SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

PRESENTATION

Route Finder

Using Travel salesman Algorithm

NAME :— GAURAV GUPTA

BRANCH :— Btech CSE specialization (AI & ML)

DEPATMENT :— CINTEL

SUBJECT :— **Design and Analysis of Algorithms**

REG. NO. :— 2211026010284

CONTENT

- INTRODUCTION
- PROBLEM STATEMENT
- REAL TIME PROBLEM
- CODE SNIPPETS
- MODULES USED
- WORKFLOW
- SCREENSHOTS
- CONCLUSION

PROBLEM STATEMENT

PROBLEM STATEMENT

The essence of the TSP revolves around finding the most efficient route or path that connects a set of points while minimizing total travel distance or cost. This problem finds practical applications in various fields, including logistics for route planning, manufacturing processes, and even DNA sequencing. The challenge intensifies with the increase in the number of cities, as the solution space expands factorially, highlighting the problem's NP-hard nature.

INTRODUCTION

In the realm of combinatorial optimization, the Traveling Salesman Problem (TSP) stands out as a cornerstone, challenging scholars and practitioners alike. This problem asks a seemingly simple question: "Given a list of cities and the distances between them, what is the shortest possible route that visits each city exactly once and returns to the starting city?" Despite its straightforward presentation, the TSP encapsulates a complex optimization dilemma, making it a fascinating subject for exploration in computer science, logistics, and operational research.

ALGORITHM ANALYSIS

❖ BRUTE FORCE

Algorithm Complexity:

Time Complexity: The time complexity of the algorithm is **O(n². 2")**, where n is the number of nodes. This is because there are n nodes, and for each node, the function 'fun' is called 2^n times due to the bitmask.

Space Complexity: The space complexity is **O(n 2")** due to the memorization array 'memo`.

❖ DYNAMIC PROGRAMMING

Held-Karp Algorithm:

The Held-Karp Algorithm, named after Richard Held and Richard M. Karp, is a dynamic programming solution to the Traveling Salesman Problem (TSP). It's one of the most significant early breakthroughs in solving TSP efficiently for relatively small sets of cities. The essence of this algorithm is to transform the TSP into a problem of managing and combining smaller subproblems, thereby avoiding the redundant computation associated with the brute-force approach.

❖ RECURRANCE RELATION

Recurrence Relation

The algorithm defines a recurrence relation to find the minimum cost of visiting each subset of cities exactly once and ending at a specified city. The recurrence relation is as follows:

$$d(S, i) = \min_{j \in S, j \neq i} \{d(S \setminus \{i\}, j) + dist(j, i)\}$$

Best Case Complexity: O(N^2.2^N)
Worst Case Complexity: O(N^2.
2^N)

