Design And Analysis

Algorithm

The name algorithm is given by the name by Abu Jafar Muhammad & ibn musa Al-khowarizmi of 9th century is defined as follows.

An algo. is a set of rules that carry out calculations either by hand or machines.

An alg. is a finite set of instruction, if followed accomplishes a particular task

An alg. is a set of rules that must be followed when solving a specific pblm.

All alg. must satisfy the following criteria:

1. Input: Zero or more quantities are externely supplied
2. Output: Atleast one quantity is produced.
3. Definiteness: ~~to clear~~ Each instruction is clear and unambiguous.
4. Finiteness: Algo. must terminate after a finite no. of steps
5. Effectiveness: Every instruction must be very basic so that it can be carried out by a person using pencil & paper

Study of Algorithm:

1. Design of Algo.
2. Alg. validation
3. Analysis of alg
4. Testing of alg.

Analysis of algo.

It depends on various factors such as memory, communication bandwidth or computer h/w but the most often used is the computational time that an alg. requires for completing the given task. In the RAM model all executions instructions are executed sequentially one after another with no concurrence operations
By counting the no. of steps the runtime of an alg. is calculated.

The analysis of an alg. evaluate the performance of an alg. based on the given models and matrix

1 Input Size
2 Running time:
- No. of operations or steps executed
3 Order of growth: To calculate the growth rate of a running time, we consider only the leading term of the time formed.

Eg: $5n^2 + 3n + 2$
→ leading term n^2

There are many criteria to judge an algo.
i) Does it do what we want it to do.
ii) Does it work correctly according to the original specifications of the task
iii) Is there documentation that describes how to use it and how it works
iv) Are procedures created in such a way that they perform logical sum function
v) Is the code readable.

There are other criteria for judging algo. that have a direct relationship with the perfm. these have to do with their computing time and storage requirements

The analysis of an alg. focuses on time and space complexity. The space complexity of an alg. is the amount of memory it needs to run to completion. The time complexity of an alg. is the amount of computer tym it needs to run to completion

Performance evaluation can be loosely divided into 2 major phases.
(i) Priorgi
(ii) Posteriori Performance Analysis

Space Complexity

The space needed will be the sum of the foll. compts
i) A fixed part ie indpt of the characteristics of the input and output. The part typically includes the instructions space for simple variables & fixed size cmpt variables, space for constants and so on...
ii) A variable part that consists of the space needed by the cmpt variables whose size is dependent on the particular pblm instance being solved. The space needed by referenced

variable and recursion stack space

The space requirement $S(P)$ of any alg. P may ∴ be written as:

$$S(P) = C + S_p$$

where $C \rightarrow$ constant

Time Complexity

The time $T(P)$ taken by a pgm P is the sum of the compile time and run time.

The compile tym doesnot depends on the instance characteristics, run tym is denoted by t_p

If we knew the characteristics of the compiler to be used we could proceed to determine the no. of additions, subtractions, multiplications, division, compares, load, stored and so on.. that would be made by the code for P.

$$t_p(n) = C_a ADD(n) + C_s SUB(n) + C_m MUL(n) + \cdots$$

where $n \rightarrow$ instance characteristics

$C_a, C_s, C_m \cdots \rightarrow$ time needed for addition, subtraction, multiplication etc...

ADD, SUB, MUL are the fns whose values are no. additions, subtraction, multiplication that are performed when the code for P is used

Possible time complexity of an alg. are:

i) Best case time complexity

Min amount of tym that an alg. requires for an input of size n. Thus it is the fn defined by the min. no. of steps taken on any instance of size n

ii) Avg case

It is the execution of an alg. having typical input data of size n. Thus it is the fn defined by the avg. no. of steps taken on any instance of size n

iii) Worst case

Is the fn defined by the max. amount of tym needed by an alg. for an input of size n. Thus it is the fn defined by the max. no. of steps taken on any instance of size n.

Exact Analysis Rule

Alg. For computing the n Fibonacci Number

```
1  : Procedure Fibonacci
2  : Read (n)
3  : If n<0 then
4  : Print 'error' stop.
5  : If n=0, then
6  : Print '0' stop.
7  : If n=1, then
8  : Print '1' stop.
9  : f2=0 , f1=1
10 : for i=2 to n do,
11 : fn = f1+f2
12 : f2 = f1
13 : f1 = fn
14 : end.
15 : Print f(n)
16 : End Fibonacci
```

Step	n<0	n=0	n=1	n>1
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	0	0	0
5	0	1	1	1
6	0	1	0	0
7	0	0	1	1
8	0	0	0	0
9	0	0	0	2
10	0	0	0	n-2+2
11	0	0	0	n-1
12	0	0	0	n-1
13	0	0	0	n-1
14	0	0	0	n-1
15	0	0	0	1
16	1	1	1	1
	5	6	7	9-4+5n = 5n+5

Q Find an alg. for the sum of digit of number.

1 : Procedure sum of digits
2 : Read (num)
3 : sum = 0, mod
4 : while num > 0
5 : mod = num % 10
6 : sum = sum + mod
7 : num = num / 10
8 : end
9 : print sum
10 : End sum of digit

Step	num > 0
1	1
2	1
3	1
4	n
5	n
6	n
7	n
8	n-1
9	1
10	1
	5n+1

12/10/21 Tuesday

ASYMPTOTIC NOTATIONS

Is a short hand way to write down and talk about fastest possible and slowest possible running tym for an algorithm using high & low bounds on speed.

Consider the 2 fns 'f' and 'g'. These fns are from $N \to R^+$, $N \to$ set of natural nos
$R^+ \to$ set of +ve real nos

Informally it state the relationships b/w the orders of fn : $\Omega(g)$, $\Theta(g)$,

1) $\Omega(g)$: Fn that grows atleast as fast as g
2) $\Theta(g)$: Fn that grow at the same rate as that of g

ii) $O(g)$: Fn that grow no faster than g (maximum)

i) Big Oh (O)

The fn $f(n) = O(g(n))$, iff $\exists$ +ve const c and n_0 s.t. $f(n) \leq c \times g(n)$ $\forall n, n \geq 0$

Big Oh is a formal method of expressing the UB of an algorithm's running time. It is the measure of the largest amount of time, it could possibly take for the alg. to complete.

ii) Omega (Ω)

The fn $f(n) = \Omega(g(n))$, iff $\exists$ +ve const c and n_0 s.t.

$f(n) \geq c \times g(n)$ $\forall n, n \geq 0$

This is almost same as Big Oh, except that $f(n) \geq g(n)$, this makes $g(n)$ a lower bound fn instead of an UB fn. It describe the best that can happen for a given data size.

iii) Theta (Θ)

The fn $f(n) = \Theta(g(n))$, iff $\exists$ +ve const c_1, c_2 and n_0 s.t.

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ $\forall n, n \geq 0$

The lower and upper bound for the fn is provided by the 'Θ' notation.

For non-negative fns $f(n)$ and $g(n)$ $\exists$ an integer n_0 and +ve const c_1 and c_2 i.e. $c_1 \geq 0$ $c_2 \geq 0$ s.t. all integers $n \geq n_0$

Common Asymptotic fns are :

i) Constant - 1
ii) Logarithmic - $\log n$
iii) Linear - n
iv) Quadratic - n^2
v) Exponential - 2^n
vi) Factorial - $n!$
vii) Cubic - n^3

20/10/2021

Divide & Conquer

→ Divide and Conquer is a technique for designing algorithms.

→ Consists of decomposing the instance to be solved into a no. of smaller sub instances of the pblm.

→ Solving successively and independently each of these sub instances

→ Combining the sub ~~instances~~ solns, thus obtained to obtain the soln of the original instance

The most-well known algorithm design strategy:

1. Divide instance of pblm into two or more small instances.
2. Solve smaller instances recursively
3. Obtain soln to original (larger) instance by combining these soln

- Breaking large pblms into smaller sub pblms

Divide and Conquer Paradigm

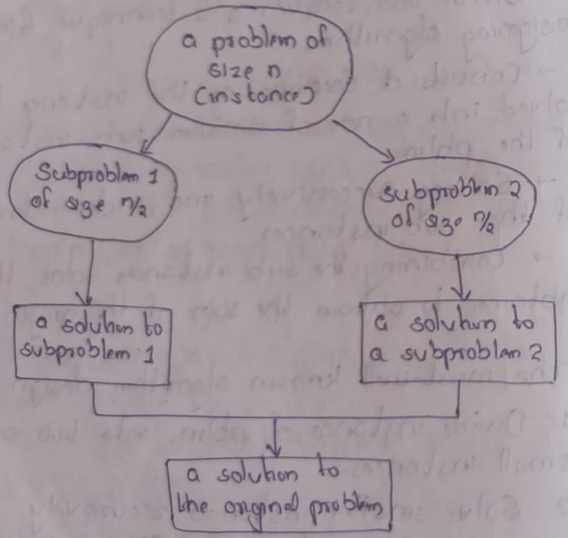
Three Steps

Divide: Whole pblm is divided into no of smaller subproblems

Conquer: Solve sub problem recursively

Combine: Solns to the sub problems combined to get the soln to the orginal pblm

Divide and Conquer Technique



Divide and Conquer Examples:

- Sorting: Merge sort & Quick sort
- Binary tree traversal
- Binary Search
- Multiplication of large integers

Control Abstraction

- A control abstraction is a procedure whose flow of control is clear but whose 1° operations are specified by other procedures whose precise meanings are left undefined
- The control abstraction for divide and conquer technique is DANDC(P), where P is the problem to be solved

```
Algorithm DAndC(p)
{
    if small(P)
        then return s(P);
    else
    {
        divide p into smaller instances p1, p2,
                        p3 --- pk  k ≥ 1;
        apply D and C to each of these sub pblms;
        return combine (DandC(p1), DandC(p2),
                    --- DandC(pk));
    }
}
```

- SMALL(P) is a Boolean valued fn which determines whether the input size is small enough so that the answer can be computed without splitting

- If this is so function 'S' is invoked otherwise, the pblm 'p' into smaller sub pblms
- These sub pblms p1, p2, ..., pk are solved by recursive application of D AND C
- If the sizes of the 2 sub pblms are approximately equal from the computing time of DANDC is:

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ 2T(n/2) + f(n) & \text{otherwise} \end{cases}$$

- where, T(n) is the time for D AND C on 'n' inputs
- g(n) is the time to complete the answer directly for small inputs and f(n) is the time for Divide and Combine

21/10/2021

Strassen's Matrix Multiplication

Basic Matrix Multiplication

Suppose we want to multiply 2 matrices of size n x n.

