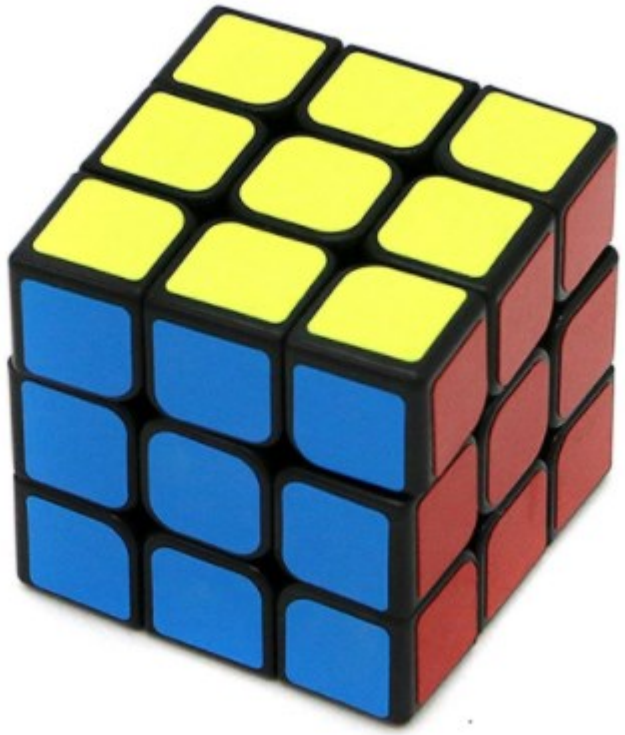
Brute Force Algorithm Paradigm


1 8 5
0 7 4
9 6 3
1 8 5
2 3 4
9 6 3



43 quintillion — **43,000,000,000,000,000,000**

- This algorithm simply tries all possibilities until a satisfactory solution is found.
- Such an algorithm can be:
- **Optimizing:** Find the best solution this may require finding all solutions, or if a value for the best solution is known, it may stop when any best solution is found.
- Ex: Find the best path for a travelling salesman .
- **Satisficing:** Stop as soon as a solution is found that is good enough.
- Ex: Finding a travelling salesman path which is found that is good enough.

Improvement

- Often brute force algorithm required exponential time
- Various heuristic and optimizations can be used
- Heuristic: A rule of thumb that helps you decide which possibilities to look at first.
- Optimization: In this case a way to eliminate certain possibilities without fully exploring them.

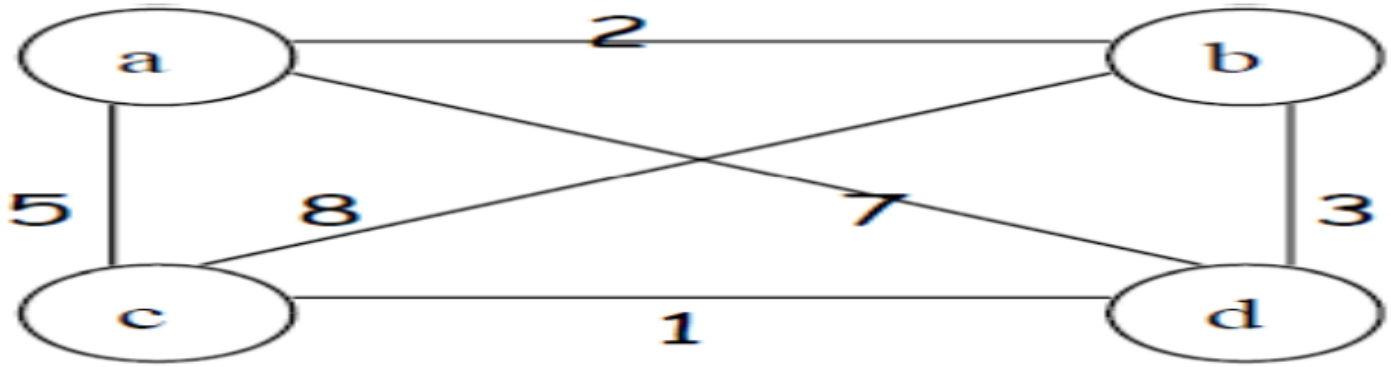
Definition

- Brute force search or exhaustive search also known as generate and test, is a very general problem solving techniques and algorithm paradigm that consist of systematically enumerating all possibilities for the solution and checking whether each candidate satisfies the problem statement.

