Lab5 Report

作者: 罗文杰
专业: 计算机科学与技术
学号: 3210102456

1 Introduction

Assume that N pages of the materials are double-sided printed, each side containing one of N chapters in the textbook. You wonder if there is a way to flip some of them so that you can cover all the N chapters at once.

2 Algorithm Explanation

Noting that the maximum value of a book page is 15, it is easy to think that a memory has 16 bits, so a memory can be used to represent the two chapters of a page. When a page has k chapters, the value of k bit can be set to 1 on a memory. At this point there is a spare bit that can be used to record whether a page has been selected or not.

In this way, we can represent the current state of a page of books with a memory value that defines the contents of such a memory as a **value**. For example, if a page of unpicked books has chapters 1 and 4, then it has a value of 9 (0000 0000 0000 1001).

In this way, we can start selecting from chapter one and use the AND operation to retrieve each page of the book, and when we have successfully found the right chapter on a page, we can set that page the selected status and then continue to select the next chapter. This recurses until all chapters have been selected.

The codes for recursion are shown below.

```
; R0 is the chapter we are looking for, R1 is a flag of right,
; R2 is the pointer, R3 is a counter for inserting the right chapter,
; R4 is the target, which means the last chapter,
; PAGE is the address of the frist page
; MASK is 0x8000 used to mark the picked page
FLIP    ......          ; PUSH R2, R7 onto the stack
        AND R7, R0, R4
        BRnp    OVER    ; reach the target
        LDR R7, R2, #0
        BRn     FNEXT   ; picked page
        BRz     FOUT    ; no page
        AND R7, R7, R0
        BRz     FNEXT   ; the page haven't the chapter
        LDR R7, R2, #0
```

```
        ......          ; PUSH the value of current page
        LD  R5, MASK
        ADD R7, R5, R7  ; there is an unpicked and suitable page, mark it
        STR R7, R2, #0
        ......          ; PUSH R0, R2, R3 before the next recursion
        ADD R0, R0, R0  ; find the next chapter
        ADD R3, R3, #1  ; add the counter
        LD  R2, PAGE    ; reset the pointer
        ADD R2, R2, #1
        JSR     FLIP    ; recursion, search for the next chapter
        ......          ; POP the R0, R2, R3 after the recursion
        STR R7, R2, #0
        ADD R1, R1, #0  ; judge the way of picking chapter
        BRz     FNEXT
        ADD R2, R2, #1  ; this is a right way,
        STR R3, R2, #0  ; so record the picked chapter to the page
        BRnzp   FOUT
FNEXT   ADD R2, R2, #2
        JSR     FLIP
        BRnzp   FOUT
OVER    ADD R1, R1, #1  ; the all chapter have been picked, set the flag
FOUT    ......          ; POP R2, R7 from the stack
        RET
```

3 Algorithm Analysis

3.1 Time Complexity

Assume that the time complexity of each call of FLIP is O(1), It is easy to see that, in the best case where the i-th page has the i chapter and any of other chapters, the time complexity of the whole algorithm is O(n) when we need n chapter.

However, the worst-case time complexity of this algorithm is very difficult to calculate. Consider a situation where if every page has chapter 1, then each looking for chapter1 will go to the next level, and if chapter 2 is exactly found on every page, then each lookup goes to the next level too, and in the third level, each page is traversed even though it is no longer possible to go to the next level(every page only has two chapters). In this case, the time complexity is O(n^3).If there are exactly n/4 pages with chapter 1, n/4 pages with chapter 2, n/4 pages with chapter 4, and n/4 pages with chapter 3, then the time complexity is O(n^5).

To make matters worse, all of the above are unsolvable, and when the condition that there must be a solution is added, things become much more complicated. What can be found is that for a perfect lookup (no wrong choices), the time complexity will be O(n^2): in order to find the chapter 1, the program will traverse all the pages, a total of n times, and when the first one is found, the program will continue traversing until it finds the chapter 2, which will also be n times, and so on, taking a total of n^2 times when the chapter n is finally found. On top of this, the remaining n nodes are added (a total of 2n chapters) and the time complexity

under a certain example can be derived by analysing the graph.

In the above analysis, we find that the average value of the time complexity is $O(n^2)$. This is a very efficient algorithm.

3.2 Improvement

In order for the performance of the algorithm to be optimized, we need to bring forward the correct selection method, which means choosing a lower chapter earlier and a higher chapter later. One available method is: when a page of data is read in, compare the largest chapter of it with the smallest chapter of the last page read, and if its largest chapter is smaller, then place the page currently read before the last page read, and if not, then it is placed after the last page read normally.

3.3 Pruning Algorithm

If we recurse the pages, with each page having two branches for two chapters, then we will build a binary tree with a max time complexity of $O(n^2)$. In order to optimise the algorithm, we need to prune. The method is: when there is already a chapter in the currently searched page that has been selected before, then this chapter will not be selected.

In pruning optimization, the final time complexity is uncertain and depends on the sample state. But one conclusion that can be drawn is that the earlier the pruning is performed, the smaller the final time complexity. The reason for this is that, although the number of child are fixed at each node, the nodes close to the root have more descendant on it.

So, in order to optimise the algorithm more, we need to make the pruning as early as possible. This is done by comparing each time a page of data is read in, we should compare its chapter with the chapters that is already read in, if its chapter has already appeared, placing it after the corresponding page who has the same chapter. In this way, during retrieval, many branches are cut off earlier because of the order and the performance of the algorithm is optimised.