St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 5

Theory of Computation - Module 5

Syllabus

Complexity classes- Tractable problems– Class P –P Complete-Reduction problem- Context grammar nonempty-
Intractable problems- Class NP – NP Complete- Cooks theorem-
Reduction problems-SAT-Clique-Hamiltonian-TSP-Vertex Cover-NP Hard problems.

Contents

Part I. Complexity

In this module, we will learn different complexity classes, such as P and NP.

In data structures, we already learned about time complexity and space complexity. Here we will be concerned about time complexity of various problems.

We learned computable and non computable functions.

A computable function is a function for which we can construct a Turing machine. Or an algorithm can be formulated.

A non computable function is one in which no Turing machine can be constructed. Here we cannot devise an algorithm.

1 Tractable and Intractable Problems

Consider computable functions. These functions are computable or solvable. We can construct a TM for them. We can devise an algorithm for them.

But many of these problems can be solved only in principle, not in practice. This is because some of the computable functions may take 1000s of years to find a solution using a computer system. Such problems are termed intractable problems. They come under complexity class NP.

Most of the problems we are familiar with can be solved within a reasonable amount of time. Such problems are said to be tractable. They come under complexity class P.

Part II. Complexity Class P

Consider some of the problems we learned in data structures, such as bubble sort, quick sort, merge sort, binary search, matrix multiplication etc..

Also we learrned that,

The time complexity of bubble sort is $\Theta(n^2)$.

The time complexity of quick sort is $\Theta(n\,log\,n)$.

The time complexity of merge sort is $\Theta(n\,log\,n)$.

The time complexity of heap sort is $\Theta(n\,log\,n)$.

The time complexity of binary search is $\Theta(log\,n)$.

The time complexity of matrix multiplication algorithm is $\Theta(n^3)$.

Consider time complexity values of all these algorithms. They are of the order of n^2, $n\,log\,n$, $log\,n$, n^3 etc.. If we calculate exact value, we get polynomials such as $5n^2 + 2n + 4$, $n\,log\,n + 5n + 9$, $7n^3 + 3n^2 + 9n + 3$ etc.. They are polynomials. This means that above algorithms are polynomial time algorithms.

These polynomial time algorithms can be solved within a reasonable amount of time. This means these problems can be solved in practice.

Consider the problem, sorting using bubble sort. The time complexity of bubble sort is $\Theta(n^2)$.

This means if there are 10 numbers in the list, a machine will take $\Theta(10^2)$ time to sort. If there are 100 numbers in the list, a machine will take $\Theta(100^2)$ time to sort. If there are 1000 numbers in the list, a machine will take $\Theta(1000^2)$ time to sort.

All these time values are reasonable. A computing machine can solve this sorting problem within a small amount of time.

Almost all the algorithms we learned have time complexity values n^3, n^2, $n\,log\,n$, n, $log\,n$, etc.. That is, for most of these algorithms, the exponent of n is at most 3.

These problems can be solved within a reasonable amount of time by a computing machine or a TM. These problems come in complexity class P.

But if an algorithm for a problem has time complexity $\Theta(n^{1000})$, it is not a reasonable amount of time. This is also a polynomial time. But for such a problem, it has observed that somebody will invent a new algorithm that has time complexity of the order of a small value of exponent such as n^4 or n^3.

2 Complexity Class P

The class P consists of those problems that are solvable in polynomial time.

This means these problems can be solved in time $\Theta(n^k)$, where

n is the size of the input,

k is a constant.

Another defintion is,

A problem is in class P, if there exists a deterministic Turing Machine of polynomial time complexity.

Examples for problems in class P are sorting using bubble sort, quick sort, heap sort etc.. ; searching using binary search, sequential search, ... ; matrix multiplication algorithm etc..

Most of the problems we are familiar with come under class P.

P - Hard Problems

A problem A, is said to be P-hard if,

every P problem can be reduced to A.

P - Complete Problems

A problem A, is said to be P-complete if,

A is P, and

A is P-hard.

Examples for P-complete problems :

Emptiness problem for context free grammars.

Circuit Value Problem (CVP) - Given a circuit, the inputs to the circuit, and one gate in the circuit, calculate the output of that gate

Linear programming - Maximize a linear function subject to linear inequality constraints

Lexicographically First Depth First Search Ordering - Given a graph with fixed ordered adjacency lists, and nodes u and v, is vertex u visited before vertex v in a depth-first search induced by the order of the adjacency lists?

Context Free Grammar Membership - Given a context-free grammar and a string, can that string be generated by that grammar?

Horn-satisfiability: given a set of Horn clauses, is there a variable assignment which satisfies them? This is P's version of the boolean satisfiability problem.

LZW Data Compression - given strings s and t, will compressing s with an LZ78 method add t to the dictionary? (Note that for LZ77 compression such as gzip, this is much easier, as the problem reduces to "Is t in s?".)

3 Emptiness of CFG

Emptiness Problem for Context Free Grammars

The emptiness problem is whether the grammar generates any terminal strings at all.

Emptiness Problem for CFGs is P - Complete

Theorem

The emptiness problem for context-free grammars is P-complete.

Proof

Consider any context-free grammar G =$\{V, \sum, P, S\}$. The emptiness of L(G) can be determined by the following algorithm.

Step 1:

Prove that the problem is P.

Mark each of the terminal symbols in $\sum$.

Search P for a production $A \longrightarrow \alpha$, in which α consists only of marked symbols and A is unmarked. If such a production rule $A \longrightarrow \alpha$ exists, then mark A and repeat the process.

If the start symbol S is unmarked, then declare L(G) to be empty. Otherwise, declare L(G) to be nonempty.

The number of iterations of Step 2 is bounded above by the number of nonterminal symbols in N. Consequently, the algorithm requires polynomial time and the problem is in P.

Step 2:

Prove that the problem is P-hard

To show that the emptiness problem for context-free grammars is P-hard, consider any problem K in P.

[This part of the proof is beyond the scope of this class.]

Part III. Complexity Class NP

Complexity class NP consists of following types of problems:

NP problems,

NP-hard problems,

NP-complete problems.

Consider computable functions. For these functions, a TM exists or they are solvable.

But some of the computable functions can be solved only in principle, not in practice. This is because a computing machine may take 1000s of years to solve such problems.

For these problems, a polynomial time algorithm has not yet been invented. We may wish somebody will invent a polynomial time algorithm for these problems in the future.

For these problems, time complexity is found to be $\Theta(2^n)$. This is not polynomial time, it is superpolynomial time complexity.

When the value of n is 10, time complexity value of such a problem will be $\Theta(2^{10})$, which is a manageable number.

When the value of n is 100, time complexity value of such a problem will be $\Theta(2^{100})$. This value is greater than the number of molecules in the universe. This means, such problems cannot be solved by a a computer in practice (when n is large). They are solvable in principle only. These problems come under complexity class NP.

