**第25节课时间: 2015年01月09，星期5，太平洋时间晚上7:00**

USC的同学没有上过CS 571 web technology的同学，建议去旁听，并完成所有作业。

[前15节课教案](https://docs.google.com/document/d/19plkV-WtGNcqYJ9d6k4Gq9T1nUwySXA5Da9NM1xGkL4/edit)

**上课语音会议链接**

<https://global.gotomeeting.com/meeting/join/163634853>

**Announcement:**

1. 班主任赵老师 email: [laioffer9@gmail.com](mailto:laioffer9@gmail.com) Tel 323-893-9079
2. 夏老师email: coding.hx@gmail.com (**负责回答算法相关问题)**
3. 本班级QQ群 316871642 （加入群请使用真实姓名）
4. Online Platform: [www.laicode.com](http://www.laicode.com)
5. 学员讨论问题论坛 <http://laicode.com/forum/>
6. 上机课时间:

Java 周六下午 太平洋时间5:00 - 7:00 pm (闫老师 [laioffer.java@gmail.com](mailto:laioffer.java@gmail.com))

C++ 周六下午 太平洋时间7:00 - 9:00 pm (王老师 [laioffer.cpp@gmail.com](mailto:laioffer.java@gmail.com))

**Homework Solution**

[Java version](https://docs.google.com/document/d/1Oaq3kQMCZNZZvAcgkN9kgAzmDRfGo3KQI-r2Yf0aRQ4/edit)

[C++ version](https://docs.google.com/document/d/1A1n76Fs6tM4ltr2dFM_RRa7MOCkChIhd8YYGWL_JAfA/edit?usp=sharing)

**课程列表**

[Class 16 Dynamic Programming (训练课)](#h.q7bxl1ktppj4)

[Class 17 System Design II (MapReduce & Big Data)](#h.fiyt7k8gpw0u)

[Class 18 Probability, Sampling, Randomization, etc.](#h.brlmcqyanpt6)

[Class 19 强化练习 1](#h.gnp2p05hqesd)

[Class 20 Midterm 2](#h.3gbntx3jaee)

[Class 21 DP (补课)](#h.a0nsaalhrjhj)

[Class 22 强化练习 2](#h.3rator3jd51a)

[Class 23 强化练习 3](#h.2gh9xz2ldi9)

[Class 24 强化练习 4](#h.l0xlisgc33l0)

[Class 25 强化练习 (Recursion 总结)](#h.razgejys670m)

[Class 26 强化练习5](#h.ry8uhx2b5ptc)

# Class 16 Dynamic Programming (训练课)

Fibonacci:

Fn=Fn-1+Fn-2

**Longest Ascending Subarray**

**Maximal Product when Cutting Rope**

**Jump Game**

A = [2,3,1,1,4], return true.

**Q0 Minimum Number of Jumps**

**Q2 Dictionary word problem**

Given a word, can it be composed by concatenating words from a given dictionary? **Example:** Dictionary:

**bob**

**cat**

**rob**

Word: bcoabt

**Q3. Edit Distance**

**(1) Replace**: a->s

**(2) Delete:**

**(3) Insert:**

S(a,b) = min(S(a-1,b-1)+1, S(a-1,b)+1, S(a,b-1)+1)

S(n,m)

**Q4** **Largest square of 1’s in a binary matrix**

0 **0 0 0 0** rt 0

1 1 1 1 0 1

**1 1 1** 1 0

**1 1 1** 0 0 rb 3

**1 1 1** 0 0 4

S(n,m)

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：
   1. Recursion 从大到小来解决问题，不记录任何sub-solution只要考虑 (2^n)
      1. recursive rule
      2. base case
      3. enumerate all possible solution
         1. O(n!) , O(2^n) , O(n^n) - exponential complexity
   2. DP 从小到大来解决问题，记录sub-solution
      1. 由size (< n) 的 subsolution(**s**) → size (n) 的solution
      2. base case
      3. polynomial complexity - O(n^k = 1,2,3)

**TODO (总结DP 常用技巧)**

**思路：**

1. 1-dimentional
   1. subproblem can start from 0 as well
      1. S(0, n) = sum { S(0, i) + simple solution }
2. matrix related problem (2 dimensional DP in general)
   1. S(n,m) depends S(n-1,m), S(n,m-1), S(n-1,m-1)
3. 石子归并 (2 dimensional)
   1. 递归formula S(0,n) 不只依赖于 S(0,i), 还依赖于 S(i,n)
   2. S(0,n) = sum { S(i, j) … } -> 2 dimensional
   3. S(0,n) = sum { S(0,i) + S(i,n) } -> 2 dimensional

注意事项：

1. base case
2. return value ( S(n)? max/min { S(0)..S(n) } )
3. be consistency with how you define n in S(n), it’s either the actual index or size of the problem (length of array)

**Question 0** (most popular DP question) **Largest sum of a subarray**

**A[ ] == 20 -3 -100 1000**

**A = -2 5 3 1 -10 1 2 3 -1 4 5 6 -2 7 8 -8 1**

**for any subarray starting from i and ending at j (0<=i<=j<=n-1)**

**S(n) ?? S(...) ??**

**S(n) is the max sum for all subarray from 0..n-1 <- not constrained on last element a[n-1]**

**S(n) is the max sum for all subarray from 0..n ending at a[n]**

**S(n) = Max{ S(n - 1) + a[n], A(n) }**

**= if S(n-1) <= 0, a[n], if S(n-1)>0, S(n-1)+a[n]**

**1. Base case: S(0) = a(0); S(1) = max {S(0)+a(1), a(1)} …S(n-1)=max{S(n-2)+a(n-1), a(n-1)}**

**2. return max s(i) i = 0 .. n-1;**

**s[0] = a[0];**

**for ( int i = 1; i < n; ++i) {**

**s[i] = max(s[i - 1] + a[i], a[i]);**

**}**

**m = 0;**

**for (int j = 0; j < n; ++j) {**

**m = max(s[j], m);**

**}**

**Question 1**. 一个 [0,1] 一维数组最长连续1的问题。

A[n] = 0011001110111**1**10010100111111111

对比题目：

**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.

S(n) = if A[n]>A[n-1] S(n-1)+1 else 1 ( S(n) 以n结尾的最长上升序列长度）

S(n) = if A[n] == 1 S(n - 1) + 1 else 0

**O(n)**

S[0] = a[0]; int max = 0;

for (int i = 1; i < n; ++i) {

if (a[i] == 0) { s[i] = 0; } else { s[i] = s[i-1] + 1;}

if (s[i]>max) { max = s[i]; }

}

return max;

S(n) = if A[n] == 1 S(n - 1) + 1 else 0

**Question 2**. Given a Matrix that contains only 1s and 0s, how to find the largest cross

with the same arm lengths and the two arms join at the central point of each other.

exmaple:

0**1**00

**111**1 for the pink 1---> min(1, 2, 2, 1) ==>1 => 1\*2+1=3

0**1**00

0100

size = n x n

return 3

S(i,j) = min(longest left arm, longest right arm, longest upper arm, longest lower arm)\*2+1

longest left arm = ? longest 1’s subarray ending at index j in row i

**step1: for each row i, from left to right, call Q1, SL(i,j) is longest left arm - O(n)**

**i = 0..n-1, since we have n row, then == > O(n x n)**

**step 2: for each row i, from right to left, call Q1, SR(i,j) is longest right arm - O(n)**

**⇒ O(n^2)**

**Step 3: for each column i, from top to bottom, call Q1, SU(i,j) is the longest upper arm - O(n) => O(n^2)**

**Step 4: for each column i, from bottom to up, call Q1, SD(i,j) is the longest lower arm - O(n) => O(n^2)**

O(4 \* n^2)

// calculate SL

// calculate SR

// calculate SU

// calculate SD

int max = 0;

for (int i = 0; i<n; ++i) {

for(Int j = 0; j<n;++j) {

S[i][j] = min(SL[i][j], SR[i][j], SU[i][j], SD[i][j]) \* 2 +1;

if (S[i][j] > max) { max = S[i][j]; }

}

}

return max;

**[课后习题]**

1. Biggest rectangle fit under a histogram.

x

y x **xx**

|| xx **xx**x

========>x

[1,2,0,3,2,1] - a histogram

max rectangle area is 4

1. 0,1 matrix, what’s the largest rectangle with all 1’s

**Question 3**. Given a Matrix of integers (positive & negative numbers & 0s), how to find the submatrix with the largest sum?

brute force algorithm complexity:

**O(n^6)** - 枚举左上角 n^2, 枚举右下角 n^2, sum up elements within the matrix n^2

xxxxx

xxxxx

xxxxx [2,1] [3,4]

xxxxx

xxxxx

xxxxx

for (int r1 = 0; r1 <n; ++r1) {

for (Int c1 = 0; c1 <m; ++c1) { // 枚举左上角

for (int r2 = r1; r2 <n; ++r2) {

for (int c2 = c1; c2<m; ++c2) { // 枚举右下角

// check the sum of submatrix [r1][c1] - [r2][c2]

s = 0;

for (int i = r1; i <=r2; ++i) {

for (int j = c1; j <= c2; ++j) { s += m[i][j]; }

}

if (s>max) { max = s ; }

}

}

}

}

**O (n^4)**

for (int r1 = 0; r1 <n; ++r1) {

for (Int c1 = 0; c1 <m; ++c1) { // 枚举左上角

for (int i = 0; i <= n; ++i) { s[i] = 0; }

for (int r2 = r1; r2 <n; ++r2) {

ss = 0;

for (int c2 = c1; c2<m; ++c2) { // 枚举右下角

s[c2] += m[r2][c2];

// s[i] = m[r1][i] + … m[r2][i]

ss += s[c2];

if (ss > max) { max = ss; }

}

}

}

}

**O(n^3)**

reduce to 1 dimension, and use “**Question 0** (most popular DP question) **Largest sum of a subarray”**

HINT: 枚举submatrix的开始row和结束row ( O(N^2) )，把中间的元素拍扁成一个1d array (sum up each column)。 use largest sum of a subarray to solve ( O(N) ).

r1 =1, c1 = 1

2 1 3 1 2 3

1 0 1 2 3 4 S[1] = 0, S[2] = 1, S[3] = 2, S[4] = 3, S[5] = 4,

[1,1]-[1,4] S[1]+S[2]+S[3]+S[4]= ss

1 5 6 7 8 9 S[1]=0+5=5, S[2]=1+6=7, S[3]=2+7=9, S[4]=3+8=11, S[5]=4+9=13

[1,1]-[2,4] = S[1]+S[2]+S[3]+S[4] = ss

1 a b c d e S[1]=0+5+a,

O(N^4)

# Class 17 System Design II (MapReduce & Big Data)

目标: 通过几个例子掌握MapReduce的原理

**I. Word Count**

text file, words separated by space

问题: 每个单词出现的次数?

例子:

Input:

Apple Mango Plum Orange Apple Plum Apple

Apple-3, Mango-1, Plum-2, Orange-1

1. 数据量不大的情况如何处理?

Using HashMap Or Sorting

2. Input非常大怎么办? E.g., Terabytes

2.1 如果可能出现的单词有范围?

2.2 如果单词没有范围 (e.g., 不一定是正确的英文单词)?

怎么做最快?

Multiple machines, counting in parallel

**Map**

Apple Mango Plum Orange Apple Plum Apple

Apple Mango | Plum Orange |....

Machine 1 | M2 |....

M1: Apple -1, Mango -1

M2: Plum-1, Orange-1

M3:.Apple-2, Plum-1

如何汇总结果?

**Reduce**

m1

m12

m2

m1234

m3

m34 m123456

m4

m5

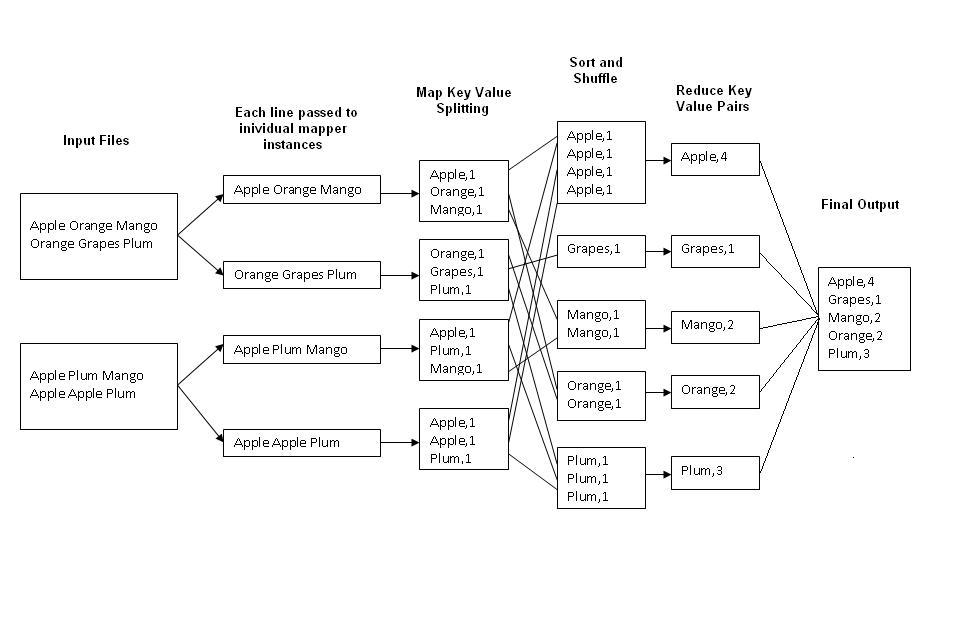
m56 m56

m6

继续利用所有的机器?

**Shuffle** intermediate results

**Map** → **Shuffle** → **Reduce**



Map: 简单切割 (不涉及到data的value)

Shuffle-Reduce: 类似于Hash的分割

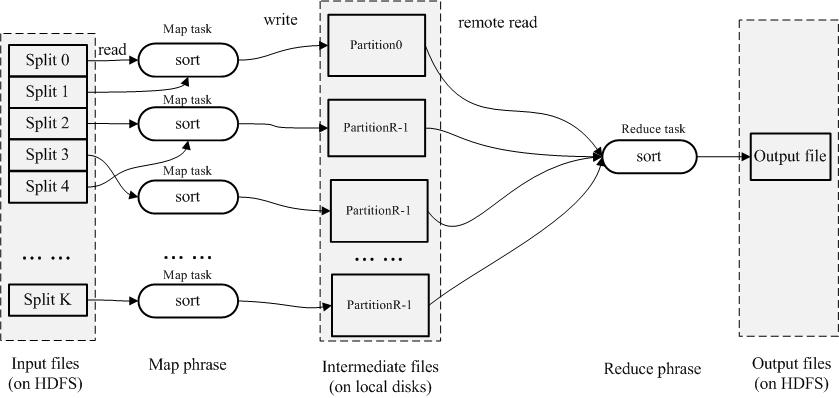
**II. Terasort**

How to quickly sort 1TB data?

每条数据可以看成是一个字符串或者byte[], 比如 aaa < abc

Solution 1:

把原有的数据分布到多个节点上分别排序,最后归并排序.



瓶颈: 归并排序. 单独一个reducer需要处理所有的数据

**Solution 2:**

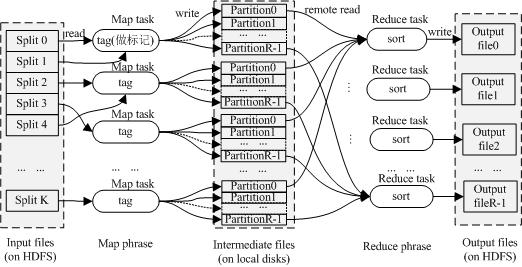
1) 在map阶段, 每个map task都会将数据划分成R个数据块(R为reduce task个数), 其中第i(i>0)个数据块的所有数据都会比第i+1个中的数据大.

E.g., 一个简单的划分策略可以是: 根据每条数据的第一个char划分 (0~9, A~Z, a~z...)

2) 各个mapper分别进行排序.

3) 在reduce阶段, 第i个reduce task处理所有map task的第i块, 这样第i个reduce task产生的结果均会比第i+1个大. -- Shuffle based on partitions

4) 最后将1~R个reduce task的排序结果顺序输出, 即为最终的排序结果.



E.g.

a, b, abc, gif, jpg, haha, abd, bcd, salfjd, abcd, dfjo, efg, oio, hii, daf, rrr, mnk, qfdk, ...

How to define the partitions? Simplest rule: partition based on the first character.

Mapper\_1

input: a, b, abc, gif, jpg

sorted and partitioned: a, abc || b || gif || jpg

Mapper\_2

input: haha, abd, bcd, salfjd, abcd

abd, abcd... || bcd... || haha... || salfjd...

Mapper\_3

input: dfjo, efg, oio, hii, daf

daf, dfjo... || efg... || hii… || oio...

Mapper\_4

input: rrr, mnk, qfdk

mnk... || qfdk... || rrr…

Q1: 如何确定每个map task数据的R个数据块的范围?

