priority rule triggered is taken as the final class.

Otherwise, we can assign votes for each class depending on some of their weights, i.e. the rules remain unordered.

Actually, Rules-based systems are best suited to situations in which there are lower volumes of data and the rules are relatively simple

12.7 Reinforcement Learning

This is concerned with how software agents should take actions in an environment. Reinforcement learning approach helps you to maximize some portion of the cumulative reward. There are few terminologies to be remembered for this study as shown in Figure 10.

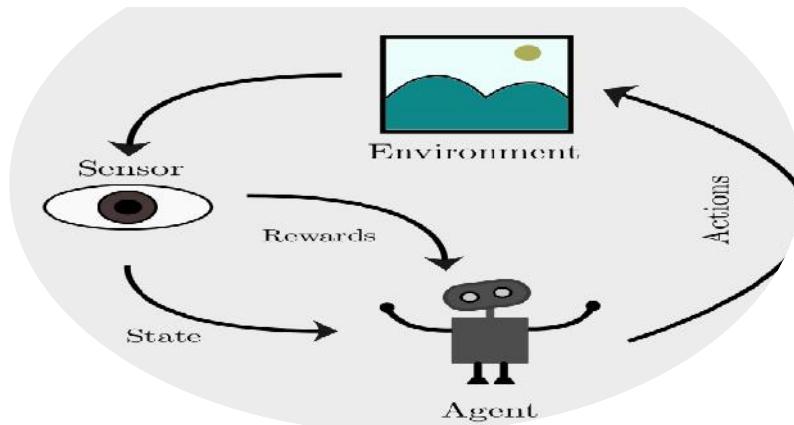


Figure 10 Reinforcement Machine Learning

Agent

It is an assumed entity which performs actions in an environment to gain some reward.

Environment (e)

A scenario that an agent has to face.

Reward (R)

An immediate return given to an agent when he or she performs specific action or task.

State (s)

State refers to the current situation returned by the environment.

Policy (π)

It is a strategy which applies by the agent to decide the next action based on the current state.

Value (V)

It is expected long-term return with discount, as compared to the short-term reward.

Value Function

It specifies the value of a state that is the total amount of reward. It is an agent which should be expected beginning from that state.

Model of the environment

This mimics the behavior of the environment. It helps you to make inferences to be made and also determine how the environment will behave.

Model based methods

It is a method for solving reinforcement learning problems which use model-based methods.

Q value or action value (Q)

Q value is quite similar to value. The only difference between the two is that it takes an additional parameter as a current action.

To make this concept simple, let us take a situation in our house. Have a look at the given figure 11. The cat is an agent that is exposed to the environment. In this case, an example of a state could be the cat sitting, and you use a specific word for the cat to walk.

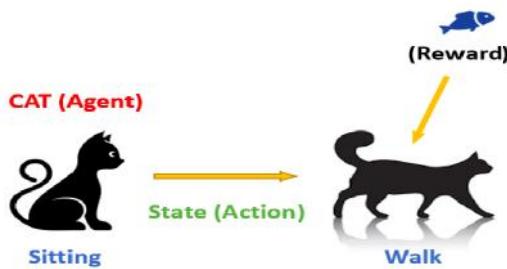


Figure 11 Example for Reinforcement learning

The cat agent reacts by performing an action transition from one “state” to another “state.” For example, the cat goes from sitting to walking. The reaction of an agent is an action, and the policy is a method of selecting an action given a state in expectation of better outcomes. After the transition, they may get a reward or penalty in return.

There are three approaches to implement a Reinforcement Learning algorithm. They are Value based, in which we should try to maximize a value function $V(s)$. In this method, the agent is expecting a long-term return of the current states under policy π . Second method is policy based, in which we develop a policy that the action is performed in every state helps you to gain maximum reward in the future. The last is Model based, in which we need to create a model for each environment and the agent learns to perform in that specific environment only.

Characteristics

- No prior experience.
- No perfect decision.
- Always trying to perform better every time.
- It is bound to learn from its own experience.

Types of Reinforcement Learning

There are two types of reinforcement learning methods, which are called positive and negative. Positive method is defining an agent that increases the positivity of the action, focusing only on positive rewards. On the other hand, negative method is defining an agent that removing the rewards given to that action, allowing the same action is repeated again for well training. Such as, A person learns to wear a raincoat, during the rainy season, to avoid getting wet and People wear helmets, to avoid getting injured, in case of a road accident, or getting fined by cops. Repeated actions created for learning.

Summary

In this unit, the concepts of machine learning are discussed along with the different approaches of machine learning. Each approach is discussed in detail with examples. The differences in each of the approaches would be better understood. Data set is very important for machine learning. Hence, it is necessary to understand about the basic data types, which is also explored thoroughly. This will help to convert or process the obtained data. But, there was also lot of challenges in processing the data set. This also covered in the name of preprocessing and data cleaning. The major tasks of preprocessing and the possible ways of data cleaning were also discussed. The terminology – feature engineering was highlighted as it was related to data cleaning.

Keywords

- Dataset
- Supervised learning
- Unsupervised learning
- Inductive learning
- Rule based learning

- Reinforcement learning

Self Assessment

Q1) Machine learning approach, which build a model based on sample data, is known as _____.

- A. Supervised
- B. Unsupervised
- C. Reinforcement
- D. None of the above

Q2) _____ approach uses the rewarding method for machine learning.

- A. Supervised
- B. Unsupervised
- C. Reinforcement
- D. None of the above

Q3) Which of the following dataset is used for supervised machine learning?

- A. Training dataset
- B. Testing dataset
- C. Validation dataset
- D. All the above

Q4) _____ machine learning approach uses unlabelled data for learning.

- A. Supervised
- B. Unsupervised
- C. Reinforcement
- D. None of the above

Q5) The two class problems are otherwise called _____.

- A. Clustering
- B. Binary Classification
- C. Multiclass Classification
- D. None of the above

Q6) Justify the statement. "Preprocessing is the process of converting raw data into data which will be suitable for machine learning".

- A. True
- B. False

Q7) Preprocessing is actually the combination of data cleaning and _____.

- A. Data integration
- B. Data transformation

- C. Data reduction
- D. All of the above

Q8) Supervised machine learning approach uses the _____ dataset.

- A. Labelled Dataset
- B. Unlabelled Dataset
- C. Both Labelled and Unlabelled
- D. None of the above

Q9) Imputation method is used for _____.

- A. Removing rows
- B. Removing columns
- C. Filling the missing values
- D. None of the above

Q10) _____ is the process of changing the format, structure or values of data.

- A. Data integration
- B. Data cleaning
- C. Data transformation
- D. Data Preprocessing

11. Justify the given statement. "A rule-based system consists of a bunch of IF-THEN rules".

- A True
- B False

12. Rule based system also known as _____.

- A. Mycin based system
- B. Human based system
- C. Knowledge based system
- D. None of the above

13. Which chaining rule has inductive approach?

- A Forward Chaining
- B Backward chaining
- C Spanning Chaining
- D All of the above

14. Reinforcement learning is _____ learning.

- A Supervised
- B Unsupervised
- C Award based
- D None of the above

15. _____ Reinforcement is defined as when an event, occurs due to a particular behavior.

- A) Negative
- B) Positive
- C) Neutral
- D) None of these

Answers for Self Assessment

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

Review Questions

1. Explain the different types of data.
2. Differentiate nominal and ordinal data types.
3. Give examples for categorical data.
4. List out the methods used for filling the missing values.
5. Identify the machine learning algorithms for each machine learning approaches.



Further readings

- S. N. Sivanandam, S.N. Deepa, Principles Of Soft Computing, Wiley Publications, Second Edition, 2011.
- Rajasekaran, S., Pai, G. A. Vijayalakshmi, Neural Networks, Fuzzy Logic and Genetic Algorithm Synthesis And Applications, Prentice Hall of India, 2013.
- N. P. Padhy, S. P. Simon, Soft Computing With Matlab Programming, Oxford University Press, 2015.



