bellmanford.cpp

```cpp
#include <iostream>
#include <climits>
using namespace std;
struct Edge
{
    int src, dest, weight;
};
void bellmanFord(Edge edges[], int V, int E, int source)
{
    int distance[V];
    for (int i = 0; i < V; i++)
    {
        distance[i] = INT_MAX;
    }
    distance[source] = 0;
    for (int i = 1; i <= V - 1; i++)
    {
        for (int j = 0; j < E; j++)
        {
            int u = edges[j].src;
            int v = edges[j].dest;
            int w = edges[j].weight;
            if (distance[u] != INT_MAX && distance[u] + w < distance[v])
            {
                distance[v] = distance[u] + w;
            }
        }
    }
    for (int i = 0; i < E; i++)
    {
        int u = edges[i].src;
        int v = edges[i].dest;
        int w = edges[i].weight;
        if (distance[u] != INT_MAX && distance[u] + w < distance[v])
        {
            cout << "Graph contains negative-weight cycle!" << endl;
            return;
        }
    }
    cout << "Vertex\tDistance from Source" << endl;
    for (int i = 0; i < V; i++)
    {
        cout << i << "\t" << distance[i] << endl;
    }
}
int main()
{
    int V, E;
    cout<<"Enter Number of Vertices & Edges (separated by space) : "<<endl;
    cin >> V >> E;
    Edge edges[E];
    cout<<"Enter Source, Destination & Weight (separated by space) : "<<endl;
    for (int i = 0; i < E; i++)
    {
        cin >> edges[i].src >> edges[i].dest >> edges[i].weight;
    }
    int source;
```

```
    cout<<"Enter Source : "<<endl;
    cin >> source;
    bellmanFord(edges, V, E, source);
    return 0;
}
```

```
cd "/home/pict/Desktop/33222/" && g++ bellmanford.cpp -o bellmanford && "/home/pict/…

Enter Number of Vertices & Edges (separated by space) :
5 7
Enter Source, Destination & Weight (separated by space) :
0 1 6
0 2 7
1 3 5
1 4 -4
2 1 -2
3 2 -2
3 4 7
Enter Source :
0
Vertex  Distance from Source
0       0
1       5
2       7
3       10
4       1
```

Travelling Salesman Problem

```
#include<bits/stdc++.h>
using namespace std;
#define N 4
#define INF INT_MAX
struct Node
{
   vector<pair<int, int> > path;
   int matrix_reduced[N][N];
   int cost;
        int vertex;
   int level;
};
Node* newNode(int matrix_parent[N][N], vector<pair<int, int> > const &path,
int level, int i, int j)
{
   Node* node = new Node;
   node->path = path;
   if (level != 0)
   {
      node->path.push_back(make_pair(i, j));
   }
   memcpy(node->matrix_reduced, matrix_parent,
      sizeof node->matrix_reduced);
   for (int k = 0; level != 0 && k < N; k++)
   {
      node->matrix_reduced[i][k] = INF;
      node->matrix_reduced[k][j] = INF;
   }

   node->matrix_reduced[j][0] = INF;
   node->level = level;
   node->vertex = j;

   return node;
}
void rowReduction(int matrix_reduced[N][N], int row[N])
{
   fill_n(row, N, INF);
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
      {
```

```
            if (matrix_reduced[i][j] < row[i]) {
                row[i] = matrix_reduced[i][j];
            }
        }
    }
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (matrix_reduced[i][j] != INF && row[i] != INF) {
                matrix_reduced[i][j] -= row[i];
            }
        }
    }

}

void columnReduction(int matrix_reduced[N][N], int col[N])
{
    fill_n(col, N, INF);

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (matrix_reduced[i][j] < col[j]) {
                col[j] = matrix_reduced[i][j];
            }
        }
    }
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (matrix_reduced[i][j] != INF && col[j] != INF) {
                matrix_reduced[i][j] -= col[j];
            }
        }
    }
}
void printMatrix(int matrix_reduced[N][N]){
    for(int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
```

```
            if (matrix_reduced[i][j] == INF){
               cout<<"INF"<<" ";
            }else{
               cout<<matrix_reduced[i][j]<<" ";
            }
         }
         cout<<"\n";
      }

}
int calculateCost(int matrix_reduced[N][N])
{
   int cost = 0;
   int row[N];
   rowReduction(matrix_reduced, row);
   int col[N];
   columnReduction(matrix_reduced, col);
   for (int i = 0; i < N; i++)
   {
      cost += (row[i] != INT_MAX) ? row[i] : 0,
         cost += (col[i] != INT_MAX) ? col[i] : 0;
   }
   cout<<"\nReduced Matrix :: \n\n";
   printMatrix(matrix_reduced);
   return cost;
}
void printPath(vector<pair<int, int> > const &list)
{
   cout << "\n\nPath :: \n\n";
   for (int i = 0; (unsigned) i < list.size(); i++) {
      cout << list[i].first + 1 <<" — "<< list[i].second + 1 << endl;
   }
}
struct comp
{
   bool operator()(const Node* lhs, const Node* rhs) const {
      return lhs->cost > rhs->cost;
   }
};
int solveTSP(int costMatrix[N][N]){
   priority_queue<Node*, vector<Node*>, comp> pq;
   vector<pair<int, int> > v;
   Node* root = newNode(costMatrix, v, 0, -1, 0);
```

```
    root->cost = calculateCost(root->matrix_reduced);
    pq.push(root);
    while (!pq.empty())
    {
        Node* min = pq.top();
        pq.pop();
        int i = min->vertex;
        if (min->level == N - 1) {
            min->path.push_back(make_pair(i, 0));
            printPath(min->path);
            return min->cost;
        }
        for (int j = 0; j < N; j++)
        {
            if (min->matrix_reduced[i][j] != INF)
            {
                Node* child = newNode(min->matrix_reduced, min->path, min-
>level + 1, i, j);
                child->cost = min->cost + min->matrix_reduced[i][j] +
calculateCost(child->matrix_reduced);
                pq.push(child);
            }
        }
        delete min;
    }
    return 0;
}
int main()
{
    int costMatrix[N][N], result;
    cout <<"Enter the cost matrix  :: \n";
    for (int i = 0; i < N; i++){
      for (int j = 0; j < N; j++){
        if ( i == j){
          costMatrix[i][j] = INF;
        }
        else{
          cout << "Enter the cost of edge "<<i+1<<" -> "<<j+1<<" : ";
          cin>>costMatrix[i][j];
        }
      }
    }
    result = solveTSP(costMatrix);
```

```
    cout << "\n\nTotal Cost :: "<< result <<"\n\n";

    return 0;
}
```

