Lab03-Greedy Strategy

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

∗ If there is any problem, please contact TA Haolin Zhou.
∗ Name:Yanjie Ze Student ID:519021910706 Email: zeyanjie@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. The attempt is correct.

Let $i_1, i_2, ..., i_k$ denote set of jobs selected by greedy, in an inverted order, which is to say i_1 is the last job and i_k is the first job.

Similarly, let $j_1, j_2, ..., j_m$ denote set of jobs in an optimal solution in an inverted order, with $i_1 = j_1, i_2 = j_2, ..., i_r = j_r$ for the largest possible value of r.

Case 1:

If j_{r+1}'s starting time is earlier than i_{r+1},as showed in Fig. 1, then the optimal choice should be the greedy one rather than this one, since the greedy choice will leave more time for the jobs before, which is more likely to have the most jobs.

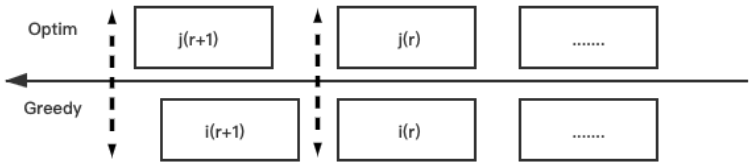

Figure 1: Case 1

Case 2:

If j_{r+1}'s starting time is later than i_{r+1},as showed in Fig. 2, then the greedy choice should be the same as the optimal one rather than this one. It's because the algorithm selects the later starting time.

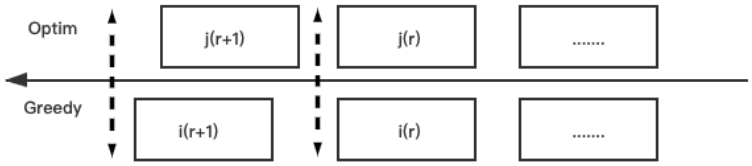

Figure 2: Case 2

□

In conclusion, the optim is the same as the greedy. Therefore, the algorithm is optimal.

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) The algorithm is showed in Alg. 1. This is actually the same policy used in **Coin Changing** problem. In each turn we select the largest contract that is smaller than n and finally we will get the deal.

Algorithm 1: Greedy Algorithm

```
Sort contracts denominations by value: c_1 = 1, c_2 = 5, c_3 = 10, c_4 = 25.;
S ← ∅;
x ← n;
while x ≠ 0 do
    let k be largest integer such that c_k < x;
    x ← x − c_k;
    S ← S ∪ {k};
return S;
```

Theorem. Greedy algorithm is optimal for Done Deal Problem.

Proof.

First we introduce several properties in this problem.

- **Property 1.** Number of $c_1 \leq 4$.
  Proof. Replace 5 c_1 with 1 c_2.
- **Property 2.** Number of $c_2 \leq 1$.
  Proof. Replace 2 c_2 with 1 c_3.
- **Property 3.** Number of $c_3 \leq 2$.
  Proof. Replace 3 c_3 with 1 c_4 and 1 c_2.
- **Property 4.** Number of c_2 + Number of $c_3 \leq 2$
  Proof. From **Property 2 & 3**, there is only one case not proved, which is Number of $c_2 = 1$ and Number of $c_3 = 2$.
  Replace them with 1 c_4.

Second, we prove the algorithm is greedy.

(By induction on n)Consider an optimal way to deal with $c_k < n < c_{k+1}$ and the greedy sells the contract c_k. We claim that any optimal solution must also sells the contract c_k. If not, it needs enough contracts of type $c_1, ..., c_{k-1}$ to add up to n.

k	c_k	Restriction	Max value of contracts 1,2,..k-1 in any OPT
1	1	Property 1	-
2	5	Property 1& 2	4
3	10	Property 1&2&3&4	4×1+5= 9
4	25	Property 1&2&3&4	2×10+4×1=24

Then the problem reduces to deal with $n - c_k$, which, by induction, is optimally solved by greedy algorithm.

(b) **Proof:**

(By induction on n)Consider an optimal way to deal with $c^p < n < c^{p+1}, p = 0, 1, ...k$ and the greedy sells the contract c^p. We claim that any optimal solution must also sells the contract c^p.

Property 1. $c^{p+1} \geq 2 \times c^p$

Proof. Integer $c > 1$ means $c \geq 2$. So $c^{p+1} = c^p \times c \geq 2 \times c^p$.

If the optimal solution doesn't select c^p, then it needs enough contracts to sum up to n. From **Property 1**, we know that it at least needs 2 contract c^{p-1} to compensate for not selecting c^p. This is unvoidable because it can not use c^{p+1}. Therefore, the optimal solution must select c^p, which is greedy.

Then the problem reduces to deal with $n - c^p$, which, by induction, is optimally solved by greedy algorithm.

(c) The set can simply be: $\{1M, 6M, 10M, 25M\}$.

For example, if $n = 12$:

By *Greedy Algorithm*, the contract set would be $\{10M, 1M, 1M\}$.

However, the optimal solution is $\{6M, 6M\}$.

Explanation:

By my observation, if the element of the contracts does not satisfy **Property 1** in the **problem b**(shown below), the greedy algorithm can not work.

Property 1. $c_{k+1} \geq 2 \times c_k$

The intuitive proof is shown in Fig. 3. If c_k and c_{k+1}, two adjacent contracts, can not satisfy a relationship that is shown in **Property 1**, the greedy choice may fail to reduce the total number of contracts. In Fig. 3, the left is when the greedy algorithm works and the right is when it does not work. When c_k and c_{k+1} can not satisfy **Property 1**, the greedy algorithm needs at least one contract to replace two c_k, possibly two or more. **This is not optimal.**

□

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.