Web Links

- <https://www.javatpoint.com/types-of-machine-learning>
- <https://www.geeksforgeeks.org/ml-types-learning-supervised-learning/>
- <https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114>

Unit 13: Interactable Problems

CONTENTS

Objectives

Introduction

13.1 Classification of Problems

13.2 Main Categories of Algorithms

13.3 Difference between Deterministic and Non-deterministic Algorithms

13.4 Detailed Difference between Deterministic and Non-deterministic Algorithms

13.5 Major Categories of Problems

13.6 Boolean Satisfiability Problem

Summary

Keywords

Self Assessment

Answers for Self Assessment

Review Questions

Further Readings

Objectives

After studying this unit, you will be able to:

- Know the basics of tractable and intractable problems
- Understand the non-deterministic algorithm
- understand what NP-Completeness is

Introduction

There are common functions which have the complexities in constant, logarithmic, linear, quadratic, cubic, exponential, factorial, and super-exponential terms.

- **Constant:** $O(1)$
- **Logarithmic:** $O(\log n)$
- **Linear:** $O(n)$
- **Quadratic:** $O(n^2)$
- **Cubic:** $O(n^3)$
- **Exponential:** $O(k^n)$, e.g. $O(2^n)$
- **Factorial:** $O(n!)$
- **Super-exponential:** $O(n^n)$

So, based upon this we can categorize the functions into two classes. These are: polynomial functions and exponential functions. Polynomial Functions are the functions that are derivable from $O(n^k)$, i.e., bounded from above by n^k for some constant k . Examples: $O(1)$, $O(\log n)$, $O(n)$, $O(n \times \log n)$, $O(n^2)$, $O(n^3)$. Exponential functions are the remaining functions. Examples: $O(2^n)$, $O(n!)$, $O(n^n)$. In the context of programming, an algorithm is a set of well-defined instructions in

Algorithm Design and Analysis

sequence to perform a particular task and achieve the desired output. Here we say, the set of defined instructions which means that somewhere user knows the outcome of those instructions if they get executed in the expected manner. Based on this classification of functions into polynomial and exponential, we can classify algorithms as well:

- **Polynomial-Time Algorithm:** An algorithm whose order-of-magnitude time performance is bounded from above by a polynomial function of n , where n is the size of its inputs.
- **Exponential Algorithm:** An algorithm whose order-of-magnitude time performance is not bounded from above by a polynomial function of n .

13.1 Classification of Problems

In a similar way, we can classify problems into two broad classes:

- **Tractable Problem:** A problem that is solvable by a polynomial-time algorithm. The upper bound is polynomial. Examples of tractable problems: Ones with known polynomial-time algorithms.
 - 1) Searching an unordered list
 - 2) Searching an ordered list
 - 3) Sorting a list
 - 4) Multiplication of integers (even though there's a gap)
 - 5) Finding a minimum spanning tree in a graph (even though there's a gap)
- **Intractable Problem:** A problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential. Examples of intractable problems: Ones that have been proven to have no polynomial-time algorithm.
 - 1) Some of them require a non-polynomial amount of output, so they clearly will take a non-polynomial amount of time, e.g.:
 - * Towers of Hanoi: we can prove that any algorithm that solves this problem must have a worst-case running time that is at least $2^n - 1$.
 - * List all permutations (all possible orderings) of n numbers.
 - 2) Others have polynomial amounts of output, but still cannot be solved in polynomial time:
 - * For an $n \times n$ draughts board with an arrangement of pieces, determine whether there is a winning strategy for White (i.e. a sequence of moves so that, no matter what Black does, White is guaranteed to win). We can prove that any algorithm that solves this problem must have a worst-case running time that is at least 2^n .

So, after seeing this classification we can easily see that the problems can be neatly divided into these two classes. But this ignores 'gaps' between lower and upper bounds. Incredibly, there are problems for which the state of our knowledge is such that the gap spans this coarse division into tractable and intractable. So, in fact, there are three broad classes of problems: - Problems with known polynomial-time algorithms. Problems that are provably intractable (proven to have no polynomial-time algorithm). Problems with no known polynomial-time algorithm but not yet proven to be intractable. Common examples of Intractable problems are travelling salesman problem and halting problem. In travelling salesman problem, the salesman has to travel from the starting city to all cities on the map and back to the starting city, for the lowest cost. In halting problem, when you run a program, it will either run for a while and then stop (halt), or get stuck in an infinite loop and crash the machine. If a program runs for a long time, do we conclude the program is stuck in a loop or that we have not allowed enough time for the program to calculate its output. The halting problem asks: 'Is it possible to write a program that can tell given any program and its inputs and without executing the program, whether it will halt?'

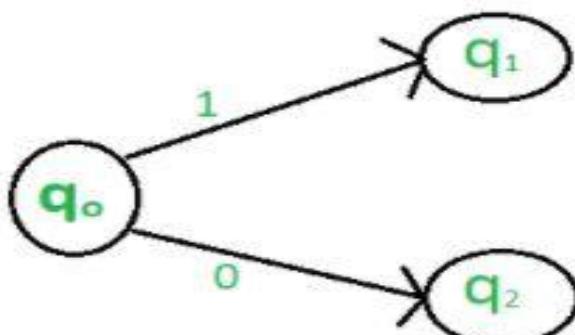
13.2 Main Categories of Algorithms

Often in computational theory, the term "algorithm" refers to a deterministic algorithm. A nondeterministic algorithm is different from its more familiar deterministic counterpart in its ability to arrive at outcomes using various routes.

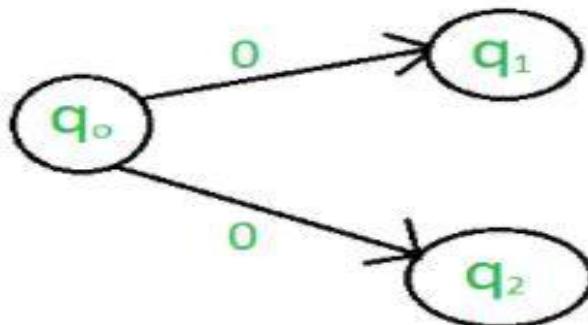
Unit 13: Interactable Problems

-
- 1) Deterministic algorithm
 - 2) Non-deterministic algorithm

Deterministic algorithm: A deterministic algorithm is an algorithm that, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states. Deterministic algorithms are by far the most studied and familiar kind of algorithm, as well as one of the most practical since they can be run on real machines efficiently. If at q_0 , input 0 is provided, we reach at q_1 and input 0 is provided, we reach at q_2 as shown below.



Non-Deterministic algorithm: A non-deterministic algorithm can provide different outputs for the same input on different executions. Unlike a deterministic algorithm which produces only a single output for the same input even on different runs, a non-deterministic algorithm travels in various routes to arrive at the different outcomes.



There are several ways an algorithm may behave differently from run to run.

- A **concurrent algorithm** can perform differently on different runs due to a race condition.
- A **probabilistic algorithm**'s behaviors depend on a random number generator.

An algorithm that solves a problem in nondeterministic polynomial time can run in polynomial time or exponential time depending on the choices it makes during execution. The nondeterministic algorithms are often used to find an approximation to a solution when the exact solution would be too costly to obtain using a deterministic one. If a deterministic algorithm represents a single path from an input to an outcome, a nondeterministic algorithm represents a single path stemming into many paths, some of which may arrive at the same output and some of which may arrive at unique outputs. This property is captured mathematically in "nondeterministic" models of computation such as the nondeterministic finite automaton. In some scenarios, all possible paths are allowed to run simultaneously. In algorithm design, nondeterministic algorithms are often used when the problem solved by the algorithm inherently allows multiple outcomes (or when there is a single outcome with multiple paths by which the outcome may be discovered, each equally preferable). Crucially, every outcome the nondeterministic algorithm produces are valid, regardless of which choices the algorithm makes while running.

