HW 10. Finding Equal Partitions

EE3980 Algorithms

104061212 馮立俞

May 20, 2018

Introduction

In this assignment, we are given a list of integers, A. The Object is to split A into to groups, S_0 and S_1, such that

$$\begin{aligned} &S_0 \cap S_1 = \emptyset \\ &S_0 \cup S_1 = A \\ &\sum_{a_i \in S_0} a_i = \sum_{a_j \in S_1} a_j. \end{aligned}$$

We'll show result if such solution exists. And of course, better efficiency is desired.

Approach

Since S_0 and S_1 are two disjoint subsets of A, and their union is equal to A, the members in A can be either in S_0 or S_1. Thus,there are 2^N possible pairs of $\{S_0, S_1\}$, where N is the size of A. We'll have to try out all possible pairs to find out all possible answers if we have no other idea except brute force approach.

However, we can use backtracking method to reduce unnecessary trials to speed up the process.

Backtracking

Below is a big picture of a recursion program.

```
Algorithm Resursion(){
    if (reach termination condition)
        show result
    else{
        do something to reduce the problem
        Recursion();
    }
}
```

We can, however, sometimes know whether a solution is legal way before it reaches the termination condition. Differently put, we can set stricter termination condition to avoid further unnecessary function calls. Here in this task, either the current Sum is larger than the target sum or completing the rest of recursion gives less sum than target sum can tell us to terminate the recursion before it has traverse all the numbers in A.

In this homework, we can apply the backtracking $SumOfSubsets$ Algorithm (7.1.3) in class, with the target sum equaling half of the sum of A.

```
Algorithm SumOfSub(s, k, r)
{
    x[k] := 1 ; // try to include w[i]
    if (s + w[k] = m) then write (x[1 : k ]) ; // one solution found
    else if (s + w[k] + w[k + 1] <= m) then
        SumOfSub(s + w[k], k + 1, r - w[k]) ;
    if ( (s + r - w[k] >= m) ) then { // x[il] = 0 case
          x[k ] := 0 ;
       SumOfSub(s, k + 1, r - w[k]) ;
    }
}
```

The algorithm traverse a binary tree whose branches at level i are decisions on whether to include i^{th} in A element into S_0. Backtracking is applied.

More tricks

Half the task

However, if we use the algorithm, and a solution S_0 is found. Then $S_1 = A \setminus S_0$ is simultaneously found.

Because the solutions always appear in pairs, we can reduce the problem to "Finding S_0 containing a_0 with sum equaling $\frac{1}{2}\sum_{a_i \in A} - a_0$". Because if we can find a set containing a_0, then the set without a_0 would appear simultaneously. This could save us half of the effort on the answers which are just swapping the elements in S_0 and S_1.

Perform sort ahead

Another trick is to sort the elements in A in decreasing order. Doing so could help us terminate unwanted solutions earlier. Take the example from lecture notes.

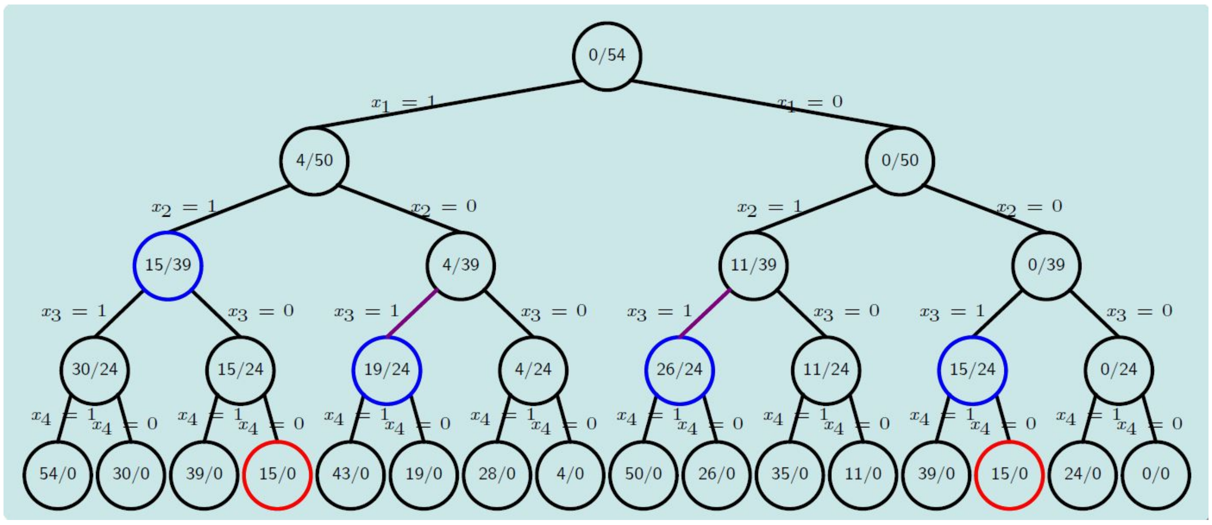

Figure 1: Execution of SumOfSub, A = { 4,11,15,24 } , target = 15

In the above figure, the elements in A are not sorted. The blue circles denote early termination. If blue circles appear on higher level, more computation can be saved.

