TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu

2022

Assignment: III

Algorithm and Complexity

Submitted By:

Sudhan Kandel

Semester: 1^{st}

Roll no: 2

Submitted To:

MR. Sarbin Sayami

~~A Mathe~~ Complexity theory.

→ A Mathematic problem is computable if it can be solved in principle by a computing device. Some common synomyms for "computable" are "solvable", "decidable", & "recursive"

→ There are extensive study and classification of which Mathematical problem are computable and which are not

→ There is an extensive classification of computable problems into computational complexity classes. according to how much computational - as a function of the size of problem instance is needed to answer that instance.

→ ~~Computer~~ Complexity theory concerned with the resources, such as time and space, needed to solve computational problem.

→ There is an abundance of problems from mathematic & computerscience that are trivially solvable by brute force search of an exponential number of instance, but for which currently no efficient algorith is know.

→ Complexity theory is the appropriate setting for the study of such problems.

~~1. Time complexity :- How long computation~~

* Complexity Classes.

→ Complexity classes help computer scientists groups problems based on how much time and space required to solve the problem and verify solutions. For example complexity can help to decide how many steps it would take a Turing Machine to decide a problem A. A big-O notation is necessary to understand the complexity classes.

Some complexity classes are subset of others. For example, the class of problem solve in deterministic polynomial time P, is subset of the class of problems solvable in nondeterministic polynomial time NP.

The time complexity of an algorithm is usually used when describing the number of steps it needs to solve the problem but it also be used to describe how long it takes to verify an answer.

The space complexity describe the how much memory that the algorithm required to solve the problems. In terms of Turing machines, the space need to solve the problem related to the number of space on the turing machine's tape it need to do the problem.

A ~~computat~~ complexity class is the set of all of the computational problems which can be solved using a certain amount of a certain computational resources.

1. P - Class

-> The class P contains decision problems (Problems with a yes or No answer) that are solved by deterministic Turing machine in Polynomial time.

-> Many problems can be solved using a brute-force approach, but this is often take ~~poly~~ exponential time. If a problem has a polynomial time algorithm, it can be solve much more efficiently than by the brute force method. Many of the algorithm that are actually used in practic are polynomial algorithms. exponential time algorithms are typically less useful and are avoid as much as possible.

-> The class P consists of those problems that are solvable in polynomial time ie these problem can be solved in time $O(n^k)$ in worst-case, where k is constant.

-> These problems are called tractable.

- Formaly, an algorithm is polynomial time algorithm, if there exists a polynomial $p(n)$ such that the algorithm can solve any instance of size n in a time $O(p(n))$.

problems in class P.

1. finding if numbers are relatively prime.
2. Maximum matching
3. Linear programming.

2. NP- Class.

-> The Class NP contains decision problems that are solvable by turing machine in non-deterministic polynomial time-This include the problem that are solvable in polynomial time up to problems that are solvable in exponential time.

-> while they can take a long time to solve, problem in NP Can be verified by a Turing machine in polynomial time. This means that if given a "yes" answer to an NP-problem you can check that it is right in polynomial time. This "yes" answer is often called a witness or a certificate.

-> For example, Factoring large number in a NP problem. If you are given a large number and asked what it's prime factorization is, it will probably take you a long time to do this. However, if someone claims they know the prime factorization for a given large number it's really easy to verify this.

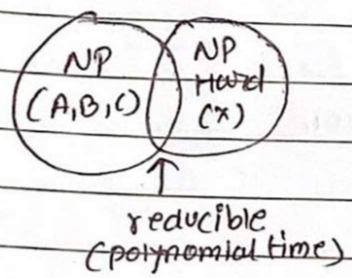
-> problem in NP

1. The travelling salesman problem
2. Graph coloring
3. Graph isomorphism.

3. NP-Hard.

→ If a particular problem is as hard as the hardest problem is NP-class then we will say that this problem is NP-hard.

→ A problem is NP-hard if every problem in NP can be polynomially reduced to it.



- If we find the solution of NP-hard problem, then we get the solution of all the problem. These are the hardest type problem.

