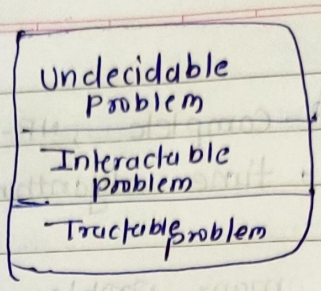
★ P, NP, NP-Complete, NP-hard

Polynomial - time algorithm

→
- There are many problem for which no-polynomial time algorithm is known. Some of these problems are travelling salesman optimal, graph coloring knapsack, hamiltonian cycle
- These problem belongs to an intresting class of problems called "NP-complete" problem whose status is known

→ Polynomial Time Algorithm

- Polynomial Time is a reduction which is computable by a deterministic turing machine in polynomial time.
- Algorithm with coorst time case running time $O(n^k)$, where k is constant, are called tractable or super polynomial.
- An algorithm is a polynomial time algorithm if there exists a polynomial $P(n)$ such that the algorithm can solve any instance of size n in a time $O(P(n))$.
- Problem requiring $\Omega(n^{35})$ time to solve are essential intractable for large n.
- Undecidable problems are halting problem of turing machine



classification of problem

- The advantages in considering the class of polynomial time algo. is that all resonable deterministic single processes model of computation can be simulated on each other with at most a polynomial slowdown

classification of problem

1) P: "P" stands for polynomial problems that can be solved in polynomial time
- More specially, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of input to the problem

2) NP: "NP stand for non-deterministics polynomial time" where non-deterministic is just a fancy way of talking about guessing a solution.
- A problem is in NP if we can quickly test whether a solution is correct

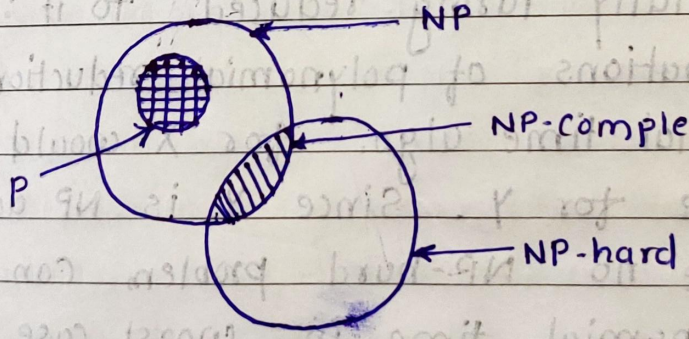
3) PSPACE: problem that can be solved using a reasonable amount of memory without regard to how much time the solution takes is known as PSPACE problem.

4) EXPTIME: These are the problems that can be solved in exponential time.

- This class contains most problems we are likely to run into including everthing in the previous three classes.

5) Undecidable :-

- For some problem, we can prove that there is no algorithm that always solves them, no matter how much time or space is allowed. These are called the undecidable problem



Relationship among P, NP, NP-Complete, NP-hard.

NP-Hard

- If a language L satisfies property 2 of NP Complete, but not necessarily property 1, we say that L is NP-hard.
- It means that there is a problem X such that every problem is NP reduces to X, then X is atleast as hard as every problem in NP. we say that X is hard for NP or NP-hard.
- Once we have one problem that is NP-hard we can use it to find others, for example by reducing X to Y, to show that Y is also NP-hard.
- Consider a problem X in NP-hard if there is an NP-complete problem Y that can be polynomially turing reduced to it: $Y \leq_T^P X$. By definations of polynomial reductions any polynomial time algo. for X would translate into one for Y. Since Y is NP accepted belo
- Therefore no NP-hard problem can be solved in polynomial time in worst case under the assumption that $P \neq NP$.

NP-Complete.

- If X is in NP and is also NP-hard we say that X is complete for NP or is NP-complete
- If any NP-hard problem is in P that every problem reduces to it is also in P.
- There are problem that are NP-hard but not known to be in NP. So we usually restrict ourselves to NP-complete problem.
- All NP-complete problem have the same hardness.
- If any NP-complete problem is in P then P = NP
- Polynomial time reductions provide a formal means for showing that one problem is atleast as hard as another, to within polynomial time factor. That is if $L_1 \leq_P^L L_2$, then L_1 is not more than a polynomial factor harder than L_2, which is why the "less than or equal to" notation for reduction NP-mnemonics
- We can now define the set of NP-complete languages, which are hardest problem in NP as follows:

A language $L \subseteq \{0,1\}^*$ is NP-complete if it satisfies the following two properties.

i) $L \in NP$;

ii) For every $L' \in NP$, $L' \leq_P^L$

- If a languages L satisfies property 2, but not necessarily property 1, we say that L is NP hard.

- We use the notation $L \in NPC$ to denote, that is NP-complete.

* Difference Between

1) NP-Hard , NP-Complete.

