3.2 Hamiltonian Circuit, Greedy Algorithm, and Edge-Picking Algorithm

Source: https://www.studocu.com/ph/document/central-philippine-university/bs-medical-laboratory-science/gemath-lesson-11-hamiltonian/60890683

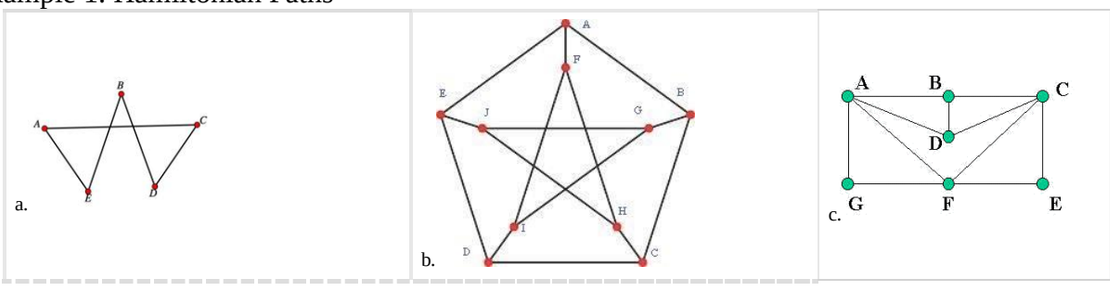
Hamiltonian Circuit

The Traveling Salesman Problem (TSP) is any problem where you must visit every vertex of a weighted graph once and only once, and then end up back at the starting vertex. Examples of TSP situations are package deliveries, fabricating circuit boards, scheduling jobs on a machine and running errands around town.

Hamilton Circuit is a circuit that must pass through each vertex of a graph once and only once.

Hamilton Path is a path that must pass through each vertex of a graph once and only once.

Example 1. Hamiltonian Paths

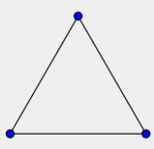
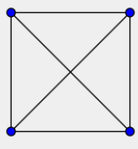
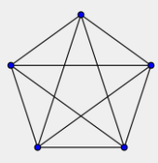


Not all graphs have a Hamilton circuit or path. There is no way to tell just by looking at a graph if it has a Hamilton circuit or path like you can with an Euler circuit or path. You must do trial and error to determine this. By the way if a graph has a Hamilton circuit, then it has a Hamilton path. Just do not go back to home.

Graph a. has a Hamilton circuit (one example is ACDBEA)
Graph b. has no Hamilton circuits, though it has a Hamilton path (one example is ABCDEJGIFH)
Graph c. has a Hamilton circuit (one example is AGFECDBA)

Complete Graph is a graph with N vertices in which every pair of vertices is joined by exactly one edge. The symbol used to denote a complete graph is K_N.

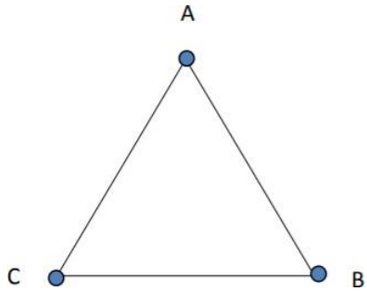
Example 2. Complete Graphs for N = 2, 3, 4, and 5

a. K_2	b. K_3	c. K_4	d. K_5
			
two vertices and one edge	three vertices and three edges	four vertices and six edges	five vertices and ten edges

In each complete graph shown above, there is exactly one edge connecting each pair of vertices. There are no loops or multiple edges in complete graphs. Complete graphs do have Hamilton circuits.

Reference Point is the starting point of a Hamilton circuit.

Example 3. Reference Point in a Complete Graph
Many Hamilton circuits in a complete graph are the same circuit with different starting points. For example, in the graph K_3, shown below, ABCA is the same circuit as BCAB, just with a different starting point (reference point). We will typically assume that the reference point is A.



