

**CE2001 ASSIGNMENT SUBMISSION**

Name: Venkat Subramanian

Matriculation Number: U1921075D

Email:VENKAT001@e.ntu.edu.sg

Tutorial Group: SEP2

**Question 1**

Given 2 graphs are the graphs similar? The answer to this question can either be yes or a no. Since the outcome is a binary outcome, the question to check if 2 graphs are similar is a decision problem. Therefore, the problem can be a NP,P,NPC or even a NP-hard problem.

The 2 graphs are said to be similar/isomorphic , if there exist a bijection between the vertex sets of the 2 graphs (G,H) such that any two vertices u and v of G are adjacent in G if and only if f(u) and f(v) are adjacent in H (Komusiewicz, 2011){Komusiewicz, 2011 #28} .

By observing the graphs given at first glance, the vertex a has a higher degree in one of the graphs then the other. Some might falsely conclude that these graphs are not isomorphic. Upon further inspection it can be concluded that there is indeed a bijection function to map the nodes on the right graph to the nodes on the left graph. The mapping are as follows (a,d),(b,a),(c,c),(d,b). The alphabet on the right represent the vertex in the graph that is on the right and the alphabet on the left represents the vertex in the graph that is on the left.

Therefore, we are not able to decide whether 2 graphs are isomorphic just by purely checking their adjacency matrix, since the nodes are represented differently in different graphs. In order to solve this question, we must calculate all possible combinations of the bijection function. Then we are able to identify if there is an existence of a bijection function that will map all the nodes in one graph to the nodes in the other graph. In the given example, the vertex a has 4 possible choices to be mapped to , the next vertex b will only have 3 choices since 2 vertexes cannot be mapped to the same vertex. Vertex c has 2 choices and so on and so forth. Therefore, in a graph of V vertexes, the number of possible bijection functions are V!

This might be possible reason as to why many scientists equate Graph Isomorphism problem to Graph Isomorphism counting problem (Köbler, 2006) since all possible bijection functions has to be computed. Hence, this decision problem is not a P problem as it cannot be solved in polynomial time by deterministic algorithm.

This problem however can be solved in polynomial time by a non-deterministic algorithm. If the non-deterministic algorithm is able to randomly assign a bijection function and verifies that the computed bijection function is valid. As a result, this problem can be classified as a NP problem (Zemlyachenko, Korneenko, & Tyshkevich, 1985).

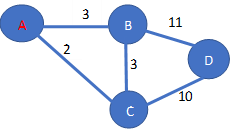
We are also able to verify whether the 2 graphs are similar in polynomial time by a deterministic algorithm if the bijection function is known. If the bijection function is known, we can map the elements in the adjacency matrix appropriately. After which, we can compare all elements in one of the graph’s matrix with the other graph’s matrix. If the decision problem is a NP-complete problem, it would fail the exponential time hypothesis (Babai, 2015). And if it was an NPC problem, it will imply that the rest of the NPC problems are also solvable in Quasipolynomial time. Thus, this problem cannot be factually concluded that it’s not an NPC problem until proven so.

**Question 2**

As we know, the travelling salesman problem is a NP-Hard problem, therefore we are unable to find the optimal path in which the Hamiltonian cycle obtain has the least weight. But through approximation algorithm, we can estimate or obtain a ‘good enough’ solution to the problem.

One such algorithm is the nearest neighbor algorithm (adapted from the lecture). The nearest neighbor algorithm first accesses the adjacent node that has the least weight from the source vertex. And from the adjacent node it will explore its adjacent node that has the nearest weight from it. This process is repeated until we traverse through all the nodes. The algorithm only accesses the adjacent nodes that have not been explored. Hence, all the nodes are only check once.

If we assume, that the graph is represented by an adjacency list then the average number of edges on each node is , where E is the total number of edges in the graph. To select the nearest neighbor, there will be comparisons done. As a result, the total number of comparisons done by this algorithm will be . And in the case of a completed graph, making the worst case of the algorithm to be . As a result, the time complexity of the nearest neighbor algorithm is O(.

The limitation of this algorithm is this is can only provide a ‘good enough’ solution. This algorithm does not always provide the optimal solution. Another possible limitation is that, if the graph is not complete some of the vertexes will not be accessible from a stationary starting vertex. Furthermore, there is a possibility the algorithm is not able to compute a cycle due to the presence of a loop as shown in the graph, and the algorithm might prematurely decide that there is no cycles present in the graph, when it is actually present.

Part b)

Best case

Worst case

20

15

2

2

1

10

4

2

1

15

4

6

10

3

5

35

15

20

30

25

24

20

19

1

For both graph the optimal path is A->B->E->G->F->D->C->A since it has a total weight of 118. However, in the worst case, the path selected will be A->F->G->E->B->C->D->A, and it has a weight of 131 which is greater than the optimal path. For the graph in the best case it can identify the optimal path.

**Question 3**

We are assuming that the input size is , therefore the input array will always be a multiple of 2 and would be divisible by 8. The Merge Sort will divide the input array into equal halves, and if the size of the sub list is lesser or equal to 8, we will do an insertion sort on the sub list. After sorting the sub list, we will then merge all the sub list into one list again.

The best case occurs when the input array given is already in ascending order. If the input array is in ascending order each sub list will be compared in order to be merged again. And in each of the sub list, the insertion sort algorithm will only compare the elements in the sub list once. The base case for this algo will be 8 comparison, since the algo switches to insertion sort after that. Let W(n) be the number of comparisons.

**Question 3b**

The best case when constructing the heap, is when the input array is already sorted in descending order. The arbitrary heap created from this array will already be a maximizing heap. All the internal nodes have to be compared twice to verify that they are indeed ordered in a maximining heap. Since the total number of elements in the array is always odd (based on our assumption) the number of internal node present will be , where n represent the number of elements in the array. Therefore, the total number of comparisons in the best case is The worst case is when the array is sorted in ascending order, the arbitrary heap created will be a minimizing heap. To form a maximizing heap from this, the first element has to be brought down by k levels, the second element in the heap needs to be brought down by k-1 levels, the elements in the third depth has to be brought down by k-3 levels and so on. When an element is brough down from one level to another level, there will be 2 comparisons done. At each level, there is elements present. Hence, the total number of comparisons done is the sum of all elements multiplied with 2 and the number of levels it moved down.

**Question 4**

From the proof of correctness, we know that the path selected by Dijkstra’s algorithm will always select path that will reduce the total distance from the starting vertex to the destination vertex. However, this does not include the number of edges to be added. Therefore, the path in the new graph (after subtracting each edge by 4) might not be the same as the path found in the original graph. An example of this is shown below.

**After**

**Before**

17

21

7

7

11

11

From the example we can see that previously the algortihm decides that the edge from the start to the destination is the shortest path. But after the change the shortest path is from the starting node to the destination through the intermediate node. Therefore the result of the shortest path will change depending on the number of edges there are between the starting node to the destination.

This concludes the assingnment, Thank you.

**References**

Babai, L. (2015). Graph Isomorphism in Quasipolynomial Time. *CoRR, abs/ 1512.03547*.

Köbler, J. (2006). *On Graph Isomorphism for Restricted Graph Classes*.

Komusiewicz, C. (2011). *Parameterized Algorithmics for Network Analysis: Clustering & Querying*: Univ.-Verlag der TU.

Zemlyachenko, V., Korneenko, N., & Tyshkevich, R. (1985). Graph isomorphism problem. *Journal of Soviet Mathematics, 29*(4), 1426-1481. doi:10.1007/BF02104746