# Project3: Safe Fruit


# Author: Jiang Yi


April 18, 2023

# Contents

# 1 Introduction

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 can not be eaten with cantaloupe, otherwise it will lead to kidney deficiency.

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.

## 1.1 Input Specification:

Each input file contains 1 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 2 blocks follow.

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.

The second one 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.

## 1.2 Output Specification:

For each case, we should do as follows:

1) print in a line the maximum number of safe fruits.

2) list all safe fruits in increasing order of their ID's. The ID's must be separated by exactly one space, and there must be no extra space at the end of the line.

3) print the total price of the above fruits. Since there may be many solutions, we should output the one with maximum number of safe fruits. In case there is a tie, output the one with the lowest total price.

# 2 Algorithm Description

In general, we use Bron–Kerbosch Algorithm with dfs and pruning to calculate the maximum fully connected component (clique) of the graph.

## 2.1 Model Establishing

We take every fruit as a vertex and all fruits form a graph. The relationship between vertexes(fruits) is whether they can be eat at the same time. If they(vertex with id $i$ and $j$) can be eaten simultaneously, we mark $G[i][j]$ to be safe(0), otherwise dangerous(1).

The input is the dangerous pair of fruits, which can be viewed as the edges in Complementary Graph($\bar{G}$). What we want to get is the max number of safe fruits in safe fruit set, which is similar to calculate the maximum fully connected component (clique) of the graph.

## 2.2 Algorithm Specification

**Algorithm 1** Main Algorithm in this Project

```
map input id to 1,2,...M
mark in graph matrix G the dangerous pairs of nodes
mx ← 0
for i = m − 1; i ≥ 0; i − − do
    possible nodes ← 0
    for j = i + 1; j < M; j + + do
        count and push possible nodes into Stack
    end for
    TempPath[top + +] ← i
    TempPrice[top + +] ← price[i]
    dfs(possible nodes, 1)
    TempPath.pop()
    TempPrice.pop()
    dp[i] ← mx
end for
print out the results
```

The algorithm shown above is the main process of solving the problem. In all, it is listing all vertexes and use dfs to judge whether the vertex can be added into the set of max clique of the graph or not. While the part shown below is the detailed descrption of how Bron–Kerbosch Algorithm with dfs works.

**Algorithm 2** Bron–Kerbosch + Pruning + dfs Algorithm in this Project

**Require:** $possible\ nodes,\ step$
  **if** $NO\ possiblenodes$ **then**
    **if** $step > mx \mid\mid (step == mx\ and\ TempPrice < TotalPrice)$ **then**
      modify the result
    **end if**
  **end if**
  **for** $i = 1; i \leq possible\ nodes; i++$ **do**
    $k \leftarrow Stack[step][i]$
    **if** $step + M - k < mx \mid\mid step + dp[k] < mx$ **then**
      return for pruning
    **end if**
    $count \leftarrow 0$
    **for** $j = i + 1; j < M; j++$ **do**
      count and push possible nodes from node with id $k$ to node with id $Stack[step][j]$ into $G[k][Stack[step][j]]$
    **end for**
    $TempPath[top++] \leftarrow k$
    $TempPrice[top++] \leftarrow price[k]$
    dfs(count, step+1)
    $TempPath.pop()$
    $TempPrice.pop()$
  **end for**

The algorithm shown above is the main process of Bron–Kerbosch Algorithm with dfs. In all, it is listing all vertexes whose id is larger than the current vertex(to some degree, it is a kind of pruning) and use dfs to judge whether the vertex can be added into the set of max clique of the graph or not. If no possible nodes can be added , check whether the max clique should be updated or not.

# 3 Test Results

## 3.1 PTA Results

I use the test sample on PTA to judge the correctness of my Algorithm and also designed test data by myself to check the correctness and the time complexity of my program.

| Submit Time | Status ⓘ | Score | Problem | Compiler | Memory | Time Used | User |
|---|---|---|---|---|---|---|---|
| 04/15/2023 3:39:56 PM | Accepted | 35 | 1021 | C++ (g++) | 604 KB | 176 ms | |

| Case | Result | Score | Run Time | Memory |
|---|---|---|---|---|
| 0 | Accepted | 18 | 4 ms | 328 KB |
| 1 | Accepted | 5 | 4 ms | 452 KB |
| 2 | Accepted | 1 | 4 ms | 452 KB |
| 3 | Accepted | 1 | 4 ms | 604 KB |
| 4 | Accepted | 1 | 4 ms | 452 KB |
| 5 | Accepted | 6 | 8 ms | 444 KB |
| 6 | Accepted | 3 | 176 ms | 448 KB |

Figure 1: PTA Test Result

The PTA test passed. And I also use random number generator to get my test samples. Here is my code for generating test data.

