Title of Your Project

names

Date: 2023-4-5

Contents

1 Introduction

介绍一下题目，千万不要照抄PTA上面的原表述，会被扣分

2 Data Structure / Algorithm Specification

2.1 Data Structure

2.1.1 `map<char,int>mp`

A mapping data structure that stores **the character symbols of the beads** and **the id we assign**.

2.1.2 `struct Strand`

A structure that stores the information of **each string of beads**. It contains:

- `las`: represents the number of beads that are not the color we need.
- `val`: represents the number of beads that are the color we need, used for sorting.
- `cnt[]`: represents the number of beads of each color that we need in this string of beads.

2.1.3 `int cnt_target[BEAD], cnt_now[BEAD]`

- `cnt_target`: represents the number of beads of each color we need.
- `cnt_now`: represents the number of beads of each color we have now.

2.2 Algorithm Specification

This algorithm solves the bead problem where a girl named Eva has to purchase some beads of certain colors to make bracelets. The objective of the algorithm is to determine the minimum number of beads that Eva needs to purchase to have the exact amount of each color needed to make bracelets for a given number of strings of beads.

2.2.1 Input

The algorithm can read input from **the console** or from **a file**, as chosen by the user. If the input is from the console, the input includes a string that contains all the bead colors needed, and an integer representing the total number of strings of beads, then all the . If the input is from a file, the user should input the file name, and the algorithm reads from the file, which should contain a string with all the bead colors needed and an integer representing the total number of strings of beads, followed by the bead strings themselves, one on each line.

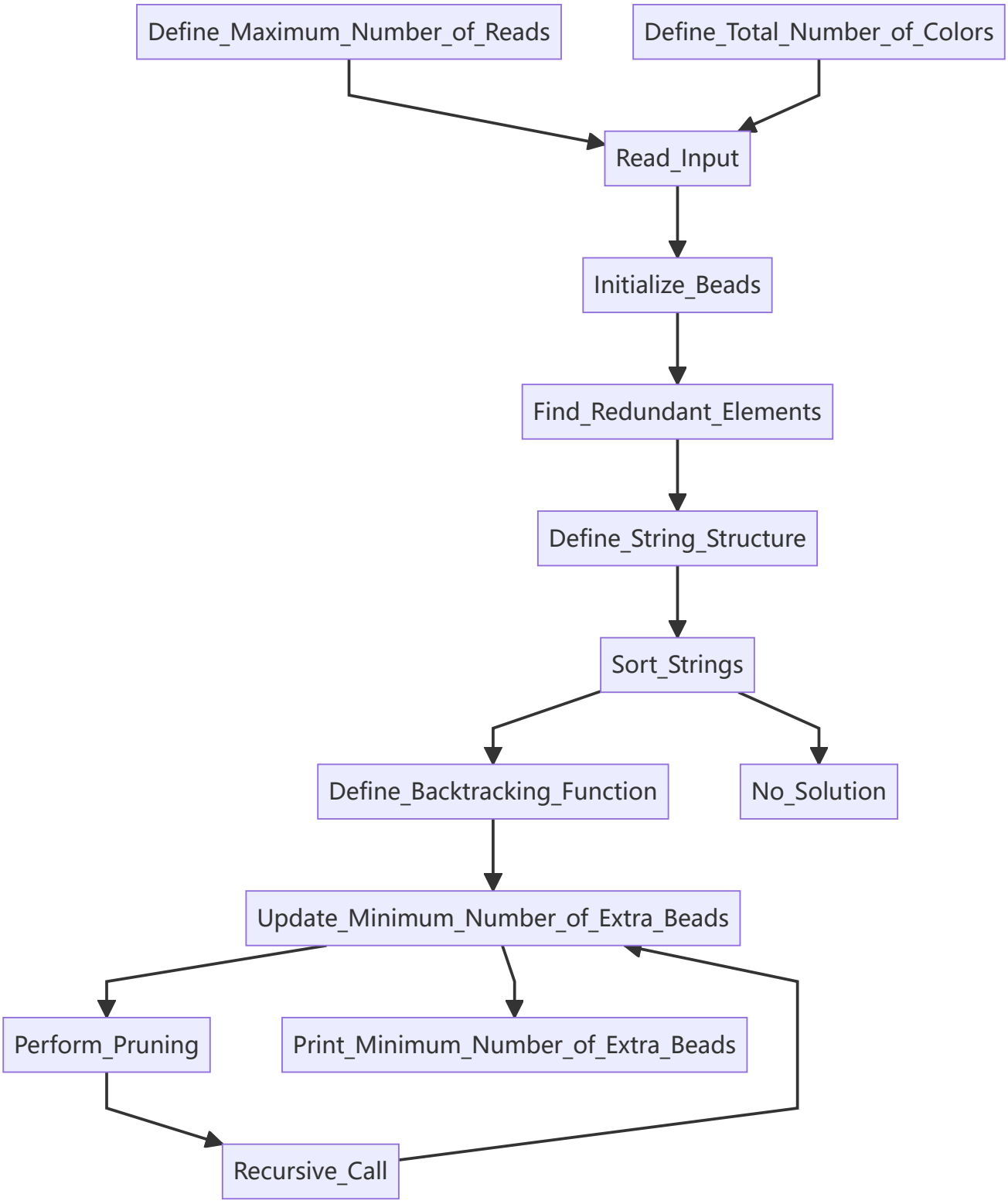
2.2.2 Output

If there has a solution, output **the minimum number of extra beads** that Eva needs to purchase to have the exact amount of each color needed to make bracelets. Else output the number of beads missing from all the strings.