Some examples for such problems in NP are,

Circuit satisfiability problem,

Boolean Formula satisfiability problem (SAT),

3-CNF satisfiability problem,

Clique problem,

Vertex cover problem,

Hamiltonian cycle problem,

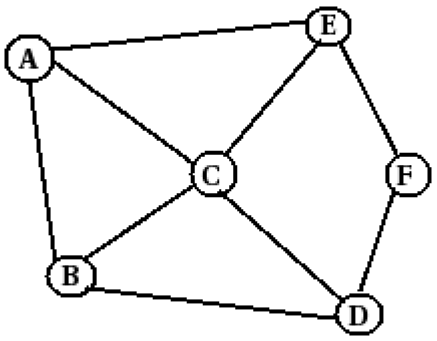
Traveling salesman problem (TSP).

4 Complexity Class NP

The class NP consists of those problems that are “verifiable” using a polynomial time algorithm.

What do you mean by “verifiable”.

This means if somebody gives a ’certificate’ of solution for such a problem, then we can verify that the certifcate is correct in polynomial time. For example, consider the hamiltonian cycle problem,



Let somebody gives us a certificate that a hamiltonian cycle, A-C-B-D-F-E-A exists in the above graph, it can be verified very easily in polynomial time. But if we are asked to find a hamiltonial cycle from the above graph, it cannot be solved in polynomial time.

Another definition is,

A problem is in class NP if there exists a non- deterministic Turing machine of polynomial time complexity.

All problems in P are also in NP. This is because all problems in P are verifiable in polynomial time. That is, $P \subseteq NP$.

Polynomial Time Reducibility

In some cases, a problem can be reduced to another problem.

Consider the problem of solving the linear equation, $bx + c = 0$.

We may transform this to the quadratic equation $0x^2 + bx + c = 0$. Solution of this quadratic equation is same as the solution of the given linear equation.

A problem A is reducible to another problem B, if it is possible to convert every instance of A to a corrsponding instance of B. If this reduction is possible in polynomial time, then we say that A is polynomial time reducible to B.

This is denoted as,

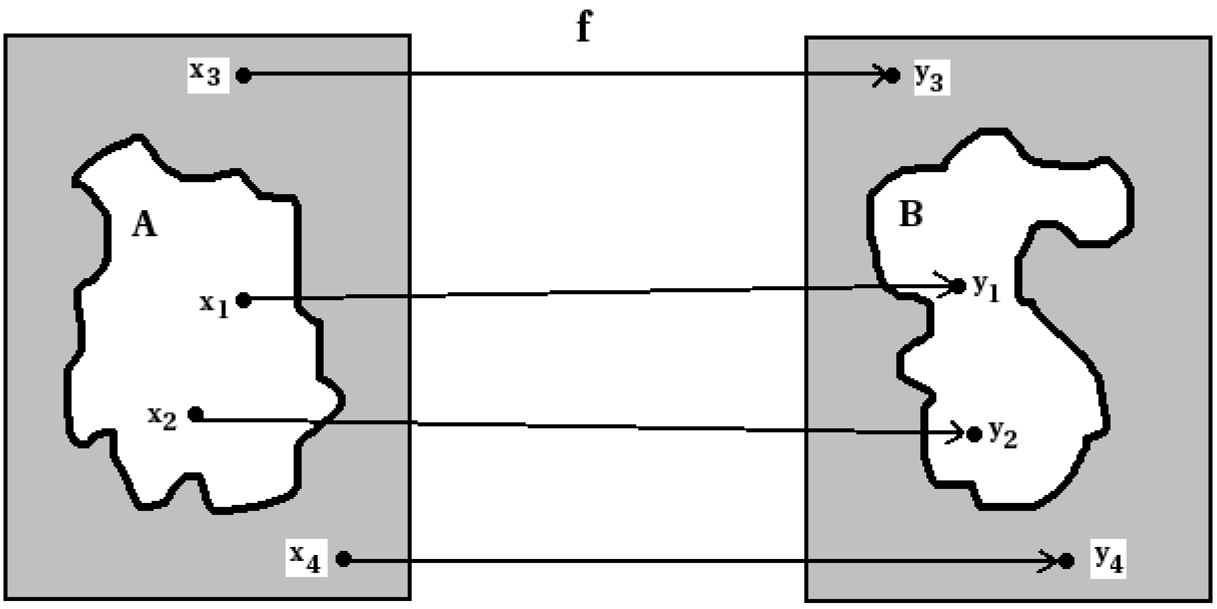
$A \leq_p B$.

This means A can be reduced to B in polynomial time.

If the algorithm used for reduction is f, then if $x \in A$, iff $f(x) \in B$, and

if $x \notin A$, iff $f(x) \notin B$.

This is shown below:



From the above figure, $x_1 \in A$,

$f(x_1) = y_1$,

$y_1 \in B$.

Also, $x_3 \notin A$,

$f(x_3) = y_3$,

$y_3 \notin B$.

5 NP - Hard Problems

A problem A, is said to be NP-hard if,

every NP problem can be reduced to A in polynomial time.

Let A be a problem. We say that A is NP-hard, if

$L \leq_p A$, for every $L \in NP$.

6 NP - Complete Problems

A problem A, is said to be NP-complete if,

A is NP, and

A is NP-hard.

Thus if we want to prove that a problem is NP complete, we need to prove that

it is NP, and

all NP problems can be reduced to this problem (NP-hard).

Examples for NP complete problems are,

Circuit satisfiability problem,

Boolean Formula satisfiability problem (SAT),

3-CNF satisfiability problem (3-CNF-SAT),

Clique problem,

Vertex cover problem,

Hamiltonian cycle problem,

Traveling salesman problem (TSP).

Proving that a problem is NP complete

We will prove that above problems are NP complete. The approach we use is as follows:

We will prove that Boolean Formula satisfiability problem (SAT) is NP complete.

This is done by proving that SAT is NP, and

All problems in NP are reduced to SAT.

To prove that 3-CNF-SAT is NP complete,

Prove that 3-CNF-SAT is NP.

SAT is reduced to 3-CNF-SAT. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. This means all NP problems can be reduced to 3-CNF-SAT.] This means 3-CNF-SAT is NP hard.

To prove that Clique problem is NP complete,

Prove that Clicque problem is NP.

3-CNF-SAT is reduced to Clique problem. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. This means all NP problems can be reduced to Clique problem.] This means Clique problem is NP hard.

To prove that vertex cover problem is NP complete,

Prove that vertex cover problem is NP.

Clique problem is reduced to vertex cover problem. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. Clique problem can be reduced to vertex cover problem. This means all NP problems can be reduced to vertex cover problem.] This means vertex cover problem is NP hard.