Sampling (采样)

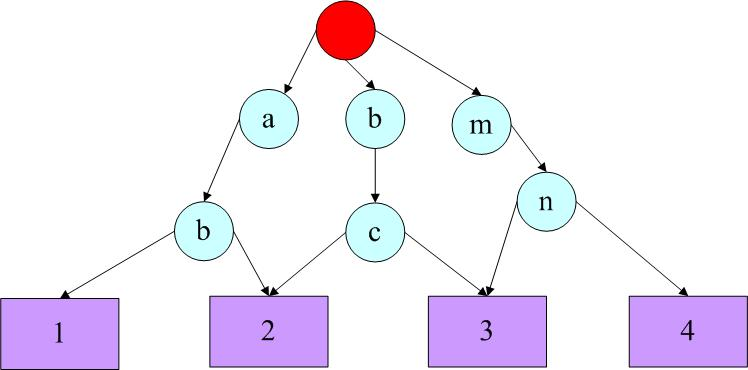
1) 随机采样: b, abc, abd, bcd, abcd, efg, hii, afd, rrr, mnk

2) 对采样数据排序: abc, abcd**, abd**, afd, b**, bcd**, efg, hii**, mnk**, rrr

3) 如果reduce task个数为4 (4个partition), 则分割点为：abd, bcd, mnk

Q2: 对于某条数据, 如何快速的确定它属于哪个数据块? (发生在每一个mapper里)

每一条数据是一个字符串! 使用2层trie树 ([prefix](http://en.wikipedia.org/wiki/Trie) tree): 基于分割点的头两个字母构建trie树.



比如如果数据是aaa, [通过trie树](http://en.wikipedia.org/wiki/Trie)可以立刻知道aaa应该被分在partition 1 (aaa的头两个字母小于ab). 如果数据是dfgh, 那么应该被分在partition 3.

**III. Parallel Breadth-First Search**

**图的定义**

Graph G=(V, E)

V: represents the set of vertices (nodes)

E: represents the set of edges (links)

Both vertices and edges may contain additional information. E.g., distance

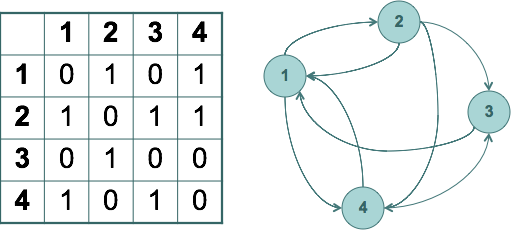
**图的表示**

1) Adjacency Matrices

Represent a graph as an n x n square matrix M:

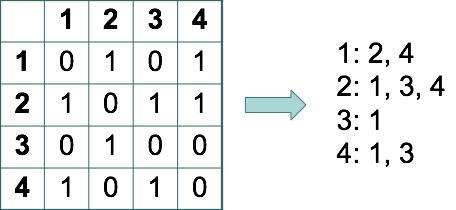
n = |V| (点的个数)

Mij = 1 means a link from node i to j



缺点: 如果不相连的点很多的话, 会有大量的0, 从而浪费空间

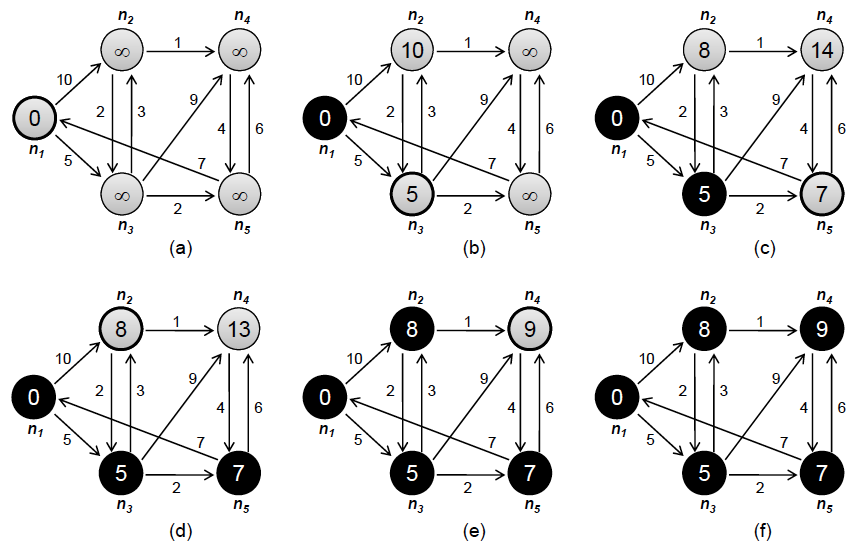
2) Adjacency Lists: 去掉没用的0



**问题: Single Source Shortest Path: find shortest path from a source node to one or more target nodes.**

单机上怎么做?

Dijkstra’s Algorithm



如何并行的处理?

**MapReduce: parallel Breadth-First Search (BFS)**

简化版问题: 假设任意两个相连的点之间的距离都是1

算法思想:

0. 起始点s到自己的距离是0: distance(s) = 0

1. 任何和起始点s相连的点, 它们到起始点的距离都是1

2. 对于任意点n, 它相邻的点集是S, 那么distance(n) = 1 + min(distance(m), m∈S)

**MapReduce 算法:**

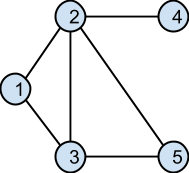
1) A map task receives

Key: node n

Value: D (distance from start), S (list of nodes reachable from n)

对于任意p ∈ S: emit (p, D+1)

2) The reduce task gathers possible distances to a given p and selects the minimum one



Step 1:

M1:

V1: 0, (V2, V3)

emit: d(V2)=d(V3)=1

----------------------------------------------------

R1 (V1, V4):

V1: 0, (V2, V3)

V4: inf, (V2)

R2(V2, V5):

V2: 1, (V1, V3, V4, V5)

V5: inf, (V2, V3)

R3(V3)

V3: 1, (V1, V2, V5)

=============================================================

Step 2:

M1 (V1):

V1: 0, (V2, V3)

emit: d(V2)=d(V3)=1

M2 (V2):

V2: 1, (V1, V3, V4, V5)

emit: d(V1)=d(V3)=d(V4)=d(V5)=2

M3 (V3)

V3: 1, (V1, V2, V5)

emit: d(V1)=d(V2)=d(V5)=2

----------------------------------------------------

R1 (V1, V4):

V1: min(0, 2)=0, (V2, V3)

V4: min(inf, 2)=2, (V2)

R2(V2, V5):

V2: min(1, 2)=1, (V1, V3, V4, V5)

V5: min(inf, 2)=2, (V2, V3)

R3(V3)

V3: min(1, 2)=1, (V1, V2, V5)

=============================================================

Step 3

M1 (V1, V4):

V1: 0, (V2, V3)

V4: 2, (V2)

emit: d(V2)=d(V3)=1

M2(V2, V5):

V2: 1, (V1, V3, V4, V5)

V5: 2, (V2, V3)

emit: d(V1)=d(V3)=d(V4)=d(V5)=2, d(V2)=3

M3(V3)

V3: 1, (V1, V2, V5)

emit: d(V1)=d(V2)=d(V5)=2

----------------------------------------------------

R1 (V1, V4):

V1: 0, (V2, V3)

V4: 2, (V2)

R2(V2, V5):

V2: 1, (V1, V3, V4, V5)

V5: 2, (V2, V3)

R3(V3)

V3: 1, (V1, V2, V5)

**Iterative** MapReduce. Termination condition?

1. 对于简化的问题(相邻点距离是1): No new Vertex found

2. 对于距离可以是任意正数的情况: No improvement

# Class 18 Probability, Sampling, Randomization, etc.

**Question 1**: shuffling algorithm (OOD):

* 1. spades (♠),
  2. hearts (♥),
  3. diamonds (♦)
  4. clubs (♣)

index 0 … **50** 51

cards: 1 2 3 4 5 …. 51 **52 (p=1/52)**

**Step 1: if we can generate a random number i1 in [0...51], pull this card out, and sawp it with position in index == 51**

**Step 2: if we can generate 2nd random number i2 in [0...50], pull this card out, and swap it with position in index == 50**

**The probability of any card that was not selected in step 1 = 1- 1/52 = 51/52**

**1/51 \* 51/52 == 1/52**

P(A) = P(A && B) + P(A && ~B) = P(A | B) \* P(B) + P(A | ~B) \* P(~B)

P(A && B) = P(A | B) \* P(B)

P(the second position is A) = P(the second position is A AND the first position is A) + P(the second position is A AND the first position is not A)

= 0 + P(the second position is A | the first position is not A) \* P(the first position is not A)

= 0 + 1 / 51 \* (1 - 1/52) = 1/52

**Question 2:** How to do sampling for an **unlimited** data flow and when reading the n-th element we are required to return one random number among all numbers read so far, such that the probability of returning any element read so far is 1/n.

