Syllabus

22 September 2022 07:04 PM

UNIT-1 **DIVIDE-AND-CONQUER**

What is an algorithm, Performance Analysis- Space complexity, Time Complexity, Asymptotic Notation, and Divide-and-Conquer- Introduction, Binary Search- Iterative and Recursive, finding the Maximum and Minimum, Merge Sort, Quick Sort, Heap Sort.

UNIT-2 **GREEDY METHOD**

Introduction , 0/1 Knapsack Problem, Job scheduling, Huffman codes, Minimum cost spanning trees- Prim's Algorithm, Kruskal's Algorithm, Optimal Merge Patterns.

UNIT-3 **DYNAMIC PROGRAMMING**

Multistage Graphs, All pairs shortest path, single source shortest path, Optimal Binary Search tree, Traveling Sales man problem, Flow shop Scheduling.

UNIT-4 **BACK TRACKING**

Introduction, The 8 queens problem, Sum of Subset, Graph coloring, Hamiltonian cycles ,Branch and Bound

2

UNIT-5 **BASIC TRAVERSAL AND SEARCH TECHNIQUES**

Techniques for binary trees, Techniques for graphs- Breadth First Search and traversal, Depth First Search and traversal, Connected components and Spanning Trees.

Algorithm & How-to 's

25 September 2022 12:39 PM

Algorithm; It is a finite set of instruction that, if followed, accomplishes a particular task

An algo. must Satisfy following Criteria

1. **Input** : Zero or more quantities are supplied externally
2. **Output** : at least one output is produced
3. **Definiteness** : Each instruction is clear & Unambiguous
4. **Effectiveness** : Every instruction is very basic & feasible
5. **finiteness** : an algorithm must terminate after finite number of steps

Four areas of Study of algorithm ;

1. How to devise Algorithm

Writing an algorithm is the technique of designing algorithm in such a way that the algo. must produce approx. correct output. Sometimes the help of dynamic programming techniques are used

2. How to validate Algorithm

After algorithm design, it is necessary to show that it computes correct answers for all possible legal inputs. The purpose of validation is to assure that the algorithm will work correctly independently of issues concerning programming

Secondly (after algorithm is written in programming lang) program Verification takes place, in this phase the Specification is verified as per programming grammer etc.

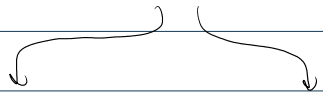
(3) How to analyse algorithm

Analysis of algorithm or performance analysis is the determination of compute time & storage required by the algorithm.

This analysis allows us to make quantitative Judgement about the value of one algorithm over another. Also it also helps us to predict weather the Software will meet any efficiency constraints of best case, worst case or in average case

(4) How to test a program

Testing of program consists of two parts :



Debugging is the process of executing programs on Sample data sets for determining weather the results occurs correct or faulty

If faults occurs then correct them. However the process of debugging can only point to the presence of error but not their absence (As such it doesn't tell us the correctness)

Profiling is the process of executing correct program on the data sets & Calculating timing & Space it takes to calculate the results.

Performance Analysis

Space Complexity : It is the amount of memory req. by the algorithm to proceed or complete it

Space complexity consist of two parts

- **fixed part :** It includes instruction space for variables & fixed sized component, space for constant (independent of char of input or output)
- **Variable part :** Consists of space needed by componet variable whoes size is dependent upon particular problem & recursion stack

Time Complexity : The amount of time required to execute a program

It includes both compile time & Run time

- compile time → doesn't depend on instant characteristic
- Run time → generally varies

Freq. Count Example

27 September 2022 08:12 PM

(for getting steps count freq.)

Example Sum(a,n)

Statement	S/E (Steps for Execution)	Freq.
Algorithm Sum(a,n)	0	0
{	0	0
S = 0;	1	1
for i=1 to n do	1	n+1 ----> +1 because it also checks for failure
S = S + a[i];	1	n
return S	1	1
}	0	0
		⇓
		2n+3

Statement	S/E (Steps for Execution)	Freq.
Algo Add(a b c m n)	0	0
{	0	0
for i=1 to m do	1	m+1 ----> +1 cuz check for fail
for j=1 to n do	1	m(n+1) ---> m cuz this executes m times
c(i,j) = a(i,j) + b(i,j);	1	m.n ---> "
}	0	0
		⇓
		total = 2mn + 2m + 1

Asymptotic Notation !

