

Safe Fruit

Author's Name: 肖策方 诸葛均豪 周承扬

Date:2022-04-10

Chapter 1: Introduction

In daily life, people will meet dilemma in where they don't know how to make the best choice, such as formulating a travel schedule that contains most scenic spots.

This project uses the knowledge learned from graph theory to find out a similar problem that some fruit can not be eaten together, we need to calculate fruit categories that can be eaten together at most.

The input mainly contains two parts, fruit pairs that cannot be eaten together and fruit with its price. Before these are given, the number of fruit pairs and the number of fruit should be given in order. In this problem, we assume that the number of fruit M indicates that the id of fruit should be ranged from 001 to M . So this may not run properly at PAT Top.

In this problem, the graph is an undirected graph, fruit is naturally the node, existing edge meaning two kinds of fruit are not incompatible, so the fruit set corresponding to the largest cluster is not incompatible with each other, than the correspondent fruit can be safely eaten together.

Our team uses the Bron-Kerbosch algorithm to find the largest group in the itemless graph by recursive backtracking.

Chapter 2: Data Structure / Algorithm Specification

Algorithm Specification:

1. Bron-Kerbosch algorithm

We can convert the problem to a graph problem by simply connecting all the fruits together and disconnect fruit pairs that cannot be eaten simultaneously and thus this project is actually a maximum clique problem. A clique is a subgraph that all the vertices in it are connected to each other. And a maximal clique is a clique that you cannot add any vertex to it any more. A known good algorithm to solve maximal clique is Bron-Kerbosch algorithm. We use this to get all the maximal clique and compare them to get the max clique. If there exists more than one max clique, only the one with smallest price sum is returned. Bron-Kerbosch algorithm uses three vertex sets, that is P , R , X . The R set collects all vertices in the maximal clique. The P set collects all the vertices that have the potential to be one of the vertex in maximal clique. The set X collects the vertices that is abandoned during backtracking. $N(v)$ denotes the neighbor set of vertex v . For every vertex in P , it is possible to be a vertex of maximal clique, so try to add it to the result set R and update the potential set P by intersecting with $N(v)$ so that all the vertices in P are connected to each other. Also update set X using intersection. Recursively do Bron-Kerbosch algorithm using these newly derived set. Backtrack by removing the vertex from P and add the vertex into X . When P and X are both empty, then no vertex can be added to the result set R any more. So a maximal clique has been obtained. The Bron-Kerbosch algorithm in pseudo-code is like this.

```
BronKerbosch(Set R, Set P, Set X):
```

```
    if  $P$  and  $X$  are both empty:
```

```
        record  $R$  as a maximal clique
```

```
for each vertex v in P:
    BronKerbosch1( $R \cup \{v\}$ ,  $P \cap N(v)$ ,  $X \cap N(v)$ )
     $P := P \setminus \{v\}$ 
     $X := X \cup \{v\}$ 
```

To find maximum clique, we just need to compare each maximal clique. So when a new maximal clique is obtained, we compare it with the former maximal clique. If the newly got one is larger or is the same size but with smaller price sum, replace the former obtained maximal clique with the newly got one. Else, keep the former obtained one and abandon the newly got one. All these can be done in **if P and X are both empty**. What we need to do is to change **record R as a maximal clique**.

2. Pruning Strategy

If we just use Bron-Kerbosch algorithm, it will be somehow a little bit slow. So we use two pruning strategy to accelerate the program.

First, if the number sum of all the potential vertices and the result vertices is smaller than the known maximum clique, there is no need to calculate this maximal clique as it will never be larger than the known maximum clique.

Second, for every vertex in P, if two vertices are neighbor, there is no need to traverse both, only one vertex is enough. Even if two maximal shares same vertices, the vertices they don't share will guarantee to generate the maximal clique without left anyone out.

3. Sort

We use Sort function in C++ STL. It is realized with a combination of quick sort and insertion sort. If the data quantity is large enough, quick sort will take effects. And when the data quantity is smaller than the threshold, insertion sort takes place. Quick sort is a sorting algorithm that using divide-and-conquer strategy. It randomly choose one num as pivot(to be specific, the midian number of the first and the middle and the last number) and make the smaller to one side and the bigger to another side, repeating.

Data Structure

A graph is constructed to denote the relations of all the fruit. Fruit are denoted as vertex and if two fruit can be eaten simultaneously, there is an edge between them. We use nested vector graph to represent the edge between vertices. It is quite similar to two-dimension array. When there is an edge between two vertices v1, v2, `graph[v1][v2]` will be true and the same with `[v2][v1]`. We initialize the value of vector to true as we assume all the fruit can be eaten together. And when reading in tips about fruit that are not ought to eat simultaneously, we set the edge to false.