To prove that Hamiltonian cycle problem is NP complete,

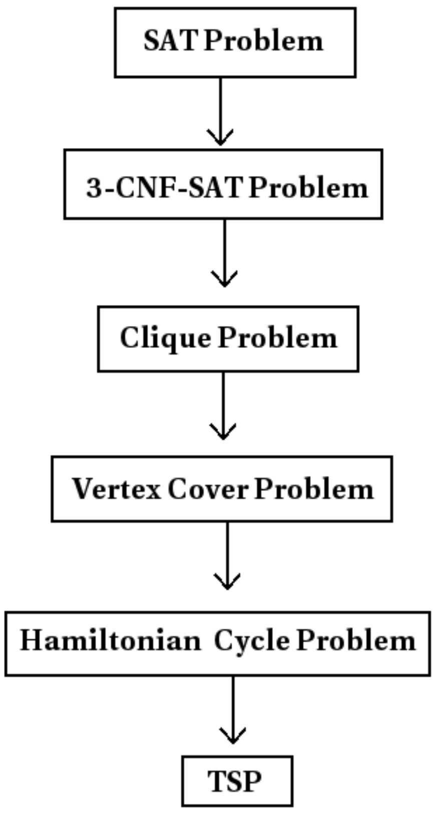
Prove that Hamiltonian cycle problem is NP.

Vertex cover problem is reduced to Hamiltonian cycle problem. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. Clique problem can be reduced to vertex cover problem. Vertex cover problem can be reduced to Hamiltonian cycle problem. This means all NP problems can be reduced to Hamiltonian cycle problem.] This means Hamiltonian cycle problem is NP hard.

To prove that Traveling Salesman(TSP) problem is NP complete, Prove that TSP is NP.

Hamiltonian cycle problem is reduced to TSP. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. Clique problem can be reduced to vertex cover problem. Vertex cover problem can be reduced to Hamiltonian cycle problem. Hamiltonian cycle problem can be reduced to TSP. This means all NP problems can be reduced to TSP.] This means TSP is NP hard.

Thus approach we use for proving a problem is NP-complete is shown below.



Part IV. SAT (Boolean Formula Satisfiability Problem)

7 SAT

Satisfiability

An example for a boolean formula is

$(\neg x_1 \cup x_2) \cap (x_1 \cup \neg x_2) \cap (\neg x_2 \cup x_3) \cap (\neg x_1 \cup \neg x_2 \cup x_3)$.

This formula is in conjunctive normal form (CNF) (intersection of unions).

This boolean formula is satisfiable if there is some assignment of the values 0 and 1 to its variables that causes it to evaluate to 1 (true).

If we put, $x_1 = 0, x_2 = 0, x_3 = 1$, we can see that the above formula evaluates to 1 (true).

This means above formula is satisfiable.

Consider the boolean formula,

$(x_1 \cap x_2) \cap (\neg x_1 \cup \neg x_2)$

If we put $x_1 = 0, x_2 = 0$, this formula evaluates to 0. If we put $x_1 = 0, x_2 = 1$, this formula evaluates to 0. If we put $x_1 = 1, x_2 = 0$, this formula evaluates to 0. If we put $x_1 = 1, x_2 = 1$, this formula evaluates to 0. This means for any values assigned to the variables the above boolean formula always evaluate to 0 (false).

This means the formula is not satisfiable.

SAT Problem

SAT (Boolean Formula Satisfiability) problem is defined as:

Is there a set of values for the boolean variables that assigns the value 1 (true) to the boolean expression given in CNF?

8 Cook's Theorem

Cook's theorem states that

satisfiability of boolean formulas (SAT problem) is NP-Complete.

SAT Problem is NP - Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Satisfiability of boolean formulas is NP-Complete.

Proof:

Step 1: Prove that SAT problem is in class NP.

A boolean variable can have two possible values. Hence, for an n variable boolean expression in CNF, there are 2^n possible combinations of values of these variables.

If we want to check whether a boolean formula is satisfiable, we need to apply each combination to the formula until we get a 1 as the result of evaluation. In the worst case, we need to apply all the 2^n possible combinations.

Consider the formula, $(\neg x_1 \cup x_2) \cap (x_1 \cup \neg x_2) \cap (\neg x_2 \cup x_3) \cap (\neg x_1 \cup \neg x_2 \cup x_3)$. There are 3 variables. Each variable can have two possible values, 0 or 1. There are 2^3 combinations.

So the time complexity of this problem is $\Theta(2^n)$. (not polynomial time)

If we are given a certificate containing a satisfiable assignment for a formula, then we can easily verify it in polynomial time.

For example, for the above formula, if we are given a certificate saying that if $x_1 = 0, x_2 = 0, x_3 = 1$, then just applying these values to the formula, we get,

$(\neg x_1 \cup x_2) \cap (x_1 \cup \neg x_2) \cap (\neg x_2 \cup x_3) \cap (\neg x_1 \cup \neg x_2 \cup x_3)$

=$(\neg 0 \cup 0) \cap (0 \cup \neg 0) \cap (\neg 0 \cup 1) \cap (\neg 0 \cup \neg 0 \cup 1)$

=$(1 \cup 0) \cap (0 \cup 1) \cap (1 \cup 1) \cap (1 \cup 1 \cup 1)$

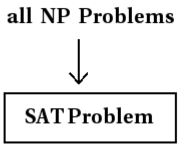
=$1 \cap 1 \cap 1 \cap 1$=1

This can be done in linear time.

So the SAT problem is in class NP.

Step 2: Prove that SAT problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.



A lot of details are involved in this proof. [This proof is beyond the scope of this class].

[Refer the text book 'Introduction to the Theory of computation' by Michael Sipser - Chapter 7- Theorem 7.37]

3-CNF Satisfiability Problem (3-CNF-SAT)

3-CNF

We know that a boolean formula is in conjunctive normal form(CNF) if it is expressed as an AND of clauses, each clause is the OR of one or more variables.

Then, a formula is in 3-conjunctive normal form (3-CNF) is each clause has exactly three distinct literals.

For example, the following boolean formula is in 3-CNF.

$(x_1 \cup \neg x_1 \cup \neg x_2) \cap (x_3 \cup x_2 \cup x_4) \cap (\neg x_1 \cup \neg x_3 \cup \neg x_4) \cap (x_2 \cup x_1 \cup \neg x_3)$

The clauses in the above formula contain exactly 3 literals. The formula is in 3-CNF.

3-CNF Satisfiability Problem

3-CNF-SAT problem is defined as:

Is there a set of values for the boolean variables that assigns the value 1 (true) to the boolean expression given in 3-CNF?

3-CNF Satisfiability Problem is NP-complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Satisfiability of boolean formulas in 3-CNF is NP-complete.

Proof:

Step 1: Prove that 3-CNF-SAT is in class NP.

A boolean variable can have two possible values. Hence, for an n variable boolean expression in CNF, there are 2^n possible combinations of values of these variables.

If we want to check whether a boolean formula is satisfiable, we need to apply each combination to the formula until we get a 1 as the result of evaluation. In the worst case, we need to apply all the 2^n possible combinations.