Eg: A x B = C

[A] [B] = [C]

2 x 2 matrix can be accomplished in 8 multiplication ($2^{\log 8} = 2^3$)

Matrix Multiplication

The problem:

Multiply 2 matrices A & B each of size [n x n]

$[A]_{n \times n} \cdot [B]_{n \times n} = [C]_{n \times n}$

The traditional way:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

Use three for-loop

$\therefore T(n) = O(n^3)$

Divide and Conquer way:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$C_{11} = A_{11} B_{11} + A_{12} B_{12}$
$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$
$C_{21} = A_{21} \cdot B_{11} + A_{22} B_{21}$
$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$

Transform the pblm of multiplying A and B, each of size [n x n] into 8 subproblems, each of size [n/2 x n/2].

$\therefore T(n) = 8 \times T(n/2) + cn^2$
$= O(n^2)$

which cn^2 is for addition

So, it is no improvement compared with the traditional way

Strassen's Matrix Multiplication

Strassen showed that 2x2 matrix multiplication can be accomplished in 7 multiplication & 18 additions or subtractions ($2^{\log_2 7} = 2^{2.807}$)

This reduce can be done by divide and conquer approach

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
$P_2 = (A_{21} + A_{22}) * B_{11}$
$P_3 = A_{11} * (B_{12} - B_{22})$
$P_4 = A_{22} * (B_{21} - B_{11})$
$P_5 = (A_{11} + A_{12}) * B_{22}$
$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$
$C_{12} = P_3 + P_5$
$C_{21} = P_2 + P_4$
$C_{22} = P_1 + P_3 - P_2 + P_6$

Example

$A = 1\ 2\ 3\ 4$
$B = 2\ 2\ 3\ 3$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 2 \\ 3 & 3 \end{bmatrix}$$

$$AB = \begin{bmatrix} 8 & 8 \\ 18 & 18 \end{bmatrix} \leftarrow \text{Traditional}$$

$P_1 = 5 \times 5 = 25$
$P_2 = 14$
$P_3 = 1(-1) = -1$
$P_4 = 4(1) = 4$
$P_5 = 3 \times 3 = 9$
$P_6 = (3-1) \times (2+2) = 2 \times 4 = 8$
$P_7 = -2 \times 6 = -12$

$C_{11} = 25 + 4 - 9 - 12 = 8$
$C_{12} = -1 + 9 = 8$
$C_{21} = 14 + 4 = 18$
$C_{22} = 25 + (-1) - 14 + 8 = 18$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} 8 & 8 \\ 18 & 18 \end{bmatrix} = AB$$

Hence Proved

$$T(n) = \begin{cases} T(n/2) + T(n/2) + 2 & n \geq 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

$T(n) = 2T(n/2) + 2$

$2(2T(n/4) + 2) + 2$

$4T(n/4) + 4 + 2$

$\vdots$

Given $n = 2^k$

$n/2 = 2^{k-1}$

We know $T(n) = 1$ when $n = 2$

$T(2) = 1$

$\Rightarrow 2^{k-1} T\left(\frac{2^k}{2^{k-1}}\right) + \Sigma 2^i$

$1 \leq i \leq k-1$

$\Rightarrow 2^{k-1} * T(2) + \Sigma 2^i$

$2^{k-1} \times 1 + \Sigma 2^i$

$\Rightarrow 2^{k-1} + 2^k - 2$

$2^{k-1} = n/2$ & $2^k = n$

$\Rightarrow n/2 + n - 2$

$\frac{3n}{2} - 2$

$S_n = \frac{a(r^n - 1)}{r-1}$

$= \frac{2(2^{k-1} - 1)}{2^1 - 1}$

$= 2^k - 2$

26/10/20 Methods For Solving Recurrences

There are 3 methods for solving recurrences

i) Substitution Method
ii) Recursion Tree "
iii) Master "

Substitution Method:

It comprises 2 steps

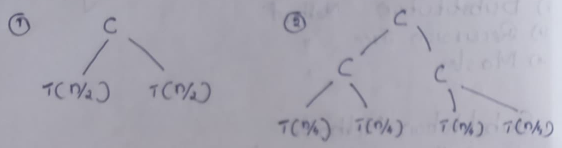
i) Guess the form of the soln
ii) Use mathematical induction to find the consts and show that the soln works

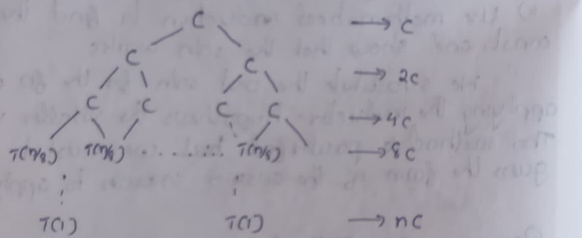
We substitute the best soln for the fn when applying the inductive hypothesis to smaller values. This method is powerful but we must be able to guess the form of the answer in order to apply it

Recursion Tree Method

It converts the recurrences into a tree whose nodes represent the cost incurred at various levels of recursion. In a recursion tree each node represents the cost of a single sub-problem somewhere in the set of recursive fn invokations. These sum the cost within each level of the tree to obtain a set of per-level cost & we sum all the per level cost to determine the total cost of all levels of recursion

Q $T(n) = \begin{cases} 2T(n/2) + c & n \geq 1 \\ c & n = 1 \end{cases}$





Soln

$c + 2c + 4c + 8c + \ldots + nc$

$c(1 + 2 + 4 + \ldots + n)$

Since $n = 2^k$

$c\left(\frac{1(2^{k+1}-1)}{2-1}\right)$

$c(2^{k+1} - 1)$

$c(2n - 1)$

$= O(n)$

$S_n = \frac{a(r^n - 1)}{r - 1}$

$\frac{1(2^{2^k} - 1)}{2-1}$

$2^{k+1} = 2 \times 2^k = 2n$

Q $a = 2, \; b = 2 \quad T(1) = 2 \quad f(n) = n$

$T(n) = 2T(n/2) + n$

$2[2T(n/4) + n/2] + n$

$4T(n/4) + n + n$

$4[T(n/4) + 2n]$

$8T(n/8) + 3n$

$= 2^3 T(n/2^3) + 3n$

$T(n) = 2T(n/2) + n$

$T(n/2) = 2T(n/4) + n/2$

$T(n/4)$

$=$

Given $T(1) = 2$

$2^i T(n/2^i) + in \quad \because \log_2 n \geq i \geq 1$

$2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n \log n$

$n(T(1)) + n \log_2 n$

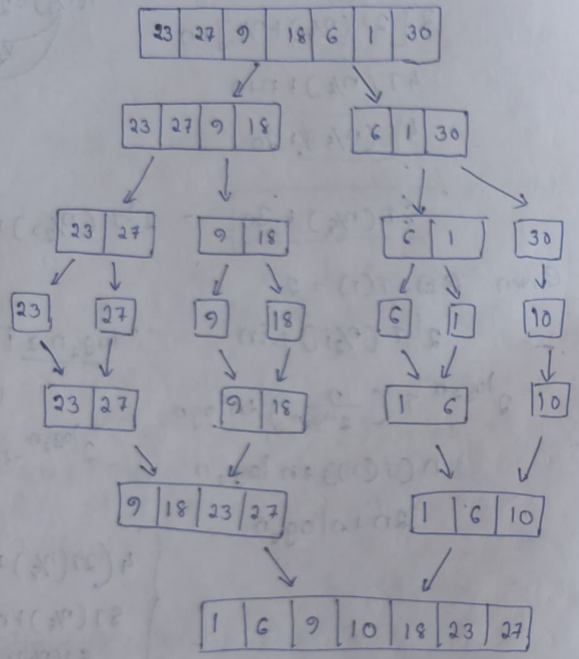
$2n + n \log_2 n$

$2^{\log_2 n} = n$

$\begin{cases} 4(2T(n/8) + n/4) + 2n \\ 8T(n/8) + n + 2n \\ 8T(n/8) + 3n \end{cases}$

27/10/2021

Merge Sort



Substitution Method

$$T(n) = 2T(n/2) + cn$$

$$\Rightarrow 2\left[2T(n/4) + \frac{cn}{2}\right] + cn$$

$$4T(n/4) + cn + cn$$

$$4T(n/4) + 2cn$$

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

$$4\left[2T(n/8) + c(n/4)\right] + 2cn$$

$$\Rightarrow 8T(n/8) + cn + 2cn$$

$$\Rightarrow 8T(n/8) + 3cn$$

$$\Rightarrow 2^3 T\left(\frac{n}{2^3}\right) + 3cn$$

$$\Rightarrow 2^k T\left(\frac{2^k}{2^k}\right) + kcn \qquad n = 2^k$$

$$\Rightarrow 2^k T(1) + kcn \qquad \log n = k$$

$$\Rightarrow an + cn\log n$$

$$\Rightarrow O(n\log n)$$

28/10/2021

Merge Sort

Is an eg. of Divide & Conquer technique. It sorts the given array $A[1 \ldots N]$ by dividing it into 2 halves $A[1 - N/2]$ and $A[N/2+1 - N]$ sorting each of them recursively and then merging the smaller sorted arrays into a single sorted one. We assume throughout that the elements are sorting in ↑ order

One pblm in merge sort is its use of $2n$ locations. The additional n locations were needed bcs we couldn't reasonably merge ~~knows~~ 2 sorted sets in place.

Another pblm about merge sort is its stack space necessitation by the use of recursion. ∵ merge sort splits each set into

2 approximately equal sized subsets, the max. depth of the stack is ~~provide~~ proposed to $\log n$

Quick Sort

Is based on the divide & conquer design technique. In this at every step each element is placed in its proper position.
It works recursively by first selecting a random pivot value from the array. Then partition the list into elements that are less than the pivot & greater than the pivot.
The pblm of sorting a given list is reduced to the pblm of sorting 2 sublists. It is to be noted that the reduction step in the quick sort, finds the final position of the partitioned element

Comparing with merge sort, the division into 2 sub-arrays is made so that the sorted sub-arrays do not need to be merge later

Analysis

In worst case, the chosen pivot is either the smallest or the largest element in an array. In this case one part is empty and the other part contains the remaining elements. The generated $n-1$ levels in the tree is $T(n) = O(n^2)$

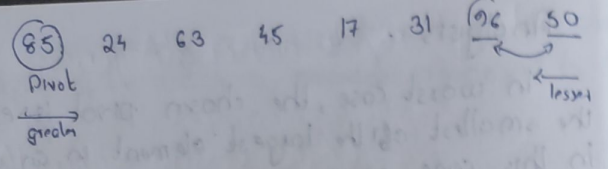
In best case analysis O and C works best if each array is divided into 2 equal sub-arrays of size $n/2$. This generates $T(n) = 2T(n/2) + n = O(n \log n)$.