Chapter 3: Testing Results

Case 1:

Case1 tests samples provided by project.

input:

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:

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

our result:

pass.

```
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
```

Case 2:

Case2 test custom sample.

input:

```
11 15
001 004
002 006
002 008
005 006
007 009
007 010
007 014
008 009
010 012
010 015
013 014
015 20
```

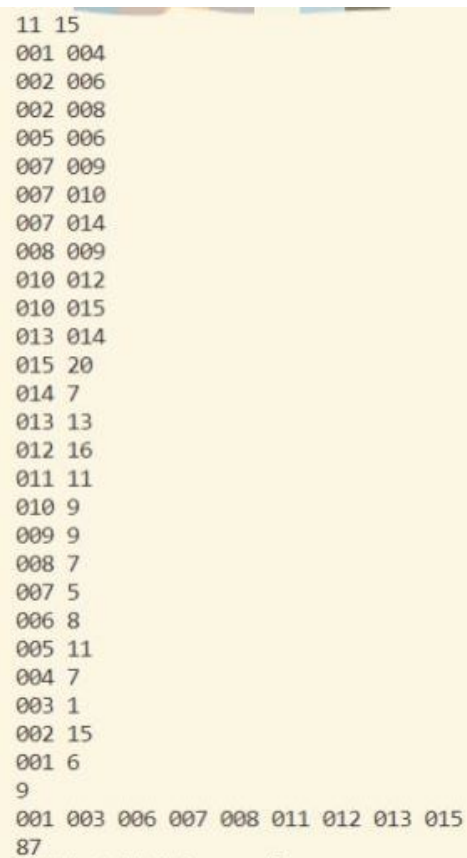
014 7
013 13
012 16
011 11
010 9
009 9
008 7
007 5
006 8
005 11
004 7
003 1
002 15
001 6

output:

9
001 003 006 007 008 011 012 013 015
87

our result:

pass



```
11 15
001 004
002 006
002 008
005 006
007 009
007 010
007 014
008 009
010 012
010 015
013 014
015 20
014 7
013 13
012 16
011 11
010 9
009 9
008 7
007 5
006 8
005 11
004 7
003 1
002 15
001 6
9
001 003 006 007 008 011 012 013 015
87
```

case3:

Case3 tests simplest sample.

input:

1 2
001 002
002 1
001 2

output:

1
002
1

our result:

pass

```
1 2
001 002
002 1
001 2
1
002
1
```

Case4:

Case4 tests self-contradiction sample.

input:

1 2
001 001
002 1
001 2

output:

2
001 002
3

our result:

pass

```
1 2
001 001
002 2
001 1
2
001 002
3
```

Case5:

Case5 shows the wrong sample.

input:

```
1 2
001 002
003 3
002 2
```

output:

```
1
002
2
```

our result:

pass

```
1 2
001 002
003 3
002 2
1
002
2
```

According to the manual calculation, the experimental results are completely correct.

Chapter 4: Analysis and Comments

Time Complexity: There is a lemma that a graph with n vertices has at most $3^{n/3}$ maximal clique. Because every maximal clique will be found by constructing and no redundant subgraph is found, so the worst case is $O(3^{n/3})$. Actually, because we will abandon those maximal cliques that can never be the maximum cliques, this program runs a little bit faster, but still in $O(3^{n/3})$. In fact, a complete proof of this should take very complex steps, but we think this is enough to roughly show the time complexity of this program. For more detailed proof, a paper listed in References may help. The referenced proof is listed in References.

Space Complexity: The space complexity is $O(n^2)$. $P \cup R \cup X$ is subset of original graph, so

it takes $O(n)$. When there is a Bron-Kerbosch algorithm called, P, R and X is created. The recursive call can nest at most n, so the worst case is $O(n + n + n + \dots + n) = O(n \cdot n) = O(n^2)$.

Comments: There are many variants of Bron-Kerbosch algorithms, such as using pivot vertex, which will do better in cases that graph has many non-maximal cliques. We can optimize our program with the optimized Bron-Kerbosch algorithm to get a quicker performance.

References

[1] <https://iq.opengenus.org/bron-kerbosch-algorithm/>

[2] Wikipedia, "Bron-Kerbosch algorithm",

https://en.wikipedia.org/wiki/Bron-Kerbosch_algorithm#Worst-case_analysis

[3] EtsujiTomita, AkiraTanaka, Haruhisa Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments",

<https://www.sciencedirect.com/science/article/pii/S0304397506003586?via%3Dihub>

Author List

肖策方: Implement the program.

诸葛均豪: Discussing the implement of program with 肖策方 and debug and write report Data Structure and Algorithm、 Analysis and Comments.

周承扬: Write report introduction and test.

Declaration

We hereby declare that all the work done in this project titled

" Safe Fruit" is of our independent effort as a group.

Signatures

周承扬

肖策方