NP-hard	NP-complete
1) The input is NP-hard problem is undefined	1) The input is NP-complete problem is not fixed
2) The process is undefined	2) The process is not fixed
3) Time is unknow	3) Time is not fixed
4) Not all NP-hard problems are NP-complete	4) All NP-complete problems are NP-hard.
5) NP-hard problem that is in a certain sense are atleast as difficult to solve as any other NP-problem	5) NP-complete problems are toughest problems that are in NP

ii) P and NP

P	NP
1) In polynomial time the input is fixed.	1) In NP the input is not fixed.
2) The process is fixed	2) The process is not fixed
3) Time is fixed	3) Time is not fixed
4) P problems can be efficiently solve	4) NP problem which, given a proposed soln, we can efficiently check if it works
5) eg. Prime no. is relatively easy problem to solve.	5) eg. Travelling Saleman problem

iii) Deterministic polynomial time & Non-deterministic.

Deterministic	Non-deterministic
1) The algo. where result is uniquely defined is terme as the deterministic algo.	1) The algo. which terminates unsuccessfully if and only if there exists no set of choic leading to a success single is known as non-deterministic algo.

2) The deterministic algo. given input will always produce same input	2) Algo. in non-deterministic if there are more than one path the algo. take Due to this one cannot determine the next state of machine.
3) eg mathematical func. is deterministic.	3) eg Random func is non-deterministic.
4) It has a unique value	4) It has diff^{n} values
5) It has single outcome	5) It has multiple outcome.

iv) Decision Problem and optimization Problem

Decision Problem	Optimization Problem
1) Any problem for which the answer is either zero or one/true or false is called decision problem.	1) Any problem that involves identification of an optimal value of given cost function is known as Optimization Problem.
2) Decision problem works as to check if Something is true. i.e. 'yes' or 'No'.	2) An optimization prob works as to find among all feasible soln, are that maximized or minimizes a given objec

3) eg Prime no.	3) eg Single source shortest path
4) If we provide evidence that a decision problem is hard, we also provide evidence that its related optimization problem is hard	4) If an optimization problem is easy then its related decision problem is easy as well
5) Decision problem is easier than optimization problem	5) optimization problem is harder than decision problem

→ Non-deterministic Algo.

Non-deterministic Sorting

- Select n position from the given array
- Create a temporary array using the elements present at the selected position.
- Checks the temporary array for sorted sequence.

Algorithm

```
Algorithm ND-Sorting (a,n)
{
   for i = 1 to n do
   {
      j = select (a,n)
      a[i] = a[j];
   }
   for i = 1 to n-1 do
   {
      if (b[i] > b[i+1] then
         failure ();
   }
   if (i=n)
      Success();
}
```

Complexity : $\theta(n)$

GOODLUCK | Page No.
Date

What is efficient certification.

- Certification algo. means "Design an algorithm that check whether proposed solution is a yes instance."
- Any algorithm C is an efficient certifier to X if:

i) C is a polynomial time algorithm that takes two input s and t.

ii) There exists a polynomial P(). So that every string s, $s \in X$ such that there exists a string t such that $|t| \leq P(|s|)$ and $C(s,t) = yes$

Example.

COMPOSITE - Given integer s, is composite?

Observation - s is composite.

- It is implied that there exists an integer $1 < t < s$ such that s is a implied of t
- Yes instance : s = 437,669
  certificate : t = 541 or 809
- No instance : s = 437,677

No witness can fool verifier into saying yes.

input s:	certificate t:
437,669	541

certifier:

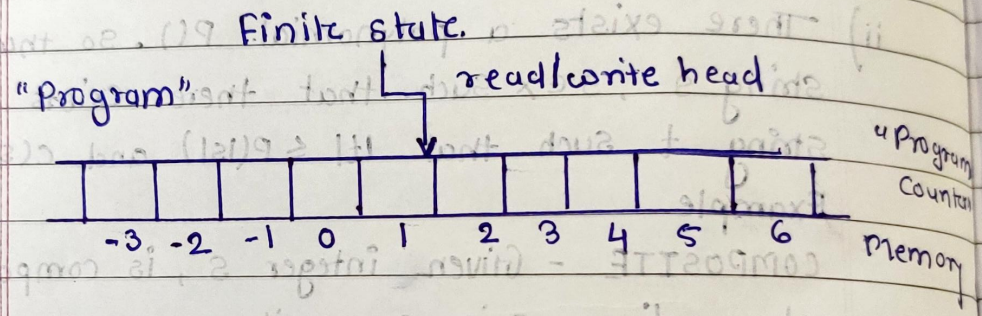
Is s is a multiple of t?

- s is a YES instance
- No conclusion

- Therefore now we have conclusion that COMPOSITE $\in$ NP.

