CSE 401: Artificial Intelligence
Lab 6: Open Ended Lab Questions

Utkarsh Gupta
A2305217557
7CSE 8Y

September 05th, 2020

1 Question 1: Water-Jug Problem

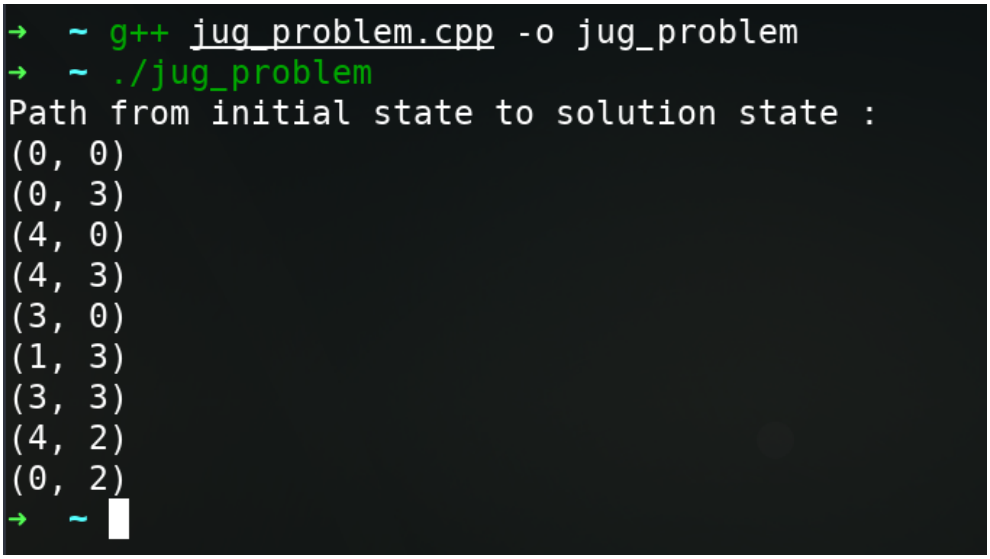
You have 3 jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon. Given the initial state, goal test, successor function and cost function. Implement the problem in any programming language.

Code:

[1]:
```
#include <bits/stdc++.h>
#define pii pair<int, int>
#define mp make_pair

using namespace std;
void BFS(int a, int b, int target)
{
    map<pii, int> m;
    bool isSolvable = false;
    vector<pii> path;
    queue<pii> q; // queue to maintain states
    q.push({ 0, 0 }); // Initialing with initial state
    while (!q.empty()) {
        pii u = q.front();
        q.pop();
        if (m[{ u.first, u.second }] == 1)
            continue;
        if ((u.first > a || u.second > b ||
            u.first < 0 || u.second < 0))
            continue;
        path.push_back({ u.first, u.second });
        m[{ u.first, u.second }] = 1;
        if (u.first == target || u.second == target) {
            isSolvable = true;
            if (u.first == target) {
```

```
                if (u.second != 0)
                    path.push_back({ u.first, 0 });
            }
            else {
                if (u.first != 0)
                    // fill final state
                    path.push_back({ 0, u.second });
            }
            int sz = path.size();
            for (int i = 0; i < sz; i++)
                cout << "(" << path[i].first
                    << ", " << path[i].second << ")\n";
            break;
        }
        q.push({ u.first, b }); // fill Jug2
        q.push({ a, u.second }); // fill Jug1

        for (int ap = 0; ap <= max(a, b); ap++) {
            int c = u.first + ap;
            int d = u.second - ap;
            if (c == a || (d == 0 && d >= 0))
                q.push({ c, d });
            c = u.first - ap;
            d = u.second + ap;
            if ((c == 0 && c >= 0) || d == b)
                q.push({ c, d });
        }
        q.push({ a, 0 });
        q.push({ 0, b });
    }
    if (!isSolvable)
        cout << "No solution";
}

int main()
{
    int Jug1 = 4, Jug2 = 3, target = 2;
    cout << "Path from initial state "
            "to solution state :\n";
    BFS(Jug1, Jug2, target);
    return 0;
}
```

Output:



---

2 Question 2: Friends Problem

The group of people consists of N members. Every member has one or more friends in the group. You have to write a program that divides this group into two teams. Every member of each team must have friends in another team. The first line of input contains the only number N (N 100). Members are numbered from 1 to N. The second, the third, and the (N+1) th line contain list of friends of the first, the second and the Nth member respectively. This list is finished by zero. Remember that friendship is always mutual in this group. The first line of output should contain the number of people in the first team or zero if it is impossible to divide people into two teams. If the solution exists you should write the list of the first group into the second line of output. Numbers should be divided by single space. If there are more than one solution you may find any of them. Also, find the number of people in the second team with its team members. Implement the problem in any programming language.