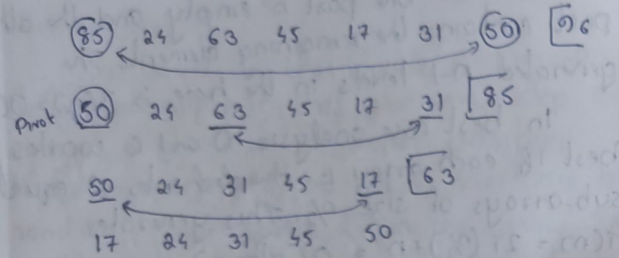
In avg. case analysis the array is partitioned by choosing any random number. In this case, some of the partitions are well-balanced while some are ~~totally~~ fairly unbalanced, ie $T(n) = O(n \log n)$.

Example Quick Sort:

85 (24) 63 45 17 31 (96) 50
Pivot

85 96 (63) 45 17 31 24 50





Sorted Order:

17 24 31 45 50 63 85 96

02/11/2021
Tuesday

MODULE : 2

GREEDY STRATEGY

The Greedy Method suggest that one can divide an alg. that works in stages considering one input at a time. At each stage a decision is made regarding whether a parbaulas input is an optimal soln. This is done by considering the inputs in an order determined by some selection procedure. If the inclusion of the next input into the parbally constructed optimal soln will result in an infeasible soln then this input is not added to the partial soln otherwise it is added. The selection procedure itself is based on some optimization procedure. This measure may be the objective fn.

The pblms have 'n' inputs and require us to obtain a subset that satisfies some constraints. Any subset that satisfies these constraints are called feasible soln. We need to find a feasible soln that either maximizes or minimizes a given objective fn. A feasible soln that does this is called an optimal soln.

Control Abstraction of Greedy Method

```
Algorithm Greedy (A, n)
{
    soln = ϕ
    for i ← 1 to n do
    {
        x = select (a)
        if Feasible (soln, x) then
            soln = union (soln, x)
    }
    return (soln)
}
```

The function 'select', selects an input from A and removes it. The selected i/p value is assigned to x. 'Feasible' is a boolean valued function that determines whether x can be included into the soln vector.

The fn 'union' combines x with the soln and updates the objective fn.

08/12/2021 Monday

Knapsack Problem.

We are given n objects and a Knapsack or a bag. Object i has a weight w_i and Knapsack has the capacity m. If a fraction x_i, $0 \le x_i \le 1$ of object i is placed into the Knapsack, then a profit of $p_i x_i$ is earned.

The objective is to obtain a filling of the Knapsack that maximizes the total profit earned.

The pblm can be formulated as maximize $\sum_{i=1}^{n} p_i x_i$ subject to $1 \le i \le n$

$0 \le x_i \le 1$ &

$w_i x_i \le m$

1. Consider the following Knapsack pblm $n = 3$, $m = 20$, $P_1, P_2, P_3 = \{25, 24, 15\}$, $w_1, w_2, w_3 = \{18, 15, 10\}$. Find out the max. profit soln

(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
$0, 1, 1/2$	$18\times0 + 1\times15 + 1/2\times10 = 20$	$25\times0 + 24\times1 + 15\times\frac{1}{2} = 31.5$
$1/2, 1/3, 1/4$	$18\times\frac{1}{2} + 15\times\frac{1}{3} + \frac{1}{4}\times10 = 16.5$	$25\times\frac{1}{2} + 24\times\frac{1}{3} + 15\times\frac{1}{4} = 24.25$
$1, 2/15, 0$	$1\times18 + \frac{2}{15}\times15 + 0\times10 = 20$	$25\times1 + 24\times\frac{2}{15} + 0\times15 = 28.2$
$0, 2/3, 1$	$0\times18 + 2/3\times15 + 1\times10 = 20$	$0\times25 + 25\times\frac{2}{3} + 1\times15 = 31$

Maximum profit is 31.5 at $(0, 1, 1/2)$ → Feasible solution

$\frac{P_i}{w_i}$

$\frac{P_1}{w_1} = \frac{25}{18} = 1.3$

$\frac{P_2}{w_2} = \frac{24}{15} = 1.6$

$\frac{P_3}{w_3} = \frac{15}{10} = 1.5$

$P_2 > P_3 > P_1$

Q Consider the following instance with Knapsack capacity $w = 100$

Item	I_1	I_2	I_3
Weight	18	15	10
Value	25	24	15

$\frac{P_1}{w_1} = \frac{25}{18} = 1.38$

$\frac{P_2}{w_2} = \frac{24}{15} = 1.6$

$\frac{P_3}{w_3} = \frac{15}{10} = 1.5$

$P_2 > P_3 > P_1$

x_1, x_2, x_3	$\sum w_i x_i$	$\sum P_i x_i$
$\frac{2}{3}, 1, 1$	$\frac{2}{3} \times 18 + 1 \times 15 + 1 \times 10 = 37$	$\frac{2}{3} \times 25 + 1 \times 24 + 1 \times 15 = 55.5$
$1, \frac{2}{3}, 0$	$18 + \frac{2}{3} \times 15 = 28$	$25 + \frac{2}{3} \times 24 = 41$
$1, 0, \frac{1}{2}$	$18 + 10 \times \frac{1}{2} = 23$	$25 + 15 \times \frac{1}{2} = 32.5$

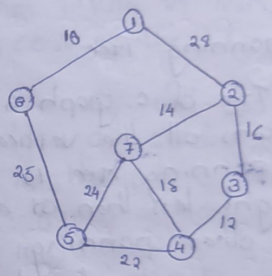
Maximum profit is 36.5 and Feasible soln is $(\frac{3}{8}, 1, 1)$

09/11/2021

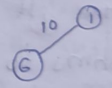
Knapsack Algorithm

(Should be write from the copy and learn)

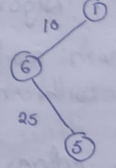
Minimum Cost Spanning Tree



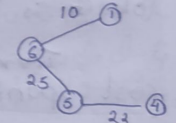
Stage 1



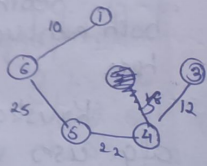
Stage 2



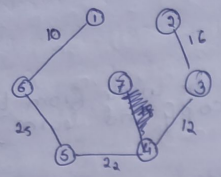
Stage 3



Stage 4

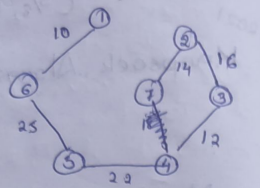


Stage 5



Minimum Cost = 99

Stage 6



Minimum-Cost Spanning Tree

A Spanning Tree of a graph is just a subgraph that contains all the vertices & is a tree. A min. cost spanning tree is a spanning tree with weight less than or equal to the weight of every other spanning tree.

Let $G = (V, E)$ is an undirected connected graph. A subgraph $T = (V, E')$ of G is a spanning tree iff it is a tree

There are 2 algorithms to find a min. cost spanning tree:

i) The Prim's Algorithm
ii) The Kruskal's Algorithm

Both differ in their methodology but both eventually end up with min spanning tree.

Kruskals alg. uses edges & Prim's alg. uses vertex connection to determine the MST

Both are greedy algorithms that run in polynomial time.

Prims Algorithm

Is a greedy method to obtain a min. cost spanning Tree, It builds the tree edge by edge. The nxt edge to include is chosen according to some optimizing criteria. The simplest such criterion is to choose an edge that results in min. increase in the sum of the cost of the edges so far included.

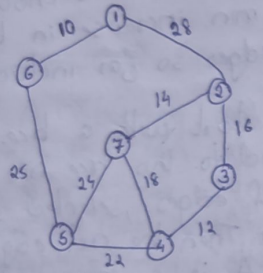
The alg. will start with a tree. Then the edges are added to this tree one by one. If A is the set of edges selected so far then A forms a tree. Then the nxt edge (u,v) to be included in A is a min. cost edge not in A, with the ppty that $A \cup (u,v)$ is also a tree. The next edge (i,j) to be added s.t i is the vertex already included in the tree & j is the vertex not yet included & cost of (i,j) is min among all the edges (k,l) s.t vertex k is in the tree & vertex l is not in the tree

The main idea of Prim's alg. is to find the shortest path in the graph

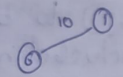
We begin with some vertex q in the graph, $G = (V, E)$ defining the initial set of vertices then in each iteration we choose a

min weight edge (u,v) connecting ~~about~~ the vertex v in the set A to the vertex x ∈ B outside A. Then vertex u is brought into A. This process is repeated until a spanning tree is formed

10/11/2021 Wednesday Kruskals Algorithm

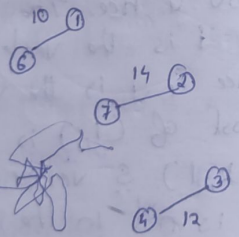


Stage I

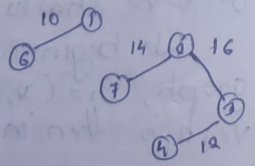


Stage II

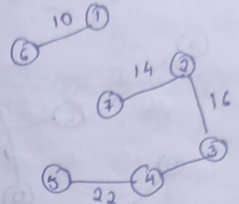
Stage III



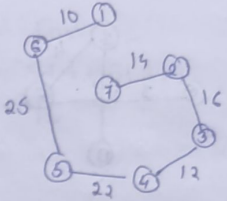
Stage IV



Stage V

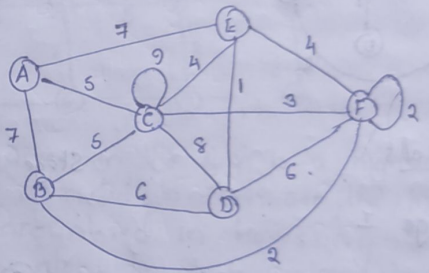


Stage VI



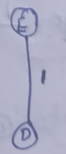
Min. Cost = 10+25+22+12+16+14 = 99

Question

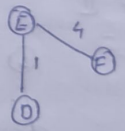


Prims:

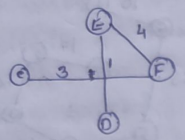
Stage I



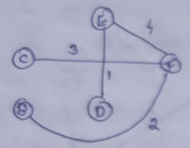
Stage II



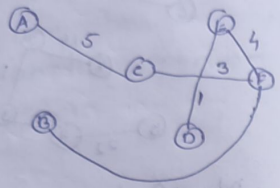
Stage III



Stage IV



Stage V

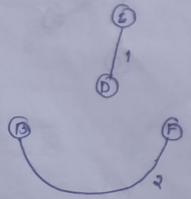


Min. Cost = 5+4+3+2+1
= 15

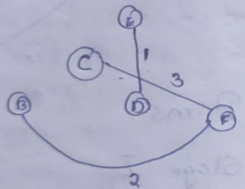
Kruskals

Stage I

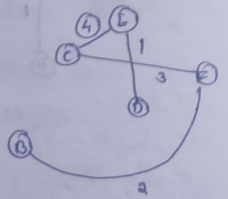
Stage II



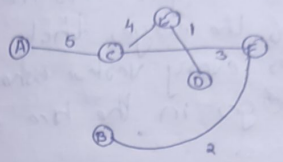
Stage III



Stage IV



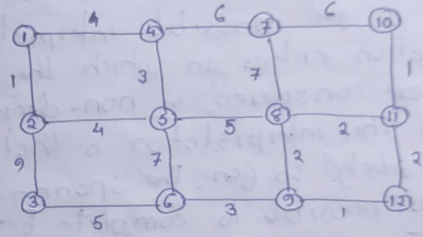
Stage VI



Min. Cost
= 5+4+3+2+1
= 15

H.W

Q:



Network given in the fig. is a highway map & the no. recorded to each arc, the maximum elivation encountered in traversing the arc. A traveller plans to drive from node 1 to 12 on this highway. The traveller dislikes high altitudes & would like to find the path connecting node 1 to 12 that minimizes the maximum altitude. Find the best path for the traveller using a MST.

Kruskal's Algorithm

It is an alg. which finds the min. spanning tree for a connected weighted graph. It finds a safe edge to add the to the growing forest by finding of all the edges

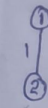
that connect any 2 tree of in the forest and an edge (v,w) of least weight. This means that it finds a subset of the edges that forms a tree, that includes every vertex where total weight of all the edges in the tree is minimized

There is a 2nd possible interpretation of the optimization criteria in which the edges of the graph are considered in non-decreasing order of cost. This interpretation is that the set of edges selected so far, the spanning tree be s. it is possible to complete T into a tree. Thus T may not be a tree at all stages in the algorithm. It will generally only be a forest ∵ the set of edges T can be completed into a tree iff there are no cycles in it

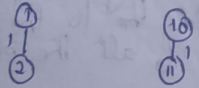
H.W (Answer)

By Kruskals Algorithm

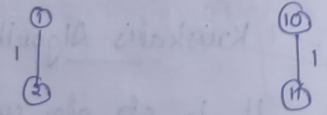
Step I



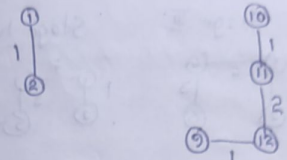
Step II



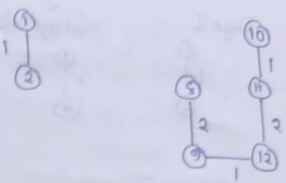
Step III



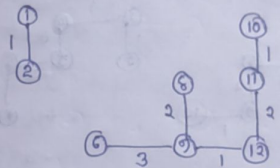
Step IV



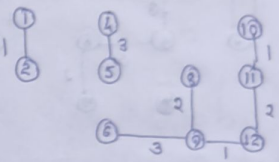
Step V



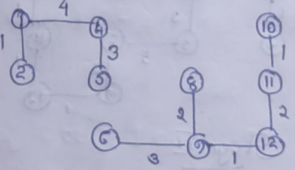
Step VI



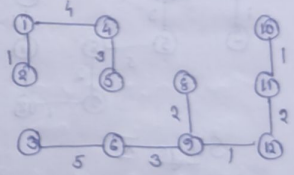
Step VII



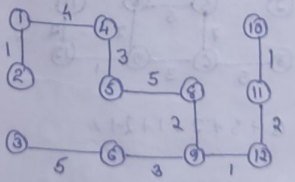
Step VIII



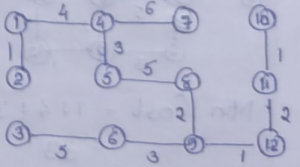
Step IX



Step X



Step XI



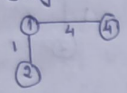
Min Cost = 1+4+6+5+2+3+5+1+2+1+3

= 33

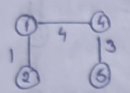
Prim's Algorithm

Stage I

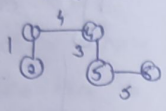
Stage II



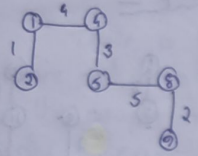
Stage III



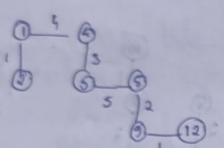
Stage IV



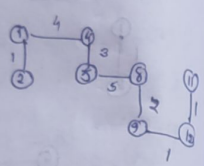
Stage V



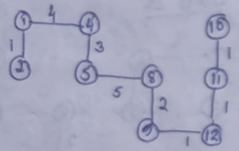
Stage VI



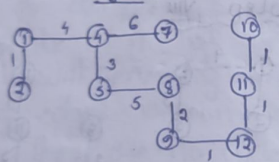
Stage VII



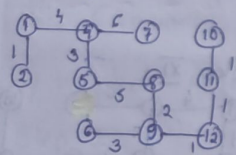
Stage VIII



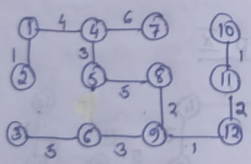
Stage IX



Stage X



Stage XI



Min. Cost = 1+4+3+3+5+6+5+2+1+2+1

= 33

The maximum value is 127

Feasible soln is (1,4) Job 1 and Job 4 taken in the order (4,1).

Job Sequencing with deadlines

Here we are given n objects jobs associated with each job i, there is an integer deadline d_i and a profit p_i ; $d_i \geq 0$

$p_i \geq 0$

For any job i, the profit p_i is earned iff the job is completed by its deadline. To complete a job one has to process a job in a machine for one unit of time. Only one machine is available for processing the jobs. The value of the feasible soln is the sum of the profits in J. An optimal soln, is a feasible soln with max. value.

Example:

Let n = 4

$(p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Find feasible soln & maximum value.

Feasible Soln	Processing Order	Value / Profit
(1, 2)	(2, 1)	110
(1, 3)	(1, 3)	115
(1, 4)	(4, 1)	127
(2, 3)	(2, 3)	25
(3, 4)	(4, 3)	42
1	1	100
2	2	10
3	3	15
4	4	27

Q n=5

$(d_1, d_2, d_3, d_4, d_5) = (2, 1, 3, 2, 1)$ 3

$(P_1, P_2, P_3, P_4, P_5) = (60, 100, 20, 40, 20)$ 60

1 2 3 4 5

1 3, 4 (1) 3

2 3 2

Soln

Feasible Solution	Processing Order	Value / Profit
1	1	60
2	2	100
3	3	20
4	4	40
5	5	20
(1,2,3)	(2,1,3)	100+60+20=180
(3,4,5)	(5,4,3)	20+40+20=80
(1,3,5)	(5,1,3)	20+60+20=100

The maximum value is 180

Feasible soln is for jobs job1, job2, job3 taken in the order (2,1,3)

Dynamic Programming (DP)

The most powerful design tehnique for optmization pblm was invented by Richard Bellman, a prominent mathematician. The solns for DP are based on multistage optmizing decision on a few common elements. The DP is closely related to D and C techniques where the pgm breaks down into smaller subproblems and each subproblem is solved recursively.

The DP differs from D and C on a way that instead of solving a subproblem recursively, it solves each of the subproblem only once and store the solns to the subproblems in a table.

D and C is a top-down method, when a problem is solved by D and C, we immediately attack the complete instance which we then divide into smaller and smaller sub instances as the algorithm progresses. DP on the other hand is a bottom-up method, we usually start with the smallest and finds the simplest sub-instances by combining their solutions., we obtain the answers of the sub instances of increasing size until finally we arrive at the soln of the original step

DP is an algorithm design method that can be used when the soln to a problem can be viewed as the result of a seq. of decision. In DP an optimal sequence of decisions is obtained by making explicit appeal to principles of optimality. The diff b/w greedy & DP is that in greedy only 1 decision seq is ever generated. in dp, many decision seq is generated.

The steps for achieving DP are divide

i) Divide, subproblem

The main pblm is divided into several smaller pblm. In this the soln of the main pblm is expressed in terms of the soln of the smaller sub problem

ii) Table, storage

Soln for subproblem is stored in a table so that it can be used many times whenever required

iii) Combine, bottom

Combine, bottom the soln move up computation, the soln to main problem is obtained by combining the solutions of smaller sub problems

For pblm to be solved by DP, it must follow the following condition:

i) Principle of Optimality

States that for solving the pblm optimally its subproblem should be solved optimally. It should be noted that not all times, each subproblem is solved optimally so in that case we should go for the optimal majority

ii) Polynomial Break Up

For solving the main pblm, the pblm be divided into several smaller subproblems and for efficient performance DP, the total no. of subproblems to be solved should be almost a polynomial no

23/11/2021 Tuesday

TRAVELLING SALESPERSON PROBLEM (TSP)

Question 1:

$$C_{ij} = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

Cost Adjacency Matrix = C_{ij}

1st level

$g(2, \phi) = C_{21} = 5$

$g(3, \phi) = C_{31} = 6$

$g(4, \phi) = C_{41} = 8$

$$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\}) \}$$

2nd level

$g(2,\{3\}) = g(2,\{3,\phi\}) = c_{23} + g(3,\phi)$
$= 9 + 6 = 15$

$g(2,\{4\}) = c_{24} + g(4,\phi)$
$= 10 + 8 = 18$

$g(3,\{2\}) = c_{32} + g(2,\phi)$
$= 13 + 5 = 18$

$g(3,\{4\}) = c_{34} + g(4,\phi)$
$= 12 + 8 = 20$

$g(4,\{2\}) = c_{42} + g(2,\phi)$
$= 8 + 5 = 13$

$g(4,\{3\}) = c_{43} + g(3,\phi)$
$= 9 + 6 = 15$

3rd level

$g(2,\{3,4\}) = \min(c_{23} + g(3,4), c_{24} + g(4,3))$
$= \min(9 + 20, 10 + 15)$
$= \min(29, 25) = 25$

