Marks Distribution

Project Demonstration (Marks: 40%)

i. Compilation of code without error (Documentation and indentation in the source code must be maintained)
ii. Testing the project with different inputs.

Presentation (Marks: 30%)

i. Content of your presentation
ii. Presentation skill

Report (Marks: 30%)

i. Problem Statement:
   Write a brief statement of the project in general terms according to the problem description provided. Problem statement should not include any example or detailed description.

ii. System Requirements:
   Processor, RAM, Operating system, IDE

iii. System Design:
   Describe the algorithm of the project.

iv. Implementation:
   Write and explain important parts of the code. Avoid explaining general parts such as declaration or initialization of variables.

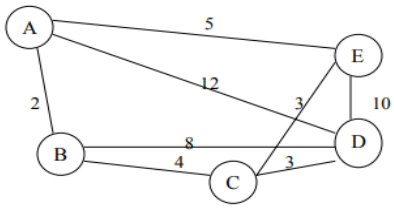
v. Testing Results:
   Give the screenshots of the output and describe the inputs and outputs in one or two sentences.

vi. Future Scope:
   Mention the limitations (if any) and future scope of your project.

CSE 245: Algorithm

Project: Travelling Salesman Problem

A traveller needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city? Given a set of cities and the cost of travel (or distance) between each possible pair, you have to find the best possible way of visiting all the cities and returning to the starting point that minimizes the travel cost (or travel distance). Consider the following set of cities:



The problem lies in finding a minimal path passing from all vertices once. For example, the path Path1 {A, B, C, D, E, A} and the path Path2 {A, B, C, E, D, A} pass all the vertices but Path1 has a total length of 24 and Path2 has a total length of 31.

In this project, you have to write a conde in C/C++ that solves the travelling salesman problem and shows the best path along with the minimal cost. You should use **Dynamic Programming** or a recursive algorithm based on **Backtracking** approach in this regard as they have the ability to give same result in far fewer attempts than Brute Force method trials and thus handles the complexity better.

Output:

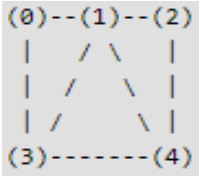
```
                                    Travelling Salesman Problem
                                ------------------------------------------------------

Enter the number of cities: 4

Enter the Cost Matrix

Enter Elements of Row 1:
0 4 1 3

Enter Elements of Row 2:
4 0 2 1

Enter Elements of Row 3:
1 2 0 5

Enter Elements of Row 4:
3 1 5 0


The cost list is:
        0       4       1       3
        4       0       2       1
        1       2       0       5
        3       1       5       0

The Path is:
1--->3--->2--->4--->1

Minimum cost is 7

Process returned 0 (0x0)   execution time : 32.863 s
Press any key to continue.
```

CSE 245: Algorithm

Project: Hamiltonian Cycle

A Hamiltonian cycle is a closed loop on a graph where every node is visited exactly once. It is a Hamiltonian Path such that there is an edge from the last vertex to the first vertex of the path. Determine whether a given graph contains Hamiltonian Cycle or not. If it contains, then what is the path?

For example, a Hamiltonian Cycle in the following graph is {0, 1, 2, 4, 3, 0}. There are more Hamiltonian Cycles in the graph like {0, 3, 4, 2, 1, 0}.



In this project, you have to design a suitable algorithm that finds all the Hamiltonian cycles present in a specific graph and the Hamiltonian path accordingly. You should use **Dynamic Programming** or a recursive algorithm based on **Backtracking** approach in this regard as they have the ability to give same result in far fewer attempts than Brute Force method trials and thus handles the complexity better.

Input:

A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

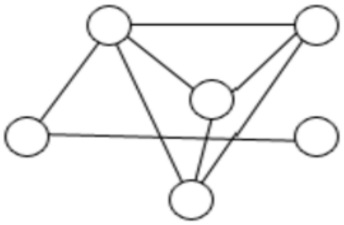
Output:

An array path[V] that should contain the Hamiltonian Path. The path[i] should represent the i^{th} vertex in the Hamiltonian Path. The code should also return false if there is no Hamiltonian Cycle in the graph.

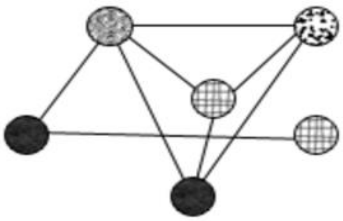
CSE 245: Algorithm

Project: Graph Coloring Problem

Graph Colouring problem is to assign colours to certain elements of a graph subject to certain constraints. The problem is, given an undirected graph and a number m, determine if the graph can be coloured with at most m colours such that no two adjacent vertices of the graph are coloured with a same colour. Here colouring of a graph means assignment of colours to all vertices. For example,



Non-coloured



Coloured

The idea is to assign colours one by one to different vertices, starting from the vertex 0. Before assigning a colour, check for safety by considering already assigned colours to the adjacent vertices.

