the system and this continues.

→ Priority Queues are used to perform greedy algorithm

→ Single Source Shortest Path problem.

15/03/2023

UNIT-1

→ Algorithm: finite set of instructions to perform a particular task.

Criteria:

→ Input & Output:

→ Definiteness: unambiguous & clear

→ Finiteness: amount of time taken for execution

→ Effectiveness: basic nature

→ ~~Appli~~ Areas of Study/Research:

* How to device algorithm acc. to problem.
* Validate the algorithm i.e. to look for correctness of algorithm.
* Analysis of algorithm i.e. to check for the complexity of the algorithm.
* Test the algorithm; Debugging - check for the faults; & changing them; Profiling - information used ~~for~~ to describe the code

→ Algorithm Specification:

1) Natural English language: every step is definite.
2) Graphic representation: flowchart - small & simple.
3) Pseudo-Code Method:-

↔ Pseudo Code conventions: * A[i], A[i][j]

1) Comments begin with // and continue until the end of line
2) Blocks are indicated with { }
3) An identifier begins with letter & datatypes are not explicitly declared.
4) Compound datatypes can be formed with records.

Ex: Node Record
{ datatype-1 data-1;
dt-2 d-2;
⋮
dt-n d-n;
node *link; }

5) <Variable> := <expression> → assignment

6) 2 Boolean values True & False:
- Logical: AND, OR, NOT
- Relational: <=, >=, ==, !=

7) Looping Statements.
8) Conditional Statements.
9) I/P & O/P are done using instructions read & write.
10) only one procedure: Algorithm

Algorithm Name(parameters_list)

*Looping Statements:
- For, While, Repeat-until

* While loop:

while <condition> do
{ <Statement>; }.

* For Loop:

to increment decrement

For variable := value-1 to value-2 step step
do
{
<statement-1>
⋮
<statement-n> }

* repeat-until:

repeat
{ <statement-1>
.
.
.
<statement-n>
}
until <condition>

* Conditional Statements:

→ If <condition> then <statement>

→ If <condition> then <statement-1>
Else <statement-1>

→ case statement:

Case
{ :<condition-1> :<statement-1>
.
.
.
:<condition-n>:<statement-n>
: else : <statement_n+1> }

→ Write an algorithm to find sum of array of elements where A is the array and n is the no. of elements.

Algorithm Sum(A,n)
{ // Let array A and n → no. of elements
sum:=0;
for i:= 0 to n-1 step step1 do
{ read A[i]
sum = sum + A[i];
}
write sum;
}

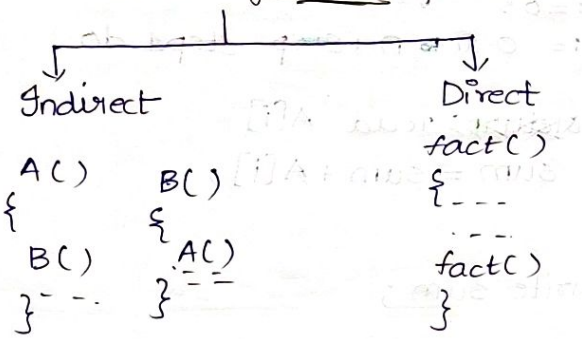
→ Write an algorithm to perform selection sort.

Algorithm Selection(A,n)
{// Let array A and n is no. of elements
min:=0; temp:=0
for i:= 0 to n-1 do
{ min:=A[i];
for j:= i+1 to n do
{ if < A[j] < min] > {min=j;}
temp=A[j]
A[j] = A[j+1]; } }

~~min = A[i]~~

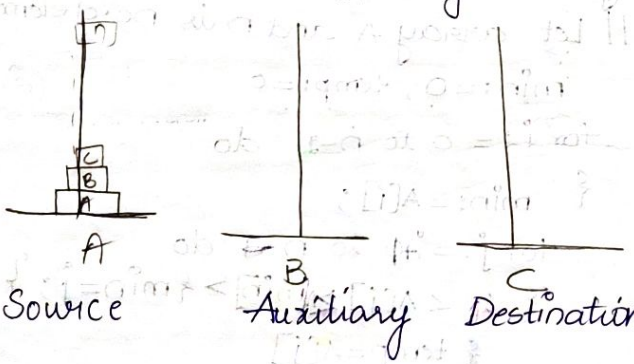
16/3/2023

* Recursive Algorithms:



* → Recursive algorithms are easy to understand rather than iterative ones.

→ Towers of Hanoi: only small discs are placed on large discs.



1) move (n-1) discs from A to B by C is auxiliary

2) move the top disc from A to C

3) move (n-1) discs from B to C by A is auxiliary.

→ Algorithm TOH(n,A,B,C)

{ if (n≥1) $n=0 \rightarrow t_n = 1$

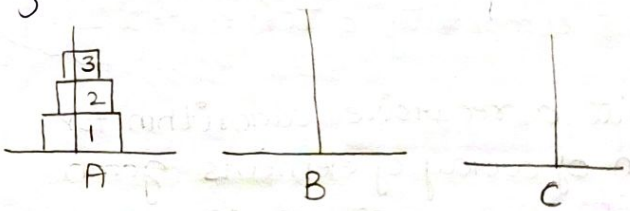
{ TOH(n-1, A, C, B) t(n-1)

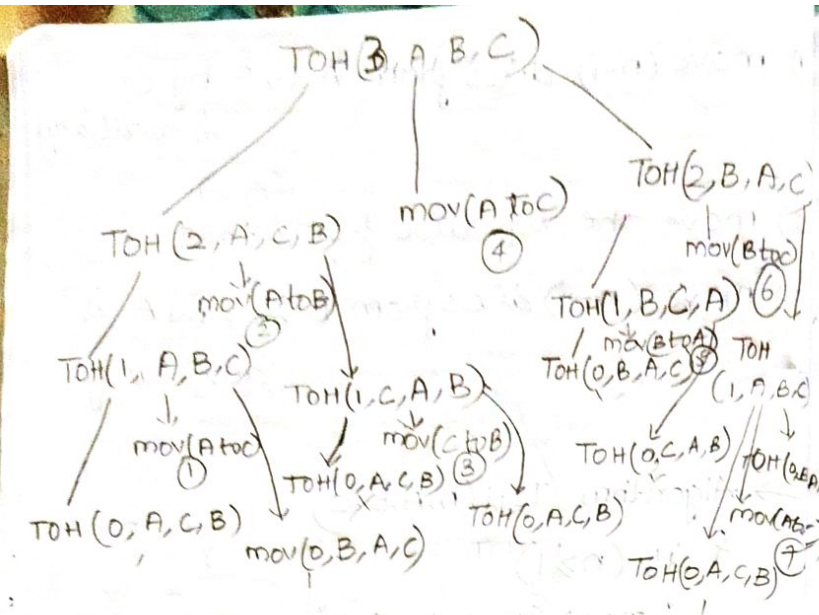
write("Move the top disc from A to C")

TOH(n-1, B, A, C); t(n-1)

} }

}





→ Time complexity of TOH:

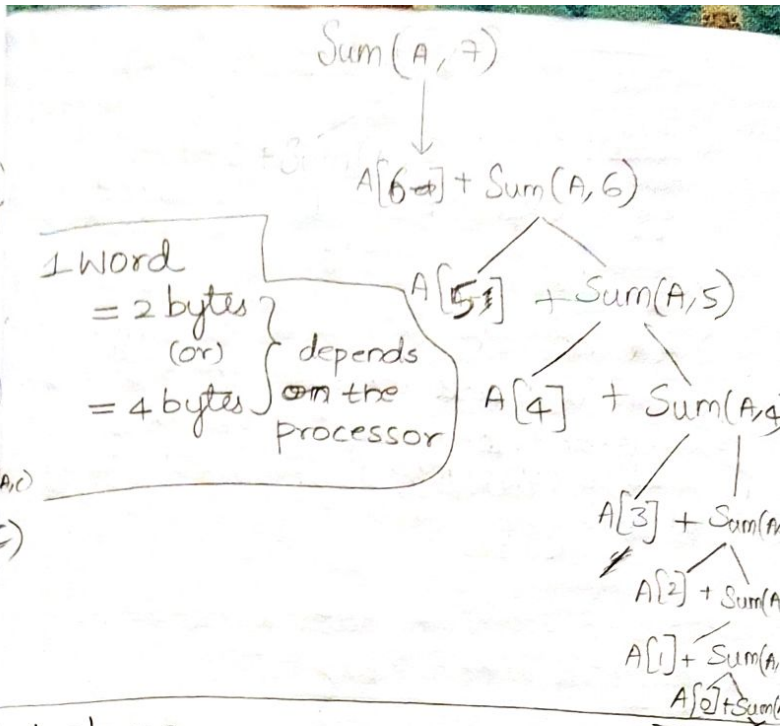
if $n=0 \longrightarrow t_n = 1$

if $n \geq 1 \longrightarrow t_n = 2 + 2t(n-1)$

time complexity of TOH $\Rightarrow t_n = 2^n - 1$

* Write a recursive algorithm for sum of array of elements given as 1, 5, 9, 15, 23, 45, 62.

```
Algorithm Sum(A, n)
{
    if <n==1> return A[n-1];
    else return A[n-1] + Sum(A, n-1);
}
```



17/03/2023

* PERFORMANCE ANALYSIS:

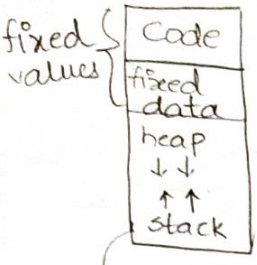
→ Prior Analysis

→ Postirior Analysis

→ Space Complexity: S(P): Amount of memory taken by the algorithm to complete the task.

S(P) = fixed path + variable path.

$S(P) = c + S_p$ $\quad S(P) \geq S_p$



* Stack & Heap areas cannot be fixed because they depend on instance variables.

recursion stack which stores the diff. fn calls and increases the size as and when required. & when pop() operation is performed size of stack decreases

push() operation

→ heap area stores the data whose memory is allocated dynamically

* Space complexity is measured in terms of words.

→ if c is negligible : $S(P) \cong S_p$

else : $S(P) \geq S_p$

* Write an algorithm to find the sum of array of elements.

```
Algorithm sum(A,n)
{  sum := 0
   {if (n==0) then  sum == 0
    else
    for i:=0 to n-1 do
        sum := sum + A[i]
    Write sum
}
```

sum = 1
i = 1
n = 1 → instant character
A[ ] = n
∵ A depends on n

$S(P) = n+3$

```
Algorithm Expr(a,b,c)
{
return a+(b*c)-(a+c)*(b-c)+(a-b)}
```

a b c r
$S(P) = 1+1+1+1$
$= 4$

* Write an algorithm to find the sum of 2 matrices of size m×n and then find the space complexity.

Algorithm MatrixSum(A, B, m, n)

```
{   C[m][n] := {0};
    for i := 0 to m-1 do
    { for j := 0 to n-1 do
      { C[i][j] = A[i][j] + B[i][j]
      } }
```

C → m×n words
A → m×n words
B → m×n words
i → 1
j → 1
m → 1
n → 1

$S(P) = 4 + 3mn$

*Write a recursive algorithm to find the sum of array of elements and find the space complexity.

sum = 0

Algorithm RSum(A, n)

```
{ if (n <= 0)
      return 0;
  else
      return (A[n-1] + RSum(A, n-1))
}
```

→ Array A is located only once and is accessed using the base address.

n = 1
return = 1
ptr to array (base address) = 1

A[ ] = n

$S(P) \geq n + 3$

RSum(A, 3) (6)
A[2] + RSum(A, 2)
3 3
A[1] + RSum(A, 1)
2 1
A[0] + RSum(A, 0)
1 + 0

→ Time Complexity: time taken by the algorithm to complete a task.

T(P) = Compile Time + Execution time

$T(P) = C + T_P$ $\quad T(P) \geq T_P$

↳ Counting Method,
↳ Step-Table Method.

// → execution time = 0

c := a + b → 1
for, while; do while → f(n) → steps
no. of iterations.

if <condition> then
<statement> } if true } 1+x
else
<statement> } else } 1+y

* Write an algorithm to find sum of array of elements and find the time complexity of the algorithm using count method:

```
count:=0
Algorithm Sum(A, n)
{  sum:=0
   count++;×1
   if (n==0) then
   count++;×1
      return 0;  → count++×1
   else
      { for i:=0 to n-1 do
                  count++×(n+1)
        { sum = sum + A[i]
          count++ × n.
        }}
   Write sum
   count++;×1
}}
```

T(P) ≥ 4 + n+1 + n

T(P) ≥ 2n + 5

18/03/2023

* Find the time complexity of the algorithm RSum which is used to find the recursive sum of array elements.

```
count:=0  sum:=0
Algorithm RSum(A, n)
{  if (n<=0) then → count++
      return 0; count++;
   else
      return (A[n-1] + RSum(A, n-1));
                       count++
}
```

T(P) = 2 n ≤ 0

T(P) = 2 + t(n-1) n ≥ 0

= 2 + 2 + t(n-2)

= 2 + 2 + 2 + t(n-3)

= ⋮

= 2 + 2 + ⋯ + t(0)

= 2 + 2 + ---- + 2

T(n) = 2(n+1)

* Write a recursive algorithm to find a given element in a sorted list.

count := 0

```
Algorithm bsearch(A, l, h, x)
{  if(l ≤ h)  --> count++   h-l=n
   {  then
      mid = (l+h)/2  --> count++;
      if (A[mid] == x) then  --> count++;
         return mid;  --> count++;
      else if (A[mid] > x)
         return (A, l, mid-1, x);   n/2  --> count++
      else
         return (A, mid+1, h, x);   n/2  --> count++
   } if (l > h)
   return -1;  --> count++;
}
```

T(P) >= 4 + t(n/2)

* Step Table:

Algorithm	s/e	frequency
Algorithm Sum(A,n)	0	1
{	0	0
sum:0	1	1
for(i:=0 to n-1) do	1	n+1
sum:=sum+A[i]	1	n
write sum	1	1
}	0	1

t(n) = 1 + n+1 + n + 1

= 2n+3

* Asymptotic Notations:

used to represent time complexity:

→ 3 methods.

1) big-O (O): upper bound of a function.
2) Omega(Ω):
3) Theta (Θ)

* Big-O:

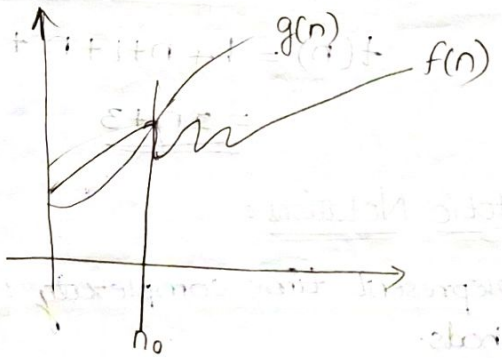
$f(n), g(n)$ are non-negative functions

$f(n) = O(g(n))$ if and only if there exists 2 +ve constants c_0 & n_0 $c > 0$ & for all $n \geq n_0$; $f(n) \leq c_0 g(n)$ then $f(n)$ is big O of $g(n)$.