Consider the 3-CNF formula, $(x_1 \cup \neg x_1 \cup \neg x_2) \cap (x_3 \cup x_2 \cup x_4) \cap (\neg x_1 \cup \neg x_3 \cup \neg x_4) \cap (x_2 \cup x_1 \cup \neg x_3)$. There are 4 variables. Each variable can have two possible values, 0 or 1. There are 2^4 combinations.

So the time complexity of this problem is $\Theta(2^n)$. (not polynomial time)

If we are given a certificate containing a satisfiable assignment for a formula, then we can easily verify it in polynomial time.

For example, for the above formula, if we are given a certificate saying that if $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1$, then just applying these values to the formula, we get,

$(x_1 \cup \neg x_1 \cup \neg x_2) \cap (x_3 \cup x_2 \cup x_4) \cap (\neg x_1 \cup \neg x_3 \cup \neg x_4) \cap (x_2 \cup x_1 \cup \neg x_3)$

=$(0 \cup \neg 0 \cup \neg 1) \cap (0 \cup 1 \cup 1) \cap (\neg 0 \cup \neg 0 \cup \neg 1) \cap (1 \cup 0 \cup \neg 0)$

=$(0 \cup 1 \cup 0) \cap (0 \cup 1 \cup 1) \cap (1 \cup 1 \cup 0) \cap (1 \cup 0 \cup 1)$

=$1 \cap 1 \cap 1 \cap 1 \cap 1$=1

This can be done in linear time.

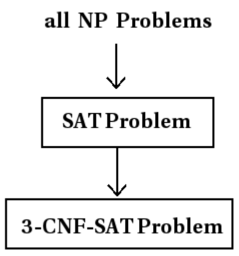
So the 3-CNF-SAT problem is in class NP.

Step 2: Prove that 3-CNF-SAT problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, if we can reduce SAT to 3-CNF-SAT in polynomial time, we can say that 3-CNF-SAT is NP-hard.



We learned in the subject, 'logic design' that any boolean formula can be expressed in CNF. Then, a boolean formula in CNF can be rewritten in 3-CNF.

Consider a boolean formula as follows:

$a \cup b \cup c \cup d$

This can be converted to 3-CNF as,

$(a \cup b \cup x) \cap (c \cup d \cup \overline{x})$

Thus any boolean formula can be transformed to a 3-CNF boolean formula. This can be done using De Morgan's laws.

That means SAT problem can be reduced to 3-CNF-SAT problem. That is, 3-CNF-SAT is NP-hard.

In step 1, we proved that 3-CNF-SAT is in NP.

In step 2, we proved that 3-CNF-SAT is NP-hard.

So, 3-CNF-SAT is an NP-complete problem.

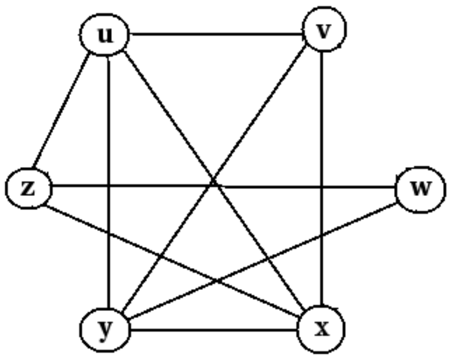
Part V. Clique Problem

9 Clique Problem

Clique

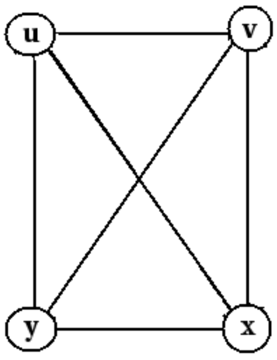
Let we are given a graph with a number of vertices. A clique is a subset of vertices, such that an edge is present between every pair of vertices.

Consider the following graph:



In the above graph, a clique is {u,v,x,y}. The size of this clique is 4.

A clique is a subgraph of the above graph which is shown below:



The above set is a clique, because uv, ux, uy, vx, vy, xy are edges in the given graph.

Clique Problem

Clique problem is to check whether a clique of size k is present in the graph.

Clique Problem is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Clique problem is NP-complete.

Proof:

Step 1: Prove that clique problem is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible subgraphs.

So an algorithm needs to check whether any one of these subgraphs form a clique. It has worst case time complexity $\Theta(2^n)$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a subset of vertices form a clique. An algorithm can verify this in polynomila time.

So clique problem is in class NP.

Step 2: Prove that clique problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Then, if we can reduce 3-CNF-SAT problem to clique problem in polynomial time, we can say that clique is NP-hard.

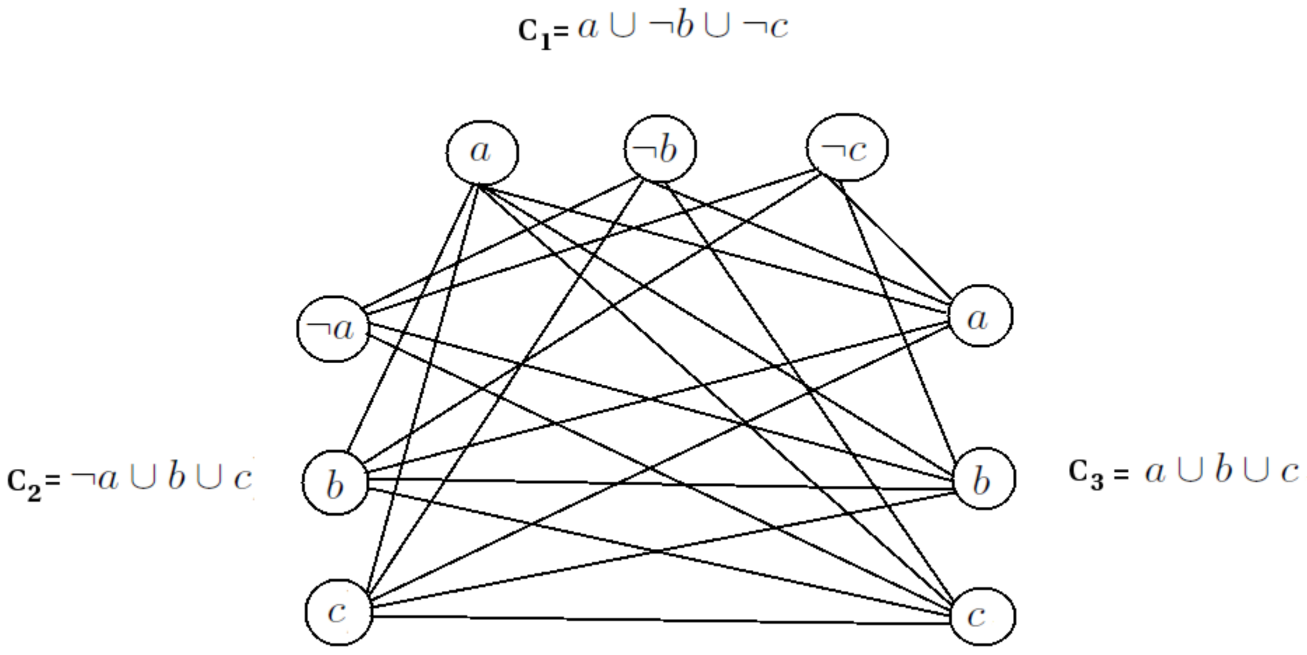
$$\begin{tikzcd}
\text{all NP Problems} \arrow[d] \\
\boxed{\text{SAT Problem}} \arrow[d] \\
\boxed{\text{3-CNF-SAT Problem}} \arrow[d] \\
\boxed{\text{Clique Problem}}
\end{tikzcd}$$

We need to reduce 3-CNF-SAT problem to clique problem.

