```cpp
/* Author: Keenan A Leman */
/* Held-Karp Algorithm Implementation, Traveling Salesman Problem Dynamic Programming Solution (Top-Down Approach) */
#include <iostream>
#include <bitset>
#include <cmath>
#include <climits>
#include <unordered_map>
#include <sstream>

using namespace std;

const int n = 4; // number of vertex
int start_index;
int G[n][n] = { // adj matrix
    {0,2,9,10},
    {1,0,6,4},
    {15,7,0,8},
    {6,3,12,0}
};

/*int G[n][n] = { // adj matrix
    {0,2,9,10,12},
    {1,0,6,4,12},
    {15,7,0,8,12},
    {6,3,12,0,12},
    {6,3,12,12,0}
};*/

unordered_map<string, int> dp;// O((2 ^ n) * n) space
// use thread safe map or 2D array of atomic integer
// O((2 ^ n) * (n ^ 2)) if memoization data structure has O(1) complexity for insert and find operation

/*
 * cost of path which start at start_index and end at index e, after visiting all vertex in set sS
 * sS is string of 0 and 1 with length n
 */
int cost(int e, string sS){
    stringstream concat_stream;
    concat_stream << e << sS;
    unordered_map<string,int>::iterator it = dp.find(concat_stream.str());
    if(it != dp.end()){
        return it->second;
    }
    bitset<n> S (sS);
    if(S.count() == 0){
        return G[start_index][e];
    }else{
        int mn = INT_MAX;
        for(int i = 0;i < n;i++){
            bitset<n> rS = S;
            if(S.test(i) && i != e){
                rS.reset(i);
                mn = min(cost(i, rS.to_string()) + G[i][e],mn);
            }
        }
        concat_stream.str("");
        concat_stream << e << sS;
        dp[concat_stream.str()] = mn;
        return mn;
    }
}

int main(){
    bitset<n> S;
    S.set();
```

```
    // start and end at 0
    start_index = 0;
    S.reset(start_index);
    cout << cost(start_index,S.to_string()) << endl;
    return 0;
}
```