$f(n) = 2n+3 \quad g(n) = 2n+3 = 5n$

$2n+3 \leq 5n \quad - (1)$

$c = 5$

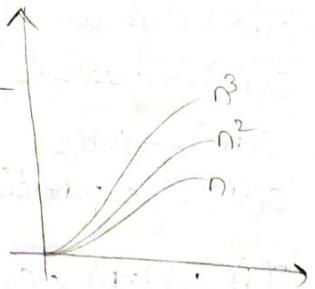


initial value of n when $c\,g(n)$ is greater than $f(n)$.

as $f(n)$ can be written as $O(n), O(n^2), O(n^3)$ etc but we need to write $f(n) = O(n)$ as it is better.

$f(n) = 10n^2+4n+2 \leq 11n^2$

$n_0 = 5.$



* Omega:

$f(n) = 2n+3$

$2n+3 \geq 2n \qquad n \geq 0$

$f(n) = \Omega(g(n))$

* Theta:

If $g(n)$ is Big-O & $g(n)$ is lower bound is same then there is tight bound to for the given $f(n)$

$f(n) = \theta(g(n))$

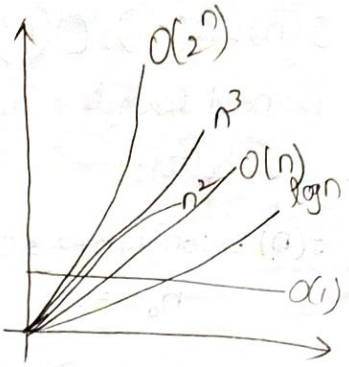
If & only if $G C_2 > 0$

$\forall\ n \geq n_0$

$G g(n) \leq f(n) \leq C_2 g(n)$

as n↑ses; value ↑ses exponentially

$O(1)$ → const.
$O(\log n)$ → logarithmic
$O(n)$ → linear
$O(n^2)$ → quadratic
$O(n^3)$ → cubic.
$O(2^n)$ → exponential.



$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n) < O(n^2) < O(n^3) < O(2^n) < \cdots O(n^n)$.

* Properties of Asymtotic Notations:

→ Reflexive: f(n) is non-negative; then fn.
$f(n) = O(f(n))$.

→ Transtlive: $f(n) = O(g(n))$ & $g(n) = O(h(n))$
then $f(n) = O(h(n))$.

→ Symmetric: $f(n) = \Theta(g(n))$, $g(n) = \Theta(f(n))$

→ Transpose Symmetric:
$f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$
and vice versa.

→ Addition & ~~Subtraction~~ Multiplication Property

if $f(n) = O(g(n))$ $d(n) = O(e(n))$
$f(n) + d(n) = O(\max(g(n), e(n)))$
$f(n) \cdot d(n) = O(g(n) \cdot e(n))$.

23|03|2023

* Write an algorithm for matrix multiplication and compute the time complexity.

Algorithm

Algorithm MM(int a[m][n]

Algorithm MatrixM(int a[m,n], int b[n,n])
{ c[i,j]:={0};
for i:=0 to m-1 → n
{ for j:=0 to n-1 → n
{ for k:=0 to n-1 → n
{ c[i,j]:=c[i,j]+(a[i,k]*b[k,j]);
}}}
} $O(n^3)$

* Amortized Analysis:

→ Average Cost of all the n operations is given by Amortized Analysis.

*→ There are 3 methods:

1) Aggregate
2) Accounting
3) Potential

→ Amortized analysis means finding avg running time per operation over a worst case sequence of operations

Eg:

$I_1, I_2, D_1, I_3, I_4, I_5, I_6, D_2, I_7$

Insert $I_j = 1$ → actual cost

Delete $D_1 = 8$

Delete $D_2 = 10$

total = 7×1 + 8 + 10 = 25

$P(0) = 0$

* Potential function P(i):

$$P(i) = \text{amortized cost}(i) - \text{Actual cost}(i) + P(i-1)$$

$$\sum_{i=1}^{n} \text{amortized cost}(i) \geq \sum_{i=1}^{n} \text{actual cost}(i)$$

if $\sum_{i=1}^{n} \text{amortized cost}(i) < \sum_{i=1}^{n} \text{actual cost}(i)$

then the computational time given for set of operations is not sufficient to perform them.

→ average amortized cost = $\frac{25}{9}$ = $\frac{\text{total actual cost}}{\text{no. of operations}}$

	I_1	I_2	D_1	I_3	I_4	I_5	I_6	D_2	I_7
actual cost	1	1	8	1	1	1	1	10	1
amortized cost	3	3	3	3	3	3	3	3	3
Potential fn	2	4	−1	1	3	5	7	0	2

$$P(n) - P(0) = \sum_{i=1}^{n} [\text{amortized}(i) - \text{actual}(i)]$$

$P(n) - P(0) \geq 0$ $\quad P(n) \geq 0$

* Find the amortized cost of a maintainence contract given by a car dealer.

The dealer charges $50 in the months Jan, Feb, Apr, May, July, Aug, Oct & Nov. and $100 in the months Mar, Jun & Sept & $200 for every december

	1	2	3	4	5	6	7	8	9	10	11	12
Actual	50	50	100	50	50	100	50	50	100	50	50	200
Amortized	75	75	75	75	75	75	75	75	75	75	75	75
P(i)	25	50	25	50	75	50	25	100	75	100	125	0
	25	40	10	30	50	20	40	60	35	60	70	

total cost = 8x50 + 3x100 + 200

= 400 + 300 + 200

= 900

average = $\frac{900}{12} = \underline{75}$

P(n) = 0

Accounting method is used check for amortized cost which is less than amortized cost found out in Aggregate method; that gives correct amortized cost

→ Potential Method:

$$P(i) = \begin{cases} 0 & ; n \bmod 12 = 0 \\ 25 & ; n \bmod 12 = 1/3 \\ 50 & ; n \bmod 12 = 2,4,6 \\ 75 & ; n \bmod 12 = 5,7,9 \\ 100 & ; n \bmod 12 = 8,10 \\ 125 & ; n \bmod 12 = 11 \end{cases}$$

* MASTER'S THEOREM: 24/03/2023

→ used to express recurrence relations which includes divide and conquer rule.

$$T(n) = a\,T(n/b) + f(n)$$

$a \geq 1, b \geq 1; f(n) = \theta\left(n^k (\log n)^p\right)$

** Case(i) if $\log_b a > k$ then

~~$T(n) = \theta(n^k \log_b a)$~~

$T(n) = \theta\left(n^{\log_b a}\right)$

** case(ii) if $\log_b a = k$ then

a) $p > -1$; then $T(n) = \theta\left(n^k (\log n)^{p+1}\right)$

b) $p = -1$; then $T(n) = \theta(n^k \log(\log n))$

c) $p < -1$; then $T(n) = \theta(n^k)$

→ Case(iii)

if $\log_b^a < k$ then

a) if $p \geq 0$ then

$T(n) = \theta(n^k (\log n)^p)$

b) if $p < 0$ then

$T(n) = O(n^k)$

* $T(n) = 2T(n/2) + 1$

$a = 2 \quad b = 2$

$f(n) = 1 = \theta(n^k (\log n)^p)$

$k = 0 ; p = 0$

$\log_b^a = 1 < k$

$\therefore T(n) = \theta(n^{\log_b^a})$

$\boxed{T(n) = \theta(n)}$

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

$a = 4 \quad b = 2 \quad f(n) = n$

$n = \theta(n^k (\log n)^p)$

$\log_b^a = \log_2^4 = 2 < k$

$p = 0 ; k = 1$

$\boxed{T(n) = \theta(n^2)}$

* $T(n) = 2T(n/2) + \frac{n}{\log n}$

$f(n) = \frac{n}{\log n} = \theta(n^k (\log n)^p)$

$p = -1 ; \quad k = 1$

$\log_b^a = 1 = 1$

$p = -1$

$T(n) = \theta(n^k \log(\log n))$

$\boxed{T(n) = \theta(n \log(\log n))}$

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

$a = 1 ; \quad b = 2$

$\log_b^a = \log_2^1 = 0$

$n^2 = \theta(n^k (\log n)^p)) \Rightarrow k = 2, \; p = 0$

$T(n) = \theta(n^k (\log n)^p)$

$\boxed{T(n) = \theta(n^2)}$

* $T(n) = 4T(n/2) + 1$

$a = 4 \quad b = 2$

$\log_b^a = 2 > k$

$1 = \theta(n^k (\log n)^p)$

$k = 0; p = 0$

$\boxed{T(n) = \theta(n^2)}$

24|03|2023 UNIT-2

** DIVIDE & CONQUER:

→ control abstraction : gives the algorithm to be written for a ~~code~~ problem.

** DAC(P)
{ if small (P) then
return solution (P)
else
Divide the problem into
$(P_1, P_2, \ldots, P_n)$
combine sol^n (DAC(P_1), DAC(P_2)...

→ Nature of Problem should remain the same but the amount of data can be reduced while performing divide & conquer rule.

25|03|2023

* Binary Search:

Algorithm BS(a, n, x)
{ low := 1; high := n+1;
while (low < (high-1)) do

Algorithm BS(a, i, l, x)
{ if (l = i) then //small (P)
{ if (x = a[i]) then
return i;
else return 0; }
else
{ mid := $\lfloor (i+l)/2 \rfloor$
if (x = a[mid]) then return mid.
else if (x < a[mid]) then
return BS(a, i, mid-1, x)
else return BS(a, mid+1, l, x); }

→ We express time complexity of the binary search in terms of no. of element comparisons.

*

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-15,	-6,	0,	7,	9,	23,	54,	82,	101,	112,	125,	131,	142,	151

* ddⁿ: x = 9

BS(a, 0, 13, 9)

mid = ⌊13/2⌋ = 6

if (9 = 54 ✗F

9 < 54

= BS(a, 0, 5, 9)

mid = ⌊5/2⌋ = 2

if (9 = 0 ✗F

9 < 0 ✗F

9 > 0

= BS(a, 3, 5, 9)

mid = ⌊3+5/2⌋ = 4

if (9 = 9) → T

⇒ return 4

* x = 82

BS(a, 0, 13, 82)

mid = 6

if (82 = 54 ✗F

82 < 54 ✗F

BS(a, 7, 13, 82)

mid = 10

82 = 125 ✗F

82 < 125

BS(a, 7, 9, 82)

mid = 8

101 > 82

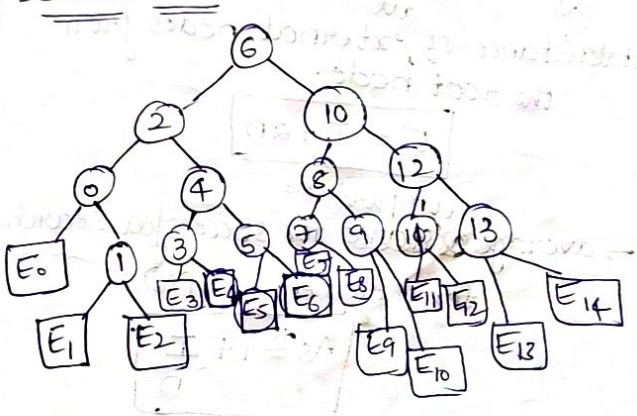
BS(a, 7, 9, 82)

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-15,	-6,	0,	7,	9,	23,	54,	82,	101,	112,	125,	131,	142,	151
3	4	2	4	3	4	1	4	3	4	2	4	3	4

$$Avg = \frac{3+4+2+4+3+4+1+4+3+4+2+4+3+4}{14}$$

$$\simeq 3.2 \simeq \log_2 14$$

Binary Decision Tree:



→ to search the element which is at i^{th} node ; then we need to do $(i-1)$ element comparisons.

→ In this case ; we need
min 3 comparisons & max 4 comparisons.

⇒
unsuccessful search
time complexity $= \frac{4 \times 14 + 3}{15}$
$= 3.9$.

* Internal Path length (I) → total distance of all internal nodes from the root node.
External Path length (E)
↓
total distance of all external nodes from the root node.

$$E = I + 2n$$

→ average comparisons for successful searches for n elements : (A_s)

$$A_s(n) = 1 + \frac{I}{n}$$

→ average comparisons for unsuccessful searches for n elements.

$$A_u(n) = \frac{E}{n+1}$$

$$A_s(n) = 1 + \frac{E-2n}{n}$$

$$= 1 + \frac{E}{n} - 2 = \frac{E}{n} - 1$$

$$A_s(n) = \frac{E-n}{n}$$

$$A_s(n) = \frac{A_u(n) \cdot (n+1)}{n} - 1$$

$$A_s(n) = \frac{(A_u(n))(n+1) - n}{n}$$

$$n(A_s(n)+1) = (A_u(n))(n+1)$$

$$A_s(n) + 1 = A_u(n)\left[1 + \frac{1}{n}\right]$$

→ If n = no. of elements is in the range of 2^{k-1} to 2^k ; then comparisons for successful searches = k.

Comparisons for unsuccessful searches $= k-1$ / k

average time complexity cases

successful & unsuccessful searches

complexity $= \theta(\log_2 n)$

27/03/2023

* Finding maximum & minimum:

```
Algorithm maxmin(i, j, max, min)
{ if (i==j) then max=min=a[i]          // looping variables not counted as comparisons >> 1 element
  else if (i==j-1) then
  { if (a[i]<a[j]) then                // element comparisons
    { max:=a[j];
      min:=a[i]; }
    else
    { max:=a[i];
      min:=a[j]; } }
```

2 elements

```
  else
  { mid:=⌊(i+j)/2⌋;                    // 2a elements
    maxmin(i, mid, max, min);
    maxmin(mid+1, j, max1, min1);
    if (max<max1) then max:=max1;
    if (min<min1) then min:=min1;
  }}
```

* Write an algorithm to find min & max without divide & conquer.

```
Algorithm maxmin(int a[], int n)
{ max=min=a[0];
  for i:=a to n-1
  { if (a[i]<min)
      min=a[i];
    else
      max=a[i]; }
}
```

time complexity $= O(n)$

** No. of element comparisons:

$n=1$ element $T(n)=0$ comparisons

$n=2$ elements $T(n)=1$ comparison

$n>2$ elements $T(n)=t(n/2)+t(n/2)+2$

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

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

$a=2$ $f(n)=2$

$b=2$

$\log_b^a=1$

$2=\theta(n^k(\log n)^p)$

$\log_{10}^2=k\log_{10}n+p\log_{10}(\log n)$

$\log\frac{2}{n^k}=p\log(\log n)$

$\frac{2}{n^k}=(\log n)^p$

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

$a=2$ $\log_b^a=1$

$b=2$

$f(n)=2$

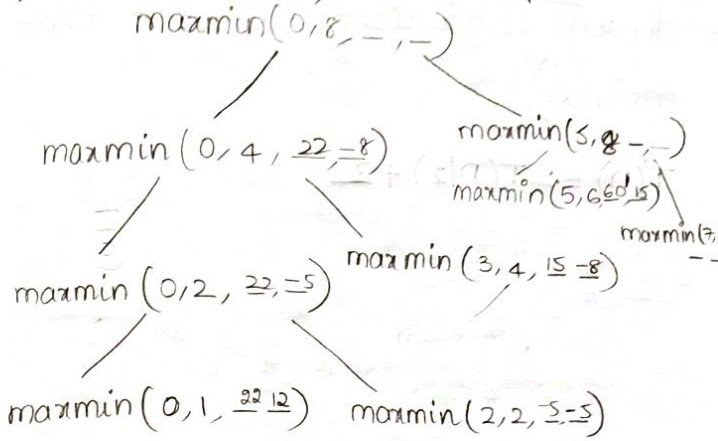
$k=0;\ p=0$

$\Rightarrow k<\log_b^a$

$T(n)=\theta(n^{\log_b^a})$

$=\theta(n)$

22, 12, -5, -8, 15, 60, 17, 31, 47.