$g(3,\{2,4\}) = \min(c_{32} + g(2,4), c_{34} + g(4,2))$
$= \min(13 + 18, 12 + 13)$
$= \min(31, 25) = 25$

$g(4,\{2,3\}) = \min(c_{42} + g(2,3), c_{43} + g(3,2))$
$= \min(8 + 15, 9 + 18)$
$= \min(23, 27) = 23$

4th level

$g(1,\{2,3,4\}) = \min(c_{12} + g(2,\{3,4\}),$
$c_{13} + g(3,\{2,4\}),$
$c_{14} + g(4,\{2,3\}))$
$= \min(10 + 25, 15 + 25, 20 + 23)$
$= \min(35, 40, 43) = 35$

Min = 35 = ie $g(1,\{2,3,4\}) = c_{12} + g(2,\{3,4\})$; $j = 2$
In $g(2,\{3,4\})$, min = 25 = $c_{24} + g(4,3)$; $j = 4$
In $g(4,3)$, min = 6 = $g(3,\phi)$; $j = 3$
In $g(3,\phi)$, min = 6 ; $j = 1$

So path = 1 2 4 3 1
The min. cost is 35
The tour is $1 \to 2 \to 4 \to 3 \to 1$

Let $G = (V, E)$ be a directed graph with edge cost c_{ij}. The variable c_{ij} is defined s.t. $c_{ij} > 0$ $\forall$ i, j and
$c_{ij} = \infty$ if $i, j \notin E$

Let $|V| = n$ and assume $n > 1$. A tour of G is a directed simple cycle that includes every vertex in V. The cost of the tour is the sum of the cost of the edges on the tour. The travelling sales person pblm is to find the tour of min. cost

Without loss of generality regard a tour to be a simple path that starts and end

at vertex 1. Every tour consist of an edge $(1,k)$ for some $k \in V-\{1\}$. The path from vertex k to vertex 1 goes through each vertex in $V-\{1,k\}$. Hence the principle of optimality holds.

Let $g(i,s)$ be the length of the shortest path starting at vertex i going through all vertices in S and terminating at vertex 1. The function $g(1, V-\{1\})$ is the length of an optimal salesperson's tour. From the principle of optimality it follows that:

$$g(1,V-\{1\}) = \min_{2 \le k \le n} \{C_{1k} + g(k, V-\{1,k\})\} \rightarrow (1)$$

Generalising (1), we obtain:

$$g(i,s) = \min_{j \in s} \{C_{ij} + g(j, s-\{j\})\} \rightarrow (2)$$

g values can be obtained by using (2).

Clearly,

$$g(i,\phi) = C_{i1} \quad ; 1 \le i \le n$$

Then we can obtain $g(i,s)$ with a cardinality, s ie $|s| = 2$, and so on $|s| < n-1$

The cardinality $g(i,s)$ is needed s.t $i \neq 1$ and $1 \notin s$ and $i \notin s$

Question 2:

$$C_{ij} = \begin{bmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{bmatrix}$$

$g(2,\phi) = C_{21} = 1$
$g(3,\phi) = C_{31} = 15$
$g(4,\phi) = C_{41} = 6$

$g(2,\{3\}) = C_{23} + g(3,\phi) = 6+15 = 21$
$g(2,\{4\}) = C_{24} + g(4,\phi) = 4+6 = 10$
$g(3,\{2\}) = C_{32} + g(2,\phi) = 7+1 = 8$
$g(3,\{4\}) = C_{34} + g(4,\phi) = 8+6 = 14$
$g(4,\{2\}) = C_{42} + g(2,\phi) = 3+1 = 4$
$g(4,\{3\}) = C_{43} + g(3,\phi) = 12+15 = 27$

$g(2,\{3,4\}) = \min(C_{23}+g(3,4), C_{24}+g(4,3))$
$= \min(6+14, 4+27) = \min(20,31) = 26$

$g(3,\{2,4\}) = \min(C_{32}+g(2,4), C_{34}+g(4,2))$
$= \min(7+10, 8+4) = \min(17,12) = 12$

$g(4,\{2,3\}) = \min(C_{42}+g(2,3), C_{43}+g(3,2))$
$= \min(3+21, 12+8) = \min(24,20) = 26$

$g(1,\{2,3,4\}) = \min(C_{12}+g(2,\{3,4\}),$
$C_{13}+g(3,\{2,4\}),$
$C_{14}+g(4,\{2,3\}))$
$= \min(2+20, 9+12, 10+20)$
$= \min(22, 21, 30) = 21$

Min cost = 21
Tour is $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$

24/11/2021
Wednesday

ALL PAIRS SHORTEST PATH PROBLEM

Let $G = (V, E)$ be a directed graph with 'n' vertices. Let cost be a cost adjacencies matrix for G s.t $cost(i,j) = 0, 1 \le i \le n$

Then cost of i,j is the length of the edge (i,j) if $i,j \in E(G)$ and cost of $(i,j) = \infty$, if $i \neq j$ & $i,j \notin E(G)$.

The all pairs shortest path problem is to determine a matrix A s.t A_{ij} is the length of the ~~select~~ shortest path from $i \to j$. A shortest $i \to j$ path will be $i \neq j$ originate at vertex i and goes through some intermediate values and terminate at some value j.

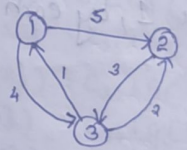
We can assume that this path contains no cycles. For if there is a cycle, then this can be deleted without increasing the path length. If k is the intermediate vertex on this shortest path, then the sub path from i to k and k to j must be shortest path from i to k and k to j respectively

The construction of the shortest i to j path as first requiring a decision as to which is the highest indexed intermediate vertex k. Once the decision has been made, we need to find two shortest path, one from i to k and other from k to j.

Using $A^k(i,j)$ to represent the length of the shortest path for i to j going through no vertex of index greater than k.

$$A^k(i,j) = \min(A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j))$$

Example



$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 5 & 4 \\ \infty & 0 & 3 \\ 1 & 2 & 0 \end{bmatrix} \end{array}$$

Considering vertex 1

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 5 & 4 \\ \infty & 0 & 3 \\ 1 & 2 & 0 \end{bmatrix} \end{array}$$

$$A^2 = \begin{bmatrix} 0 & 5 & 4 \\ \infty & 0 & 3 \\ 1 & 2 & 0 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 0 & 5 & 4 \\ 4 & 0 & 3 \\ 1 & 2 & 0 \end{bmatrix}$$ → Final Matrix

Q 

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

Passing through vertex 1, by each pairs

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

Final Matrix showing shortest path

25/11/2021
Thursday

1 TSP $= \begin{bmatrix} 1 & 2 & 0 & 6 & 1 \\ 1 & 6 & 4 & 4 & 2 \\ 5 & 3 & 0 & 1 & 5 \\ 4 & 7 & 2 & 0 & 1 \\ 2 & 6 & 3 & 6 & 6 \end{bmatrix}$

Eqn: $g(i,s) = \min_{j \in s} \{ c_{ij} + g(j, s-\{j\}) \}$

$g(2,\phi) = c_{21} = 1$
$g(3,\phi) = c_{31} = 5$
$g(4,\phi) = c_{41} = 4$
$g(5,\phi) = c_{51} = 2$

$g(2,\{3\}) = c_{23} + g(3,\phi) = 4+5 = 9$
$g(2,\{4\}) = c_{24} + g(4,\phi) = 4+4 = 8$
$g(2,\{5\}) = c_{25} + g(5,\phi) = 2+2 = 4$
$g(3,\{2\}) = c_{32} + g(2,\phi) = 3+1 = 4$
$g(3,\{4\}) = c_{34} + g(4,\phi) = 1+4 = 5$
$g(3,\{5\}) = c_{35} + g(5,\phi) = 5+2 = 7$
$g(4,\{2\}) = c_{42} + g(2,\phi) = 7+1 = 8$
$g(4,\{3\}) = c_{43} + g(3,\phi) = 2+5 = 7$
$g(4,\{5\}) = c_{45} + g(5,\phi) = 1+2 = 3$
$g(5,\{2\}) = c_{52} + g(2,\phi) = 6+1 = 7$
$g(5,\{3\}) = c_{53} + g(3,\phi) = 3+5 = 8$
$g(5,\{4\}) = c_{54} + g(4,\phi) = 6+4 = 8$

$g(2,\{3,4\}) = \min(c_{23} + g(3,4), c_{24} + g(4,3))$
$= \min(4+5, 4+7) = \min(9, 11) = 9$

$g(2,\{4,5\}) = \min(c_{24} + g(4,5); c_{25} + g(5,4))$
$= \min(4+3, 2+8) = \min(7, 10) = 7$

$g(2,\{3,5\}) = \min(c_{23}+g(3,5), c_{25}+g(5,3))$
$= \min(4+7, 2+8) = \min(11,10) = 10$

$g(3,\{2,4\}) = \min(c_{32}+g(2,4), c_{34}+g(4,2))$
$= \min(3+8, 1+8) = \min(11,9) = 9$

$g(3,\{2,5\}) = \min(c_{32}+g(3,2), c_{35}+g(5,2))$
$= \min(3+4, 5+7) = \min(7,12) = 7$

$g(3,\{4,5\}) = \min(c_{34}+g(4,5), c_{35}+g(5,4))$
$= \min(1+3, 5+8) = \min(4,13) = 4$

$g(4,\{2,3\}) = \min(c_{42}+g(2,3), c_{43}+g(3,2))$
$= \min(7+9, 2+4) = \min(16,6) = 6$

$g(4,\{3,5\}) = \min(c_{43}+g(4,3), c_{45}+g(5,3))$
$= \min(2+7, 1+8) = \min(9,9) = 9$

$g(4,\{2,5\}) = \min(c_{42}+g(2,5), c_{45}+g(5,2))$
$= \min(7+4, 1+7) = \min(11,8) = 8$

$g(5,\{2,3\}) = \min(c_{52}+g(2,3), c_{53}+g(3,2))$
$= \min(6+9, 3+4) = \min(15,7) = 7$

$g(5,\{2,4\}) = \min(c_{52}+g(2,4), c_{54}+g(4,2))$
$= \min(6+8, 6+8) = \min(14,14) = 14$

$g(5,\{3,4\}) = \min(c_{53}+g(3,4), c_{54}+g(4,3))$
$= \min(3+5, 6+7) = \min(8,13) = 8$

$g(2,\{3,4,5\}) = \min(c_{23}+g(3,\{4,5\}), c_{24}+g(4,\{3,5\}),$
$c_{25}+g(5,\{3,4\}))$
$= \min(4+7, 4+9, 2+8)$
$= \min(11,13,10) = 10$