O(1) space

t = 10000

for the 10000-th element, the probability of returning it is == 1/10000.

t = 1, the p of returning it is == 1/1 (result\_so\_far = 1)

t = 2, the p of returning the 2nd element == ½, x = random[1...2], iff x = 1 we return the most recent element (== 2nd element (result\_so\_far is set to the 2nd element) ). 1\* (1-½) vs ½

t = 3, the p of returning the 3rd element == ⅓ , x = random[1...3], iff x = 1 we return the most recent element (== 3rd element). ⅓ , the p of 1st and 2nd elements are selected before t= 3 is ½, after t= 3, ½ \* ⅔ = ⅓

**数学归纳法的思维方式**

t = 1 holds (base case)

Let’s **assume** the rule holds until t = k-1; (that is, for the first numbers [1...k-1], the probability of each number in [1….k-1]) to be returned is 1/k-1)

case 1: if the random number at t = k = 1, then we return the k-th number, whose **p = 1/k**

case 2: else , then we still keep (t = k)

p = **1/ (k-1)**  \* (1 - 1/k)

**assumption** when t = k , **x(t=k-1)** is **not replace** by the k-th number

**p= 1/(k-1) \* (k-1)/k = 1/k**

**P(A) = P(A|B) \* P(B) + P(A | ~B) \* P(~B)**

**P(position i is sampled) = P(position i is sampled | position k is not sampled) \* P(position k is not sampled) + P(position i is sampled | position k is sampled) \* P(position k is sampled)**

**= P(position i is sampled | position k is not sampled) \* P(position k is not sampled) + 0**

**= 1/ (k-1)**  \* (1 - 1/k) = 1/k

00 int S[...];

01 int solu = S[1]; //unlimited data flow, which might not stay in the memory forever,

02 while (i++) { // current element is the i-th element

03 int r = random(1，i); // randomly generate a number from 1 to i (**inclusive**);

04 if (r == 1 ) {

05 **solu** = S[i]; // the probability of choosing i-th element as the final solution is 1/i;

06 }

07 // return a number “**solu**” if someone asks for it.

}

[**Reservoir sampling**](http://en.wikipedia.org/wiki/Reservoir_sampling) is a family of [randomized algorithms](http://en.wikipedia.org/wiki/Randomized_algorithm) for randomly choosing a sample of *k* items from a list *S* containing *n* items, where *n* is either a very large or unknown number. Typically *n* is large enough that the list doesn't fit into [main memory](http://en.wikipedia.org/wiki/Main_memory).

Assuming the number of items to select, *k*, is smaller than the size of the source array, *S*):

k element out of n (k << n) → P(1/n)

**Question 3a**: How to design a random number generator Random(7), with **Random**(5).

[0..4] → [0 … 6]

p=⅕ → p=1/7

**x = Random(5)**

**y = Random(5)**

**x + y => [0, 8]**

**0 = 0 + 0**

**4 = 2+2 = 1+3 = 3+1 = 4+0**

High level idea: call Random(5) twice

**R7** == 5 \* **R5\_1** + **R5\_2;**

[0--4] [0--4]

determine row column

5\*  **2**  + 4

**R7**= [0--24]

**0 1 2 3 4**

**0 0 1 2 3 4**

**1 5 6 7 8 9**

**2 10 11 12..**

**3 15 16...**

**4 20 21 22 23 24**

assume we have a random generator R(10) → R(7)????

0 1 2 3 4 5 6 | 7 8 9

p = 7/10 3/10

1/7

**0 1 2 3 4**

**0 0 1 2 3 4**

**1 5 6 0 1 2**

**2 3 4 5 6 0**

**3 1 2 3 4 5**

**4 6 0 1 2 3 21 numbers remain, last 4 numbers are discarded**

**Question 3b**: How to design a random number generator Random(1,000,000), with **Random**(5).

Naive solution:

R5→ R25 → R625 → R (625^2)

1 2 4 8 16

Random(2): 0 vs 1

index **210 [0 - 7]**

--------------------

000

001

010

011

100

101

110

111

Random(10) [0-9] => Random(1000000) ?

**6 random(10)**

**123,456**

Random(5) → Random(2) → Random(1million)

2 ^ 20 > 1,000,000 call Random(2) 20 times → only return when the number generated < 1million

Random(5) → Random(10)

how many time do you need to call Random(5) 2

Random(5) -> Random(1 million) 2 \* 6 = ~12

R(7) => R(5)

[0 - 4] return

[5,6] repeat

5/7 [0-4]

2/7 \* 5/7 [0-4]

2 / 7 \* 2/ 7 \* 5/7 [0-4]

…

..

**Question 4** : Given an **unlimited data flow,** how to keep track of the **median** of the numbers read so far?

1, 2,3, 5, 106,107, 108, 109

example:  **1 3 7 6 -2, 100…………. n**

**1 1&3 3 3&6**

One Min-heap && one Max-heap

**Min-heap**: is to store ~50% large numbers

**Max-heap**: is to store ~50% small numbers

**key idea** is to (somewhat) maintain the number of elements in these two heaps are “roughly” the same.

max\_heap min\_heap

1 2 3 4 5 7 8 9 10

// use two integers to keep track of numbers in each heap. n\_min & n\_max;

When a new elements x comes in:

// use two integers to keep track of numbers in each heap.

// **n\_min**: the number of element in the MIN\_HEAP

// **n\_max**: the number of element in the MAX\_HEAP

There are three cases:

1. if n\_min == n\_max

if x >= MIN\_HEAP.top(), insert this element into MIN\_HEAP

else insert this element into MAX\_HEAP

1. if n\_min < n\_max

if x>= MIN\_HEAP.top(), insert this element into MIN\_HEAP (n\_min == n\_max)

else insert x into MAX\_HEAP, insert MAX\_HEAP.top() into MIN\_HEAP

1. if n\_min > n\_max

if x<= MAX\_HEAP.top() ….

O(nlogn)

**Question 5**: Given 200 of urls, how to find 95-th **percentile** of all url’s length

<http://en.wikipedia.org/wiki/Percentile>

The longest length of a url is 4010

Step 1: allocate an array of a[4010 + 1], and each element i in this array represents

the number of URLs with the length equal to i.

a[i] => how many urls have length == i

Step 2: count the number of URLs with each particular length by scanning each URL.

for each url

a[url.length] ++

O(n)

Step 3: iterate over the array from right-hand side to the left, and sum up all the values on the way until sum = 5% \* 200

sum = 0

for each element in a (from right to left)

sum += a[i]

if sum == 5% \* 200

return i;

O(n)

**Question 6**: Given only two dices, we can only put one digit [0 … 9] on a face. Then how to represent an arbitrary date in a month [1… 31] by using these two dices.

1--31

**Requirements**:

1.All [0...9] must show up at least once.

2.11 and 22 are special case. → 1 and 2 must show up on both dices.

3. 10 20 **30→ must have at least 0 and 3 are on different dices.**

Dice1 Dice2

1 1

2 2

3 0

4 5

6 7

8 9

# Class 19 强化练习 1

**Q1** Array deduplication.

隔板题：

**基本思想**：用两个变量，一个变量记录当前指针位置，一个变量记录隔板位置。

性质：隔板左边是处理好的元素，当前指针右边是未处理的元素，隔板和当前指针之间的区域是无用的元素。每次只要分析当前元素性质是否要加入或者移动隔板就可以了。

**Q1.1**：给定一个排好序的数组，消除里面重复的元素.

a 对于重复元素只保留一个怎么做

input 112223

output 123

public void remove(int[] array) {

if (array == null || array.length == 0 || array.length == 1) {

return;

}

int fast = 1;

int slow = 0 ;

while (fast < array.length) {

if (array[slow] == array[fast]) {

fast++;

} else {

slow++;

array[slow] == array[fast];

fast++;

}

}

//print [0, slow]

return;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for (int i = 0 ; i < array.length; i++) {

if (array[i] == array[j]) {

}

// index:隔板, i 当前元素

int index = 1;

for (int i = 1; i < n; i++) {

if (A[index - 1] != A[i])

A[index++] = A[i];

}

**Q1.2** 只保留2个怎么做

index 0 1 2 3 4 5

input 1 1 1 2 2 3

output 1 1 2 2 3

int index = 2;

for (int i = 2; i < n; i++) {

if (A[index - 2] != A[i])

A[index++] = A[i];

