Class 30 加强练习 7 (DP 4)

1.

- What's working behind when you enter "yahoo.com" in browser?
- When database query is slow, what do you think has problem?
- 2sum,
- OO design a coffee machine.

2.

- validate BST,
- give two arrays a[] and b[]. say each entry is sum of a[i] + b[i]. Find the first K sums in ascending order

3.

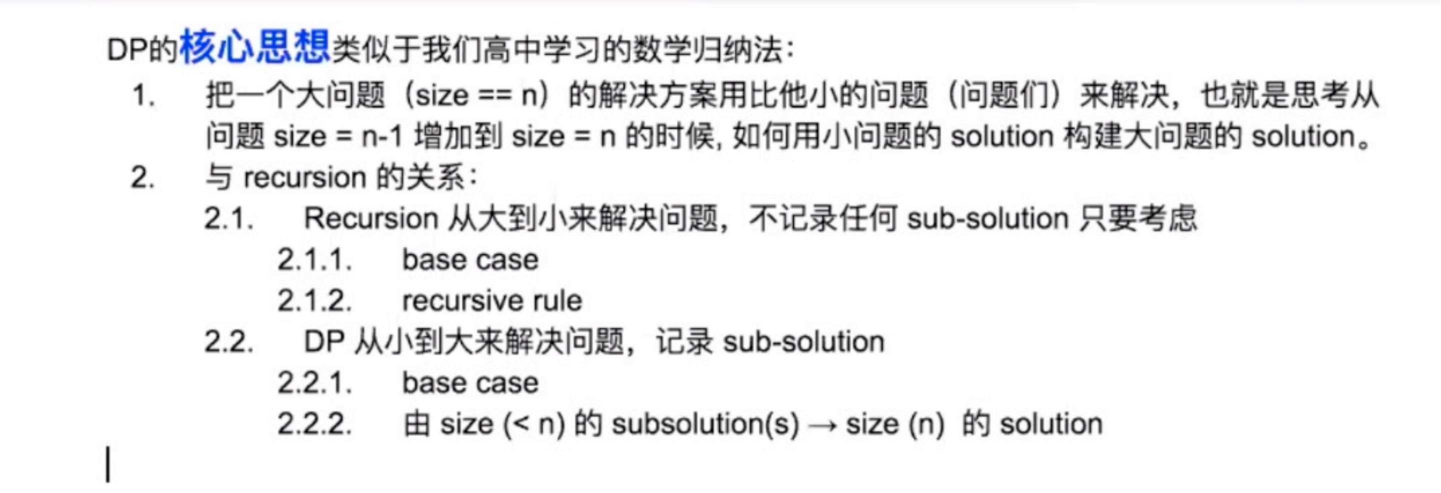
- LRU cache.

4.

- Java questions: what is Final. Finally. Finalize?
- What is stringbuffer what is stringbuilder?
- What is binary search?
- How's java's hashMap implemented?
- What is linkedHashMap?
- Find first non duplicate character in a string, do it in one pass.

5.

- What is polymorphism?
- OO design a file system.



DP的**核心思想**类似于我们高中学习的数学归纳法：

1. 把一个大问题（size == n）的解决方案用比他小的问题（问题们）来解决，也就是思考从问题 size = n-1 增加到 size = n 的时候，如何用小问题的 solution 构建大问题的 solution。
2. 与 recursion 的关系：
    - 2.1. Recursion 从大到小来解决问题，不记录任何 sub-solution 只要考虑
        - 2.1.1. base case
        - 2.1.2. recursive rule
    - 2.2. DP 从小到大来解决问题，记录 sub-solution
        - 2.2.1. base case
        - 2.2.2. 由 size (< n) 的 subsolution(s) → size (n) 的 solution

Q1. Longest common substring/subsequence between two strings.

Q1.1 Longest common substring（solution 中字母必须连续）

Example, student & sweden, then return “den”.