$g(3,\{2,4,5\}) = \min(c_{32}+g(2,\{4,5\}), c_{34}+g(4,\{2,5\}),$
$c_{35}+g(5,\{2,4\}))$
$= \min(3+7, 1+8, 5+14)$
$= \min(10,9,19) = 9$

$g(4,\{2,3,5\}) = \min(c_{42}+g(2,\{3,5\}), c_{43}+g(3,\{2,5\}),$
$c_{45}+g(5,\{2,3\}))$
$= \min(7+10, 2+6, 1+7)$
$= \min(17,8,8) = 8$

$g(5,\{2,3,4\}) = \min(c_{52}+g(2,\{3,4\}), c_{53}+g(3,\{2,4\}),$
$c_{54}+g(4,\{2,3\}))$
$= \min(6+9+3+9, 6+6)$
$= \min(15,12,12) = 12$

$g(1,\{2,3,4,5\}) = \min(c_{12}+g(2,\{3,4,5\}),$
$c_{13}+g(3,\{2,4,5\}), c_{14}+g(4,\{2,3,5\}),$
$c_{15}+g(5,\{2,3,4\}))$
$= \min(2+10, 0+9, 6+8, 1+12)$
$= \min(12,9,14,13) = 9$

So minimum cost = 9

Tour is : $1 \to 3 \to 4 \to 5 \to 2 \to 1$

07/12/2021

MODULE : 3

BACKTRACKING

Backtracking alg. is applicable to the wide range of algorithm. The key point of it is a binary choice that means 'Yes' or 'No'. Whenever the backtracking has choice 'No' that means the alg. has encountered a deadend, and it backtracks one step and tries a diff path for choice Yes. The Backtracking resembles a DFS tree in a di-graph, where graph is either a tree or atleast doesnot have any cycles

The soln for the pblm according to the backtracking can be represented as implicit graph on which backtracking performs an intelligent DFS, so as to provide one or more all possible soln to the given pblm. The whole task is accomplished by maintaining partial solns, as the search proceeds. It can be seen that such partial soln bind the fn where a complete soln to the pblm can be obtained. Initially no soln to the pblm is known When search proceeds a new element is added to the partial soln which in turns ↓ the remaining possibilities for a complete soln

The search is successful, if a complete soln for the pblm is defined, in this case the search either terminate or continue to search for all other possible soln. If at any stage the search is unsuccessful that means the partial solns, constructed so far are unable to define the complete soln, then the search backtracks one step. It should be noted that, the element from the partial soln is also removed from backtracking

In many applications of the backtrack method, the desired soln is expressible as an n-tuple $(x_1, x_2, x_3, \ldots x_n)$ where the x_i are chosen from finite set S_i often the pblm to be solved calls for finding 1 vector that maximizes a criterion fn $P(x_1, x_2 \ldots x_n)$

Suppose m_i is the size of the set S_i, then there are $m = m_1, m_2, \ldots m_n$ that are possible candidates for satisfying the fn P. Its basic idea is to build up the soln vector one component at a time and to use modifying criterion fn $P(x_1, x_2 \ldots x_i)$, whether the vector being formed has any chance of success.

Constrains can be divided into two categories:

i) Explicit Constraints

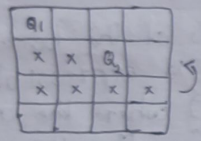
Are rules that restrict each x_i to take m values. only from a given set.

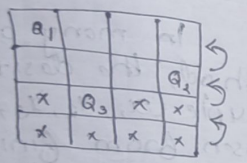
ii) Implicit Constraints

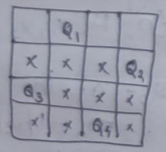
Are rules that determine which the tuples in the soln phase satisfy the criterion function

9/12/2021 Thursday.

N-Queens Problem.







The famous comparitorial N Queen's Pblm is to place N-Queens on an NxN chess board that no two queens attack each other by being in the same row, column, or diagonal.

It can be seen that, for N=1, the pblm has a trivial soln and no solution exists for N=2 and N=3.

4-Queens Problem

Given a 4x4 chess board, let us no. the rows & columns of chess board 1-4, ∴ we have to place 4 queens on a chess board ∴ no two queens attack each other. We inumber this as Q_1, Q_2, Q_3 & Q_4

Each queen must be placed on a diff row, so we place queen Q_i on row i

First we place queen Q_1 on the very first acceptable position, i.e (1,1). The first acceptable position for queen Q_2 is (2,3) But later this position proves to be dead end as no position is left for placing Q_3 safely. So we back-track one step & place queen Q_2 in (2,4), the next possible location. Each node describes its partial solution, one possible soln is shown above. For other soln, the whole method is repeated for the whole partial soln.

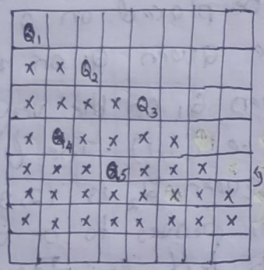
It can be seen that all the soln to the 4 queens pblm can be represented as 4-tuples (t_1, t_2, t_3, t_4) where t_i represents the column no, which Queen Q_i is placed.

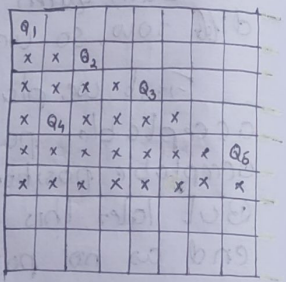
The explicit constrains for this are:

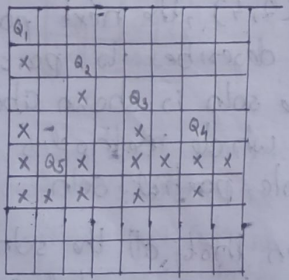
$S_i = \{1, 2, 3, 4\}$, where $1 \le i \le 4$

The implicit constraint is that no queen can be placed in the same row, same column or same diagonal.

8-Queens Problem







We can formulate soln to 8-Queens problem which need 8 tuples for the representation t_1 to t_8, where t_i represents the column on which queen Q_i is placed in. The soln phase consists of all 8! permutations.

Suppose two queens are placed at positions (i, j) & (k, l), then these are on the same diagonal if $i - j = k - l$ or $i + j = k + l$, ie, two queens lie on the same diagonal iff absolute value of $|i - l| = |j - k|$

A simple algorithm yielding a soln to the N-queen puzzle for $n = 1$ or any $N \ge 4$:

Step 1:

Divide ~~N/12~~ N by 12, remember the remainder

Step 2:

Write a list of even no. from 2 to ~~N~~ N in order

Step 3:

If the remainder is 3 or 9 move 2 to the end of the list

Step 4:

Write odd nos 1 - N in in order, ~~work~~ if the remainder is 8 switch pairs

Step 5:

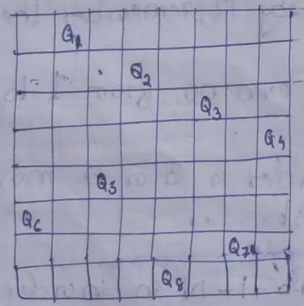
If remainder is 2, switch the place of 1 and 3 then move 5 to the end of the list.

Step 6:
If remainder is 3 or 9, move 1 and 3 to the end of the list.

Step 7:
Place the first column queen in the row, with the first row in the list and place the 2nd column queen with the 2nd row in the list

Example
N = 8

Step 1: Remainder = 8
Step 2: 2,4,6,8,
Step 3: 1,3,5,7
(1,3) (5,7) $\xrightarrow[\text{pairs}]{\text{switch}}$ (3,1) (7,5)
(2,4,6,8,3,1,7,5)




```
Algorithm NQueens (k,n)
// Using backtracking, this procedure
   prints all
// possible placements of n queens on an
// nxn chess board so that they
// are non-attacking
{
   for i=1 to n do
   {
      if Place(k,i), then
      {
         x(k) = i;
         if (k=n), then write (x[1:n]);
         else NQueen(k+1, n);
      }
   }
}
```

```
Algorithm Place (k,i)
/* Returns true if a queen can be placed
in kth row & ith column. Otherwise it
returns false. x[] is a global array
whose first (k-1) values have been set.
Abs(r) returns the absolute value of r. */
{
   for j=1 to k-1 do
      if ((x[j] = i) // two in the same column
         or (Abs(x[j]-i) = Abs(j-k)))
```

// or in the same diagonal.
then return false;
return true;

Assignment:

Write algorithm & explain to find the max & min. element from a list of elements using D and C technique.

15/12/2021 Wednesday

Sum of Subsets

Suppose we are given n distinct +ve nos and we have to find all combinations of these numbers whose sums are m are called sum of subsets problem

In this problem, we have to find a subset S of given set $S = \{s_1, s_2, \ldots s_n\}$ where the element of set S are n positive integers in such a manner that $s' \in S$ and sum of the ~~subset~~ elements of subset is equal to a +ve integer m.

If a given set $n = \{1, 2, 3, 4\}$ and $m = 5$, then $s' = \{1, 4\}$ or $s' = \{2, 3\}$

The sum of subset pblm can be solved using the backtracking approach. In this implicit trees is created which is a binary tree, the root of the tree is selected in such a way that no decision is yet taken on any input. We assume that the elements of a given set are arranged in an ↑ order.

The left child of the root node indicates that we have to include s_1 from set S and the right child of the root node indicates that, we have to exclude s_1 proceeding to next level.

Starting from the root left child indicates inclusion of s_2 and right child indicates exclusion of s_2. Each node stores the sum of the partial solution elements If at any stage, the number = m then the search is successful and terminates

The dead end in the tree occurs only when either of the following two conditions:

i) Sum of S is too large
ii) Sum of S is too small.

We consider a backtracking using fixed tuple sized strategy. In this case

the element x_i of soln vector is either '1' or '0' depending on whether the weight w_i is included or not.

A bounding function B_k is:

$$B_k(x_1, x_2, \ldots, x_k) = \text{true iff } \sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \leq m$$

Clearly $x_1, x_2, \ldots, x_k$ cannot lead to an answer node if this condition is not satisfied.

