project3

小组二

2024 年 4 月 13 日

题目要求

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 cannot be eaten with cantaloupe (哈密瓜), otherwise it will lead to kidney deficiency (肾虚).

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.

Input Specification

Each input file contains one test case. For each case, the first line gives two positive integers: N, the number of tips, and M, the number of fruits in the basket. Both numbers are no more than 100.

Then two blocks follow:

1. The first block contains N pairs of fruits which must not be eaten together, each pair occupies a line and there is no duplicated tips.
2. The second block contains M fruits together with their prices, again each pair in a line. To make it simple, each fruit is represented by a 3-digit ID number. A price is a positive integer which is no more than 1000. All the numbers in a line are separated by spaces.

Output Specification

For each case:

- First print in a line the maximum number of safe fruits.
- Then in the next line list all the safe fruits, in increasing order of their IDs. The IDs must be separated by exactly one space, and there must be no extra space at the end of the line.
- Finally, in the third line print the total price of the above fruits. Since there may be many different solutions, you are supposed to output the one with a maximum number of safe fruits. In case there is a tie, output the one with the lowest total price. It is guaranteed that such a solution is unique.

算法描述

这段代码实现了一个用于解决“安全水果选择”问题的深度优先搜索（DFS）算法。问题的核心是从一个大篮子中选择出最多数量的可以安全共同食用的水果，如果有多种选择方法，则选择总价最低的组合。

算法概述

- **输入处理:**首先读取不可以一起食用的水果对,这些对储存在 `deficiency` 数组中。如果 `deficiency[i][j]` 为 `false`，则表示水果 i 和水果 j 不能一起食用。接着读取每个水果及其价格，存储在 `price` 数组中。
- **初始化：**使用 `memset` 将 `deficiency` 初始化为 `true`，然后根据输入调整为 `false` 表示不能共同食用的水果对。

深度优先搜索（DFS）

- 函数 `DFS` 被用于尝试各种不同的水果组合，从而找到最大数量且成本最低的安全食用组合。
- 参数 `pos` 表示当前选择的水果 ID，`sum` 是到目前为止已选择的水果数量，`money` 是当前已选水果的总价格。

剪枝优化

- 如果从当前水果 pos 开始，即使加上后续可能的最大选择数量，其总和仍然小于已知的最大数量 ansNum，则直接返回，不再继续搜索。
- 如果当前选择的水果与已选择的任何一个水果有冲突（即不能一起食用），则跳过当前选择。

更新最优解

- 如果当前的水果组合数量超过已知的最大数量 ansNum，更新 ansNum，记录当前组合到 ansFruit，并更新总价 ansMoney。
- 如果当前的水果组合数量等于 ansNum 但价格更低,则同样更新 ansFruit 和 ansMoney。

主函数中的循环

从每个可能的水果开始，使用 DFS 尝试构建有效的水果组合，从而确保每种可能的起点都被考虑到。

结果输出

- 输出可以安全食用的最大水果数量。
- 输出所有选择的水果的 ID。
- 输出这些水果的总价格。

特点和考虑

- 通过 DFS 探索所有可能的水果组合，能够有效处理组合优化问题。
- 使用剪枝技术，提高搜索效率，避免不必要的计算。
- 此算法在问题规模较大时可能会遇到性能瓶颈，但对于问题给定的规模（水果和提示数量都不超过 100），这种方法是可行的。