}

**Q1.3** 对于重复的元素一个都不保留怎么做

input 11233

output 2

xxxxxxxxxxxxxxx

|

index

[0, index): processed area

[index, i) useless

[i , n) unknown area to explore.

bool flag = false; // indicates whether we are currently having duplication.

int index = 0;

for (int i = 1; i < n; i++) {

if (A[i] == A[index]) { **//case 1:** if the element scanned == index element

flag = true; // flag indicates we are currently having duplication

} else if (flag == false) {  **// case 2:** not case 1 AND we do not have duplication

A[++index] = A[i];

} else {  **// case 3:** not case 1,2 AND A[i] != A[index]

A[index] = A[i];

flag = false;

}

}

**Q1.4** unsorted array, deduplication repeatedly. (taught already in previous class)

**Q2 (Array number comparisons**)

**Q2.1** Use the least number of comparisons to find the **largest** and **smallest** number.

n element

find the largest element : n-1 comparison to find the largest

find the smallest, n-2 comparison to find the smallest

-> 2n

**Better solution**:

Step 1: binary reduction and for each pair of numbers, put the larger ones to an array, (big array), and put the smaller ones to another array (small array)

1 2 3 4 5 6 7 8

small[] = {1, 3, 5, 7}

large[] = {2, 4, 6, 8}

{1, 2, 3, 4, 5, 6, 7, 8}

[1, 2], [3, 4], [5, 6], [7, 8] => {1, 3, 5, 7}, {2, 4, 6, 8}

{2, 4, 6, 8}

[2, 4], [6, 8] => {2, 6}, {4, 8}

n/4+n/8+.... = n/2

n/2 + n/4x2 + n/8x2 + … = 1.5n

**Q2.2** How to use the least number of **comparisons** to find the largest and second largest number?

{1, 2, 3, 4, 5, 6, 7, 8}

[1, 2], [3, 4], [5, 6], **[7, 8]** => {1, 3, 5, 7}, {2, 4, 6, 8}

{2, 4, 6, 8}

[2, 4], **[6, 8]** => {2, 6}, {4, 8}

{4, 8}

**[4, 8]** => 8

{7, 6, 4} => log(N)

N+log(N)

100 n == O(n)

=====================================================

index = 0 1 2 3 4 5 6 7 8

8 7 6 5 4 3 2 1

7

/ \

5 6

/ \ / \

2 4 3

n + 2 log(n)

**Q3. 2D array print in spiral order or rotate**

**Q3.1**: How to print 2D array in spiral order （NxN）

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

1 2 3 4 5

16 17 18 19 6

15 24 **25**  20 7

14 23 22 21 8

13 12 11 10 9

start: (i, j)

width: k

a[N\*N]

void PrintSpiral(int n) {

int\* p = new int[n\*n];

int max\_layer = (n+1)/2;

int cnt = 0; // counter

for (int k=0; k<max\_layer; ++k) {

for (int i=k, j=k; j<n-k; ++j) p[i\*n+j]=cnt++; // top row

for (int i=k+1, j=n-k-1; i<n-k; ++i) p[i\*n+j]=cnt++; // right col

for (int i=n-k-1, j=n-k-2; j>=k; --j) p[i\*n+j]=cnt++; // bottom row

for (int i=n-k-2, j=k; i>=k+1; --i) p[i\*n+j]=cnt++; // left col

}

/\*

if (k==n-k-1) { p[k\*n+k]=cnt; break; }

int i=k;

int j=k;

while (j<n-k-1) p[i\*n+j++]=cnt++;

while (i<n-k-1) p[i++\*n+j]=cnt++;

while (j>k) p[i\*n+j--]=cnt++;

while (i>k) p[i--\*n+j]=cnt++;

\*/

for (int i=0; i<n; ++i) {

for (int j=0; j<n; ++j) {

printf("%02d ", p[i\*n+j]);

}

printf("\n");

}

}

**Q3.2** How to rotate an NxN matrix clockwise by 90 degree?

1 2 3 4 5

16 17 18 19 6

15 24 **25**  20 7

14 23 22 21 8

13 12 11 10 9

[0][0] -> [0][n-1] -> [n-1][n-1] -> [n-1][0]

**Q4:** BFS print binary tree

a b c d e f g

a

/ \ ---------------------

b c head <- a <- tail

/ \ / \ ---------------------

d e f g

**a**

/ \ ---------------------

b c <- b c <-

/ \ / \ ---------------------

d e f g

a

/ \ ---------------------

**b** c <- c d e <-

/ \ / \ ---------------------

d e f g

a

/ \ ---------------------

b **c** <- d e f g <-

/ \ / \ ---------------------

d e f g

while(!queue.isEmpty()) {

int size = queue.size();

for(int i = 0; i< size; i++) {

Node temp = queue.pop();

if(temp.left != null) {

queue.offer(temp.left);

}

if(temp.right != null) {

queue.offer(temp.right);

}

System.out.print(temp);

}

}

**Q4.2** (Tree) How to print the value of all nodes in a binary tree in a **zig-zag** way?

a

/ \

b c

/ \ / \

d e f g

/ \ / \ / \ / \

h i j k l m n o

a c b d e f g o n m l k j i h

deque -- double ended queue

queue + stack

a

/ \

b c

/ \ / \ --------------------

d e f g <- -> a <- ->

/ \ / \ / \ / \ --------------------

h i j k l m n o

**a**

/ \

b c

/ \ / \ --------------------

d e f g b c ->

/ \ / \ / \ / \ --------------------

h i j k l m n o

a

/ \

b **c**

/ \ / \ --------------------

d e f g f g | b ->

/ \ / \ / \ / \ --------------------

h i j k l m n o

a

/ \

**b** c

/ \ / \ --------------------

d e f g <- d e f g

/ \ / \ / \ / \ --------------------

h i j k l m n o

a

/ \

b c

/ \ / \ --------------------

**d** e f g <- e f g | h i

/ \ / \ / \ / \ --------------------

h i j k l m n o

a

/ \

b c

/ \ / \ --------------------

d e **f** g <- g | h i j k l m

/ \ / \ / \ / \ --------------------

h i j k l m n o

a

/ \

b c

/ \ / \ ------------------------

d e f g h i j k l m n o ->

/ \ / \ / \ / \ ------------------------

h i j k l m n o

public void printZigzag(TreeNode root) {

if (root == null) {

return;

}

Deque<TreeNode> deq = new Deque<TreeNode>();

boolean flag = false; // from\_right\_to\_left ?

deq.offer(root);

while (!deq.isEmpty()) {

int size = deq.size();

for (int i = 0; i < size; ++ i) {

if (!flag) {

TreeNode temp = deq.poll();

System.out.print(temp.val + “ ”);

if (temp.left != null) {

deq.offer(temp.left);

}

if (temp.right != null) {

deq.offer(temp.right);

}

} else {

TreeNode temp = deq.removeLast();

System.out.println(temp.val + “ ”);

if (temp.right != null) {

deq.addFirst(temp.right);

}

if (temp.left != null) {

deq.addFirst(temp.left);

}

}

}

System.out.println();

flag = !flag;

}

}

void PrintZigZag() {

const char tree[] = "abcdefghijklmnopqrstu";

const size\_t len = strlen(tree);

deque<int> d;

d.push\_back(0);

bool left\_to\_right = true;

while (!d.empty()) {

if (left\_to\_right) {

int size = d.size();

while (size-- > 0) {

int parent = d.front(); d.pop\_front();

cout << tree[parent] << ", ";

int lchild = parent\*2+1;

int rchild = parent\*2+2;

if (lchild < len) d.push\_back(lchild);

if (rchild < len) d.push\_back(rchild);

}

} else {

int size = d.size();

while (size-- > 0) {

int parent = d.back(); d.pop\_back();

cout << tree[parent] << ", ";

int lchild = parent\*2+1;

int rchild = parent\*2+2;

if (rchild < len) d.push\_front(rchild);

if (lchild < len) d.push\_front(lchild);

}

}

left\_to\_right = !left\_to\_right;

}

}

**Q5** (Tree) Lowest Common Ancestor

**Q5.1** have two pointers pointing to the children

**Variant 1:**  each node only have two pointers pointing to its children node;

// Lowest Common Ancestor

// Time Complexity: O(n), n is number of nodes in the tree

// 1) if a and b exist in the tree of ‘root’, return their LCA

// 2) if a or b (only one) exist in the tree, return a or b

// 3) return null

00 TreeNode\* **LCA**(TreeNode\* root, TreeNode\* a, TreeNode\* b) {

01 if (root == NULL) {

02 return NULL;

03 }

04

05 if (root == a || root == b) {

06 return root;

