Greedy Techniques

Raveesh Gupta

March 2019

Chapter 1

Exchange Arguments

1.1 Generalized

You have a greedy solution G defined by some properties (found by probing) $G = (g_1, g_2, ...g_k)$and you have an optimal solution $O^* = (o_1, o_2, o_3..o_k)$.

In G and O^* you are not looking at the algorithm but you are looking at what the algorithm gives in the end.

Step 1:
Label the solutions,
always label the greedy solution G and optimal solution O^*

Step 2:
Compare the greedy vs optimal.
$\exists x \in O^*, x \notin G$ or $\exists x \in G, x \in O^*$

Step 3:
Exchange the properties
$\exists x \in O^*, x \notin G$ and see how it effects the solution.

This helps in proving some lemmas

I would like to give an example of a simple math problem for an introduction to proving your lemma, Given an integer n , find two integers a, b to maximize $a * b$ such that $a + b = n$.

One way to do this , is to break this down this in 2 cases , n is even and a case when n is odd.

when n is even, and make every number equal to one another make each number

in our example $a = b = x = \frac{n}{2}$ and our product $a * b = x^2$.

Proof: - Characterize every other solution, $a = x + delt$, $b = x - delt$, (without loss of generality a ¿ b) $\forall delt > 0$ and our product is $a * b = (x^2 - delt^2)$ and since we know that $delt^2 > 0$ implies that every other solution is less than x^2.

1.2 Coin denomination problem

Given a value S, and a multi-set of coins $s_1, s_2, s_3 ... s_k$. Find the minimum number of coins such that their summation is S.

Generally greedy doesn't work here. *for ex:* $S = 1000$ and $[502, 499, 1]$, if we use a greedy approach, it will result in $1 \times [502]$ and $499 \times [1]$, that is no way the minimum . A better solution for this is to take $2 \times [499]$ and $2 \times [1]$ coin denominations. So greedy doesn't work in this problem.

A certain subset of this problem will accept a greedy solution i.e.

Given a value S and a multi-set of coins $[25, 10, 5, 1]$, Find the minimum number of coins such that their summation is S. For ex: $37 = [25] + [10] + 2 \times [1]$

To prove this:
We want to know what characteristics does an optimal solution is gonna have. The process its called *probing.*

Suppose $3 \times [10], lesscanbeformed 1 \times [25] + 1 \times [5] = 30$ using just 2 coins can be formed. Our solution can never have more than $2 \times [10]$ coins.

Suppose $2 \times [5]$, less can be formed $1 \times [10] = 10$ using just 1 coin can be formed. Our solution can never have more than $1 \times [5]$ coins.

Suppose $5 \times [1]$, less can be formed $1 \times [5] = 5$ using just 1 coin. Our solution can never have more than $4 \times [1]$ coins.

one more observation is that our solution can never have $2 \times [10]$ and $1 \times [5]$ in conjunction. So the maximum value O^* can get from $[10, 5, 1]$, $[5, 1]$, $[1]$ sets of coins is
$2 \times [10] + 4 \times [1] = 24$,
$4 \times [1] + 1 \times [5] = 9$,
$4 \times [1] = 4$

This gives us a way that how our optimal solution looks like. Label O^*: Optimal solution
G : Greedy solution

O^* is always going to have equal or less number of coins than our solution.
Lets prove lemmas,
Let q_1 be of [25] coins in O^* and q_1^* be of [25] coins in G.

Note here that q_1^* can never be less than q_1.

Assuming both are valid solutions and O^* have the minimum number of coins, Suppose $q_1 < q_1^*$, O^* can have at-most 24 (in value) other coins so they must have same [25] coins. There is a contradiction.

For both solutions to be valid condition $q_1^* = q_1$ must be true.

Let q_2 be of [10] coins in O^* and q_2^* be of [10] coins in G.

Assuming both are valid solutions and O^* have the minimum number of coins, Suppose $q_2 < q_2^*$, O^* can have at-most 9 (in value) other coins so they must have same [10] coins because can't replace [10] with $[5], [1]$ optimally. There is a contradiction.

For both solutions to be valid condition $q_2^* = q_2$ must be true

Let q_3 be of [5] coins in O^* and q_3^* be of [5] coins in G.

Assuming both are valid solutions and O^* have the minimum number of coins, Suppose $q_3 < q_3^*$
O^* can have at-most 4 (in value) other coins so they must have same [5] coins because can't replace [5] with [1] optimally. There is a contradiction.

