Mod-4

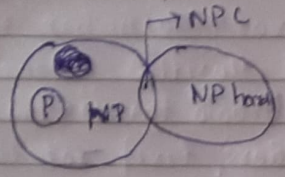
Date __/__/____

Saathi

Complexity Classes

Based on degree of difficulty, problems are classified in 4 classes

1. P-class
2. NP-class
3. NP-hard
4. NP-Complete



P class

Class P problems are solvable in polynomial time

Time Complexity → $O(n^k)$

n → i/p size

k → constant

P is a class of problem accepted by a deterministic time.

~~NP Class~~ eg:- Linear Search — n
Binary Search — log n
Quick Sort — n^2
Merge Sort — n log n

NP Class

Problems that can be solved in non deterministic polynomial time is called NP class problem.

NP class problem are solvable in exponential time, not solvable in polynomial time.

Every P class problem is NP class
Every NP class problem is not P class

Date __/__/____

Saathi

- All P class problems are subset of NP
- P class problem can be solved efficiently, NP class problem cannot be solved efficiently as P-class Problems

eg:- Hamiltonian Path problem
Graph Coloring

NP - hard

If there is no solution for problem A and if A can be converted to another problem B, which is with a polynomial time, and which is solvable, then this process is called reduction

NP-hard class problems are not-solvable in polynomial time and but can be reduced to another problem solvable in polynomial time.

eg:- Halting Problem
Qualified Boolean Formulas

NP-Complete

- A problem is NP complete if it is both NP hard and NP ~~complete~~
- NP-Complete problems are the hardest problems in NP
- A problem L is NP-Complete if,
  1. L is in NP
  2. Every L_2 in NP is polynomial time reducible to L

eg:- Vertex Cover
Clique Problem.

Saathi

Date / /

Polynomial time Reduction

- Solving one problem using another
- When a problem A is polynomial time reducible to B, it means that for given instance of A, there is an algorithm for transforming instances of A into instances of B

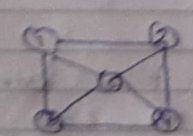
- Polynomial - time reduction proves that the first problem is no more difficult than the second one, because whenever an efficient algorithm exists for second problem, one exits for first problems also.

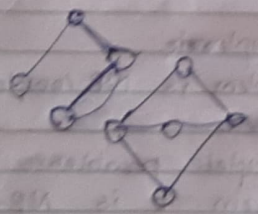
If a problem A is reducable to B, then we denote it as $A \propto B$

Clique

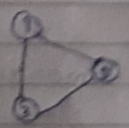
- For a graph $G(V, E)$, Clique is a complete subgraph of G.
- The size of Clique is the number of vertices it contains

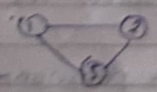
G →





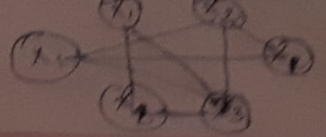
Clique with the size 2





CLIQUE = { $\langle G, k \rangle$: G is a Graph containing a Clique of size k }

Page No.



Saathi

Date / /

- Show that Clique is a NP complete.

→ The clique problem is an optimization problem of finding a clique of maximum size in a graph

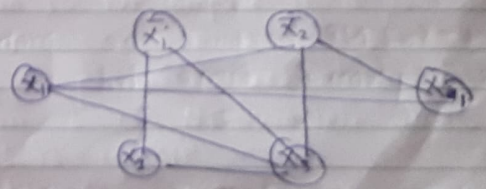
→ As a decision problem, we ask simply whether a clique of a given size k exists in the graph.

→ NP Complete def

→ ~~If a problem is NP means which~~.
→ Clique should belong to NP
→ A NPC Complete problem can be reduced to clique decision problem.

SAT $\propto$ CDP
$L_1 \propto L_2$

Rule for constructing Clique:
(1) Vertex of same clause should not be connected
(2) We can connect vertex of one clause to vertex of another clause
(3) We should not connect vertex of one clause with negation vertex in another clase.



For above graph, maximum size of ~~cl~~ clique is 3
$\therefore k = 3$

Page No.