→ Consider the recurrence relation and solve by using repeating substitution and compare this complexity with straight forward maxmin algorithm (compute time complexity by considering only one element comparisons using step-table method).

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

Straight forward

```
for i:=2 to n
{ if(a[i]<min) ---> (n-1)
    min = a[i]
  if (a[i] >max) ---> (n-1)
    max = a[i]
```

$T(n) = (n-1) + (n-1)$

$= 2(n-1)$

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

Repeated Substitution

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

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

$= 8T(n/8) + 8 + 4 + 2$

$= 16T(n/16) + 16 + 8 + 4 + 2$

$n = 2^k$

$= 2^{k-1} T(n/2^{k-1}) + \sum_{n=1}^{k-1} 2^i$

$T(2)$

$= 2^{k-1}(1) + \frac{2(2^{k-1} - 1)}{2-1}$ → sum of GP

$= 2^{k-1} + 2^k - 2$

$= \frac{2^k}{2} + 2^k - 2$ $\quad 2^k = n$

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

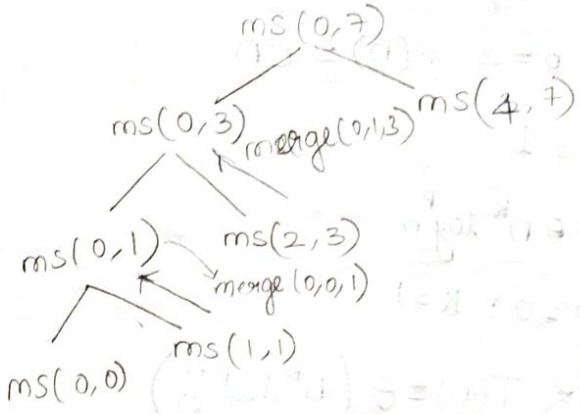
$T(2^k) = 2^{k-1} T(2) + \sum_{n=1}^{k-1} 2^i$

* Algorithm: Merge sort:

→ Algorithm Merge(low, mid, high)

```
{ h:= low; i:= low; j:= mid+1;
  while((h≤mid) and (j≤high))
    do
  { if (a[h] ≤ a[j]) then
    { b[i]:= a[h]; h:= h+1;}
    else
    { b[i]:= a[j]; j:= j+1;}
    i:= i+1;
  }
  if(h> mid) then
    for k:= j to high do
    { b[i]:= a[k]; i:= i+1;}
  else
    for k:= h to mid do
    { b[i]:= a[k]; i:= i+1;}
for k:= low to high do
    a[k]:= b[k]; }
```

7, 2, 9, 4, 3, 8, 6, 1.



→ Algorithm MergeSort(low, high)

```
{ if (low < high)
  { mid:= ⌊(low+high)/2⌋
    MergeSort(low, mid)
    MergeSort(mid+1, high)
    merge(low, mid, high)}
}
```

Time complexity:

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

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

$a=2 ; b=2 \quad f(n)=cn$

$\log_b^a = 1$

$cn = n^k \log^p n$

$p=0 ; k=1$

$T(n)=\theta(n^k \log^{p+1} n)$

$\boxed{T(n)=\theta(n \log n)}$

Merge Sort Repeated Substitution.

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

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

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

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

$T(n)=4(2T(n/8)+cn/4)+2cn$

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

$n=2^k$

$\log_2^n = k$

$T(n)=2^k T(1)+kcn$

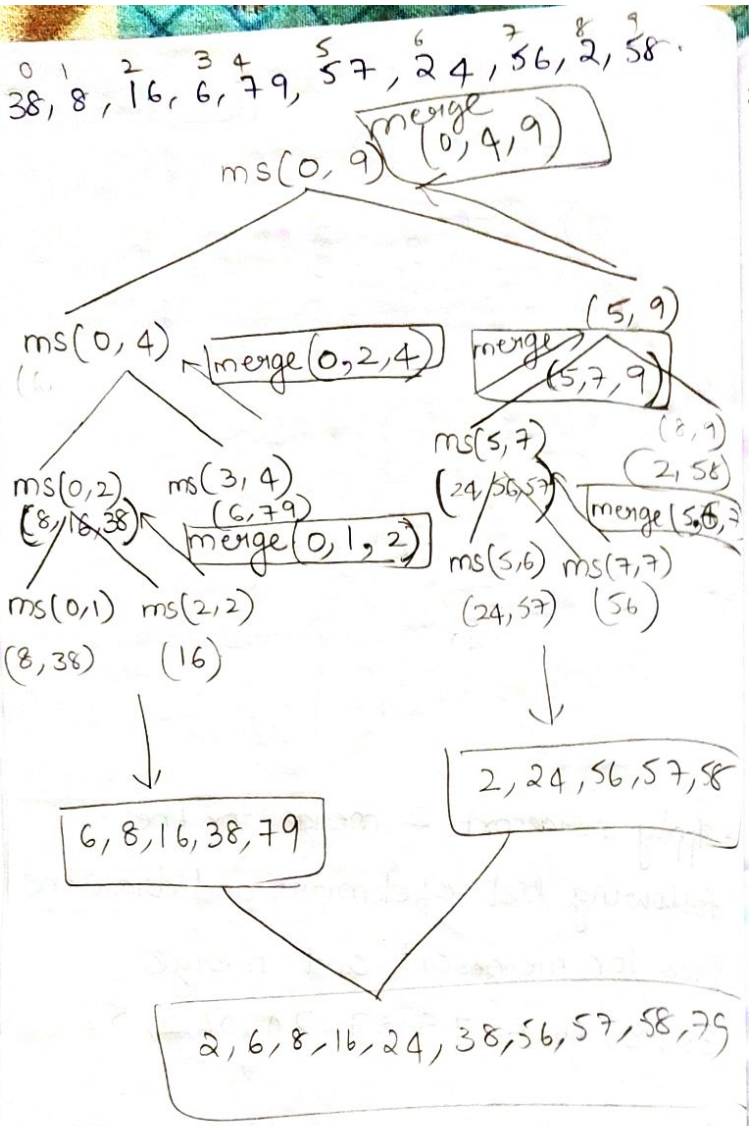
$=2^k a + kcn$

$=an + cn \log_2^n$

$\boxed{T(n) \approx n \log_2^n}$

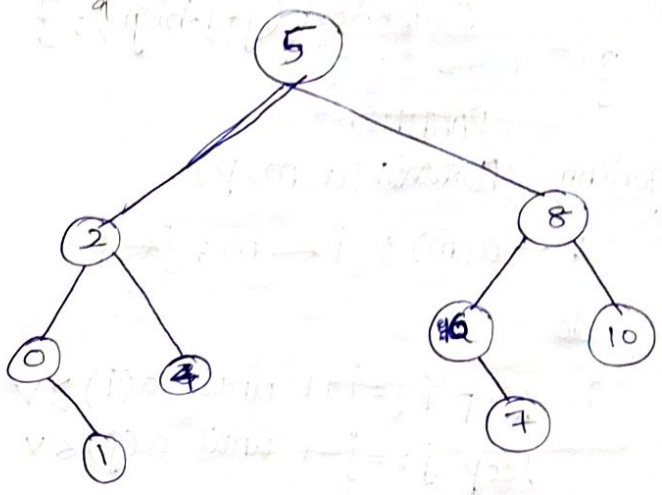
→ Apply mergesort & merge for the following list of elements and draw the tree for mergesort and merge

38, 8, 16, 6, 79, 57, 24, 56, 2, 58.



*Find avg no. of comparisons in case of successful & unsuccessful searches for following elements using Binary decision tree.

0 1 2 3 4 5 6 7 8
-25, -12, 3, 14, 19, 23, 52, 76, 94,
9 10 11
112, 152, 179 n=12



03/04/23

* QUICKSORT:

```
Quicksort(low, high)
 { if low < high then
    { j := PARTITION(a, low, high+1)
      QuickSort(low, j-1);
      QuickSort(j+1, high); }
 }
```

PARTITION

```
algorithm Partition(a, m, p)
  v ← a(m); i ← m; j ← p;
  do
  { loop i := i+1 until a(i) ≥ v
    loop j := j-1 until a(j) ≤ v
    if (i<j) then INTERCHANGE(a,i,j)
  } while(i ≥ j);
  a[m] := a[j];
  a[j] := v;
  return j;
```

38, 8, 16, 6, 79, 57, 24, 56, 2, 58, 4, 70, 45, +∞

24, 8, 16, 6, 4, 2, (38), 56, 57, 58, 79, 45, ∞

(2), 8, 16, 6, 4, (24), (38), 56, 57, 58, 79, 45, ∞

(2), 8, 16, 6, 4, (24), (38), 56, 57, 58, 79, 45, ∞

→ at most no. of comparisons for partition call

(2), (4), 16, 6, 8, (24), (38), 56, 57, 58, 79, 45, ∞

(2), (4), 8, 6, (16), (24), (38), 56, 57, 58, 79, 70, 45, ∞

2, 4, 6, 8, 16, 24, 38, 56, 57, 58, 79, 70, 45

2, 4, 6, 8, 16, 24, 38, 45, 57, 58, 79, 70, 56

2, 4, 6, 8, 16, 24, 38, 45, 56, 58, 79, 70, 57

2, 4, 6, 8, 16, 24, 38, 45, 56, 57, 79, 70, 58

2, 4, 6, 8, 16, 24, 38, 45, 56, 57, 58, 70, 79

08/04/2023

* Quick Sort Time Complexity Analysis:

$O_W(n)$ → worst case (n+1) comparisons are required

$O_W(n) \longrightarrow (n+1)+(n)+(n-1)+\cdots+1$

$= \frac{n(n+1)}{2}$

$\boxed{O_W(n) = O(n^2)}$

$O_A(n) = (n+1) + \frac{1}{n}\left[C_A(k-1) + C_A(n-k)\right]$

$\boxed{C_A(n) = (n+1) + \frac{1}{n}\left[C_A(k-1) + C_A(n-k)\right]} \quad -(1)$

$C_A(0) = C_A(1) = c$

Sub (n-1) in (1) in place of 'n'

$C_A(n-1) = n + \frac{1}{n-1}\left[C_A(k-1) + C_A(n-k-1)\right]$

put k = 1, 2, 3 ...

$C_A(n) = n+1 + \frac{1}{n}\left[C_A(0) + C_A(1) + \cdots + C_A(n)\right] \quad -(2)$

$C_A(n-1) = n + \frac{1}{n-1}\left[C_A(0) + C_A(1) + \cdots + C_A(n-1)\right] \quad -(3)$

(2) × n

(3) × (n-1)

$n\,C_A(n) = n(n+1) + \left[C_A(0) + C_A(1) + \cdots + C_A(n-1)\right] \quad -(5)$

$(n-1)C_A(n-1) = n(n-1) + \left[C_A(0) + C_A(1) + \cdots + C_A(n-2)\right] \quad -(6)$

(5) − (6)

$(n-1)C_A(n-1) + n\,C_A(n)$

$= n^2 + n - n^2 + n$

$+ 2C_A(n-1)$

$-(n-1)C_A(n-1) + n\,C_A(n) = +2n + 2C_A(n-1)$

$$nC_A(n) = 2n + C_A(n-1)(n-1+2)$$

$$nC_A(n) = 2n + C_A(n-1)(n+1)$$

$$C_A(n) = 2 + \frac{C_A(n-1)(n+1)}{n}$$

$$\boxed{\frac{C_A(n)}{n+1} = \frac{2}{n+1} + \frac{C_A(n-1)}{n}}$$

$$\frac{C_A(n)}{n+1} = \frac{2}{n+1} + \frac{2}{n} + \frac{2}{n-1} + \cdots + \frac{C_A(n)}{3}$$

$$= 2 \sum_{3 \le K \le n} \frac{1}{K}$$

$$\sum_{3 \le K \le n} \frac{1}{K} \le \int_{2}^{n+1} \frac{1}{x} dx$$

$$\frac{C_A(n)}{n+1} \le \log_e(n+1)$$

$$C_A(n) \le (n+1)\log_e(n+1)$$

$$C_A(n) \le n \log_e n$$

$$\boxed{C_A(n) = n \log n}$$

* STRASSEN'S MATRIX MULTIPLICATION

→ Multiplication is costlier compared to addition.

1) Small $n = 1$

A	B
a_{11}	b_{11}

* $n = 2$

$$A\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad B\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21}$$
$$c_{12} = a_{11} \times b_{12} + a_{12} \times b_{22}$$
$$c_{21} = a_{21} \times b_{11} + a_{22} \times b_{21}$$
$$c_{22} = a_{21} \times b_{12} + a_{22} \times b_{22}$$

} 8 products + 4 sum

* $n = 4$

$$A\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

A_{11} A_{12} / A_{21} A_{22}

B_{11} B_{12}

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

B_{21} B_{22}

$$C_{11} = MM(A_{11} \times B_{11}) + MM(A_{12} \times B_{21})$$

$$C_{12} = MM(A_{11} \times B_{12}) + MM(A_{12} \times B_{22})$$

$$C_{21} = MM(A_{21} \times B_{11}) + MM(A_{22} \times B_{21})$$

$$C_{22} = MM(A_{21} \times B_{12}) + MM(A_{22} \times B_{22})$$

→ for multiplication (1 ton, 1 top)

$$T(n) = \begin{cases} 8T(n/2) + n^2 & n > 2 \\ 1 & n \leq 2 \end{cases}$$

$$T(n) = 8T(n/2) + n^2$$

$$T(n) = n^2 = \theta(n^k \log^p n)$$

$p = 0 \quad k = 2$

$a = 8; \quad b = 2$

$$\log_2^{2^3} = 3$$

$$\log_b^a = 3 > k = 2$$

$$T(n) = \theta(n^{\log_b^a})$$

$$\boxed{T(n) = \theta(n^3)}$$

→ Strassen's Matrix:

* Strassen's reduced the no. of multiplications to 7

$$\therefore T(n) = \begin{cases} 7T(n/2) + n^2 & n > 2 \\ 1 & n \leq 2 \end{cases}$$

$$\log_2^7 = 2.8$$

$$\boxed{T(n) = \theta(n^{2.8})}$$

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22}) B_{11}$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12}) B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{22} + B_{21})$

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

* Write a recursive algorithm for Strassen's Matrix Multiplication using divide and conquer:

$C[i,j] := \{0\}$;

→ Algorithm SMM(n, A[i,j], B[i,j])

```
{ if (n<=2)
  {
    c[1,1] = a[1,1] * b[1,1]
             + a[1,2] * b[2,1];
    c[1,2] = a[1,1] * b[1,2]
             + a[1,2] * b[2,2];
    c[2,1] = a[2,1] * b[1,1] +
             a[2,2] * b[2,1];
    c[2,2] = a[2,1] * b[1,2]
             + a[2,2] * b[2,2];
  }
else
{ n := n/2
  c[1,1] := MM(n/2, A11, B11)
            + MM(n/2, A12, B21)
  c[1,2] = MM(n/2, A11, B12)
            + MM(n/2, A12, B22)
  c[2,1] = MM(n/2, A21 x B11) + MM(n/2, A22, B21)
  c[2,2] = MM(n/2, A21, B12) + MM(n/2, A22, B22)
}}
```

* KARATSUBU'S LARGE INTEGER MULTIPLICATION:

→ $x = 145236$.
$y = 789213$

* Multiplying x and y :

→ we can divide the nos into 2 equal parts i.e. $x \to a, b$; $y \to c, d$.