A non-deterministic algorithm usually has two phases and output steps. The first phase is the guessing phase, which makes use of arbitrary characters to run the problem. The second phase is the verifying phase, which returns true or false for the chosen string. There are various related terms to non-deterministic algorithm:

- **choice(X)** : chooses any value randomly from the set X.

Algorithm Design and Analysis

- **failure()** : denotes the unsuccessful solution.
- **success()** : Solution is successful and current thread terminates.

Problem Statement : Search an element x on $A[1:n]$ where $n \geq 1$, on successful search return j if $a[j]$ is equals to x otherwise return 0.

Non-deterministic Algorithm:

```

1.      j= choice(a, n)
2.      if(A[j]==x) then
{
    write(j);
    success();
}
3. write(0);
failure();

```

13.3 Difference between Deterministic and Non-deterministic Algorithms

Deterministic algorithm	Non-Deterministic algorithm
For a particular input the computer will give always same output.	For a particular input the computer will give different output on different execution.
Can solve the problem in polynomial time.	Can't solve the problem in polynomial time.
Can determine the next step of execution.	Cannot determine the next step of execution due to more than one path the algorithm can take.

13.4 Detailed Difference between Deterministic and Non-deterministic Algorithms

Category	Deterministic algorithm	Non-Deterministic algorithm
Definition	The algorithms in which the result of every algorithm is uniquely defined are known as the Deterministic Algorithm. In other words, we can say that the deterministic algorithm is the algorithm that performs fixed number of steps and always get finished with an accept or reject state with the same result.	On other hand, the algorithms in which the result of every algorithm is not uniquely defined, and result could be random are known as the Non-Deterministic Algorithm.

Unit 13: Interactable Problems

Execution	In Deterministic Algorithms execution, the target machine executes the same instruction and results same outcome which is not dependent on the way or process in which instruction get executed.	On other hand in case of Non-Deterministic Algorithms, the machine executing each operation is allowed to choose any one of these outcomes' subjects to a determination condition to be defined later.
Type	On the basis of execution and outcome in case of Deterministic algorithm, they are also classified as reliable algorithms as for a particular input instruction the machine will give always the same output.	On other hand Nondeterministic algorithm are classified as non-reliable algorithms for a particular input the machine will give different output on different executions.
Execution Time	As outcome is known and is consistent on different executions so Deterministic algorithm takes polynomial time for their execution.	On other hand as outcome is not known and is non-consistent on different executions so non-Deterministic algorithm could not get executed in polynomial time.
Execution path	In deterministic algorithm the path of execution for algorithm is same in every execution.	On other hand in case of Non-Deterministic algorithm the path of execution is not same for algorithm in every execution and could take any random path for its execution.

13.5 Major Categories of Problems

- Tractable
- Intractable
- Decision
- Optimization

Tractable Problems

- Problems that can be solvable in a reasonable (polynomial) time. So what constitutes reasonable time?
- 1) On an input of size n the worst-case running time is $O(n^k)$ for some constant k
 - 2) $O(n^2), O(n^3), O(1), O(n \lg n), O(2^n), O(n^n), O(n!)$
 - 3) Polynomial time: $O(n^2), O(n^3), O(1), O(n \lg n)$
 - 4) Not in polynomial time: $O(2^n), O(n^n), O(n!)$

Optimization/Decision Problems

- Optimization Problems - An optimization problem is one which asks, "What is the optimal solution to problem X?". An optimization problem tries to find an optimal

Algorithm Design and Analysis

solution. Examples: 0-1 Knapsack problem, fractional knapsack problem and minimum spanning tree.

- Decision Problems - A decision problem is one with yes/no answer. Examples: Does a graph G have a MST of weight $\leq W$? A decision problem tries to answer a yes/no question. Many problems will have decision and optimization versions. Eg: Traveling salesman problem:

Optimization- find Hamiltonian cycle of minimum weight. Decision: is there a Hamiltonian cycle of weight $\leq k$.

The Class P

The class of problems that have polynomial-time deterministic algorithms. That is, they are solvable in $O(p(n))$, where $p(n)$ is a polynomial on n . A deterministic algorithm is (essentially) one that always computes the correct answer. Sample problems in P class are: fractional knapsack, MST and sorting problem.

The Class NP

NP stands for "Nondeterministic Polynomial-time". The class of decision problems that are solvable in polynomial time on a nondeterministic machine (or with a nondeterministic algorithm). A deterministic computer is what we know. A nondeterministic computer is one that can "guess" the right answer or solution. Think of a nondeterministic computer as a parallel machine that can freely spawn an infinite number of processes. Thus NP can also be thought of as the class of problems "whose solutions can be verified in polynomial time". Examples are traveling salesman, graph coloring and Satisfiability (SAT) problem.

P is the set of problems that can be solved in polynomial time - Examples: Fractional Knapsack. NP = set of problems for which a solution can be verified in polynomial time - Examples: Fractional Knapsack..., TSP, CNF SAT, 3-CNF SAT. So, clearly it is seen that $P \subseteq NP$.

The Class NP Hard

A lot of times you can solve a problem by reducing it to a different problem. I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B. "Easily" means "in polynomial time".

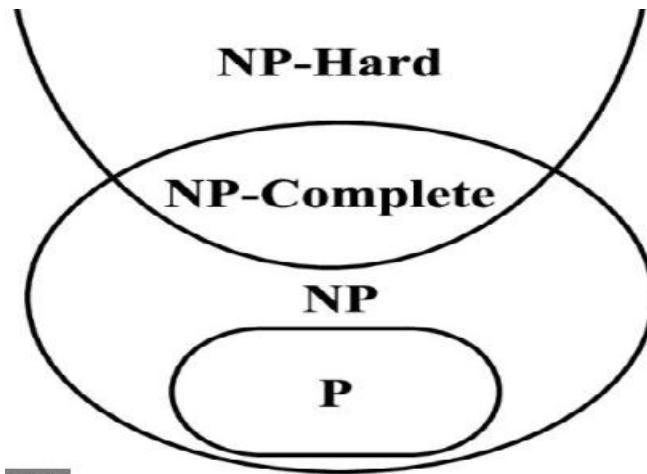
The Class NP Complete

A problem is NP-complete if the problem is both NP-hard, and NP. In computational complexity theory, a problem is **NP-complete** when a brute-force search algorithm can solve it, and the correctness of each solution can be verified quickly and the problem can be used to simulate any other problem with similar solvability. The name "**NP-complete**" is short for "**nondeterministic polynomial-time complete**". **Nondeterministic** refers to nondeterministic Turing machines, a way of mathematically formalizing the idea of a brute-force search algorithm. **Polynomial** time refers to an amount of time that is considered "quick" for a deterministic algorithm to check a single solution, or for a nondeterministic Turing machine to perform the whole search. **Complete** refers to the property of being able to simulate everything in the same complexity class. At present, all known algorithms for NP-complete problems require time that is superpolynomial in the input size, and it is unknown whether there are any faster algorithms. The following techniques can be applied to solve computational problems in general, and they often give rise to substantially faster algorithms:

- **Approximation:** Instead of searching for an optimal solution, search for a solution that is at most a factor from an optimal one.
- **Randomization:** Use randomness to get a faster average running time and allow the algorithm to fail with some small probability. Note: The Monte Carlo method is not an example of an efficient algorithm in this specific sense, although evolutionary approaches like Genetic algorithms may be.

Unit 13: Interactable Problems

- **Restriction:** By restricting the structure of the input (e.g., to planar graphs), faster algorithms are usually possible.
- **Parameterization:** Often there are fast algorithms if certain parameters of the input are fixed.
- **Heuristic:** An algorithm that works "reasonably well" in many cases, but for which there is no proof that it is both always fast and always produces a good result. Metaheuristic approaches are often used.