Cook's Theorem

- Cook's Theorem proves that Satisfiability is NP-complete by reducing all non-deterministic turing machine to SAT.
- Each turing machine has access to a two way infinite tape (read/write) and a finite state control, which serves as a program.



A program for non-deterministic TM is:

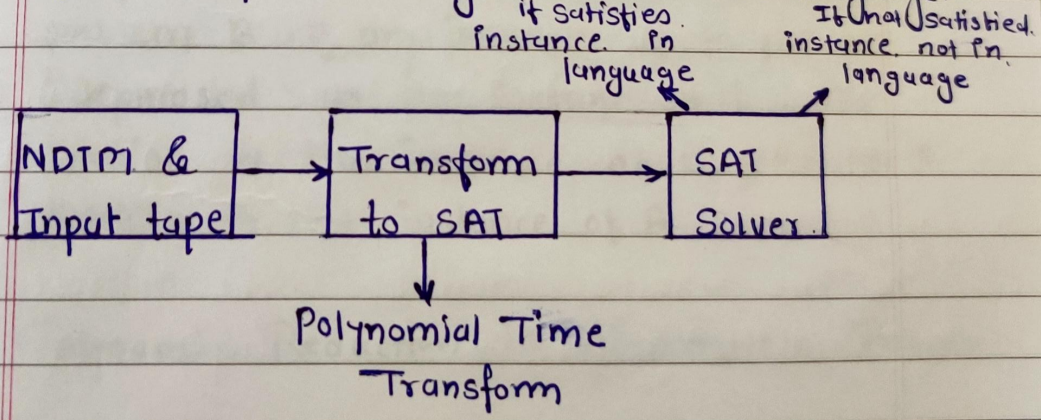
a) Space on the tape for guessing a solution and certificate to permit verification.
b) A finite set of tape symbol.
c) A finite set of states Q for the machine, including the start state q_0 and final state z_{yes}, z_{no}.
d) A transition function, which takes the current machine state and current tape.
- Symbol and returns the new state, symbol and head position.

- We know a problem is in NP if we have a NDTM program to solve it in worst case time $P(n)$, where P is a polynomial and n is the size of the input.

"Cook's theorem Satisfiability is NP-complete"

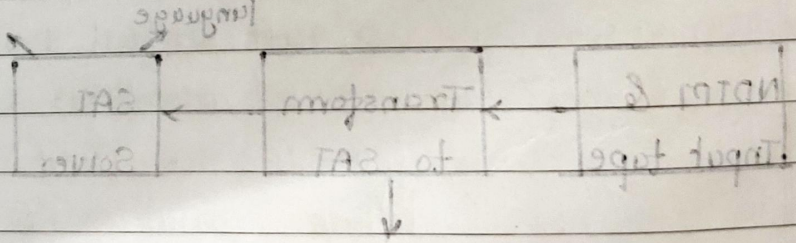
Proof:

- We must show that any problem in NP is atleast as hard as SAT.
- Any problem in NP has a non deterministic TM program which solves it in polynomial time, specifically P(n).
- We will take this program and create from it an instance of satisfiability if and only if the input string was in the language.



- If a polynomial time transform exist, then SAT must be NP-complete, since a polynomial solution to SAT gives a polynomial time algo. to anything in NP.

- Since a polynomial time algo. for SAT would imply a polynomial time algo. for everything in NP, SAT is in NP-hard since we can guess a solution to SAT, it is in NP and thus NP-complete. Therfore, now Cook's theorem has been proved.



GOODLUCK | Page No.
Date

Polynomial Reduction

- Let A and B be two problems, we say that A is a polynomial ~~turing~~ turning reducible to B if there ~~is~~ A exists an algo. for sorting A in a time that would be polynomial if we could solve arbitary instance of problem B at unit cost. This is denoted as

$$A_T = <^P_B$$

- Note that if $A_T = <^P_B$ and $B_T = <^P_A$ then $A == ^P B_T$ is called polynomial turning equivalent and polynomial reduction are transitive. if:

$A_T = <^P_B$ and $B_T = <^P_A$

$A_T = <^P_C$

- A problem A can be reduced to another problem B if any instance of A can be "rephrased" as an instance of B, the solution to the instance of B provides a solution to the instance of A.

Polynomial Reduction in NP-Complete Problem

- Let L and L' be two languages and $L' \in$ NP-complete. If $L' \leq P^L$. Then L is NP-hard. If an addition L is NP, then L is NP-complete.

- Let f be the polynomial time function that reduces L_1 and L_2 and A; a polynomial time algo. that decides L_2. Then for any x, run A on $y = f(x)$. If it accepts y then $x \in L_1$; otherwise $x \notin L_1$. Since the decision on x is done in polynomial time $L_1 \in P$.

★ clique Algo.

It is a part of graph which is considered as complete.