A[] = sweden; **size = m**
i
B[] = student; **size = n**
j

Solution:

M[i][j] represents the longest common substring between the first i letters from A and the first j letters from B (including the i-th letter form A and the j-th letter from B.)

Base case:

M[0][0] = 0
M[0][j] = 0 for all j
M[i][0] = 0 for all i

Induction rule:

M[i][j] = M[i-1][j-1] + 1 Case 1: If A[i] == B[j]
0 else

ind_j		0	1	2	3	4	5	6	7
i			s	t	u	d	e	n	t
0		0	0	0	0	0	0	0	0
1	s	0	1	0	0	0	0	0	0
2	w	0	0	0	0	0	0	0	0
3	e	0	0	0	0	0	1	0	0
4	d	0	0	0	0	1	0	0	0
5	e	0	0	0	0	0	2	0	0
6	n	0	0	0	0	0	0	3	0

return 3

Q1.2 Longest common sub-sequence (字母可不连续)

A == studen t   w   5

i

B == sweden w s

j

Solution:

M[i][j] represents the longest common subsequence between the first i letters from A and the first j letters from B (might not include the i-th letter form A and the j-th letter from B.)

Base case:

M[0][0] = 0

M[0][j] = 0 for all j

M[i][0] = 0 for all i

Induction rule:

M[i][j] = M[i-1][j-1] + 1   if A[i] == B[j]

max(M[i-1][j], M[i][j-1])   if A[i] != B[j]

ind_j		0	1	2	3	4	5	6	7
i			s	t	u	d	e	n	t
0		0	0	0	0	0	0	0	0
1	s	0	1	1	1	1	1	1	1
2	w	0	1	1	1	1	1	1	1
3	e	0	1	1	1	1	2	2	2
4	d	0	1	1	1	2	2	2	2
5	e	0	1	1	1	x	x	x	x
6	n	0	x	x	x	x	x	x	x

Q2 Longest Increasing Sub-Array vs Sub-Sequence problem

Q2.1 Longest Ascending Subarray

Given an unsorted array, **find the length** of the longest subarray in which the numbers are in ascending order. For example: If the input array is {7, 2, 3, **1, 5, 8, 9,** 6}, the subarray with the most numbers in ascending order is {1, 5, 8, 9} and the expected output is 4.

Solution:

M[i] represents from the 0-th element to the i-th element, the value of the longest ascending subarray (including the i-th element)

Base case:

M[0] = 1

Induction rule:

```
M[i] = M[i-1] + 1                if input[i] > input[i-1]
        1                        else
```

2.2 Longest Ascending Subsequence

Given an unsorted array, **find the length** of the longest **subsequence** in which the numbers are in ascending order.

```
For example,
{1,2,4,3,7,6,4,5}
longest ascending subsequence is {1,2,3,4,5}
```

Solution:

M[i] represents from the 0-th element to the i-th element, the value of the longest ascending subsequence (including the i-th element)

Base case:

M[0] = 1

Induction rule:

```
M[i] =  max(M[j]) + 1                    where input[i] > input[j] for 0<=j<i
        1                                if there is no such j
```

```
input = {1,2,      4,          3,          7,                  6,4,5}
M     =  1  1+1=2  3           3           4
                   max(2+1,    max(2+1,    max(3+1
                   1+1)        1+1)        3+1
                                           2+1
                                           1+1)
```

Time = O(n^2)

Q3. There is an array of positive integers with no duplicate, in which each integer represents a piece of Pizza's size, you and your friend take turns to pick pizza from the array. Your friend's strategy is pretty simple, he always picks a larger size pizza from either end of the remaining pizzas each time. You strategy is also to pick a piece of pizza from either end each time. What's the **largest** total sum of all pizza **you can pick assuming you start first.**

Example: 2 4 10 3

```
index 0 1 2 3  4 5 6   7  8   9
        2  3 4 1  xxxxx  4  6  5
```