In this project, you have to design a suitable algorithm that solves the Graph Colouring problem generating all possible combinations of colours with the given constraints. You should use a **Greedy** approach or a recursive algorithm based on **Backtracking** approach in this regard as they have the ability to give same result in far fewer attempts than Brute Force method trials and thus handles the complexity better

Input:

1) A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.
2) An integer m which is maximum number of colors that can be used.

Output:

An array color[V] that should have numbers from 1 to m. Array color[i] should represent the color assigned to the ith vertex. The code should also return false if the graph cannot be colored with m colors.

CSE 245: Algorithm

Project: N-Queen Problem

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, to solve the eight queens puzzle (when N=8), the problem is to place the eight chess queens on an 8×8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. To solve the problem, keep a simple rule: last placed, first displaced. In other words, if you place successfully a queen on the i^{th} column but cannot find solution for $(i+1)^{th}$ queen, then going backwards you will try to find other admissible solution for the i^{th} queen first. This is a Backtrack approach where you try to build a solution one step at a time. If at some step it becomes clear that the current path that you are on cannot lead to a solution, you go back to the previous step (backtrack) and choose a different path. Briefly, once you exhaust all your options at a certain step you go back.

In this project, you have designed a suitable algorithm that solves the N-queen problem and shows all possible placements of N queens. You have to use **Backtracking** algorithm in this regard. You should not use **Brute Force** as using it will increase the runtime and complexity with the increasing value of N.

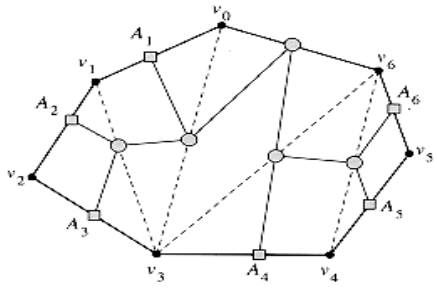
Input:
Number of queens or the value of N.
Output:
All possible placements of the queens. You can show it using matrix.

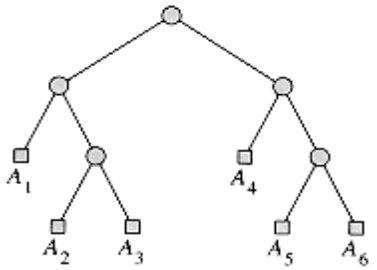
CSE 245: Algorithm

Project: Optimal Polygon Triangulation

A polygon is a two-dimensional closed shape defined by connections between points or vertices.A triangulation of a polygon can be thought of as a set of chords that divide the polygon into triangles such that no two chords intersect (except possibly at a vertex). It can be formed by drawing diagonals between non-adjacent vertices such that the diagonals never intersect. The cost of a triangulation is sum of the weights of its component triangles. Weight of each triangle is its perimeter (sum of lengths of all sides). An optimal triangulation is one that minimizes some cost function of the triangles.

The idea is to divide the polygon into three parts: a single triangle, the sub-polygon to the left, and the sub-polygon to the right. Try all possible divisions like this until you find the one that minimizes the cost of the triangle plus the cost of the triangulation of the two sub-polygons. You can get the cost of triangulation of the two sub-polygons recursively. The base case of the recursion is a line segment (i.e., a polygon with zero area), which has cost 0.





In this project, you have to design a suitable algorithm implementing the Optimal Polygon Triangulation. Show the triangulations along with the minimal cost. You should use **Dynamic Programming** in this regard. Avoid using **Brute Force** as it may take exponential time, while dynamic programming is much faster in comparison.

CSE 245: Algorithms

Project: Matrix Chain Multiplication

Given a sequence of matrices, find the most efficient way to multiply these matrices together. However, the problem is not actually to perform the multiplications, rather you have to decide in which order the multiplications should be performed.

Since matrix multiplication is associative, there are so many options to multiply a chain of matrices. In other words, no matter how we parenthesize the product, **the result will be the same**. For example, if we had four matrices A, B, C, and D, we would have:

A(BCD) = (AB)(CD) = (ABC)D = ...

However, the order in which we parenthesize the product **affects the number of simple arithmetic operations** needed to compute the product, or the efficiency. For example, suppose A is a 10 × 30 matrix, B is a 30 × 5 matrix, and C is a 5 × 60 matrix. Then,

1. (AB)C = (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations
2. A(BC) = (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations.