Graph K3

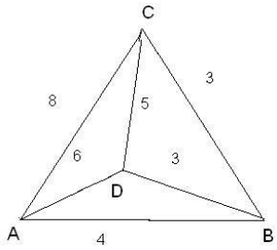
Number of Hamilton Circuits. A complete graph with N vertices is (N-1)! Hamilton circuits. Since half of the circuits are mirror images of the other half, there are actually only half this many unique circuits.

Example 4. Number of Hamilton Circuits
How many Hamilton circuits does a graph with five vertices have?
(N – 1)! = (5 – 1)! = 4! = 4*3*2*1 = 24 Hamilton circuits.

How to solve a Traveling Salesman Problem (TSP)?
A traveling salesman problem is a problem where you imagine that a traveling salesman goes on a business trip. He starts in his home city (A) and then needs to travel to several different cities to sell his wares (the other cities are B, C, D, etc.). To solve a TSP, you need to find the cheapest way for the traveling salesman to start at home, A, travel to the other cities, and then return home to A at the end of the trip. This is simply finding the Hamilton circuit in a complete graph that has the smallest overall weight. There are several different algorithms that can be used to solve this type of problem.

A. Brute Force Algorithm
1. List all possible Hamilton circuits of the graph.
2. For each circuit find its total weight.
3. The circuit with the least total weight is the optimal Hamilton circuit.

Example 5. Brute Force Algorithm



Complete Graph for Brute Force Algorithm

Suppose a delivery person needs to deliver packages to three locations and return to the home office A. Using the graph shown above (Complete Graph for Brute Force Algorithm), find the shortest route if the weights on the graph represent distance in miles.

Recall the way to find out how many Hamilton circuits this complete graph has. The complete graph above has four vertices, so the number of Hamilton circuits is:

(N – 1)! = (4 – 1)! = 3! = 3*2*1 = 6 Hamilton circuits.

However, three of those Hamilton circuits are the same circuit going the opposite direction (the mirror image).

Hamilton Circuit	Mirror Image	Total Weight (Miles)
ABCDA	ADCBA	18
ABDCA	ACDBA	20
ACBDA	ADBCA	20

The solution is ABCDA (or ADCBA) with total weight of 18 mi. This is the optimal solution.

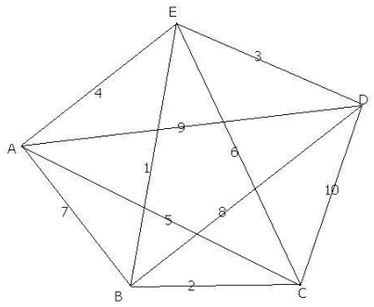
B. Nearest-Neighbor Algorithm

1. Pick a vertex as the starting point.
2. From the starting point go to the vertex with an edge with the smallest weight. If there is more than one choice, choose at random.
3. Continue building the circuit, one vertex at a time from among the vertices that have not been visited yet.
4. From the last vertex, return to the starting point.

Example 6. Nearest-Neighbor Algorithm

A delivery person needs to deliver packages to four locations and return to the home office A as shown (Complete Graph for Nearest-Neighbor Algorithm) below. Find the shortest route if the weights represent distances in miles.

Starting at A, E is the nearest neighbor since it has the least weight, so go to E. From E, B is the nearest neighbor so go to B. From B, C is the nearest neighbor so go to C. From C, the first nearest neighbor is B, but you just came from there. The next nearest neighbor is A, but you do not want to go there yet because that is the starting point. The next nearest neighbor is E, but you already went there. So go to D. From D, go to A since all other vertices have been visited.