So, based upon the discussion the problems are:

- **P** – Problems that can be decided in polynomial time.
- **NP** – Problems that can be verified in polynomial time.
- **NP-hard** – Problems to which anything in NP can be reduced in polynomial time.
- **NP-complete** – Problems in both NP and NP-hard.

We have been discussing about efficient algorithms to solve complex problems, like **shortest path**, **Euler graph**, **minimum spanning tree** etc. Those were all success stories of algorithm designers. But there are some failures too. There are computational problems that cannot be solved by algorithms even with unlimited time. For example, **Turing Halting problem** (Given a program and an input, whether the program will eventually halt when run with that input, or will run forever). Alan Turing proved that general algorithm to solve the halting problem for all possible program-input pairs cannot exist. Status of NP Complete problems is another failure story, NP complete problems are problems whose status is unknown. No polynomial time algorithm has yet been discovered for any NP complete problem, nor has anybody yet been able to prove that no polynomial-time algorithm exists for any of them. The interesting part is, if any one of the NP complete problems can be solved in polynomial time, then all of them can be solved.

A decision problem L is NP-complete if:

- 1) L is in NP (Any given solution for NP-complete problems can be verified quickly, but there is no efficient known solution).
- 2) Every problem in NP is reducible to L in polynomial time (Reduction is defined below).

How to prove that a Given Problem is NP Complete?

From the definition of NP-complete, it appears impossible to prove that a problem L is NP-Complete. It requires us to show every problem in NP is polynomial time reducible to L. Fortunately, there is an alternate way to prove it. The idea is to take a known NP-Complete problem and reduce it to L. If polynomial time reduction is possible, we can prove that L is NP-Complete by transitivity of reduction (If a NP-Complete problem is reducible to L in polynomial time, then all problems are reducible to L in polynomial time). Fortunately, there is an alternate way to prove it. The idea is to take a known NP-Complete problem and reduce it to L. If polynomial time reduction is possible, we can prove that L is NP-Complete by transitivity of reduction (If a NP-

Algorithm Design and Analysis

Complete problem is reducible to L in polynomial time, then all problems are reducible to L in polynomial time).

13.6 Boolean Satisfiability Problem

Boolean Satisfiability or simply SAT is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.

- **Satisfiable:** If the Boolean variables can be assigned values such that the formula turns out to be TRUE, then we say that the formula is satisfiable.
- **Unsatisfiable:** If it is not possible to assign such values, then we say that the formula is unsatisfiable.



Examples are: $F = A \wedge B$ (A AND B) is satisfiable and $G = A \wedge A'$ (A AND A') is unsatisfiable

Boolean Satisfiability problem is NP Complete

- 2-SAT is a special case of Boolean Satisfiability Problem and can be solved in polynomial time.

Important Term (CNF)

CNF: CNF is a conjunction (AND) of clauses, where every clause is a disjunction (OR). Now, 2-SAT limits the problem of SAT to only those Boolean formula which are expressed as a CNF with every clause having only 2 terms(also called 2-CNF).**Example:** $(A_1 \text{ OR } B_1) \text{ AND } (A_2 \text{ OR } B_2) \text{ AND } (A_3 \text{ OR } B_3)$ Thus, Problem of 2-Satisfiability can be stated as:

Given CNF with each clause having only 2 terms, is it possible to assign such values to the variables so that the CNF is TRUE?

Example:

- Input : $F = (x_1 \vee x_2) \wedge (x_2 \vee x_1) \wedge (x_1 \vee x_2)$
Output: The given expression is satisfiable.
(for $x_1 = \text{FALSE}$, $x_2 = \text{TRUE}$)
- Input : $F = (x_1 \vee x_2) \wedge (x_2 \vee x_1') \wedge (x_1 \vee x_2') \wedge (x_1' \vee x_2')$
Output: The given expression is unsatisfiable.
(for all possible combinations of x_1 and x_2)

Approach for 2-SAT Problem

For the CNF value to come TRUE, value of every clause should be TRUE. Let one of the clause be $(A \vee B)$ be true. If $A = 0$, B must be 1 i.e. $(A' \Rightarrow B)$. If $B = 0$, A must be 1 i.e. $(B' \Rightarrow A)$. Thus, $(A \vee B) = \text{TRUE}$ is equivalent to $(A' \Rightarrow B) \wedge (B' \Rightarrow A)$. Now, we can express the CNF as an Implication. So, we create an Implication Graph which has 2 edges for every clause of the CNF. $(A \vee B)$ is expressed in implication Graph as edge $(A \rightarrow B)$ edge $(B \rightarrow A)$. Thus, for a Boolean formula with 'm' clauses, we make an Implication Graph with: 2 edges for every clause i.e. '2m' edges. 1 node for every Boolean variable involved in the Boolean formula. 3-SAT Problem: $(x_1 \vee x_2 \vee x_3) \wedge (x_2' \vee x_3 \vee x_4) \wedge (x_1' \vee x_2 \vee x_5)$. So, the problem is NP-hard.

Summary

- P is set of problems that can be solved by a deterministic Turing machine in Polynomial time.
- NP is set of problems that can be solved by a Non-deterministic Turing Machine in Polynomial time.

Unit 13: Interactable Problems

- P is subset of NP (any problem that can be solved by deterministic machine in polynomial time can also be solved by non-deterministic machine in polynomial time) but $P \neq NP$.
- Some problems can be translated into one another in such a way that a fast solution to one problem would automatically give us a fast solution to the other.
- There are some problems that every single problem in NP can be translated into, and a fast solution to such a problem would automatically give us a fast solution to every problem in NP. This group of problems are known as NP-Complete. Ex:- Clique.
- A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.

Keywords

- **Polynomial-Time Algorithm:** An algorithm whose order-of-magnitude time performance is bounded from above by a polynomial function of n , where n is the size of its inputs.
- **Exponential Algorithm:** An algorithm whose order-of-magnitude time performance is not bounded from above by a polynomial function of n .
- **Tractable Problem:** A problem that is solvable by a polynomial-time algorithm. The upper bound is polynomial.
- **Intractable Problem:** A problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential.
- **Concurrent algorithm:** A concurrent algorithm can perform differently on different runs due to a race condition.
- **Probabilistic algorithm:** Probabilistic algorithm's behaviors depends on a random number generator.
- **P Problems:** The problems that can be decided in polynomial time.
- **NPPProblems:** The problems that can be verified in polynomial time.
- **NP-hard Problems:** The problems to which anything in NP can be reduced in polynomial time.
- **NP-completeProblems:** The problems in both NP and NP-hard.