Clearly the first parameterization requires less number of operations.

Use Dynamic Programming (DP) approach to find the best association such that the result is obtained with the minimum number of arithmetic operations.

CSE 245: Algorithms

Project: Optimal Binary Search Tree

Crispher is a very good coder. After reading anything he tries to implement it in code. About a week ago he was reading about Optimal Binary Search tree. Now he wants to implement it in the code but he finds some problem to do it. His solution was greedy as well as complexity was high. So he wants someone who has good knowledge in Algorithms and knows how to minimize complexity. He stated the problem as below:

Given a sorted array keys [0... n-1] of search keys and an array freq[0... n-1] of frequency counts, where freq[i] is the number of searches to keys[i]. Construct an optimal binary search tree of all keys such that the total cost of all the searches is as small as possible. For example:

Input: keys[] = {10, 12, 20}, freq[] = {34, 8, 50}

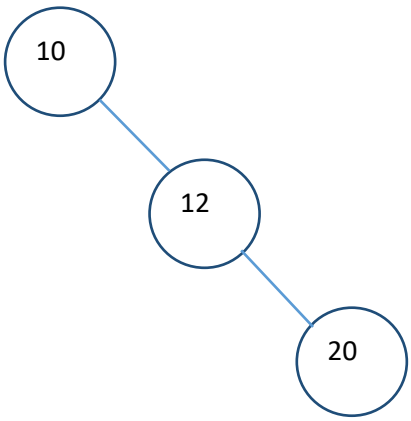


Figure 1: Normal Greedy Solution

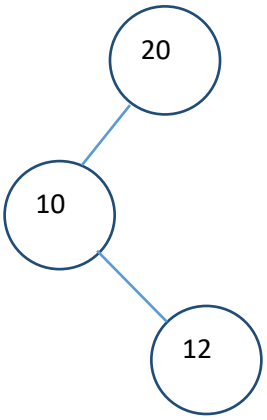


Figure 2: Optimal Solution

Now can you help Crispher to make it optimal and dynamic?

Use Dynamic Programming (DP) approachto find the optimal solution of the above stated problem. Keep in mind that Brute Force solution is not alloed.

CSE 245: Algorithm

Project: Johnson's Algorithm

Johnson's algorithm is a way to find the shortest paths between all pairs of vertices in a sparse, edge-weighted, directed graph. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph.

In this project, you have to v that finds shortest paths between every pair of vertices in a given weighted directed Graph where weights may be negative as well using by implementing **Johnson's Algorithm**. You should not use Brute Force as it may increase the complexity and runtime of the program.

Input:

A graph with general edge weights (can be negative) with no negative cycles.

Output:

The shortest (u, v) path for all pairs of vertices (u, v). If the input graph has any negative cycles, the program will report this and halt

CSE 245: Algorithms

Project: Sequence Alignment Problem

In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. It is a fundamental problem in Biological Science.

Given as an input two strings, X = x_1 x_2....x_m, and Y = y_1 y_2....y_m, output the alignment of the strings, character by character, so that the net penalty is minimized. The penalty is calculated as:

1. A penalty of P_{gap} occurs if a gap is inserted between the strings.

2. A penalty of P_{xy} occurs for miss-matching the characters of X and Y.

For Example:

Test Case 1:	Input: X = CG, Y = CA, p_gap = 3, p_xy = 7
	Output: X = CG_, Y = C_A, Total penalty = 6
Test Case 2:	Input: X = AGGGCT, Y = AGGCA, p_gap = 3, p_xy = 2
	Output: X = AGGGCT, Y = A_GGCA, Total penalty = 5
Test Case 3:	Input: X = CG, Y = CA, p_gap = 3, p_xy = 5
	Output: X = CG, Y = CA, Total penalty = 5

Write a C/C++ program to find the maximum match of the given strings. Use Dynamic programming to solve the problem. Keep in mind that Brute Force solution is not allowed.

Input:

String 1 = "AGCAGTGT";

String 2 = "ACGTATC";

Output:

Minimum penalty aligning two problem is: 9

Aligned as:

AGCAGTGT_

A_C_GTATC

CSE 245: Algorithm

Project: Clique Problem

The Clique problem is the computational problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph. It has several different formulations depending on which cliques, and what information about the cliques, should be found. Common formulations of the clique problem include finding a maximum clique (a clique with the largest possible number of vertices), finding a maximum weight clique in a weighted graph, listing all maximal cliques (cliques that cannot be enlarged), and solving the decision problem of testing whether a graph contains a clique larger than a given size.

