Project 2
Safe Fruit

Author Names:
omitted

Date: 2020-4-5

Chapter 1：Introduction

The title gives m kinds of fruits. The price of each kind of fruit is known and there are n pairs of fruits that cannot be eaten at the same time. Now our task is to find the largest set of fruits that can be eaten at the same time. If there are multiple schemes, we choose the lowest total price.

We use graph to solve the problem. Consider each fruit as a point in graph G and connect fruits which cannot be eaten together with an undirected edge. Then our task is to find the largest independent point set of this graph.

We know that the maximum independent set problem of graphs is equivalent to the maximum clique problem. Both are proved to be *NPC* problems so that there is no polynomial complexity algorithm. It can only be solved by searching.

Chapter 2：Algorithm Specification

🔷 Data structure: Graph

We use the adjacency matrix to represent the graph $G = (E, V)$, where the vertex V represents fruit, and the edge E connects fruits that can be eaten at the same time. Now our task is to find the largest group of G.

```
typedef struct
    fgraph{ int v;
    bool graph[1000][1000];
}FG;
```

Initially, the adjacency matrix is

$$\begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 1 & \cdots & 1 \\ 1 & 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 1 & 1 & \cdots & 0 \end{bmatrix}$$

where 0 indicates that there is no edge connection, and 1 indicates that there is an edge connection. The following pseudo code illustrates the process of initializing the graph.

```
function graphinitial(m)
    G.v ← m;
    for i ← 0 to 999
        for j ← 0 to 999
            if i!=j then
                G.graph[i][j] ← 1
            else
                G.graph[i][j] ← 0
            end if
        end for
    end for
end function
```

🔷 Algorithm: DFS + Backtracing

We use depth-first-search(dfs) to find the largest group. We define an array leftf to indicate the points that can join the current group. That is to say, when a point in the array leftf joins the current group, it is still a group. In each step we search, we select a point from leftf to join the current group, while maintaining leftf, and then perform the next search. Finally, restore the state and trace back after searching.

```
function dfs(leftf[], k, curn)
    for i ← 0 to k-1
        x ← 0
        for j ← i+1 to k-1
            if G.graph[leftf[i]][leftf[j]] == true then
                newleftf[x++] ← leftf[j]
            end if
        end for
        currentf[top++] ← leftf[i]
        curprice ← curprice+price[leftf[i]]
        dfs(newleftf,x,curn+1)
        curprice ← curprice-price[leftf[i]]
        top ← top-1
    end for
end function
```

🔷 Algorithm: Pruning + DP

Simply using dfs cannot pass the test. We need to prun the search tree to reduce the running time.

The method of pruning is dynamic programming. We define an array $dp[i]$ to represent the size of the largest group containing node i. Assume node j will join in the next search. If the size of current group plus $dp[j]$ is still less than the optimal solution that has been searched, there is

no need to search and you can prun. The maintenance of dp array is achieved through memory search.

```
function FindMaxFruit()
    for i ← G.v to 1
        currentf[top++] ← fruits[i]
        curprice ← curprice+price[fruits[i]]
        for j ← i+1 to G.v
            if G.graph[fruits[i]][fruits[j]] == true then
                leftf[k++] ← fruits[j]
            end  if
        end for
        dfs(leftf,k,1)
        curprice ← curprice-price[fruits[i]]
        top ← top-1
        dp[fruits[i]] ← maxf
    end for
end function
```

Another idea of pruning is that if the size of the currently searched group after adding all the points that can be added is still smaller than the optimal solution that has been found, then there is no inevitable search and pruning can be done.