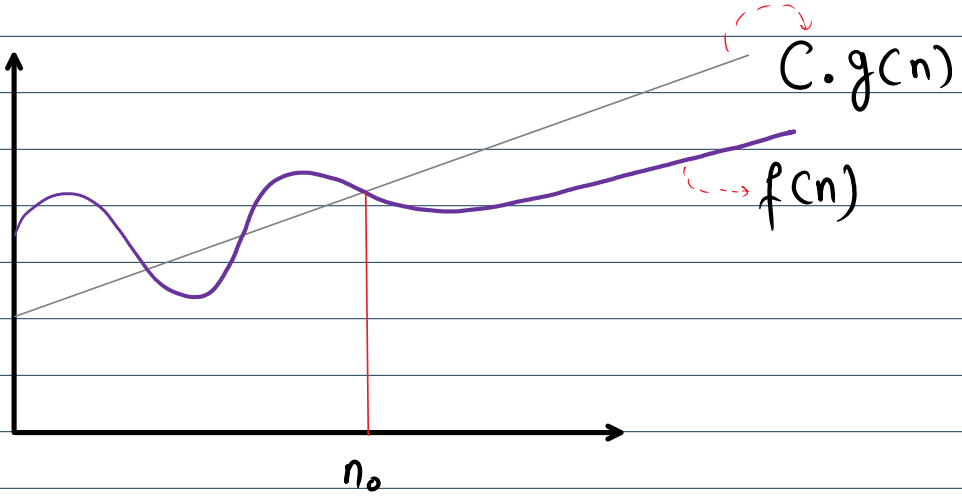
These are mathematical notation used to describe the runtime of algorithm

Big (O) notation

The function $f(n) = O(g(n))$

if & only if there exists a positive constant "c" & "n_0" such that

$$f(n) \leq c \cdot g(n) \quad \forall \; n, n \geq n_0$$



Example,

Suppose $10n^2 + 4n + 2 = O(n^2)$

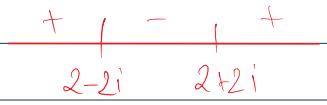
$10n^2 + 4n + 2 \leq 11n^2$ (at random cause the true could have taken $12n^2$)

Solving in equality

$$4n+2 \leq n^2$$

$$n^2 - 4n - 2 \geq 0$$

$\frac{-b \pm \sqrt{b^2-4ac}}{2a} \Rightarrow \frac{4 \pm \sqrt{16-16\cdot 2}}{2} \Rightarrow \frac{4 \pm \sqrt{-16}}{2} \Rightarrow \frac{4 \pm 4i}{2} = 2 \pm 2i$



for $n \geq 2+2i$ $\quad n^2 - 4n - 2 \geq 0$

for $n \leq 2-2i$ $\quad n^2 - 4n - 2 \geq 0$

Thus $10n^2 + 4n + 2 \leq 11n^2 \quad \forall n \geq 2+2i$

Divide & Conquer (DnC)

Given a function to compute n inputs, suggests splitting inputs into k subsets of k subproblems

These subproblems must be solved & then a method must be used to combine subsolution

If subproblems are relatively large (often subproblem are of same nature) can be further devided into by this method

```
Algo BinarySearch( a, i, l, x)
{
  if( l=i ) then
  {
    if( x = a[i] )
      then return i;
    else
      return 0;
  }
  else
  {
    mid = (i+l)/2
    if (x = a[mid]) then return mid;
    else if( x < a[mid] then
          return BinarySearch(a, i, mid-1, x)
    else
      return BinarySearch(a, mid+1, l, x)
  }
}
```

array a[i:l]

Binary Search Recursive

Binary Search { ITERATIVE }

```
Algo Binary Search (a, n, x)
{ low := 1 ; high = n ;
   while (low ≤ high) do
   {
      mid = [ (low + high)/2 ] ;
      if ( x < a[mid] )
          then high = mid-1 ;
      else if ( x > a[mid] )
          then low = mid+1 ;
      else return mid;
   }
}
```