SelfAssessment

1. Which of these words refer to the property of being able to simulate everything in the same complexity class?
 - Non-deterministic
 - Polynomial
 - Complete
 - None of the above
2. The set of problems that can be solved by deterministic machine in Polynomial time is
 - P class
 - NP class
 - NP Hard class
 - NP Complete class
3. The set of problems that can be solved by non- deterministic machine in Polynomial time is
 - P class

Algorithm Design and Analysis

- B. NP class
 - C. NP Hard class
 - D. NP Complete class
4. Which of these is the first NP-Complete problem?
- A. Boolean Satisfiability problem
 - B. Linear Searching
 - C. Matrix multiplication
 - D. None of the above
5. In computational theory, the term algorithm generally refers to a _____ algorithm.
- A. Deterministic
 - B. Non-Deterministic
 - C. Un-deterministic
 - D. None of the above
6. Which of these represents the non-deterministic behavior?
- A. Concurrent algorithm
 - B. Probabilistic algorithm
 - C. Both of the above
 - D. None of the above
7. In which algorithm the behavior of the algorithm depends upon a random number generator?
- A. Concurrent algorithm
 - B. Probabilistic algorithm
 - C. Both above
 - D. None of the above
8. Which of these are related to non-deterministic algorithms?
- A. Guessing stage
 - B. Verifying stage
 - C. Both above stages
 - D. None of the above
9. The major categories of problems are
- A. Tractable and intractable problems
 - B. Decision problems
 - C. Optimization problems
 - D. All of the above
10. Which of these represents the exponential function?
- A. $O(n)$

Unit 13: Interactable Problems

- B. $O(n^2)$
 C. $O(\log n)$
 D. $O(2^n)$

11. Find the odd one out.

- A. $O(\log n)$
 B. $O(n)$
 C. $O(n!)$
 D. $O(n \log n)$

12. Which of these problems is solved in polynomial time?

- A. Tractable problems
 B. Intractable problems
 C. Distract-able problems
 D. None of these

13. For which kind of problems, the lower bound is exponential?

- A. Tractable problems
 B. Intractable problems
 C. Distract-able problems
 D. None of these

14. 'Sorting a list' problem is a kind of _____

- A. Tractable problems
 B. Intractable problems
 C. Distract-able problems
 D. None of these

15. Which of these is an intractable problem?

- A. Travelling salesman problem
 B. Halting problem
 C. Both above
 D. None of the above

Answers for Self Assessment

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

Review Questions

1. What is the classification of classes of functions? Give few examples.

Algorithm Design and Analysis

2. What is an algorithm? Explain its classifications.
3. What is the difference between deterministic and non-deterministic algorithms?
4. What is a non-deterministic algorithm? What is its use? Explain its phases and related terms.
5. Write the detailed difference between deterministic and non-deterministic algorithms.
6. What are four classes of problems? Explain their relations.



Further Readings

<https://www.britannica.com/technology/intractable-problem>

<https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/lectures/19/Small19.pdf>

Unit 14: More on Interactable Problems

CONTENTS

- Objectives
- Introduction
- 14.1 Polynomial Time Algorithm
- 14.2 Non-Polynomial Time Algorithm
- 14.3 Satisfiability Problem
- 14.4 Cook's Theorem
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the examples of NP Hard and NP Complete problems
- Understand the Cook's theorem
- Understand the reduction of problems

Introduction

Based upon the time complexity, the problems can be divided into two categories:

- 1) Polynomial time
- 2) Non-polynomial time

14.1 Polynomial Time Algorithm

There are various problems which can be solved in polynomial time. These are known as polynomial time algorithm. These are:

- 1) Linear search = n
- 2) Binary search = $\log n$
- 3) Insertion sort = n^2
- 4) Merge sort = $n \log n$
- 5) Matrix multiplication = n^3

14.2 Non-Polynomial Time Algorithm

There are various problems which we can't solve in polynomial time. So that is why these are known as non-polynomial time algorithm. These are:

- 1) 0/1 knapsack = 2^n

Algorithm Analysis and Design

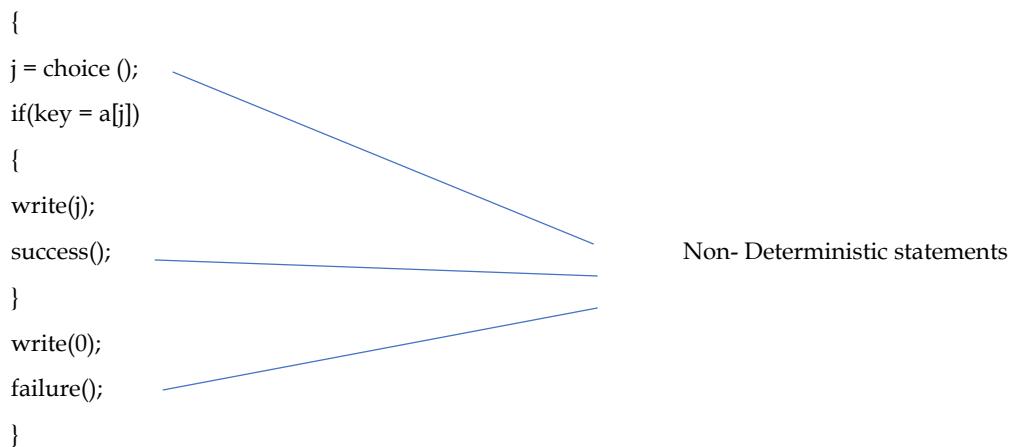
-
- 2) TSP = 2^n
 - 3) Sum of subset = 2^n
 - 4) Graph colouring = 2^n
 - 5) Hamiltonian Cycle = 2^n

Two important considerations in this are there is no solution so far by which we can solve the problem in polynomial time. If we cannot solve these problems in polynomial time, atleast we need to find the similarity between the problems, so that if we are able to find the solution of one problem, then we easily we can solve other problems as well. As we are not able to write the polynomial time deterministic algorithm for these problems, we can write polynomial time non-deterministic algorithm.

Non-Deterministic Search Algorithm

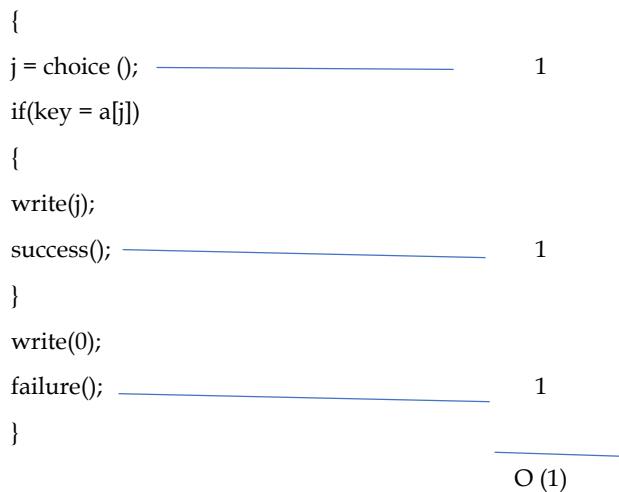
```
algorithm Nsearch (a, n, key)
```

```
{
j = choice ();
if(key = a[j])
{
write(j);
success();
}
write(0);
failure();
}
```



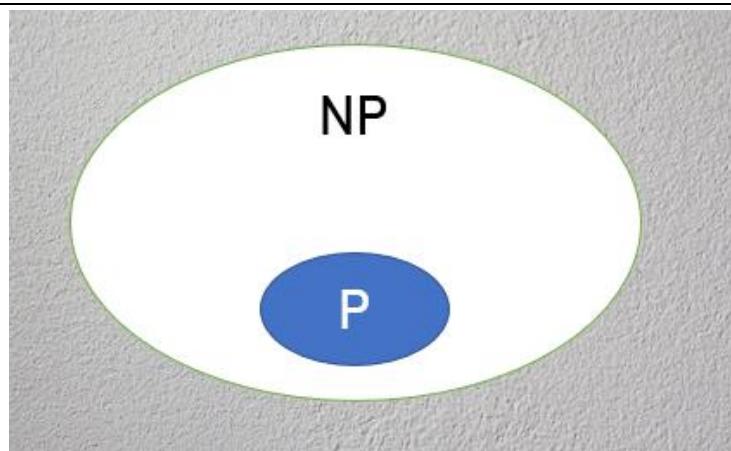
```
algorithmNsearch (a, n, key)
```

```
{
j = choice (); 1
if(key = a[j])
{
write(j);
success(); 1
}
write(0);
failure(); 1
}
```



The non - deterministic part of algorithm is how the choice came to know about the position of the element in the array?

P is the set of deterministic algorithms which takes polynomial time. Examples are linear search, binary search, bubble sort, merge sort, single source shortest path, minimum cost spanning tree, optimal merge pattern, Huffman coding etc. NP is the set of non - deterministic algorithms which takes polynomial time. P is a subset of NP; it means that the deterministic algorithms were a part of non-deterministic algorithms.