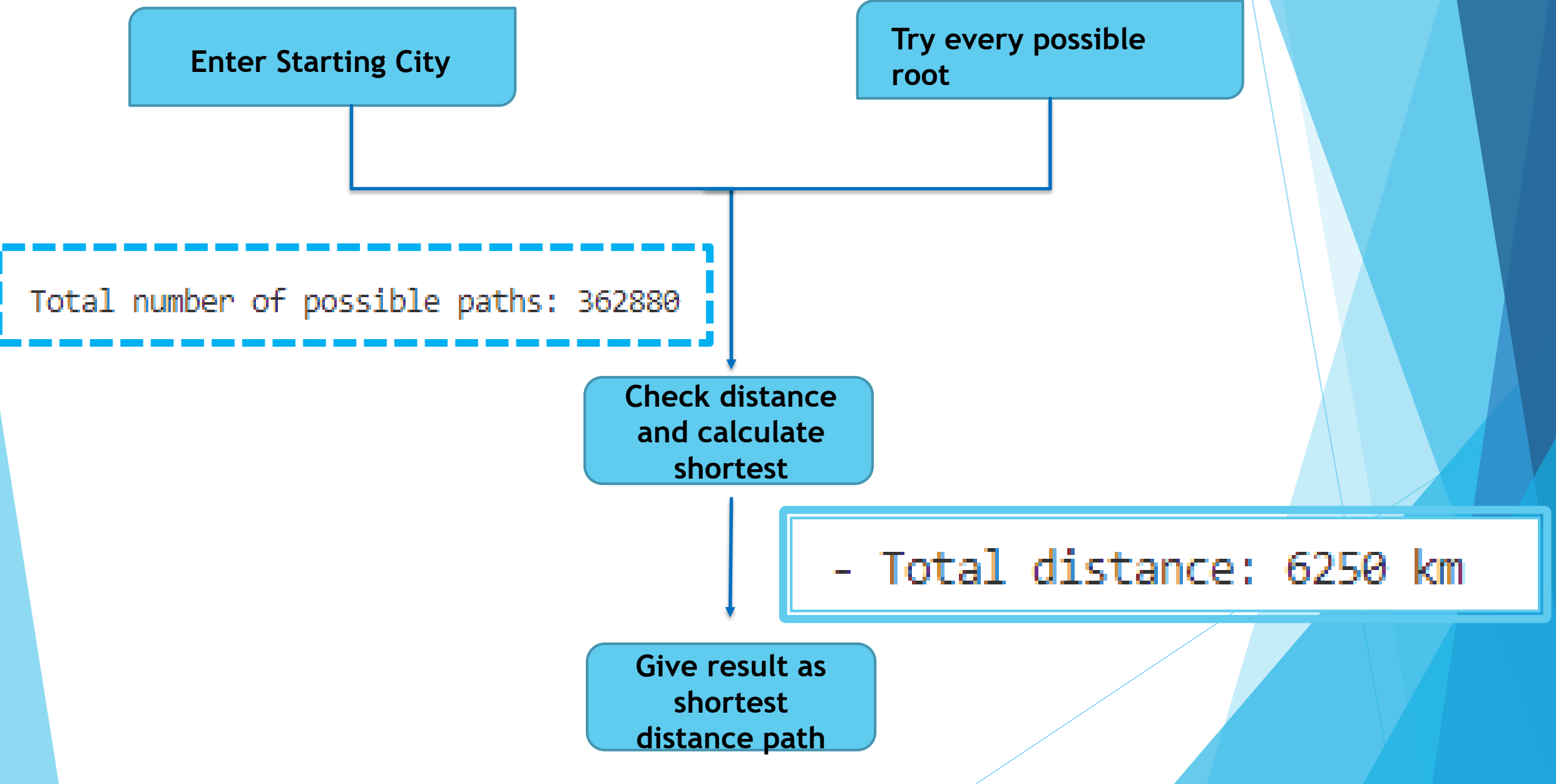
REAL TIME PROBLEM

REAL TIME PROBLEM

In India's major cities like Mumbai, Delhi, and Bangalore, real-time traffic management is crucial to mitigate congestion and enhance urban mobility. Smart traffic signals, data-driven analysis, public transport optimization, emergency response coordination, and citizen engagement apps are key solutions. These initiatives face challenges such as infrastructure costs and technological adaptation but are essential for sustainable urban development, improving commute times, reducing pollution, and making cities more livable for their rapidly growing populations.

WORKFLOW

WORKFLOW


Enter Starting City
Try every possible root
Total number of possible paths: 362880
Check distance and calculate shortest
- Total distance: 6250 km
Give result as shortest distance path

CODE SNIPPETS

CODE SNIPPETS

```cpp
#include <iostream>
#include <vector>
#include <algorithm> // for std::next_permutation
#include <climits> // for INT_MAX
#include <sstream> // for stringstream

using namespace std;

#define N 10 // Number of cities

// Function to calculate the total distance of the tour
int calculateDistance(const int path[], const int dist[N][N]) {
    int totalDistance = 0;
    for (int i = 0; i < N; i++) {
        totalDistance += dist[path[i]][path[(i + 1) % N]];
    }
    return totalDistance;
}

// Utility function to convert path and distances between cities to string
string pathToString(const int path[], const int dist[N][N], const vector<string>& cityNames) {
    stringstream ss;
    for (int i = 0; i < N; i++) {
        int from = path[i], to = path[(i + 1) % N];
        ss << cityNames[from] << " -> " << cityNames[to] << " (" << dist[from][to] << " km), ";
    }
    string pathStr = ss.str();
    pathStr = pathStr.substr(0, pathStr.length() - 2); // Remove the trailing comma and space
    return pathStr;
}

// Factorial function to calculate the total number of paths
int factorial(int n) {
    return (n == 1 || n == 0) ? 1 : factorial(n - 1) * n;
}

int main() {
    // Distance matrix representing the distances between each pair of cities
    int dist[N][N] = {
        {0, 1420, 980, 2050, 1330, 710, 150, 530, 1150, 280},
        {1420, 0, 2150, 1460, 2200, 1580, 1430, 930, 280, 1160},
        {980, 2150, 0, 1860, 350, 570, 840, 1490, 2000, 1260},
        {2050, 1460, 1860, 0, 1660, 1460, 2050, 2060, 1520, 1910},
        {1330, 2200, 350, 1660, 0, 630, 1150, 1820, 2100, 1590},
        {710, 1580, 570, 1460, 630, 0, 560, 1200, 1560, 910},
        {150, 1430, 840, 2050, 1150, 560, 0, 660, 1200, 420},
        {530, 930, 1490, 2060, 1820, 1200, 660, 0, 670, 270},
        {1150, 280, 2000, 1520, 2100, 1560, 1200, 670, 0, 930},
        {280, 1160, 1260, 1910, 1590, 910, 420, 270, 930, 0}
    };

    vector<string> cityNames = {
    "Mumbai",
    "Delhi",
    "Bangalore",
    "Kolkata",
    "Chennai",
    "Hyderabad",
    "Pune",
    "Ahmedabad",
    "Jaipur",
    "Surat"
};

    cout<<"\n\n\t\t\t\t\t\tRoute Finder\n\n";
    cout << "\n\nList of Cities:\n\n\n";
    for (int i = 0; i < N; ++i) cout << i+1 << ": for " << cityNames[i] << "\n";
    cout << "\nEnter your chosen city index: ";

    int path[N], start;
    for (int i = 0; i < N; i++) path[i] = i;
```