$G(V, E) \rightarrow [V_1 \subseteq V]$ such that $G(V_1)$ = complete

Assumption:

G represent a graph of n vertices and it is required to find out clique of k vertices

$n = 7$, $k = 3$ (size of clique)

condition: A

The vertices selected should represent complete graph <connectivity>

Method:

Using select function k vertices from the graph and generate a set of representing k vertices.

S = <2, 4, 2>

After selection test the connectivity for the vertices present in S.

```
Algo. clique (G, n : k, s)
{
    S = NULL;
    for i = 1 to k do
    {
        j = select (G, n)
        if (j ∉ S) then
            S = S ∪ {j};
    }
    for each pair of vertex (i, j) && (i = j) or
        (G [i, j] ≠ 0)
    failure ();
    Success ();
}
```

★ Deterministic and Non-deterministic Algo.

- The algo. where result is uniquely defined is termed as the deterministic algo. Such algorithm agree with the way programs are executed on a computer.
- In theoretical framework we can remove this restriction on the outcome of every operations whose outcomes are not uniquely defined but limited to specified set of possibilities.
- The machine executing such operations is allowed to choose anyone of the outcomes subject to termination condition.
- This leads to the concepts of a non-deterministic algo..
- To specify such algo. we introduce three new functions :

i) choose(s) : arbitarily chooses one of the elements of sets.

ii) failure() : signal a unsuccessful completion

iii) success() : ~~single~~ signals a successful completion

- A deterministic algo. terminates unsuccessfully if and only if there exists no set of choices leading to success signal

- The computing time for choice, success and failure are taken to be O(1)

Decision Problem

- Any problem for which the answer is either zero or one or equivalently either true or false is called as decision problem.
  eg. "Find a Hamiltonian cycle in graph G" is not decision problem; but is graph G is Hamiltonian?" is a decision problem
- The class NP - corresponds to the decision problem, that have an efficient proof system, which means that each yes instance must have atleast one certificate whose validity can be verified quickly
- The theory of NP - completeness is concerned with notion of polynomially verifiable properties.
- Intuitively, a decision problem X is polynomial time verifiable if some one could show that x is X wherever this is so.
- Given this eg. (x is X) one should be able to verify in polynomial time that indeed x is X However, if in fact x not in X, then one should not falsely show that x in X.

Page No.
Date

optimization Problem

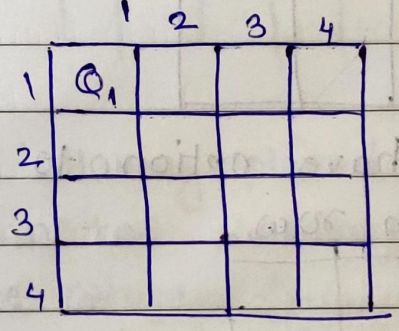
- Any problem that involves identification of an optimal value of given cost function known as optimization problem.
- An optimization algo. is used to solve an optimization problem.
- The problem in which some value must be minimized or maximized, optimization problems can be recost in terms of decision problem by placing a bound on value to be optimized.

eg. n recosting the shorted path problem as the decision

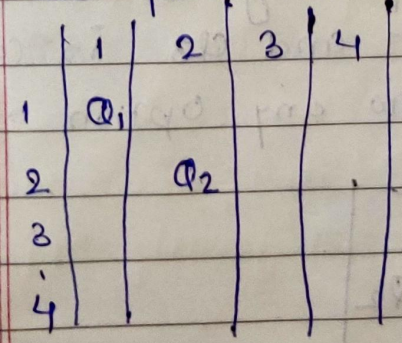
4-Queen Using Backtracking

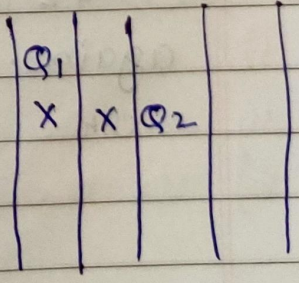
- Given 4x4 chess board and 4 queens to be placed on 4x4 chess board so that no two queen are on the same row, column or diagonal.
- Here the solution to the problem is 4-tuple where 4 queens are placed with the above constraints. Now we show the step by step solution of 4 queen problem.

Step 1 : placed first queen Q_1 in the first column.