To find out the relationship between problems, a base problem is required here so that we can relate the problem. Here the base problem here is CNF - Satisfiability.

14.3 Satisfiability Problem

Boolean Satisfiability or simply **SAT** is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.

Satisfiable: If the Boolean variables can be assigned values such that the formula turns out to be TRUE, then we say that the formula is satisfiable.

Unsatisfiable: If it is not possible to assign such values, then we say that the formula is unsatisfiable.



Example

- $X_i = \{ X_1, X_2, X_3 \}$
- $CNF = (X_1 \vee X_2 \vee X_3) \wedge (X_1' \vee X_2' \vee X_3)$

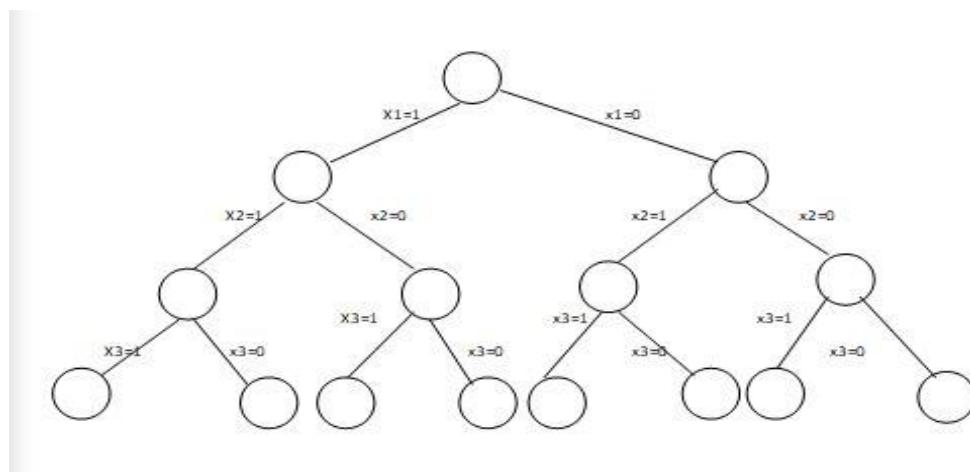
X1	X2	X3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0

1

1

1

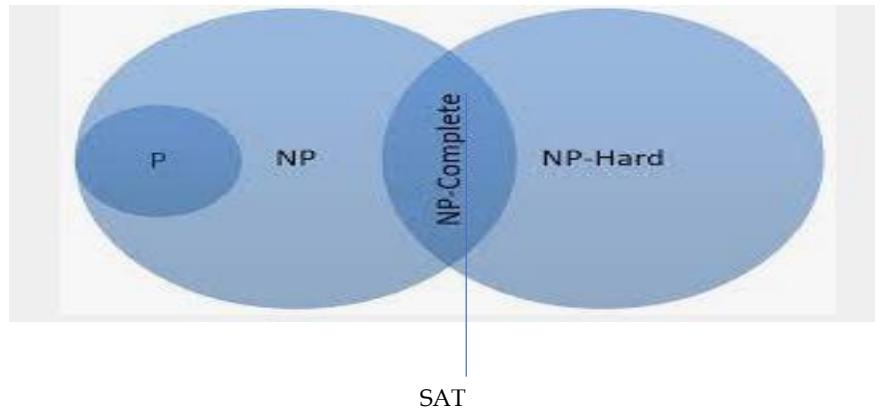
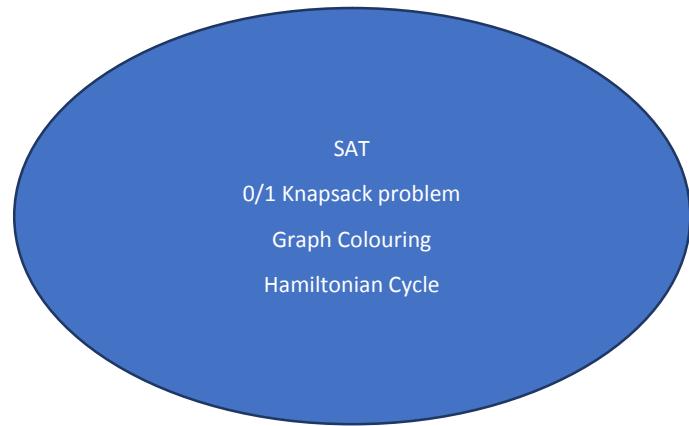
State Space Tree



Relationship between problems is here we need to show that if Satisfiability problem can be solved in polynomial time, all the problems can be solved in polynomial time. Examples: 0/1 knapsack, TSP, sum of subset, graph colouring and Hamiltonian cycle.

Reduction

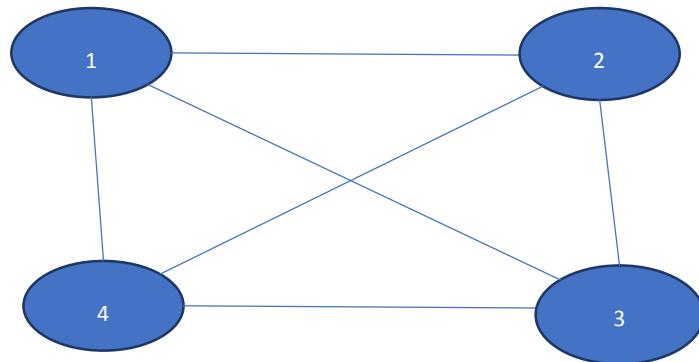
These all are NP Hard Problem



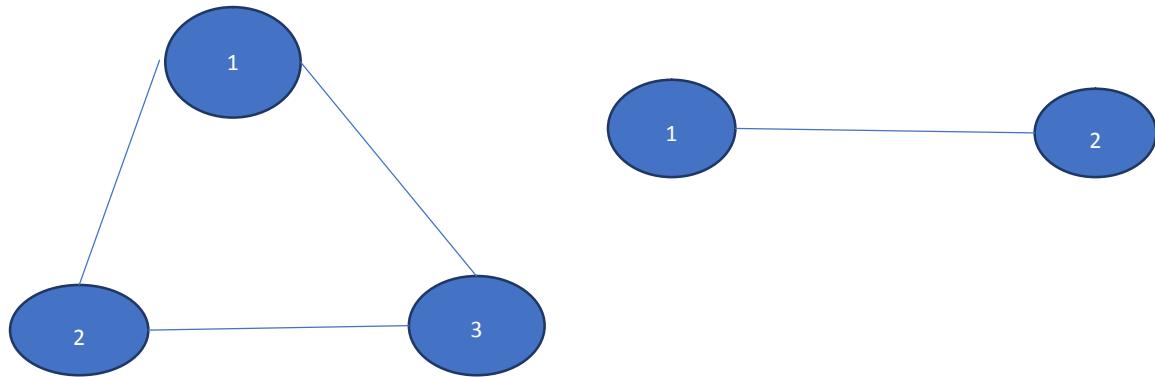
NP-Hard Graph Problem

Clique Decision Problem (CDP): This is a NP-Hard graph problem.