The pseudo code of dfs after adding pruning is as follows:

```
function dfs(leftf[], k, curn)
    for i ← 0 to k-1
        if curn+k-1<maxf then
            return
        end if
        if curn+maxleft[leftf[i]]<maxf then
            return
        end if
        x ← 0
        for j ← i+1 to k-1
            if G.graph[leftf[i]][leftf[j]] == true then
                newleftf[x++] ← leftf[j]
            end if
        end for
        currentf[top++] ← leftf[i]
        curprice ← curprice+price[leftf[i]]
        dfs(newleftf,x,curn+1)
        curprice ← curprice-price[leftf[i]]
        top ← top-1
    end for
end function
```

Chapter 3：Testing Results

Case 1：The sample

Input data	Output data
16 20 001 002 003 004 004 005 005 006 006 007 007 008 008 003 009 010 009 011 009 012 009 013 010 014 011 015 012 016 012 017 013 018 020 99 019 99 018 4 017 2 016 3 015 6 014 5 013 1 012 1 011 1 010 1 009 10 008 1 007 2 006 5 005 3 004 4 003 6 002 1 001 2	12 002 004 006 008 009 014 015 016 017 018 019 020 239

This case is exactly the sample.

Well, it’s clear that it must be right.

Case 2：Discrete fruit number

Input data	Output data
13 20 011 023 034 031 031 012 012 014 014 017 017 019 019 034 029 010 029 011 029 012 029 013 010 014 011 015 011 13 023 28 034 41 031 23 012 37 014 68 017 51 013 2 012 3 011 4 010 5 029 10 015 1 007 2 006 5 005 3 004 4 003 6 002 1 001 2	14 001 002 003 004 005 006 007 010 012 013 015 017 023 034 154

This is a case which is similar with the sample.But the fruit number is discrete.

This output is right,for we can exam it from the input.

Case 3：Same numbers but different price

Input data	Output data
96 12 001 007 001 008 001 009 001 010 001 004 001 005 001 006 001 011 002 007 002 008 002 009 002 010 002 004 002 005 002 006 002 011 003 007 003 008 003 009 003 010 003 004 003 005 003 006 003 011 012 007 012 008 012 009 012 010 012 004 012 005 012 006 012 011 007 001 007 002 007 003 007 012 007 004 007 005 007 006 007 011 008 001 008 002 008 003 008 012 008 004 008 005 008 006 008 011 009 001 009 002 009 003 009 012 009 004 009 005 009 006 009 011 010 001 010 002 010 003 010 012 010 004 010 005 010 006 010 011 004 001 004 002 004 003 004 012 004 007 004 008 004 009 004 010 005 001 005 002 005 003 005 012 005 007 005 008 005 009 005 010 006 001 006 002 006 003 006 012 006 007 006 008 006 009 006 010 011 001 011 002 011 003 011 012 011 007 011 008 011 009 011 010 004 5 005 6 006 7 011 12 001 10 002 11 003 12 012 21 007 12 008 13 009 14 010 15	4 004 005 006 011 30

This case is for if there are two solution with the same maximum number of safe fruits,however their price is different.According to the rules,we should output the lower one.

This is actually made the graph like this:

- 1,2,3,12 these four are connected each other, they can be safe fruits.
- 4,5,6,11 these four are connected each other, they can be safe fruits.
- 7,8,9,10 these four are connected each other, they can be safe fruits.

And it’s clear that 4,5,6,11 this group has the smallest cost.

Case 4: Maximum data

Input data	Output data
Check it in *test.txt, please.*	25 002 004 006 008 010 019 025 027 037 038 042 043 045 049 050 054 065 067 072 082 091 093 096 097 100 14693

This is for the largest number test. So we take the maximum number of fruits, make the graph extremely complex, and check whether the algorithm can get the correct answer quickly. Because the input and output are too large, I put it in the attachment.

Our algorithm actually completed the task very quickly, but because there is no requirement for the speed of the algorithm, just check whether it is correct, we are correct, I am very confident about this.

Case 5: Only one fruit

Input data	Output data
3 3 001 002 002 003 003 001 001 3 002 4 003 5	1 001 3

A simple case if the fruit is very small that there is only one safe fruit. It's right.

Case 6: All fruits

Input data	Output data
0 7 001 5 002 1 003 12 004 14 005 21 006 1 007 1	7 001 002 003 004 005 006 007 55

A extreme case if all fruit is safe fruit. It’s obviously right.

