**CS1105: Design and Analysis of Algorithm**

****

Institute of Engineering & Technology (IET)

JK Lakshmipat University, Jaipur

**Submitted by: Submitted to:**

Prachika Kanodia: 2019/btech/cse068 Mr. Santosh Kumar Verma

(Assignment)

**ASSIGNMENT-5**

1. Optimization versus search. Recall the traveling salesman problem:

TSP

Input: A matrix of distances; a budget b

Output: A tour which passes through all the cities and has length <=b, if such a tour exists.

The optimization version of this problem asks directly for the shortest tour.

TSP-OPT

Input: A matrix of distances

Output: The shortest tour which passes through all the cities.

Show that if TSP can be solved in polynomial time, then so can TSP-OPT.

**Ans.** Let TSP(H,b) will returns false if no tour of length b or less than b exists in H.

Now TSP-OPT(H):-

i=0

for all t,u belongs to Z:

i = i + dist(t,u)

return BINARY-SEARCH-TOUR(H,0,i)

BINARY-SEARCH-TOUR(H,l,t)

b = ((l + l + t)/2)

if TSP (H,b) ≠ false

return BINARY-SEARCH-TOUR(H,l,b)

else

return BINARY-SEARCH-TOUR(H,b,t)

So, this algorithm will do a binary search using all the lengths.

So, if TSP is solved using polynomial time then TSP-OPT can also be solved using polynomial time. We have to make polynomial number of calls from varying out output b using binary search algorithm.

Binary Search Algorithm is important right here and we can’t simplify increment the value of b by 1 because the sum of distances is exponentially less than the length of the input.

Now using BSA, no tour has weight > b. So the bound for all the shortest tour is (0,b]. Let start algorithm from b=t/2 and if tour not possible then we can able to know that the shortest path is between (t/2,t]. But if possible then it is between the (0,t/2].

So, we use recursive function till we get minimum edge length in visual graph. The total time taken will be log(t) \* log(nx) where log(t) is the polynomial time and log(nx) is the time to solve TSP. Therefore, TSP-OPT can also be solved in the polynomial time if TSP can able to solved in that polynomial time.

1. Search versus decision. Suppose you have a procedure which runs in polynomial time and tells you whether or not a graph has a Rudrata path. Show that you can use it to develop a polynomial time algorithm for RUDRATA PATH (which returns the actual path, if it exists).

**Ans. Rules of Polynomial time algorithm for RUDRATA-PATH:-**

Assume that there is a Procedure P which starts from s and ends with e. It runs in a polynomial time and this search problem contains RUDRATA-PATH if relation P(s,e) = true satisfies.

Suppose a graph has a rudrata-path which starts from edge s and ends with edge e. Therefore, s 🡪 e has rudrata-path. Let Ar is an algorithm. First run algorithm on edge s and if yes then that means rudrata-path exists.

Now remove s and return the algorithm again. If yes then rudarata-path exists in the smaller graph means exists in the original graph.

But if answer comes no then reverse the procedure and run the same Ar algorithm in the edge e side of the graph and store the edges that has been removed till the time we got the rudrata-path.

**Let’s implement the algorithm:-**

Function RudrataPath(G)

if not R(G) then return “no path”

E’ 🡨 E

for each e belongs to E

G’ 🡨 (V,E’ – {e})

if R(G’) then E’ 🡨 E’ – {e}

return E’ //Return the RUDRATA-PATH

1. Give a simple reduction from 3D MATCHING to SAT, and another from RUDRATA CYCLE to SAT.

(Hint: In the latter case you may use variables xij whose intuitive meaning is vertex i is the jth vertex of the Hamilton cycle; you then need to write clauses that express the constraints of the problem.)

**Ans.** **For 3D MATCHING to SAT:-**

Let there is a set of n edges which every includes 3 items. A legitimate matching exists if there are set of edges such that each object seems in an edge, however no item seems a part of it.

So, {e1, e2,………..en} be the edges having a as an k-vector ex. If the item exists at e1, e2 and e7 edges then a={1,2,7} and k=3.

**Now create clause:-**

* Add a clause (ea1 V ea2 V … eak). This clause ensures that the item appears at least once.
* For every pair i, j belongs to {1…k}, add clause (eai V eaj). After taking together this clause will ensure that it will not appear more then once.

To show that this reduction done using polynomial time assume m items and n edges then iterate over m items and every item generate n entries and n(n-1) clauses with 2 entries means O(mn2).

According to the hint given we need to make sure that each vertex appears only one time in a cycle and neighbours connected by the edges.

Now the clauses includes two requirements which are (xij V xkj) and (xij V xk,j+1) where j + 1 = 1 and j = n.

So overall this reduction is improved by construction. And the construction of SAT instance includes creation of n clauses with n entries plus n3 clauses with 2 entries in clauses of both the requirements. Therefore, the reduction takes O(n3) time.

1. On page 266 we saw that 3SAT remains NP-complete even when restricted to formulas in which each literal appears at most twice.

a. Show that if each literal appears at most once, then the problem is solvable in polynomial time.

