Коледни базари

locked

Problem | Submissions | Leaderboard | Discussions

Тази година сте решили, че искате да обикаляте коледните базари по света. Разполагате със самолетни билети за градове с коледни базари, като формата на всеки билет е CITY1 CITY2. Въпреки че Коледния дух ви е обзел напълно, вие все пак сте здраво стъпили на земята и знаете, че е хубаво после да можете да се върнете от мястото, от което сте започнали. Ако знаете, че започвате пътешествието си от града X, покажете примерен план за пътуване, който да ви върне обратно в стартовия град.

Input Format

На първия ред на стандартния вход ще получите 2 числа: N - броя различни градове, до или от които имате билети и M, броя на билетите, които имате.

На следващите M реда ще получите по 2 низа - името на градовете от и до които е полета.

На последия ред на стандартния вход ще получите низ X - стартовия град, за който трябва да отговорите дали можете да се върнете.

Constraints

1 <= N <= 10^5

1 <= M <= 5 * 10^5

Името на всеки град е дадено с низ с дължина <= 4 символа.

Output Format

На единствения ред на стандартния изход изведете пътя, който трябва да следвате, за да се върнете в началния град. Ако има повече от един такъв път, изведете някой от тях. Ако няма такъв път, изведете -1.

Sample Input 0

```
7 10
SOF LON
SOF PAR
PAR LSA
LON NYC
NYC SYD
SYD TOK
TOK PAR
PAR TOK
NYC SOF
SOF TOK
LON
```

Sample Output 0

```
LON NYC SOF LON
```

Submissions: 104
Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <string>
#include<list>
#include <unordered_map>
#include <stack>
bool b=false;
using namespace std;
struct Node{
  list<string> neighbours;
  bool hasNeighbour(string s){
  for(auto i=neighbours.begin();i!=neighbours.end();i++){
    if(*i==s){
      return true;
      }
    }
    return false;
  }
  void addNeighbour(string s){
    neighbours.push_back(s);
  }
};
struct Graph{
  unordered_map<string,Node> nodes;
  Graph(int nodeCount=0){
    nodes.reserve(nodeCount);
  }
  void connect(string from,string to){
    if(!nodes[from].hasNeighbour(to)){
      nodes[from].addNeighbour(to);
    }
  }
  void isCyclic(string s,unordered_map<string,bool> &visited,string start,list<string> &path){
        visited[s] = true;
        list<string>::iterator i;
   path.push_back(s);
        for(i = nodes[s].neighbours.begin(); i != nodes[s].neighbours.end(); ++i) {
          if(*i==start){
            b=true;
            for(auto it=path.begin();it!=path.end();it++){
    cout<<*it<<" ";

  }
          }
            if ( !visited[*i] ) {
                isCyclic(*i, visited,start,path);
              path.pop_back();
             }
        }

  }
};

int main() {
    int citiesNumber;
  int ticketsNumber;
  cin>>citiesNumber;
  cin>>ticketsNumber;
  Graph g(citiesNumber);
  string from;
  string to;
  unordered_map<string,bool> visited;
  visited.reserve(citiesNumber);
  for(int i=0;i<ticketsNumber;i++){
  cin>>from;
  cin>>to;
    g.connect(from,to);
    visited[from]=false;
    visited[to]=false;
  }
  string firstCity;
  cin>>firstCity;


  list<string> path;

  g.isCyclic(firstCity,visited,firstCity,path);
  if(b){
  cout<<firstCity;
  }
  else{
    cout<<"-1";
  }
  return 0;
}

```

Line: 1 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code