Give examples of the following graphs or explain why such examples cannot exist.

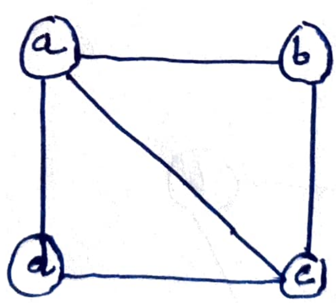
a) Graph with a Hamiltonian Circuit but without an Eulerian circuit.

b) Graph with an Eulerian circuit but without a Hamiltonian circuit

c) Graph with both a Hamiltonian circuit and an Eulerian circuit.

d) Graph with a cycle that includes all the Vertices but with neither a Hamiltonian circuit nor an Eulerian Circuit.

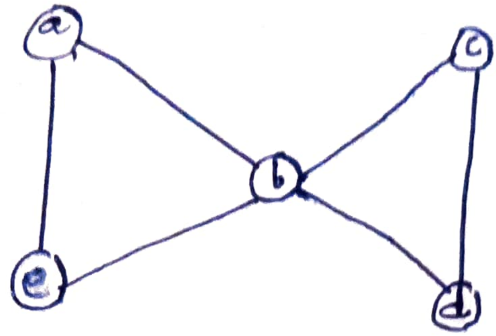
a)



Graph (a) has a Hamiltonian Circuit (a-b-c-d-a) but no Eulerian circuit because

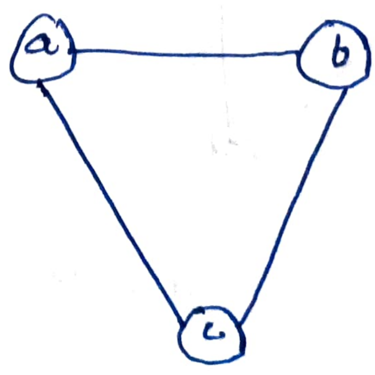
It has vertices of odd degrees (a and c)

b)



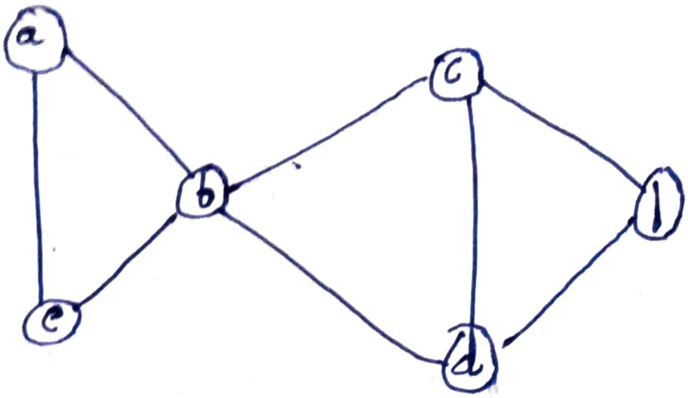
Graph (b) has an Eulerian circuit (a-b-c-d-b-e-a) but no Hamiltonian circuit. If a Hamiltonian circuit existed, we could consider a as its starting vertex without loss of generality. But then it would have to visit b at least twice: once to reach c and the other to return to a.

c)



Graph (c) has both a Hamiltonian circuit and an Eulerian circuit (a-b-c-a)

d)



Graph (d) has a cycle that includes all the vertices (a-b-c-f-d-b-e-a). It has neither a Hamiltonian circuit (by the same reason graph (b) doesn't) nor an Eulerian circuit (because vertices c and d have odd degrees).

2\. Find a Trivial lower-bound class for each of the following problems and indicate, if you can, whether this bond is tight.

a) Finding the largest element in an array.

b) checking completeness of a graph represented by its adjacency matrix.

c) Generating all the subsets of an n-element set

d) Determining whether n given real numbers are all distinct.

a) All n elements of a given array need to be processed to find its largest element (otherwise, if a unprocessed element is larger than all the others, the output cannot be correct) and just one item needs be produced (if just the value of the largest element or a position of the largest element needs to be returned). Hence the trivial lower bound is linear

It is tight because the standard one-pass algorithm for this problem is in $\Theta(n)$.

b) Since the existence of an edge between all $n(n-1)/2$ pairs of vertices needs to be verified in the worst case before establishing completeness of a graph with n vertices, the trivial lower bound is linear. It is tight because this is the amount of work done by the Brute-force Algorithm that simple checks all the elements in the upper-triangular part of the matrix until either a zero is encountered or no unchecked elements are left.

c) The size of the problem's output is 2^n. Hence, the lower bound is exponential. The bound is tight because efficient algorithms for subset generation spend a constant time on each of them (except, possible, for the first one).

d) The size of the problem's input is n while the output is just one bit. Hence, the trivial lower bond is linear. It is not right : according to the known result quoted in the section, the right lower bound for this problem is n log n.

3. Compare the P, NP and NP complete problems with suitable examples :

P (Polynomial Time) :

Problems that can be solved in polynomial time by a deterministic Turing machine. These are considered efficiently solvable problems.

