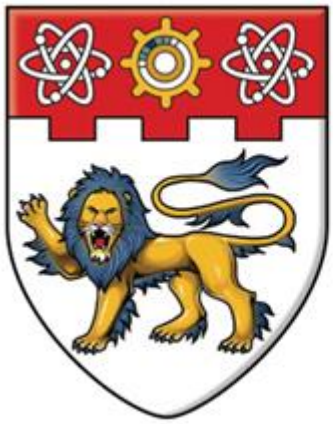


NANYANG TECHNOLOGICAL UNIVERSITY

CZ2001 Algorithms Final Submission

Submission Date:

20 November 2020, 1800hrs

Name: Ernest Ang Cheng Han

Matriculation Number: U1921310H

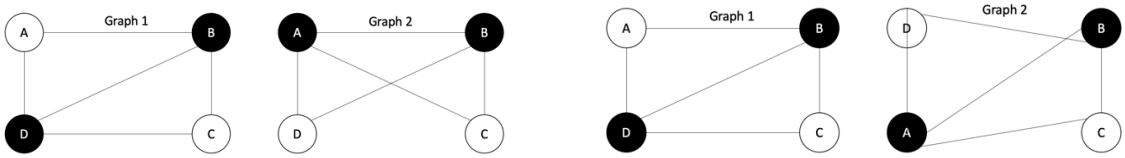
Email: ERNE0009@e.ntu.edu.sg

Lab Group: SS4

1) This question is a decision problem, where the 2 graphs given are either the same or not the same. In my opinion, the two given graphs are the same, or isomorphic, if: (1) They have equal number of vertices, (2) They have equal number of edges, (3) They have the same degree sequences, (4) If cycle of length k is formed by the vertices of graph 1, then the cycle of the same length k must be formed by the vertices of comparing graph 2.

The following 2 graphs have the same number of vertices and edges, with 5 vertices and 5 edges but different degree sequences: Both 2-degree nodes A & C from graph 1 are connected to both 3-degree nodes B & D, likewise, both 2-degree nodes D & C from graph 2 are connected to both 3-degree nodes A & B. Both 3-degree nodes B & D from graph 1 are connected to 2 2-degree and 1 3-degree nodes, likewise, both 3-degree nodes A & B from graph 2 are connected to 2 2-degree and 1 3-degree nodes. Hence, both graphs have the same degree sequences.

Graph 1 {A, B, C, D} can be mapped to Graph 2 {D, B, C, A} as seen below, which allows them to create cycles of similar length from any vertex v from graph 1 to any vertex v' from graph 2. Hence, we can say graph 1 & 2 are isomorphic and hence the same.



The Graph Isomorphism problem is a NP class problem given that it can be verified in polynomial time as all it takes is to iterate through the 2 graphs to verify if the bijection of graph 1 onto graph 2 is an isomorphism, resulting in a verifying time complexity of $O(n^2)$, where n is the number of nodes. However it is not solvable in polynomial time and there is no known algorithm that can solve it in polynomial time, hence causing it to belong in class NP (nondeterministic polynomial time) as the name suggests. To further prove that the Graph Isomorphism problem is in NP complete, we would need to show that the problem is NP Hard since a problem needs to be in NP and NP Hard for it to be NP complete. To prove NP Hardness, we need to select any known problem in NP and reduce it to the graph isomorphism problem, which is not possible or have not been done yet. Further, there are no algorithm bounded by polynomial time deduced for the general case of this solution. Thus, as for now, Graph Isomorphism is a NP class problem.

2a) The Travelling Salesman Problem asks given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits every city exactly once and returns to the original city. To solve this problem, we can implement the Nearest Neighbour Algorithm which finds the path of the lowest cost from each node locally.

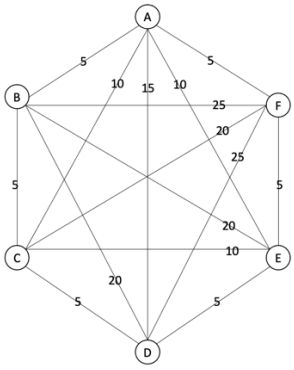
Nearest Neighbour Algorithm:

function Algo(V, E) {
    choose vertex a as origin node to start cycle c;
    $v \leftarrow a$;
    while there are vertices not in c **do**
        select edge vw of minimum weight and w not in c;
        add edge vw to c;
        $v = w$;
    add edge va to c;
    return c;
}

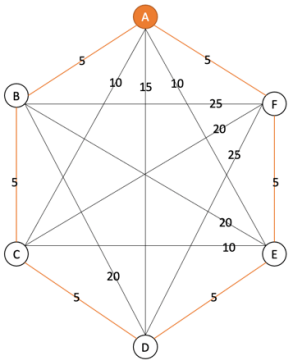
The time complexity of this algorithm would be $O(n^2)$ since the algorithm would have the iterate through every node of the graph, and at every node of the graph, it needs to iterate $\leq n-1$ number of edges where n is the size of the graph to find the nearest neighbour to the parental node.