Example

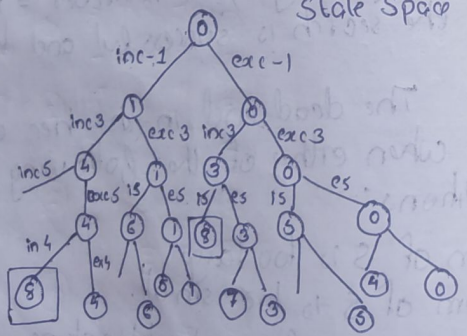
$S = \{1, 3, 5, 4\}$

$m = 8$

$S_1 = \{1, 3, 4\}$

$S_2 = \{3, 5\}$

State Space Tree:



$\{1, 3, 4\} \rightarrow (1, 2, 4)$

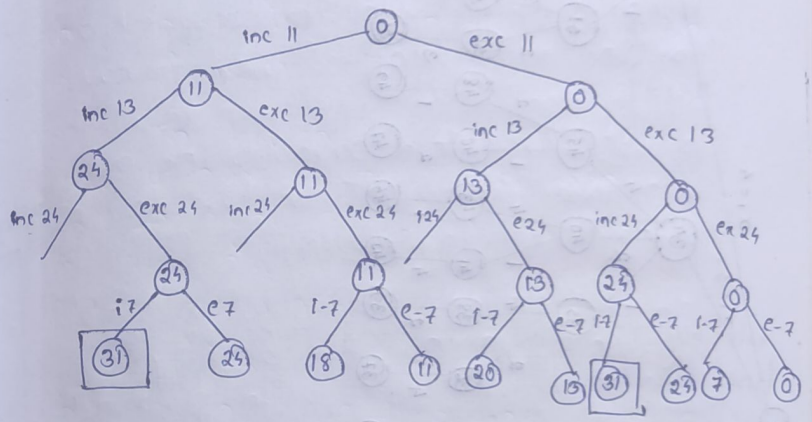
$(1, 1, 0, 1)$

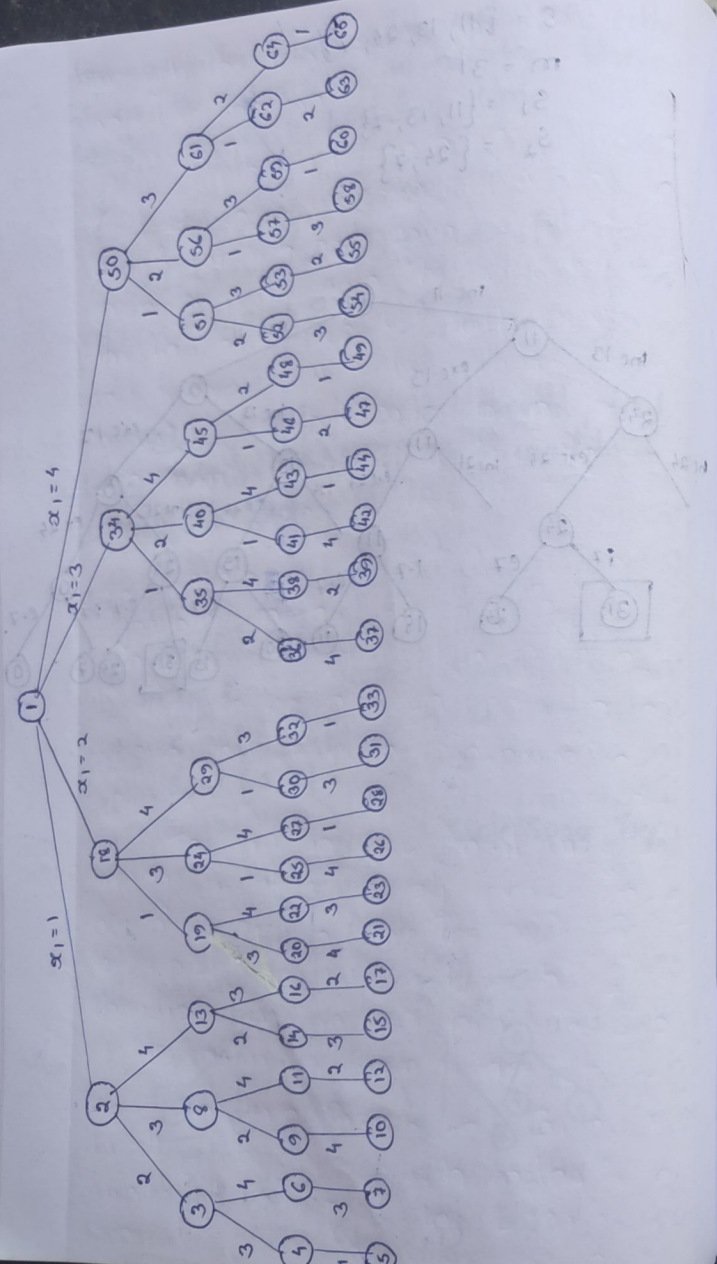
$S = \{11, 13, 24, 7\}$

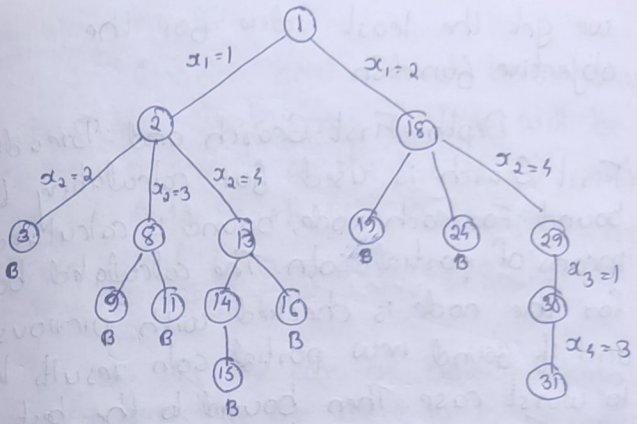
$m = 31$

$S_1 = \{11, 13, 7\}$

$S_2 = \{24, 7\}$







17/12/21
Friday

Branch and Bound.

The Branch and Bound technique like backtracking explores the implicit graph and deals with the optimal soln to a given problem. In this technique at each stage we calculate the bound for a particular node and check whether this bound will be able to give the solution or not.

That means we calculate how far we are from the solution in a graph. If we find that at any node the solution so obtained is appropriate but the remaining solution is leading to a worst case, then we leave this part of the graph without exploring. It can be seen that optimal soln in a implicit graph where

we get the least value for the objective function

Depth-First Search and Breadth First Search is used for calculating the bound. For each node, bound is calculated by means of partial soln. The calculated bound for the node is checked with previous node and if bound new partial soln result lead to worst case. Then bound to the best soln so far selected and we leave this part without exploring further

Branch and Bound is a general algorithmic method for finding optimal soln of various optimization problem. It is basically an enumeration approach in a fashion that prunes the non-promising space tree

The first one is a smart way of covering possible region by several smaller feasible subregion since the procedure may be repeated recursively to each of the sub-regions and all produced sub regions naturally form a tree structure

The term Branch and Bound refers to all State Space Search methods in which all children of the E-node are generated before any other live node can become the E-node

In Branch and Bound terminology a BFS like State Space Search will be called FIFO search as the list of live nodes is a FIFO list. A DFS like State Space Search will be called LIFO search as a list of live node is a LIFO list.

As in the case of backtracking bounding function are used to help avoiding the generation of subtree that donot contain an answer node.

Algorithm LCSearch(t)
// Search t for an answer node.
{
if *t is an answer node then output *t and return;
E = t; // E-node
Initialize the list of live nodes to be empty;
repeat
{
for each child x of E do
{
if x is an answer node then output the path from x to t and return;
Add(x); // x is a new live node.
(x → parent) := E; // Pointer for path to root.
}

```
    if there are no more live nodes then
    {
      write ("No answer node"); return;
    }
    E = Least();
  }
  unbl(false);
}
```

The search for an answer node can often be speeded by using an intelligent ranking ~~branching~~ function $\hat{c}(\cdot)$ for live nodes. The next E node is selected on the basis of this branching function. The ideal way is to assign branchs would be on the basis of the additional computational effort, cost needed to reach an answer node from the live node.

For any node n, the cost would be
i) The no. of nodes in the subtree n that need to be generated before an answer node is generated
ii) The no. of levels the nearest answer node is from x.

Let $\hat{g}(x)$ be an estimate of the additional effort to needed to reach an answer node from x. The node x is assigned a rank using a fn $\hat{c}(\cdot)$ s.

$$\hat{c}(x) = f(h(x)) + \hat{g}(x)$$

where $h(x)$: Cost of reaching x from the root.

function f: Any non-decreasing fn

A search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next ~~key~~ E-node would always choose for its next E-node, a live node with least $\hat{c}$. Hence such a strategy is called an LC search

N^2-1 Puzzle

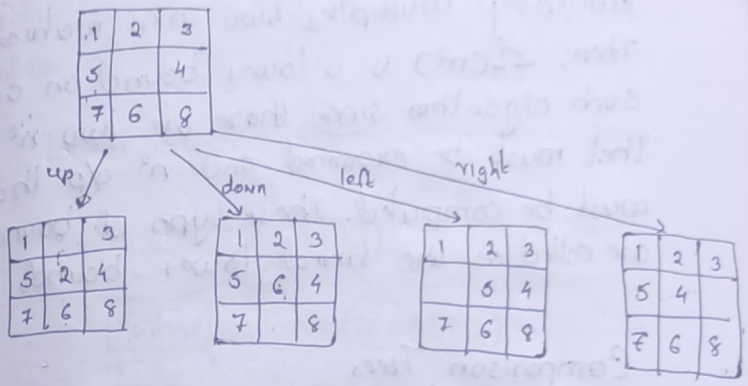
15-Puzzle consist of 15 number tiles on a square frame with a capacity of 16 tiles Our objective is to transform the initial arrangement into the goal arrangement through a series of legal moves. The 15-puzzle has diff. sized variables. The smallest size involved a board 2x3 and is called the 5-puzzle.

The 8-puzzle involves a board 3x3
The 35 puzzle involves a board 6x6.

The family of these puzzle is called as the N-puzzle, where N stands for no of tiles In all of the n-puzzle we use the tiles in the

goal state where ordered from left-right and top-bottom with an empty space located in the bottom right corner. It is known as the family of n-puzzle belongs to the class of NP complete problems. which means that the no. of paths grows exponentially with no. of tiles and finding the shortest path from the start to the goal where required performing an exhaustive search

Thus from the initial arrangement 4 moves are possible. The only legal move are the one in which a tile adjacent to the empty spot is moved to ES. Each move creates a new arrangement & is called the state of the puzzle. The initial and goal arrangements are called the initial and goal states. The state space of an initial state consist of all states that can be reached from the initial state. The most straight forward way to solve the puzzle would be to search the state space for the goal space & use the path from the initial stage to the goal state as the answer



Lower Bound Theory

The concept of lower bound theory establish that the given algorithm is the most efficient possible. The way this is done is by discovering a function g(n) that is a lower bound on the time that any algorithm must take to solve the given problem. If we have an algorithm whose computing time is the same order as g(n) then we know that asymptotically we can do more better

Deriving good lower bounds is more difficult than devising efficient algorithm. However, for many problems, it is possible to easily observe that a lower bound identical to n exists, where n is the no. of I/P. Suppose, we wish to find an algorithm that

efficiently multiplies two nxn matrices. Then, $\Omega(n^2)$ is a lower bound on any such algorithm since there are two n^2 i/p that must be examined and n^2 o/p that must be computed. These types of bounds are called as the trivial lower bounds.

