1.Not the same graph but isomorphic. Let left Graph be G1, right be G2.

$$\alpha = \begin{array}{ccccc} G1 & a & b & c & d \\ G2 & c & a & d & b \end{array}$$

$$\alpha = V(G1) \rightarrow V(G2)$$

$$\text{G1} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 1 \\ b & 1 & 0 & 1 & 1 \\ c & 0 & 1 & 0 & 1 \\ d & 1 & 1 & 1 & 0 \end{array} \qquad \text{G2} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 1 & 1 \\ b & 1 & 0 & 1 & 1 \\ c & 1 & 1 & 0 & 0 \\ d & 1 & 1 & 0 & 0 \end{array}$$

NP. This is a non-deterministic problem, G1 is isomorphic to G2 and vice versa, as there is a bijection between vertices of each Graph. For every edge in G1, there is also an edge in G2 that can be mapped to.

The Graph Isomorphism problem is in **NP.**

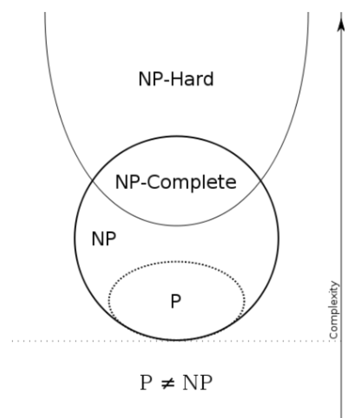


Figure 1: NPC diagram

2 (a). We will use the Christofides Algorithm for the TSP problem.

1. Find minimum spanning tree **T,** for a given graph **G**, using **<u>Prim + Fibb Heaps</u>.**
2. Find vertices with odd degree in **T**, and place into a set **S**.

3. From **S**, find perfect minimal-weight matching edges **M,** using **<u>Blossom V</u>.**
4. Add edges **M** into **T.**
5. Find the Eulerian cycle, using **<u>Fleury's</u>.**
6. Remove the edges that causes a vertex to be visited repeatedly. This will then create a Hamiltonian Path.

The algorithm is a **1.5 Approximation:** $\boldsymbol{O}(|\boldsymbol{V}|^2|\boldsymbol{E}|)$; $\boldsymbol{M} \leq \frac{\boldsymbol{OPT}}{\boldsymbol{2}}$

The cost of finding **T** is: $\boldsymbol{O}(|\boldsymbol{E}| + \boldsymbol{log}\ (|\boldsymbol{V}|))$**,** E=No. of Edges, V=Vertices.

The cost of finding **S** is: $\boldsymbol{O}(|\boldsymbol{V}|)$

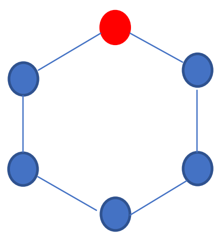
The cost of finding **M** is: $\boldsymbol{O}(|\boldsymbol{V}|^2|\boldsymbol{E}|)$

The most time-consuming cost would be finding **M**, which brings the total complexity to $\boldsymbol{O}(|\boldsymbol{V}|^2|\boldsymbol{E}|)$. For step 6 to work, we assume that the graph satisfies the triangle inequality, therefore the weights of the edges must satisfy the inequality:

$$\boldsymbol{e}(\boldsymbol{v},\boldsymbol{u}) + \boldsymbol{e}(\boldsymbol{v},\boldsymbol{w}) \geq \boldsymbol{e}(\boldsymbol{u},\boldsymbol{w}), \forall \boldsymbol{p};\ \boldsymbol{p} = \boldsymbol{all\ possible\ pairs\ of\ vertices\ u, v, w}$$

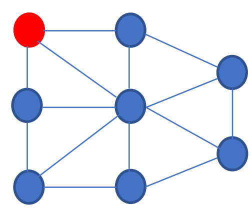
2 (b). "Best" Solution Example 1; "Not-The-Best" Solution Example 2

Red=Start



<u>Example 1</u>

Red=Start



<u>Example 2</u>

1: Simple MST, least odd degree vertices, simple Eulerian cycle.

2: More complicated to find MST, more vertices are odd degrees.

3 (a). Best case for Mergesort will be when array is already sorted and divided into 2 equal halves, $\frac{\boldsymbol{n}}{\boldsymbol{2}}$ comparisons. Since Mergesort will only be performed until the input size

is 8 or lesser, there will be 3 recursions of Mergesort that will be subtracted out of the total number of comparisons.