CNF → Conjuntive normal form

Saathi

Date __/__/____

$k = 3$ $\quad x_1 \quad x_2 \quad x_3$

$\quad\quad\quad 0 \quad\quad 1 \quad\quad 1$

$F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_3)$

$= (0 \vee 1) \wedge (1 \vee 0) \wedge (0 \vee 1)$

$= 1$

Hence we prove that if we take a CNF with k clause, the we can make a clique of size k.
Thus proved that Clique problem is NP Complete

Prove that Vertex Cover is NPC

- A vertex cover of a graph is a set of vertices that covers all the ~~eg~~ edges in the graph.
- The size of the vertex cover is the number of vertices in it.
- The vertex-cover problem is to find a ~~cov~~ vertex cover of minimum size in a given graph.
- VERTEX-COVER = $\{ \langle G, k \rangle$: Graph G has Vertex cover of size k $\}$
- Vertex cover is NP because the problem cannot be solved on polynomial time but is verified in polynomial time.

Proof :

1. Vertex cover problem is a NP
2. Clique reducable to vertex cover
3. Given an instance of clique, we will produce a graph $G(V, E)$ and integer k such that G has maximum clique of size k,

Page No.

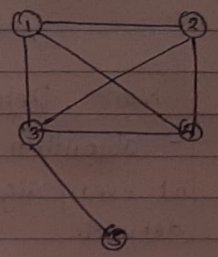
Date __/__/____

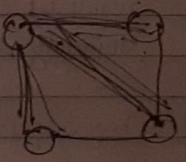
Saathi

if and only if
instance in $G'(V, E')$ has vertex cover of size $(v-k)$

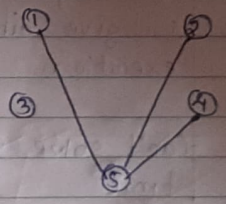
eg:

G →

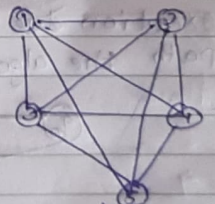




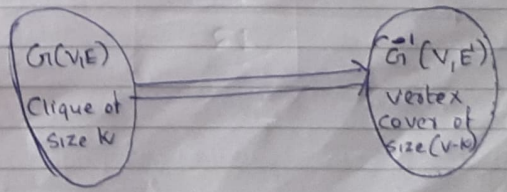
G' →



Merge G and G', the we get a complele graph



So $G(V, E) \cup G'(V, E')$ is a complete graph.

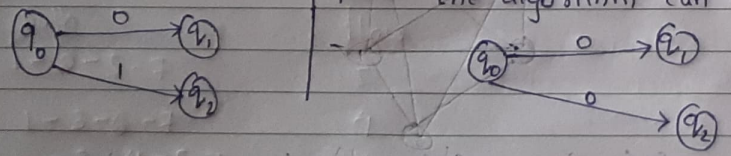


Let G has a clique of size k, that implies G' has vertex cover of size $(v-k)$

Page No.

Hence we prove that some of the instance of Clique can be reduced to vertex cover. Hence vertex cover is also NPC for that particular instance.

Deterministic Algo	Non Deterministic Algorithm
Algorithm in which the result of every algo is uniquely defined	- algorithm in which the result of every algorithm is not uniquely defined.
- For a particular input the computer will give always the same output	- For a particular input the computer will give different output on different execution.
Can solve problem in polynomial time	Can't solve the problem in polynomial time
Can determine the next step of execution	- Cannot determine the next step of execution due to more than one path the algorithm can take



Flow Networks

A flow network $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a non negative capacity $C(u, v) \geq 0$

Network flow

Let $G = (V, E)$ be a flow network. Let s be the source of the network, and let t be the sink. A flow in G is a real-valued function $f : V \times V \to R$ such that the following properties hold:

① Capacity constraint:

for all $u, v \in V$, we need $f(u, v) \leq C(u, v)$

② Skew Symmetry:

for all $u, v \in V$, we need $f(u, v) = -f(u, v)$

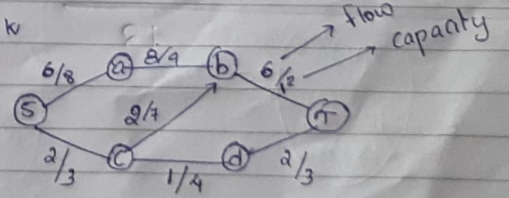
③ Flow Conservation:

For all $u \in V - \{s, t\}$ we require

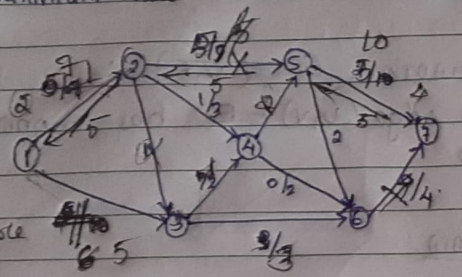
$$\sum_{v \in V} f(u, v) = \sum_{u \in V} f(u, v) = 0$$

In Maximum-flow problem, we are given a flow network G with source s and sink t, and we need to find a flow of maximum value from s to t.

