Name: Inchara Raveendra

SCU ID: 00001653600

Assignment - 8

34.1-1

Define the optimization problem LONGEST-PATH-LENGTH as the relation that associates each instance of an undirected graph and two vertices with the number of edges in a longest simple path between the two vertices. Define the decision problem LONGEST-PATH $= \{\langle G, u, v, k\rangle : G = (V, E)$ is an undirected graph, $u, v \in V$, $k \geq 0$ is an integer, and there exists a simple path from u to v in G consisting of at least k edges$\}$. Show that the optimization problem LONGEST-PATH-LENGTH can be solved in polynomial time if and only if LONGEST-PATH $\in$ P.

To show that optimization problem LONGEST-PATH -LENGTH can be solved in polynomial time and only if LONGEST-PATH belongs to P, we have to prove the below:

1. If LONGEST-PATH-LENGTH can be solved in polynomial time, then LONGEST-PATH belongs to P
2. If LONGEST-PATH belongs to P, then LONGEST-PATH- -LENGTH can be solved in polynomial time

To prove (1)

Suppose we can solve the optimization problem in polynomial time. To show the latter, we

have to show that there exists a poly time algorithm that decides whether there is a simple path at least k edges between u and v in G
This can be achieved using binary search.
We first compute the longest simple paths in G using LONGEST-PATH-LENGTH. If this path has fewer than 'k' edges, we return "no"
Otherwise, we perform binary search on the path length by repeated path division in half and checking whether the two resulting sub-paths each have length of at least 'k'. This process takes at most O(log n) iterations, since the path length is at most n-1 and each iteration can be done in polynomial time. Therefore, we have a polynomial time algorithm for LONGEST-PATH

To prove ②, we need to show there exists a polynomial time algorithm that given an undirected graph G and two vertices u and v, computes the number of edges in a longest simple paths between u and v.

This can be solved by binary search again. We first compute the length of the shortest path between u and v using a polynomial-time algorithm for the shortest-path problem. We then perform binary search on the length by repeatedly checking if there is a simple path between u and v with length at least halfway between the lengths of the shortest and longest paths found so far. This process takes at most $O(\log n)$ iterations, and each iteration can be done in polynomial time using the decision algorithm for LONGEST-PATH. Therefore, we have a polynomial time algorithm for LONGEST-PATH-LENGTH

Therefore, we have shown that LONGEST-PATH-LENGTH can be solved in polynomial time and only if LONGEST-PATH belongs to P.

34.1-4

Is the dynamic-programming algorithm for the 0-1 knapsack problem that is asked for in Exercise 16.2-2 a polynomial-time algorithm? Explain your answer.

No. It is a problem solvable in pseudo-polynomial time, where the running time is proportional

to the product of input size and the maximum value of the items $O(nW)$

Therefore, the running time of this algorithm is not polynomial in size of input, which is proportional to sum of number of bits required to represent each item's value and weight

In other words, the algorithm's running time grows exponentially in the number of bits required to represent the input, making it an exponential time algorithm. As such, 0-1 Knapsack is not considered to be a polynomial time algorithm.

34.2-2
Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian.

This can be proved by means of contradiction. Assume G is hamiltonian, i.e, it contains a Hamiltonian cycle that visits every vertex exactly once. Since G is bipartite, the Hamiltonian cycle must alternate between the two bipartite set

of vertices. Lets say, the cycle starts in one set and ends in other, without loss of generality.

Since Hamiltonian cycle visits every vertex exactly once, it must have an even number of edges.
However, since G has an odd number of vertices, the two bipartite sets must have different sizes.
Therefore, the Hamiltonian cycle must have an odd number of edges that cross between the two bipartite sets.
This is impossible because such an edge must connect a vertex in one set to a vertex in the other set. Therefore, such edge contributes to an even number of vertices to each bipartite set.
Since the two sets have different sizes, it is impossible for an odd number of edges to cross between them.
Therefor, our assumption is wrong and G is non hamiltonian.

34.2-3
Show that if HAM-CYCLE $\in$ P, then the problem of listing the vertices of a hamiltonian cycle, in order, is polynomial-time solvable.

If HAM-CYCLE $\in$ P, then there exists a polynomial time algorithm that can decide whether a given graph contains a Hamiltonian cycle or not.

We can use the following approach:

a) Start with any vertex in the graph

b) If the graph contains a Hamiltonian cycle, then there must be an edge from the current vertex to some other vertex in the cycle.

c) Follow this edge to the next vertex in the cycle, and add it to the list of vertices

d) Repeat steps a and 3 until all vertices in the cycle have been added to the list.

This will correctly list the vertices in the order in which they appear in the cycle.

We can use HAM-CYCLE algorithm to check if there exists an edge between the current vertex and some other vertex in the cycle. If such an edge exists, we know that current vertex is a part of Hamiltonian cycle and we can follow it

to the next vertex in the cycle. If no such edge exists, then current vertex is not a part of Hamiltonian cycle and we can terminate the algorithm.

Since HAM-CYCLE algorithm runs in polynomial time, and the above algorithm performs a polynomial number of calls the HAM-CYCLE algorithm, it follows that the problem of listing the vertices of a Hamiltonian cycle in order is also polynomial-time solvable.

34.2-5
Show that any language in NP can be decided by an algorithm running in time $2^{O(n^k)}$ for some constant k.

Let L be a language in NP E non-determin-istic Turing Machine M that decides L in polynomial time.

a) Given an input x, N first guesses a certificate c of length at most n^k
b) N then runs M on input (x, c) to check whether M accepts (x, c)

c) If M accepts (x, c), then N accepts x; rejects otherwise

Running time of N depends on time needed to guess the certificate. Since, the length of c is at most n^k, there are at most 2^{n^k} possible certificates. Time needed to run M on input (x, c) is polynomial in n since M runs in polynomial time

∴ Total running time of N is $2^{O(n^k)}$ where 'k' is constant that determines the length of the certificate guessed by N

34.2-6

A ***hamiltonian path*** in a graph is a simple path that visits every vertex exactly once. Show that the language HAM-PATH $= \{\langle G, u, v\rangle$: there is a hamiltonian path from u to v in graph $G\}$ belongs to NP.

To prove the above, we need to show that given a potential certificate for a given instance (G, u, v), we can verify in polynomial time whether the certificate is correct or not.

A certificate for (G, u, v) would be a sequence of vertices of G such that $v_1 = u$, $v_n = v$ and every v_i is distinct. To verify this sequence represents a Hamiltonian path in G, we can check the following conditions in polynomial time -

a) $v_1 = u$

b) $v_n = v$

c) $\forall i \in \{1, 2, \ldots, n\}$, $v_i \in V$

d) $\forall i \in \{1, 2, \ldots, n-1\}$, $j \in \{i+1, \ldots, n\}$, $v_i \neq v_j$

e) $\forall i \in \{1, 2, \ldots, n-1\}$, $\{v_i, v_{i+1}\} \in E$

All of these conditions can be checked in polynomial time. Therefore, we can verify a potential certificate HAM-PATH in polynomial time. HAM-PATH belongs to NP