浙江大学

Advanced Data Structures and Algorithm Analysis

Laboratory Projects

Safe Fruit

Group 15

聂俊哲 张琦 张溢弛

Date: 2020-04-10

Content

Chapter 1: Introduction

- Food Safety is a significant problem for us nowadays. If we eat some fruit at the same time, we may get kidney deficiency or other disease. **We should keep healthy especially in such a hard period of time as the COVID-19 is still a big challenge to us.** In this project, we will find the safe fruit combination with largest amount and lowest cost, we could call such combination of fruit a 'best' combination. It is guaranteed that such a solution is unique.
- The input data will be in such form
    - two integer n and m
    - n combinations of unsafe fruit which could not be eaten together
    - m kinds of fruit with their cost
- The output data will be in such from
    - The number of fruit in the best combination
    - The 3-digit ID of fruit in the best combination in increasing order
    - The cost of this combiantion

Chapter 2: Data Structure / Algorithm Specification

- Some **improtant data structure** in our program
    - unsafe[ ][ ]: the adjacency table of the fruits that can not be eaten together
    - fruit[ ][ ]: the realtion matrix of all the fruits **you can buy in the store**, where fruit[i][j]=1 if they can not be eaten together otherwise 0
    - s[ ][ ]: a table to store all the safe fruit of each fruit i in the backtracking
- The **pseudo-code of our program** is as following

```
1  //main function
2  ALGORITHM:Safe Fruit
3  Procedure:Safe Fruit
4  INPUT: 2 integers n and m, m pairs of unsafe fruit, n
5  OUTPUT: The number of combination, 3 digit IDs of fruits and cost of this
   combiantion
6
7  read in n and m
8  rean in n lines of unsafe fruit and fill in the adjacency list
9  read in m lines of fruit and their price
10 give the m fruits a new ID in the order of input to simplify the problem
11 traverse the adjacency list and fill in the relation matrix
12 backtracking()
13 print the result in a proper form
14
15
16 //backtracking function
17 ALGORITHM:backtracking
18 Procedure:backtracking
19 INPUT: n, the ID of this fruit, num, the number of unsafe fruit of that
   fruit, step, the backtracking level.
20 OUTPUT: None
21
22 if num == 0:
23     check the result of this backtracking
24     update the result if it's better
25     finishing the backtracking
26
```

```
find all the unsafe fruit of this fruit in the list
put them in a list
while the list is noe empty:
    get one element of the list
    add it to the combination
    begin next backtracking
    move it from the combination
```

Chapter 3: Testing Results

- Test 1: the same as sample, a normal condition

```
Input 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

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

- Test 2: all the fruit can be eaten together

```
Input data:
0 5
001 100
003 200
020 3
002 4
005 300

Output data:
5
001 002 003 005 020
607
```

- Test 3: you can buy no fruit in the store

```
Input data:
3 0
001 002
002 003
003 004

Output data:
0


0
```

- Test 4: **some unsafe fruit can not be bought from store**, which means the unsafe fruit tips will be a waste of time to read

```
Input data:
5 5
001 002
002 099
099 088
088 077
003 077
005 100
001 80
002 3
004 12
003 30

Output data:
4
002 003 004 005
145
```

- Test 5: an edge cases with the max n normal m

```
Input data:
n = 999 and m = 50, you can get the data in the "bigtestdata.txt"

Output data:
47
022 027 028 050 055 111 112 119 156 157 160 181 200 215 232 233 243 298
302 358 369 373 374 399 399 452 498 541 563 571 580 635 641 642 658 740
778 793 817 827 876 909 933 962 966 969 980
22994
```

- Test 6: an an edge cases with the max m normal m

```
Input data:
n = 50 and m = 100, you can get the data in the "bigtestdata.txt",we think
the m should not be too large because it's the main crimer of running time

Output data:
99
006 007 017 021 039 062 073 114 118 133 142 152 177 183 197 209 211 228
241 245 254 262 270 271 276 284 289 292 292 292 305 305 306 310 312 335
337 362 385 386 393 427 440 444 446 449 457 482 483 486 496 512 512 514
539 561 582 586 588 596 599 607 609 613 650 657 665 671 681 682 683 689
691 704 706 710 715 715 716 722 730 737 796 829 844 848 858 875 883 893
900 901 925 926 942 984 988 992 992
51671
```

- Test 7: an edge cases with the max n and m

```
Input data:
n = 999 and m = 100, you can get the data in the "bigtestdata.txt"

OutPut data:
95
006 021 023 029 037 052 055 063 066 067 088 160 171 172 191 195 200 218
223 235 237 256 263 276 278 279 282 299 300 311 324 350 350 356 363 363
366 383 386 389 411 422 454 471 483 487 492 523 532 533 545 552 569 582
613 615 615 626 629 631 652 652 654 656 692 694 697 702 716 728 736 764
769 774 795 807 807 811 812 812 813 821 828 833 840 841 884 892 899 933
963 966 976 977 979
49471
```

Chapter 4: Analysis and Comments

- The **background** of out Program: **Bron-Kerbosch Algorithm**
    - Bron-Kerbosch Algorithm is an algorithm to find the max clique of a graph G, which inludes the largest amount of vertices that are connected with each other.
    - In this program, if 2 fruits couldn't be eaten together, we could regard them as unconnected and the max clique of such a graph can be the result
    - Bron-Kerbosch Algorithm is implentmented by backtracking in out program
    - The pseudo-code of the Bron-Kerbosch Algorithm is