For example, array a =>

1	2	3	4	5	6	7	8	9	10	11	12	13	14 = n
-15,	-6,	0,	7,	9,	23,	54,	82,	101,	112,	125,	131,	142,	151

finding if 151 is present or not

low	high	mid
1	14	7
8	14	11
12	14	13

0	11	11
12	14	13
14	14	14

= @ 14 found ISI

Cheking of -14 present or not

low	high	mid	
1	14	7	
1	6	3	
1	2	2	(!1)
2	1	fail	∴ Not found

!Find Min and Max (DnC)

28 September 2022 12:20 AM

?Merge Sort (DnC) (Algo)

28 September 2022 12:41 AM

Merge Sort

```
Algorithm MergeSort (low, high)
{
    if (low < high) then
    {
        mid = (low + high)/2;
        MergeSort (low, mid)
        MergeSort (mid+1, high)
        Merge (low, mid, high)
    }
}
```

Merge Algorithm

```
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];
            i++;
```

```
        h++;
    }
    else {
        b[i] = a[j]
        j++;
    }
}

if(h > mid) then
    for k = j to high do
    {   b[i] = a[k]
        i++;
    }

else {
    for k = h to mid do
    {   b[i] = a[k];
        i++;
    }
for k = low to high do
    a[k] = b[k]
}
```

Merge Sort Time Complexity –

Merge Sort Time Complexity

for a process

if $n = 1$

$T(n) = a$ → some constant time

if $n > 1$ time → $T(n)$

the process is divided into two

$$T(n) = 2T(n/2) + cn \quad \text{— (1)}$$

Some const. affected by n

Similarly

replacing $n \to n/2$ in prev eq

$$T(n/2) = 2T(n/4) + cn/2 \quad \text{— (2)}$$

(2) in (1)

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

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

⋮

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

⋮

So on.

Thus

$$T(n) = 2^k T(n/2^k) + kcn \quad \text{— (6)}$$

Assuming $n = 2^k$ (10K) — (5)

$$\log n = k \log 2$$

$$\log n = k \quad \text{— (4)}$$

(5) in (6)

$$T(n) = n\,T(n/n) + kcn$$

$$= n\,T(1) + c\,n \log n$$

$$T(n) = c\,n \log n + n.a$$

$$(0) = n \log n$$

Greedy Method

A straightforward method in which problem have n inputs & require us to obtain subset that satisfys a constraints

Any subset which satisfys is called feasible solution. We need to find a feasible soln which either Maximizes or Minimizes the given function & such soln are called optimal solution

Greedy method suggests that the algorithm works in stages, considering one input at a time.

At each stage decision is taken weither a particular soln is optimal or not

Knapsack problem

Given n objects & a knapsack object i has weight w_i

The knapsack capacity m the object i is placed into the knapsack then profit $p_i x_i$ is earned.

Example

$n = 3 \qquad m = 20$

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

$(w_1, w_2, w_3) = (18, 15, 10)$

feasible Solution

x_1	x_2	x_3	$\sum w_i x_i$	$\sum p_i x_i$	
1	2/15	0	20	28.2	$(25 \times 1 + 24 \times \frac{2}{15})$
0	1	5/10	20	31.5	→ optimal Solution

Best way is to Calc. $\frac{p_i}{w_i}$

$\frac{25}{18}, \frac{24}{15}, \frac{15}{10}$

1.38 1.6 1.5

b) Select higher values

Algorithm for Knapsack problem

```
Algo GreedyKnapsack(m, n)
{
  for i=1 to n do x[i] = 0.0 ;
  U = m;
  for i=1 to n do
  { if (w[i] > U) then break;
    x[i] = 1.0 ;
    U = U - w[i];
  }
  if (i<=n) then
      x[i] = U/w[i]
}
```

Time Complexity : O(n)

Job Scheduling with deadline

Given the set of n jobs, job i is an integer deadline with $d_i \geq 0$ & $P_i > 0$

for any job i, profit P_i is earned if & only if the job is completed by its deadline

To complete a job one has to process a job on a machine for one unit of time

Only one machine is available for processing jobs

feasible soln for problem is subset of jobs such that each job in the subset can be completed by its deadline

The value of feasible soln "J" is sum of profits of jobs & optimal soln is soln with max value