Complete Graph for Nearest-Neighbor Algorithm

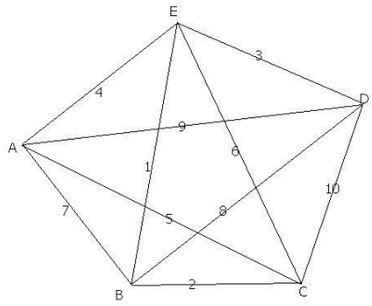
The solution is AEBCDA with a total weight of 26 miles. This is not the optimal solution, but it is close and it was a very efficient method.

C. Repetitive Nearest-Neighbor Algorithm

1. Let X be any vertex. Apply the Nearest-Neighbor Algorithm using X as the starting vertex and calculate the total cost of the circuit obtained.
2. Repeat the process using each of the other vertices of the graph as the starting vertex.
3. Of the Hamilton circuits obtained, keep the best one. If there is a designated starting vertex, rewrite this circuit with that vertex as the reference point.

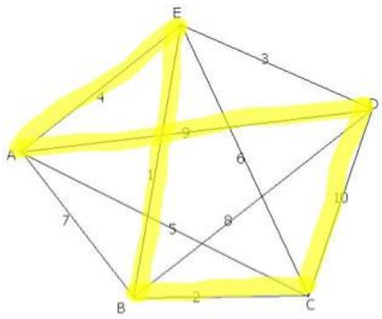
Example 7. Repetitive Nearest-Neighbor Algorithm

Suppose a delivery person needs to deliver packages to four locations and return to the home office A. Find the shortest route if the weights on the graph represent distances in kilometers.



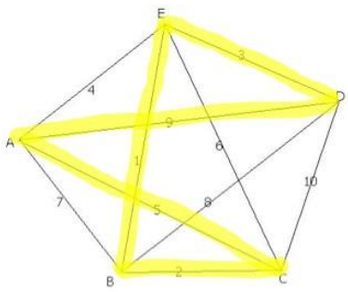
Complete Graph for Repetitive Nearest-Neighbor Algorithm

Starting at A, the solution is AEBCDA with total weight of 26 miles as we found in Example 6. See this solution below (Starting at A).



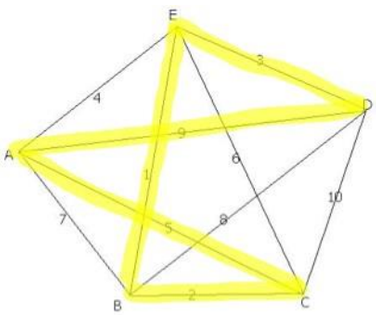
Starting at A

Starting at B, the solution is BEDACB with total weight of 20 miles.



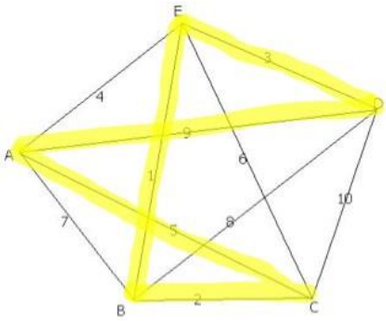
Starting at B

Starting at C, the solution is CBEDAC with total weight of 20 miles.



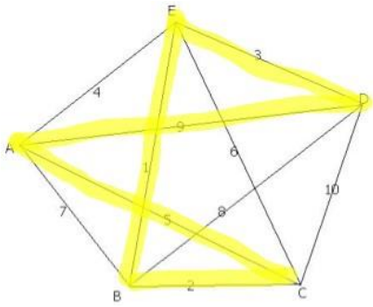
Starting at C

Starting at D, the solution is DEBCAD with total weight of 20 miles.



Starting at D

Starting at E, solution is EBCADE with total weight of 20 miles.