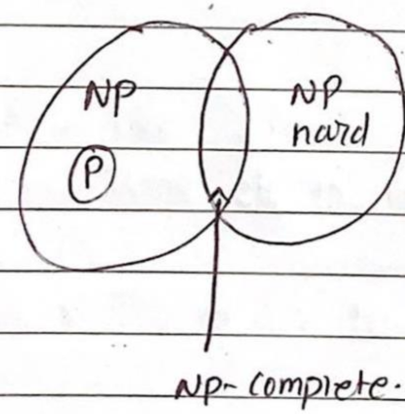
Eg:- Travelling salesman problem (TSP)

4. NP-complete.

→ Decision problem.

→ NP and NP-hard.

Relationship between P, NP, NP-hard and NP-complete.



The ~~theo~~ proof of the theorem consist two step.

1. Convert the execution of a polynomial time non Deterministic turing machine to a bunch of well formed formula such that formula satisfied iff the machine accept input.

2. Show the sum of lengths of formulae is polynomial in the size of problem.

Steps

1. If a language is NP-Hard. can polynomially reduce any NP problem to L.

2. If it's NP-complete then $L \in NP$

3. $L \in NP$ Then Non deterministic Turing Machine for L that runs in polynomial time

4. An Non-deterministic Turing machine is only Model we have for NP problem so that $SAT \in NP$.

5. Therefore, if we can polynomially reduce. and an arbitrary polynomial NDTM to SAT. it mean we have Proven SAT is NP-complete.

* Cooks Theorem

→ Theorem :- The ~~satisbilit~~ satisfiability problem (SAT) is NP-Complete

What is SAT?

→ A propositional logic formula ϕ is called satisfiable if there is some assignment to its variables that makes it evaluate to true.

→ $p \wedge q$ is satisfiable if $p = 1; q = 1$

→ $p \wedge \neg p$ is not satisfiable.

→ Boolean satisfiability or simply SAT is the problem of determining if a boolean formula is satisfiable or unsatisfiable.

→ The study of Boolean functions generally is concerned with the set of true assignments that make the function true.

Example

$(x + y)(-x + y)$ is satisfiable.

There are, in fact, two satisfying truth assignments:

1. $x = 0 ; y = 1$
2. $x = 1 ; y = 0$

$x(-x)$ is not satisfiable.

* Clique

→ In the field of computer science, the clique decision problem is a kind of computational problem for finding the cliques or the subsets of the vertices which when all of them are adjacent to each other are ~~set~~ also called complete subgraphs

The clique decision problem has many formulations based on which the cliques and about the cliques the information should be found. There are some common formulations based on which the cliques are based such as finding the maximom clique, finding the maximum weight of the clique in weighted graph, then listing all the maximum or maximal cliques and finally solving the Problem based on the decision of testing whether the graph has the larger cliques than that of the given size

Maximum clique: A particular clique that has the largest Possible number of vertices.

Maximal cliques:- The cliques Which further cannot be enlarged.

Application of clique decision problem.

1. clique decision finding problems algorithms are most abundantly used in chemistry to find the chemical which have a match with a target structure.