Example,

$n = 4$

$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$

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

Sort by d & write pos^n

possible soln → optimal

27	1	4
10	1	2
100	2	1
15	2	3

possible sol^n → optimal

~~42~~, (41), 43, 13

21, 23, 1, 2, 3, 4

4, 1 → 100 + 27 = (127) ✓

Algo HighGreedyJob (d, J, n)
{
    J = {1} ;
    for i = 2 to n
    { if (all jobs in J ∪ {i} can be completed by their deadline)
        then
            J = J ∪ {i} ;
    }
}

Algorithm JS(d, j, n)
    d[0] = J[0] = 0
    J[1] = 1
    K = 1

    for i = 2 to n do
    {   r = K

```
{
    r = k
    while ( d[J[r]] > d[i] & d[J[r]] ≠ r ) do
    { r = r - 1 }
    if ( d[J[r]] > d[i] & d[i] > r ) then
    { for q = k to (r + 1) step -1 do
      { J[q+1] = J[q]
      }
    }
    return k
}
```

Minimum Spanning Tree

28 September 2022 08:33 AM

!Prim's Method

18 December 2022 06:22 PM

!Kruskal's Method

18 December 2022 06:22 PM

Optimal Merge Pattern

18 December 2022 06:23 PM

In optimal merge pattern three or more sorted files are merged into one sorted file with min. number of computation

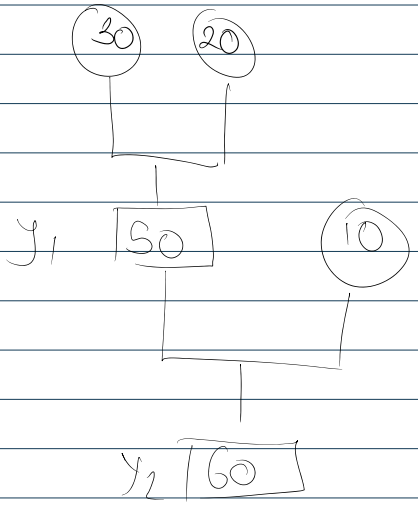
merging done by taking two pairs at a time

Example,

Three files $(x_1\ x_2\ x_3) = (30, 20, 10)$

In case we merge directly then

x_1 x_2 x_3

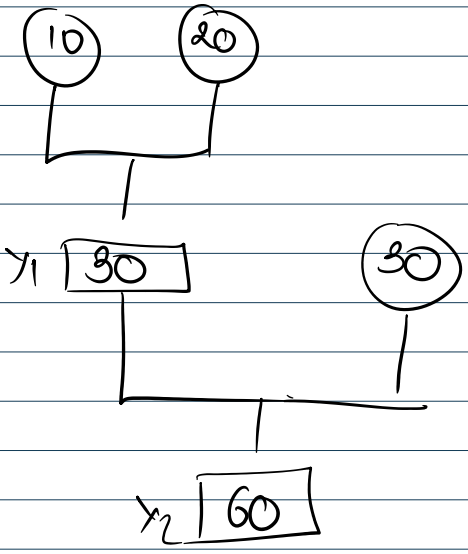


total comparisons $= y_1 + y_2$

$= 110$

however if we sort it out

x_3 x_2 x_4



Now $y_1 + y_2 = 90$

optimal ans

Steps

- first of all sort the files then merge into bigger file & repeat

here,

here merging x_3 & x_2 to get y_1

then merging y_1 & x_1 to get y_2

total comparison $\Rightarrow y_1 + y_2$

```
treenode = record {
    treenode * leftchild ; treenode * right child
    integer weight
}

Algorithm Tree(n) {
// list is global list of n single node
for i = 1 to n-1 do {
    pt = new tree node ;
    (pt -> Lchild) = least(list)
    (pt -> Rchild) = least(list)
    (pt -> weight) = (pt -> Lchild -> weight)
                   + (pt -> Rchild -> weight)
    Insert(list, pt)
}
return (list)
}
```

