Hongsup OH
GTID:903232384

1.Simple Complexity

a)

(a) $f = (n + 1000)^4, g = n^4 - 3n^3$

ANS: $f = \theta(g)$

(b) $f = log_{1000}(n), g = log_2(n)$

ANS: $f = \theta(g)$

(c) $f = n^{1000}, g = n^2$

ANS: $f = \Omega(g)$

(d) $f = 2^n, g = n!$

ANS: $f = O(g)$

(e) $f = n^n, g = n!$

ANS: $f = \Omega(g)$

(f) $f = log(n!), g = nlog(n)$

ANS: $f = \theta(g)$

b)

In any case, outer loop operates; n times. But I need check how many time inner loop operates for each case. I use j value to check how many times inner loop works. First, I check the case when n = 30 and then generalize the case.

(a).n = 30

(1).j is larger than 1

All case's j is larger than 1: the number of element is n, which can be expressed as following equation: $n - (3^0 - 1)$

(2) j is larger than 2

28 elements' j are larger than 2: the number of element is n - 2 can be expressed as following equation: $n - (3^1 - 1)$

(3) j is larger than 3

22 elements' j are larger than 3: the number of element is n - 8 can be expressed as following equation: $n - (3^2 - 1)$

(4) j is larger than 4

4 elements' j are larger than 4: the number of element is n - 26 can be expressed as following equation: $n - (3^3 - 1)$

Hongsup OH
GTID:903232384

(b).Generalize the running time

Based on the case for n = 30, I can make following equation:

$$\mathrm{T(n)} = [n-(3^0-1)] + [n-(3^1-1)] + [n-(3^2-1)] + \cdots \left[n-\left(3^{\lfloor log_3(n)\rfloor}-1\right)\right]$$

$$\text{where, last term's power is } \lfloor log_3(n)\rfloor$$

And this equation can be expressed as:

$$\mathrm{T(n)} = (n+1)\times\lfloor log_3(n)\rfloor - \sum_{k=1}^{\lfloor log_3(n)\rfloor}(3)^k$$

It is easy to know $(n+1)\times\lfloor log_3(n)\rfloor \equiv O(nlog(n))$. Now I will prove complexity of second term.

If $\mathrm{a} = 3$ and $\mathrm{b} = \lfloor log_3(n)\rfloor$, then

$$\sum_{k=1}^{a}(3)^k = \sum_{k=1}^{b}(a)^k = \frac{a(1-a^b)}{1-a}$$

Then this equation becomes

$$\frac{a(1-a^b)}{1-a} = \frac{3(1-3^{\lfloor log_3(n)\rfloor})}{1-3} = \frac{3}{2}\left(3^{\lfloor log_3(n)\rfloor}-1\right) \le \frac{3}{2}\left(3^{log_3(n)}-1\right) = \frac{3}{2}(n-1)$$

Thus, second term’s complexity is O(n)

Therefore, total time complexity O(n log(n)) (ANS)

2.Greedy1

(a). Algorithm

Index start from 1

```
L = [l_1, l_2, … , l_n]
n = length(L)
i, count = 1, 0
s = length of strip

while i != n:
    Span = [L[i], L[i] +s]

    if (Span covers from L[i] to L[j]):
        i = j + 1

    count += 1
```

Now, I will explain my code. First, we need the data for leaking points, which is L array and its length is n. For example, l_n is nth point for the leaking. Without loss of generality suppose the strips are given in sorted order; such as i < j then $l_i < l_j$. The length of the strip is s. Now,

Hongsup OH
GTID:903232384

I will explain the sequence of the greedy algorithm

1. Start with i = 1
2. Put the first strip at point l_i. It will cover all leaks from l_i to l_j in the interval $[l_i, l_i + s]$ where $l_j < l_i + s$.
3. update i = j+1
4. Repeat step 2 and 3 until i > n
5. Then final value for the count is the number of the strip to cover all leaks.

The algorithm has a runtime complexity of O(n). If L array is not sorted O(n log(n)). Let the solution obtained by the greedy algorithm be represented as G = $\{G_1, G_2, \dots, G_t\}$, where G_i means a point on the strip representing the location of the starting point of the strip; $[G_i, G_i + s]$. Since G array is sorted, if i < j, then $G_i < G_j$. Based on similar notation, the optimal solution will be O = $\{O_1, O_2, \dots, O_r\}$.

(b). Greedy Choice Property

O_1 is placed from point $[O_1, O_1 + s]$ and it covers the first leak, l_1. This optimal solution must cover all leaks to become a feasible solution. Based on the concept of a greedy choice, $G_1 = l_1$, because the greedy solution lays the first strip as far right as possible while covering l_1 ($O_1 < G_1$). Let O' be the solution obtained by replacing O_1 with G_1 in O. We know that there is not any other leaking point between O_1 and G_1, since l_1 is the first and leftmost leak. Thus, strip at G_1 covers all the leaks that strip at O_1 cover. Therefore, O' = $\{G_1, O_2, \dots, O_r\}$.

