Lab03-Greedy Strategy

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2021.

* If there is any problem, please contact TA Haolin Zhou.
* Name: Zirui Liu   Student ID:519021910343   Email: L.prime@sjtu.edu.cn

1. *Interval Scheduling.* Interval Scheduling is a classic problem solved by **greedy algorithm**: given n jobs and the j-th job starts at s_j and finishes at f_j. Two jobs are compatible if they do not overlap. The goal is to find maximum subset of mutually compatible jobs. Tim wants to solve it by sort the jobs in descending order of s_j. Is this attempt correct? Prove the correctness of such idea, or else provide a counter-example.

   Solution. This is not correct. Assume we have a mission which starts at time 0 and finishes at time 100, and we only have 100 time units available. In another way, we have 10 missions lasting 5 time units, starting from time 2, each with time interval of 2 time units. In this case, Tim's algorithm will never be correct. $\square$

2. *Done deal.* In a basketball league, teams need to complete player trades through matching contracts. Every player is offered a contract. For the sake of simplicity, we assume that the unit is M, and the size of all contracts are integers. The process of contract matching refers to the equation: $\sum_{i\in A} a_i = \sum_{j\in B} b_j$, where a_i refers to the contract value of player i in team A involved in the trade and b_j refers to the value of player j in team B.

   Assume that you are a manager of a basketball team and you want to get **one** star player from another team through trade. The contract of the star player is $n (n \in \mathbb{N}^+)$. The goal is to complete the trade with as few players as possible.

   (a) Describe a **greedy** algorithm to get the deal done with the least players in your team. Assume that there are only 4 types of contracts in your team: $25M$, $10M$, $5M$, $1M$, and there is no limit to the number of players. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available contract sizes are powers of c, i.e., the values are $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

   (c) Give a set of contract sizes for which the greedy algorithm does not yield an optimal solution. Your set should include a $1M$ so that there is a solution for every value of n.

   Solution. (a):
   We assume n be the total price we want to achieve. We also let $c_{25} = \left\lfloor \frac{n}{25} \right\rfloor$, This is the largest number of $25M$ we can use in this trading. Then we set $n_{25} = n - 25c_{25}$. This is the price remaining after using c_{25} players. Then we set $c_{10} = \left\lfloor \frac{n_{25}}{10} \right\rfloor$ (largest number of $10M$ that can be used) Then we set $n_{10} = n_{25} - 10c_{10}$ (price remains) Then we set $c_5 = \left\lfloor \frac{n_{10}}{5} \right\rfloor$ Then we set $c_1 = n_1 = n_{10} - 5c_5$. Solution uses c_{25} $25M$, c_{10} $10M$, c_5 $5M$ and c_1 $1M$. Proof of optimality: Assume my greedy algorithm c is not optimal. Let p be an alternative optimal solution using p_{25} $25M$, p_{10} $10M$, p_5 $5M$ and p_1 $1M$. First note that if $p_1 \geq 5$ we can replace every 5 $1M$ with one $5M$, reducing the number of players we let out, so it is safe to assume to $p_1 < 5$. The same reason, we can assume $p_{10} < 3$ and $p_5 < 2$. Now suppose that $10p_{10} + 5p_5 + p_1 \quad 25$. The only way that this can happen is if $p_{10} = 2$ and $p_5 = 1$. In this case we can replace the two $10M$ and one $5M$ with one $25M$, reducing the number of players we used, contradicting optimality of p. So contradiction, another better algorithm is impossible. This means that

$10p_{10} + 5p_5 + p_1 < 25$. Since n $= 25p_{25} + 10p_{10} + 5p_5 + p_1$, then it is easy to see that this optimized p algorithm is exactly the same as our algorithm. So our algorithm's optimality is approved.
(b):
There is a unique way to write n in base c, it's form is like : $n = \sum_i a_i * c^i$ with $\forall i, 0_i < c$. Then we consider the greedy algorithm. We assume we chooses a_i players with prices of c^i. If $a_i \geq c$ then we must replace some c^i with c^{i+1}, contradiction for greedy algorithm. So we must have $a_i < c$, so we assign p_i to each a_i Now we assume o_i to represent the number of players with price of c^i used in an optimal solution for filling up n prices. So $\forall i$, $o_i < c$. If not true, then we could always replace the o_i players of price c^i with one player of price c^{i+1} and other players with price of c^i, reducing the number of players being used, contradicting the optimality of the algorithm. This, for all i, we have $o_i < c$ which means for all i, $o_i = a_i$. So in conclusion, the optimality of the algorithm is proofed.
(c):
For example, we choose $1M$, $5M$, $7M$ to make 10. In this case, the greedy algorithm will not be true since it will choose one $7M$ and three $1M$ instead of the better choose of two $5M$.