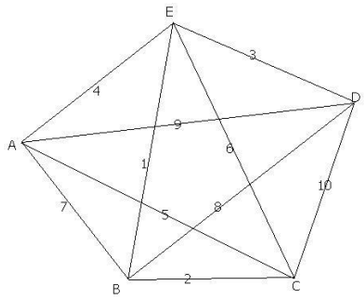
Starting at E

Now, you can compare all of the solutions to see which one has the lowest overall weight. The solution is any of the circuits starting at B, C, D, or E since they all have the same weight of 20 miles. Now that you know the best solution using this method, you can rewrite the circuit starting with any vertex. Since the home office in this example is A, let's rewrite the solutions starting with A. Thus, the solution is ACBEDA or ADEBCA.

D. Cheapest-Link Algorithm

1. Pick the link with the smallest weight first (if there is a tie, randomly pick one). Mark the corresponding edge in red.
2. Pick the next cheapest link and mark the corresponding edge in red.
3. Continue picking the cheapest link available. Mark the corresponding edge in red except when a) it closes a circuit or b) it results in three edges coming out of a single vertex.
4. When there are no more vertices to link, close the red circuit.

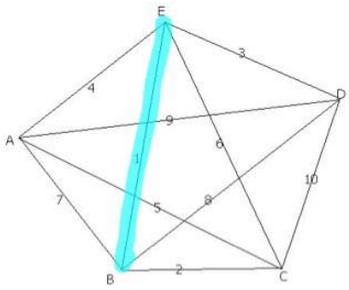
Example 8. Cheapest-Link Algorithm



Complete Graph for Cheapest-Link Algorithm

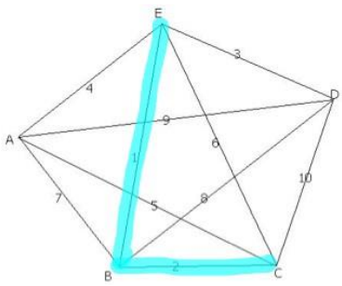
Suppose a delivery person needs to deliver packages to four locations and return to the home office A. Find the shortest route if the weights represent distances in miles.

Step 1: Find the cheapest link of the graph and mark it in blue. The cheapest link is between B and E with a weight of one mile.



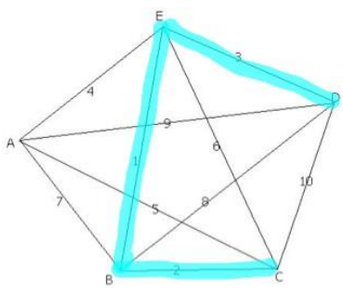
Step 1

Step 2: Find the next cheapest link of the graph and mark it in blue. The next cheapest link is between B and C with a weight of two miles.



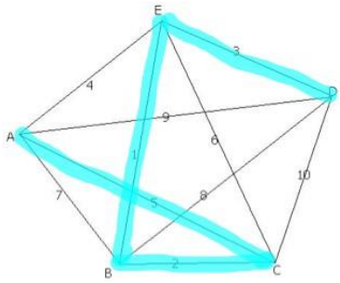
Step 2

Step 3: Find the next cheapest link of the graph and mark it in blue provided it does not make a circuit or it is not a third edge coming out of a single vertex. The next cheapest link is between D and E with a weight of three miles.



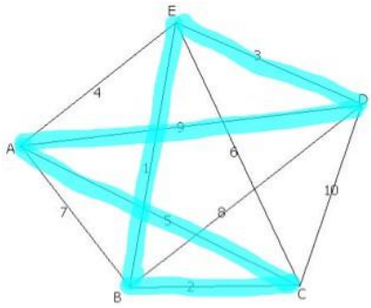
Step 3

Step 4: Find the next cheapest link of the graph and mark it in blue provided it does not make a circuit or it is not a third edge coming out of a single vertex. The next cheapest link is between A and E with a weight of four miles, but it would be a third edge coming out of a single vertex. The next cheapest link is between A and C with a weight of five miles. Mark it in blue.



Step 4

Step 5: Since all vertices have been visited, close the circuit with edge DA to get back to the home office, A. This is the only edge we could close the circuit with because AB creates three edges coming out of vertex B and BD also created three edges coming out of vertex B.