Consider a 3-CNF boolean formula given below:

$(a \cup \neg b \cup \neg c) \cap (\neg a \cup b \cup c) \cap (a \cup b \cup c)$

A graph is produced from the above formula as:



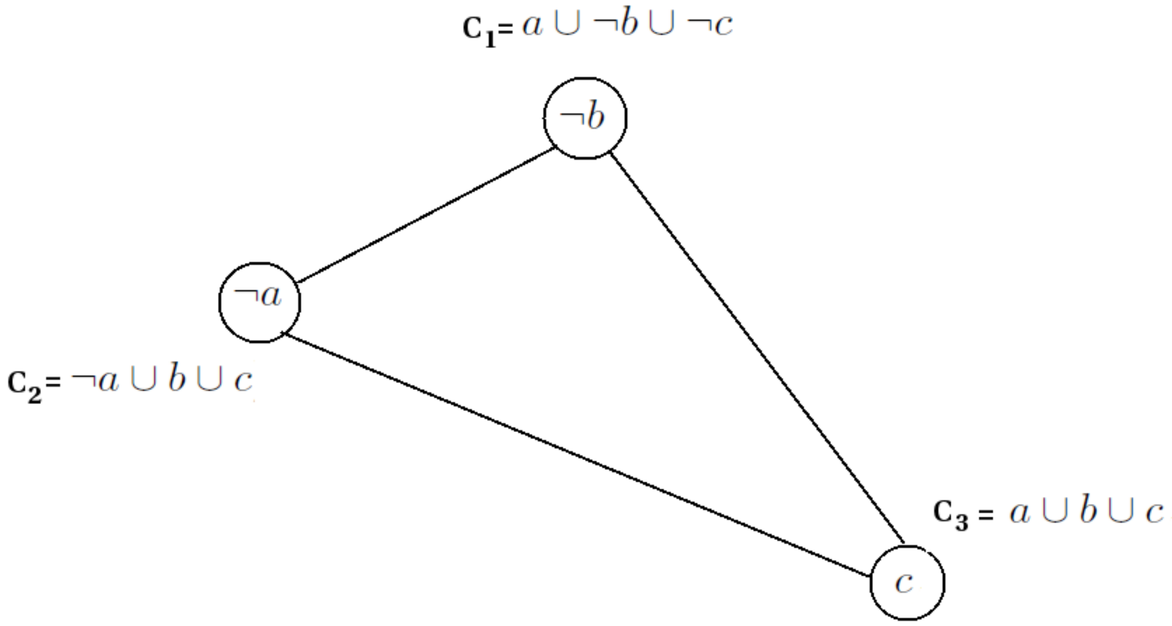
C$_1$= $a \cup \neg b \cup \neg c$
a
$\neg b$
$\neg c$
C$_2$= $\neg a \cup b \cup c$
$\neg a$
b
c
C$_3$ = $a \cup b \cup c$
a
b
c

The graph is produced as follows:

Every vertex in a clause is connected to every other vertex in another clause. But a vertex x should not be connected to vertex $\overline{x}$.

This graph corresponding to 3-CNF-SAT has a solution if the corresponding vertices form a clique.

In the above graph, vertices $\neg b$ of C_1, $\neg a$ of C_2, c of C_3 form a clique.



C$_1$= $a \cup \neg b \cup \neg c$
$\neg b$
$\neg a$
C$_2$= $\neg a \cup b \cup c$
C$_3$ = $a \cup b \cup c$
c

In the above clique, C_1has the vertex $\neg b$. So put b=0.

C_2has the vertex $\neg a$. So put a=0.

C_3has the vertex c. So put c=1.

Thus a satisfying assignment to the given boolean formula is a=0, b=0, c=1.

Consider the given boolean formula,

$$(a \cup \neg b \cup \neg c) \cap (\neg a \cup b \cup c) \cap (a \cup b \cup c)$$

Put a=0, b=0, c=1, we get,

$(0 \cup \neg 0 \cup \neg 1) \cap (\neg 0 \cup 0 \cup 1) \cap (0 \cup 0 \cup 1)$

$=(0 \cup 1 \cup 0) \cap (1 \cup 0 \cup 1) \cap (0 \cup 0 \cup 1)$=$1 \cap 1 \cap 1$=1

Thus the given formula in 3-CNF gives true for a=0, b=0, c=1.

Thus the given 3-CNF-SAT problem is reduced to clique problem. This means clique problem is NP-hard.

In step 1, we proved that clique is in NP.

In step 2, we proved that clique is NP-hard.

So, clique is an NP-complete problem.

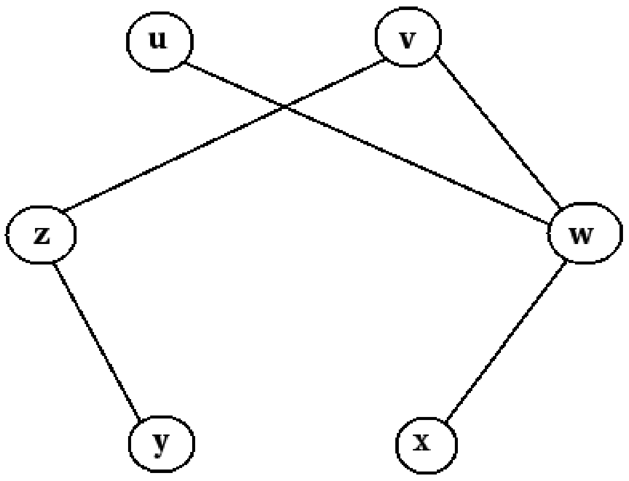
Part VI. Vertex Cover Problem

10 Vertex Cover Problem

Vertex Cover

A vertex cover of a graph is a set of vertices, V such that if (a, b) is an edge of a graph, then a or b or both must be present in V.

Consider the following graph:



In the above graph, vertex cover is {z, w}.

Let V= {z, w}

The edges of the above graph are uw, vw, xw, vz, yz. Note that in every edge, at least one vertex is a member of V.

For example,

Consider the edge uw. Here, vertex $w \in V$.