2.2.3 Procedure

1. Define the maximum number of reads and how many colors there are in total for the beads.
2. Read the input, either from the console or from a file.
3. Define a **map** to store the character symbols of the beads and the id assigned to each color.
4. **Initialize** all the beads read in and find out how many redundant elements each string has. Define a structure to **store the information of each string of beads**, including the number of beads that are not the color we need, the number of beads that are the color we need, and the number of beads of each color that we need in this string of beads.
5. **Sort the strings of beads** by their values, i.e., the number of beads that are the color we need, in descending order, then by the number of beads that are not the color we need, in ascending order.
6. Define **a backtracking function** to determine the minimum number of extra beads Eva needs to purchase. The function takes two arguments, **the current index** of the string of beads, and **the number of extra beads purchased so far**. The function will first check if Eva has already purchased enough beads of each color to make the bracelets. If so, it will **update the minimum number of extra beads** needed to the current number of extra beads purchased. If not, it will **recursively call itself** to check if adding the next string of beads will reduce the number of extra beads needed. The function will also perform pruning by checking if the current number of extra beads purchased plus the number of extra beads that would be needed if using the next string of beads is greater than or equal to the current minimum number of extra beads needed. If so, it will **skip that branch of the recursion**.
7. Print the minimum number of extra beads Eva needs to purchase.

This is the **flowchart** of program operation:


Define_Maximum_Number_of_Reads
Define_Total_Number_of_Colors
Read_Input
Initialize_Beads
Find_Redundant_Elements
Define_String_Structure
Sort_Strings
Define_Backtracking_Function
No_Solution
Update_Minimum_Number_of_Extra_Beads
Perform_Pruning
Print_Minimum_Number_of_Extra_Beads
Recursive_Call

3 Innovation

3.1 Preprocessing

In this program, we performed preprocessing on the storage method of beads. Instead of using numbers 0-61 to represent 62 different types of beads, we only **assigned an index to the needed beads** and stored them in a map. When reading in the strands sold in the store, we calculated the number of needed and unneeded beads in each strand, as well as the number of each type of needed bead. We then **sorted the strands** based on the number of effective and ineffective beads, making it easier for subsequent calculations. The time complexity of reading in the preprocessed data is $O(\sum len * logm)$.

3.2 Pre-Judgment

If none of the strands sold in the store can meet our requirements, the output of this problem is "No". Calculating this scenario using the backtrack algorithm has a high time complexity of $O(m * 2^n)$. Therefore, we optimized this scenario by utilizing the sorting performed in the Preprocessing section to obtain the order of effective information for all strands in the store. We then purchased the strands in this order and checked if they meet the requirements. If they do, we proceed with the backtrack program as usual. If not, we output the result directly, thereby reducing the time complexity of the "**No Solution**" scenario to $O(n * m)$.

3.3 Pruning

In the backtrack algorithm, we perform pruning on each step. If the calculated `las` at a step exceeds the current minimum `las`, we exit the step and backtrack to the previous step to continue the search.

When the last strand is found and still cannot meet the requirements, it can be determined that the right subtree of the current search tree cannot meet the conditions and can be pruned together.

4 Testing Results

4.1 Input Test

The file directory structure is as follows

```
.
├── main.cpp
└── output
    ├── main.exe
    └── test.txt
```

After running the exe file, the output is as follows:

- Read from file

```
\code\output> main.exe
Please choose whether to read files or input from the console
1. Read from file
2. Input from console
1
Please enter the read file name
test.txt
Yes 3
```

The content of `test.txt` is as follows:

```
RYg5
8
gY5Ybf
8R5
12346789
gRg8h
5Y37
pRgYgbR52
8Y
8g
```

- Read from console

```
\code\output>main.exe
Please choose whether to read files or input from the console
1. Read from file
2. Input from console
2
YrRR8RRrY
3
ppRGrrYB225
8ppGrrB25
Zd6KrY
No 3
```