For both solutions to be valid condition $q_3^* = q_3$ must be true

If q_1, q_2, q_3 coins in both O^* and G then q_4 must be same too.

Thus $O^* = G$, a corollary to this proof is that there can be only one optimal solution.

In conclusion our greedy solution is always optimal.

1.3 Minimizing Dot Product

Given 2 vectors $v_1 =< -3, 4, 2 > v_2 =< 1, -4, 11 >$, rearrange the components of v_1, v_2, so $v_1.v_2$ is minimized. it can be expressed as $MIN(\sum_{i=0}^{k} v_i)$.

List the components of 1^{st} and pairing them up with components of 2^{nd}. Greedy pairing will lead to the minimum dot product.

One intuition is to sort the vector v_1 in increasing order and sort the v_2 in decreasing order and pair the i^{th} component of $v_1$to i^{th} component of v_2.

To prove this, we take v_1, v_2 such that,

$$x_1 <= x_2 <= .... <= x_n,$$
$$y_1 >= y_2 >= ... >= y_n,$$

then suppose a pair that doesn't follow up the property.

$$i < j,$$
$$x_i <= x_j,$$
$$y_i > y_j \textbf{ (swapped property)}$$

Suppose this (swapped property) is the optimal solution O^* and its better,

The case of pairing in our greedy solution G $(x_i * y_i + x_j * y_j)$, and case of pairing in the optimal solution O^* $(x_i * y_j + x_j * y_i)$ to ensure O^* is better we subtract G with O^*

$$\begin{aligned}(x_i * y_i + x_j * y_j - x_i * y_j - x_j * y_i) &= x_i * (y_i - y_j) - x_j * (y_i - y_j) \\ &= (x_i - x_j) * (y_i - y_j) <= 0\end{aligned}$$

it turns out to be non-positive which contradicts that our assumption that optimal solution O^* is always better than the greedy solution G .

Hence greedy solution G is always better or equal to the optimal solution O^*.

So swapping two positions such that $y_i >= y_j$ given that i¡ j makes the optimal solution better.

Repeatedly doing it will eventually make O^* cost equal to the G cost.

1.4 Problems (Topcoder SRM-502 Div1 medium

Chapter 2

Questions on Greedy

2.1 Split the Sequence (Greedy in Dynamic Programming)

We have a sequence of numbers we like to split the sequence of n integers into *k* partitions. The cost of making a partition is the product of sum of the left partition and sum of the right partition. Split the sequence to maximize the cost of the k split.

$1 <= k < n <= 50$
Parenthesis Placements DP Approach
((4, 1)(3, 2))(4, 5, 6, 1)

DP state: $dp(l, r, k)$: Maximum cost of splitting the sequence $a[l..r]$ into k parts.

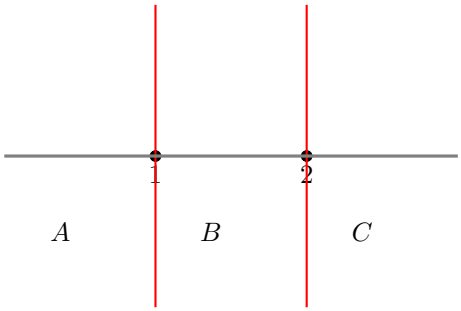
Transitions : $\forall k_1 \in [0, k] \forall i \in [l, r-1] \mid MAX(dp(l, i, k_1) + dp(i+1, r, k - k_1) + cost(l, i) * cost(i+1, r))$

ans: $dp(0, n-1, k)$

```
long long DP(int l , int r , int k)
{
        if(l == r && k != 0) return -INF;
    if(l == r) return return a[l];

    if( dp[l][r][k] != -1) return dp[l][r][k];

    long long sum_total =0, sum_left =0;
        for(int i = l; i <= r; i++) sum_total += a[i];
```




```
    for(int k1 = 0; k1 < k; k1++)
    {
        int k2 = k - k1 -1;
        for(int i = l; i < r; i++)
        {
                sum_left += a[i];
            sum_ right = sum_total - sum_left;
                long long cost = sum_left * sum_right;
                dp[l][r][k] = max(dp[l][r][k],  DP(l, i, k1) + DP(i+1, r, k2) +
        }
    }
    return dp[l][r][k];
}
```

This Approach takes $O(n^3 * k^2)$ complexity.

Better *dp* **Approach for** $1 <=< n <= 10^3, 1 <= k <= min(200, n-1)$

Greedy observation by Probing **Changing the order of the splits doesn't change the total cost.**

Proof by cases: Making the split 1 and then split 2 with total cost $= (B + C) * A + B * C$ is same as making the split 2 and then split 1 with total cost $= (A + B) * C + A * B$ as the cost is equal $(A * B + B * C + A * C)$.