Comparison Trees

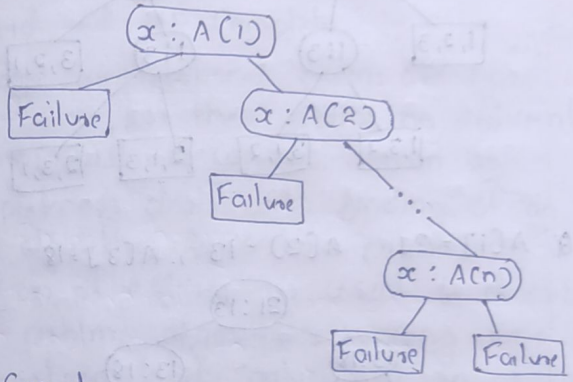
Comparison trees are useful for modeling or deriving lower bounds on problems such sorting and searching.

Suppose that we are given a set S of distinct values on which an ordering relation $<$ holds. The ordered searching pblm ask whether a given element, $x \in S$ occurs within the elements $A[1:n]$. that are ordered so that $A[1] < A[2] < \ldots < A[n]$

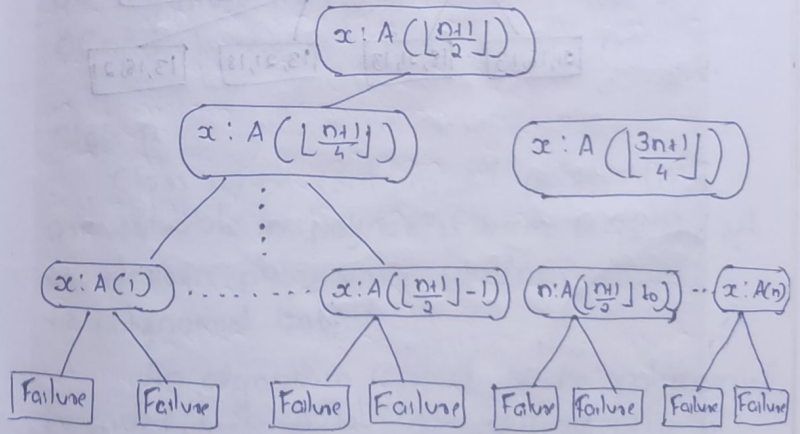
If x is in $A[1:n]$ then we have to determine an i btw 1 and n.
Following tree shows a comparison tree for searching: One using linear search and one using binary search.

The comparison tree for any search algorithm must contain atleast n internal nodes corresponding to n different values of i for which $x = A[i]$

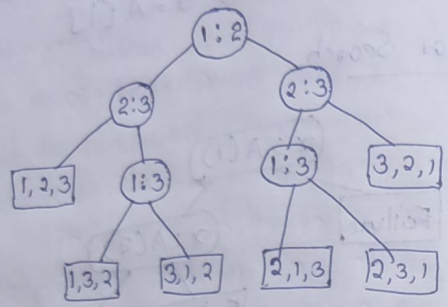
Linear Search



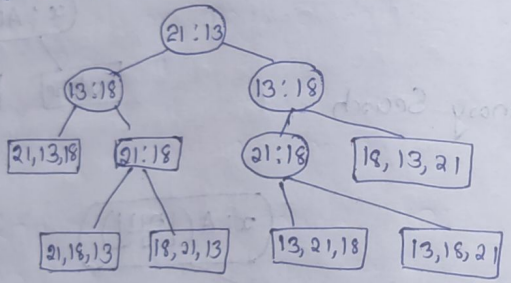
Binary Search



Comparison Tree for Sorting



Q: A[1] = 21 , A[2] = 13 , A[3] = 18



11/01/2022
Tuesday

MODULE :

Class Polynomial And Non-Polynomial {Class P & NP}

NP Hard and NP Complete

Some computabonal pblms are hard and difficult, for these pblms no efficient algorithms exists. A concept known as NP completeness deals with finding of an efficient algorithm for certain problems. The notion of efficient is used to describe that a pblms algorithm's running time is proportional to the polynomial fn of its input size n. Thus for some constant $k > 0$, the algorithm is efficient, if it runs in $O(n^k)$ times for input of size n

Class P

Class which contains all pblms that are solvable in polynomial time. P is the set of decision pblm with yes/no answer, i.e polynomial bound.

An algorithm is said to be polynomial bounded, if its worst case complexity is bound by a polynomial fn P of input size n in that case for each input of size n

the algorithm terminates after almost $p(n)$ steps.
For example
$n^2+13n+10$

V.Imp

Decision Problem

The pblms under this class have a single bit output which shows 0 or 1 i.e. the answer for the pblm is either 0 or 1.
For example some decision pblms are:
1) Given 2 sets of strings S_1 and S_2, then S_2 is a substring of S_1
2) Given 2 set of elements S_1 and S_2. Thus both the set contain same no. of elements

Any problem that involves the identification of an optimal value of a given cost fn is known as optimization pblm.
For example:
Given a weighted graph G, and an integer i. Thus G have a minimal spanning tree of weight of atmost i.

Basic Concepts

Algorithms are divided into 2 gps based on their computing time. The first gp consist of pblms whose soln times are bounded by polynomials of small degree
Eg: Order searching - $O(\log n)$
Polynomial Evaluation - $O(n)$ etc..

The 2nd gp is made up of pblms whose best known algorithms are ~~known as~~ Non-Polynomial.
Eg: Travelling Salesperson pblm - $O(n^2 2^n)$
Knapsack Pblm - $O(2^{n/2})$

V.Imp

There are 2 classes of pblms: np hard and NP complete.

The pblm is NP complete, has the ppty that it can be solved in polynomial time iff all other NP complete pblms can also be solved in ~~NP~~ polynomial time

If an NP hard pblm can be solved in polynomial time. Then all NP complete pblms can be solved in polynomial time. All NP complete pblms are NP hard, but some NP hard pblms are not known to be NP complete

19/01/2022
Wednesday

Deterministic / Non-Deterministic

Non-Deterministic Algorithm

Deterministic Algorithms are algorithms which have the ppty that the result of every operation is uniquely defined

Non-deterministic algorithms are algorithms which contains operations whose outcomes are not uniquely defined but are limited to specified set of possibilities.

To specify such algorithms 3 new functions are introduced:

i) Choice of S

Arbitrarily chooses one of the elements of set S

ii) Failure

Signals a unsuccessful completion

iii) Success

Signals a successful completion

Whenever there is a set of choices that leads to a successful completion, then one such set of choices is always made and the algorithm terminates successfully. A non-deterministic algorithm terminates unsuccessfully iff there is no set of choices leading to a success signal

Algorithm Non-Deterministic_Search

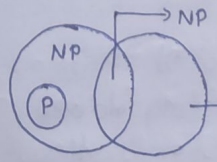
```
1 j = choice (1,n)
2 If a(j) = S
    {
      write(j)
      success;
    }
3. write(0); failure
```

The classes NP hard & NP complete

P is the set of all decisions solvable by deterministic algorithms in polynomial time
NP is the set of all decision pblms solvable by non-deterministic algorithms in polynomial time. ie $P \subseteq NP$.

Relationship b/w P and NP

Relationship b/w P, NP, NP complete & NP hard



There are NP hard pblms, that are not NP complete. Only a decision pblm can be NP complete. An optimization pblm may be NP hard.

Maximum Clique Problem

A maximal complete subgraph of a graph $G = (V, E)$ is a clique, the size of the clique is the no. of vertices in it. The max clique pblm is an optimization pblm, that has to determine the size of largest clique in it. The corresponding decision pblm is to determine whether G has a size of at least k for some given k.

1. What do you understand by the terms space complexity and time complexity? Point out the asymptotic notations (5 mark)

Time Complexity

The time T(P) taken by a program P is the sum of the compile time and run time. Run time is denoted by t_p. If we know the characteristics of a compiler to be used we could proceed to determine the no. of additions, subtractions, multiplications, divisions and so on. that would be made by the code for P.

$t_p(n) = C_a ADD(n) + C_s SUB(n) + C_M MUL(n) + \ldots$

where C_a, C_s, C_m : Time need for addition, subtraction, multiplication etc.

ADD, SUB, MUL : Functions

Space Complexity

A fixed part that is independent of the characteristics of the i/p and o/p. The part includes the instruction space for simple variables and fixed size component variables, space for constants and so on

A variable part consist of the space needed by the component variables whose size

is dependent on particular problem instance being solved, the space needed by referenced variables and recursion stack space

$S(P) = C + S_P$ where C is constant

Asymptotic Notations

Asymptotic notation is a shorthand way to write down and talk about fastest possible and shortest possible running time for an algorithm using high and low bound speed

Big Oh (O)

The function $f(n) = O(g(n))$ iff $\exists$ a +ve constant c and $n_0, n_0 \geq$

$f(n) \leq c \times g(n) \quad \forall n, n \geq 0$

Omega (Ω)

The function $f(n) = \Omega(g(n))$ iff $\exists$ a positive constant c and n_0, $\ni$

$f(n) \geq c \times g(n) \quad \forall n, n \geq 0$

Theta (Θ)

The function $f(n) = \Theta(g(n))$ iff $\exists$ a +ve constants c_1 & c_2 and n_0, $\ni$

$c_1 g(n) \leq f(n) \leq c_2 (g(n)) \quad \forall n \; ; n \geq 0$

2 Write down the control abstraction of divide and conquer (3 marks)

Control Abstraction of divide and conquer.

Control abstraction for divide and conquer technique is DANDC(P), where P is the problem to be solved

```
Algorithm DANDC(P)
{
  if small(P)
    then return s(P);
  else
  {
    divide P into smaller instances
          P1, P2, P3 ---- Pk ; k ≥ 1;
    apply D and C to each of these subproblems
    return combine (DANDC(P1),
          DANDC(P2), ..., DANDC(Pk));
  }
}
```