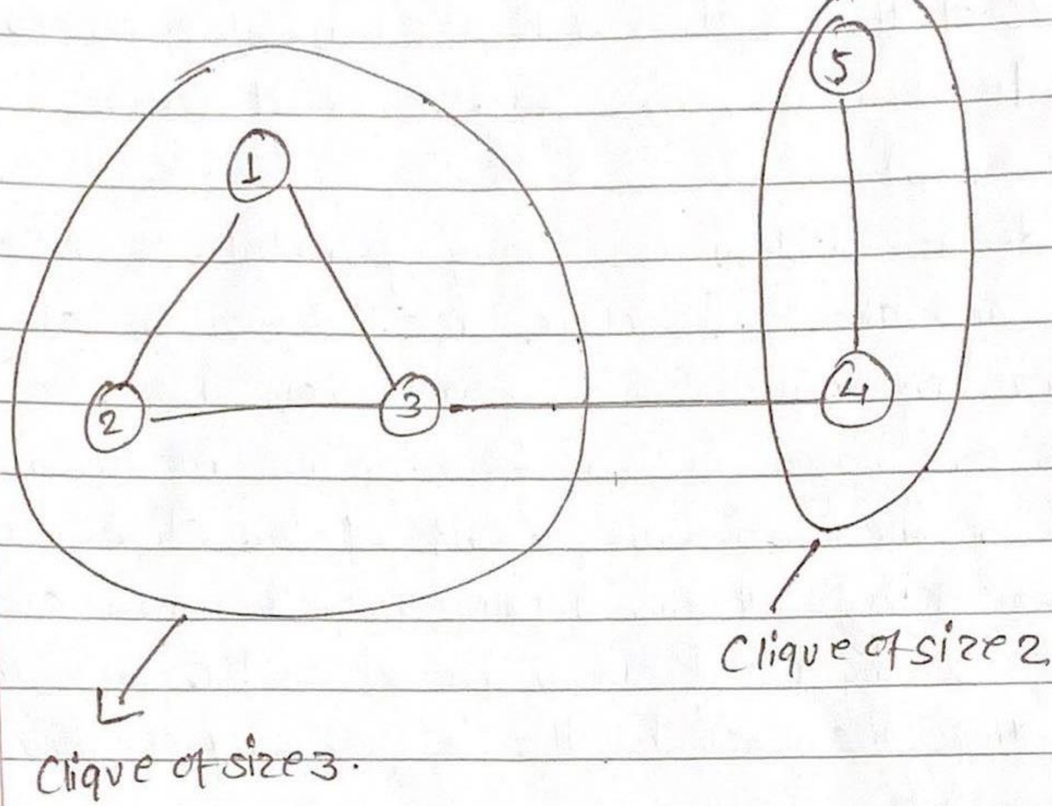
2. It can be used in automatic test pattern generation, where

ample : Boolean satisfiability problem.
This is the first proved by the cook and levin.

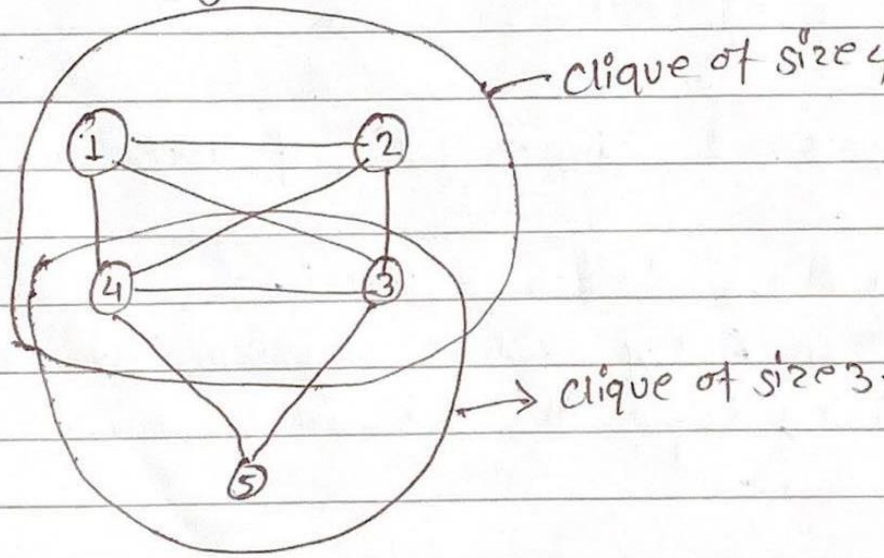
1. $(x+y)(-x+y)$ is satisfiable.
There are two satisfying truth assignment
a. $x=0$: $y=1$
b. $x=1$: $y=0$

2. $x(-x)$ is not satisfiable because it doesnot have any satisfiable input of x for $x(-x)$ form.

finding the number of cliques can also help to bound the size of a test case or set.



The above figure contains a maximum clique of size 3.



The above graph Contains a maximum Clique of size 4.

* Vertex cover.

→ An approximation algorithm is way of dealing with NP-completeness for optimization problem. The goal of Approximation Algorithm is come close as possible to optimal Solution in polynomial time.