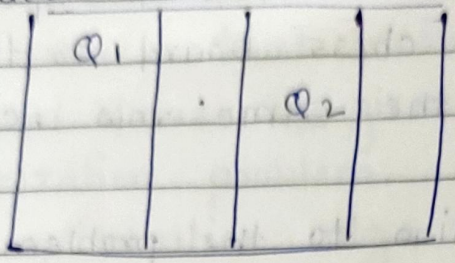
Step 2: After placing first queen in the first column we cannot placed the 2nd queen in the first or 2nd column so, we place Q_2 in the third column



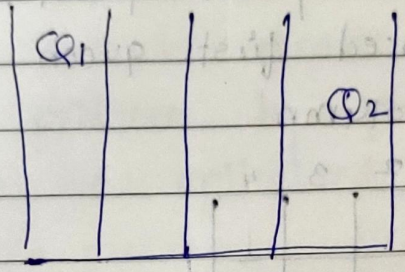


Cannot be placed Q_2 in column digonal

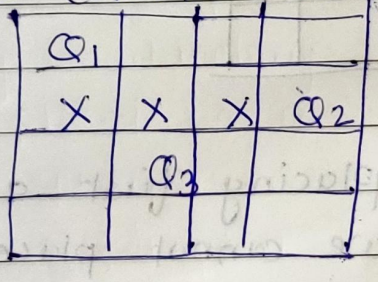
Step 3 : After placing the 1st and 2nd queen we cannot place Q_3 anywhere



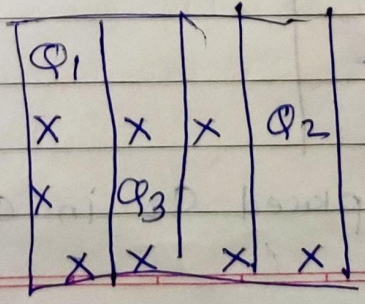
we cannot place Q_3 any where.
So, we apply another placement for Q_2 as shown in below.



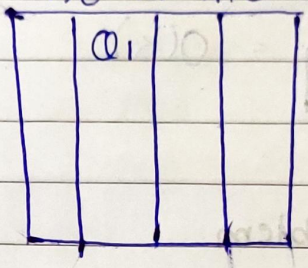
Now, we only have option to place Q_3 in the column now.



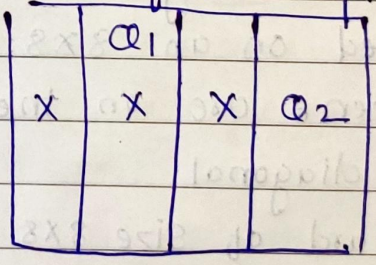
Step 4 : Here, after placing Q_1 in column 1, Q_2 in column 4 and Q_3 in column 2 again we have no any option to place Q_4



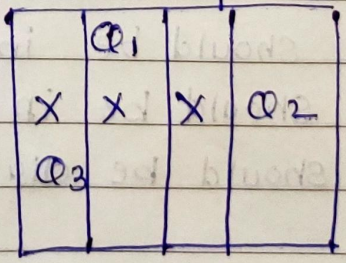
Step 5 : From step 4 it is noticable that we have no any option in change the position of Q_1, Q_2 and Q_3 for the placement of Q_4. Hence, we now have to make backtracking and examine whether there is any option for placement of the queen we now start with placement of the queen one in the column two



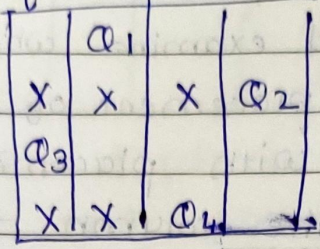
Step 6 : Having placed the queen one in the column two. we can place the queen two in the column four only.



Step 7 : Now, we place Q_3 in the col^n 1



Step 8 : Now, after placing queen one in column one two, queen two in column four, queen three in column one we can place queen four in column three



Time complexity = O(k)

★ 8-Queen Problem

- 8-Queen problem is a classic combinational problem. This problem can be solved by applying backtracking we can define 8-queens to be placed on an 8×8 chessboard so that no two queens are on the same row, column or diagonal
- A chess board of size 8×8 is given along with 8-Q's. It is required to place the Q's such that

i) No two Q's should be in same row.
ii) No two Q's should be in same column.
iii) No two Q's should be diagonally opposite.

```
Algorithm NQueen (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
            NQueen (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.
      }
  }
  ```