## 3.2 Random number generator for my test

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int base = 60, valid = 30;/*can be modefied by people*/
  int value, start, end;
  srand(time(NULL));
  printf("%d %d\n",base,valid);
  for(int i = 0 ; i < base ; i++)
  {
```

```
    do{
      start = rand() % base + 1;
      end = rand() % base + 1;
      while(start == end)
      start = rand() % base + 1;
      if(start > end) swap(&start,&end);
      value = 100 * start + end ;
    }while(hash[value] == 1);
    hash[value] = 1;
    printf("%03d %03d\n",start,end);
  }
  for(int i = 0 ; i < valid; i++)
    printf("%03d %d\n",i+1,rand()%1000+1);
  return 0;
}
```

## 3.3 My test results

Here are my test samples and results. Since some of the samples are too long, they're placed in Appendix.

Table 1: my test samples and results

| case | $n$ | $m$ | purpose | result |
|---|---|---|---|---|
| 1 | 16 | 20 | PTA sample | passed |
| 2 | 2 | 2 | small number for correctness | passed |
| 3 | 4 | 1 | correctness for dealing with invalid nodes | passed |
| 4 | 66 | 2 | correctness for dealing with invalid nodes | passed |
| 5 | 100 | 51 | correctness for extreme $n$ | passed |
| 6 | 45 | 100 | correctness for extreme $m$ | passed |
| 7 | 100 | 100 | correctness for extreme $m$ and $n$ | passed |
| 8 | 66 | 31 | correctness and time measurement for some $m$ and $n$ | passed |

I also plot the time wasted by the program when $M$ grows as follows. Since the numbers random number generator generates are not always good, I take average time of many tests and use average time for different size of $M$ to draw the result.

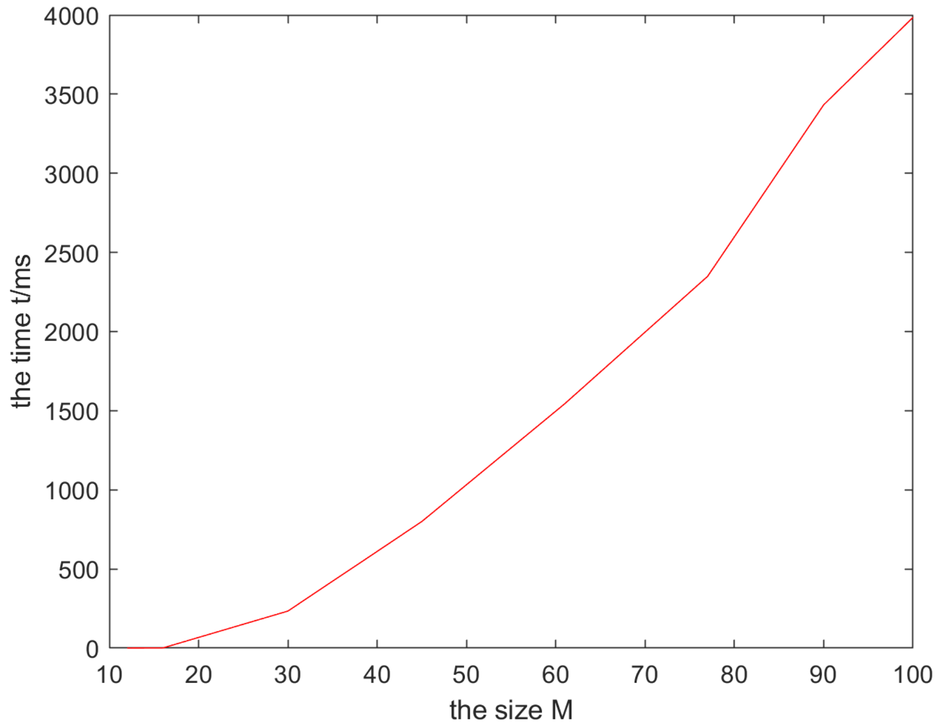


Figure 2: time and $M$ size with $N = M/2$

# 4 Analysis

From the results shown above, we can concluded that the Bron–Kerbosch Algorithm with dfs and pruning to calculate the maximum fully connected component (clique) of the graph is correct and solves problems in acceptable time.

## 4.1 Time Complexity

Considering the worst case that dfs with pruning is almost invalid. We have:

$$T(m) = O(2^{m/3})$$

Just as Robert Endre Tarjan and Anthony E. Trojanowski proved in their paper.

But this case is rare to appear. In most cases, we have:

$$T(n) = (T(n-n_1)+T(n_1))+(T(n-n_2)+T(n_2))+...+(T(n_k)+T(n-n_k))+k(1)$$

$n_1, n_2, ..., n_k$ represent the nodes not adjacent to the current node of dfs when starting to recursion again. Take $kO(1) = k$ for convenience. To use approximation, take $k << n$ and $n_1, n_2, ..., n_k$ almost the same $(n/2)$ to Solve this formula to get $T(n) = O(n \times k^{log_2^n})$.

### 4.2 Advantage and disadvantage of my Algorithm

My algorithm runs in acceptable time to solve the problems with different sizes and get correct answers. But my algorithm is sensitive to the input data, when the random data changes(the graph is not so good), my algorithm may run for long time to get the answer.