Limitation of this algorithm is that due to the "greedy" nature of the algorithm, it tends to mostly produce sub-optimal results. Note that for each source node, the total number of paths possible for TSP is $n!$ Though the algorithm is polynomial efficient and can quickly produce a relatively short path, because of the fact that the algorithm essentially picks the shortest route at each node locally without considering the paths it will take ahead or the paths it has taken so far, this creates the higher likelihood of it failing to produce the most optimal results, or even producing paths that are worst off and much more expensive.

2b) Input 1 (Best case):



Input Graph 1

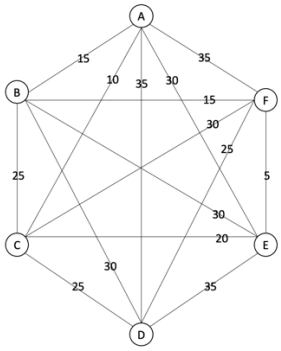


Algorithm Generated Path

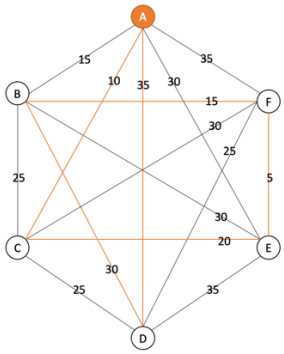
Cost of shortest path from A (A, B, C, D, E, F, A) or (A, F, E, D, C, B, A): 30

In this case, Nearest Neighbour Algorithm would be successful in producing the most optimum path from every source node.

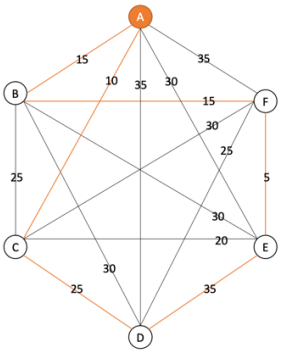
Input 2 (Not the best case):



Input Graph 2



Algorithm Generated Path



Alternative Shorter Path

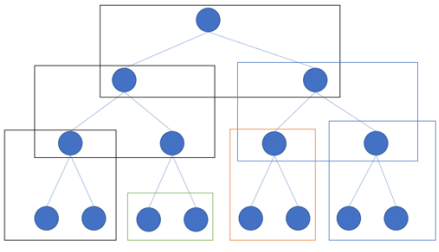
Cost of "shortest" path from A (A, C, E, F, B, D, A): 115 (Generated by algorithm)
Cost of alternative path from A (A, B, F, E, D, C, A): 105

In this case, Nearest Neighbour Algorithm would be unsuccessful in producing the most optimum path from every source node as seen from comparatively less expensive shorter path contrasting the path generated by the algorithm.

3a) Given problem size $n = 2^k$, the hybrid algorithm would split the main array into 2 repeatedly till each of the subarrays size are $\leq 2^3$. Insertion Sort are then performed on

these subarrays, before merge sort is then invoked to sort the subarrays recursively. Given that constant a exist such that $2^{k-a} \leq 2^3$ and that we will be using the best case when implementing the hybrid algorithm, number of comparisons for insertion sort is $2^a(2^3-1)$, where there are 2^a number of subarrays of size 2^3 and each subarray takes 7 comparisons. Number of comparisons for merge sort is $a * 2^{k-1}$ in the best case since there are a number of recursions before dividing the array to size 8, and the total number of comparisons at each level of a is 2^{k-1}. Thus, total number of comparisons is $7 * 2^a + a * 2^{k-1} = 7 * 2^{k-3} + (k-3) * 2^{k-1}$, since k – 3 = a. Overall, we get $\frac{7}{2^3} * 2^k + \frac{(k-3)}{2} * 2^k = \frac{4k-5}{8} * 2^k$

3b) Since we constructing a maximum heap from an ascending array of size $2^k - 1$, we will need to calculate the number of comparisons for each level of the maximum heap as well as the number of additional comparisons for each level of the maximum heap (if any) when performing fixHeap().



Referencing the pattern observed above to visualize the number of fixHeap() hence comparisons, we can hence derive the following formula to count comparisons:

$\sum_{i=1}^{k} 2^{i-1} * 2 * (k-i)$,

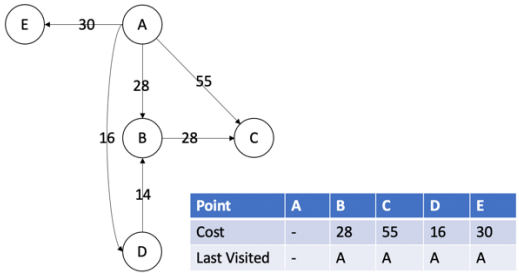
where k is the number of levels of the maximum heap,
2^{i-1} *refers to the number of nodes at the* i^{th} *level of the maximum heap,*
2 *refers to the number of comparisons required per fixing (per fixHeap()) for each node at* i^{th} *level of the maximum heap,*
$k-i$ *refers to the number of fixings required by each node at* i^{th} *level of the maximum heap implemented by the fixHeap() function*

$$\sum_{i=1}^{k} 2^{i-1} * 2 * (k-i) = 2k * \sum_{i=1}^{k} 2^{i-1} - 2 * \sum_{i=1}^{k} 2^{i-1} * i = \frac{(2k)(2^k-1)}{2-1} - [(k-1)2^{k+1}+2]$$
$$= (2k)(2^k-1) - (k-1)2^{k+1} - 2 = k * 2^{k+1} - 2k - k * 2^{k+1} + 2^{k+1} - 2$$
$$= 2^{k+1} - 2k - 2$$