Step 5

The solution is ACBEDA or ADEBCA with total weight of 20 miles.

Efficient Algorithm: an algorithm for which the number of steps needed to carry it out grows in proportion to the size of the input to the problem.

Approximate Algorithm: any algorithm for which the number of steps needed to carry it out grows in proportion to the size of the input to the problem. There is no known algorithm that is efficient and produces the optimal solution.

Greedy Algorithm

A Greedy algorithm is an approach to solving a problem that selects the most appropriate option based on the current situation. This algorithm ignores the fact that the current best result may not bring about the overall optimal result. Even if the initial decision was incorrect, the algorithm never reverses it.

This simple, intuitive algorithm can be applied to solve any optimization problem which requires the maximum or minimum optimum result. The best thing about this algorithm is that it is easy to understand and implement.

The runtime complexity associated with a greedy solution is pretty reasonable. However, you can implement a greedy solution only if the problem statement follows two properties mentioned below:

- Greedy Choice Property: Choosing the best option at each phase can lead to a global (overall) optimal solution.

- Optimal Substructure: If an optimal solution to the complete problem contains the optimal solutions to the subproblems, the problem has an optimal substructure.

Moving forward, we will learn how to create a greedy solution for a problem that adheres to the principles listed above.

Steps for Creating a Greedy Algorithm
By following the steps given below, you will be able to formulate a greedy solution for the given problem statement:

- Step 1: In a given problem, find the best substructure or subproblem.
- Step 2: Determine what the solution will include (e.g., largest sum, shortest path).
- Step 3: Create an iterative process for going over all subproblems and creating an optimum solution.

Let's take up a real-world problem and formulate a greedy solution for it.

Problem: Alex is a very busy person. He has set aside time T to accomplish some interesting tasks. He wants to do as many tasks as possible in this allotted time T. For that, he has created an array A of timestamps to complete a list of items on his itinerary.

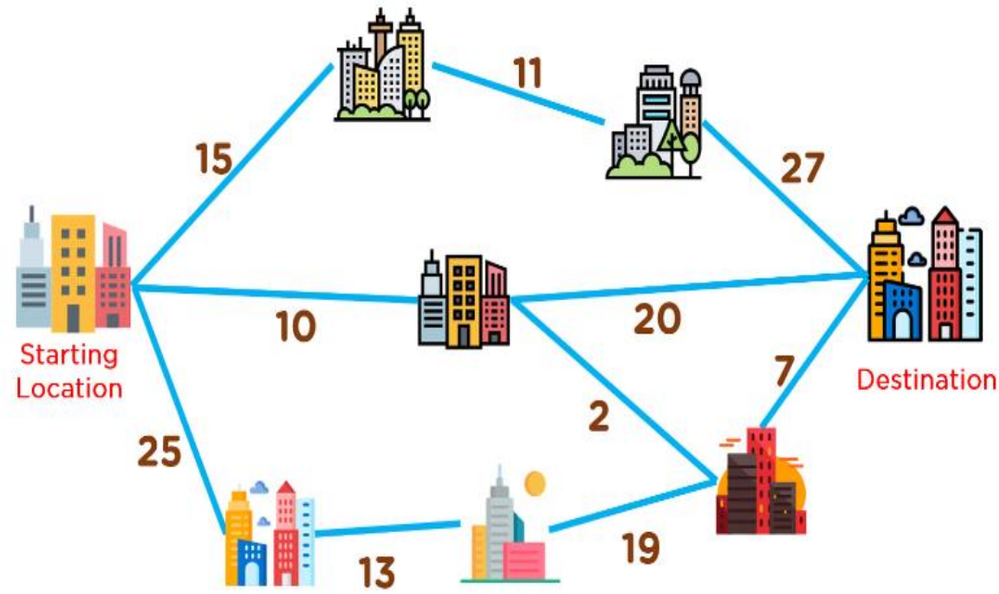
Now, here we need to figure out how many things Alex can complete in the T time he has.