Huffman Codes

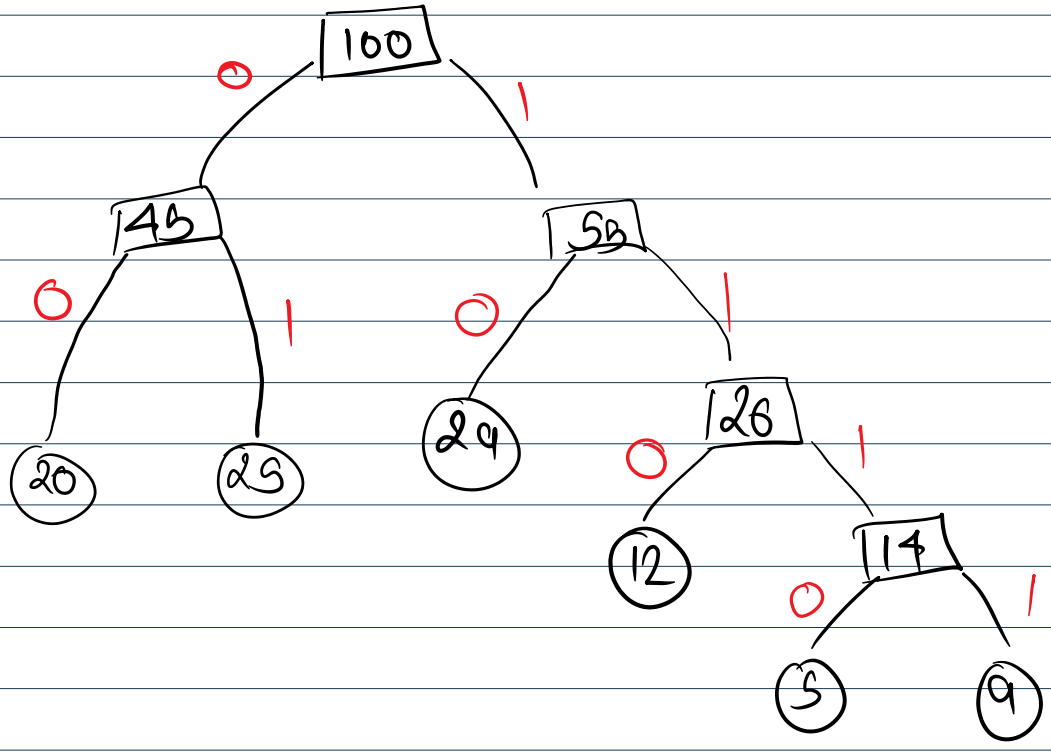
Another application of Binary tree

In this basically transmission of message is done where
Code is decoded by decode tree in which external nodes represent messages.

For Example,

Given characters with following characters

P { 29, 20, 25, 12, 5, 9 }
x_6 x_5 x_4 x_3 x_2 x_1



So far code

24 = {10} 25 = {01} 20 = {00}

12 = {110} 5 = {1110} 9 = {1111}

{ Same algo as Optimal Merge pattern }

!Dynamic Programming

18 December 2022 07:10 PM

All pairs Shortest-path

Given $G(V, E)$ is adjacent graph with n vertices

Cost adjacency matrix for G is such that

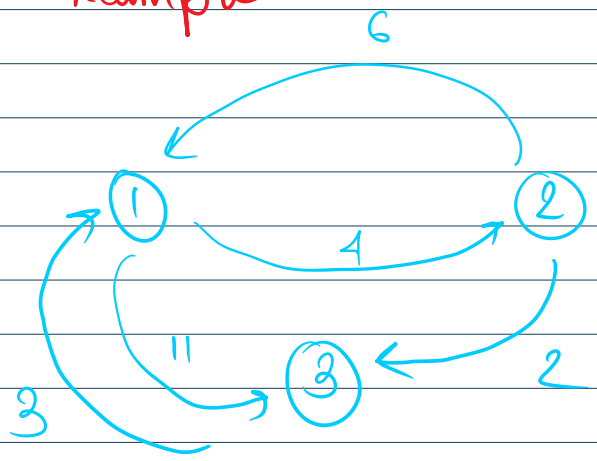
- Cost $c[i,i] = 0$
- Cost $c[i,j]$ = length of edge $\langle i,j \rangle$

  Until & unless there is no edge $\langle i,j \rangle$ then $c[i,j] = \infty$

```
Algorithm AllPairs(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] }
            }
        }
    }
```