→ Then
$x = a \times 10^{n/2} + b$ $\quad n \to$ no. of digits
$y = c * 10^{n/2} + d$

$x * y = (a \times 10^{n/2} + b) * (c \times 10^{n/2} + d)$

$= ac \times 10^{2n/2} + ad \times 10^{n/2} + bc \times 10^{n/2} + bd$

$$\boxed{xy = ac \times 10^n + (ad + bc) 10^{n/2} + bd}$$

$xy = ac \times 10^n + ((a+b)(c+d) - ac - bd) 10^{n/2} + bd$

→ $x = 146123$ $\quad y = 352120$

KIM(146/123, 352/120)

ac | (a+b)(c+d) | bd

KIM(14/6, 35/2) $\quad$ 12

490 | 740

KIM(1|4, 3|5) $\quad$ 20 $\quad$ KIM(2|0, 3|7) $\quad$ KIM(6,2)

ac | (a+b)(c+d) | bd

(3) | 40 | 6 | KIM(6,10) | 0

KIM(1,3) | KIM(5,8) | KIM(4,5) | 0 | 2 | 0

$xy = ac \times 10^n + ((a+b)(c+d) - ac - bd) 10^{n/2} + bd$

$= 3 \times 10^2 + ((40) - 3 - 20) 10^1 + 20$

$= 300 + 170 + 20$

$= 490$

$(a+b)(c+d) = 0 + (2 - 0 - 0) 10 + 0$

$= 20$

$KIM(20, 37) = 6 \times 10^2 + (20 - 6 - 0) 10$

$= 600 + 140 + 0 = 740$

KIM(146, 352)

$= 490 \times 10^{3} + (740 - 490 - 12)$

$+ 12$

$= 490000 + (250 - 12) + 12$

$= 490000 + 2380 + 12$

$= 492380 + 12$

$= 492392$

10/04/23

GREEDY METHOD:

→ optimal sol^n is maximize the sol^n

stagewise/level-wise algorithms

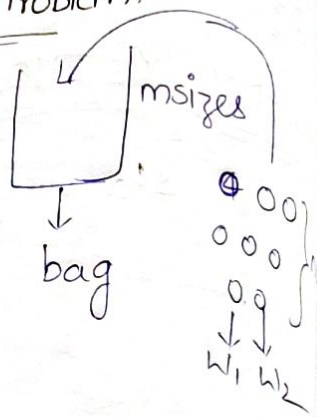
- finding subset
- ordering the subset

* → Feasible Sol^n: Any subset of given input sating the condition
* → Optimal sol^n: from feasible sol^n set there is one sol^n which is optimising the objective fn.

```
Algorithm Greedy (a,n)
 { solution:=0;
   for i:=1 to n do
     { x:=select(a);
       if feasible (solution,x) then
                 solution:=union(solution,x);
     }
 return solution ; }
```

→ Fractional KnapSack Problem:

$\sum_{i=1}^{n} w_i \le m$



maximum $\sum_{i=1}^{n} P_i w_i$

$1 \le i \le n$

to get max. profit

1) Greedy by profit
2) Greedy by weight
3) Greedy by P/W

→ 3 ways

Maximise $P_i w_i$ value by keeping in mind

$\sum_{i=1}^{n} w_i \le m$

$n=3$, $m=20$ $(P_1, P_2, P_3) = (25, 24, 15)$

$(W_1, W_2, W_3) = (18, 15, 10)$ Greedy by profit

$w_1 = 18 \le 20 = m$.

profit $= 1 \times 25 = 25$

wt of task $= 20 - 18 = 2$

$w_2 = 15 \le 2 \Rightarrow \frac{2}{15}$

profit $= 25 + \frac{2}{15} \times 24 = 28.2$

Wt of knapsack $= 0$

	C_1	C_2	C_3
x_i	1	2/15	0
Profit	25	$\frac{2}{15} \times 24$	13

Greedy by profit

Greedy by weight = Greedy by profit = 28.2

Greedy by P/W =

C_1 $\frac{P_1}{w_1} = \frac{25}{18} = 1.38$ — III

C_2 $\frac{P_2}{w_2} = \frac{24}{15} = 1.6$ — I

C_3 $\frac{P_3}{w_3} = \frac{15}{10} = 1.5$ — II

	O_2	O_3	O_1
sol^n	1	5/10	0
profit	24	$\frac{5}{10} \times 15$	0

profit = 24 + 7.5 = 31.5

31.5 → feasible sol^n

↓

optimal sol^n

* Algorithm GreedyKnapSack (m, n)

```
{ for i := 1 to n do
    x[i] := 0;
  u := m
for i := 1 to n do
{ if (w(i) > 0) then break;
  x[i] := 1; u := u - w[i]; }
if (i ≤ n) then x[i] := u/w[i]; }
```

O(n) → sorted order

O(n log n) → unsorted

overall → O(n log n)

* Consider a knapsack of capacity 60kgs & find optimal sol^n for objects given below

obj	1	2	3	4	5
wt	5	10	15	22	25
profit	30	40	45	77	90

Greedy by profit

m = 60

5 4 3 2 1

$w_5 = 25 < 60$

p = 90

m = 60 − 25 = 35

15/04/23

→ Find optimal sol^n for knapsack problem given n = 7; m = 15 $(P_1, P_2, \ldots P_7) = (10, 5, 15, 7, 6, 18, 3)$ and $(w_1, w_2 \ldots w_7) = (2, 3, 5, 7, 1, 4, 1)$.

$\frac{P_1}{w_1} = 5$, $\frac{P_2}{w_2} = 1.6$, $\frac{P_3}{w_3} = 3$, $\frac{P_4}{w_4} = 1$

$\frac{P_5}{w_5} = 2.6$; $\frac{P_6}{w_6} = 4.5$; $\frac{P_7}{w_7} = 3$

→ Maximize: $\sum_{i=1}^{n} p_i x_i$

$m = 15$

$\sum_{i=1}^{n} w_i x_i \leq m$

* subset which optimizes the object function

1, 1.6, 3, 3, 4.5, 5, 6

$\{O_5, O_1, O_6, O_3, O_7, O_2, O_4\}$

* $O_5 - W_5 \leq m$

$1 \leq 15 \Rightarrow T \quad : W_5 = 1 = x_5.$

RCK = 15 − 1 = 14

Profit = 0 + 6 × 1 = 6

$O_1 - W_1 \leq m$

5 − 2 ≤ 14

3 ≤ 14 $\quad x_1 = 1$

RCK = 14 − 2 = 12

Profit = 6 + 10 × 1 = 16

$O_6 - W_6 \leq m$

4.5 − 4 ≤ 12

RCK = 12 − 4 = 8 $\quad x_6 = 1$

Profit = 16 + 15 × 1 = 34

$O_3 - W_3 \leq m$

3 − 5 ≤ 8

RCK = 8 − 5 = 3 $\quad x_3 = 1$

Profit = 34 + 15 × 1 = 49

$O_7 - W_7 \leq m$

3 − 1 ≤ 3

2 ≤ 3 $\quad x_7 = 1$

RCK = 3 − 1 = 2

Profit = 49 + 3 = 50

$O_2 \; W_2 \leq m$

1.6 − 3 ≤ 2

RCK = 2 − 3 = −1 ✓

RCK = −

Profit = $50 + 5 \times \frac{1.66}{3} = 50 + 2.32$

$= 52.32$

* Optimal Storage on tapes

↳ application of Greedy method.

↳ find the retrieval time for accessing programs that are stored on tape.

* → tapes are used for sequential retrieval hence we must store the code (or) data in a proper sequence so as to minimize the mean retrieval time of the data.

* Programs : $i_1, i_2 \ldots i_n$ tape : L

Length : $l_1, l_2 \ldots l_n$

t_j = access time for the j^{th} program

$$t_j = \sum_{1 \le k \le j} l_{i_k}$$

* Mean Retrieval Time (MRT)

$$\sum_{i=1}^{n} l_i \le L$$ → condition for feasible sol^n

* $n = 3$; $(l_1, l_2, l_3) = (5, 10, 3)$ $n! = 6$.

$1, 2, 3$

$(1,2,3) = (5) + (5+10) + (5+10+3) =$

$= 5 + 15 + 18 = 38$

$(2,1,3) = 10 + 15 + 18 = 43$

$(3,1,2) = 3 + 8 + 18 = 29$

$(3,2,1) = 3 + 13 + 18 = 34$

$(2,3,1) = 10 + 13 + 18 = 41$

$(1,3,2) = 5 + 8 + 18 = 31$

$(3,1,2)$ → minimum retrieval time $= 29$

L, if we give the ascending order of length of the programs.

→ If there are no. of tapes; the programs are divided among different tapes and find MRT for each tape.

* Algorithm for multiple tapes

```
procedure STORE(n,m)
Integer m,n,j
  j ← 0
  for i ← 1 to n do
    print('append program', i,' to
    j ← (j+1) mod m | permutation for tape
  repeat
end STORE
```

* Find the optimal placement for 13 programs on 3 tapes T_0, T_1, T_2 where the program lengths are (12, 5, 8, 32, 7, 5, 18, 26, 4, 3, 11, 10, 6)

$n = 13$; $m = 3$

T_0	T_1	T_2
12	5	8
32	7	5
18	26	4
3	11	10
6		

~~MRT = 12+44+62+65+71~~

$T_0 \rightarrow$ 3, 6, 12, 18, 32

$$MRT_0 = \frac{1}{5}[3+9+21+39+71]$$

$$= \frac{1}{5}[143]$$

$$= 28.6$$

1
71
39
12
21
143

$T_1 = 5, 7, 11, 26$

$$MRT_1 = \frac{1}{4}[5+12+23+49]$$

$$= \frac{89}{4} = 22.25$$

1
49
23
17
89

$T_2 = 4, 5, 8, 10$

$$= \frac{1}{4}[4+9+17+27]$$

$$= \frac{57}{4} = 14.25$$

27
17
13
57

1
28.6
22.25
14.25
65.10

$$m(MRT) = \frac{65.10}{3}$$

$$= 21.7$$

17/09/23

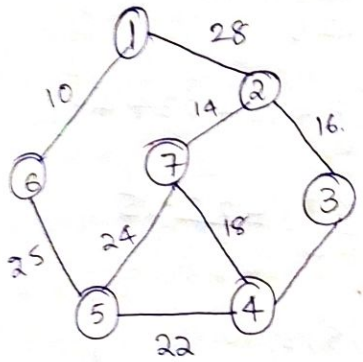
* Minimum Spanning Trees

$G = (V, E)$

$t = (V, E')$

→ Prims

→ Kruskals.



Algorithm Prims (E, cost, n, t)

{ Let (k, l) be minimum cost edge

t[1,1] = k, t[1,2] = l;

m.c = cost[k, l] ; near[k] = 0;

near[l] = 0;

for i := 1 to n do

{ if (cost[i, l] < cost[i, k]) then

near[i] = l;

else

near[i] = k; }

for i := 2 to n-1 do

{ let j be an index such that

cost[j, near[j]] is minimum &

near[j] ≠ 0

t[i,1] = j ; t[i,2] = near[j];

mc = mc + cost[j, near[j]]

near[j] = 0.

~~for k := 1 to n do~~

~~if (~~

for k := 1 to n do

{ if (near[k] ≠ 0 && (cost[k, near[k]]

> cost[k, j])

then near[k] = j }}

return mc; }

→ Time Complexity : $O(n^2)$ (generally)

20/04/2023 → Time Complexity |E| log |E|

* Kruskal's algorithm:

Algorithm Kruskal (E, cost, n, t)

{ construct a heap out of Edge cost using heapify } Edge

for i := 1 to n do parent[i] = -1 }

i = 0; mc = 0

while ((i < n-1) and (heap not empty))

do

{ Delete a min cost edge (u, v) from the heap and reheapify using Adjust } Ele

j := find(u) ; k := find(v)

if (j ≠ k) then

{ i := i + 1;

t[i, 1] := u ; t[i, 2] := v

mc = mc + cost[u, v];

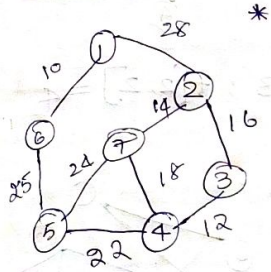
union(j, k); } }

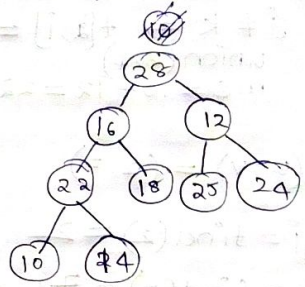
if (i ≠ n-1) then

{ write ("No spanning tree"); }

else return mc; }

* find() → gives the root of that tree.





Initially all the vertices are selected as individual nodes.

Then all its parent nodes are initialized to -1. Then we construct a min/max heap trees; so as to construct min cost spanning tree.

* ① ② ③ ④ ⑤ ⑥ ⑦

1) (u, v) = (1, 6) = (u, v)

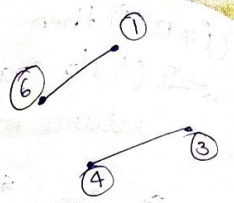
j = find(1) = 1

k = find(6) = 6

j ≠ k t[1, 1] = 1 ; t[1, 2] = 6

union(1, 6)

mc = 0+10 = 10



2) (3,4) = (u,v)

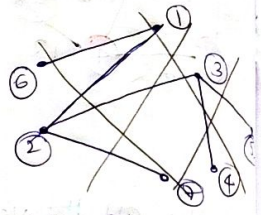
j = find(3) = 3

k = find(4) = 4

j ≠ k ⇒ t[2,1] = 3 ; t[2,2] = 4

union(3,4)

mc = 10+12 = 22.