Approach to Build a Solution: This given problem is a straightforward greedy problem. In each iteration, we will have to pick the items from array A that will take the least amount of time to accomplish a task while keeping two variables in mind: current_Time and number_Of_Things. To generate a solution, we will have to carry out the following steps.

- Sort the array A in ascending order.
- Select one timestamp at a time.
- After picking up the timestamp, add the timestamp value to current_Time.
- Increase number_Of_Things by one.
- Repeat steps 2 to 4 until the current_Time value reaches T.

Example of Greedy Algorithm

Problem Statement: Find the best route to reach the destination city from the given starting point using a greedy method.



Greedy Solution: In order to tackle this problem, we need to maintain a graph structure. And for that graph structure, we'll have to create a tree structure, which will serve as the answer to this problem. The steps to generate this solution are given below:

- Start from the source vertex.
- Pick one vertex at a time with a minimum edge weight (distance) from the source vertex.
- Add the selected vertex to a tree structure if the connecting edge does not form a cycle.
- Keep adding adjacent fringe vertices to the tree until you reach the destination vertex.

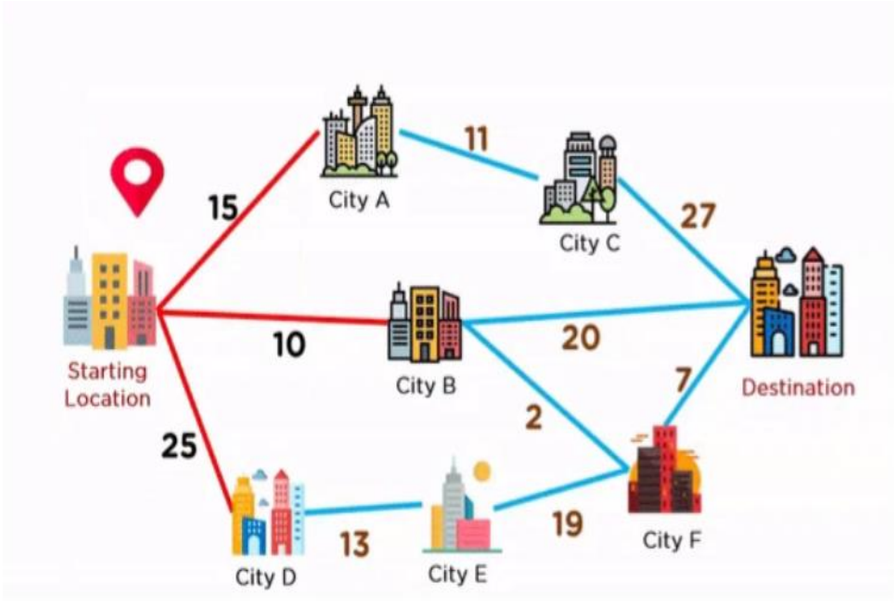


Figure 1.

Figure 2.