Example :



$$A^0 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 9 & 0 \end{array}$$

{K = 1}

$A^1(1,1) \quad \infty$

$A^1(1,2) \quad \infty$

$A^1(1,3) \quad \infty$

$A^1(2,1) \quad \infty$

$A^1(2,2) \quad \infty$

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

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

$\rightarrow 2$

$A^1(3,1) \quad \infty$

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

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

$\rightarrow$ (7)

$A^1(3,3)$ α

$$A^1 = \begin{Bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{Bmatrix}$$

$\{k=2\}$

$A^2(1,1)$ α

$A^2(1,2)$ α

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

$\{11, 4+2\}$

$\Rightarrow$ (6)

$A(2,1)$ α

$A(2,2)$ α

$A(2,3)$ α

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

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

α

$A(3,2)$ α

$A(3,3)$ α

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

$\{k=3\}$

$A^3(1,1)$ α

$A^3(1,2)$ & so on.....

Optimal binary Search Tree

Total variations $T(n) = \frac{2n C_n}{n+1}$

Example,

	10	20	30	40
p(i)	3	3	1	1
q(i)	2	3	1	1	1

Cost $[0:n] \Rightarrow \sum_{1}^{n} p_i * level(a_i) + \sum_{0}^{n} q(i) * level(\epsilon_{i-1})$

Solution requires getting minimum cost

$$C[i,j] = \min_{i<k\le j} \{ C[i,k-1] + C[k,j] \} + w[i,j]$$

for example,
C[0,3] we need C[0,0], C[1,3] & C[2,2], C[3,3]
C[0,1] & C[1,2], C[2,3]
C[0,2], C[1,3]
C[0,3]

& $w[i,j] = w[i,j-1] + p_j + q_j$
(basically failure probability)

key { 10, 20, 30, 40 }
p{ 3, 3, 1, 1 }
q{ 2, 3, 1, 1, 1 }

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

w[0,0] = q_0
w[1,1] = q_1
Similarly for w_{22} to w_{44}

C[0,0] = 0 as $i<k\le j$

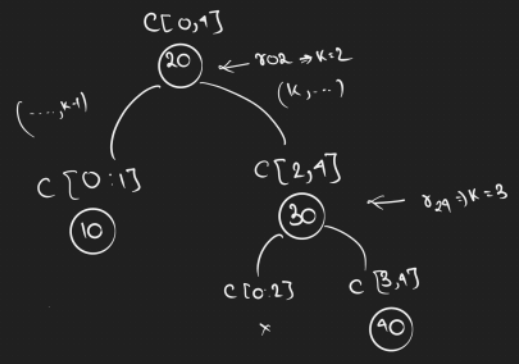
$w[i,j] = w[i,j-1] + p_j + q_j$

$C[i,j] = \min_{k} \{ C[i,k-1] + C[k,j] \} + w_{ij}$

$C[0,1] = \min_{k=1} \{ C[0,0] + C[1,1] \} + 8$

$\Rightarrow 0 + 8 \Rightarrow 8$

$C[1,2] = \min \{$



$C[1,4] = \min_{1<k\le 4} \{$

2 C[1,1] + C[2,4] → 0 + 8
3 C[1,2] + C[3,4] → 7 + 3
4 C[1,3] + C[4,4] → 12 + 0

$\} + w_{14}$

$C[0,4]$
$= \min_{0<k\le 4} \{$

1 C[0,0] + C[1,4] → 0 + 19
C[0,1] + C[2,4] → 8 + 8

$C[1,2] = \min_{1<k\leq 2} \{$

$k=2 \rightarrow \{C[1,1] + C[2,2]\} + w_{12}$

$0 + 0 + 7$

$k=2$ ✓
1 $C[0,0], C[1,1] \rightarrow 0+19$
2 $C[0,1] + C[2,2] \rightarrow 8+8$
3 $C[0,2] + C[3,3] \rightarrow 19+3$
4 $C[0,3] + C[4,4] \rightarrow 23+0$

Thus for c_{12} c_{22} c_{32} → their resp weights

$C[0,2] = \min_{0<k\leq 2} \{$

✓ $C[0,0] + C[1,2], \rightarrow 0+7$
2 $C[0,1] + C[2,2], \rightarrow 8+0$

$\} + w[0,2]$

$\Rightarrow 7 + w[0,2]$

$C[1,3] = \min_{1<k\leq 3} \{$

$k=2$ ✓ 2 $C[1,1] + C[2,3], \rightarrow 0+3$
3 $C[1,2] + C[3,3], \rightarrow 7+0$