代码

```
#include <bits/stdc++.h>
using namespace std;
#define N 105
int numTips, numFruit;
bool deficiency[N][N]; // deficiency[i][j]=false 表示 i 和 j 不能一起吃。deficiency[i][j]=true 表示 i 和 j 可以一起吃。
int price[N];
int fruit[N]; // fruit[] 记录当前选择的水果的状态。sum 记录已选择的水果数量。
int ansFruit[N], ansNum = -1, ansMoney = 0; // ansFruit 记录最大水果集合，ansNum 记录最大数量的水果。ansMoney 记录最小金额。

int sumFruit[N]; // sumFruit[i]: 如果我只从 'i 到 numFruit' 选择水果，我能选择的最大水果数。

void DFS(int pos, int sum, int money) // pos: 现在我选择了水果'pos'，已经选择了 'sum' 数量的水果。money: 当前花费的总金额。
{
    //if(now == numFruit) { CHECK(); return; }
    for(int i = pos + 1; i <= numFruit; ++i)
    {
        // 如果现在选择的水果数量加上从 i 到 numFruit 可以选择的最大水果数仍然小于 ansNum，我们可以忽略剩余的情况。
        if(sumFruit[i] + sum < ansNum) return;
        // 如果这个水果不能和已经选择的水果一起吃，则跳过。
        if(deficiency[i][pos] == false) continue;
```

```
        bool flag = true;
        for(int j = 0; j < sum; ++j) if(deficiency[
            fruit[j]][i] == false) { flag = false;
            break; }
        if(flag == false) continue;
        // 将这个水果添加到我们当前的选择中
        fruit[sum] = i;
        DFS(i, sum + 1, money + price[i]);
    }
    if(sum > ansNum) // 记录最大数量的水果和详细信息
    {
        ansNum = sum;
        for(int i = 0; i < sum; ++i) ansFruit[i] =
            fruit[i];
        ansMoney = money;
    }
    else if(sum == ansNum && money < ansMoney) // 检查
        这种选择是否可以用更少的钱
    {
        for(int i = 0; i < sum; ++i) ansFruit[i] =
            fruit[i];
        ansMoney = money;
    }
    return;
}

int main()
{
    memset(deficiency, 1, sizeof(deficiency));
    scanf("%d%d", &numTips, &numFruit);
    for(int i = 0, x, y; i < numTips; ++i)
    {
        scanf("%d%d", &x, &y);
```

```
        deficiency[x][y] = deficiency[y][x] = 0;
    }
    for(int i = 0, x, y; i < numFruit; ++i)
    {
        scanf("%d%d", &x, &y);
        price[x] = y;
    }

    for(int i = numFruit; i > 0; --i)
    {
        fruit[0] = i;
        DFS(i, 1, price[i]);
        sumFruit[i] = ansNum;
    }

    printf("%d\n", ansNum);
    printf("%03d", ansFruit[0]);
    for(int i = 1; i < ansNum; ++i) printf(" %03d",
       ansFruit[i]);
    printf("\n%d\n", ansMoney);
    return 0;
}
```

Listing 1: main.cpp

生成测试数据的程序

```
import random

def generate_test_data(num_tips, num_fruits):
    fruits = [f"{i:03d}" for i in range(1, num_fruits
       + 1)]
    tips = []
    prices = []
```

```
    # Generate tips (pairs of fruits that cannot be
       eaten together)
    while len(tips) < num_tips:
        f1, f2 = random.sample(fruits, 2)
        pair = f"{f1} {f2}"
        reverse_pair = f"{f2} {f1}"
        if pair not in tips and reverse_pair not in
           tips:
            tips.append(pair)

    # Generate prices for each fruit
    for fruit in fruits:
        price = random.randint(1, 100)  # Prices from
           1 to 100
        prices.append(f"{fruit} {price}")

    return num_tips, num_fruits, tips, prices

# Example usage
num_tips = input("输入不能同时吃的水果数量:  ")

num_fruits = input("输入所有的水果数量:  ")
num_tips, num_fruits, tips, prices =
   generate_test_data(num_tips, num_fruits)

print(f"{num_tips} {num_fruits}")
for tip in tips:
    print(tip)
for price in prices:
    print(price)
```

Listing 2: 测试数据生成程序，language

测试

测试 1

```
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
```

output1

```
12
002 004 006 008 009 014 015 016 017 018 019 020
239
```

input2

```
22 32
014 017
030 007
022 008
016 009
014 016
023 015
005 017
002 015
019 009
018 008
012 011
032 024
016 020
015 014
029 013
028 013
004 013
028 003
```

003 023
022 027
005 007
032 005
001 93
002 11
003 72
004 52
005 5
006 42
007 27
008 74
009 80
010 71
011 43
012 3
013 65
014 23
015 60
016 57
017 34
018 40
019 71
020 78
021 35
022 24
023 78
024 7
025 98
026 68
027 76
028 26
029 61

```
030 60
031 48
032 66
```

output2

```
21
001 002 004 005 006 010 012 014 018 019 020 021 022 023 024 025 026 028 029 030 031
994
```

时间复杂度分析

深度优先搜索 (DFS)

在 DFS 函数中，对于每个可能的水果选择，都有一个递归调用，这个调用依赖于两个主要因素：当前位置 pos 和剩余可选的水果数量。在最坏的情况下，每次递归可以选择所有剩余的水果。设 M 是水果的总数，最坏情况下，DFS 的时间复杂度是指数级的，大约为 $O(2^M)$。这是因为每个水果都有被选中或不被选中两种可能，形成了一个二叉决策树。

剪枝优化

代码中使用了剪枝技术，尤其是在判断 $\texttt{sumFruit}[i] + \texttt{sum} < \texttt{ansNum}$ 时直接返回的逻辑。这种剪枝有助于减少不必要的递归调用，因为如果当前路径已经不可能超过已找到的最优解，就没有继续探索的必要。这种优化虽然有助于减少运行时间，但在最坏情况下仍然是指数级。