Examples :

Sorting Algorithm :

Algorithms like Merge Sort, Quick sort, and Heap sort operate in o(n log n) time.

Shortest path Problem:

Dijkstra's algorithm finds the shortest path in a graph with non-negative weights in $O(V^2)$ or $O(E + V \log V)$ time with a min-priority Queue.

Matrix Multiplication:

Standard Matrix Multiplication is solvable in $O(n^3)$ time, with more advanced algorithms like Strassen's reducing this to $O(n^{2.81})$.

NP (Nondeterministic Polynomial Time)

Problems for which a given solution can be verified in polynomial time by a deterministic Turing machine. These problems may or may not be solvable in polynomial time, but if a solution is presented, its correctness can be checked Quickly.

Examples :

Hamiltonian Path :

Determining whether there exists a path in a graph that visits each vertex exactly once.

Subset Sum :

Given a set of integers, determine if there is a non-empty subset whose sum is zero. Verification is straightforward if the subset is provided.

3-SAT (Boolean satisfiability problem) :

Determining if there exists an assignment of variables that makes a given Boolean Formula true. While finding such an assignment might be hard, checking if given assignment is correct is easy.

NP-complete :

Problems that are both in NP and as hard as any problem in NP. Formally, a problem X is NP-complete.

Examples :

Traveling salesman problem (Decision Version):

Given a set of cities and distances between them, determine if there is a route that visits each city exactly once and returns to the origin city with a total distance less than or equal to a given value.

3-SAT :

Already mentioned in the NP class, but also an NP-complete problem because it is among the first problems proven to be NP-complete.

Knapsack problem (0/1 knapsack):

Given a set of items, determine if there is a subset of items that fits within a given weight limit and maximizes the total value.

4. Discuss the approximation Algorithm for NP-Hard problems:

Principles of Approximation Algorithms:

1) Efficient computation:

They run in polynomial time, making them feasible for large inputs where exact algorithms would be impractical.

2) Near-optimal solutions:

They provide solutions that are close to the best possible, measured by an approximation ratio or factor.

3) Guarantee on performance:

They offer a provable guarantee on how close the solution is to the optimal one.

1) Traveling Salesman Problem (TSP)

Given a set of cities and the distances between every pair of cities, find the shortest possible route that visits each city exactly once and returns to the origin city

Approximation Algorithm :

• create a minimum spanning tree (MST) of the graph.

• Find a minimum - weight perfect matching for the vertices with odd degrees in the MST.

• combine the edges of the MST and the Matching to form an Eulerian circuit

• convert the Eulerian circuit into a Hamiltonian circuit by shortcutting repeated vertices.

• This Algorithm achieves a 3/2 - approximation, meaning the tour length is at most 1.5 times the optimal tour length.

2) Knapsack problem :

Given a set of items, each with a weight and value, determine the number of each item to include in a collection so that the total weight is within a given limit and the total value is maximized.

Fully polynomial - Time Approximation scheme (FPTAS).

- use dynamic programming to solve a relaxed v... of the problem.

- Scale down the values of the items by a factor related to the desired approximation ratio.

- use the scaled values in a dynamic programming algorithm to find an approximate solution.

- This scheme allows for an $(1-\epsilon)$ - approximate solution for any $\epsilon > 0$, with a running time polynomial in the input size and $1/\epsilon$.

5. Discuss the Approximate Algorithm for Travelling Salesman problem.

The Travelling salesman problem (TSP) is a fundamental NP-hard problem in computer science and operation research. Given a set of cities and distance between every pair of cities, the objective is to find the shortest possible route that visits each city exactly once and returns to the starting city. Because finding the exact solution is computationally infeasible for large instances, approximation algorithms are used to find near-optimal solutions efficiently.

Here, we will focus on Christofides' Algorithm, one of the most well-known approximation algorithms for TSP when the distances satisfy the triangle inequality (the direct distance between two points is always less than or equal to the sum of the distances via a third point).

Christofides' Algorithm:

Steps:

1) Construct a Minimum Spanning Tree (MST):

use Algorithms like Prim's or Kruskal's to construct an MST of the graph G. An MST is a subset of the edges that connects all vertices together without any cycles and with the minimum possible total edge weight.

2) Find a Minimum Weight Perfect Matching:

Identify all vertices with an odd degree in the MST (let's denote this set as O)

3) Combine the MST and the Matching:

• Add the edges from the Matching M to the MST. The resulting graph H is Eulerian (all vertices have even degree), as adding the matching ensures that all vertices have even degree.

4) Find an Eulerian circuit:

• Find an Eulerian circuit in H, which is a cycle that visits every edge exactly once. This can be done using algorithms like Hierholzer's.

5) Shortcut to form a Hamiltonian circuit:

• Convert the Eulerian circuit into a Hamiltonian circuit by skipping repeated vertices while traversing the Eulerian circuit. The triangle inequality guarantees that this shortcutting does not increase the tour length disproportionately.