Safe Fruit

Author Names
(temporary vacancy during anonymous peer review)

Date: 2020-04-01

Chapter 1: Introduction

There are a lot of tips telling us that some fruits must not be eaten with some other fruits, or we might get ourselves in serious trouble. For example, bananas can not be eaten with cantaloupe, otherwise it will lead to kidney deficiency.

Now you are given a long list of such tips, and a big basket of fruits. You are supposed to pick up those fruits so that it is safe to eat any of them.

Chapter 2: Data Structure / Algorithm Specification

In the chapter, we will briefly discuss the algorithm. We use DFS and pruning method to implement the algorithm.

2.1 DFS

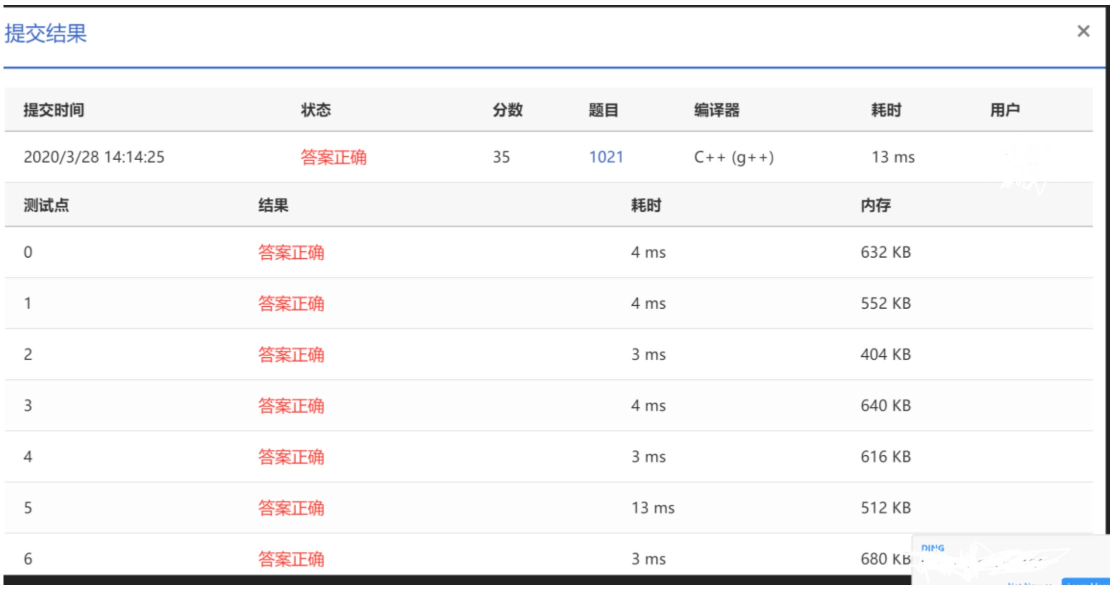
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

The time and space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time {\displaystyle O(|V|+|E|)}O(|V| + |E|),[4] linear in the size of the graph. In these applications it also uses space {\displaystyle O(|V|)}O(|V|) in the worst case to store the stack of vertices on the current search path as well as the set of already-visited vertices. Thus, in this setting, the time and space bounds are the same as for breadth-first search and the choice of which of these two algorithms to use depends less on their complexity and more on the different properties of the vertex orderings the two algorithms produce.

For applications of DFS in relation to specific domains, such as searching for solutions in artificial intelligence or web-crawling, the graph to be traversed is often either too large to visit in its entirety or infinite (DFS may suffer from non-termination). In such cases, search is only performed to a limited depth; due to limited resources, such as memory or disk space, one typically does not use data structures to keep track of the set of all previously visited vertices. When search is performed to a limited depth, the time is still linear in terms of the number of expanded vertices and edges (although this number is not the same as the size of the entire graph because some vertices may be searched more than once and others not at all) but the space complexity of this variant of DFS is only proportional to the depth limit, and as a result, is much smaller than the space needed for searching to the same depth using breadth-first search. For such applications, DFS also lends itself much better to heuristic methods for choosing a likely-looking branch. When an appropriate depth limit is not known a priori, iterative deepening depth-first search applies DFS repeatedly with a sequence of increasing limits. In the artificial intelligence mode of analysis, with a branching factor greater than one, iterative deepening increases the running time by only a constant factor over the case in which the correct depth limit is known due to the geometric growth of the number of nodes per level.