$\} + w_{13}$

$C[2,4] = \min_{2<k\leq 4} \{$

3 $C[2,2] + C[3,4], \rightarrow 0+3$
1 $C[2,3] + C[4,4], \rightarrow 3+0$

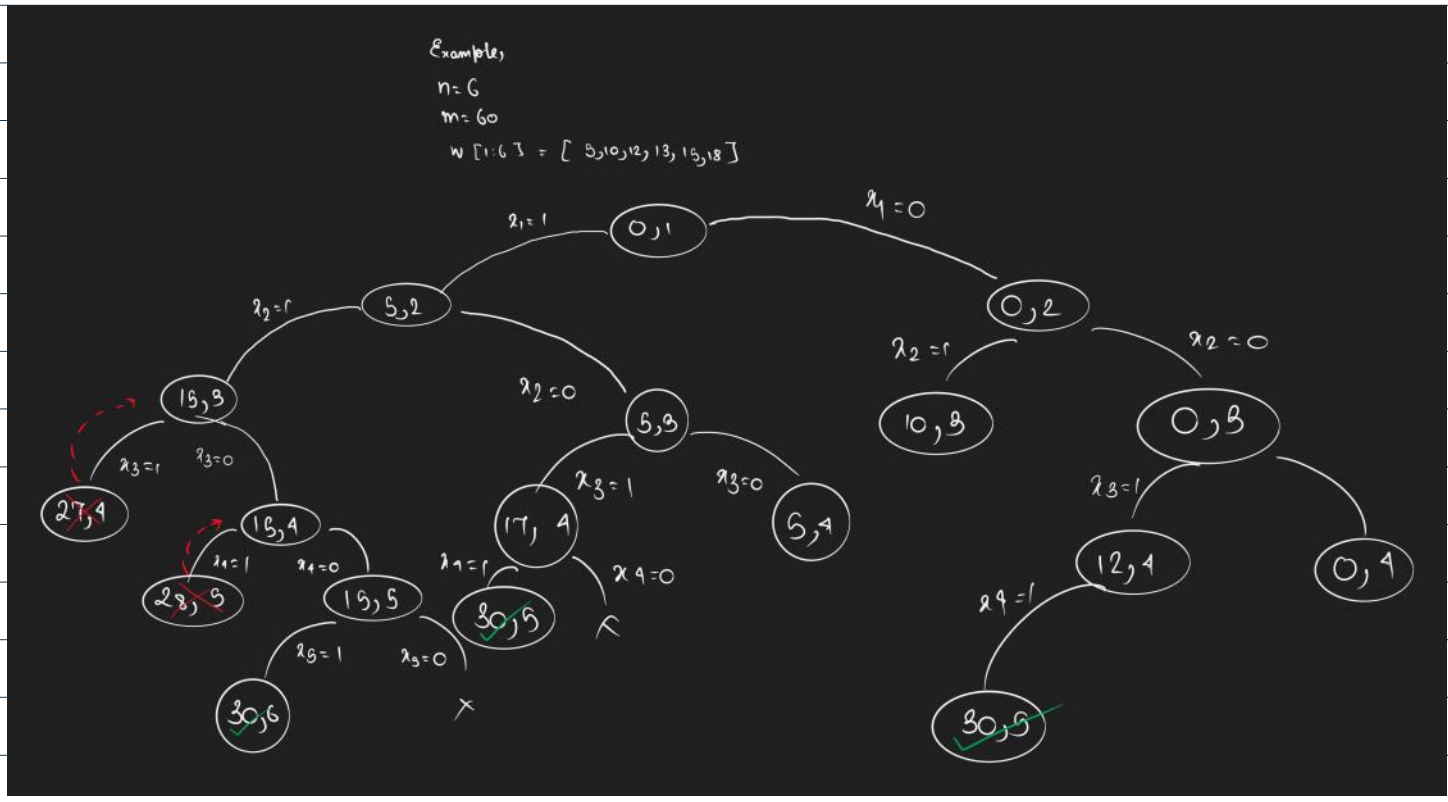
$\} + w_{24}$

$C[0,3] = \min_{0<k\leq 3} \{$

1 $C[0,0] + C[1,3] \rightarrow 0+12$
✓ 2 $C[0,1] + C[2,3] \rightarrow 8+3$
3 $C[0,2] + C[3,3] \rightarrow 19+0$