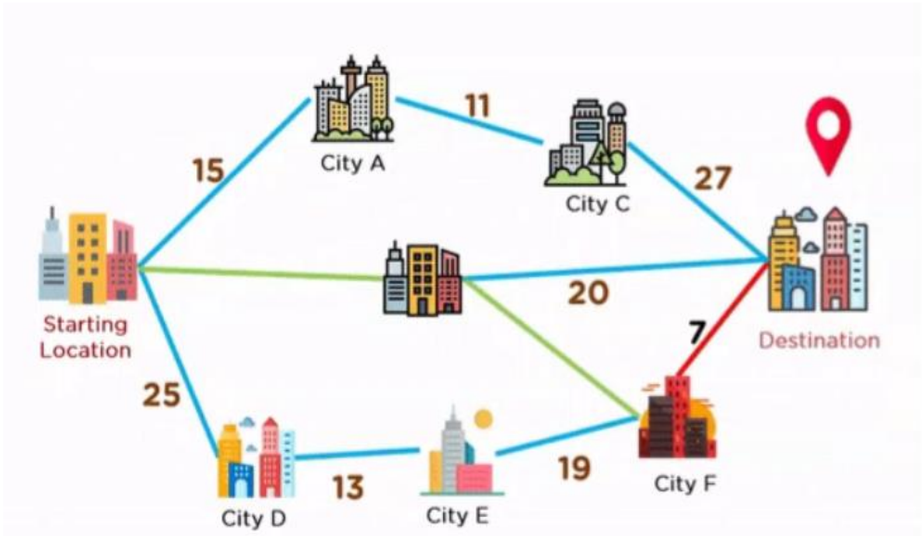


Figure 3.

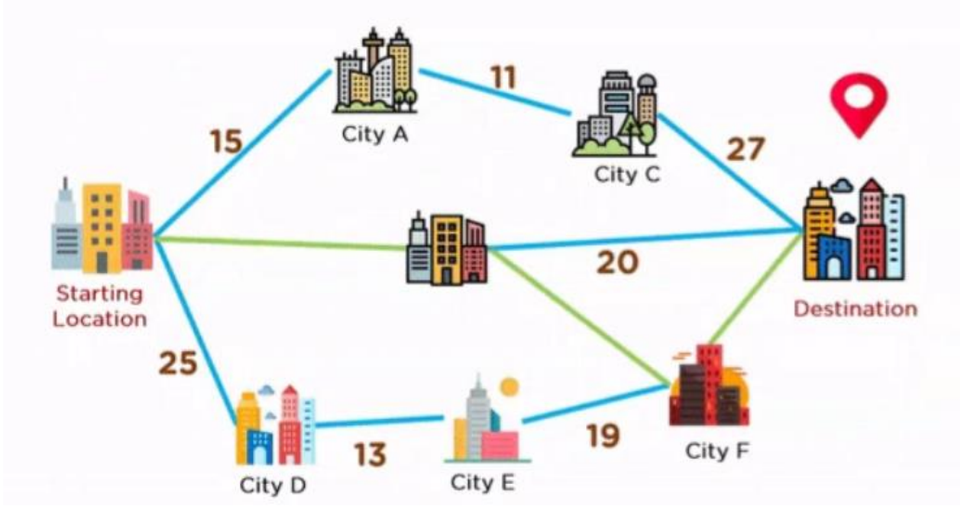


Figure 4.

Limitations of Greedy Algorithm

Factors listed below are the limitations of a greedy algorithm:

1. The greedy algorithm makes judgments based on the information at each iteration without considering the broader problem; hence it does not produce the best answer for every problem.
2. The problematic part for a greedy algorithm is analyzing its accuracy. Even with the proper solution, it is difficult to demonstrate why it is accurate.
3. Optimization problems (Dijkstra's Algorithm) with negative graph edges cannot be solved using a greedy algorithm.

Edge-Picking Algorithm

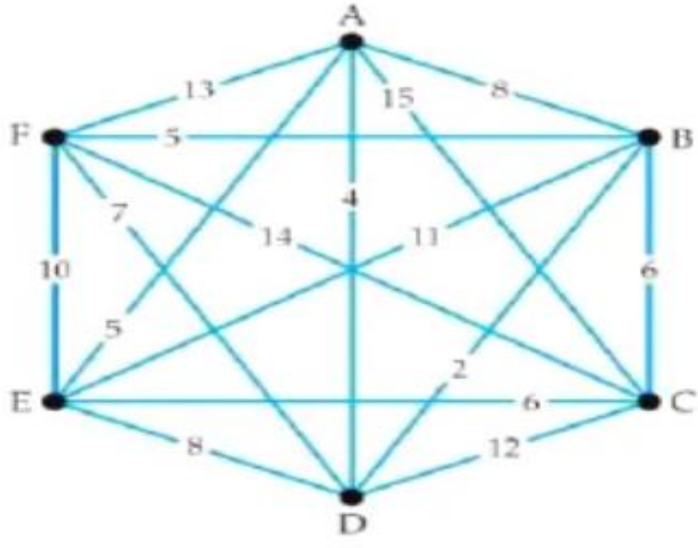
The edge-picking algorithm states to mark the edge that has the smallest weight in the complete graph. Then, the edge with the next smallest weight is marked as long as it does not complete a circuit and does not add a third marked edge to a single vertex. This process continues till no longer an edge can be marked.