07 }

08 TreeNode\* left = **LCA**(root->left, a, b); // recursive rule

09 TreeNode\* right = **LCA**(root->right, a, b);

10 if (left != NULL && right != NULL) { // a and b are in different subtrees

11 return root;

12 } else {

13 return left ? left : right;

14 }

15}

**root**

**/ \**

**d e**

**/ \**

**b c**

**\**

**a**

**Q5.2:** Given k nodes stored in a **vector<Node\*>,** try to find the LCA for these k nodes

Node\* LCA(Node\* root, const vector<Node\*>& nodes);

**Idea1: 2 (a,b) → 3 nodes (a, b, c)**

\_\_\_

d, c → final solution

call how many times LCA(a, b) : n-1 times

**Idea2**: binary reduction: n -> n/2 → n/4

n/2 n/4 + … + 1 == n

**Q5.3**: what if we have a parent pointer for each node in the tree?

**Main idea 1:** use a hashtable to store all ancestors of Node a

Iteratively go up from b to check whether the current node is in the

Node b ancestors

Main idea 2: take layer of the nodes into consideration

# Class 20 Midterm 2

请同学们打开之前孙老师跟每个人share 的1:1 google doc

A complete answer will include the following:

1. Document your assumptions

2. Explain your approach and how you intend to solve the problem

3. Provide code comments where applicable

4. Explain the big-O run time complexity of your solution. Justify your answer.

5. Identify any additional data structures you used and justify why you used them.

6. Only provide your best answer to each part of the question.

Q1. Print string permutations with duplicated letters. Do not print duplicated ones.

E.g. input string “aab”

→ Output:

aab

aba

baa

**index 0 1 2**

**a1 a2 b**

**level 0 a1 a2 b hashset**

**/ \ / \**

**level 1 a2 b a1 a2 hashset**

**| |**

**level 2 b a2**

**// using DFS**

public void **permutation**(String input) {

if (input == null || input.length == 0) {

return;

}

char[] c = input.toCharArray();

permutation(c, 0);

}

// level is the current level to try

public void **permutation**(char[] c, int index **== 0**) {

if (index == c.length) { // base case

printArray(c);

return;

}

**HashSet<Character> done = new HashSet<Character>();**

for (int i = **index**; i < c.length; i++) {

char tmp = c[i];

**if (!done.contains(tmp)) {**

**done.add(tmp);**

swap(c, index, i);

**permutation**(c, index + 1);

swap(c, index, i);

}

}

}

private void swap(char[] c, int i, int j) {

int tmp = c[i];

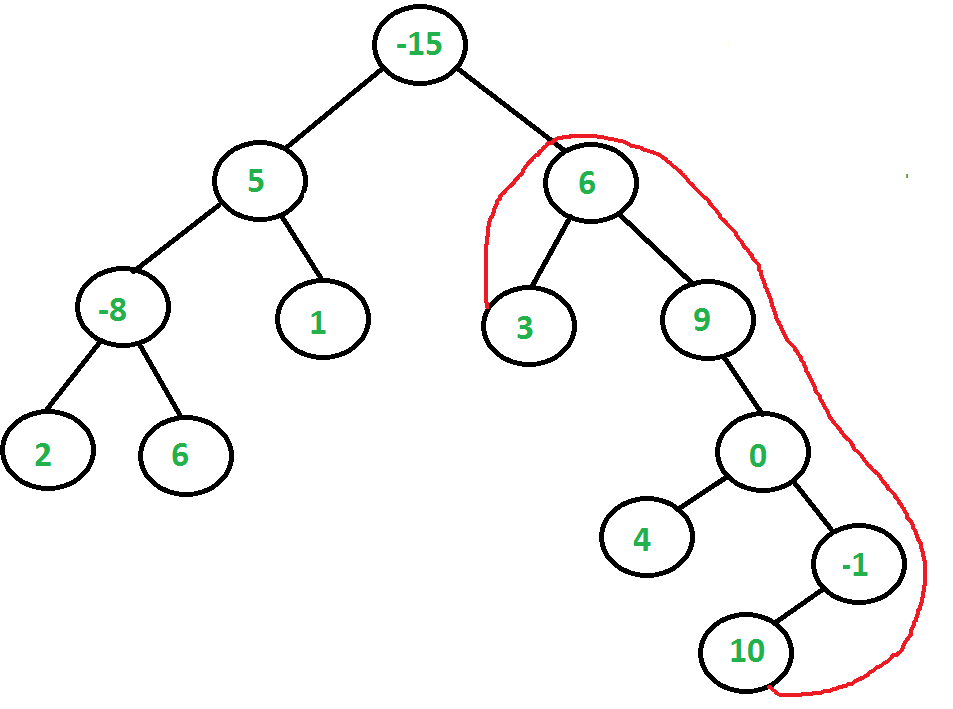
c[i] = c[j];

c[j] = tmp;

}

**Q2**  Given a binary tree in which each node element contains a number. Find the maximum possible sum **from one leaf node to another**.

The maximum sum path may or may not go through root. For ex ample, in the following binary tree, the maximum sum is 27(3 + 6 + 9 + 0 – 1 + 10). Expected time complexity is O(n).



1. **what kind of value do you want to return to your parent node?**

**Answer:** **return** Max( the max path sum from your left subtree,

the max path sum from your right subtree) + root.value;

**2. when should we update the global solution?**

**Sum =**  the max path sum from your left subtree + the max path sum from right left subtree + root.value;

public int sum = Integer.MIN\_VALUE;

public int findMaxSum(TreeNode root) {

if (root == null) {

return Integer.MIN\_VALUE;

}

findMaxSumHelper(root);

return sum;

}

// return max sum of one way of root

private int **findMaxSumHelper**(TreeNode root) {

if (root == null) {

return 0; // base case

}

int leftRes = **findMaxSumHelper**(root.left);

int rightRes = **findMaxSumHelper**(root.right);

int tmpSum = leftRes + rightRes + root.val;

if (tmpSum > sum **&& (root.left != null && root.right != null)){**

sum = tmpSum;

}

**if (root.left == null){**

**return rightRes + root.val;**

**} else if (root.right == null) {**

**return leftRes + root.val;**

**｝**

return Math.max(leftRes, rightRes) + root.val;

}

**Q3**  Given a string, a partitioning of the string is a *palindrome partitioning* if every substring of the partition is a palindrome. For example, “aba |b | bbabb |a| b| aba” is a palindrome partitioning of “ababbbabbababa”. Determine the **fewest** cuts needed for palindrome partitioning of a given string. For example, minimum 3 cuts are needed for “**ababbbabbababa**”. The three cuts are “**a | babbbab | b | ababa**”. If a string is palindrome, then minimum 0 cuts are needed.

1. Fibo M[i] = M[i-1] + M[i-2] // look back to latest two sub-solution
2. Cut rope n meters

\_\_ \_\_ \_\_ \_\_ \_\_ n-1 meters M[1]... M[5]

\_\_ \_\_ \_\_ \_\_ \_\_ | \_\_ M[6] ?????

\_\_ \_\_ \_\_ \_\_ | \_\_ \_\_ M[4] \* M[2]

\_\_ \_\_ \_\_| \_\_ \_\_ \_\_ M[3] \* M[3]

\_\_ \_\_ | \_\_ \_\_ \_\_ \_\_ M[2] \* M[4]

1. Cut string to make each section a palindrome

| i=10

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| string | **a** | **b** | **o** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** |

Solution: M[0] M[1]... M[9]

M[10] = min{ M[9] + 1 iff string[10,10] is a palindrome }

M[8] + 1 iff string [9,10] is a palindrome }

M[7] + 1 iff string [8,10] is a palindrome

……..

M[1] + 1 iff string[1, 10] is a palindrome

}

TIme complexity = O(n^3)

for (i = 1; i < size; i++) {

**M[i] = i-1;**

for (j = 0; j < i; j++) {

if (**str.substring(j, i ) is a palindrome**) {

**M[i] = min(M[i], M[j]+1);**

}

}

}

# Class 21 DP (补课)

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：
   1. Recursion 从大到小来解决问题，不记录任何sub-solution只要考虑
      1. recursive rule
      2. base case
   2. DP 从小到大来解决问题，记录sub-solution
      1. 由size (< n) 的 subsolution(**s**) → size (n) 的solution
      2. base case

**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. if the **weight** of each smallest element in the original data is identical/similar
      1. e.g. **identical**: 1 meter of rope
      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. If the **weight** are not the same:
     1. e.g. DP1 课后题： 沙子归并
     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. **(TODO： 稍微复杂)** 二维的original data (such as two words 求 longest common substring) 2D matrix 求最大sub-matrix 和最大),

**Fibonacci:**

Fn = Fn-1 + Fn-2

F1 = F2 = 1

Naive solution:

Go from Fn down recursively until base cases are reached

Fn 1

Fn-1 breaking point Fn-2 2

Fn-2 Fn-3 Fn-3 Fn-4 4

8…

00 public int **fibN**(int n) {

01 if (n == 1 || n == 2) // base case

02 return 1;

03 return **fibN**(n-1) + **fibN**(n-2); // recursive rule

04 }

**M[i]** represents the i-the element’s value in the Fino sequence.

M[i] = M[i-1] + M[i-2]

index 0, 1 2 3 4 5 ...