DFS may also be used to collect a sample of graph nodes. However, incomplete DFS, similarly to incomplete BFS, is biased towards nodes of high degree.

Chapter 3: Testing Results



提交时间	状态	分数	题目	编译器	耗时	用户
2020/3/28 14:14:25	答案正确	35	1021	C++ (g++)	13 ms	[redacted]

测试点	结果	耗时	内存
0	答案正确	4 ms	632 KB
1	答案正确	4 ms	552 KB
2	答案正确	3 ms	404 KB
3	答案正确	4 ms	640 KB
4	答案正确	3 ms	616 KB
5	答案正确	13 ms	512 KB
6	答案正确	3 ms	680 KB

3.1 Test Environment

Our test environment is just as following:

OS: Windows 10 Pro for Workstations, Build 18363.720

Compiler: gcc 8.1.0, x86_64-posix-seh-rev0

Compile command: g++ -Wall -std=c++14 source.cpp -o source -O2

RAM: 64GiB

3.2 Test Results and Analysis

We use one input to test the correctness of the program. It lies in the /data/test1/data.in folder. The result agrees with our expectations. Hence the algorithm is correctly implemented.

Output:

```
12

002 004 006 008 009 014 015 016 017 018 019 020

239
```

Chapter 4: Analysis and Comments

4.1 Time Complexity Analysis

We use O(V+E) time to brutely traverse the data structure.

4.2 Space Complexity Analysis

O(V).

Appendix: Source Code

The source code folder consists of one file: source.cpp.