3) (u,v) = (2,7)

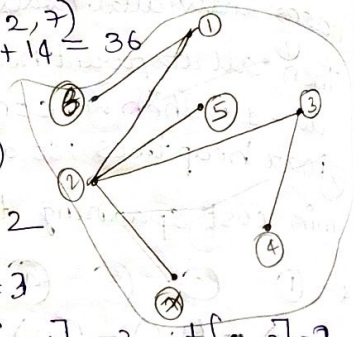
j = find(2) = 2

k = find(7) = 7.

j ≠ k ⇒ t[3,1] = 2 ; t[3,2] = 7

union(2,7)

mc = 22+14 = 36



4) (u,v) = (2,3)

j = find(2) = 2

k = find(3) = 3

j ≠ k ⇒ t[4,1] = 2 ; t[4,2] = 3

union(2,3)

mc = 36+16 = 52

5) (u,v) = (4,7)

j = find(4) = 2

k = find(7) = 2

~~j ≠ k~~ ⇒ ~~t[5,1] = 3 ; t[5,2] = 2~~

~~j = k~~ union (2,3)

mc = 52+18 = 70

6) (u,v) = (4,5)

j = find(4) = 2

k = find(5) = 5

j ≠ k ; t[6,1] = 2 ; t[6,2] = 5.

union(2,5)

mc = ~~[illegible]~~ 52+22 = 74

7) (u,v) = (5,7)

j = find(5) = 2

k = find(7) = 2 j = k ; ×

~~j ≠ k ; t[7,1] = 5~~

8) (u,v) = (5,6) ⟹ find(5) = 2 union(2,1)

find(6) = 1

mc = ~~74~~ +25 = 99

20/04/2023

* Optimal merge patterns: *

total no. of record moves $= \sum_{i=1}^{n} d \times l$

d → distance from root
l → length of the file

```
tree node = record.
{ treenode *lchild, *rchild; int weight
};

Algorithm Tree (n)
{ for i:=1 to n-1 do
    { pt:= new treenode;
      (pt→ lchild) = Least( list)
      (pt→rchild)= Least( list)
      (pt→weight) := ((pt→lchild)→wei
                    + ((pt→rchild)→wei
      insert(list, pt); }
    return Least (list); }
```

* Total no. of record moves for files given as 23, 15, 65, 72, 15, 35

28/04/2023

*Single Source Shortest Path: Time complexity: $O(n^2)$

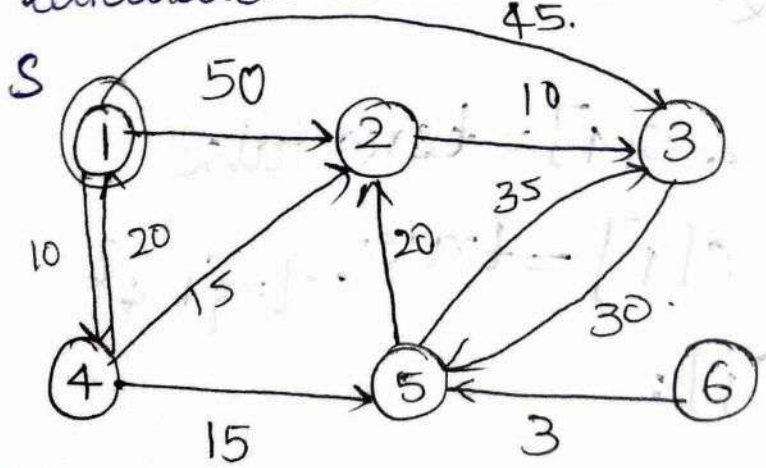
$G = (V, E)$

$S \rightarrow$ source

We must find the shortest path between (S and V-S).

*Graph is represented as cost adjacency matrix.

*We must add every shortest distance calculated to the solution set.



V → set of vertices = (1, 2, 3, 4, 5, 6)

→ All the edge weights should be non-negative

Cost =

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						

for i := 1 to n

dist[i] = cost[1, i]

dist[1] = 0

dist[2] = 50

dist[3] = 45

dist[4] = 10

dist[5] = ∞

dist[6] = ∞

s[1] = true;

sol = {1}

dist[4] min & s[4] = ~~true~~ false

u = 4 ; s[4] = true. sol = {1, 4}

w = adjacent[4];

4 - 1, 5 → s[5] = false.

↓

s[1] = true

dist[5] ⇒ dist[u] + cost[4, 5]

dist[5] = 25.

u = 5

s[5] = true. ; sol = {1, 4, 5}

w = adjacent[5];

5 — 2, 3 s[2] = false

s[3] = false

dist[2] > dist[5] + cost[5, 2]

> 25 + 20

dist[2] = 45

dist[3] > dist[5] + cost[5, 3]

25 + 35 ×

60

dist[3] = 45.

sol = {1, 4, 5, 2} s[2] = true.

u = 2

w = adjacent[2]

2 — 3.

s[3] = false

dist[3] > dist[2] + cost[2, 3]

> 50 + 10

60

dist[3] = 45

s[3] = true

sol = {1, 4, 5, 2, 3}

u = 3

adj = 5.

The shortest path solution set = {1, 4, 5, 2, 3, 6}

There is no path from 1 to 6.

* Algorithm:

source ↑ ε → edges

Algorithm SP(v, cost, dist, n)

{ for i := 1 to n do

O(n) { s[i] = false; dist[i] = cost[v, i]; }

s[v] = true;

for num := 2 to n do

{ choose u from (n-1) vertices such that dist[u] is minimum. O(n²)

s[u] = true.

O(nE) for (each w adjacent to u with s[w] = false)

{ if (dist[w] > dist[u] + cost[u, w])

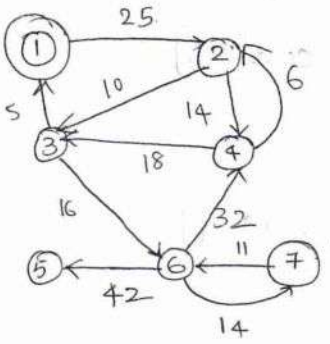
{ dist[w] = dist[u] + cost[u, w]; } } }

If no. of edges > no. of vertices

mostly $T(n) = O(n|E|)$

* If we are using efficient datastructures : $T(n) = O(n \log E)$

*



① → source.

for i := 1 to n.

dist[1] = 0

dist[2] = 25

dist[3] = ∞

dist[4] = ∞

dist[5] = ∞

dist[6] = ∞

dist[7] = ∞

sol = {1}.

u=2 s[2]=false

dist[2] is min & s[2]= false

u=2 s[2]= true

sol={1,2}

w = adjacent[2]

2 - 4, 3 → s[3] false

↓

s[4] false

dist[3] > dist[2] + cost[2,3]

> 25 + 10

dist[3] = 35

dist[4] > dist[2] + cost[2,4]

> 25 + 14

dist[4] = 39

sol = {1,2,3}

u = 3 s[3]=false

w = adjacent[3]

3 →

29/04/2023

NETWORK ANALYSIS PROBLEM:

FORD-FULKERSON ALGORITHM:

* G=(V, E) → each edge is having a capacity i.e. no. of items that can flow from one node to another

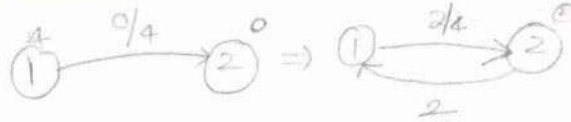
Set: V → source, sink.

Maximum flow of items that can flow from source to sink. from the given network.

** Capacity: Max. flow of items from one node to another.

** Flow: current flow from one node to another.

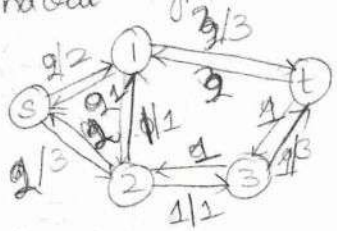
** Residual Capacity: Capacity - Flow



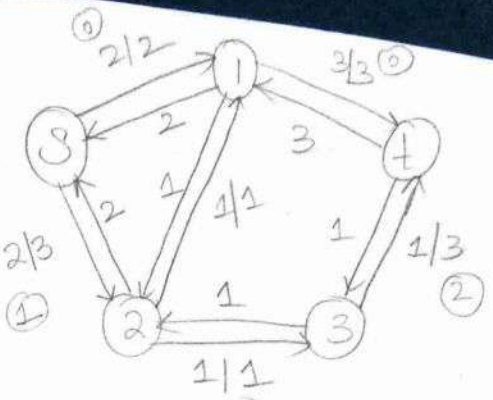
* Augmented Path: Path from source to sink such that; each node included in the path has a residual capacity >0, then such path is called augmented path

* Bottle Neck Capacity: minimum residual capacity to allow flow of data.

Edmond Karp has given that we can use BFS to find out Augmented Path.

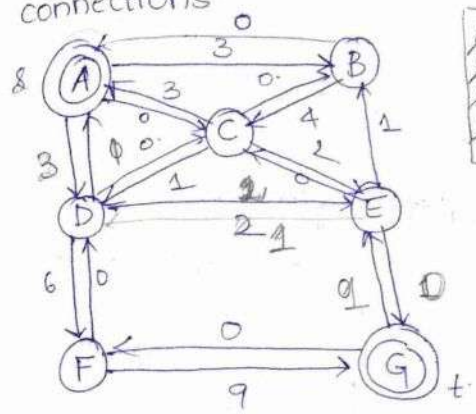


Flow	Augmented Path	Bottle Neck Capacity
2	$S \xrightarrow{2} 1 \xrightarrow{3} t$	2
1	$S \xrightarrow{3} 2 \xrightarrow{1} \overset{3}{3} \rightarrow t$	1
1	$S \xrightarrow{2} 2 \xrightarrow{1} 1 \xrightarrow{1} t$	1
4		



Maximum flow from source to sink = 4

→ Applications: water pipelines; current connections.



Queue

Parent map

B → A
D → A
C → B
E → D
F → D
G → E

Visited vertex

A
B
D
C
E
F
G

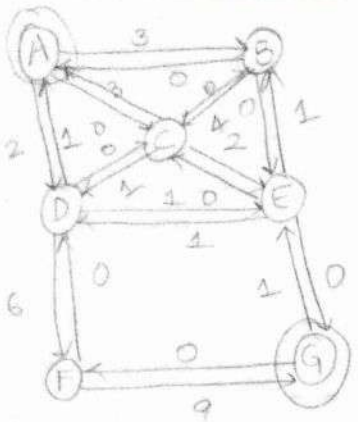
current = A B D C E

Flow = 0

max_flow = 0 + 1

Augmented path

$A \xrightarrow{3} D \xrightarrow{2} E \xrightarrow[1]{} G$



Queue

A
~~B~~
~~D~~
~~C~~
~~E~~
~~F~~
G

Parent Map

B → A
D → A
~~E~~ → ~~B~~
E → D
F → D
G → F

Visited

A
B
D
C
E
F

current = ~~A~~ ~~B~~ ~~D~~ ~~C~~ ~~E~~ F

A $\xrightarrow{2}$ D $\xrightarrow{6}$ F $\xrightarrow{9}$ G

(2)

max flow = 1 + 2 = 3

* Start with initial flow as zero.
* While there exist an augmented path from source to sink.
* Find an augmented path by using either BFS (or) DFS
* Determine the bottle neck flow for augmented path found.
* Increase flow with bottle neck capacity of augmented path found in previous step.

*Return the max. flow.

→ Time Complexity = K*(Time for DFS/BFS)

$= kO(E+V)$

K → no. of augmented paths.

Dynamic Programming

→ We can apply Dynamic Programming, if we get solution by applying sequence of steps.

→ It is also known as 0/1 knapsack problem.

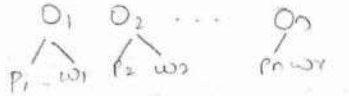
→ If objects are kept in knapsack,

$x_i = 1$, } $x_i \in [0,1]$
else $x_i = 0$.

If previous capacity is m1
If object is placed then with profit p1, weight w1,

then capacity becomes m1 - w1.

and $p = p + p_1$



=> profit should be max.

Principle of optimility: Taking the opitimal decision which means that, all the other decisions are less than the optimal the initial decision is known as principle of optimility.

→ So, sequence of these decisions gives us globally best decision.

$$\max\left(f_{n-1}(j),\ P_n + f_{n-1}(j-w_n)\right)$$

compare and replace with maximum

Multi-Stage Graph:

$G=(V, E)$

$V = \{V_1, V_2 \cdots, V_k\}$ → disjoint vertices

V_1 → saree

V_k → sink (or) destination.

$E \in G_1,\ E \in V_i \rightarrow V_{i+1}$ → only next vertex.

→ Complexity: $O(|V|+|E|)$

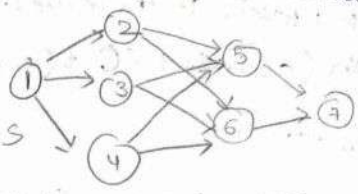
→ This type of graph is also known as k-stage graph.

→ We should find shortest path from source to sink.

→ 2 methods to find shortest path.

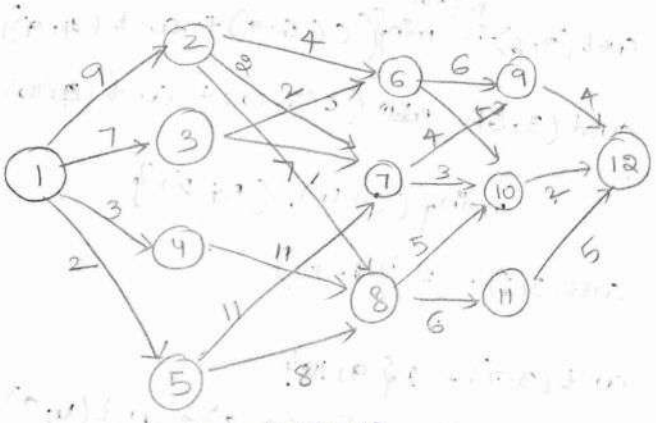
1) Forward method

2) Backward method.



→ we select i^{th} vertex, by observing the cost of the $(i+1)^{th}$ stage → forward

Backward: we will take i^{th} edge by considering $(i-1)^{th}$ stage cost



$c(2,6)$ + Cost$(6,t)$

Forwards:

$cost(i,j) = \min \{ c(j,l) + cost(i+1,l) \}$ $l \in V_{i+1}$

↕ ↓

stage vertex

$\langle j, l \rangle \in E$

we start from k-1 level, if there are k levels

$cost(4,9) = \min (c(9,12) + cost(5,12))$

$= 4+0 = 4.$

$cost(4,10) = \min (c(10,12) + cost(5,12))$

$= 2+0 = 2$

$cost(4,11) = \min (c(11,12) + cost(5,12))$

$= 5+0 = 5$

$cost(3,6) = \min_{l=\{9,10\}} \{ c(6,9) + cost(4,9) \}$

$cost(3,6) = \min \{ c(6,10) + cost(4,10) \}$

$\min \{ (6+4), (5+2) \}$

$cost(3,6) = 7 \quad (l=10)$

$cost(3,7) = l\{9,10\}$

$= \min \{ c(7,9) + cost(4,9),$

$c(7,10) + cost(4,10) \}$

$= \{ (4+4), (3+2) \}$

$= \{ 8, 5 \}$

$l = 10$ and minimum cost = 5.