OUTPUT: -

```
Enter the cost matrix  ::
Enter the cost of edge 1 -> 2 : 10
Enter the cost of edge 1 -> 3 : 15
Enter the cost of edge 1 -> 4 : 20
Enter the cost of edge 2 -> 1 : 5
Enter the cost of edge 2 -> 3 : 9
Enter the cost of edge 2 -> 4 : 10
Enter the cost of edge 3 -> 1 : 6
Enter the cost of edge 3 -> 2 : 13
Enter the cost of edge 3 -> 4 : 12
Enter the cost of edge 4 -> 1 :
8
Enter the cost of edge 4 -> 2 : 8
Enter the cost of edge 4 -> 3 : 9
Reduced Matrix ::
INF 0 4 5
0 INF 3 0
0 7 INF 1
0 0 0 INF
Reduced Matrix ::
INF INF INF INF
INF INF 3 0
0 INF INF 1
0 INF 0 INF
Reduced Matrix :: 4
INF INF INF INF
0 INF INF 0
INF 6 INF 0
0 0 INF INF
Reduced Matrix ::
INF INF INF INF
0 INF 3 INF
0 7 INF INF
INF 0 0 INF
Reduced Matrix ::
```

INF INF INF INF
INF INF INF INF
INF INF INF 0
0 INF INF INF
Reduced Matrix ::
INF INF INF INF
INF INF INF INF
0 INF INF INF
INF INF 0 INF
Reduced Matrix ::
INF INF INF INF
INF INF INF INF
INF INF INF INF
INF INF INF INF

Path ::
1 — 2
2 — 4
4 — 3
3 — 1

Total Cost :: **35**

```
#include <iostream>
#include <vector>
#include <iomanip>

static int total_nodes;

void printValues(const std::vector<int>& A) {
    for (int value : A) {
        std::cout << std::setw(5) << value;
    }
    std::cout << std::endl;
}

void subset_sum(const std::vector<int>& s, std::vector<int>& t, int s_size, int sum, int ite, int const target_sum) {
    total_nodes++;
    if (target_sum == sum) {
        printValues(t);
        return;
    }

    if (ite == s_size) {
        return;
    }

    t.push_back(s[ite]);
    subset_sum(s, t, s_size, sum + s[ite], ite + 1, target_sum);
    t.pop_back(); // Backtrack
```

```
    subset_sum(s, t, s_size, sum, ite + 1, target_sum);
}

void generateSubsets(const std::vector<int>& s, int target_sum) {
    std::vector<int> tuplet_vector;
    subset_sum(s, tuplet_vector, s.size(), 0, 0, target_sum);
}

int main() {
    std::vector<int> set = {5, 6, 12, 54, 2, 20, 15};
    int target_sum = 25;

    std::cout << "The set is: ";
    printValues(set);
    generateSubsets(set, target_sum);

    std::cout << "Total Nodes generated: " << total_nodes << std::endl;

    return 0;
}

/tmp/ZgpHZn5DPl.o
The set is:     5    6   12   54    2   20   15
    5    6   12    2
    5   20
Total Nodes generated: 247
```