M[i][j] represents [from the i-th pizza to the j-th pizza] the largest sum I can eat when I pick first.

```
M[0][9] = case 1: if we take the left pizza
                M[2][9] + input[0]                                if input[1] > input[9]
                M[1][8] + input[0]                                else
```

```
            M[1][8] + input[0]                              else
        case 2: if we take the right pizza
            M[1][8] + input[9]                              if input[0] > input[8]
            M[0][7] + input[9]                              else
)
```

Base case:

1 piece of pizza: M[i][i] = input[i]

2 adjacent pieces of pizza: M[i][i+1] = max(input[i], input[i+1])

Induction rule:

```
M[i][j] = max(
        case 1: if i take the left pizza
            M[i+2][j] + input[i]                        if input[i+1] > input[j]
            M[i+1][j-1] + input[i]                      else
        case 2: if i take the right pizza
            M[i+1][j-1] + input[j]                      if input[i] > input[j-1]
            M[i][j-2] + input[j]                        else
)
```

Time = O(n^2)

DP 的解题常用方法：

1. 一维的original data (such as a rope, a word, a piece of wood)，求MAX or MIN (cut, merge, etc..)
   1.1. if the **weight** of each smallest element in the original data is identical/similar
      1.1.1. e.g. **identical**: 1 meter of rope
      1.1.2. e.g. **similar**: a letter, a number

   Then this kind of problem is usually simple:
   Linear scan and look back to the previous element(s)

For example:

Longest Ascending Subarray (when at i, look back at i-1)
Longest Ascending Subsequence (when at i, look back at 1….i-1)
Cut rope
Cut palindrome

1.2. If the **weight** are not the same:

1.2.1. e.g. DP1 课后题： 沙子归并

1.2.2. e.g. 强化练习题： 切木头

从中心开花， [index = 0.1.2.3. N-1], for each M[i, j], we usually need to try out all possible k that (i<k<j), M[i, j] = max (M[i, k] +/-/* M[k, j]) (for all possible k)

1.3 Pizza 问题，两头凑

2. 2D的original data

2.1. Matrix 问题，大班课基本涵盖

2.2. **Two String** 寻找 Minimum Edit Distance, Longest Common Substring/Subsequence, 一般解题方法都是 S1的前i 个letter 和 S2 的前j 个letter 比较。 Induction rule 一般要看 M[i][j] 和 M[i-1][j], M[i][j-1], M[i-1][j-1] 之间关系

Solution 2:

HashMap<key = <a,b>, value = Set<Point>>   // normal case

HashMap<key = x-axis, value = Set<Point>>   // a = +oo

```
for Pi {
        for Pj {
                determine the line y = a * x + b
                <key = <a, b>, value.add(Point)>
        }
}
```

Find the line with the max number of points.

Time = O(n^2)

Corner case: if a = +oo which means that the line is perpendicular to the x-axis.

Q4.2 Given an array of coordinates of points, how to find **the largest number of points** that can form a set such that any pair of points in the set can form a line with positive slope.

Solution:

P1<x1, y1> P2<x2, y2>

$\text{slope} = (y2-y1) / (x2-x1) > 0$

when $x1 < x2$, we must have $y1 < y2$.

The question can be converted to the **LONGEST ASCENDING SUB-SEQUENCE Q(2.2).**

Step1: Sort the points according to their x-coordinates. ⇒ A[N]

Time = O(n log n)

Step2: A[N] = <x0, y0>, <x1, y1> … <xn, yn>

Find the longest ascending subsequence in A[N] according to their y-coordinates.

Time = O(n^2) ⇒ O(n log n)

Q6 **(Advanced topic)** count-array problem

Given an array A[N] with all positive integers from [1...N]. How to get an array B[N] such that B[i] represents how many elements A[j] (j > i) in array A[] that are smaller than A[i]. For example, given A[N] = { 4, 1, 3, 2 }, we should get B[N] = { 3, 0, 1, 0 }. Requirement: Time = O(nlogn).

性质：在**log(n)**层中，combine function 能够让每个element 都会和其他所有的元素compare至少一次。 总的时间复杂度依然是**O(nlogn)**