$cost(3,8) = \min_{l=\{10,11\}} \{ c(8,10) + cost(4,10)$

$c(8,11) + cost(4,11) \}$

$= (5+2, 6+5)$

$= (7, 11)$

$l = 10$, minimum cost = 7.

$cost(2,2) = \min_{l=\{6,7,8\}} \{ c(2,6) + cost(3,6),$

$c(2,7) + cost(3,7),$

$c(2,8) + cost(3,8) \}$

$= \min \{ 4+7, 2+5, 1+7 \}$

$= \min \{ 11, 7, 8 \}.$

$cost(2,2) = 7, \quad l = 7$

$cost(1,1) = \min_{l=\{6,7\}} \{ c(1,6) + cost(3,6),$

$c(1,7) + cost(3,7) \}$

$= \min (2+7, 4+5) = 9$

$cost(1,1) = 9, \quad l = 6.$

$l = 8$

$cost(4,8) = \min\{ c(4,8) + cost(8,8) \}$

$= 11 + 7 = 18.$

$l = 7,8$

$cost(2,5) = \min\{ c(5,7) + cost(8,7),$

$c(5,8) + cost(3,8) \}$

$= \min\{ (11+5), (8+7) \}$

$= \min(16, 15)$

$= 15, \quad l = 8.$

$l = \{2,3,4,5\}$

$cost(1,1) = \min\{ c(1,2) + cost(2,2),$

$c(1,3) + cost(2,3),$

$c(1,4) + cost(2,4),$

$c(1,5) + cost(2,5) \}$

$= \min\{ (9+7), (7+9), (3+18),$

$(2+15) \}$

$= \min\{ 16, 16, 21, 17 \}$

$= 16, \quad l = 2.$

path :

$1 \to 2 \to 7 \to 10 \to 12$

Backward cost.

$bcost(i,j) = \min_{l \in V_{i-1}, \langle l,j \rangle \in E} \{ bcost(i-1,l) + c(l,j) \}$

(stage, vertex)

$bcost(2,2) = \min_{l=\{1\}} \{ bcost(1,1) + c(1,2) \}$

$= 0 + 9 = 9.$

$bcost(2,3) = \min_{l=1} \{ bcost(1,1) + c(1,3) \}$

$= 0 + 7 = 7.$

$bcost(2,4) = \min_{l=1} \{ bcost(1,1) + c(1,4) \}$

$= 0 + 3 = 3.$

$bcost(2,5) = \min_{l=1} \{ bcost(1,1) + c(1,5) \}$

$= 0 + 2 = 2$

$l = \{2,3\}$

$bcost(3,6) = \min\{ bcost(2,2) + c(9,6),$

$bcost(2,3) + c(3,6) \}$

$= \min\{ (9+4), (7+2) \}$

$= 9 //. \quad l = 3.$

bcost(3,7) = $l = \{2,3,5\}$

$= \min\{ bc(2,2) + cost(2,7),$
$bc(3,3) + cost(3,7),$
$bc(5,5) + cost(5,7)\}$

$= \min\{(9+2), (7+4), (2+11)\}$

$= \min(11, 14, 13)$

$= 11, \quad l = 2.$

bcost(3,8) = $l = \{2,4,5\}$

$= \min\{ bc(2,2) + cost(2,8),$
$bc(2,4) + cost(4,8),$
$bc(2,5) + cost(5,8)\}$

$= \min\{(9+1), (3+11), (2+8)\}$

$= \min\{10, 14, 10\}$

$l = 2, \quad min = 10.$

bcost(4,9) = $l = \{6,7\}$

$= \min\{ bc(3,6) + cost(6,9)$
$bc(3,7) + cost(7,9)\}$

$= \min((9+6), (11+4))$

$= 15, \quad l = 6.$

bcost(4,10) = $l = \{6,7,8\}$

$= \min\{ bc(3,6) + cost(6,10)$
$bc(3,7) + cost(7,10)$
$bc(3,8) + cost(8,10)$

$= \min\{(9+5), (11+3), (10+5)\}$

$= \min\{14, 14, 15\}$

$l = 6. \quad = 14$

bcost(4,11) = $l = \{8\}$

$= \min\{ bc(3,8) + cost(8,11)\}$

$= 10 + 6 = 16$ // $l = 8$

bcost(5,12) = $l = \{9,10,11\}$

$= \min\{ bc(4,9) + cost(9,12),$
$bc(4,10) + cost(10,12),$
$bc(4,11) + cost(11,12)\}$

$= \min(15+4, 14+2, 16+5)$

$= \min(19, 16, 21)$

$= 16, \quad l = 10$

shortest path : $1 \to 3 \to 6 \to 10 \to 12$

All pairs shortest path problem: / Bellmen-ford

$G = (V, E)$

→ We take directed graph.

→ All the vertices are sources, sinks.

Suppose i, j are vertices, $c(i, j)$

$i \neq j$ and edge should be present

$(i, j) \in E$

→ If edge is present → $c(i, j)$

If edge is not present → ∞

$i = j \rightarrow 0$

→ If there is an intermediate vertex between i, j, let it be k.

the indirect path $c(i, j) = c(i, k) + c(k, j)$

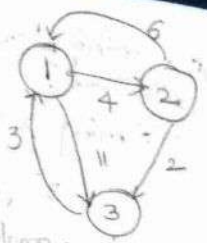
$$A^k(i,j) = \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$$

where $k \geq 1$

A^0 → graph without taking any intermediate

A^1 → with intermediate.

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



If we take i as intermediate, then corresponding row & column will not change.

A^1 → taking '1' as intermediate.

$cost(2,1) = \min\{c(2,1), c(2,1)\}$

= no change i.e 6.

$cost(2,3) = \min\{c(2,3), c(2,1) + c(1,3)\}$ (k=1)

$= \min\{2, 6+11\}$

$= 2.$

$cost(3,2) = \min\{c(3,2), c(3,1) + c(1,2)\}$

$= \min\{\infty, 3+4\}$

$= \min(\infty, 7)$

$cost(3,2) = 7.$

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

A^2 → taking 2 as intermediate vertex.

$cost(1,3) = \min\{A^1(1,3), A^1(1,2) + A^1(2,3)\}$ (k=2)

$= \min\{11, 4+2\} = 6$

$A^2(3,1) = \min\{A^1(3,1), A^1(3,2) + A^1(2,1)\}$

$= \min\{3, 7+6\}$

$= 3$

$$A_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

consider $K = 3$

$A^3(1,2) = \min_{K=3}\{A^2(1,2), A^2(1,3) + A^2(3,2)\}$

$= \min\{4, 6+4\}$

$A^3(1,2) = 4$

$A^3(2,1) = \min_{K=3}\{A^2(2,1), A^2(2,3) + A^2(3,1)\}$

$= \min\{6, 2+3\}$

$= 5.$

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

Algorithm Allpaths (cost, A, n)

```
{ for i := 1 to n do
  for j := 1 to n do
    A[i,j] := cost[i,j]
  for k := 1 to n do
  for i := 1 to n do
  for j := 1 to n do
    A[i,j] = min( A[i,j], A[i,k] + A[k,j] )
}
```

complexity $= O(n^3)$

0/1 knapsack Problem

$n \rightarrow$ no. of objects. $m \rightarrow$ capacity of knapsack.

maximize $\sum P_i x_i$ = subjected + $\sum w_i x_i \leq m$

where $x_i \in [0,1]$; $1 \leq i \leq n$.

$f_n(m) \rightarrow$ profit we get

$f_n(m) = \max\{f_{n-1}(m), f_{n-1}(m-w_n) + P_n\}$

$f_i(y) = \max\{f_{i-1}(y), f_{n-1}(m-y_i) + P_i\}$

monotonically increasing. $\rightarrow f(y_1) < f(y_2)$ ----

$\rightarrow$ profit of i object (should be less than 2nd one.

Purging rule:
(or)
rule of dominance.

- If we have 2 tuples (P_1, w_1), (P_2, w_2) in the soln, then if any tuple have the following rule, then; we discard that tuple.
  $(P_2 < P_1, w_2 > w_1)$

ex: $n=3$; $(w_1, w_2, w_3) = (2, 3, 4)$
$n=6$, $(P_1, P_2, P_3) = (1, 2, 5)$

$S^0 = \{(0,0)\}$

$S_1^0 = \{(1,2)\}$

$S^1 = S^0 \cup S_1^0$

$= \{(0,0), (1,2)\}$

$S_1^1 = \{(2,3), (3,5)\}$

$S^2 = S^1 \cup S_1^1 = \{(0,0), (1,2), (2,3), (3,5)\}$

$S_1^2 = \{(5,4), (8,9), (7,7), (6,6)\}$

× weigh > capacity (6).

$S_i = \{(P_i, w_i)\}$

$S_i^i = \{(P_{i+1}, w_{i+1})\}$

$S^{i+1} = S^i \cup S_1^i$

$S^3 = S^2 \cup S_1^2 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6)\}$

purging rule

→ Take maximum profit $(6,6) \in S^3$

we consider 3rd object $\notin S^2$

$(x_3 = 1)$

⇒ $(6-4)(-1,2) \in S^2$

$\in S^1$ ⇒ $x_2 = 0$

$(1,2) \in S^1$

$\notin S^0$ ⇒ $x_1 = 1$.

$S^3 = S^2 \cup S_1^2 = \{(0,0)(1,2)(2,3)(3,5),(5,4)$
$(6,6)\}$

purging rule

→ Take maximum profit $(6,6) \in S^3$

$\notin S^2$

we consider 3rd object.

$(x_3 = 1)$

$\Rightarrow (6-4)(-1,2) \in S^2 \Rightarrow x_2 = 0$
$\in S^1$

$(1,2) \in S^1$
$\notin S^0 \Rightarrow x_1 = 1.$

12/05/2023

* MATRIX CHAIN MULTIPLICATION:

↳ Minimum no. of multiplications should be done.

Eg: A_1 A_2 A_3 A_4
2×4 4×3 3×4 4×2

$((A_1 \times A_2) A_3) A_4$

$2\times3\times4$

2×3

$2\times4\times3$ 2×4

2×2

$2\times2\times4$

total multiplications
$= 24+24+16$
$= 64$

$((A_1 \times A_2)\ (A_3 \times A_4))$

$2\times3\times4 \quad 3\times2\times4$

total multiplications $= 24+24+12 = 60$

24 24

$2\times3 \quad 3\times2$

$2\times3\times2 = 12$

C

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

cost[1,2] → j−i=1 → no. of combinations of k

cost[1,3]

cost[1,4] j−i=2

j−i=3 split number

S

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

① $j-i=1$

$C[1,2] = 2\times3\times4 = 24 \quad (k=1)$

$C[2,3] = 4\times4\times3 = 48 \quad (k=2)$

$C[3,4] = 3\times2\times4 = 24 \quad (k=3)$

C

	1	2	3	4
1	0	24	48	60
2		0	48	48
3			0	24
4				0

S

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

$C[i,j] = \min_{i\le k<j} \{C[i,k] + C[k+1,j] + d_{i-1} \cdot d_k \cdot d_j\}$

d = dimension

② $j-i=2$

$C[1,3] = \min \{C[1,1] + C[2,3] + d_0 d_1 d_3 \quad (k=1)$

$, C[1,2] + C[3,3] + d_0 d_2 d_3\} \quad (k=2)$

$1 \le k \le 3$

$A_1 \quad A_2 \quad A_3$

$2\times4 \quad 4\times3 \quad 3\times4$

$d_0 d_1 \quad d_1 d_2 \quad d_2 d_3$

$= \min \{0 + 48 + 2\times4\times4,$

$24 + 0 + 2\times3\times4\}$

$= \min\{80, 48\}$

$\therefore k = 2$

$A_2 \quad A_3 \quad A_4$

$4\times3 \quad 3\times4 \quad 4\times2$

$d_1 \quad d_2 \quad d_3 \quad d_4$

$C[2,4] = \min\{C[2,2] + C[3,4] + d_1 d_2 d_4 \quad (k=2)$

$, C[2,3] + C[4,4] + d_1 d_3 d_4\} \quad (k=3)$

$= \min\{0 + 24 + 4\times3\times2,$

$48 + 0 + 4\times4\times2\}$

$= \min\{48, 80\}$

$k = 2$

$j-i=3 \quad k=1,2,3$

$c[1,4] = \min\{c[1,1]+c[2,4]+d_0 d_1 d_4,$

$c[1,2]+c[3,4]+d_0 d_2 d_4,$

$c[1,3]+c[4,4]+d_0 d_3 d_4\}$

A_0	A_1	A_2	A_3	A_4
2×4	4×3	3×4	4×2	
d_0 d_1	d_1 d_2	d_2 d_3	d_3 d_4	

$= \min\{0+48+2\times4\times2,\ 24+24+2\times3\times2,$

$48+0+2\times4\times2\}$

$= \min\{64, 60, 64\}$

$k=2$

order: $((A_1 A_2)(A_3 A_4))$

A_1	A_2	A_3	A_4
5×4	4×6	6×2	2×7

$j-i=1 \quad 5\times6 \quad 5\times2 \quad 5\times7$

$c[1,2] = 5\times6\times4 = 120 \quad (K=1)$

$K=1$

$c[2,3] = 4\times2\times6 = 48 \quad (K=2)$

$K=2$

C	1	2	3	4	5	6	7
1	0	120	88				
2		0	48	104			
3			0	84			
4				0			
5					0		

$c[3,4] = 6\times7\times2 \quad (K=3)$

$= 84$

S	1	2	3	4	5	6	7
1	0	1	1				
2		0	2	3			
3			0	3			
4				0			
5					0		

$j-i=2$

$c[1,3] = \{K=1,2\}$

A_1	A_2	A_3
5×4	4×6	6×2
d_0 d_1	d_1 d_2	d_2 d_3

$= \min\{c[1,1]+c[2,3]+d_0 d_1 d_3,$

$c[1,2]+c[3,3]+d_0 d_2 d_3\}$

$= \min\{0+48+5\times4\times2,$

$120+0+5\times6\times2\}$

$= \min\{88, 180\} \quad (K=1)$

48
40

120
6

$c[2,4] = \{K=2,3\}$

A_2	A_3	A_4
$d_1 d_2$	$d_2 d_3$	$d_3 d_4$
4×6	6×2	2×7

$= \min\{c[2,2]+c[3,4]+d_1 d_2 d_4,$

$c[2,3]+c[4,4]+d_1 d_3 d_4\}$

42
14
168
84
84
52

$= \min\{0+84+4\times6\times7,\ 48+0+4\times2\times7\}$

48
56
104

$= \min\{252, 104\}$

$c[1,4] = \{K=1,2,3\}$

* OPTIMAL, BINARY SEARCH TREE:

22/05/2023

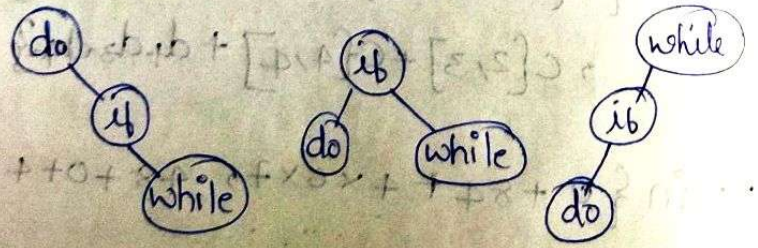
→ Construct a binary tree; where the cost of binary tree is minimum.