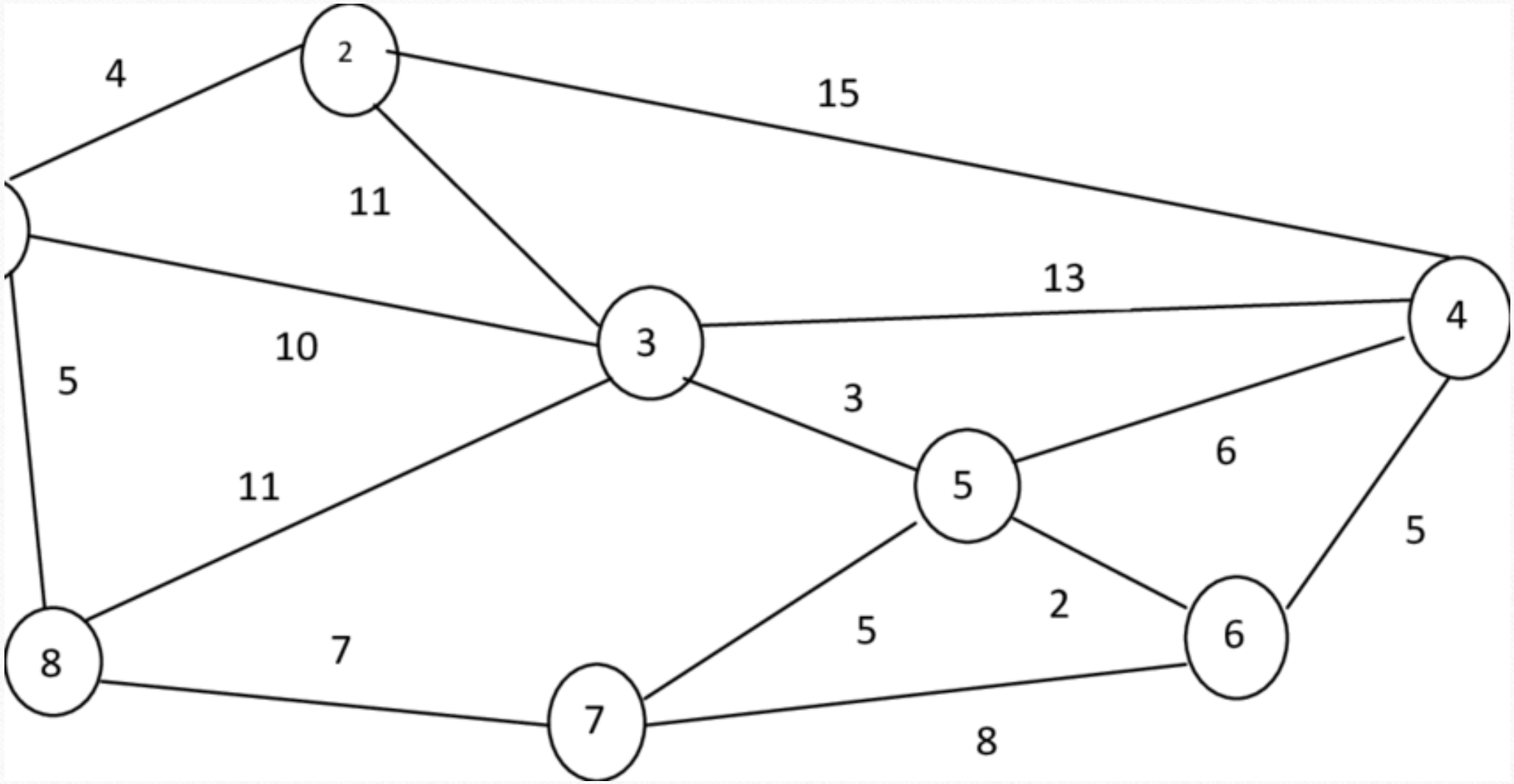
Traveling salesman problem

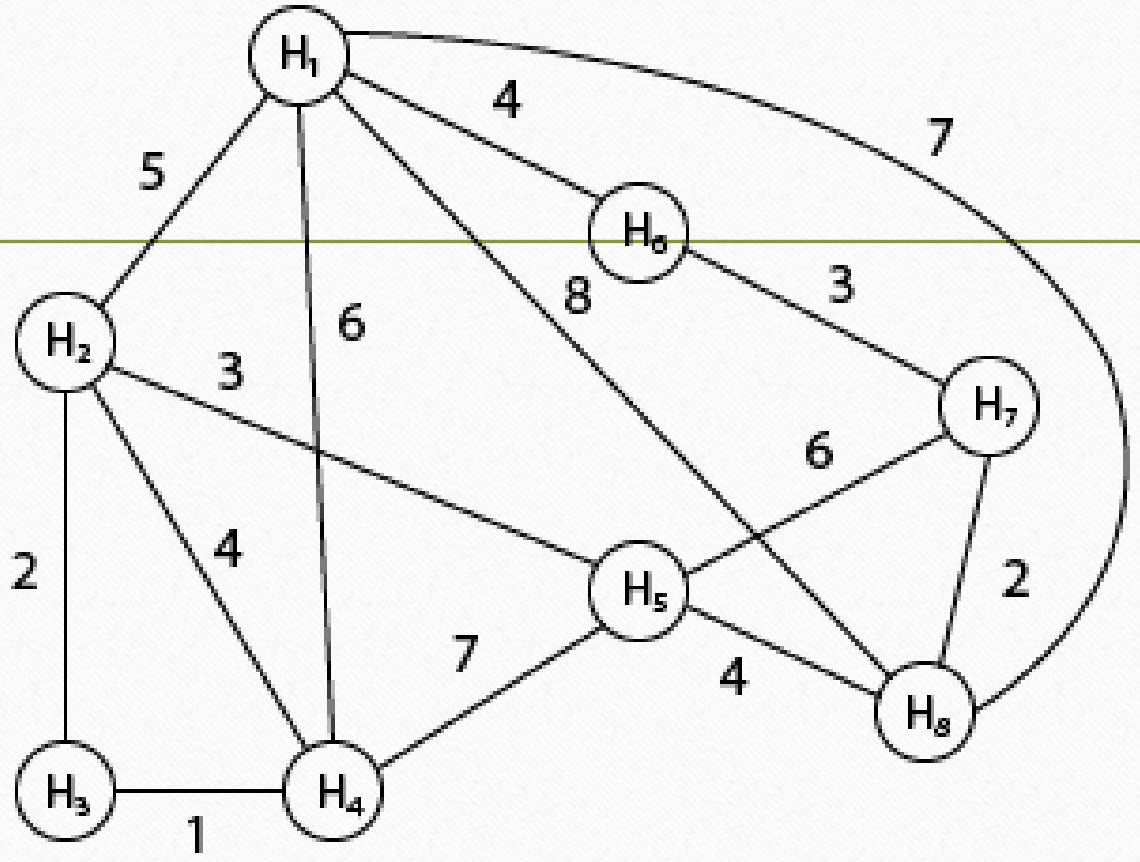
- The problem asks to find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it started.
- The problem can be stated as the problem of finding the shortest Hamiltonian circuit of the graph-which is a weighted graph, with the graph's vertices representing the cities and the edge weights specifying the distance.
- Hamiltonian circiut is defined as a cycle that passes thru all the vertices of the graph exactly once.