Total num of comparisons: $\frac{\boldsymbol{n}}{\mathbf{2}} - \frac{\mathbf{8}}{\mathbf{2}} = \frac{\boldsymbol{n}-\mathbf{8}}{\mathbf{2}}$

Best case for Insertionsort will be when the array is already sorted. Since Mergesort will divide an array of size 16 into 2 arrays of size 8, Insertionsort will be performed on **2 arrays**. Therefore, Insertionsort will have: $\mathbf{2} * (\boldsymbol{n} - \mathbf{1})$ comparisons.

Total num of comparisons: $(\frac{\boldsymbol{n}-\mathbf{8}}{\mathbf{2}}) + (\mathbf{2}(\boldsymbol{n} - \mathbf{1}))$

3 (b). Let **n** = $2^k - 1$, nodes (distinct elements). Let height of a tree (**h**) = log(**n**).

A complete binary tree of h levels has $2^k - 1$ nodes:

$$k = 1, 1 \leq 2^1 - 1$$

$$k = 2, 2 \leq 2^2 - 1$$

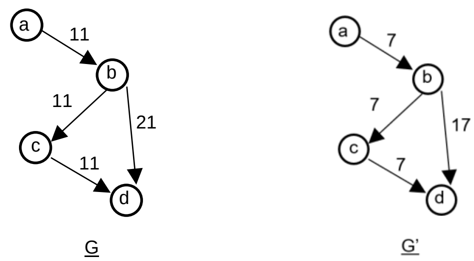
$$\ldots$$

$$k \leq 2^k - 1$$

Since the input array B is in ascending order, the **<u>first (smallest)</u>** inserted element will be at the top and the **<u>last (biggest)</u>** will be at a root. And since there are 2 comparisons for each level, the total number of comparisons will be

2*Height of Tree: $\mathbf{2} * \boldsymbol{h} = \mathbf{2} * \boldsymbol{log}(\mathbf{2}^{\boldsymbol{k}} - \mathbf{1})$

4. No. Shortest path in G’ != G. Assume Dijkstra applied to G and G’. Consider the graphs below as example.



Let Edges be **e**, Weight be **w**, Source be **a**, Destination be **d,** Path be **s**

Shortest path from Vertex **a** to **d** for G is {a-b-d} with w1=**32**.

Shortest path from Vertex **a** to **d** for G’ is {a-b-c-d} with w2=**21**.

s1 = {a-b-d}

s2 = {a-b-c-d}

In the above example there are 2 paths s1, s2, from the source(a) to destination(d). s1 has e=3, s2 has e=2. By reducing the weight of every single edge in the graph by a constant x, we are reducing the weight of the paths by,

$$\boldsymbol{w = \sum(e_i - x), i = edges\ of\ path}$$

So, the weights for each path, s1 and s2, before reduction:

$$\text{s1}: (e_1 - 0) + (e_2 - 0) = (11 - 0) + (21 - 0) \ = \ \mathbf{32}$$

$$\text{s2}: (e_1 - 0) + (e_2 - 0) + (e_3 - 0) = (11 - 0) + (11 - 0) \ + (11 - 0) = \ \mathbf{33}$$

After reduction of weights:

$$\text{s1}: (e_1 - 4) + (e_2 - 4) = (11 - 4) + (21 - 4) \ = \ \mathbf{24}$$

$$\text{s2}: (e_1 - 4) + (e_2 - 4) + (e_3 - 4) = (11 - 4) + (11 - 4) \ + (11 - 4) = \ \mathbf{21}$$

We can see that the shortest path (min weight) has changed from s1 to s2 after the reduction. Therefore, the addition/subtraction of weights on every edge in a graph will affect the shortest path of the graph.

References

[1] L.Babai “How hard is Graph Isomorphism?” *Graph Isomorphism in Quasipolynomial Time,* v2, p. 82, January 2016. [Online]. Available: https://arxiv.org/pdf/1512.03547v2.pdf. [Accessed: Nov. 11, 2020].

[2] N. Christian “Heuristics for the Travelling Salesman Problem” 2003. [Online]. Available: http://pirun.ku.ac.th/~fengpppa/02206337/htsp.pdf. [Accessed: Nov. 13, 2020].