* cost of binary tree = cost of left subtree + cost of right subtree.

* This optimal binary search tree is used to efficiently differentiate b/w keywords and identifiers in the compiler.

*→ If a set of keywords are given; we must find out optimal BST whose cost is minimum.

*



i → node

* cost of successful searches = for internal nodes. Probability(i) * Level at which it is present

= p(i) * level(Ei)

cost of unsuccessful searches = qi * Level(Ei − 1)

↓ Probability of unsuccessful searches

(i−1)th level

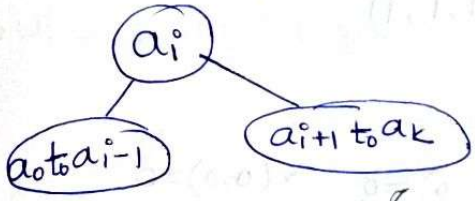
total cost = $\sum_{i=1}^{n} p_i$ * level(Ei) + $\sum_{i=0}^{n} q_i$ * (level(Ei − 1))

Ex: $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{4}$, $p_3 = \frac{1}{4}$

$q_0 - q_3 = \frac{1}{8}$

* Given set of identifiers $a_1 < a_2 < \cdots < a_k$

→ Let a_i ⇒ optimal identifier for root



cost(T) = Pi + cost(left) + cost(right)

↑ const (cannot be changed)

↓ ↓

should be min

• $cost(i,j) = \min_{i \le k \le j} \{ c(i,k-1) + c(k,j) \} + w(i,j)$

• $w(i,j) = p(j) + q(j) + w(i,j-1)$

• $w(i,i) = q(i)$, $c(i,i) = 0$,

$r(i,i) = 0$

p = probability of successful search

q = probability of unsuccessful search

* $n = 4$; (a_1, a_2, a_3, a_4)

$(p_1 \cdots p_4) = (3,3,1,1)$

$(q_0 \cdots q_4) = (2,3,1,1,1)$

* $j - i = 0$

$w(0,0) = q_0 = 2$; $c(0,0) = 0$; $r(0,0) = 0$

$w(1,1) = q_1 = 3$; $c(1,1) = 0$; $r(1,1) = 0$

$w(2,2) = q_2 = 1$; $c(2,2) = 0$; $r(2,2) = 0$

$w(3,3) = q_3 = 1$; $c(3,3) = 0$; $r(3,3) = 0$

$w(4,4) = q_4 = 1$; $c(4,4) = 0$; $r(4,4) = 0$

* $j - i = 1$

$\rightarrow w(0,1) = p(1) + q(1) + w(0,0)$

$= 3 + 3 + 2 = 8$

$r(0,1) = k(1) = 1$

$k = 1$

$c(0,1) = \{ c(0,0) + c(1,1) + w(0,1)$

$c(0,1) = 0 + 0 + 8 = 8$

$c(0,1) = 8$

$\rightarrow w(1,2) = p(2) + q(2) + w(1,1)$

$= 3 + 1 + 3 = 7$

$r(1,2) = k(2) = 2$

$c(1,2) \overset{k=2}{=} c(1,1) + c(2,2) + w(1,2)$

$= 0 + 0 + 7 = 7$

$\rightarrow w(2,3) = p(3) + q(3) + w(2,2)$

$= 1 + 1 + 1 = 3$

$c(2,3) \overset{k=3}{=} c(2,2) + c(3,3) + w(2,3)$

$= 0 + 0 + 3 = 3$

$r(2,3) = 3$

$\rightarrow w(3,4) = p(4) + q(4) + w(3,3)$

$= 1 + 1 + 1 = 3$

$c(3,4) \overset{k=4}{=} c(3,3) + c(4,4) + w(3,4) = 3$

$r(3,4) = 4$

$\star\rightarrow j-i=2$

$\rightarrow$ $w(0,2) = p(2) + q(2) + w(0,1)$

$= 3 + 1 + 8 = 12$

$c(0,2) = \min_{k=1,2}\{(c(0,0) + c(1,2) + w(0,2)),$

$(c(0,1) + c(2,2) + w(0,2))\}$

$= \min\{0 + 7 + 12, 8 + 0 + 12\}$

$= \min\{19, 20\} = 19$

$r(0,2) = k = 1$

$\rightarrow w(1,3) = p(3) + q(3) + w(1,2)$

$= 1 + 1 + 7 = 9$

$c(1,3) = \min_{k=2,3}\{(c(1,1) + c(2,3) + w(1,3)),$

$(c(1,2) + c(3,3) + w(1,3))\}$

$= \min\{0 + 3 + 9, 7 + 0 + 9\}$

$= 12$

$r(1,3) = 2$

$\rightarrow w(2,4) = p(4) + q(4) + w(2,3)$

$= 1 + 1 + 3 = 5$

$c(2,4) = \min_{k=3,4}\{(c(2,2) + c(3,4) + w(2,4)),$

$(c(2,3) + c(4,4) + w(2,4))\}$

$= \min\{0 + 3 + 5, 3 + 0 + 5\}$

$= 8$

$r(2,4) = 3$

$\star\rightarrow j-i=3$

$w(0,3) = p(3) + q(3) + w(0,2)$

$= 1 + 1 + 12 = 14$

$c(0,3) = \min_{k=1,2,3}\{(c(0,0) + c(1,3) + w(0,3)),$

$(c(0,1) + c(2,3) + w(0,3)),$

$(c(0,2) + c(3,3) + w(0,3))\}$

$= \min\{(0 + 12 + 14), (8 + 3 + 14),$

$19 + 0 + 14\}$

$c(0,3) = \min\{26, 25, 33\}$

$c(0,3) = 25$

$r = k = 2$

$\rightarrow w(1,4) = p(4) + q(4) + w(1,3)$

$= 1 + 1 + 9 = 11$

$$c(1,4) = \min_{k=2,3,4}\{c(1,1) + c(2,4) + w(1,4),\ c(1,2) + c(3,4) + w(1,4),\ c(1,3) + c(4,4) + w(1,4)\}$$

$= \min\{(0+8+11), (7+3+11), (12+0+11)\}$

$c(1,4) = 19$

$j - i = 4$.

$w(0,4) = p(4) + q(4) + w(0,3)$

$= 1 + 1 + 14 = 15$.

$$c(0,4) = \min_{k=1,2,3,4}\{$$

23/05/2023

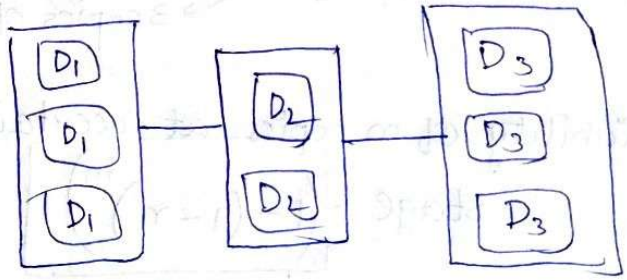
→ Reliability Design problem:

* Given devices $D_1, D_2, \ldots, D_n$.

for any device D_i → Cost(c_i)
→ reliability (r_i)

* Connecting the no. of devices in such a way that; the reliability of the system is improved.

* Using Dynamic Programming; we construct different stages where in each stage; there are 'm' copies of each device D_i.



* Keeping the cost constraint in mind; we shall select a connection which has maximum reliability; but minimum cost

* $C = 105 \quad (c_1, c_2, c_3) = (30, 15, 20)$

$(r_1, r_2, r_3) = (0.9, 0.8, 0.5)$

$$u_i = \left[\left(C + c_i - \sum_{j}^{n} c_j \right) / c_i \right]$$

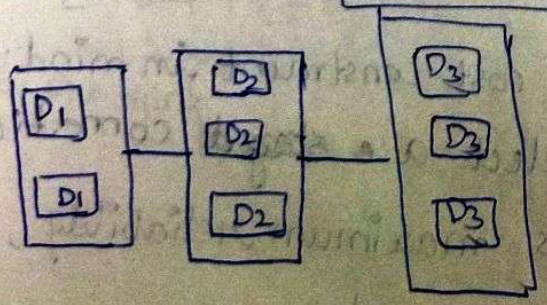
$$u_1 = \left[(C + c_1 - (65)) / 30 \right]$$

$$u_1 = \left[(105 + 30 - 65)/30 \right] = 2.33 \rightarrow$$ 2 copies of D_1

$$u_2 = \left[(105 + 15 - 65)/15 \right] = 3.67 \rightarrow$$ 3 copies of D_2

$$u_3 = \left[(105 + 20 - 65)/20 \right] = 3 \rightarrow$$ 3 copies of D_3

* reliability of m copies at a certain stage $= \left(1 - (1 - r_i)^m\right)$



* Principle of dominance:

Cost ↑ses but reliability↓

* stage → $S_0^0 = \{(30, 0.9)\}$ → 1st stage ; 1 copy

copy

$S_2^1 = \{(60, (1 - (1 - 0.9)^2)\} = \{(60, 0.99)\}$

$S^1 = \{S_1^1, S_2^1\}$ → complete stage 1

$= \{(30, 0.9), (60, 0.99)\}$

$S_1^2 = \{(15, 0.8)\}$

$S_2^2 = \{(30, (1 - (1 - 0.8)^2)\} = \{(30, 0.96)\}$

$S_3^2 = \{45, (1 - (1 - 0.8)^3)\} = \{(45, 0.992)\}$

* $S_1^2 + S^1 = \{(45, 0.72), (75, 0.792)\}$

$S_2^2 + S^1 = \{(60, 0.864), (90, 0.95)\}$ ↓ ∵ cost exceeded

$S_3^2 + S^1 = \{(75, 0.89), (105, 0.98)\}$

∵ cost exceeded

$c_1 < c_2$, $r_1 > r_2$ acc. to principle of dominance

$S^2 = \{(45, 0.72), (60, 0.864), (75, 0.792), (75, 0.89)\}$

$S_1^3 = \{(20, 0.5)\}$

$S_2^3 = \{(40, (1-(1-0.5)^2)\}$

$= \{(40, (1-0.25))\} = \{(40, 0.75)\}$

$S_3^3 = \{(60, (1-(1-0.5)^3)\}$

$= \{(60, 0.875)\}$

0.125
1.000
0.125
0.875

$S_1^3 + S^2 = \{(65, 0.36), (80, 0.432), (95, 0.445)\}$

$S_2^3 + S^2 = \{(85, 0.54), (100, 0.648), (115, 0.6675)\}$

↓ cost exceeded

$S_3^3 + S^2 = \{(105, 0.63), (120, 0.756), (135, 0.875)\}$

cost exceeded

$S^3 = \{(65, 0.36), (80, 0.432), (85, 0.540), (95, 0.445), (100, 0.648), (105, 0.63)\}$.

(Law of dominance)

*Max. reliability $= 0.648 \in S_2^3$.

Cost = 100

Stage-3 : 2 copies $\cong$ 40

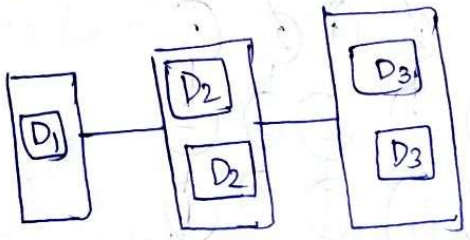
100 - 40 = (60, –)

Stage-2 : $60 \in S_2^2$: 2 copies $\cong$ (30, –)

60 - 30 = 30

Stage-1 : $30 \in S_1^1$: 1 copy $\cong$ (30, –)

*Design



→ Maximize: $\prod_{1 \le i \le n} \phi_i(m_i)$

Subjected to $\sum_{1 \le i \le n} c_i m_i \le c$

where $m_i \ge 1 \; ; 1 \le i \le n$.

24/05/23

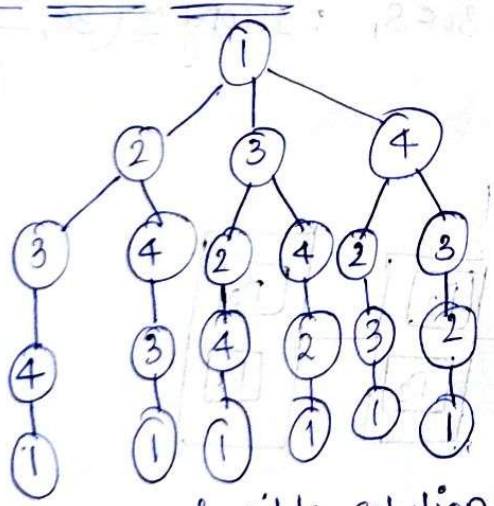
Travelling Sales person problem:

$G = (V, E)$ directed graph

Source to source again.

* This problem deals with tour travelled with minimum cost.

→ Brut Force Method:

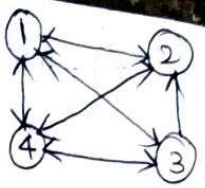


→ This is not a feasible solution:

By dynamic programming, using the cost function:

$$g(i, s) = \min_{k \in s} \{c(i, k) + g(k, s - \{k\})\}$$

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



$g(2, \phi) = c(2,1) = 5$

$g(3, \phi) = c(3,1) = 6$

$g(4, \phi) = c(4,1) = 8$

$g(2, \{3\}) = c(2,k) + g(3, \phi)$

$= c(2,3) + g(3, \phi) = 9 + 6 = 15$

$g(2, \{4\}) = c(2,4) + g(4, \phi)$

$= 10 + 8 = 18$

$g(3, \{2\}) = c(3,2) + g(2, \phi)$

$= 13 + 5 = 18$

$g(3, \{4\}) = c(3,4) + g(4, \phi)$

$= 12 + 8 = 20$

$g(4, \{2\}) = c(4,2) + g(2, \phi)$

$= 8 + 5 = 13$

$g(4, \{3\}) = c(4,3) + g(3, \phi)$

$= 9 + 6 = 15$

$g(2,\{3,4\}) = \min\{\ c(2,3) + g(3,4),$
$c(2,4) + g(4,3)\};$

$= \min\{29, 25\}$

$= 25 \ (k=4)$

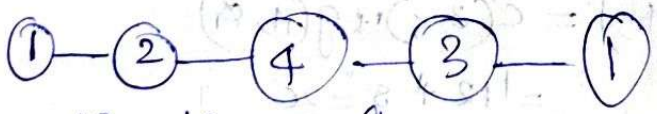
$g(3,\{2,4\}) = 25 \ (k=4)$

$g(4,\{2,3\}) = 23 \ (k=2)$

$g(1,\{2,3,4\}) = \min\{10+25, 15+25, 20+23\}$

$= \min\{35, 40, 43\}$

$= 35 \ (k=2)$



10 10 9 6 = 35

minimum cost = 81.

* Longest Common Subsequence:

$S_1 = abcfde$

$S_2 = bdeaf$

* Finding the string which has longest length which is common to both.

→ This checking for common sequence can be done only in one direction.