## 5 Conclusion

Bron–Kerbosch + Pruning + dfs Algorithm solves the problem Safe Fruits well. Though the time complexity is not satisfying in some extreme cases, the algorithm is suitable for calculating the maximum fully connected component (clique) of the graph.

# 6 Appendix

## 6.1 Test data with extreme $n$ and $m$

100 100

018 090 056 073 068 099 088 097 009 061 002 064 024 078 007 096 045 077 012 076 016 040 024 065 097 100 052 096 001 096 076 082 009 075 074 082 058 059 016 068 074 090 062 075 065 098 027 058 045 081 074 097 005 068 041 078 049 059 041 099 077 087 077 084 042 059 063 096 016 079 007 090 014 035 044 048 048 071 066 095 014 047 061 093 021 097 031 060 018 026 026 053 023 055 001 097 005 037 044 078 006 030 015 021 052 055 032 069 095 097 075 076 066 083 071 078 020 077 028 045 005 093 018 088 077 099 050 066 012 022 006 023 002 076 043 062 035 059 003 060 052 057 036 078 010 054 039 052 026 074 058 068 042 053 039 100 067 085 011 017 029 077 034 095 077 086 078 086 047 087 024 048 065 084 015 078 051 052 033 046 036 055 046 087 012 047 092 095 048 091 062 085 011 093 002 094 024 082 062 066

001 116 002 948 003 465 004 158 005 36 006 997 007 786 008 275 009 470 010 514 011 319 012 848 013 248 014 996 015 324 016 850 017 830 018 640 019 479 020 376 021 320 022 862 023 410 024 827 025 347 026 996 027 472 028 232 029 475 030 371 031 532 032 436 033 767 034 265 035 814 036 584 037 933 038 94 039 583 040 869 041 127 042 266 043 999 044 672 045 622 046 959 047 789 048 497 049 863 050 115 051 439 052 705 053 371 054 298 055 186 056 43 057 619 058 769 059 108 060 373 061 419 062 365 063 440 064 750 065 15 066 677 067 211 068 234 069 196 070 349 071 516 072 372 073 720 074 149 075 713 076 891 077 324 078 474 079 296 080 608 081 640 082 571 083 738 084 639 085 360 086 806 087 759 088 820 089 350 090 428 091 294 092 992 093 921 094 322 095 868 096 588 097 772 098 106 099 247 100 453

## 6.2 Test data with normal $n$ and $m$

40 40

002 040 023 034 018 027 002 038 018 030 011 013 033 036 017 040 015 032 003 025 011 015 028 033 009 038 002 028 001 018 023 040 007 027 011 027 026 028 026 034 001 017 016 019 012 034 010 036 016 034 007 030 007 038 004 040 022 037 002 011 018 028 031 036 029 033 002 026 011 022 003 030 004 017 002 016 001 033 013 032

001 879 002 889 003 775 004 166 005 482 006 34 007 141 008 428 009 886 010 577 011 848 012 79 013 970 014 535 015 231 016 726 017 158 018 428 019 348 020 886 021 193 022 607 023 773 024 112 025 158 026 761 027 875 028 990 029 575 030 620 031 875 032 398 033 890 034 222 035 444 036 255 037 779 038 250 039 191 040 772

## 6.3 Reference

http://i.stanford.edu/pub/cstr/reports/cs/tr/76/550/CS-TR-76-550.pdf