How do you solve edge-picking algorithm?

1. Choose a vertex as a starting point, and travel. along the edge that has the smallest weight.
2. After arriving at the next vertex, travel along the edge of smallest weight that connects to a vertex that has not yet been visited.
3. Continue until all vertices have been visited.
4. Return to the starting vertex.

Example of Edge-Picking algorithm

Use the edge-picking to find the Hamiltonian circuit and the weight (shortest distance) of it in the figure below.



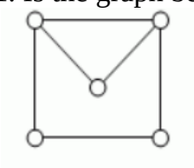
Solution:

The Hamiltonian circuit is A-D-B-F-E-C-A.

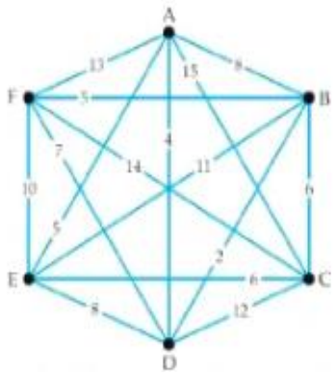
The weight of the circuit is 4 + 2 + 5 + 10 + 6 + 15 = 42

Exercise 3.2 Hamiltonian Circuit, Greedy Algorithm, & Edge-Picking Algorithm

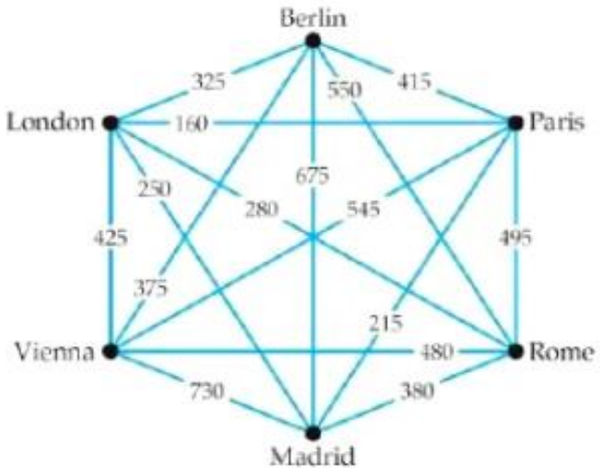
1. Is the graph below a Hamiltonian Circuit? Justify your answer.



2. Consider the graph below. Identify the Hamiltonian Circuit and get the total weight of it.



3. A traveler might be more interested in the cost of flights than the time or distance between cities. Consider the graph below. Use the greedy algorithm, start at London, and get the total airfare for the trip.



4. & 5. The cost of flying between various European cities is shown in the table below.

	London, England	Berlin, Germany	Paris, France	Rome, Italy	Madrid, Spain	Vienna, Austria
London, England	—	$325	$160	$280	$250	$425
Berlin, Germany	$325	—	$415	$550	$675	$375
Paris, France	$160	$415	—	$495	$215	$545
Rome, Italy	$280	$550	$495	—	$380	$480
Madrid, Spain	$250	$675	$215	$380	—	$730
Vienna, Austria	$425	$375	$545	$480	$730	—

Use both the greedy algorithm and the edge-picking algorithm to find a low-cost route that visits each city just once and starts and ends in London. Which route is economical?