Ex: S_1 = stone.

S_2 = longest

		l	o	n	g	e	s	t
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
s	1	0	0	0	0	0	1	1
t	2	0	0	0	0	0	1	2
o	3	0	0	1	1	1	1	2
n	4	0	0	1	2	2	2	2
e	5	0	0	1	2	2	3	3

Subsequence = one

if (A[i] == B[j])

LCS[i,j] = 1 + LCS[i-1, j-1]

else

LCS[i,j] = max(LCS[i-1, j], LCS[i, j-1]);

$L[u] = \min\{df[u], \min\{L[w] \mid w \text{ is a child of } u\}, \min\{df[w] \mid (u,w) \text{ is a back edge}\}\}$.

29/05/2023

UNIT-4

BACK-TRACKING:

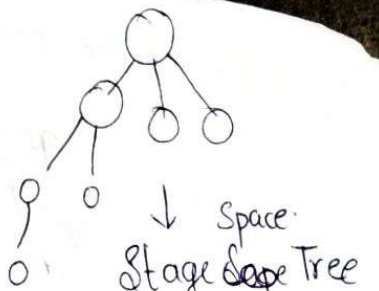
4 queens: 4 × 4 chessboard is given; placing each queen in each row; but any 2 queens should not be in same column (or) diagonal.

→ Back tracking deals with finding all the solutions for a certain problem.

→ Brute Force: Deals with finding sol^n with the help of bounding fn and check for all the possibilities.

*Implicit constraints.

*Explicit Constraints.



*Algorithm Backtrack(k)

```
{ for (each x[k] ∈ T[x[1], ....., x[k-1]]) do
  { if (Bk(x[1], x[2], ....., x[k]) ≠ 0)
    then
    { if (x[1], x[2], ... x[k] is a
                path to an answer node).
      then write( x[1:k]);
    if (k < n) then Backtrack(k+1);}}
}
```

*N Queens problem:

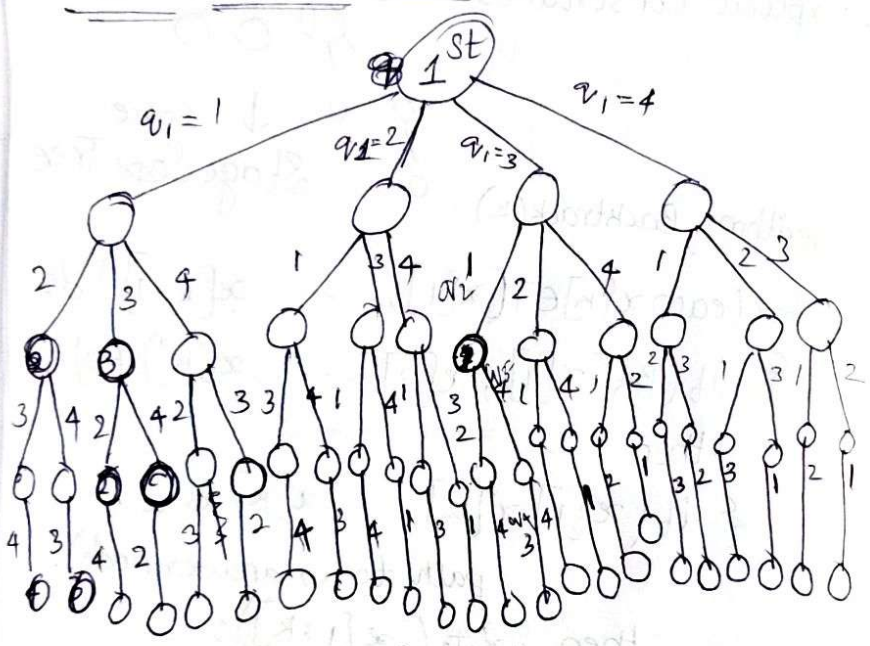
N×N → chess board is given

N → queens are given

Bounding function:

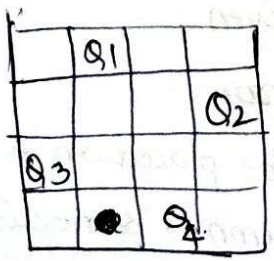
No 2 queens can be placed in same row, same column or same diagonal.

* Consider 4 queens problem.

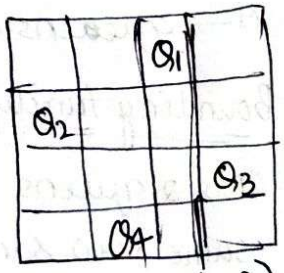


* $1 + \sum_{i=0}^{3} \left(\prod_{j=0}^{i} (4-i) \right)$

$$1 + \sum_{i=0}^{N} \left(\prod_{j=0}^{n-1} (N-i) \right)$$



2, 4, 1, 3



(3, 1, 4, 2)

→ Algorithm NQueens(k, n)

```
{ for i := 1 to n do
  { if Place(k,i) then
    { x[k] := i;
      if (k = n) then write (x[1:n]);
      else
        NQueens(k+1, n); }} }.
```

→ Algorithm Place(k, i)

```
{ for j := 1 to k-1 do
  { if ((x[j] = i) or (Abs[x[j]-i)
                       = Abs(j-k)))
    then return false;
    return true; }
```

→ Algorithm NQueens(k,n)

```
{ for i:= 1 to n do
  { if Place(k,i) then
    { x[k]:=i;
      if (k=n) then write (x[1:n]);
      else
        NQueens(k+1,n);}}}.
```

→ Algorithm Place(k,i)

```
{ for j:= 1 to k-1 do
  { if ((x[j]=i) or (Abs[x[j]-i)
                    = Abs(j-k)))
      then return false;
    return true;}
```

31/05/2023

→ Graph Colouring problem:

- Decision problem (Yes/No kind)
- Optimization problem (finding optimal soln).

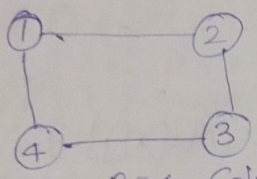
$G = (V, E)$, given a set of colours.

→ Bounding function

* We must colour the graph such that no 2 adjacent vertices have same colour.

→ We must check for minimum no. of colours to colour the graph ↓ (called chromatic no. for graph)

* Example:

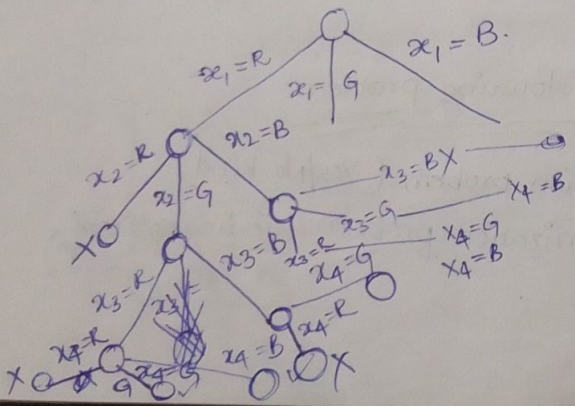


$n = 4$ Colours = {R, G, B}

If C → chromatic no. of graph

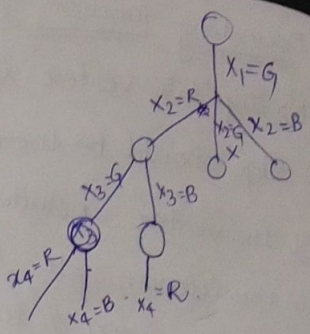
n → no. of vertices.

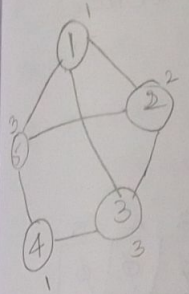
∴ time complexity $\propto C^n$



→ RGRG
RGRB
→ RGBG
→ RBRG
→ RBRB
→ RBGB

GRGR
GRBR





→ Chromatic no = 3

∵ In any graph, the min no. of colours required from the set of colours to colour all the nodes checking with bounding function helps in get chromatic no.

* Applications:

→ Map Colouring.

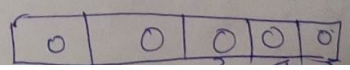
→ Register allocation & assignment.

09/06/2023

→ Hamiltonian Cycle:

$G = (V, E)$

V_1, V_2, V_3, V_4, V_5.



* Bounding function

1) No duplicate vertex in Hamiltonion cycle.

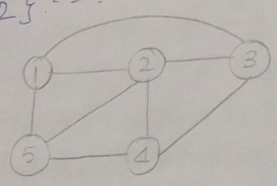
2) edge should be there from k to (k+1)

3) If the vertex included is n^{th} vertex; in Hamiltonion cycle; then there should be an edge from n to ~~to~~ the 1^{st} vertex.

1	2	3	4	5
~~ϕ~~	0	0	0	0
1	~~2~~			

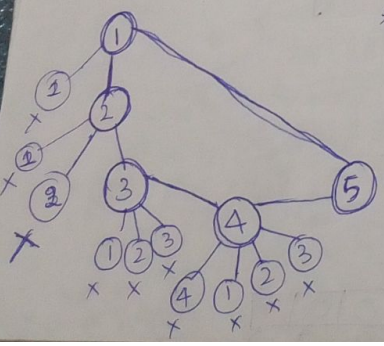
{1;2} ...

*



Set = {1,2,3,4,5;1}

1	2	3	4	5
~~ϕ~~	~~ϕ~~	~~ϕ~~	~~ϕ~~	~~ϕ~~
1	~~2~~	~~3~~	~~4~~	~~5~~
	2	~~3~~	~~4~~	~~5~~
		3	~~4~~	~~5~~
			4	~~5~~
				5



* After finding a soln we must check for alternate in each step.

1	2	4	~~ϕ~~	~~ϕ~~
5	~~2~~	~~3~~	~~4~~	~~5~~
	2	5	~~4~~	~~5~~
		3	4	~~5~~

ans = {1,5,2,~~4,3~~}

= {1 2 5 3 4 3 1}

= {1 3 4 5 2 1}

* Algorithm:

```
Algorithm Ham(k)
{ repeat {
    NextValue(k);
    if (x[k] = 0) then
        return;
    if (k = n) then
    write(x[1:n])
    else
        Ham(k+1) } until(false); }
```

```
Algorithm nextvalue(k)
{ repeat {
  x[k] := (x[k]+1) mod (n+1)
  if(x[k]=0) then return;
  if(G[x[k-1], x[k]] ≠ 0) then
```

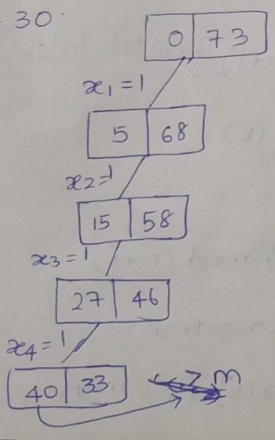
```
for j:=1 to k-1 do
    if (x[j] = x[k]) then break.
if (j=k) then
    if ((k<n) or ((k=n) and (x[n],
                    x[i] ≠ 0))
    then return;
}
} until (false)
}
```

* Sum of subsets:

→ fixed size sol^n = no. of elements given
→ variable size sol^n = no. of elements selected.

w[1-6] = {5, 10, 12, 13, 15, 18} = 73

m = 30



* Bounding function:

* $\sum_{i=1}^{k} (w_i x_i + w_{k+1} x_{k+1}) \leq m$

$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i > m$

BRANCH & BOUND:

→ Minimization of bounding function.
→ Follows BFS

Types of branch & bound: — (DSP, 0/1 Knapsack)

1) FIFO BB (queue)
2) LIFO BB (stack)
3) LC BB (Least cost)

* Control Abstraction:

```
Listnode = record {
        Listnode * next, * parent;
        float cost; }

Algorithm LC search( )
{ if *t is an answer node then
    output *t and return;
  E:=t;
  Initialize 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 ← parent = E }
If there are no more live nodes
then
{ write ('No answer node');
return;
}
E := least ( ) (least value from list of values)
}
until (false);
}

Travelling Sales Person

* TSP with least Cost Branch & Bound:

15/06/2023

	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	∞

⇒ Reduced cost matrix

1 row & 1 col is 0

∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
15	3	15	∞	0
12	0	3	12	∞

$R_1 = 10$
$R_2 = 2$
$R_3 = 2$
$R_4 = 3$
$R_5 = 4$

∞	10	20	0	1
12	∞	14	2	0
0	3	∞	0	2
15	3	15	∞	0
11	0	3	12	∞

(i, j) edge i^{th} row & j^{th} col to ∞ and (j, 1) too to go from 1 to 2

∞	∞	∞	∞	∞
∞	∞	11	2	0
0	∞	∞	0	2
5	∞	12	∞	0
11	∞	0	12	∞

All col & row has 0 so no need to reduce: $C(2) = C(1) + A[i, j] + x$

$= 25 + 10 + 0$ (no reduction)

$= 35$.

to go from 1 to 3

∞	∞	∞	∞	∞
12	∞	∞	2	0
∞	3	∞	0	2
15	3	∞	∞	0
11	0	∞	12	∞

$C_1 - 11 \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$

$C(3) = C(1) + A[i,j] + r$

$= 25 + 17 + 11 = 53.$

to go from 1 to 4 $\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$.

$C(4) = C(1) + A[i,j] + r$

$= 25 + 0 + 0 = 25.$

to go from 1 to 5

$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$

$R_2 \rightarrow R_2 - 1 \; ; \; R_4 \rightarrow R_4 - 3$

$C(5) = 25 + 1 + 5 = 31$

We keep i^{th} row & j^{th} col to ∞ because we dont want to change their values as we are moving across them. After every movement we try to reach easily.

15/06/2023

UNIT-5

NP-Hard and NP Complete Problem:

Polynomial time (P) (Deterministic)

- Searching
- Sorting
- Matrix Chain Multiplication

Non-Polynomial Time (NP)

- 0/1 Knapsack
- N-Queens
- TSP
- Hamiltonian Cycle
- Graph Colouring

↓ complexity is in exponential form.

* Algorithm NSearch (A, n, key)

```
{ j = choice( );        ⟶ O(1)
  if (key == A[j])
  { write(j)
    success(); }        ⟶ O(1)
write(0);
failure(); }            ⟶ O(1)
```

→ choice( ): → returns a position which will probably have a key element from the given array.

* choice()
  success()
  failure()
} Non-deterministic functions.

→ NP Hard & NP complete provide a set of rules
* In order to solve the exponential time algorithms in polynomial time complexity

→ Reducibility is a property which deals with if we are having one instance of one problem; if we can convert it to other problem and if we are having a solution; which satisfies all the problem.

→ Satisfiability Problem (SAT)

Conjunctive Normal Form :

$(x_1 \wedge \bar{x}_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \bar{x}_3)$.

↳ We need to check 2^n combinations in order to tell for which formula it is true

→ It can be defined as whenever conjunctive normal form of n boolean variables are given; find all possible combinations of $x_1, x_2 \ldots, x_n$ for which the formula is true.

* Algorithm Eval(E, n)

```
{ for i := 1 to n do
      xi = choice(T, F)
  if (E(x1, x2 ...., xn)) then
      success();
  else
      failure();
}
```