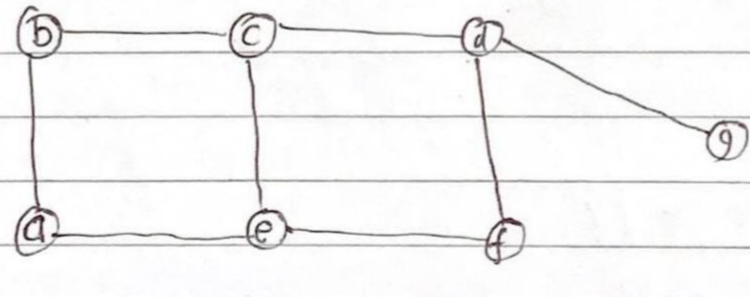
A vertex cover of a graph is a subset of vertices which cover every edges. Given a graph $G = (V, E)$ and an integer k is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S.

Algorithm

Approx cover-Prob(G)

1. $C \leftarrow \phi$
2. $E' \leftarrow E[G]$
3. while $E' \neq \phi$
4. do let (u, v) an arbitary edge in E'
5. $C \leftarrow C \cup \{u, v\}$ Remove from E' every edge incident either u or v
6. return C.

Example



Sol^n,

$$C = \phi$$

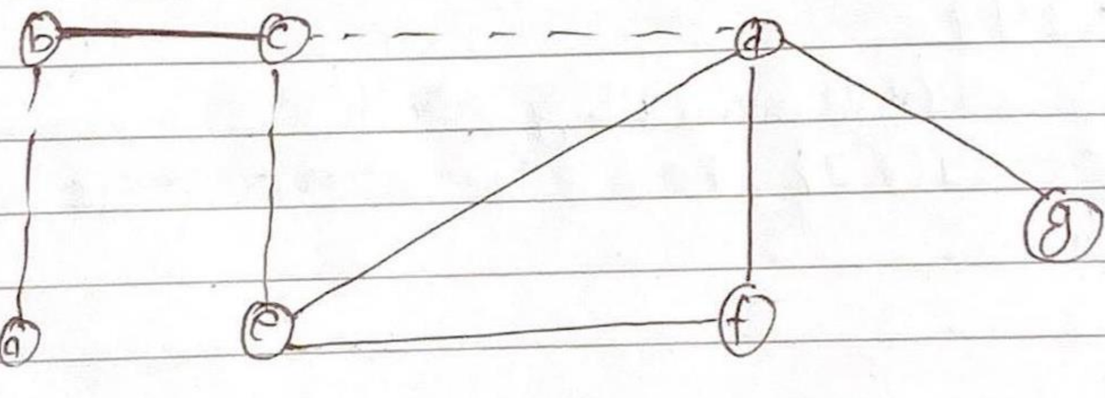
$$E' = \{(a, b), (b, c), (c, d), (c, e), (e, f), (d, f), (d, e), (d, g)\}$$

Initially let select (b, c) edge where,

$$u = b$$
$$v = c$$
$$C = \phi \cup \{b, c\}$$
$$= \{b, c\}$$

Now remove from E' every edge incident

$$E' = \{(d, e), (e, f), (d, f), (d, g)\}$$



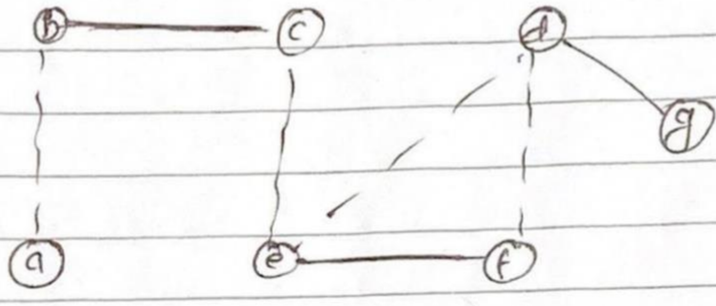
Again let's select (e, f) edge vertex where

$$u = e$$
$$v = f$$

$$C = \{b, c\} \cup \{e, f\}$$
$$= \{b, c, e, f\}$$

Now, remove E' every edge incident

$E' = \{d, g\}$



Again let's select (d, g) ~~ve~~rtex where,

$u = d$

$v = g$

$C = \{b, c, e, f\} \cup \{d, g\}$

$= \{b, c, d, e, f, g\}$

$E' = \phi$