- The Hamiltonian circuit can also be defined as a sequence of n+1 adjacent vertices vi^0, vi^1 , …. Vi^{n-1} , where vi^0, the first vertex of the sequence is the same as the last one while all other n-1 vertices are distinct.
- Obtain the tours by generating all the permutations of n-1 intermediate cities, compute the tour lengths, and find the shortest among them.



Tour	Length	
a → b → c → d → a	l = 2 + 8 + 1 + 7 = 18	
a → b → d → c → a	l = 2 + 3 + 1 + 5 = 11	optimal
a → c → b → d → a	l = 5 + 8 + 3 + 7 = 23	
a → c → d → b → a	l = 5 + 1 + 3 + 2 = 11	optimal
a → d → c → b → a	l = 7 + 3 + 8 + 5 = 23	


2
4
15
11
13
4
3
10
5
3
6
5
11
5
2
6
8
7
5
7
8


H1
H2
H3
H4
H5
H6
H7
H8
4
7
5
8
3
6
3
6
2
4
2
7
4
1

- Consider the condition that the vertex B precedes C then, The total no of permutations will be (n-1)! /2, which is impractical except for small values of n. On the other hand, if the starting vertex is not considered for a single vertex, the number of permutations will be even large for n values.

Knapsack problem

- The problem states that: given n items of known weights w^1 ,w^2 , … w^n and values v^1 , v^2 ,…, v^n and a knapsack of capacity w, find the most valuable subset of the items that fit into the knapsack. Eg consider a transport plane that has to deliver the most valuable set of items to a remote location without exceeding its capacity.

- Example: W=10, w^1 ,w^2 , w^3 , w^4 = { 7,3,4,5 } and v^1 ,v^2 , v^3 , v^4 = { 42,12,40,25 }

Subset	Total weight	Total value
∅	0	0
{ 1 }	7	$42
{ 2 }	3	$12
{ 3 }	4	$40
{ 4 }	5	$25
{ 1,2 }	10	$54
{ 1,3 }	11	Not feasible
{ 1,4 }	12	Not feasible
{ 2,3 }	7	$52
{ 2,4 }	8	$37
{ 3,4 }	9	$65
{ 1,2,3 }	14	Not feasible
{ 1,2,4 }	15	Not feasible
{ 1,3,4 }	16	Not feasible
{ 2,3,4 }	12	Not feasible
{ 1,2,3,4 }	19	Not feasible

- This problem considers all the subsets of the set of n items given, computing the total weight of each subset in order to identify feasible subsets(i.e., the one with the total weight not exceeding the knapsack capacity) and finding the largest among the values, which is an optimal solution.
- The number of subsets of an n-element set is 2^n the search leads to a $\Omega(2^n)$ algorithm, which is not based on the generation of individual subsets

- Thus, for both TSP and knapsack, exhaustive search leads to algorithms that are inefficient on every input. These two problems are the best known examples of NP-hard problems. No polynomial-time algorithm is known for any NP-hard problem. The two methods Backtracking and Branch & bound enable us to solve this problem in less than exponential time

Objects	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1

m = 7

n = 15

Assignment problem

- The problem is: given n people who need to be assigned to execute n jobs, one person per job. The cost if the i^{th} person is assigned to the j^{th} job is a known quantity c[i,j] for each pair i,j=1,2,….,n. The problem is to find an assignment with the smallest total cost.

Example:

	Job1	Job2	Job3	Job4
Person1	9	2	7	8
Person2	6	4	3	7
Person3	5	8	1	8
Person4	7	6	9	4

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

‹1,2,3,4› cost = 9 + 4 + 1 + 4 = 18

‹1,2,4,3› cost = 9 + 4 + 8 + 9 = 30

‹1,3,2,4› cost = 9 + 3 + 8 + 4 = 24

‹1,3,4,2› cost = 9 + 3 + 8 + 6 = 26 etc.