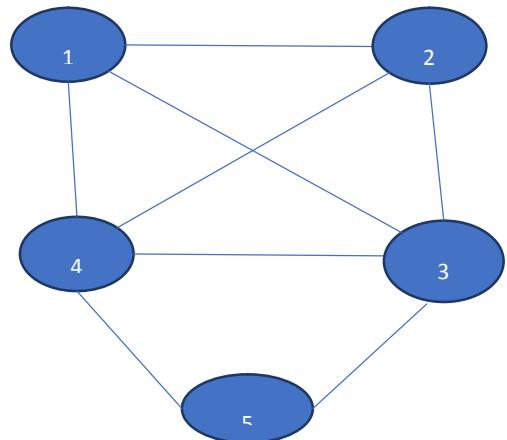
Complete graph



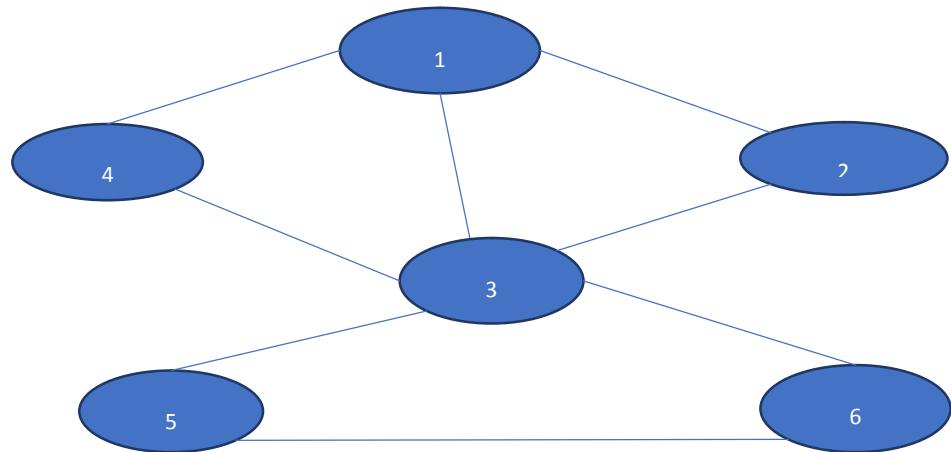
One property of complete graph is if you take, number of vertices as n , then the number of edges will be $n*(n-1)/2$.



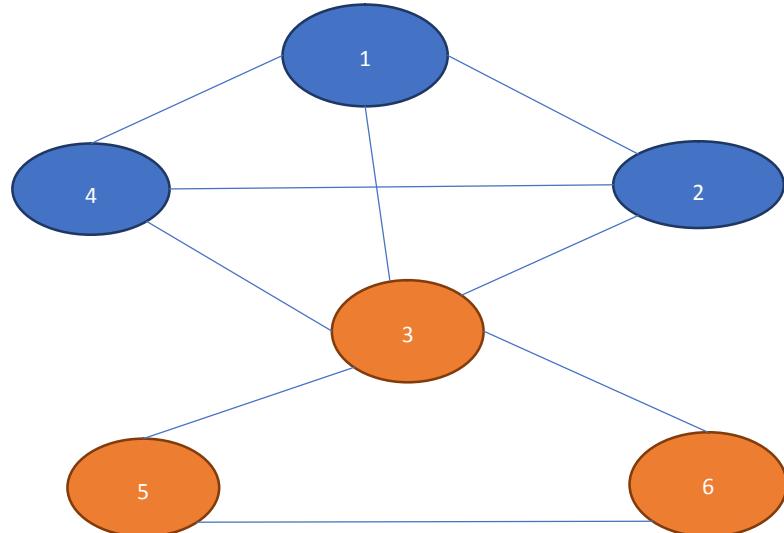
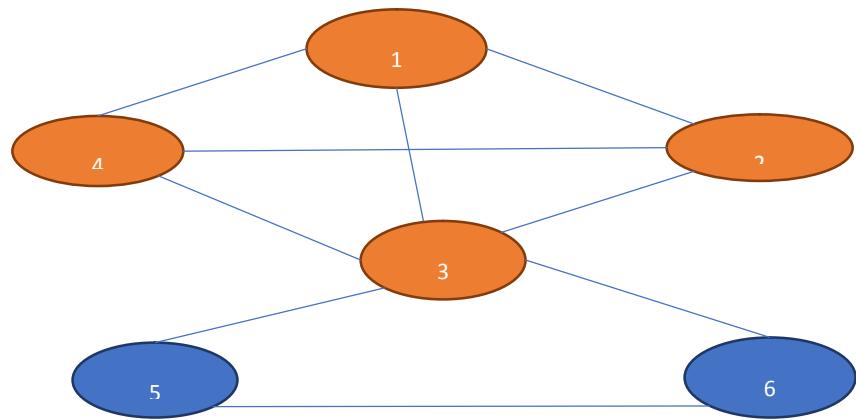
Clique graph

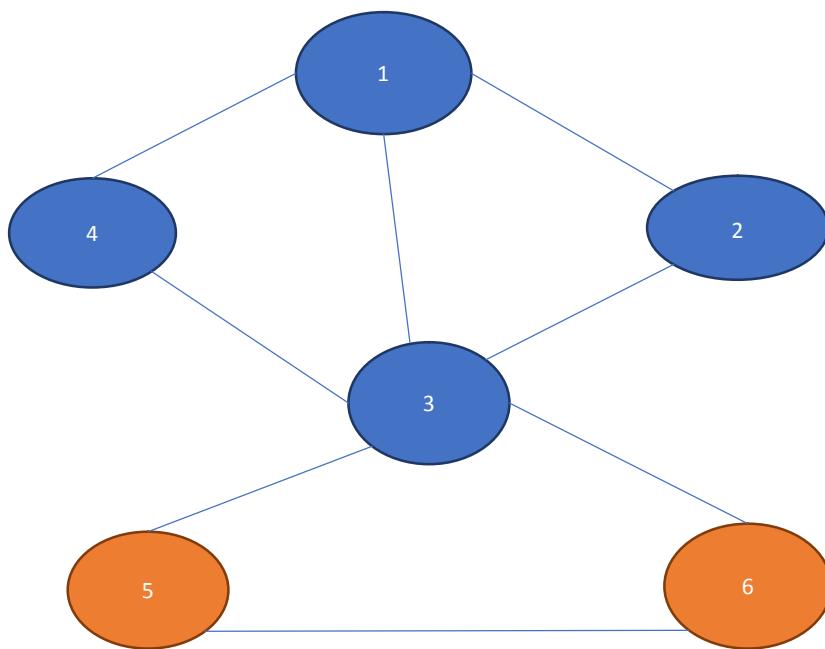


Add one more vertex to the graph, i.e., vertex 5. Connect it to some of the vertices. Check is this is a complete graph? One of the sub-graphs of this must be a complete graph.



Cliques available are





There are multiple cliques available in the graph. The maximum clique is of size 4.

14.4 Cook's Theorem

In computational complexity theory, the **Cook-Levin Theorem**, also known as **Cook's Theorem**, states that the Boolean Satisfiability problem is NP-complete. That is, it is in NP, and any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the Boolean Satisfiability problem. The theorem is named after Stephen Cook and Leonid Levin. An important consequence of this theorem is that if there exists a deterministic polynomial time algorithm for solving Boolean Satisfiability, then every NP problem can be solved by a deterministic polynomial time algorithm.

History

Stephen Arthur Cook and L.A. Levin in 1973 independently proved that the **Satisfiability problem(SAT)** is NP-complete. Stephen Cook, in 1971, published an important paper titled 'The complexity of Theorem Proving Procedures', in which he outlined the way of obtaining the proof of an NP-complete problem by reducing it to SAT. He proved **Circuit-SAT** and **3CNF-SAT** problems are as hard as **SAT**. Similarly, Leonid Levin independently worked on this problem in the then Soviet Union. The proof that **SAT** is NP-complete was obtained due to the efforts of these two scientists. Later, Karp reduced 21 optimization problems, such as Hamiltonian tour, vertex cover, and clique, to the **SAT** and proved that those problems are NP-complete.

Problem

Hence, an **SAT** is a significant problem and can be stated as follows: Given a boolean expression F having n variables x_1, x_2, \dots, x_n , and Boolean operators, is it possible to have an assignment for variables true or false such that binary expression F is true? This problem is also known as the **formula - SAT**. An **SAT(formula-SAT or simply SAT)** takes a Boolean expression F and checks whether the given expression(or formula) is satisfiable. A Boolean expression is said to be satisfactory for some valid assignments of variables if the evaluation comes to be true.