So the associative law applies to splitting the sequence.

The important result is that we can split in order.

Now our *dp* state changes.

DP state: $dp(i, m)$ Maximum cost by splitting sequence $a[i...n-1]$ into m segments.

Transitions: $\forall j \in [i{+}1, n{-}1] \mid MAX(sum(i, j{-}1){*}sum(j, n{-}1){+}dp(j, m{-}1))$

Ans: $dp(0, k)$

```
long long DP(int i, int k)
{
        if(i == n-1 && k != 0) return -INF;
        if(k == 0 || i == n-1) return 0;
    if( dp[i][k] != -1) return dp[i][k];

    long long sum_total = 0, sum_left = 0, sum_right = 0;
        for(int j = i; i < n; j++)
        sum_total += a[j];

        for(int j = i+1; j < n; j++)// placing a split between j-1 and j
    {
        sum_left += a[j-1];
        sum_right = sum_total - sum_left;
        long cost = sum_left * sum_right;
        dp[i][k] = MAX(dp[i][k], cost + dp(j, k-1));
    }
    return dp[i][k];
}
```

This Approach takes $O(n^2 * k)$ complexity.
better complexity than before.

2.2 Nearest Point Problem

Given a sequence of points $a[i] \mid i \in [0, n-1]$ Find a point p such that the maximum of distances $d[i]$ from the points $a[i]$ is minimum. Find $d_{min} = min[max[(|a_i - x|)]]$.

lemma: It is optimum to take p as a mid-point of a_{min} and a_{max}.

Proof:
Let assume that $p \in Z$ is at maximum of distance d_{min} from every point a_i. Observation made from probing is that in our solution p will be furthest away from a_{min} and a_{max} , hence , $d_{min} = max(|a_{min} - p|, |a_{max} - p|)$.
If $p > a_{max}$ then $d_{min} = a_{min} - p$ such that $d_{min} > a_{max} - a_{min} + 1$ and similar to $p < a_{min}$ case. Consequently its always better for $p \in [a_{min}, a_{max}]$ such that the $d_{min} <= a_{max} - a_{min} + 1$.

From sweeping p from a_{min} to a_{max} it is optimal if and only if $|a_{min} - p| <= |a_{max} - p|$ and $|a_{min} - (p+1)| > |a_{max} - (p+1)|$. Its evident that floor division of $\frac{(a_{min}+a_{max})}{2}$ follows such property hence $p = \lfloor \frac{(a_{min}+a_{max})}{2} \rfloor$.

To avoid overflow $p = a_{min} + \lfloor \frac{(a_{max}-a_{min})}{2} \rfloor$. Hence Proved.

2.3 k - Nearest Point Problem

You are given n points $a_1, ...a_n, \mid a_i > 0$ on the OX axis. Now you are asked to find such an integer point x on OX axis that $f_k(x)$ minimal possible.

The function $f_k(x)$ can be described in the following way:

form a list of distances $d_1, d_2 \ldots, d_n$ here $d_i = |a_i - x|$ (distance between a_i and x);
sort list d_i in non-descending order
take d_{k+1} as a result.
If there are multiple optimal answers you can print any of them. Given that $\sum n < 10^5$.

This solution is similar to the Nearest Point Problem but instead of minimizing the maximum distance of x from $a[i]$ we have to minimize $f_k(x)$. $f_k(x)$ is a function that maps the sequence $a[i]$ to d_{k+1} where d_{k+1} is the distance of some point a_j from x such that a_j is the $k + 1^{th}$ point further from x. Finding x such that $f_k(x)$ is minimized is equivalent to finding x such that distance of x from $k + 1^{th}$ point from it (in mod distance) is minimized.

Every $a_i, ...a_{i+k}$ will have an x_i where $x_i = \lfloor \frac{(a_{i+k}-a_i)}{2} \rfloor + a_i$, Find x such that $x_i = \lfloor \frac{(a_{i+k}-a_i)}{2} \rfloor + a_i$. Iterating $\forall i \in [1, n-k]$ and calculate x_i and choose minimum $max(|x_i - a_{i+k}|, |x_i - a_i|)$.

Observation: Closely located k points will have minimum largest distance of them from x **than** *minimum largest distance of far away located k points from x. Hence find x_i such that $(a_{i+k} - a_i)$ is minimum.*

2.4 Array Splitting (Codeforces)