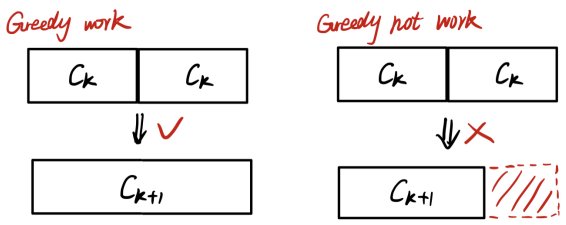

Figure 3: When Greedy Algorithm Works?

(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)

The algorithm. 2 **Greedy Cover Set** takes greedy strategy. To make the number of intervals small as far as possible, each interval begins at a point. This algorithm is intuitively like **Sliding Window**.

Algorithm 2: Greedy Cover Set

```
1  Sort Input points' coordinates by increasing order:{a_1, a_2, ..., a_n};
2  i := 1, num := 0;
3  while i < n do
4      {a_i, a_{i+1}, ..., a_{i+m}} = interval[a_i, a_i + k] cover;
5      num++;
6      i := i + m;
7      if i == n then
8          return num;
9      i++;
10     if i == n then
11         return num+1;
```

While finding the interval's cover sets, we use a loop to scan through $\{a_i, a_{i+1}, ..., a_n\}$ and this loop will stop when finding a_{i+m+1} exceed the interval.

To illustrate this process, we use the data in (c) and simulate the process. Fig. 4 show the process that the intervals are counted by the order.

Worst-case Complexity:

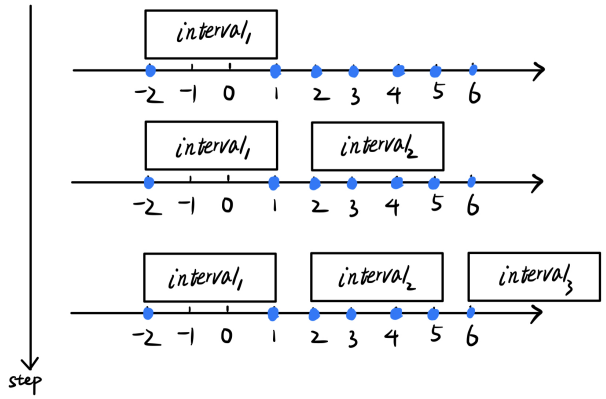

Figure 4: Greedy Cover Set

We use *QuickSort* to sort the coordinates of points, whose worst-case time complexity is $O(n^2)$ when the pivot of *QuickSort* fails to divide and conquer. The space complexity is $O(logn)$.

In the *while* loop, the algorithm is actually traversing all the points for one time. Therefore, the worst case, the average case and the best case are equal, all $O(n)$. The space complexity is $O(1)$.

In conclusion, the time complexity in the worst case is $O(n^2)$, and the space complexity is $O(logn)$.

(b) **Theorem:** Greedy Cover Set is optimal with fixed length k.

Proof: Since the algorithm simply selects the point not covered in order as the start of an interval and then the following process is natural, we should prove the optimal solution also **takes the same action when choosing an interval**.

Assume the optimal solution does not act the same as the greedy at interval q, where q is the smallest number that the two solutions are different.

Then there are two cases:

Case 1: The intervals q in two solutions both cover the same number of points.

In this case, the optim and the greedy will perform similar function.

However, there are slight differences between the two.

As shown in Fig. 5, the greedy can leave more space for other points. Though the extra space is not utilized, this illustrates the wider range of exploration by Greedy Cover Set.

Case 2: The intervals q in the greedy covers more points than that in the optim.

As shown in Fig. 6, the greedy utilizes the length best and can have more points covered, while for the optim, it has wasted some space before.

In conclusion, in case 1 the greedy works the same or even better than the optim, and in case 2 the greedy works better than the optim. Therefore, the assumption is false, which is to say, **Greedy Cover Set** is optimal.

(c) The code is in **Code-SetCover.cpp**, and the result is shown in Fig. 7.

□

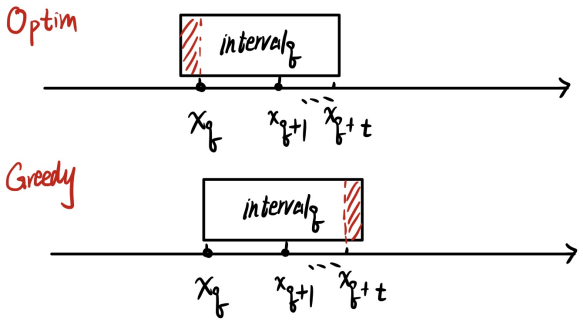

Figure 5: Cover Set Case 1

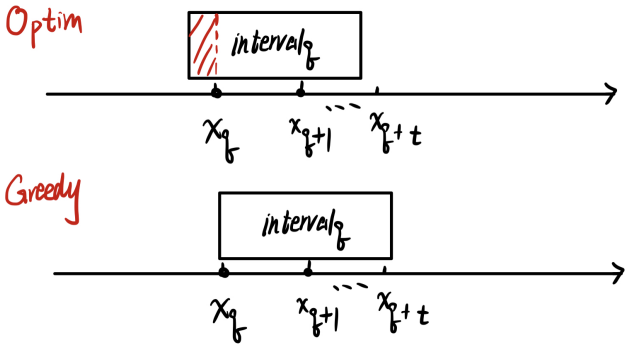

Figure 6: Cover Set Case 2

```
yanjieze@YanjieZedeMacBook-Pro Lab3 % make
g++ Code-SetCover.cpp -o setcover
yanjieze@YanjieZedeMacBook-Pro Lab3 % ./setcover
3
```

Figure 7: Code Result

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