Consider the edge vz. Here, vertex $z \in V$.

Thus, {z, w} form a vertex cover.

Vertex Cover Problem

The vertex cover problem is to check whether a graph has a vertex cover of size k.

Vertex Cover Problem is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Vertex cover problem is NP-complete.

Proof:

Step 1: Prove that vertex cover problem is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible subsets of vertices.

So an algorithm needs to check whether any one of these subsets form a vertex cover. It has worst case time complexity $\Theta(2^n)$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a subset of vertices form a vertex cover. An algorithm can verify whether at leat one vertex of every edge in the graph is an element of this vertex cover in polynomial time.

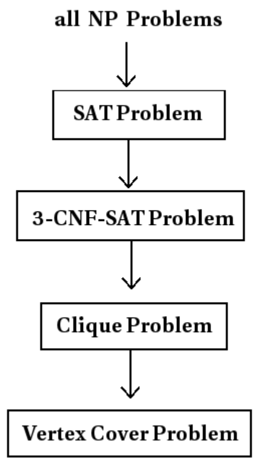
So vertex cover problem is in class NP.

Step 2: Prove that vertex cover problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

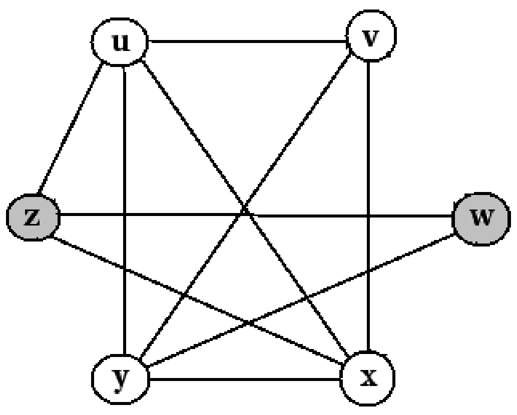
The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Again, we reduced 3-CNF-SAT problem to clique problem. Then, if we can reduce clique problem to vertex cover problem in polynomial time, we can say that vertex cover problem is NP-hard.



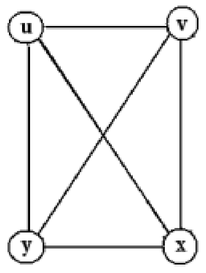
We need to reduce clique problem to vertex cover problem.

Consider the following graph with clique {u, v, x, y}.



Graph, G

V={u,v,w,x,y,z}

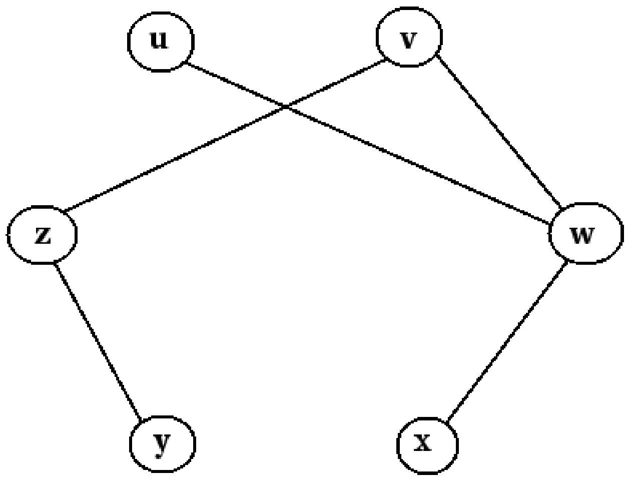


Clique of G = {u,v,x,y}

V ' ={u,v,xy}

Take the complement of the above graph, G. That is $\overline{G}$.

$\overline{G}$ is



Graph, $\overline{G}$

For the graph $\overline{G}$, vertex cover is {z,w}.

This vertex cover is found by V - V'.

That is, {u,v,w,x,y,z} - {u,v,x,y} = {z,w}

So from the clique, we can find out the vertex cover of a graph using the above mechanism.

Thus the given clique problem is reduced to vertex cover problem. This means vertex cover problem is NP-hard.

In step 1, we proved that vertex cover problem is in NP.

In step 2, we proved that vertex cover problem is NP-hard.

So, vertex cover problem is an NP-complete problem.

Part VII. Hamiltonian Cycle Problem

11 Hamiltonian Cycle Problem

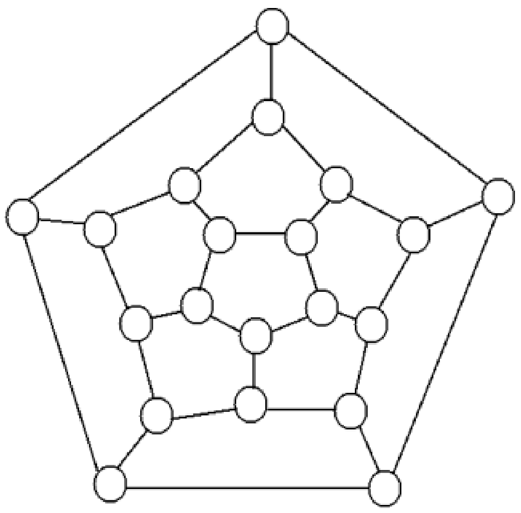
Hamiltonian Cycle

A hamiltonian cycle in an undirected graph is a simple cycle which contains every vertex of the graph.

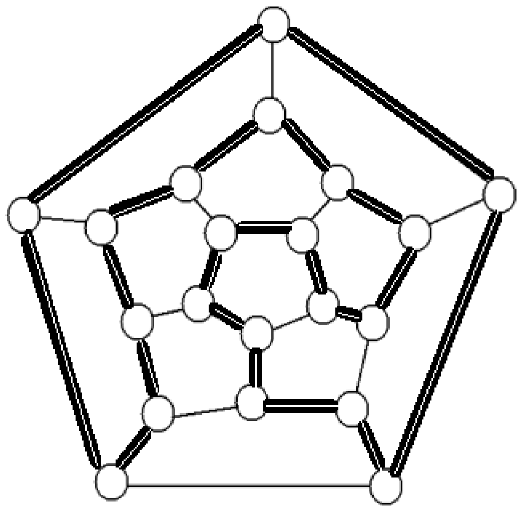
A graph which contains a hamiltonian cycle is called hamiltonian. A graph which does not contain a hamiltonian cycle is called nonhamiltonian.

Example:

Consider the following graph:



Graph



Ham Cycle is shown in thick lines

Note that the hamiltonian cycle path passes through every vertex.

Hamiltonian Cycle Problem

The hamiltonian cycle problem is to check whether a graph has a hamiltonian cycle..

Hamiltonian Cycle Problem is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Hamiltonian cycle problem is NP-complete.

Proof:

Step 1: Prove that hamiltonian cycle problem is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible permutations of vertices.

