Project Report

Algorithms - CSE246

Problem ID: 02

Project: Hamiltonian Cycle

Submitted by:

Md. Sanim Hossain
ID: 2020-1-60-267
Md. Suman Akanda
ID: 2020-1-60-266
Shams E Siam
ID: 2020-1-60-265

Submitted to:

Redwan Ahmed Rizvee
Lecturer
Department of Computer Science and Engineering

Date of Submission: 02-09-2022

Problem Statement:

The given problem is “Hamiltonian Cycle”. A Hamiltonian cycle is a closed loop on a graph where every node (vertex) is visited exactly once. A loop is just an edge that joins a node to itself; so, a Hamiltonian cycle is a path traveling from a point back to itself, visiting every node and route. The purpose of this problem is to calculate if the graph is Hamiltonian cycle or not.

Pseudocode:

```
const int MAX = 1e5;

int N, E;
vector<int> G[MAX];
int path[MAX];

bool check(int v, int pos) {
   initial  bool as  false;
   int u = previous node
   for, i is 0 to G[u].size()
      if (G[u][i] == v)
         This node is connected with the previous node

     if (mk == false)
         This node is not connected with the previous node

   For, i is 0 to pos
      if (path[i] == v)
         Here, v is taken multiple times
         So return false;

   return true;
}

bool hamiltonianCycle(int pos) {
   if (pos == N) {
      initial  bool as  false
      int u = last node
       for, i is 0 to G[u].size()
         if (G[u][i] == 0)
```

```
            then 0 is connected with last node
        Return returns if the last node is connected with 0
    }

     For, i is 0 to N

        if (check(v, pos)){
            path[pos] = v;
            check (hamiltonianCycle(pos + 1))
                return true;
            path[pos] = -1;

    return false;
```

Implementation :

```
bool check(int v, int pos) {
    bool mk = false;
    int u = path[pos - 1];
    for (int i = 0; i < G[u].size(); i++) {
        if (G[u][i] == v)
            mk = true;     }
    if (mk == false)
        return false;

    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return false;

    return true;
}

bool hamiltonianCycle(int pos) {
    if (pos == N) {
        bool mk = false;
        int u = path[pos - 1];
        for (int i = 0; i < G[u].size(); i++) {
            if (G[u][i] == 0)
```

```
            mk = true;
        }
        return mk; /// returns if last node is connected with 0
    }

    for (int v = 0; v < N; v++) {
        if (check(v, pos)) {
            path[pos] = v;
            if (hamiltonianCycle(pos + 1))
                return true;
            path[pos] = -1;
        }
    }
    return false;
}
```

Time complexity:

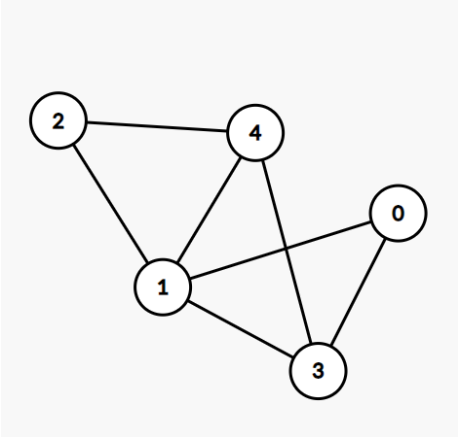
The most straightforward approach to solve the given problem is to of N vertices. For each permutation, check if it is a valid Hamiltonian path by checking if there is an edge between adjacent vertices or not. So, the time complexity of the above algorithm is O(*2^n.n^2*). DFS and backtracking can also help check whether a Hamiltonian path exists in a graph.

Time Complexity: O(2^n.n^2)

Application:

If by Hamiltonian you mean it is a graph that has at least one Hamiltonian cycle, then yes. In fact, there are (n−1)! /2 distinct Hamiltonian cycles in a complete graph. It is pretty simple to find one. Just compute a path where you begin by identifying a starting vertex in the traversal, visit all the vertices by following edges of unmarked/unvisited vertices, then include the edge that's back to the starting vertex; this final edge is guaranteed to exist because the graph is complete. There are many examples of the Hamiltonian graphs. A school bus can be a good example. School bus use Hamiltonians to plan the best route to pick up students from across the area. Here students may be considered nodes, the paths between them edges, and the bus wishes to travel a route that will pass each student's house exactly once. The same strategy can be applied to a newspaper hawker, milkman, home delivery services, and even for circular bus services like hatirjheel.

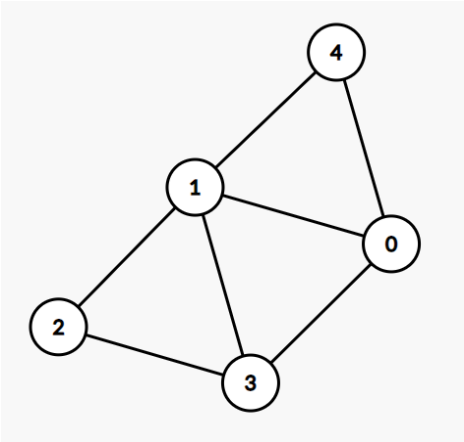
Test Cases: 1




```
5 7
0 1
0 3
1 2
1 3
1 4
2 4
3 4
YES
0 1 2 4 3 0

Process returned 0 (0x0)   execution time : 24.977 s
Press any key to continue.
```

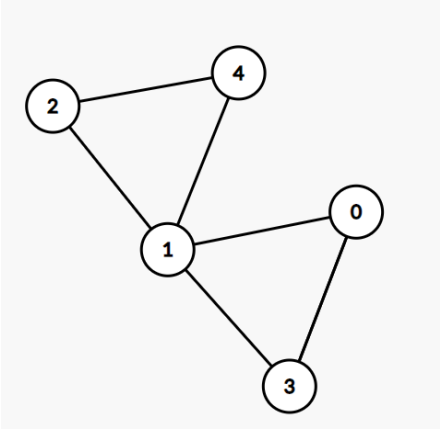
Test cases: 2




```
5 7
0 1
0 3
1 2
1 3
1 4
0 4
2 3
YES
0 3 2 1 4 0

Process returned 0 (0x0)   execution time : 16.126 s
Press any key to continue.
```

Test cases : 3




```
5 7
0 1
0 3
1 2
1 3
1 4
2 4
3 0
NO

Process returned 0 (0x0)   execution time : 4.680 s
Press any key to continue.
```