```
ALGORITHM:Bron-Kerbosch
Procedure:Bron-Kerbosch
INPUT: a graph G, 2 set P and R
OUTPUT: the max clique R
if G and P are both empty:
    return R as the result
for each vertex v in G
    put v in P
    find all the vertices connected with v in G and R,named as g and
r
    Bron-Kerbosch(g,P,r)
    remove v from G
    add v to r
```

- **Comments** about the program
  - According to the statement of project2, we should learn that the unsafe fruit in the tips **might not able to be bought from the store**, so when we create a relation matrix of fruit, we should use the m kinds fruit you can by as standard.
  - The input order of m fruit is not regular, so we make a transfrom on these input data, that is make a map from the input order the its real 3-digit ID and a map from its ID to its input order.
  - Prunning is necessary to speed up the running time, otherwise you will get a time-out
- Algorithm analysis
  - Space complexity
    In the program, the main cost of memory space is from the amounts, we use several arrays in our program such as `fruit[][], Maxnumber[], s[][], unsafe[][], temp[].path[], price[], list1[], list2[]`ect, the size of some arrays depends on n, the number of pairs of unsafe fruit and the size of other arrays depends on m, the number of fruit you can buy, and we should notice that some of the arrays are 2-dimension array, and all the array up to unsafe fruit is $O(n)$ because we have n pairs of unsafe fruit, so the total time complexity is $O(m^2+n)$.
  - Time complexity

    The main cost of running time comes from the following steps
    - Read in the data: it's $O(m+n)$ obviously because we should read in m lines of unsafe fruit and n lines of fruit you can buy in the store
    - Set the relation matrix of fruits, it's $O(n)$ because we should traverse the n pairs of unsafe fruit and write a '1' in to the array of $O(m^2)$ named `fruit[][]`
    - Backtracking:

      Now we give the time complexity analysis:

      We begin with the following definitions:

      (1) Let $T(n)$ be the worst-case running time of $Bron-Kerbosch(Q,S)$ when vertexes of Q is n. (Because parameter P in $Bron-Kerbosch(Q,S)$ doesn't affect this analysis, we omit it in this module.)

(2) Let $T_k(n)$ be the worst-case running time of $Bron-Kerbosch(Q,S)$ when vertexes of Q is n, and vertexes of $S-N\{v\}$ is k that v makes vertexes of $S\cap N\{v\}$ in the first entrance.

So, we have

$$T(n)=T_k(n)_{max}\leq\sum_{i=1}^{k}T(n-i)+P(n) \tag{1}$$

where $P(n)$ means a nonrecursive procedure time complexity that is $O(n^2)$, which is much smaller than $O(3^{\frac{n}{3}})$ which is the time complexity of $T(n)$ and we will prove this in the latter part.

Then, the formula will be simplified as

$$T(n)\leq\sum_{i=1}^{k}T(n-i)+O(n^2) \tag{2}$$

which can be furtherly simplified to

$$T(n)\leq k\times T(n-i)_{max}+O(n^2) \tag{3}$$

Because it can iterate for k times, $T(n-k)$ is the largest one

$$T(n)\leq O(k^{\frac{n}{k}})_{max} \tag{4}$$

It's easy to prove that when k equals 3, $O(k^{\frac{n}{k}})$ can be maximized.

So the time complexity is $O(3^{\frac{n}{3}})$.

- Print the result: It's $O(m)$ because the size of output data couldn't be larger than the size of fruit you can buy, which is $O(n)$.

Appendix: Source Code (if required)

```c
#include <stdio.h>
#include <stdlib.h>

// The max size of test data
#define MAX 1000
#define MAXNUM 100000

// The graph relation of the fruits
int fruit[MAX][MAX] = {0};
// The max number of safe fruit after this fruit
int Maxnumber[MAX];
// a stack that will be used to store the 'safe fruit' of fruit i
int s[MAX][MAX];
// a matrix to store the unsafe fruits of each fruit
// unsafelen is the length of each fruit's unsafe list
int unsafe[MAX][MAX], unsafelen[MAX] = {0};
// temp[] is The possible combination of safe fruits in the backtrack
// path[] is the final result
// templen and pathlen are the length of the two array
int temp[MAX], path[MAX];
int templen = 0, pathlen = 0;
// price[] store the cost of each kind of fruit
// tmp is the possible totoal cost of a safe fruit combination, and cost
is the final result
// number is the final amount of fruit
int price[MAX], tmp, cost, number = 0;

```