CODE SNIPPETS

```
71      for (int i = 0; i < N; i++) path[i] = i;
72      cin >> start;
73
74      swap(path[0], path[start]);
75
76      int minPath[N], minDistance = INT_MAX;
77      vector<pair<string, int>> sampledPaths;
78
79      do {
80          int currentDistance = calculateDistance(path, dist);
81          if (sampledPaths.size() < 10) { // Collect up to 10 sample paths
82              string currentPathStr = pathToString(path, dist, cityNames);
83              sampledPaths.push_back(make_pair(currentPathStr, currentDistance));
84          }
85          if (currentDistance < minDistance) {
86              minDistance = currentDistance;
87              copy(path, path + N, minPath);
88          }
89      } while (next_permutation(path + 1, path + N));
90
91
92      cout << "\nSample paths:\n\n\n";
93      for (const auto& p : sampledPaths) {
94          cout << p.first << "Total distance: " << p.second << " km\n\n\n";
95      }
96
97
98      cout << "\nTotal number of possible paths: " << factorial(N - 1) << endl;
99      cout << "\nBest path:\n\n";
100     cout << pathToString(minPath, dist, cityNames) << "\n\n - Total distance: " << minDistance << " km\n";
101
102     return 0;
103 }
```

MODULES USED

MODULES USED

- Iostream
- Vector
- Limits
- Queue
- String
- Unordered Map
- Cmath
- IOmanip

SCREENSHOTS

HEADING

Route Finder

DISTANCES IN MATRIX FORMAT

```
int dist[N][N] = {
    {0, 1420, 980, 2050, 1330, 710, 150, 530, 1150, 280},
    {1420, 0, 2150, 1460, 2200, 1580, 1430, 930, 280, 1160},
    {980, 2150, 0, 1860, 350, 570, 840, 1490, 2000, 1260},
    {2050, 1460, 1860, 0, 1660, 1460, 2050, 2060, 1520, 1910},
    {1330, 2200, 350, 1660, 0, 630, 1150, 1820, 2100, 1590},
    {710, 1580, 570, 1460, 630, 0, 560, 1200, 1560, 910},
    {150, 1430, 840, 2050, 1150, 560, 0, 660, 1200, 420},
    {530, 930, 1490, 2060, 1820, 1200, 660, 0, 670, 270},
    {1150, 280, 2000, 1520, 2100, 1560, 1200, 670, 0, 930},
    {280, 1160, 1260, 1910, 1590, 910, 420, 270, 930, 0}
};
```

LIST OF CHENNAI STATIONS

TOTAL POSSIBLE PATHS

```
Total number of possible paths: 362880
```

FINAL RESULT

Best path:

Chennai -> Bangalore (350 km), Bangalore -> Hyderabad (570 km), Hyderabad -> Pune (560 km), Pune -> Mumbai (150 km), , Mumbai -> Surat (280 km), Surat -> Ahmedabad (270 km), Ahmedabad -> Jaipur (670 km), Jaipur -> Delhi (280 km), , Delhi -> Kolkata (1460 km), Kolkata -> Chennai (1660 km)

CONCLUSION

This project helps us see how tricky the TSP can be, especially as we add more cities. By solving it, even in a simple way, we learn a lot about planning and making decisions. While our method works well for a small number of cities, it also shows why we need smarter ways to solve the problem when we have lots of cities. It's a great example of how solving puzzles can teach us to think critically and creatively.