Flow network

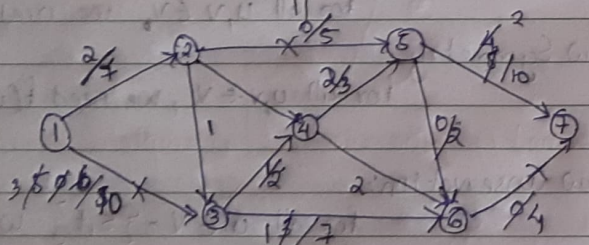


Q. Find the maximum flow



Ans.

Path	Flow
1-2-5-7	5
1-3-6-7	4
1-2-3-4-5-7	1
1-3-4-5-7	1
	11



Path	flow
1-2-5-7	5
1-3-6-7	4
1-3-4-5-7	1
1-3-6-5-7	2
	12

Page No.

Date

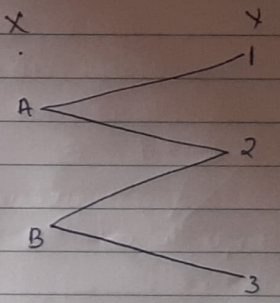
Saathi

Bipartite Graph

Graph $G = (V, E)$ in which vertex set V is divided into 2 disjoint sets X and Y. Every edge of graph has on endpoint in X and other endpoint in Y and vertices same set are not connected.

$X \cup Y = V$ and $X \cap Y = \phi$

example



Matching

No two edge can share common vertex

Page No.

Mod-5

Date

Approximation algorithm

An approximation algo is way of dealing with NP completeness for optimization problem. Goal of a approximation algo is to come close as possible to optimal solution in polynomial time

eg:-

1. all pair Shortest path problem
2. knapsack problem
3. Sum of Subset

} optimization problem

↓

objective is to maximize or minimize

In approximation algo,

$C \rightarrow$ cost of solution

$C^* \rightarrow$ cost of optimal soln

$P(n) / \rho(n) \rightarrow$ $\rho(n) \rightarrow$ determines approximation ratio

$n \rightarrow$ input size

we deal with two problems

1. maximation problem

$$\frac{C^*}{C} \leq \rho(n)$$

2. Minimization

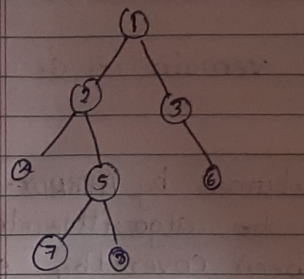
$$\frac{C}{C^*} \leq \rho(n) \quad \text{and} \quad \rho(n) > 1$$

Approximation ration gives the ratio between our solution (cost) and optimal solution (cost)

Saathi

Date

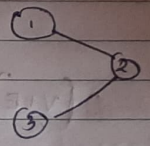
Vertex Cover Problem



$\{1, 2, 5, 3\}$

$\{5, 2, 3, 1\}$

$\{2, 5, 3\}$ → minimization



$\{1,2\}$ $\{3,2\}$ $\{1,3\}$

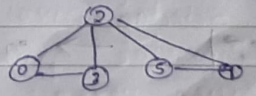
$\{2\}$ → mimization.

approximation algorithm for Vertex Cover

APPROX-VERTEX-COVER

1) $C = \phi$
2) $E' = G.E$
3) while $E' \neq \phi$
4) let (u,v) be an arbitary edge of E'
5) $C = C \cup \{u, v\}$
6) remove every edges incident on either u or v from E'
7) return C

2-approximation algo for vertex covers



Let A be the set of edges chosen by the algorithm

$|A| = 2$

The minimum lower bound have only 2 end points.