```
27 // the main function to backtracking
28 void dfs(int n, int num, int step);
29 // a self-definition function to compare 2 integers
30 int comp(const void *a, const void *b)
31 {
32     return *(int *)a - *(int *)b;
33 }
34
35 int main()
36 {
37     // n is the number of pairs of unsafe fruit
38     // m is the number of fruit you can buy
39     int m, n, a, b;
40     int i, j;
41     // read in m and n
42     scanf("%d %d", &n, &m);
43
44     for (i = 0; i < n; i++)
45     {
46         // read in the paris of unsafe fruits
47         scanf("%d %d", &a, &b);
48         unsafe[a][unsafelen[a]++] = b;
49     }
50     // make 2 array to store the fruit you can buy
51     int list1[MAX], list2[MAX];
52     for (i = 0; i < MAX; i++)
53     {
54         list1[i] = MAXNUM;
55         list2[i] = MAXNUM;
56     }
57
58     // you should learn that not all the fruits in the unsafe pairs can
   be bought from store.
59     // So we need to give them new lables, this time we remark the fruit
   in the order of input from 0 to m-1.
60     // and we use 2 arrays to store the corresponding relations between
   its origin lable and new lable
61     for (i = 0; i < m; i++)
62     {
63         scanf("%d %d", &a, &price[i]);
64         list2[i] = a;
65         list1[a] = i;
66     }
67
68     // now we construct a graph relation of the m kinds of fruits
69     // fruit[i][j] = 0 means they can eat safely one by one
70     for (i = 0; i < m; i++)
71     {
72         fruit[i][i] = 1;
73         // travle around the unsafe list of each fruit to mark all the
   fruit which can't be eaten together
74         for (int j = 0; j < unsafelen[list2[i]]; j++)
75         {
76             int k = unsafe[list2[i]][j];
77             // keep all the fruit in the graph are in the kind 0 to m-1
78             if (list1[k] < m)
79                 fruit[i][list1[k]] = fruit[list1[k]][i] = 1;
80         }
```

```
    }

    /* a period of test code to print the graph
    for(i=0;i<m;i++){
        for(j=0;j<m;j++){
            cout<<fruit[i][j]<<" ";
        }
        cout<<endl;
    }
    */

    //traverse all the fruit to find a max combination of safe fruit using dfs
    for (i = m - 1; i >= 0; i--)
    {
        // cout store the number of safe fruit of each fruit[i]
        int count = 0;
        // push all the safe fruit of fruit[i] in the stack
        for (j = i + 1; j < m; j++)
        {
            if (!fruit[i][j])
                s[0][count++] = j;
        }

        //start the backtracking, first push the node i into the temp_path
        temp[templen++] = i;
        tmp += price[i];
        //solve the problem using dfs, m is the number of fruit
        //and count is the number of safe fruit in the stack
        dfs(m, count, 1);
        templen--;
        tmp -= price[i];
        Maxnumber[i] = number;
    }
    // change the new lable into its origin lable
    for (i = 0; i < pathlen; i++)
        path[i] = list2[path[i]];
    // use the C library function qsort in <stdlib.h> to sort the fruits in increasing order
    qsort(path, pathlen, sizeof(int), comp);

    // print the answers in a proper format.
    printf("%d\n", number);
    for (i = 0; i < pathlen; i++)
    {
        if (!i)
            printf("%03d", path[i]);
        else
            printf(" %03d", path[i]);
    }
    printf("\n");
    printf("%d", cost);
    return 0;
}
//the most significant function to implement the backtracking algorithm
// n: the number of different fruit
// num: the number of safe fruit
```

```
// step: the level of backtracking
void dfs(int n, int num, int step)
{
    // when a backtracking finished
    if (num == 0)
    {
        // if find a better result with more fruit or less cost
        if (step > number || (step == number && tmp < cost))
        {
            //update the result and return, finish one backtracking
            cost = tmp;
            number = step;
            pathlen = templen;
            for (int i = 0; i < templen; i++)
            {
                path[i] = temp[i];
            }
        }
        return;
    }
    int i;
    // traverse the fruits in the 'safe stack' to begin the next level of backtracking
    for (i = 0; i < num; i++)
    {
        // count is the number of safe fruit
        int j, k, count = 0;
        k = s[step - 1][i];
        // special conditions that can end the backtracking early, prunning
        if (step + Maxnumber[k] < number)
            return;
        //repeat the method of begin a backtracking by find all the safe fruits
        for (j = i + 1; j < num; j++)
        {
            if (!fruit[k][s[step - 1][j]])
            {
                s[step][count++] = s[step - 1][j];
            }
        }
        // start a new backtracking
        temp[templen++] = k;
        tmp += price[k];
        // the level will up 1
        dfs(n, count, step + 1);
        tmp -= price[k];
        templen--;
    }
}
```

References

List all the references here in the following format:

[1] 《算法导论》第三版

[2] PTA website https://pintia.cn/

[2] Author, “Title of the book”, *Name of the publisher*, year

[3] Author, “Title of the article”, Web site links

Author List

- Zhang Yichi 3180103772
- Zhang Qi 3180103162
- Nie Junzhe 3180103501

Declaration

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

Signatures