Chapter 4：Analysis and Comment

Algorithm analysis

In fact, we need to find the largest independent set of a graph, that is, to find the largest clique of a complementary graph. The largest clique is the largest fully connected component of a graph, and the problem of finding the largest clique is *NPC*, that is, we can only get the result by guessing and verifying. Here, our group chooses *DFS + pruning* to reduce the search time, which is essentially a backtracking algorithm.

The following is about the time complexity analysis of our group's algorithm.

During *DFS*, we have N choices in the first step, $N-1$ in the second step, $N-2$ in the third step, and so on. Hence there are $N!$ choices during the whole *DFS* so that we call the function *DFS* $N!$ times. Besides, we scan the nodes in graph G by $O(N)$ in every function call so that the globle time complex is $O(N \cdot N!)$ which is unacceptable for $n, m \leq 100$.

But with pruning, we can pass the maximum data. That shows the great power of pruning.

For space complex, we use adjacency matrix to store the graph so that our space complex is $O(N^2)$.

Comment

As for this project, I'm very clear that our programmers don't use a better algorithm to find the maximum clique, because there is obviously a better backtracking algorithm, that is, the Bron-Kerbosch algorithm, but I think it's fine. Our programmer have tried his best to write a less perfect code, and he has tried his best, because he said he can't understand Bron-Kerbosch Algorithm, in fact, as far as this problem is concerned, as far as several sample inputs I put forward, the time of using the Bron-Kerbosch algorithm is slower than that of our team members' pruning, even after the optimization of the Bron-Kerbosch algorithm, I think its essence is the time change caused by different input data, In addition, our group's programmers refer more to the number of nodes in the group in pruning, that is, if the number of nodes that a node may get is less than the current number of nodes in the largest group, then prune this branch, which is not considered by the Bron-Kerbosch algorithm. in fact, the Bron-Kerbosch algorithm is originally an algorithm which is only a recursive backtracking algorithm, and it can not be the best choice in every situation, just like for the maximum clique problem, there is a slightly better priority branch boundary algorithm in time than backtracking algorithm. But I think this code can also be improved. In terms of duplicate search, you can refer to the Bron-Kerbosch algorithm to

better delete some nodes of duplicate search. But I think this code is quite fast, which should be the fastest code that our group programmers can write.

Declaration

We hereby declare that all the work done in this project titled "Performance Measurement (POW)" is of our independent effort as a group.

Duty Assignments:

Programmer: xxx
Tester: xxx
Report Writer: xxx

Appendix: Source Code

```
//ADSproject2 safe fruit
#include <stdio.h>
#include <stdlib.h>
#define INF 1000000
typedef enum
{
        false=0,true=1
}bool;
typedef struct fgraph//fruit graph
{
        int v;//quantity of fruits
        bool graph[1000][1000];//since all fruits are expressed by 'xxx',we can use 000-99
}FG;

FG      G;//define a fruit graph

int price[1000];//price[i] to express  the price of fruit i
int bestprice=INF;// the least price of max fruits  set
int curprice=0; //the price of currenr price set
int currentf[1000];//what fruits in the current clique,it is a stack
int top=0;//the stack top of currentf
int fruits[1000];//these fruits order is not continous,so we need an array to store these
int maxf=0;//max quantity of the fruits in clique
int maxclique[1000];//to store the max clique of the fruits
int maxleft[1000];//to store how many fruits can there exists in clique from fruits[i] to
void graphinitial(int m);//to initialize the graph
void FindMaxFruit();//to find the max clique of the fruits
void dfs(int *leftf,int k,int curn);//use dfs to backtrack,curn means the quantity of the

int cmp(const void *a, const void *b)//the function parameter of  qsort()
{
     return(*(int *)a-*(int *)b);
}
int main()
{
        int i,j,n,m;
        int f1,f2,x;
        scanf("%d %d",&n,&m);
        graphinitial(m);//initialize the graph

        for(i=0;i<n;i++)//input the fruits can't be eaten together
        {
                scanf("%d %d",&f1,&f2);
                G.graph[f1][f2]=0;//f1,f2 can't be eaten together
                G.graph[f2][f1]=0;
        }
        for(i=0;i<m;i++)//assign the fruits with their price
        {
                scanf("%d %d",&f1,&x);
                price[f1]=x;
                fruits[i+1]=f1;//from fruits[1] to fruits[G.v] store the order of these fr
        }
```

```
        FindMaxFruit();//to find the clique with max quantity and min  price
        //output
        printf("%d\n",maxf);//out put the max quantity
        int a1=1,p=0;//a1 used to count, p is the total price

        qsort(maxclique,maxf,sizeof(int),cmp);//store the fruits order in ascending order

        for(i=0;i<maxf;i++)//print the fruits
        {
                printf("%03d%c",maxclique[i],a1!=maxf?' ':'\n');
                a1++;
                p+=price[maxclique[i]];
        }
        printf("%d",p);//print the price
        return 0;
}

void graphinitial(int m)
{
        G.v=m;
        int i,j;
        for(i=0;i<=999;i++)//to make every vertex connected with the other  vertexs
        {
                for(j=0;j<=999;j++)
                {
                        if(i!=j)//the vertex isn't connected with itself. to avoid infinit
                        {
                                G.graph[i][j]=1;
                        }
                        else
                        {
                                G.graph[i][j]=0;
                        }
                }
        }
}

void FindMaxFruit()
{
        int i,j,k;
        int leftf[100];//leftf records the fruits which can eat from fruits[i+1] to fruits
        for(i=0;i<=999;i++)
        {
                maxleft[i]=INF;//initialize maxleft
        }
        for(i=G.v;i>=1;i--)//tranverse from G.v can optimize the performance
        {
                k=0;//k is the stack top of leftf
                currentf[top++]=fruits[i];//store fruits[i] into currentf
                curprice+=price[fruits[i]];//add the price to curprice
                for(j=i+1;j<=G.v;j++)
                {
                        if(G.graph[fruits[i]][fruits[j]])leftf[k++]=fruits[j];//leftf reco
```

```
                }
                dfs(leftf,k,1);
                curprice-=price[fruits[i]];//minus the price
                top--;//pop the fruit from the currentf
                maxleft[fruits[i]]=maxf;//record the fruits can br put in the clique from
        }
}

void dfs(int *leftf,int k,int curn)//leftf records the fruits form i+1 to G.v which can be
{
        int i;
        if(k!=0&&curn>maxf)//Updating the maxf early can help improve the  efficiency
        {
                maxf=curn;
                bestprice=curprice;
        }
        if(k==0) {//This is the last node of the maxclique, it won't have any other edges
                if (curn>maxf){
                        maxf=curn;
                        bestprice=curprice;
                        for(i=0;i<top;i++)
                        {
                                maxclique[i]=currentf[i];// put the fruits into maxclique
                        }
                }
                else if(curn==maxf)
                {
                        if(curprice<bestprice)
                        {
                                for(i=0;i<top;i++)
                                {
                                        maxclique[i]=currentf[i];// put the fruits into
                                                                    maxclique
                                }
                                bestprice=curprice;
                        }
                }
                return;
        }
        int newleftf[100],j,x;
        for(i=0;i<k;i++)//tranverse the  leftf
        {
                x=0;
                //pruning
                if(curn+k-i<maxf)return;//if the curn+(fruits left)<maxf,then prun
                if(curn+maxleft[leftf[i]]<maxf)return;//if the curn+(fruits can be in the
                for(j=i+1;j<k;j++)
                {
                        if(G.graph[leftf[i]][leftf[j]])newleftf[x++]=leftf[j];//add the fr
                }
                currentf[top++]=leftf[i];
                curprice+=price[leftf[i]];
                dfs(newleftf,x,curn+1);
                curprice-=price[leftf[i]];
```

```
                top--;
        }
        return;
}
```