So an algorithm needs to check whether any one of these permutation form a hamiltonian path. If the graph is represented as an adjacency matrix, it will take $n!$ comparisons. It has worst case time complexity $\Theta(2^{\sqrt{n}})$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a sequence of vertices form a hamiltonian cycle. An algorithm can verify whether this path is hamiltonian in polynomial time.

So hamiltonian cycle problem is in class NP.

Step 2: Prove that hamiltonian cycle problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Again, we reduced 3-CNF-SAT problem to clique problem. again, we reduced clique problem to vertex cover problem. Then, if we can reduce vertex cover problem to hamiltonian cycle problem in polynomial time, we can say that hamiltonian cycle problem is NP-hard.

$$\begin{tikzcd}
\textbf{all NP Problems} \arrow[d] \\
\boxed{\textbf{SAT Problem}} \arrow[d] \\
\boxed{\textbf{3-CNF-SAT Problem}} \arrow[d] \\
\boxed{\textbf{Clique Problem}} \arrow[d] \\
\boxed{\textbf{Vertex Cover Problem}} \arrow[d] \\
\boxed{\textbf{Hamiltonian Cycle Problem}}
\end{tikzcd}$$

We need to reduce vertex cover problem to hamiltonian cycle problem.

* [The proof of this part is beyond the scope of this class]

* [Refer the text book: Introduction to Algorithms by Cormen, Chapter 34-Section 34.5.3]

Part VIII. Travelling Salesman Problem (TSP)

12 Travelling Salesman Problem (TSP)

Travelling Salesman

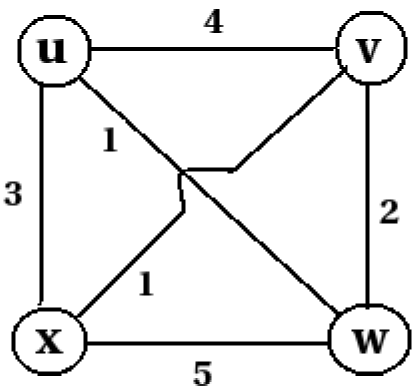
A salesman begins his tour from a city. He visits a set of cities and he finishes at the city he starts from. Each city must be visited exactly once.

Travelling Salesman Problem (TSP)

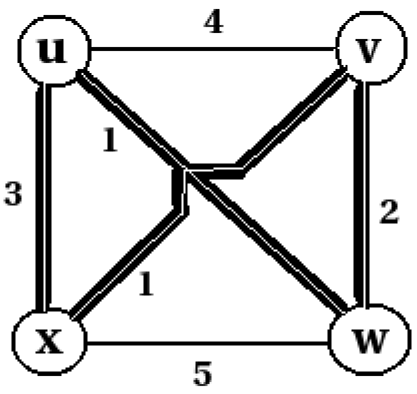
A graph of a set of cities is given. A salesman starts from a city. He must visit each city exactly once and should finish at the city he starts from. The path he follows should be the shortest one or should have a cost value less than k.

Thsu TSP is a special case of Hamiltonian cycle problem where the path cost should be minimum.

Consider the following graph containing cities u, v, w, x.



Graph



cycle x-u-w-v-x

Here the traveling salesman can follow the path x-u-w-v-x which forms the shortest path with cost 7km.

TSP is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

TSP is NP-complete.

Proof:

Step 1: Prove that TSP is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible permutations of vertices.

So an algorithm needs to check whether any one of these permutation form a hamiltonian path with cost value less than k. If the graph is represented as an adjacency matrix, it will take $n!$ comparisons. It has worst case time complexity $\Theta(2^{\sqrt{n}})$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a sequence of vertices form a hamiltonian cycle with cost value less than k. An algorithm can verify whether this cycle has cost less than k in polynomial time.

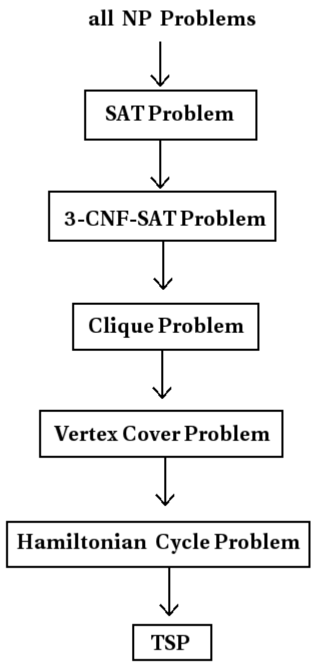
So TSP is in class NP.

Step 2: Prove that hamiltonian cycle problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

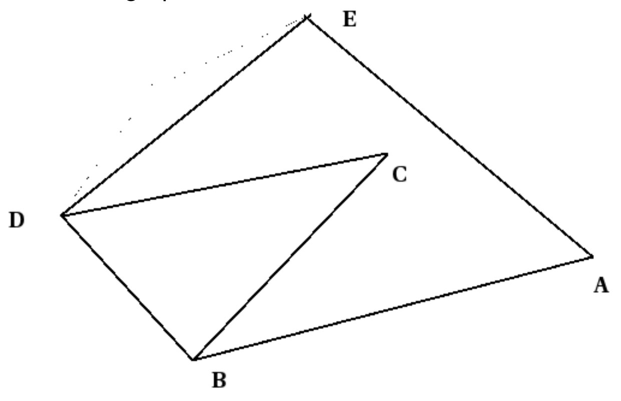
The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Again, we reduced 3-CNF-SAT problem to clique problem. Again, we reduced clique problem to vertex cover problem. Again we reduced vertex cover problem to hamiltonian cycle problem. Then, if we can reduce hamiltonian cycle problem to TSP in polynomial time, we can say that TSP is NP-hard.

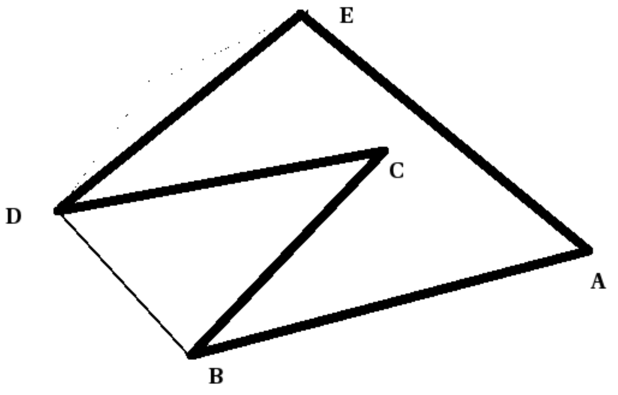


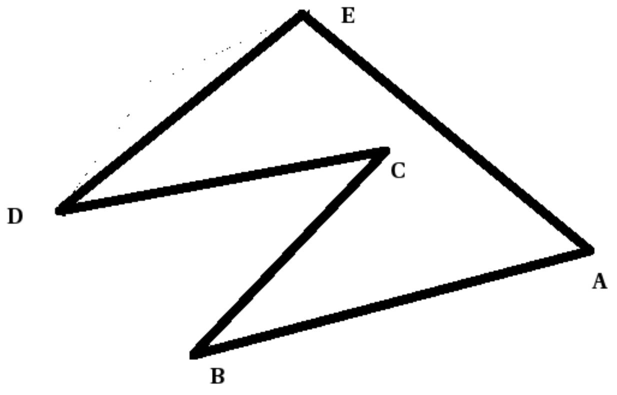
We need to reduce hamiltonian cycle problem to TSP.

