Assignment3

卫卓虹
2018E8013261019

2018/11/22

1 Job Selection

1.1 Solution

We first sort the jobs to a non-increasing order according to f_i value.Then we put the jobs to supercomputer successively according to the sequence.

```
function MIN PC TIME($f$)
    $F \leftarrow$ nonIncreasingSort($f$)
    return $F$
end function
```

1.2 Correctness Validation

The problem is to decide the order of every two adjointed jobs.Without losing generality,we define p_1 and f_1 for job1,p_2 and f_2 for job2.Because the PC's resource is absolutely sufficient,we should start f_i at once when p_i is finished.Also,we should start another p_j at once when p_i finished. Then the overall time to finish job1 and job2 holds 4 possible cases:

(a)if p_1 is in front of p_2 and $f_1 > p_2$:$p_1 + f_1$

(b)if p_1 is in front of p_2 and $f_1 < p_2$:$p_1 + f_1 + f_2$

(c)if p_2 is in front of p_1 and $f_1 > p_2$:$p_1 + f_1 + p_2$

(d)if p_2 is in front of p_1 and $f_1 < p_2$:$p_1 + f_1 + f_1$

Without losing generality,we define $f_1 > f_2$,then there is only two possible cases:(a) and (d).Because time(d)-time(a) =$p_2 > 0$,we should chose case(a).Thus we should have a schedule according to a non-increasing order of job list.

1.3 Complexity Analysis

With an ideal sorting algorithm, the complexity is O(nlogn).Thus the total time T(n) = nlogn+cn.

2 Graph Judge

2.1 Solution

We first sort the number list into a non-increasing list S.For each step, we substract d_1 from the sequence and substract 1 from the left first number of d_1 elements.Sort the left elements again.Now the subproblem is that given one non-increasing sequence S, we should find whether it satisfies the rule abouve.Noticed that if any $d_j < 0$ situation occurs, the output judgement should be failure.If all the elements in one subsequnce are 0,we can return success.

```
function GRAPH JUDGE($S, N$)
    $S \leftarrow$ nonIncreasingSort($S$)
    if $S[1] == 0$ then
        return $true$
    end if
    if $S[N] < 0$ then
        return $false$
    end if
    $k \leftarrow d_1$
    $N \leftarrow N - 1$
    $S \leftarrow S - S[1]$
    for $j = 1$ to $k$ do
        $S[j] \leftarrow S[j] - 1$
    end for
    return GRAPH JUDGE($S, N$)
end function
```

2.2 Correctness Validation

True Output: If there is one step we get the sorted sequence S, and the biggest element equals to 0 and smallest element bigger than 0, thus all the elements are 0.From the first operation to this step,we connect every substracted element with those k elements,there is no points left anymore in the end, and the rule is satisfied.

False Output:If there is at least one point's value$<$ 0,at the subproblem before this step,the element substracted should have the degree larger than the length of points sequence,which could not find enough pair points to connect,which is not correct.

2.3 Complexity Analysis

For every recursion step, the sorting operation holds O(nlogn) complexity.Thus,considered the iteration,at the worst case, we will take O(n) recursing operations.$T(n) = O(n(nlogn + cn)) < O(n^2)$.

3 Check Subsequence

3.1 Solution

The solution is simple.We can check the two strings iteratively.We first set two pointers pointing to the first characters of strings respectively. Because the relative positions of the remaining characters can't be disturbed, we go through two strings to calculate the number of letters of s occur in t. At last we compare that to the length of s.If equals,we return true,else return false.

if we define the number of same letters as n(i,j),then we have

n(i,j) = n(i-1,j-1)+1 (if s[i] is the same as t[j])or n(i,j) = n(i,j-1)(if s[i] is not the same as t[j])

```
function CHECK SUB($s, t$)
    $s\_len \leftarrow strlen(s)$
    $t\_len \leftarrow strlen(t)$
    $i \leftarrow 0$
    $j \leftarrow 0$
    while $i < s\_len$ and $j < t\_len$ do
        if $s[i] == t[j]$ then
            $i \leftarrow i + 1$
            $j \leftarrow j + 1$
        else
            $j \leftarrow j + 1$
        end if
    end while
    return $s\_len == i$
end function
```

3.2 Correctness Validation

It's similar to DP problem.If the letter of s occurs in t orderly,then s is the subsequence of t.

3.3 Complexity Analysis

For every recursion step, there is only constant level complexity.Thus T(n) = O(n)

4 Maximum Product Cutting

4.1 Solution

If we see some examples of this problems, we can easily observe the pattern:The maximum product can be obtained be repeatedly cutting parts of size 3. While size is greater than 4, keeping the last part as size of 2 or 3 or 4. For example, n = 10, the maximum product is obtained by 3, 3, 4. For n = 11, the maximum product is obtained by 3, 3, 3, 2.

```
function MAXIMUM PRODUCT CUTTING($n$)

    if $n == 2||n == 3$ then return $n$
    end if
    $res \leftarrow 1$
    while $n > 4$ do
        $n \leftarrow n - 3$
        $res \leftarrow res * 3$
    end while
    return $n * res$
end function
```

4.2 Correctness Validation

When the length of rope is bigger than 4,if we cut the rope into m parts(each part bigger than 1),the product of the length of each part is bigger than the length of the rope.When the length of rope is smaller than 4,it's better not to cut.Therefore for any $n > 4$, it will be finally cut into parts whose length are 2 or 3. That is to say,$n = 2^x + 3^y$.The bigger the y,the bigger the product.

4.3 Complexity Analysis

For every recursion step, n substracts 3. So the time complexity is O(n/3)=O(n)