Step 1: pick any edge $\{u, v\}$ from edges of Graph G
" 2: Add both u & v in S
3:- delete u and v and all its incident edges from G
④ Repeat until any edges remain in G

set C The set C of vertices that is returned by APPROX-VERTEX COVER is a vertex cover, since the algorithm loops until every edge in G.E has been covered by some vertex in C

time Time complexity of vertex cover = $O(V+E)$, and so it is polynomial.

size It remains to be shown that the solution is no more than twice the size of the optimal cover. We will do so by finding a lower bound on the optimal solution C^*

Set - Let A be the set of edges chosen in line 4 of the algorithm APPROX-VERTEX COVER algorithm.
- Any vertex cover must cover atleast one endpoint of every edge.
- No two edges in A share a vertex (algorithm) so in order to cover A, the optimal solution C^* must have atleast be atleast :-

$$|A| \leq |C^*| \quad \rightarrow \text{eq } ①$$

Since execution of line 4 of algorithm picks an edge for which neither endpoint is yet in C, and adds the two vertices to C, we know that.

$$|C| = 2|A| \quad \rightarrow \text{eq } ②$$

From eq 1 and eq 2

$$|C| \leq 2|C^*| \quad \rightarrow \text{approximation ratio}$$

By relating the size of the solution returned to lower bound, we obtain ours
That is, |C| cannot be larger than twice the optimal, so APPROX-VERTEX-COVER is a polynomial time 2-approximation algorithm.

Schwartz - Zippel Lemma and Polynomial Identity Testing

Given a polynomial $P(x_1, x_2 \cdots x_n)$
1. Fix a finite set, S
2. Choose n values, $r_1 \cdots r_n$, independently and uniformly at random from S
3. Evaluate $P(r_1 \cdots r_n)$ If $P(r_1 \cdots r_n) = 0$ output "P = 0" otherwise return "$P \neq 0$"

eg: $P(x) = x^2 - 1$
$n = 1$ (no. of variables)
$d = 2$ (degree)
Probability $[P(1) = 0) \leq \frac{d}{|S|} \Rightarrow \frac{1}{6} \leq \frac{2}{6}$

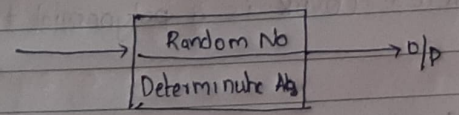
$P(x) = x^4 - 1$
$n = 1 \quad d = 4$
Probability $[P(1) = 0) \leq \frac{d}{|S|} \rightarrow \frac{1}{6} \leq \frac{4}{6}$

Saathi

Date __/__/____

Randomised Algorithm

- An algorithm that uses random numbers to decide what to do next any where in its logic is called randomised algorithm.
- It is used to reduce time and space complexity



In this algorithm output depends on selection of I/p random no.

Two types of randomised algo
1. Las Vegas Algo
2. Monte Carlo Algo

In Las Vegas, o/p is always correct
eg randomized Quick sort.

Randomized Quick Sort

Best case time complexity of Quick sort = $O(n \log n)$

Worst " " " = $O(n^2)$.

eg: 3 10 9 8 2

In the normal quick sort we have to select pivot as the last element in the list and after the 1st iteration the selected pivot get correct position and the value less than the pivot become the left side of pivot and vice versa.

The new quick sort called Randomized-partion in place of partition ←

After 1st iteration

1 2 3 10 9 8

The problem is one side becomes small and second side becomes big. It make the running time of quick sort into worst case condition. In order to avoid this condition we select pivot as random number in every iteration.

When partition is fair or median, we will get best case $O(n \log n)$

If pivot is closer to the median of I/p. There is a chance of achieving best case complexity.

Introducing randomness may achieve better results, so
(a) input can be shuffled
(b) Choice of pivot can be done randomly everytime

Algorithm (Randomization algorithm)

```
Randomized_partion (A,P,r)
    i = Random (P,r) // picking any random numbers
② Exchange A[r] with a[i] ③ return Partition(A,P,r)
                          // exchanging r & make it pivot
Randomized_Quick sort (A,P,r)
① if P<r
    ② q = Randomized_partion (A,P,r)
③ Randomized_Quick Sort (A,P,q-1)
④ Randomized_Quick Sort (A,q+1,r)
```

Saathi

Date __/__/____

Rounding algorithms

The process of converting fractional solution $\{0,1\}$ to integers $\{0,1\}$ is called Rounding

Saathi

Date __/__/____