Safe Fruit

date:2022-4-4

Chapter 1 Introduction

In the question, there are several kinds of fruits that cannot be eaten at the same time and the corresponding prices of all fruits. We want to find the most number of combinations of fruits that can be eaten together. If there are multiple combinations, then we need to output the combination with the lowest total price.

Chapter2 Method

DFS recursively traverses the nodes in the graph, selects the nodes that meet the requirements and pushes them into the stack. When traversing the last layer, the paths in the stack are pushed into another path stack, and the paths back to the upper layer until all paths are found

Find the longest path from the found path, calculate the total price of each longest path, and press the price and corresponding path into map. Since the map is automatically sorted, you only need to enter the path and price of the first element of the map

The disadvantage of this approach is that it can cause a large memory overhead.

Chapter3 Testing results

(We list the output at the end of the table to verify correctness)

	testing reason	testing sample	testing result	correct result
1	The example they gave us	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	12 002 004 006 008 009 014 015 016 017 018 019 020 239	12 002 004 006 008 009 014 015 016 017 018 019 020 239

	testing reason	testing sample	testing result	correct result
		004 4 003 6 002 1 001 2		
2	Verify whether the correct result can be obtained if the serial number is interrupted in the middle	2 5 001 002 011 012 001 5 011 3 012 6 002 2 003 3	3 002 003 011 8	3 002 003 011 8
3	Verify the smallest sample case	0 1 005 3	1 005 3	1 005 3
4	Verify that serial numbers do not start at 1	1 5 011 012 011 3 012 6 002 2 003 3	3 002 003 011 8	3 002 003 011 8

In summary, the correctness of the code is verified

Chapter4 Analysis of the algorithm

The time complexity of the recursive function is the number of nodes in the recursion tree, and the space complexity is the height of the recursion tree

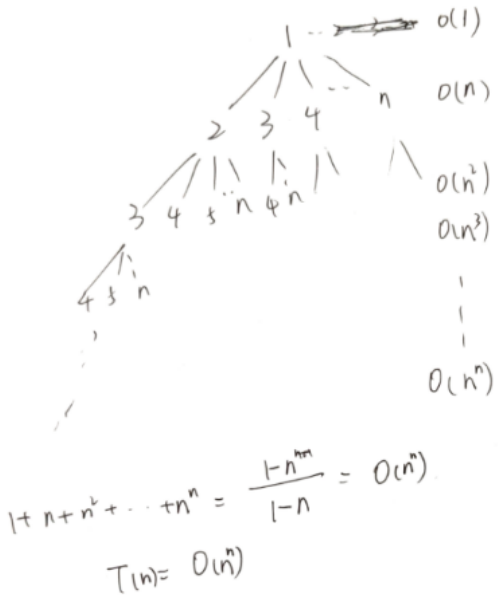
Complexity of Space

According to the recursive relationship, it is necessary to traverse each node whose serial number is larger than the current node, so the sum of nodes in each layer of the tree is a geometric sequence with n as the common ratio. By summation of the geometric sequence, the time complexity is O(n^n).

Complexity of Time

According to the figure, the height of the recursion tree is N, so the time space complexity is O(n).

the picture of tree



Chapter5 Appendix

```
#include<iostream>
#include<vector>
#include <map>
#define maxn 1005
using namespace std;
int Fruits[maxn][maxn]={0};//建立表示图的二维数组
int n,m;
int price[maxn]={0};//记录价格
vector< vector<int>> setof;//将dfs中每次走的路径存到其中
map<int,vector<int>> paths;//选出所有路径中符合要求的
vector<int> s;//记录dfs每次所走的路径
int temp=0;
bool backtrace(int x)//看新添加的水果能不能和已经添加到路径中的水果一起吃
{if(x==0)return true;
    for(int i=0;i<s.size();i++)
    {
        if(Fruits[s[i]][x]==1)//如果不能
        return false;//则返回不能
    }
    return true;//反之则返回可以
}
void dfs(int index)//dfs遍历图
{if(s.size()>m)//如果所有点都已经入栈
return;//剪枝退出
    s.push_back(index);//新遇到一个点则将其入栈
    for(int i=index+1;i<maxn;i++)//遍历比当前节点大的有价格的节点
    {//如果此节点与当前节点连接
        if(Fruits[index][i]==0&&price[i]!=0&&backtrace(i)==true)//并且可以和之前已经
遍历的水果一同食用
            dfs(i);//将此节点入栈
    }
    setof.push_back(s);//如果没有下一个符合条件节点，则将这条路径入栈
    s.pop_back();//将路径的最末端一位出栈
}
int main()
```

```
{
    cin>>n>>m;
    int x,y;
    for(int i=0;i<n;i++)
    {
        cin>>x>>y;//读入图中
        Fruits[y][x]=Fruits[x][y]=1;//不能一起吃
    }
    for(int i=0;i<m;i++)
    {
        cin>>x>>y;
        price[x]=y;//记录价格
    }
    dfs(0);
    int max=0;
    for(int i=0;i<setof.size();i++)
    {
        if(setof[i].size()>max)
        max=setof[i].size();//从所有路径中找出最长路径的值
    }
    cout<<max-1<<endl;
    for(int i=0;i<setof.size();i++)
    {
        if(setof[i].size()==max)//找出最长路径
        {int perprice=0;
            for(int j=0;j<setof[i].size();j++)
            {
                perprice+=price[setof[i][j]];//计算最长路径的价格
            }
            paths.insert(pair<int, vector<int>>(perprice,setof[i]));//将路径及其价
格压入map中
        }
    }
    auto itor = paths.begin();//
    printf("%03d",itor->second[1]);
        for(int j=2;j<itor->second.size();j++)
            {
                printf(" %03d",itor->second[j]);//按格式输出
            }
        cout<<endl<<itor->first;//由于map已经自己排序好了，所以可以直接输出
}
```