M[n] = { 1 1 2 **3 5** x }

---------->

2. **Longest Ascending Subarray (vs sub-sequnce) 1 2 3 4 5 6 7**

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.

**M[i]** represents the current longest ascending subarray ended at the i-th element (including the i-th element)

index = 0 1 2 3 4 5 6 7 ….

A[] = {7, 2, 3**, 1, 5, 8, 9,** 6}

M[] = {1,1,2,1,2,

int global\_max = 2;

**3. Maximal Product when Cutting Rope**

Given a rope with **integer-length *n*,** how to cut the rope into ***m*** integer-length **parts** with length *p*[0], *p*[1], ...,*p*[*m*-1], in order to get the maximal product of *p*[0]\**p*[1]\* ... \**p*[*m*-1]? ***m* is determined by you** and must be greater than 0 **(at least one cut must be made**).

M[0] = 0

**M[1] = 0**

M[2] = 1

M[3] =

base case: 1 meter-long rope

**—— M[1] = 0**

**\_\_ | \_\_ M[2] = max(1,M[1]) \* max(1,M[1]) = 1 \* 1 = 1;**

**\_\_ | \_\_ \_\_**  M[3] = **max(1,**M[1]) \* **max(2,**M[2])

**\_\_ \_\_ | \_\_**  M[3] = M[2] \* M[1] M[i] = M[k] \* M[i-k]

**\_\_ |\_\_ | \_\_ 左大段 6 右大段 5**

**3 \*3 \* 2\*3**

**2\*4.. 1\*4**

**\_\_ |\_\_ \_\_ \_\_ M[4] = M[1] \* M[3]**

**\_\_ \_\_| \_\_ \_\_ M[4] = M[2] \* M[2]**

**\_\_ \_\_\_ \_\_ |\_\_ M[4] = M[3] \* M[1]**

**Method 2: 右大段再不可分 （左大段是可分的）**

**\_\_ \_\_ L=i-1\_\_ \_\_ \_\_ \_\_ | \_\_ R=1**  **M[i]** =MAX(

**max**(i-1, M[i-1]) \* **1,**

**\_\_ \_\_ \_\_ \_\_ \_\_| \_\_ \_\_**  **max**(i-2, M[i-2]) \* **2,**

**\_\_ \_\_ \_\_ \_\_ |\_\_ \_\_ \_\_**  **max**(i-3, M[i-3]) \* **3,**

…… …. )

**\_\_| \_\_ \_\_ \_i-1meter\_ \_\_ \_\_ \_\_ last possible cut**

**Q3**  Given a string, a partitioning of the string is a *palindrome partitioning* if every substring of the partition is a palindrome. For example, “aba |b | bbabb |a| b| aba” is a palindrome partitioning of “ababbbabbababa”. Determine the **fewest** cuts needed for palindrome partitioning of a given string. For example, minimum 3 cuts are needed for “**ababbbabbababa**”. The three cuts are “**a | babbbab | b | ababa**”. If a string is palindrome, then minimum 0 cuts are needed.

j=0 ... j =4 ← i=4

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | **10** |
| string | **a** | **b** | **o** | **b** | **c** | **d** | **e** | **f** | **g|** | **h** | **i** |

M[] 1 2 ….. 8 x?

**Similarity** between this question and cutting rope

**look back and check all possible cut position (which cut the string/rope into two big sections)**

M[0..8] must be in the M[ ] table

**Difference** is the way to deal with the **right big section**

Example:

index 0, 1, 2, 3, 4, 5, ...

M[i] = { 0, 0 x }

a

a | b

a b **(reuse subsolution)** | o **(is palindrome or not)**

a (**reuse subsolution)**  | bo **(is palindrome or not)**

min( sub-solution for “ab”, o== yes, sub-solution for “a” , bo == “no” )

a b | o b min (cost of all three possible cut **that is valid**)

j=2 i =7 if you want to determine whether sub-string [j, i ] is a **palindrome** or not

**a** **bx| xxx** **b [2, 7]**

charAt(7) == charAt(2) && **IsPalindrom**[3, 6] **// that is [2, 7] depends on [3, 6]**

**for (i = 0; i < n; i++) { // i is right end of the sub-string**

**for (j = 0; j < i; j++) { // j is the left end of the sub-string**

**isPalindrome**(j, i) + M[j]  **//O(n) → O(1)**

}

} // naive : O(n^3) → O(n^2)

s t u d e n t

j→ i

i = 0  **[j = left end, i = right end]**

s **t u d e n** t

[ j **[j+1 i-1]**  i]

**int M1[i]**  represents the sub-solution of the MIN number of cut that can make each sub-section of sub-string [0..i] palindrome

**boolean** **M2[**j][i]: represents whether the sub-string [j, i] is a palindrome or not.

**[j, i] = [start, end]**

i = 0

i = 1, j = 0; [0, 1]

i = 2, j = 0 , 1. [0, 2] [1, 2]

i = 3 [0, 3] [1, 3] [2, 3]

i = 5 j = 0 1 2 3 4 [0,5] [1, 5] [2, 5] [3, 5]… [4,5]

i = 6 [0,6] [1, 6] [2, 6], **[3, 6] ...**  [5, 6]

i = 7 [0, 7] [1, 7] [2,7]… [6, 7]

**Q3. Edit Distance**

Given two strings of alphanumeric characters, determine the minimum number of **Replace**, **Delete**, and **Insert** operations needed to transform one string into the other.

**Example:**

s1 = “asdf”

s2 = “sghj”

s1 == c1 | s1r ← rest of s1

s2 == c2 | s2r ← rest of s2

Example:

s1= a | sdf

s2= s | ghj

**(1) Replace**: a->s

**s** sdf

s ghj

editDistance(sdf, ghj) + 1

**(2) Delete:**

\_sdf

sghj

editDistance(sdf, sghj) + 1

**(3) Insert:**

s asdf

s ghj

editDistance(asdf, ghj) + 1

00 public int **editDistance**(String word1, String word2) {

// Base case

01 if (word1.isEmpty()) return word2.length();

02 if (word2.isEmpty()) return word1.length();

// **(a)** Check what the distance is if the characters are equal

// and we do nothing first

03 int nothing = Integer.MAX\_VALUE;

04 if (word1.charAt(0) == word2.charAt(0)) {

05 nothing = **editDistance**(word1.substring(1),

06 word2.substring(1))

07 }

// **(b)** Check what the distance is if we do a **Replace** first?

08 int replace = 1 + **editDistance**(word1.substring(1),

word2.substring(1))

// **(c)** Check what the distance is if we do a **Delete** first?

09 int delete = 1 + **editDistance**(word1.substring(1), word2);

// **(d)** Check what the distance is if we do a **Insert** first?

10 int insert = 1+ **editDistance**(word1, word2.substring(1));

// Return best solution

11 return **min**(nothing, replace, delete, insert);

12 }

**O(4^(m+n))**

**s1 = “a”**

**s2 = “s”**

c1 is the last letter of the string s1

c2 is the last letter of the string s2

**s1 = s1r + c1**

**s2 = s2r + c2**

**we grow the string from the left hand side to the right hand side =====>**

**Case 1**. do nothing does not apply here, since s1[0] != s2[0]

**M[i][j] = M[i-1][j-1] iff(S1[i] == S2[j])**

**Case 2**. **replace** c1 with c2: distance(s1r + c1, s2r + c2) = 1 + distance(s1r,s2r)

a

s

editDistance(1,1) -> 1 + editDistance(0,0)

**editDistance(i,j) -> 1 + editDistance(i-1, j-1)**

M[i][j]

**Case 3.** **delete** c1: distance(s1r+c1, s2r+c2) = 1 + distance(s1r, s2)

a

s

editDistance(1,1) -> 1 + editDistance(0,1)

**editDistance(i,j) -> 1 + editDistance(i-1, j)**

**Case 4.** **insert** a new char (c2) to the right side of c1: distance(s1+c2, s2r+c2) = 1 + distance(s1, s2r)

as

s

editDistance(1,1) -> 1 + editDistance(1,0)

**editDistance(i,j) -> 1 + editDistance(i, j-1)**

**s2** **s g h j**

ind 0 1 2 3 4

**s1** 0 **0 1 2 3 4**

**a** 1 **1 1 2 3 4**

**s** 2 **2 1 2 3 4**

**d** 3 **3 x x x x**

**f** 4 **4 x x x Y**

Time = O(m\*n)

Space = O(m\*n)

**Question 3**. Given a Matrix of integers (positive & negative numbers & 0s), how to find the submatrix with the largest sum?