- One possible soln for 8 queens problem
  ```

Implicit constraints

1) No two Q's should be present in same row
2) No two Q's should be present in same col^n
3) No two Q's should be diagonally opposite.

Explicit constraints

Required to place the Q's on the chessboard. Solution space consist of 8^8 combinations possible solution.

	1	2	3	4	5	6	7	8
1				Q_1				
2						Q_2		
3								Q_3
4		Q_4						
5							Q_5	
6	Q_6							
7			Q_7					
8					Q_8			

The sequence of queen's are :

[4, 6, 8, 2, 7, 1, 3, 5]

	1	2	3	4	5	6	7	8
1		Q_1						
2				Q_2				
3						Q_3		
4								Q_4
5					Q_5			
6							Q_6	
7	Q_7							
8			Q_8					

The sequence of the Queen's are :
[2, 4, 6, 8, 5, 7, 1, 3]

formula : abs (i-k) = abs (k-L)
abs (k-i) = abs (j-i)

Complexity : O(k)

N-Queens Problem.

- N-Queens problem is to be place n-Queens in such a manner on an n*n chess board that no two queens attack each other by being in same row, column or diagonal.
- It can be seen that for n=1, the problem has a trivial solution and no solution exists for n=2 and n=3, so first we consider 4-queen problem and the generalized it to n-queen problem.
- N-queen problem is classic problem that is often is used in terms of standard 8*8 chessboard and can be solvable for N≥4
- In general, problem is to place the queen on the board so that no two queens attack each other.
- In an N- by - N board, each queen is located on exactly one row, one column and two diagonals.

★ Graph Coloring with suitable eg.

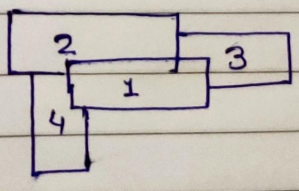
⇒ Graph coloring

- The main objective of graph colouring problem is to colour the given graph with minimum number of colors. such that two adjacent vertices will be in different colors.
- The number of colors to be used are fixed.
- It is required to color the graph such that no two adjacent vertices will be in same color.
- Number of colors used should not exceed the given value.

$$color(A) \neq color(B)$$

- If d = degree of graph the number of colors required = d+1.
- The number of colors required to color the graph is called as "chromatic number"

eg Coloring maps



Explicit constraints :

Fixed no. of vertices and colors.

Implicit constraints:
Adjacent vertices should be in different colors

Algorithm:

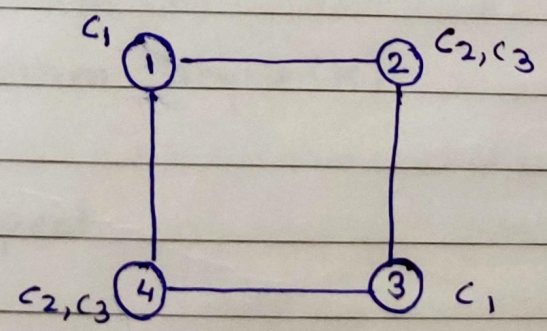
1) The graph is represented in the form of matrix. In this matrix if the edge is present then G[i, j]=1, otherwise =0.
2) Let variable 'm' represent no. of colors.
3) The algo. will ~~give~~ generate an array x[1...n] which will provide information about one solution initially all the values in x are 0.
4) The algo. uses a function NEXT VALUE(k) where k represents a vertex to be colored

```
Algorithm Gcolor(k)
{
   do
   {
      NEXTVALUE(k)
      if (x[k]=0) then
        Return
      if (k=n) then
         write(x[1...n])
      else
         Gcolor(k+1)
   }
   while(false)
}
```

Algorithm NEXTVALUE (k)

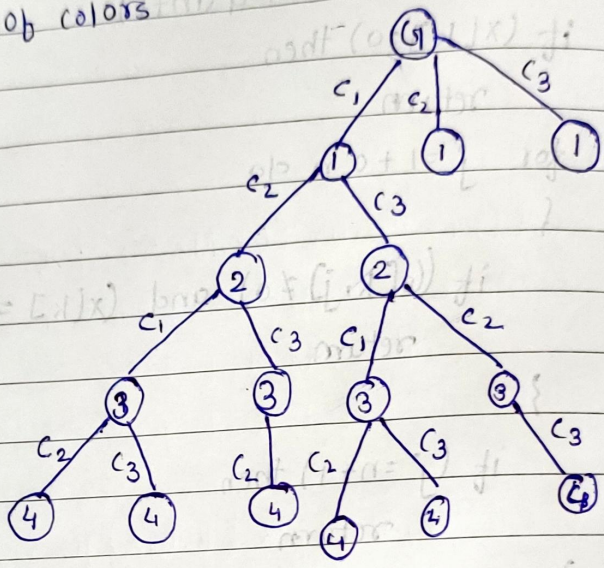
```
{
    do
    {
        x[k] = (x[k] + 1) mod (m+1)
        if (x[k] = 0) then
            return
        for j = 1 to n do
        {
            if (G[k,j] ≠ 0) and (x[k] = x[j]) then
                return
        }
        if (j = n+1) then
            return
    }
    while (false)
}
```

eg



Solution Space tree:

This tree will define all possible soln which can exists for the given graph & no. of colors



Total soln = $6 \times 3 = 18$

no. of colors = 3 (i.e. d+1)

Hence, degree of graph = 2

Hamiltonian Cycle.

- It is a cycle generated for the given graph with following characteristics:

1) All the vertices of the graph should be visited only one except the source vertex.
2) If the edge is used in the cycle then it can be repeated.

- The soln for this problem can be more than one. The algo. design will generate any one of the valid sequences from the given starting vertex.

Explicit concern: If n is given vertices in graph, it is not possible to visit more than n vertices

Implicit concern: If the vertex is already visited then it will be available in previously generated sequence and it cannot be used again

Algorithm :-

