LAB ASSIGNMENT DAA

Hrishikesh Prakash
22BRS1017

TSP using DP

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits.h>
using namespace std;

const long long INF = LLONG_MAX;

int tsp(const vector<vector<int>>& dist) {
   int n = dist.size();
   vector<vector<long long>> dp(1 << n, vector<long long>(n, INF));

   for (int i = 1; i < n; i++) {
      dp[1 << i][i] = dist[0][i];
   }

   for (int mask = 0; mask < (1 << n); mask++) {
      for (int i = 0; i < n; i++) {
         if (mask & (1 << i)) {
            for (int j = 0; j < n; j++) {
               if (mask & (1 << j) && i != j) {
                  dp[mask][i] = min(dp[mask][i], dp[mask ^ (1 << i)][j] + dist[j][i]);
               }
```

```
            }
        }
    }
}

    long long result = INF;
    for (int i = 1; i < n; i++) {
        result = min(result, dp[(1 << n) - 1][i] + dist[i][0]);
    }

    return result+1;
}

int main() {
    vector<vector<int>> dist = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    cout << "Minimum tour cost: " << tsp(dist) << endl;
    return 0;
}
```

```
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> g++ .\tsp_dp.cpp
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> .\a.exe
Minimum tour cost: 80
```

TSP using branch and bound

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

#define N 4

int findMinEdgeCost(const vector<vector<int>>& dist, int i) {
    int minCost = INT_MAX;
    for (int j = 0; j < N; j++) {
        if (i != j) {
            minCost = min(minCost, dist[i][j]);
        }
    }
    return minCost;
}

int findSecondMinEdgeCost(const vector<vector<int>>& dist, int i) {
    int firstMin = INT_MAX, secondMin = INT_MAX;
    for (int j = 0; j < N; j++) {
        if (i == j) continue;
        if (dist[i][j] <= firstMin) {
            secondMin = firstMin;
            firstMin = dist[i][j];
        } else if (dist[i][j] <= secondMin) {
```

```
            secondMin = dist[i][j];
        }
    }
    return secondMin;
}

void tspBranchAndBound(const vector<vector<int>>& dist, vector<bool>& visited, int currBound,
                       int currWeight, int level, vector<int>& currPath, vector<int>& finalPath, int& minCost) {

    if (level == N) {
        if (dist[currPath[level - 1]][currPath[0]] != 0) {
            int currCost = currWeight + dist[currPath[level - 1]][currPath[0]];
            if (currCost < minCost) {
                minCost = currCost;
                finalPath = currPath;
                finalPath.push_back(currPath[0]);
            }
        }
        return;
    }

    for (int i = 0; i < N; i++) {
        if (!visited[i] && dist[currPath[level - 1]][i] != 0) {
            int temp = currBound;
            currWeight += dist[currPath[level - 1]][i];

            if (level == 1) {
                currBound -= ((findMinEdgeCost(dist, currPath[level - 1]) + findMinEdgeCost(dist, i)) / 2);
            } else {
```

```
            currBound -= ((findSecondMinEdgeCost(dist, currPath[level - 1]) + findMinEdgeCost(dist, i)) / 2);
        }

        if (currBound + currWeight < minCost) {
            currPath[level] = i;
            visited[i] = true;

            tspBranchAndBound(dist, visited, currBound, currWeight, level + 1, currPath, finalPath,
minCost);
        }

        currWeight -= dist[currPath[level - 1]][i];
        currBound = temp;
        fill(visited.begin(), visited.end(), false);
        for (int j = 0; j <= level - 1; j++) {
            visited[currPath[j]] = true;
        }
      }
   }
}

void tsp(const vector<vector<int>>& dist) {
   vector<int> currPath(N + 1);
   vector<int> finalPath(N + 1);
   vector<bool> visited(N, false);

   int currBound = 0;
   for (int i = 0; i < N; i++) {
      currBound += (findMinEdgeCost(dist, i) + findSecondMinEdgeCost(dist, i));
```

```
    }

    currBound = (currBound & 1) ? (currBound / 2 + 1) : (currBound / 2);

    visited[0] = true;
    currPath[0] = 0;

    int minCost = INT_MAX;
    tspBranchAndBound(dist, visited, currBound, 0, 1, currPath, finalPath, minCost);

    cout << "Minimum tour cost: " << minCost << endl;
    cout << "Path taken: ";
    for (int i = 0; i <= N; i++) {
        cout << finalPath[i] << " ";
    }
    cout << endl;
}

int main() {
    vector<vector<int>> dist = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    tsp(dist);

    return 0;
```

}

```
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> g++ .\tst_bb.cpp
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> .\a.exe
Minimum tour cost: 80
```

ROBIN KARP ALGORITHM

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

#define d 256

void rabinKarp(string pattern, string text, int q) {
    int m = pattern.size();
    int n = text.size();
    int i, j;
    int p = 0;
    int t = 0;
    int h = 1;

    for (i = 0; i < m - 1; i++)
        h = (h * d) % q;

    for (i = 0; i < m; i++) {
        p = (d * p + pattern[i]) % q;
        t = (d * t + text[i]) % q;
    }
```

```
    for (i = 0; i <= n - m; i++) {

        if (p == t) {
            bool match = true;
            for (j = 0; j < m; j++) {
                if (text[i + j] != pattern[j]) {
                    match = false;
                    break;
                }
            }

            if (match)
                cout << "Pattern found at index " << i << endl;
        }


        if (i < n - m) {
            t = (d * (t - text[i] * h) + text[i + m]) % q;
            if (t < 0)
                t = (t + q);
        }
    }
}

int main() {
    string text = "GEEKS FOR GEEKS";
    string pattern = "GEEK";
    int q = 101;
```

```
    rabinKarp(pattern, text, q);

    return 0;
}
```

```
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> g++ .\robin_karp.cpp
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> .\a.exe
Pattern found at index 0
Pattern found at index 10
```

KMP algorithm

```
#include <iostream>
#include <vector>
using namespace std;

void computeLPSArray(string pattern, int m, vector<int>& lps) {
    int len = 0;
    lps[0] = 0;

    int i = 1;
    while (i < m) {
        if (pattern[i] == pattern[len]) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len != 0) {
```

```
            len = lps[len - 1];
        } else {
            lps[i] = 0;
            i++;
        }
    }
  }
}

void KMPSearch(string pattern, string text) {
  int m = pattern.size();
  int n = text.size();

  vector<int> lps(m);
  computeLPSArray(pattern, m, lps);

  int i = 0;
  int j = 0;
  while (i < n) {
    if (pattern[j] == text[i]) {
      i++;
      j++;
    }

    if (j == m) {
      cout << "Pattern found at index " << i - j << endl;
      j = lps[j - 1];
    } else if (i < n && pattern[j] != text[i]) {
      if (j != 0) {
```

```
            j = lps[j - 1];
        } else {
            i++;
        }
    }
  }
}

int main() {
  string text = "ABABDABACDABABCABAB";
  string pattern = "ABABCABAB";

  KMPSearch(pattern, text);

  return 0;
}
```

```
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> g++ .\KMP.cpp
PS C:\Academics\VIT\VIT sem 5\Design and Analysis of Algorithms\Codes> .\a.exe
Pattern found at index 10
```