In this project, you have to design a suitable algorithm to find the cliques of a given size in a graph. You should not use Brute Force as it may affect the complexity and runtime of the program.

Input:

A graph with n number of vertices.

Output:

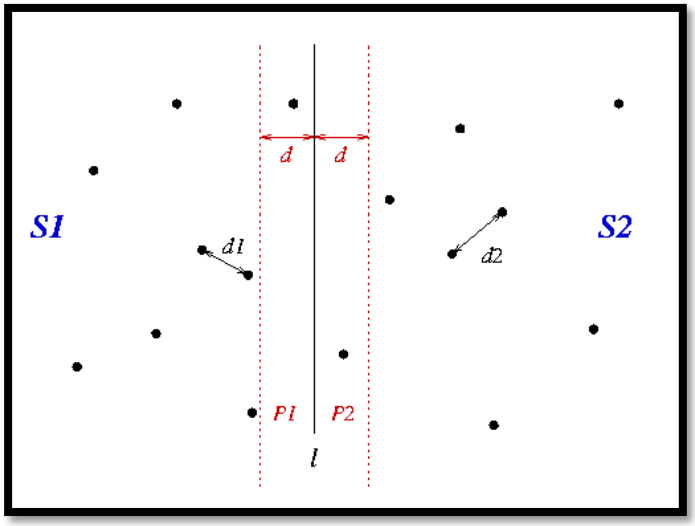
A maximum clique of size k.

CSE 245: Algorithms

Project: Closest Pair of Points

East West University authority suddenly noticed that the structure of the building is becoming weak. It tilted in the south side. Authority becomes tensed and wanted to fix it. They decided that they will take sample from each pillar and they will test it in their own campus. They were using divide and conquer algorithm to do that. But unfortunately, they were using brute force approach which increases the complexity and runtime of the program. At the meantime, Robin a student of algorithm course decided to help East West University.

Help Robin to make such an algorithm that will give real-time decision. Robin finds a formula and a graph which might help you.



$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

design a suitable algorithm to find the closest pair using Dynamic Programming approach. Also discuss about the future scope closest pair of points.

Input:

An array of n points in the plane.

Output:

The smallest distance between two points in the given array.

CSE 245: Algorithm

Project: Maximum Sum Interval

The maximum subarray problem is the task of finding the contiguous subarray within a one-dimensional array of numbers which has the largest sum. The list usually contains both positive and negative numbers along with 0. Some properties of this problem are:

1. If the array contains all non-positive numbers, then the solution is the number in the array with the smallest magnitude.
2. If the array contains all non-negative numbers, then the problem is trivial and the maximum sum is the sum of all the elements in the list.
3. An empty sets not valid.
4. There can be multiple different sub-arrays that achieve the same maximum sum to the problem.

For example, in the array [-5, 6, 7, 1, 4, -8, 16], the maximum sum is 26. That is because adding 6 + 7 + 1 + 4 + -8 + 16 gives us 26.

In this project, you have to design a suitable algorithm to find the maximum sum interval for a given array. You can use **Divide and Conquer** or **Dynamic Programming** in this regard. You should not use Brute Force as it may affect the complexity and runtime of the program.

Input:

An array with n number of index.

Output:

The maximum sum interval for the given array.

CSE 245: Algorithms

Project: Maze Path Finding

A maze is a path or collection of paths, typically from an entrance to a goal. The word is used to refer both to branching tour puzzles through which the solver must find a route, and to simpler non-branching ("unicursal") patterns that lead unambiguously through a convoluted layout to a goal. Generally, a Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e., maze [0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. Possible move is forward and down.

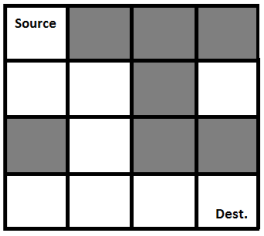


Figure 1: A Simple Maze

Gray blocks are dead ends. Now design a suitable algorithm to find the path from source to destination. You have to use backtracking approach to find the path. Also keep in mind that, Brute Force solution is not allowed.

Inputs:

```
{1, 0, 0, 0}
{1, 1, 0, 1}
{0, 1, 0, 0}
{1, 1, 1, 1}
Where 1 is the path and 0 is the wall(block).
```

Outputs:

```
{1, 0, 0, 0}
{1, 1, 0, 0}
{0, 1, 0, 0}
{0, 1, 1, 1}
```

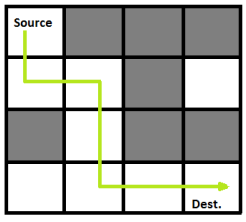


Figure 2: Solution of Figure 1

Future Scope: Modify the algorithm such that the object in the maze can move four directions.