```
Algorithm Hcycle(k)
{
  repeat
  {
    nextvalue(k)
      if (x[k]>=0)
        if [k=n] then
          write x[1..n]
    else
```

```
        Hcycle (k+1)
    }
    until (false)
}
Algorithm nextvalue(k)
{
repeat
    {
      x[k] = [(x[k+1]) mod (n+1)]
      if (x[k] = 0) then return
      if (G[k-1, k] ≠ 0) then
      for j=1 to k-1 do
      {
        if (x[k] = x[j]) then
          breaks;
      }
    if (j=k) then return
      {
        if (k<n) then
          return
        if (k=n) then
        {
          if (∈[n,1] ≠ 0) then
        }
        return
        until (false)
      }
    }
}
```

GOODLUCK | Page No.
Date / /

Sum of Subset

Given an array [1...n] of positive integer and a variable 'm' sum of subset problem will find a sequence from the array consisting of distinct element such that sum of elements = m.

Explicit Constraint :

1) Only positive values can be used.
2) Only elements from the array can be used.

Implicit Constraint:

1) Element can be used only once.
2) If sequence starts with a[i] then it can be contain elements from index a[i+1]

```
Algorithm SumofSub(s, k, r)
{
    x[k] = 1;
    If (s + W[k] = m) then
        write (x[1: k]);
    else if (s + W[k] + w[k+1] ≤ m)
        then SumofSub(s + W[k], k+1, r - w[k]);
        if ((s + r - W[k] ≥ m) and (s + W[k+1] ≤ m)) then
        {
            x[k] = 0
            SumofSub(s, k+1, r - w[k]);
        }
}
```

Time and Space Complexity

- It is very convienient to classify algo. based on the relative amount of time or relative amount of space required and specify the growth of time/space requirements as a function of the input size.

1) Time complexity

Running time of the program as a function of the size of input is called as time complexity

2) Space complexity

- Space complexity means the memory required for the program to run to completion.
- The space complexity analysis was critical in the early day's of computing when storage space on a computer (both internal & external) was limited
- When considering space complexity algo. are divided into those that need extra space to do their work and those that work in place.
- It was that not unusual for programmers to choose an algo. that was slower just because it worked in place, because there was not enough extra memory for a faster algo

How coould you/we calculate running time of an algorithm?

- We can calculate the running time of an algorithm reliable by running the implementation of the algo. on a computer. Alternatively we can calculate the running time by using a technique called algo. analysis
- We can estimate an algo. performance by counting the no. of basic operation required by the algo. to process an input a certain size

Basic Operation:

- The time to complete a basic operation does not depend on the perticular values of its operands. So it takes a constant amount of time

eg. Arithmatic operation, Boolean operation, Comparison operation, Module operation, Branch operation

- Input size: It is the no. of input processed by the algo.
- eg. for sorting algo the input size is measured by no. of records to be sorted

Algorithm - Four distinct area of study.

⇒ Algorithm :

- An algo. is a set of rules for carrying out calculation either by hand or on machine.
- An algorithm is a sequence of computatio steps that transform the input into the output.
- An algo. is an abstraction of program to be executed on a physical machine

The four distinct area of study are as follows

1) To device an algo.

→ This is an activity where human intelligence is definately required ; some of the strategies used have a general applicability like dynamic divide and conqu

2) To express an algo.

→ An algorithm can be expressed in various ways such as flowchart, psedo-code program.

3) To validate an algo.

→ This means that a program satisfies its precise specification.

4)

5)

4) To analyze an algo.

→ This field of study is called analysis of algorithm. It is concerned with the amount, computer time & storage that is required by the algo.

5) To test a program

→ Testing a program consists of two parts - debugging and profiling. Profiling is a process of executing a correctly working program with the appriciate data sets