4.2 Time Test

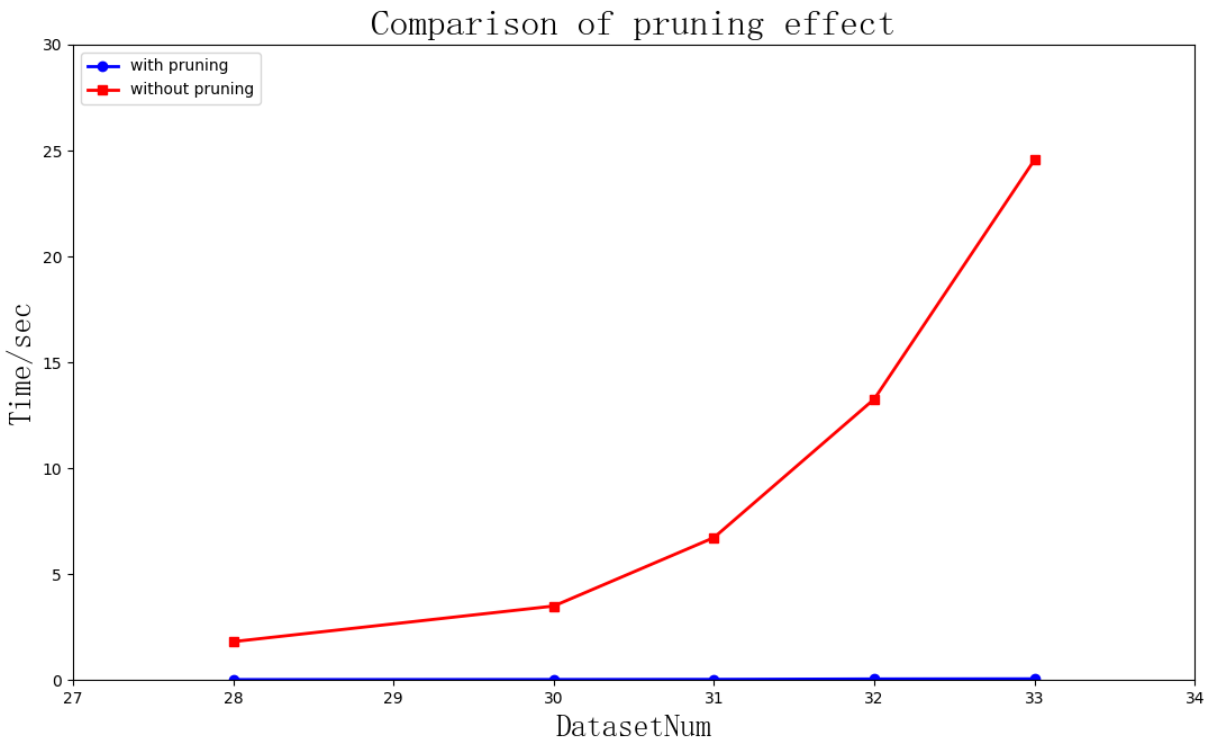
Time with pruning on big data:

```
Please choose whether to read files or input from the console
1. Read from file
2. Input from console
1
Please enter the read file name
test2.txt
Yes 8
ElapsedTime: 0.487
```

Time without pruning on big data:

```
Please choose whether to read files or input from the console
1. Read from file
2. Input from console
1
Please enter the read file name
test2.txt
Yes 8
ElapsedTime: 5.368
```

The following is a comparison of program runtime with and without pruning within a certain data range



4.3 PTA Test

Our program successfully executes on the PTA program platform in only 8ms, this is far less than the time limit required by the question itself.

提交时间	状态 ⓘ	分数	题目	编译器	内存	用时	用户
2023/04/11 10:48:57	答案正确	35	1004	C++ (g++)	456 KB	8 ms	

测试点	结果	分数	耗时	内存
0	答案正确	17	3 ms	456 KB
1	答案正确	5	3 ms	312 KB
2	答案正确	5	4 ms	324 KB
3	答案正确	3	8 ms	312 KB
4	答案正确	1	3 ms	324 KB
5	答案正确	1	4 ms	456 KB
6	答案正确	1	4 ms	324 KB
7	答案正确	1	4 ms	444 KB
8	答案正确	1	3 ms	324 KB

5 Analysis and Comments

5.1 Time complexity

5.1.1 Pre-Operation

The time complexity of the operation in the map is $O(logm)$, and the preprocessing involves processing the length of a strand, resulting in a time complexity of $O(\sum len * logm)$. The time complexity for determining the "No Solution" case is $O(n * m)$.

5.1.2 backtrack

The time complexity of the backtracking algorithm is $O(m * 2^n)$. However, the actual time complexity is much lower than the worst-case scenario due to the inclusion of pruning in the program.

5.1.3 Summary

From the above analysis, it can be seen that the overall time complexity is $O(m * 2^n)$.

5.2 Space complexity

The space complexity of the algorithm is $O(m)$ to store the count of each bead color needed and $O(n * m)$ to store the information of each string of beads.

6 Appendix: Source Code

```
1 课程要求，不允许公开代码
```

7 References

[1] Advanced Data Structure Slides "Backtracking". https://tcmedia.zju.edu.cn/download/processed/d1e59f3d22b9d9aab55bbbb13dd2caa27a139b87.pdf?timestamp=1680699600&token=4a488ec205693b232926adafbec3994e&name=ADS06Backtracking_Stu%282%29.pdf

8 Author List

- Author1: Depth first search function and report writing
- Author2: Program optimization, pretreatment and report writing
- Author3: Main function and PPT production

9 Declaration

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

10 Signatures

签名