If we sort A to $\{24, 15, 11, 4\}$. The execution could look like the following figure

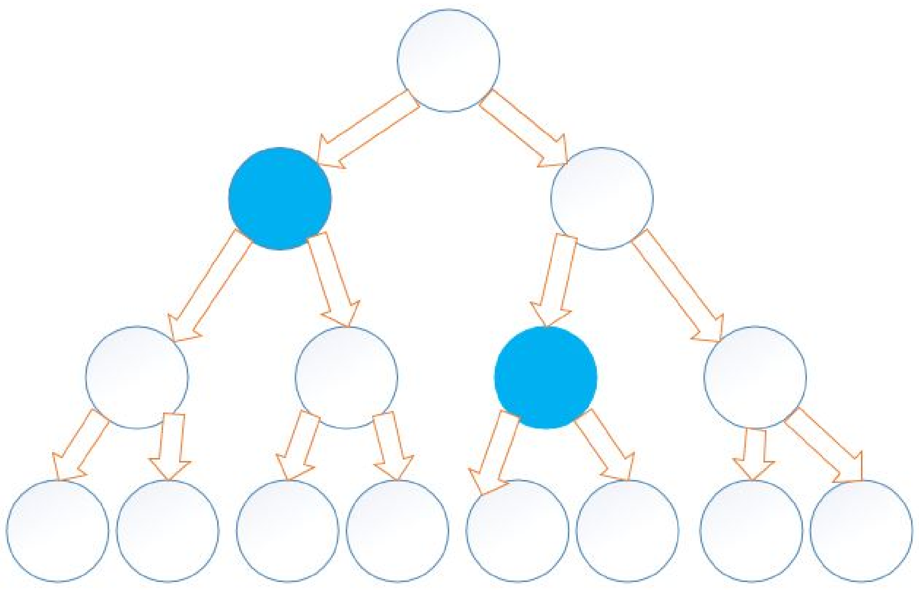

Figure 2: Execution of SumOfSub with prior sort , A = { 24,15,11,4 } , target = 15

We can see termination occur at higher level, when we have accumulated 24 or 15. Plus, sorting has $nlog(n)$ complexity, so the sorting operation wouldn't be a great overhead compared to 2^N trials.

Using Iteration

Theoretically, all recursive programs can be realized using iteration. This would require more programming effort, yet the overhead of function calling can be drastically alleviated.

Results and Analysis

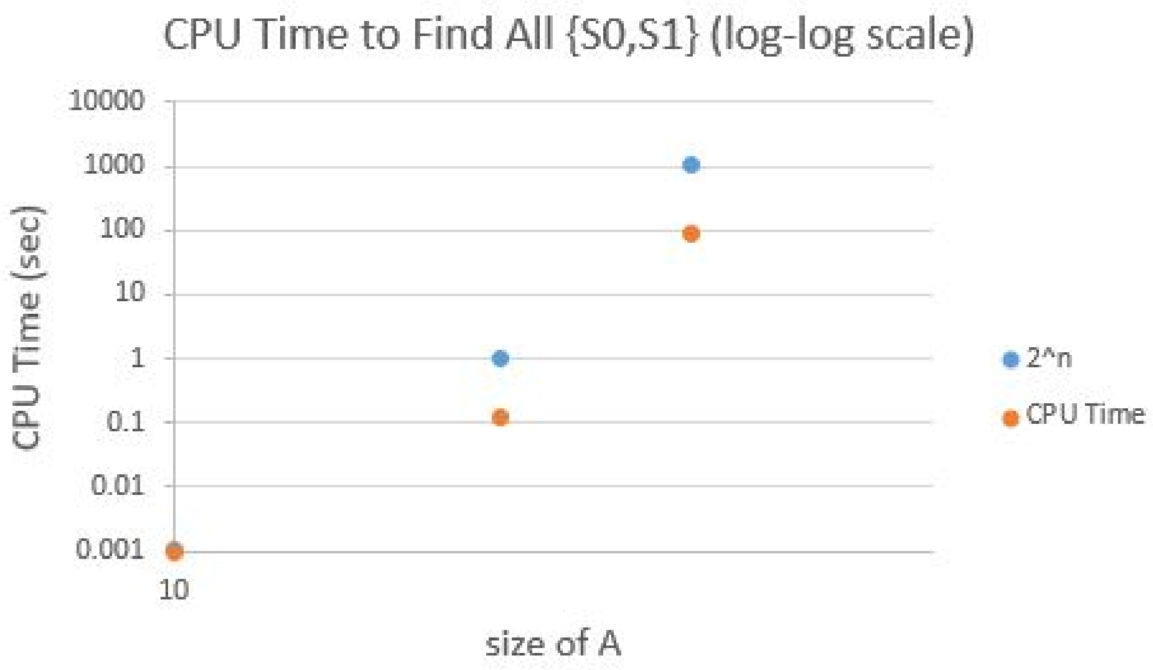

Figure 3: CPU Time needed to find all solutions v.s task size, $lg - lg$ scale

The execution time complexity is comparable to 2^N, yet backtracking has saved us some time.

During execution on work station, the actual CPU time deviates quite a lot. Thus it's hard to verify if sorting could really help boost efficiency.

Observations and Conclusion

1 Actually, we may not be able to print all answers correctly for input over 32 integers. Overflow could occur when we use 32 bit integer to record solution number.

2 The change in efficiency due to sorting is not effective as expected.

3 From the tree diagram, we can see that the branches are independently separated. It may be possible to assign each task to individual processors to utilize the power of

parallel computing.