(c). Optimal Substructure Property

I assume optimal solution for total problem P is O. After putting the first strip at O_1, we need to solve the rest of the parts (or problem) P', which cover all leaking points to the right of the point O_1+ s. Let the optimal solution for P' is O'. Then, Cost(P) = cost(P') + Cost(one strip). Thus, optimal solution for entire problem O includes the solution O'.

(d). Exchange Argument Proof

Let's assume greedy is not optimal solution. Take the optimal solution that agrees with the greedy one for the longest time: Suppose k is the smallest values for Greedy and Optimal solutions are not equal, such as i < k, $G_i = O_i$ but $G_k \neq O_k$. I will examine the optimal solution that agrees with greedy for the maximum value of k.

Based on the theory, we know $O_k < G_k$ (G_k must be as far right as possible, while it covers leaking the G_k's position.). I will explain detailly why $O_k < G_k$:

The greedy algorithm, by definition, places strips left aligned with the location of a leak. Therefore, G_k is representing the strip covering the next unconvered leak in order on the pipe that strips 1⋯k−1 are not covering. Call this leak l'. Optimal solution O_k is placed left to the G_k. So O_k could not cover the leak l'. Thus, it must begin before the greedy's strip to be a feasible solution.

Hongsup OH
GTID:903232384

I will replace O_k with G_k to make new solution; O' = $\{O_1, O_2, \dots O_{k-1}, G_k\ , O_{k+1,\dots}\}$. I will show this solution is feasible as well as optimal.

Feasible

Since $O_k < G_k$, G_k can cover all leaks which is covered by O_k. Thus, all leaking points covered in original solution are still covered. Thus, new solution O' is still feasible

Optimal solution

The number of strip is not increased. Only kth strip's position is changed. Thus, the solution is still optimal (length(O) = length(O'))

Now I show that O' is still feasible and optimal solution. Thus, I can contradict original assumption and thus greedy solution is optimal.

3.Greedy2

(1). Algorithm

Input: (w[1, 2,…, n], v[1, 2,…, n], L)

Calculate density: $\rho_i = \frac{v_i}{w_i}$ (density array is length n)

Sort the density array decreasing order (from bigger density to lower density)

Weight = 0

```
for i = 1 to n:
    if (weigth + w[i] ≤ L):
        x[i] = 1
        weight = weight + w[i]
    else:
        x[i] = (L - weight)/w[i]
        weight = L
        break
```

Now I explain the algorithm. Given data are weight (w) and value (v) of mineral and weight limit (L). Based on weight and value data, density can be calculated, which is value/weight. And then sort density array in decreasing order ($\rho_1 > \rho_2 > \cdots > \rho_n$). In the algorithm x[i] represents the fraction of the element. For example, $0 \le$ x[i] ≤ 1. Thus, our goal is:

$$\text{maximize} \sum_{i=1}^{n}(x[i] * w[i]) \text{ with constraint } \sum_{i=1}^{n}(x[i] * w[i]) \le L$$

If collected minerals' weight are less than L, x[i] is equal to 1 to maximize the value. Otherwise, fraction becomes x[i] = (L-weight)/w[i], where weight is collected minerals total weight, and

Hongsup OH
GTID:903232384

w[i] is ith mineral's weight.

(2). Proof (Exchange Argument)

Let the greedy solution be $G = \{x_1, x_2, x_3, \dots, x_{k,}\}$. Where x_i indicates fraction of item i taken. Here, $\forall x_i = 1, except\ i = k$. And considering any optimal solution $O = \{y_1, y_2, y_3, \dots, y_{n,}\}$. Where y_i indicates fraction of item i taken. Here, $\forall i, 0 \le y_i \le 1$.

And bag must be full in both G and O solution, that is:

$$\sum_{i=1}^{k} x_i w_i = \sum_{i=1}^{n} y_i w_i = L$$

Consider the first item i where the two selections are different. By definition, solution G takes a greater amount of item i than solution O, since the greedy solution always take as much as it can.

Now let $d = x_i - y_i$. And considering the following new solution $O' = \{y'_1, y'_2, y'_3, \dots, y'_{n,}\}$ which is constructed from O.

For j < i , keep $y'_j = y_j$.

Set $y'_i = x_i$

In optimal solution O, remove items of total weight $d \times w_i$ form items i + 1 to n, resetting y'_j appropriately. This is always possible, since $\sum_{j=i}^{n} x_j = \sum_{j=i}^{n} y_j$. The total value of solution O' is greater than or equal to the total value of solution O, since O is the largest possible solution and value of O' cannot be smaller than that of O, O and O' must be equal. Thus O' is optimal.

By repeating this process, we will eventually convert O into G, without changing the total value of selection. Therefore, G is optimal.