The source.cpp:

```cpp
#include <bits/stdc++.h>
#define N 1005
#define M 10050
using namespace std;

int E, V, mx; //E is the number of edges, V is the number of vertices

int tipa[M], tipb[M], mp[M], invmp[M], val[N], tot; //Variable used for
relabeling.

int p[N], mark[N], ans1; //Variables used to calculate the answer
int sum, cv, cd, ins[N], vis[N], F[105][105], o[N], tp, ans2;
bitset<105> b[105][105], curb, msk, hm;

/*
Key for vertex sorting.
*/
bool cmp(int p1, int p2) {
    return val[p1] < val[p2];
}

/*
The structure Edge is used to store edges, where b represents
adjacent vertices and n represents the next edge. h[] is the label
that stores the first edge of each vertex.
*/
int h[N], cnt;
struct Edge{int b,n;}e[M];
inline void link(int a, int b) {
    e[++cnt] = (Edge){b,h[a]}, h[a] = cnt;
}

/*
The functions enc() and dec() are an encoder and a decoder to
relabel vertices.
*/
inline int enc(int t) {
    !mp[t] ? mp[t] = ++tot : 0;
    invmp[ mp[t] ] = t;
    return mp[t];
}

inline int dec(int t) {
    return invmp[t];
}

/*
The main process of this program is DFS (), which is a depth
first search process.
*/

void dfs(int dep, int cx) {

    //If the recursion reaches the boundary, that is, all the
    //points have been taken, the answer is updated and returned.
    if (dep == 0) {
        F[cv][cd] = sum;
        b[cv][cd] = curb;
        if (cd > ans1 || (cd == ans1 && sum < ans2)) {
            ans1 = cd;
            ans2 = sum;
            hm = curb;
        }
        return ;
    }

    //There's nothing to expand
    if (cx == 0) return ;

    //Enumerate a vertex and consider adding it to the answer.
    for (int i=cx;i>=(dep==cd ? cx : 1);i--) {
        int u = p[i];

        //If this point can be added to the answer, consider adding
        //it to the answer. Mark the point next to it and mark it in
        //BitSet.
        if (!vis[u]) {
            vis[u] = 1;
            ins[u] = 1;
            for (int j=h[u];j;j=e[j].n) {
                vis[ e[j].b ]++;
                if (vis[ e[j].b ]) msk[ e[j].b ] = 1;
            }
            sum += val[u]; //Weight h of the currently selected independent
set
            curb[u] = 1; //A BitSet is used to mark which points are in the
answer.
            msk[u] = 1; //A BitSet is used to mark which points cannot be added
to the independent set.

            //Using inequality and evaluation function to judge whether the
answer may be better.
            //Keep going if possible, otherwise stop the search.
            if (sum + F[i-1][dep-1] < F[cv][cd]) {
                if ((msk & b[i-1][dep-1]).count() == 0) {
                    F[cv][cd] = sum + F[i-1][dep-1];
                    b[cv][cd] = curb | b[i-1][dep-1];

                    //Update the answer if there is a legal situation.
                    if (cd > ans1 || (cd == ans1 && F[cv][cd] < ans2)) {
                        ans1 = cd;
                        ans2 = F[cv][cd];
                        hm = b[cv][cd];
                    }
                } else{
                    dfs(dep-1, i-1);
                }

            }

            //Backtracking in depth first search. Remove all previous marks.
            sum -= val[u]; //Weight h of the currently selected independent
set
            for (int j=h[u];j;j=e[j].n) {
                vis[ e[j].b ]--;
                if (!vis[ e[j].b ]) msk[ e[j].b ] = 0;
            }
            vis[u] = 0;
            msk[u] = 0;//A BitSet is used to mark which points cannot be added
to the independent set.
            ins[u] = 0;
            curb[u] = 0;//A BitSet is used to mark which points are in the
answer.
        }
    }
    return ;
}

int main() {
/*
The following part of the code is used to read in all the input data,
relabel the input data and complete various initialization operations.
*/
    scanf("%d%d",&E,&V);
    for (int i=1;i<=E;i++)
        scanf("%d%d",&tipa[i],&tipb[i]);

    for (int i=1;i<=V;i++) {
        int x, y;
        scanf("%d%d",&x,&y);
        val[ enc(x) ] = y;
        mx += y;
    }

    srand(mx);

    for (int i=1;i<=E;i++)
        if (mp[ tipa[i] ] && mp[ tipb[i] ]) {
            int u = enc(tipa[i]), v = enc(tipb[i]);
            link(u, v);
            link(v, u);
        }

//--------------------------------------------------------------
/*
This part of the code is used to calculate the answer
*/
    for (int i=1;i<=V;i++) p[i] = i;
    if (rand()&1) sort(p+1, p+V+1, cmp);

    memset(F, 127/3, sizeof(F));
    for (int i=0;i<=V;i++) F[i][0] = 0;
    ans2 = 1<<29;
    for (int i=1;i<=V;i++)
        for (int j=ans1+1;j>=1;j--) {
            cv = i;
            cd = j;
            //It will not cost much to get the same independent set from fewer
points.
```

```
            //Use this to update the initial value.
            if (F[cv-1][cd] < F[cv][cd]) {
                F[cv][cd] = F[cv-1][cd];
                b[cv][cd] = b[cv-1][cd];
            }

            //Taking a smaller independent set from the same point set will
not cost more.
            //Use this to update the initial value.
            if (F[cv][cd+1] < F[cv][cd]) {
                F[cv][cd] = F[cv][cd+1];
                b[cv][cd] = b[cv][cd+1];
            }

            dfs(cd, cv);
        }

//-----------------------------------------------------------
/*
This code is used to sort, decode and output the answers.
*/
    cout << ans1 << endl;

    for (int i=1;i<=V;i++) if (hm[i]) o[++tp] = dec(i);
    sort(o+1, o+ans1+1);
    for (int i=1;i<=ans1;i++) printf("%03d%c", o[i], i==ans1?'\n':' ');
    cout << ans2 << endl;

    return 0;
}
```

References

Author List

Temporary vacancy during anonymous peer review.

Declaration

We hereby declare that all the work done in this project titled " Safe Fruit" is of our independent effort as a group.

Signatures

Temporary vacancy during anonymous peer review.