Terminologies of a Boolean Expression

- **Boolean variable:** A variable, say x , that can have only two values, true or false, is called a Boolean variable
- **Literal:** A literal can be a logical variable, say x , or the negation of it, that is \bar{x} or x ; x is called a positive literal, and \bar{x} is called the negative literal
- **Clause:** A sequence of variables(x_1, x_2, \dots, x_n) that can be separated by a logical **OR** operator is called a clause. For example, $(x_1 \vee x_2 \vee x_3)$ is a clause of three literals.
- **Expressions:** One can combine all the preceding clauses using a Boolean operator to form an expression.
- **CNF form:** An expression is in CNF form(conjunctive normal form) if the set of clauses are separated by an **AND** (\wedge), operator, while the literals are connected by an **OR** (\vee) operator. The following is an example of an expression in the CNF form: $f = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2)$
- **3 - CNF:** An expression is said to be in 3-CNF if it is in the conjunctive normal form, and every clause has exact three literals.

Thus, an **SAT** is one of the toughest problems, as there is no known algorithm other than the brute force approach. A brute force algorithm would be an exponential-time algorithm, as 2^n possible assignments need to be tried to check whether the given Boolean expression is true or not. Stephen Cook and Leonid Levin proved that the SAT is NP-complete.

Types of SAT

- 1) **Circuit- SAT:** A circuit-SAT can be stated as follows: given a Boolean circuit, which is a collection of gates such as **AND**, **OR**, and **NOT**, and n inputs, is there any input assignments of Boolean variables so that the output of the given circuit is true? Again, the difficulty with these problems is that for n inputs to the circuit, 2^n possible outputs should be checked. Therefore, this brute force algorithm is an exponential algorithm and hence this is a hard problem.
- 2) **CNF-SAT:** This problem is a restricted problem of **SAT**, where the expression should be in a conjunctive normal form. An expression is said to be in a conjunction form if all the clauses are connected by the Boolean **AND** operator. Like in case of a **SAT**, this is about assigning truth values to n variables such that the output of the expression is true.
- 3) **3-CNF-SAT(3-SAT):** This problem is another variant where the additional restriction is that the expression is in a conjunctive normal form and that every clause should contain exactly three literals. This problem is also about assigning n assignments of truth values to n variables of the Boolean expression such that the output of the expression is true. In simple words, given an expression in 3-CNF, a 3-SAT problem is to check whether the given expression is satisfiable.

Reduction

We take two problems, say Satisfiability problem and 0/1 Knapsack problem. Satisfiability

problem \propto 0/1 Knapsack problem. $I_1 \propto I_2$. If any algorithm can solve the problem of Satisfiability in polynomial time, then 0/1 Knapsack problem can also be solved and vice-versa. The conversion from one problem to another also takes polynomial time. Satisfiability problem, if this is proved,

then L is also NP Hard problem. If $L \propto L_1$, then L_1 is also NP hard

problem. So, Satisfiability problem $\propto L$. This Satisfiability problem has non-deterministic polynomial time algorithm. So, Satisfiability problem is both NP-Hard and NP-Complete problem.

Summary

- Based upon the time complexity, the problems can be divided into two categories: polynomial time and non-polynomial time.
- It is required to show that if Satisfiability problem can be solved in polynomial time, all the problems can be solved in polynomial time. Examples: 0/1 knapsack, TSP, Sum of subset, Graph colouring and Hamiltonian Cycle.
- Clique Decision Problem (CDP) is a NP-Hard graph problem.
- Karp reduced 21 optimization problems, such as Hamiltonian tour, vertex cover, and clique, to the **SAT** and proved that those problems are NP-complete.
- NP Class is the set of non - deterministic algorithms which takes polynomial time.

Keywords

- **P Class:** It is the set of deterministic algorithms which takes polynomial time. Examples: linear search, binary search, bubble sort, merge sort, single source shortest path, minimum cost spanning tree, optimal merge pattern, huffman coding etc.
- **0/1 knapsack Problem:** Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.
- **Travelling Salesman Problem:** Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?.
- **Cook's theorem,** states that the Boolean Satisfiability problem is NP-complete.
- **Boolean variable:** A variable, say x , that can have only two values, true or false, is called a boolean variable
- **Literal:** A literal can be a logical variable, say x , or the negation of it, that is x or \bar{x} ; x is called a positive literal, and \bar{x} is called the negative literal
- **3 - CNF:** An expression is said to be in 3-CNF if it is in the conjunctive normal form, and every clause has exactly three literals.

Self Assessment

1. Which of these are polynomial time taking problems?
 - A. Insertion sort
 - B. Linear and binary search
 - C. Matrix multiplication
 - D. All of the above
2. Which of these is not a non-polynomial time taking algorithm?
 - A. Sum of subset
 - B. Graph coloring
 - C. Merge sort
 - D. TSP
3. Which of these problems falls under P class category?
 - A. Huffman encoding
 - B. Optimal merge pattern
 - C. Single source shortest path

Algorithm Analysis and Design

- D. All of the above
4. Which of these represents the base problem?
- A. TSP
 - B. Graph coloring
 - C. SAT
 - D. None of the above
5. Satisfiability problem comes under
- A. NP-Hard
 - B. NP-Complete
 - C. Both of the above
 - D. None of the above
6. Cook's theorem states that Boolean Satisfiability problem is
- A. P problem
 - B. NP problem
 - C. NP Hard problem
 - D. NP Complete problem
7. A Boolean variable has
- A. 0
 - B. 1
 - C. Either 0 or 1
 - D. None of the above
8. A literal holds the value
- A. X
 - B. X'
 - C. Either of the above
 - D. None of the above
9. In a CNF expression, the set of clauses can be separated by _____ operator.
- A. OR
 - B. AND
 - C. NOT
 - D. None of the above
10. Which of these represents the variant of SAT?
- A. Circuit - SAT
 - B. CNF - SAT
 - C. 3CNF-SAT
 - D. All of the above

Unit 14: More on Interactable Problems

11. Which of these are polynomial time taking problems?
- Insertion sort
 - Linear and binary search
 - Matrix multiplication
 - All of the above
12. Which of these is not a non-polynomial time taking algorithm?
- Sum of subset
 - Graph coloring
 - Merge sort
 - TSP
13. Which of these problem falls under P class category?
- Huffman encoding
 - Optimal merge pattern
 - Single source shortest path
 - All of the above
14. Which of these represents the base problem?
- TSP
 - Graph coloring
 - SAT
 - None of the above
15. Satisfiability problem comes under
- NP-Hard
 - NP-Complete
 - Both of the above
 - None of the above

Answers for Self Assessment

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

Review Questions

- How can we categorize the problems and corresponding algorithms based upon time complexity? Explain.
- What is Satisfiability problem? Take one example and draw its state space tree. How can we solve the problem by finding the relationship between problems?
- Explain how clique decision problem is NP hard problem.
- What is Cook's theorem? Explain its history and problem.

Algorithm Analysis and Design

5. Write the terminologies of Boolean expressions for Boolean satisfiability problem.
6. Explain the types of Boolean Satisfiability problem.
7. How can we reduce the problem by finding out relationship between problems? Give example.



Further Readings

<https://www.geeksforgeeks.org/cook-levin-theorem-or-cooks-theorem/>

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_cooks_theorem.htm

<https://www3.cs.stonybrook.edu/~algorith/video-lectures/1997/lecture25.pdf>

LOVELY PROFESSIONAL UNIVERSITY

Jalandhar-Delhi G.T. Road (NH-1)

Phagwara, Punjab (India)-144411

For Enquiry: +91-1824-521360

Fax.: +91-1824-506111

Email: odl@lpu.co.in

ISBN 978-81-19334-37-7



9 788119 334377