```
for k = p to r
    if L(i) ≤ R(j)
        A(k) = L(i)
        i = i+1
    else
        A(k) = R(j)
        i = j+1
```

Merge sort

15	10	5	20	25	30	40	35
p=1							r=8

q = 4
merge_sort (A, 1, 4)
merge_sort (A, 5, 8)
merge_ (A, 1, 4, 8)

→ merge_sort (A, 1, 4)

15	10	5	20

q = 2
merge sort (A, 1, 2)
merge sort (A, 3, 4)
merge_ (A, 2, 4)

→ merge_sort (A, 1, 2)

15	10
1	2

q = 1
merge sort (A, 1, 1)
merge sort (A, 2, 2)
merge sort (A, 1, 2, 2)

merge sort (A, 5, 8)
q = 6
merge sort (A, 5, 6)
merge sort (A, 7, 8)
merge (A, 5, 6, 8)

↳ merge sort (A, 5, 6)

q = 5
merge sort (A, 5, 6)
merge sort (A, 6, 6)
merge (A, 5, 5, 6)
— ×

↳ merge sort (A, 5, 5)
↳ merge sort (A, 6, 6)
↳ merge (A, 5, 5, 6)

| 25 | ∞ | | 30 | ∞ |

A(K) = | 25 | 30 |

| 25 | 30 |

merge sort (A, 7, 8)
q = 7
merge_sort (A, 7, 7)
merge_sort (A, 8, 8)
merge (A, 7, 7, 8)

↳ merge_sort (A, 7, 7)
↳ merge_sort (A, 8, 8)
↳ merge (A, 7, 7, 8)

| 40 | 35 |

40	∞		35	∞

A(K) = | 35 | 40 |
|---|---|

merge (A, 5, 6, 8)

25	30	∞		35	40	∞

A(K) =	25	30	35	40

merge (A, 1, 4, 8)

5	10	15	20	∞		25	30	35	40	∞

A(K) =	5	10	15	20	25	30	35	40

merge sort (A,1,1)
merge sort (A,2,2)
merge (A,1,2,2)

15	∞

10	∞

A(K) = | 10 | 15 |

merge sort (A,3,4)

5	20
3	4

q = 3
merge sort (A,3,3)
merge sort (A,4,4)
merge (A,3,3,4)

→ merge sort (A,3,3)
→ merge sort (A,4,4)
→ merge (A,3,3,4)

5	20

5	∞

20	∞

A(K) = | 5 | 20 |

merge (A,1,2,4)

10	15	∞

5	20	∞

A(K) = | 5 | 10 | 15 | 20 |

GOODLUCK | Page No.
Date

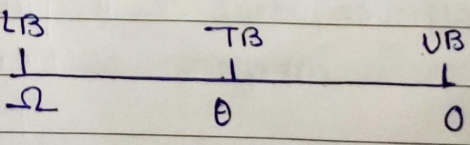
Asymptotic Notations.

- Asymptotic notations are used to find out three phase complexity of a function.
- The asymptotic running time of an algo. is defined in terms of function.
- This functions are set of natural no. and real no.
- A way of comparing functions that ignores constant factors and small input size is known as asymptotic notation.
- These are three types of asymptotic notation.

1) Theta Notation (θ)
2) Omega Notation (Ω)
3) Big Oh Notation (O)

- The complexity of any algo. is based upon the following ~~size~~ three classes.

i) Best case.
ii) Worst case.
iii) Average case.



1) Theta Notation (Θ)

→ • This notation is also called as tight bound.

• The function $f(n) = \Theta(g(n))$ if there exists positive constants c_1, c_2 and n_0 such that

• The function $f(n)$ is sandwitched betn $c_1 g(n)$ and $c_2 g(n)$ Since $\Theta(g(n))$ is a set and $f(n) \in \Theta(g(n))$ we can write:

$$\boxed{f(n) = \Theta(g(n))} \quad \forall \; n > n_0$$

2) Omega Notation (Ω)

→ • This notation is called as lower bound.

• The function $f(n) = \Omega(g(n))$ is called as lower bound of 'g' if there exists a positive constant c and n_0 such that

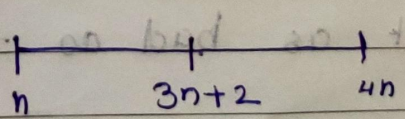
$$\boxed{f(n) \geq g(n) * c} \quad \text{for} \quad n \geq n_0$$

• To calculate lower bound the value closest to $f(n)$ should be selected. This is possible by ignoring the time for singular instruction involved in $f(n)$

eg $f(n) = 3n + 2$

Here, $f(n) = 3n + 2$

$g(n) = 3n$



3) Big-oh Notation

→ The function $f(n) = O(g(n))$ is called as big oh g if there exists a positive constant 'c' and no such that

$$f(n) <= g(n) * c$$

* Best case & Average Case Analysis

- Worst case performance analysis and average case performance analysis have similarities but usually require different tools and approaches in practice.
- Determining what average input means is difficult and often that average input has properties which makes it difficult to characterize mathematically.
- When a sensible discription of a particular "average case" is possible they tend to result in more difficult to analyse equations.
- Worst case analysis has similar problems. typically it is impossible to determine the exact worst case scenario.
- Instead, a scenario is considered which is atleast as bad as the worst case.

eg When analyzing an algorithm it may be possible to find the longest possible path through the algo. even if it is not possible to determine the exact input that could generate this indeed, such an input way may not exist. This lead to safe analysis, but which is pessimistic.

- When analyzing algo. which often take a small time to complete but periodically require a much larger time, amorized analysis can be used to determine the worst curse running time over a series of operation.
- The other thing that has to be decided when making these consideration is whether what is of intrest is the averaye performance of a program or whether it is important to guarantee that even in the worst case, performance obeys certain rules