Sum of Subsets

19 December 2022 04:03 AM



place queens such that they aren't under attack

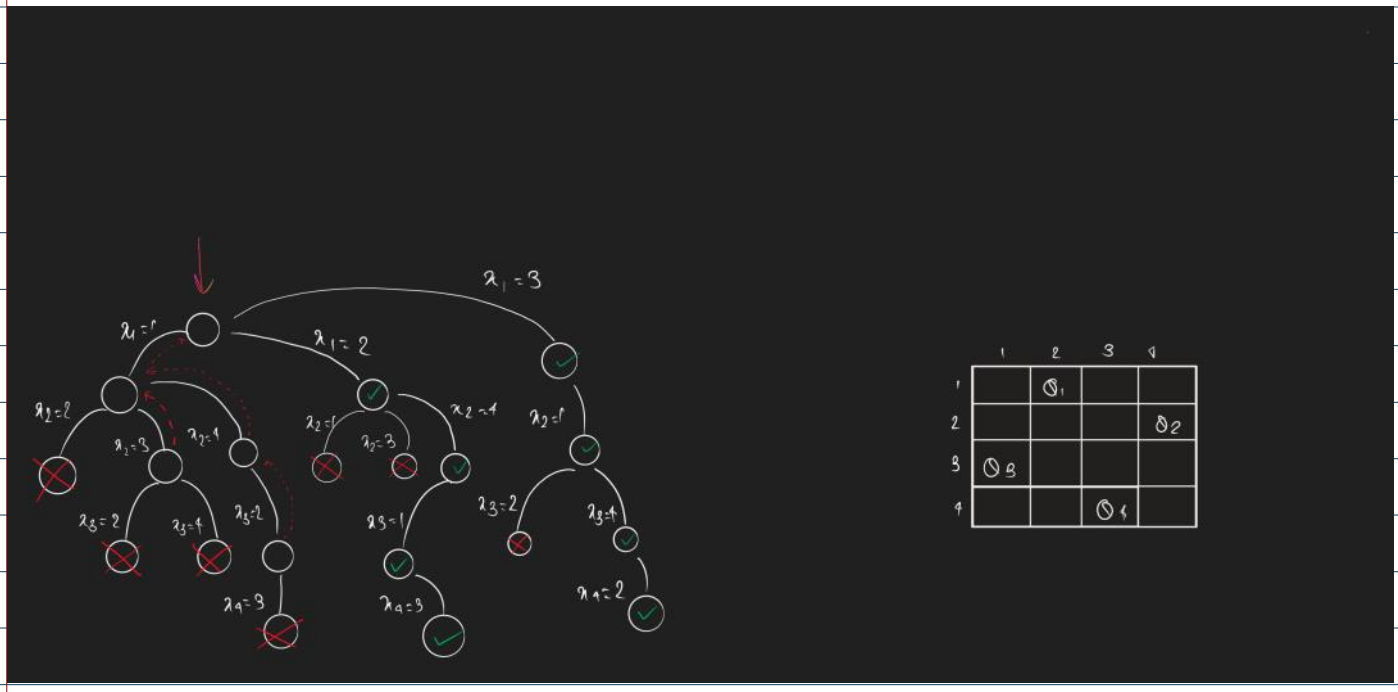
ie. ◦ not on same row, column & diagonal

◦ So we place we decide that each queen is placed on her respective rows

↓

row condition taken care of

◦ then we take measure such that no two queens take single column & then resolve diagonal condition



Algorithm Place (k, i) {

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

```
Algorithm N-Queen (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);
        }
    }
}
```

Graph Coloring

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

Algorithm NextValue(k) {
    repeat {
        x[k] = (x[k]+1) mod (m+1)
        if(x[k]=0) then return;
        for j=1 to n do {
```

```
if (G[k,j] ≠ 0 & (x[k] = x[j])) {
                break;
        }
    }
    if( j = n+1) then return
} until(false)
}
```