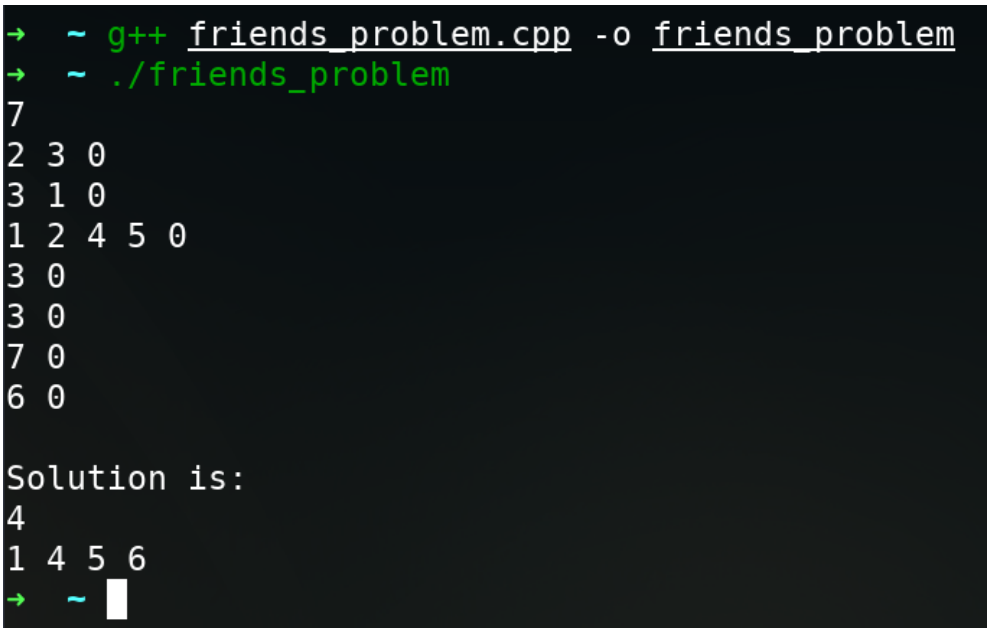
Code:

[2]:
```cpp
#include <iostream>
#include <vector>
#include <queue>

using namespace std;
int main() {
    int N;
    cin >> N;
    vector <vector <int>> g(N);
```

```
    int h;
    for (int i = 0; i < N; i++) {
        for (;;) {
            cin >> h;
            if (!h)
                break;
            g[i].push_back(h - 1);
        }
    }
    int n = g.size();
    queue<int> q;
    vector<bool> used(n);
    vector<int> d(n);
    int s;
    auto bfs = [&](int start) {
        s = start;
        q.push(s);
        used[s] = true;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (size_t i = 0; i<g[v].size(); ++i) {
                int to = g[v][i];
                if (!used[to]) {
                    used[to] = true;
                    q.push(to);
                    d[to] = d[v] + 1;
                }
            }
        }
    };
    for (int start = 0; start < n; ++start)
        if (!used[start]) {
            bfs(start);
        }
    vector <int> ans;
    for (int i = 0; i < n; i++) {
        if (d[i] % 2 == 0)
            ans.push_back(i + 1);
    }
    int size = ans.size();
    cout << "\n";
    cout << "Solution is: \n";
    cout << size << "\n";
    for (int i = 0; i < size; i++)
        cout << ans[i] << " ";
    cout << endl;
```

```
}
```

Output:



---

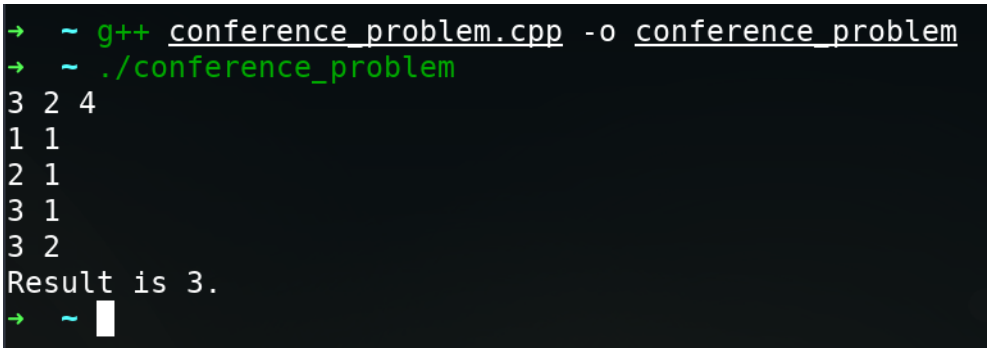
3 Question 3: Conference Problem

On the upcoming conference were sent M representatives of country A and N representatives of country B (M and N 1000). The representatives were identified with 1, 2... M for country A and 1, 2... N for country B. Before the conference K pairs of representatives were chosen. Every such pair consists of one member of delegation A and one of delegation B. If there exists a pair in which both member #i of A and member #j of B are included then #i and #j can negotiate. Everyone attending the conference was included in at least one pair. The CEO of the congress center wants to build direct telephone connections between the rooms of the delegates, so that everyone is connected with at least one representative of the other side, and every connection is made between people that can negotiate. The CEO also wants to minimize the amount of telephone connections. Write a program which given M, N, K and K pairs of representatives, finds the minimum number of needed connections. The first line of the input contains M, N and K. The following K lines contain the chosen pairs in the form of two integers p1 and p2, p1 is member of A and p2 is member of B. The output should contain the minimum number of needed telephone connections.

Code:

[3]:
```cpp
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
const int V=1100;
int n,m,k,x,y,pre[V];
bool v[V],a[V][V];

bool dfs(int i)
{
    for(int j=1;j<=m;j++)
    if((!v[j])&&(a[i][j])){
        v[j]=1;
        if(pre[j]==0 || dfs(pre[j])){
            pre[j]=i;
            return 1;
        }
    }
    return 0;
}

int main()
{
    cin>>n>>m>>k;
    memset(a,0,sizeof(a));
    memset(pre,0,sizeof(pre));
    for(int i=1;i<=k;i++){
        cin>>x>>y;
        a[x][y]=1;
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        memset(v,0,sizeof(v));
        if(dfs(i))
        ans++;
    }
    cout<<"result: "<<n+m-ans<<endl;
    return 0;
}
```

Output:


➜ ~ g++ conference_problem.cpp -o conference_problem
➜ ~ ./conference_problem
3 2 4
1 1
2 1
3 1
3 2
Result is 3.
➜ ~
