Project 3

Safe Fruit

Group 17

Date:2022-04-16

Chapter 1: Introduction

```
In our daily life, there are plenty of tips that tell us that some
fruits should not be eaten with others or we could be in serious
trouble. For example, bananas can not be eaten with cantaloupe,
otherwise it will lead to kidney deficiency.

In this project, we are to find and pick up the safe fruits. For a
file including N pairs of fruits which must not be eaten together
and M fruits together with their prices, we program to find the
maximum number of safe fruits and list them out. If more than 1
solutions, we shall output the one with minimum cost.
```

Chapter 2: Algorithm Specification

We mainly use Arrays to store the data. The edges are stored in a Linked List.

Main Program

```
signed main(signed, char**, char**) {
    input();
    dfs(1);
    output();
}
```

Input

This part is to get the content of the test case, store the tips and give each fruit in basket an id.

```
Procedure input()
    Input n,m;

    For i = 0 To n-1 Step 1
    Input lx[i][0],lx[i][1];

    For i = 0 To m Step 1
        Input  a,value[i];
```

```
        ori[i] := a, getid[a] := i;

    For i = 0 To n-1 Step 1 if getid[lx[i][0]] && getid[lx[i][1]]1=0 then
        minx := min(getid[lx[i][0]], getid[lx[i][1]]);
        maxx := max(getid[lx[i][0]], getid[lx[i][1]]);
        addedge(minx, maxx);

```

dfs

This part is to search the list and then memorize the most fruits that can be selected in $[now, m]$ and return the most fruits in $[now, m]$ currently.

You can see the prunning strategy in the Time Complexity Analysis part.

```
Procedure dfs(now:integer)
    if now = m + 1 then
  {
        check();
        return 0;
    }
    if siz = 0 then
  {
        if calc[now]!=0  them
      return remi[now];
        else calc[now] := 1;
    }

    res1 := -inf, res2 := -inf;
    if siz + remi[now + 1] >= maxf then
  {
        res1 := dfs(now + 1);
    }
    if bansel[now] = 0 && siz + 1 + remi[now + 1] >= maxf then
  {
        sol[siz++] := now, cost := cost+value[now];
        for edge* ptr = head[now];ptr;ptr = ptr->next
    {
            ++bansel[ptr->t];
        }
        res2 := dfs(now + 1);
        --siz, cost -= value[now];
```

```
        for (edge* ptr = head[now];ptr;ptr = ptr->next)
    {
            --bansel[ptr->t];
        }
    }
    if siz = 0 then
    remi[now] := max(res1, res2 + 1);
    return max(res1, res2 + 1);

```

Output

```
Procedure output()
    Output maxf;
    For i = 0 To maxf-1 Step 1
        fin[i] := ori[fin[i]];
    sort(fin, fin + maxf);
    For i = 0 To maxf-1 Step 1
  Output fin[i];
    Output minc;

```

Chapter 3: Testing Results

We have given some input cases in data folder, and we use these data to analyse. The result table is as follows.

case	input	note	output	time
1	in1.txt	Sample input	out1.txt	0.000s
2	in2.txt	A complete graph	out2.txt	0.001s
3	in3.txt	A complete graph except one edge	out3.txt	0.000s
4	in4.txt	Normal data with 100 tips and 20 fruits	out4.txt	0.001s
5	in5.txt	Normal data with 100 tips and 50 fruits	ou5.txt	0.091s
6	in6.txt	All fruits independent	out6.txt	0.000s
7	in7.txt	Random data generated by program	out7.txt	0.599s

Their purposes:

2&6: Test the cost function.

3: Test the maximum number selection.

4&5: Test efficiency.

7: Random condition.

Here we have listed several test results with either specified or random input to show the correctness of the program. You can also test the program in PAT-Top 1021.

Space Complexity

- The program use $O(N+M)$ space to store the graphs and solutions.

Time Complexity

- input : 3 separate loops, so the time complexity is $O(N+M)$.
- dfs : dfs M times, each time we seperate into 2 cases, apparently the time complexity is $O(2^M)$, though we have cut some search routes.

The prunning strategy

We use an array $remi$ to memorize the maximum number of fruits can be selected in $[i, M]$, and when considering $i-1$ fruit we can use $remi[i]$ to cut the search routes which cannot earn more fruits than the current solution.

We consider maintaining a set of unsafe fruits, and when searching fruits 1~i we can get the set about $i+1$~m. For there unsafe fruits, we just omit them.

We shall also consider that, that graph may be divided into several strongly connected components, and we should deal with these SCCs seperately to use less time. However, this strategy is not realized in our program.

Appendix(readme.txt)

```
We suppose read the source files to get more comments. The files:
datagen.cpp is used to generate some random data. This program is by
Pan.
main.cpp is used to solve the problem. This program is by Zuo.

The generated data is put in data/input.txt,
and we have prepared some cases in data/in1~7.txt as the above table
shows.

The report is by Han and Shi.

We recommend using DEV-C++ v5.15 to open the project, and TDM-
GCC9.2.0 to compile.
The C++ standard shall be C++11 or above.
```