□

3. *Set Cover.* **Set Cover** is a typical kind of problems that can be solved by greedy strategy. One version is that: Given n points on a straight line, denoted as $\{x_i\}_{i=1}^n$, and we intend to use minimum number of closed intervals with fixed length k to cover these n points.

   (a) Please design an algorithm based on **greedy** strategy to solve the above problem, in the form of *pseudo code*. Then please analyze its *worst-case* complexity.

   (b) Please prove the correctness of your algorithm.

   (c) Please complete the provided source code by C/C++ (The source code *Code-SetCover.cpp* is attached on the course webpage), and please write down the output result by testing the following inputs:

      i. the number of points $n = 7$;

      ii. the coordinates of points $x = \{1, 2, 3, 4, 5, 6, -2\}$;

      iii. the length of intervals $k = 3$.

      Remark: Screenshots of running results are also acceptable

Solution. (a):

First we sort $\{x_i\}_{i=1}^n$, and we get $x_1, x_2, ..., x_n$ in nondecreasing order. The algorithm can be described as following:

Algorithm 1: Greedy

```
Input: An array $x[0, \cdots, n-1]$, two integers $k$,$n$
Output: one integer $num$, meaning the number of pieces we need to cover
sort(x,x+n);
int start=x[0];
int end=start+k;
int index=0,num=0;
while $index<=n-1$ do
    while $x[index]<=end$ do
        index++;
    num++;
    start=x[index];
    end=start+k;
return num;
```

The time complexity for this algorithm is $O(n \log n + n) = O(n \log n)$, in which case $O(n \log n)$ is the major time needed for this algorithm, that is the quicksort process. The $O(n)$ complexity is for the linear scan for all the points. In the worse case, our written quicksort can get to time complexity of $O(n^2)$, that is also the whole time complexity for the greedy algorithm.

(b):

Assume this algorithm is S, for the first cover we have $[x_0, x_0 + k]$, Suppose that there exists an optimal solution $S*$, $S*$ has $[y_0, y_0 + k]$ to cover the pieces. Since y_0 can'y be placed on the right side of x_0, y_0 also can't be equal to x_0, so y_0 must be on the left side of x_0, but obviously our original solution of x_0 is a better solution, since it can cover more places on the right side where there may be some points can be covered by $x_0 + k$ but can not be covered by $y_0 + k$. and the remaining sub-problem after removing some points is the same. So inclusion, the original solution is the optimized solution.

(c):

The result for the test case is 3. For original code, please see the appendix.

□

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.

1 First appendix

Input C++ source1:

```
#include <iostream>
#include<algorithm>
using namespace std;

void quickSort(int s[], int l, int r)
{
    if (l< r)
    {
        int i = l, j = r, x = s[l];
        while (i < j)
        {
            while(i < j && s[j]>= x)
                j--;
            if(i < j)
                s[i++] = s[j];
            while(i < j && s[i]< x)
                i++;
            if(i < j)
                s[j--] = s[i];
        }
        s[i] = x;
        quickSort(s, l, i - 1);
        quickSort(s, i + 1, r);
    }
}

int Greedy(int x[], int k, int n)
{
    /*
    Please write your Greedy function here.
    If you want to use sorting, please use the quickSort function above.
    */
    //quickSort(x,k,n);
    // for(int i=0;i<n;++i){
    //      cout<<x[i]<<'\t';
    // }
    // cout<<endl;

    // the quicksort above may not be correct
    sort(x,x+n);
    int start=x[0];
    int end=start+k;
    int index=0,num=0;
    while(index<=n-1){
        while(x[index]<=end){
            index++;
        }
```

```
            num++;
            start=x[index];
            end=start+k;
        }
        return num;

    }

    int main()
    {
        //x is the point set P with n=7 nodes in total, and the length of interv
        int x[7]={1,2,3,4,5,6,-2};
        int k=3;
        int n=sizeof(x) / sizeof(x[0]);
        int num_interval=Greedy(x,k,n);

        // for(int i=0;i<n;++i){
        //      cout<<x[i]<<'\t';
        // }
        // cout<<endl;

        cout << num_interval << endl;
        return 0;
    }
```