**Ans.** Assume x number of variables and y number of clauses such that each variable satisfy at most one clause. Now using graph visualization having variables on left and clauses on right side connect each variable with the clause it matches.

According to the formula, each clause must pick at least one variable from which it is connected to. Now if we connect all the variables with source u and all the clauses with sink v with fixing capacity of all edges as e, and this is equal to the flow of y units from u to v. This is achieved in polynomial time so we can achieve the satisfiability problem also in the polynomial time.

b. Show that INDEPENDENT SET remains NP-complete even in the special case when all the nodes in the graph have degree at most 4.

**Ans.** Considering the reduction from the 3SAT to INDEPENDENT SET then if each literal appears at most 2 times then it can be said that graph has degree 4 at each vertex. Now assume x1, x2 and x3 be the three clauses then inside the graph each vertex let say x1 has one edge each to x2 and x3. Also, it has edges to all the occurrences inside the graph that means appears at most 2 times means graph has degree 4 at each vertex in a same way that is in the case of x1.

1. Determine which of the following problems are NP-complete and which are solvable in polynomial time. In each problem you are given an undirected graph G = (V;E), along with:

(Hint: All the NP-completeness proofs are by generalization, except for one.)

1. A set of nodes L is subset of V , and you must find a spanning tree such that its set of leaves includes the set L.

**Ans.** This problem can be solved in a polynomial time.

Remove vertices in L. Build a spanning tree T with inside the closing graph the usage of BFS or DFS. For every vertex v belongs to L, join it to any of its neighbours found in T. If a few vertex in L doesn’t have a neighbour in T, then we can’t discover a spanning tree that has all vertices in L, in any other case the spanning tree have to encompass all vertices in L.

1. A set of nodes L is subset of V, and you must find a spanning tree such that its set of leaves is precisely the set L.

**Ans.** This is a NP-complete problem.

Given a spanning tree T, we will verify in the polynomial time, that whether or not its leaves are precisely vertices in L. It may be decreases from RUDRATA (s,t)-PATH. Now input is G,s,t to RUDRATA (s,t)-PATH then allow G,L := {s,t} form in input. If G has a RUDRATA (s,t)-PATH P, then G has a spanning tree, that is P, with the set of its leaves being L, then G has a RUDRATA (s,t)-PATH, which is T. The reduction done in the in polynomial time.

1. A set of nodes L is subset of V, and you must find a spanning tree such that its set of leaves is included in the set L.

**Ans.** This is NP-complete problem.

Given a spanning tree T, we will verify this also in the polynomial time, that whether or not its leaves are included in L. It can reduced from RUDRATA (s,t)-PATH. Now input is G,s,t to RUDRATA (s,t)-PATH i.e. G,L := {s,t} form. If G has a RUDRATA (s,t)-PATH P, then G has a spanning tree, that's P, with the set of its leaves being included in L then L = {s,t}. If G has a spanning tree T with the set of its vertices being included in L, then G has a RUDRATA (s,t)-PATH (T). Here the simplest considerable graphs having at-least 2 vertices. The rduction is done in the polynomial time.

1. An integer k, and you must find a spanning tree with k or fewer leaves.

**Ans.** This is NP-complete problem.

The evidence is achieved using the RUDRATA-PATH i.e. lowering from RUDRATA (s,t)-PATH with k=2.

1. An integer k, and you must find a spanning tree with k or more leaves.

**Ans.** This is NP-complete problem.

Given a spanning tree (T), we will verify in polynomial time, that whether it has k or more leaves. It may be decreased from CONNECTED DOMINATING SET which has desires to discover a related dominating set in G that has maximum b vertices.

Now input as G,b to CONNECTED DOMINATING SET that means G,k := n−b wherein n = |V|. If G has a related dominating set D of length at maximum b, then for every vertex v belongs to V − D, join to certainly considered among its neighbours in D. We get a spanning tree T with at the least n − b leaves. If G has a spanning tree with atleast k leaves, then G has a related dominating set D, that’s formed by eliminating all leaves from T. Than D has maximum n − k vertices. The reduction is done in the polynomial time.

1. An integer k, and you must find a spanning tree with exactly k leaves.

**Ans.** This is NP-complete problem.

The evidence is achieved by lowering the RUDRATA (s,t)-PATH with k=2.

1. Given an undirected graph G = (V;E) in which each node has degree <=d, show how to efficiently find an independent set whose size is at least 1=(d+1) times that of the largest independent set.

**Ans.** Let G be the graph and I =0.

Algorithm has to be repeated until we get G = 0.

* Pick node v with smallest degree and let I = I U {v}.
* Delete v and all the neighbors of v from the graph G.
* Now, let G is the new graph.

I is an independent set. At each step, I increases by one vertex and delete at most d + 1 vertices from graph since v has at most d neighbours. Hence there at least exists |v|/(d + 1) iterations of it. Now assume that k is the size of maximum independent set.

Then previous algorithm implies:- |I| >= |v|/(d + 1) >= k/(d + 1)