Consider a graph,

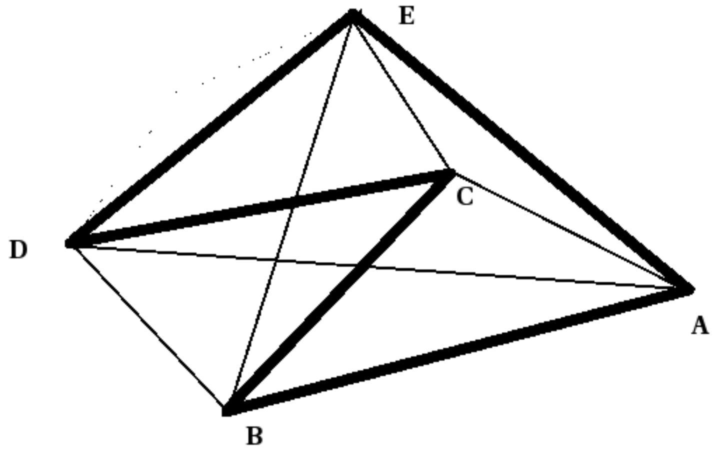


Following is an instance of hamiltonian cycle.

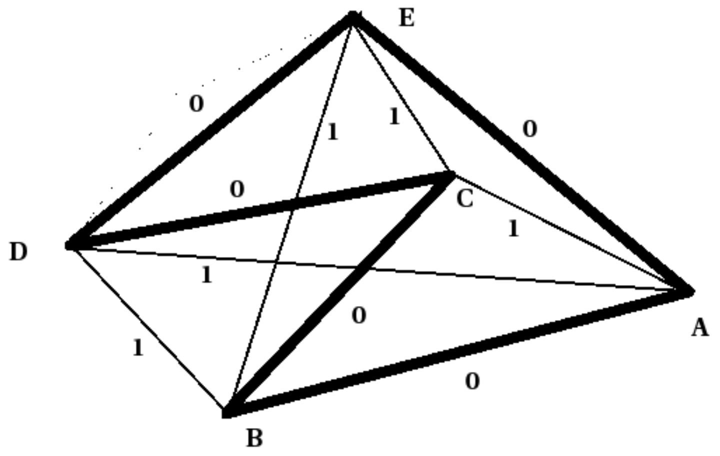




Now form a complete graph from the above istance of Hamiltonian cycle.



Next we assign cost value to every edge of this graph. If the edge is present in hamiltonian cycle, assign cost value 0 to it. If it is not, assign cost value 1 to it.



From this, an instance of TSP can be produced from the above graph by traversing through the edges which have cost value 0.

Thus hamiltonian cycle problem is reduced to TSP. So TSP is NP-hard.

In step 1, we proved that TSP is in NP.

In step 2, we proved that TSP is NP-hard.

So, TSP is an NP-complete problem.

Questions (from Old syllabus of S7CS TOC)

MGU/Nov2011

1. What is meant by intractable problem (4marks)?

2. Briefly explain NP hard problem (4marks).

3a. Explain the different classification of problems based on their complexity.

OR

b. Prove that the satisfiability problem in NP complete (12marks).

MGU/April2011

1. What are complexity classes (4marks)?

2a. Is travelling salesman problem is NP-complete or not? Prove (12marks).

OR

b.

MGU/Nov2010

1. Differentiate tractable and intractable problems (4marks).

2. What is an NP hard problem? Give example (4marks).

3a. Show that halting problem of a Turing machine is not NP complete.

OR

b. What are complexity classes and tractable and intractable problems (12marks).

MGU/May2010

1a. Define the class NP (4marks).

b.

2a.

OR

b. i. Let L be an NP complete language. Then P=NP if and only if $L \sum P$.

ii. Construct truth tables for the following formula:- $(A \longleftrightarrow (A \longleftrightarrow B))$. (12marks).

3a. Show that Travelling salesman problem is NP-complete.

OR

b. Show that the following formula of propositional calculus is a Tautology (12marks).

MGU/Nov2009

1. What are tractable and intractable problems (4marks)?

2a.

OR

b. Explain in detail about class P, NP complete and NP hard problems (12marks).

MGU/May2009

1. Give an example of an NP complete problem (4marks).

2a. Prove that Travelling Salesman's problem is NP complete.

OR

b. State the halting problem. Show that it is not NP complete (12marks).

MGU/Nov2008

1. Explain tractable and intractable problems (4marks).

2a. Explain P, NP, NP-hard and NP complete problems with example (12marks).

OR

b.

MGU/May2008

1. Cite example for NP hard problem (4marks).

2a. Prove that Traveleing Salesman's problem is NP complete (12marks).

OR

b.

MGU/Dec2007

1. Explain classes P, NP and NP completeness (4marks).

2a. Explain algorithmic complexity and NP hard problems.

OR

b. What is integer programming and show how it is a NP-complete problem (12marks).

MGU/July2007

1. When do you call a decision problem as Np-hard (4marks)?

2. a.

OR

b. Show that the problem of checking whether a graph has a clique of size k is NP-complete (12marks).

MGU/Jan2007

1. What will happen when somebody finds a deterministic polynomial time algorithm for an NP complete problem? Comment on its consequence on the complexity classes (4marks).

2. Define class P and class NP (4marks).

3a. Prove that 3SAT problem is NP-complete.

OR

b. State Cook's theorem and give an outline of its proof (12marks).

MGU/July2006

1. What is an NP problem? Give an example (4marks).

2a. Discuss any two NP hard graph problems in detail.

OR

b. i. What are NP-complete problems? Give two examples.

ii. Briefly explain the satisfiability problem. Is it an NP problem (12marks)?

MGU/Nov2005

1. Differentiate between tractable and intractable problems (4marks).

2a. Define NP, NP-hard, NP-complete and P problems. Explain with examples.

OR

b. State the halting problem. Show that it is not NP-complete (12marks).

References

.

Cormen, T, H; Leiserson, C, E; Rivest, R, L; (2001). Introduction to Algorithms. PHI.

Nagpal, C, K (2011). Formal Languages and Automata Theory. Oxford University Press.

Sipser, M (2007). Introduction to the Theory of Computation. Thomson Course Technology.

Hopcroft, J, E; Motwani, J; Ullman, J, D (2002). Introduction to Automata Theory, Languages and Computation. Pearson Education.

website: http://sites.google.com/site/sjcetcssz