For the highlighted portions, it is derived via:

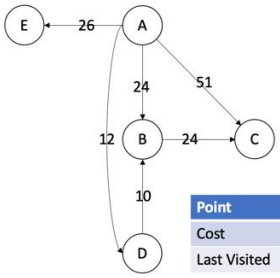
$$S = \sum_{i=1}^{k} 2^{i-1} * i, 2S = \sum_{i=1}^{k} 2^i * i = \sum_{i=2}^{k+1} 2^{i-1} * (i-1), S = 2S - S,$$
$$S = \sum_{i=2}^{k+1} 2^{i-1} * (i-1) - \sum_{i=1}^{k} 2^{i-1} * i = \sum_{i=2}^{k+1} 2^{i-1} * i - \sum_{i=2}^{k+1} 2^{i-1} - \sum_{i=1}^{k} 2^{i-1} * i$$
$$= 2^k * (k+1) - 1 - \sum_{i=2}^{k+1} 2^{i-1} = 2^k * (k+1) - 1 - \frac{2(2^k-1)}{2-1}$$
$$= 2^k * (k+1) - 1 - 2^{k+1} + 2 = (k-1) * 2^k + 1$$

4) Depending on the structure of the directed and weighted G, the shortest paths computed in G' may vary or stay the same as G.



Point	A	B	C	D	E
Cost	-	28	55	16	30
Last Visited	-	A	A	A	A

Graph G1

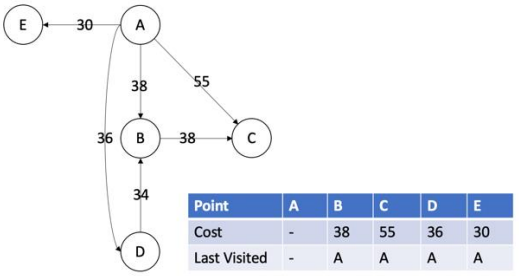


Point	A	B	C	D	E
Cost	-	22	46	12	26
Last Visited	-	D	B	A	A

Graph G1'

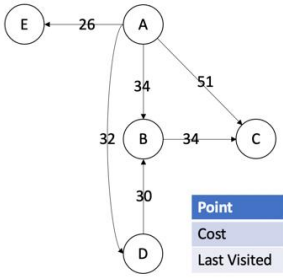
As seen above, is this particular context, when the cost of 4 units is reduced from every edge in G1 without changing the direction of edges, the shortest paths from A to B and A to C are different. For A to B, this is because there are 2 routes from A to B (A to B & A to D to B). When removing the weight of 4 from all edges in G1 to produce G1', the path from A to B is affected only once while the path from A to D to B is affected twice. Hence, the magnitude of reduction of the cost of path from A to D to B is twice compared to A to B, which makes A to D to B now cheaper compared to A to B as illustrated above. This effect is also seen in determining the shortest path from A to C, where now the cheapest path is from A to B to C.

However, as previously mentioned, in some cases, the multiple reductions of weight 4 in the possible shortest paths from any node to another particular and reachable node may not be significant enough to change the shortest paths generated by Dijkstra's Algorithm. This is seen below in another example. In this context, reduction in cost of weight 4 from every edge in G to produce G' will not influence the shortest paths generated and determined by Dijkstra's Algorithm.



Point	A	B	C	D	E
Cost	-	38	55	36	30
Last Visited	-	A	A	A	A

Graph G2



Point	A	B	C	D	E
Cost	-	34	51	32	26
Last Visited	-	A	A	A	A

Graph G2'

According to the proof of correctness of Dijkstra's Algorithm, Lemma 1 states that for a given shortest path from X to Z, any nested paths within X to Z are also the shortest; D1 states that any vertex V added to Z whereby the cost from V to Z is the smallest will result in an overall shortest path from X to V; D2 states that any path generated by Dijkstra's Algorithm from node n to n' reachable from n within graph G will be the shortest. Applying this proof of correctness, even though when 4 units is reduced from each edge of graph G proportionately, the reduction in cost to the many paths from a particular node n to another particular and reachable node n' may not be proportional as illustrated above.

Overall, depending on the structure of Graph G in terms of the weight distribution of its edges and the directions of the individual edges, the shortest paths in G may or may not be different from the shortest paths in G' after the reduction in weight 4 from every edge of G.