A[i][j]=

**1111111** 1111

**2222222** 2222

**3333333** 3333

Sum of [1][4] -- [2][6] = M[2][6] - **Red - blue + green**

Common property :[0][0] is color area’s top left corner

**M[i][j] = how to fill in the form????**

M[i][j] =

123456789…

369 M[i][j] = (sum so far on this row to the j-the colomn + M[i-1][j])…..

xxxxxxxxxxxx …. we can keep trace of sum of the element in the current row.

xxxxxxYxxxxx

**how many ways of selecting sub-matrix????**

**Cn\_2 n^2 how to select two values from n values ???? Cn-2**

**Cn\_2 n^2 → O(n^4)**

**in order to fill in the M[i][j] table O(n^2)**

**M[i][j]** represents **top-left** corner [0][0], **Bottom-right**[i][j], its sum is stored in M[i][j].

**拍遍方法 O(n^3)**

**Question 0** (most popular DP question) **Largest sum of a subarray O(N)**

1. **base case** ?
2. **induction rule**
   1. M[i] represents what????
   2. M[i] = M[i-1] ??????

index 0 1 2 3 4 5 6

**A[i] 1, 2, 4,** -1, -2, -1 -100  **98**  10000 |||| -1 10

S[i] 1 3 7 6 4 3 -97 98 10098

**base case: M[0] = A[0]**

**M[i] : resprents the largest sum of subarray which end with A[i];**

**M[i] = M[i - 1] + A [i] if (M[i - 1] > 0)**

**A [i] if(M[i - 1] <=0)**

**Example**

index = 1 2 3 4 5 6 7 8 9 4 x 9 matrix

**2 3 4 -6 3 2 1 1 4 row\_upper =1 up**

**1 2 3 -4 5 6 7 8 9 row\_bottom=2 down how can we calculate**

**M[i] = A[up][i] + A[up+1][i] + A[up+1][i] + …. + A[down][i]**

**| | | | | | | | |**

**3 5 7 -10 8 8 8 9 13 = sum of row1 && row2**

3 5 7 -10 8 8 8 9 13

2 3 4 6 3 2 1 1 4

How many possible ways of selectfind the upper and bottom row

**2 3 4 -6 3 2 1 1 4 row\_upper =1 up**

**1 2 3 -4 5 6 7 8 9 row\_bottom=2 down how can we calculate**

3 5 7 -10 8 8 8 9 13

2 3 4 6 3 2 1 1 4

**M[i][j]** represents the sum on each column j from the 0-the row to the i-th row. O(n^2)

23xxxxxxxx

**35xxxxxxxx**

**610xxxxxxxx →**

**4(6-2) 7(10-3) …...**

813xxxxxxxx

# Class 22 强化练习 2

**Q1: skiplist / graph copy problems**

**Q1.1 Copy a skip list**

**input:**

**N1 -> N2 -> N3 -> N4 -> N5 -> NULL**

**| | ^ ^**

**| | | |**

**|-----|------ |**

**|------------------**

**output:**

**N1’ -> N2’ ->N3’ ->N4’ ->N5’ -> NULL**

**| | ^ ^**

**| | | |**

**|-----|------ |**

**|------------------**

**Two pass copy:**

**mapping: <N1’s address, N1’ ’s address>**

**<N2’s address, N2’ ’s address>**

**…..**

**<Nn’s address, Nn’ ’s address>**

**iteration 1**: make a copy of the linkedlist with -->next pointer only, using **hashmap** to build 1:1 mapping between…..N\_i and N\_i’

**iteration 2:** make a copy of the linkedlist with -->forward pointer only

public ListNode copy(ListNode head) {

if (head == null) {

return null;

}

HashMap<ListNode, ListNode> map = new HashMap<ListNode, ListNode>();

ListNode newHead = new ListNode(head.value);

map.put(head, newHead);

ListNode newCur = newHead; // curr pointer in the new copied list

while (head != null) {

if (head.next != null) {

if (!map.containsKey(head.next)) {

map.put(head.next, **new** ListNode(head.next.value));

}

newCur.next = map.get(head.next);

}

if (head.forward != null) {

if (!map.containsKey(head.forward)) {

map.put(head.forward, new ListNode(head.forward.value));

}

newCur.forward = map.get(head.forward);

}

head = head.next;

newCur = newCur.next;

}

return newHead;

}

**Q1.2: (Graph)** How to copy a graph with possible cycles? G-> G’

N1 -- N2 N1’ --- N2’

\ / \ /

N3 N3’

single link vs double link

If we use **BFS**

Step1: Expanding N1:

a) N1’=copy\_node(N1), node\_map={N1->N1’}

b) copy\_edges(N1-N2, N1-N3)

b1) N2 not in node\_map => N2’=copy\_node(N2), node\_map={N1->N1’, N2->N2’}

b2) N3 not in node\_map => N3’=copy\_node(N3),

node\_map={N1->N1’, N2->N2’, N3->N3’}

b3) connect N1’-N2’, N1’-N3’

Step2: Expanding N2:

a) N2 already in node\_map, no need to copy node.

b) copy\_edges(N1-N2, N2-N3)

b1) N1 already processed, no need to process again.

b2) N3 already in node\_map, no need to copy node.

b3) connect N2’-N3’

Step3: Expanding N3:

a) N3 already in node\_map, no need to copy node

b) copy\_edges(N1-N3, N2-N3)

b1) N1, N2 already processed, no need to process again.

b2) nothing to do.

**Q2**  **k-way merge problems**

**Q2.1** How to merge  **k sorted array** into one big sorted array?

m elements in each array

Method1: Use a size-k min-heap,

Time: O(k+km lgk)

Space: O(k+km)

Method2: binary-reduction

Time: O(mklgk)

Space: O(2\*km)

IO: more

**Q2.2** How to merge  **k sorted LinkedList** into one big LinkedList?

**Q3 Binary search tree (BST: find, insert, remove a node)**

**Q3.1: (Find a node whose value is closest to the target value)**

10

/ \

5 15

/ \ / \

2 7 12 20

Case (1) if the current\_node value < target value, compare the difference of current node value with the target value, update if it is closer; Go to the rChild node

Case (2) if the current\_node value > target value, compare the difference of current node value with the target value, update if it is closer; Go to the lChild node;

pseudo code:

def find\_closest(root, x):

min\_diff=max

ret=root.val

n=root

while true:

diff = x - n.val

if abs(diff) < min\_diff:

min\_diff = abs(diff)

ret = n.val

n = diff<0 ? n.left : n.right

if n==null: break

return ret

**Q3.2**  Given a **BST**, how to **find** the largest element in the tree that is smaller than a target number x.

pseudo code:

def find\_biggest\_among\_smallers(root, x):

min\_diff=max

ret=root.val

n=root

while (true):

diff = x - n.val

if diff>0 and diff<min\_diff:

min\_diff = diff

ret = n.val

n = diff<0 ? n.left : n.right

if n==null: break

return ret

**Q3.3** How to **remove** a target node from BST

**Method:** three cases

1. root.val = target
   1. if root has both lChild and rChild, **First**, find the value x of the smallest element from the right-subtree, and replace root’s value with x. **Second**, remove the smallest element from the right-subtree
   2. else replace root with (1) non-null child if any, or (2) null
2. root.val > target go to left subtree by recursion.
3. root.val < target go to right subtree by recursion.

pseudo code

def delete\_node(root, x):

if root.val > x: delete\_node(root.left, x)

elif root.val < x: delete\_node(root.right, x)

else:

if root.left == null and root.right == null: del(root)

elif root.left != null and root.right == null: swap(root.left, root)

elif root.right != null and root.left == null: swap(root.right, root)

else:

n = find\_smallest(root.right) // or find\_biggest(root.left)

swap(root, n)

del(root)

def find\_smallest(root):

n = root

while n.left != null: n = n.left

return n

**Q3.4** How to **insert** a target node to a BST

**Q4 (DP 1D different weight for each smallest element)**

**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. if the **weight** of each smallest element in the original data is identical/similar
      1. e.g. **identical**: 1 meter of rope
      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)**

* 1. If the **weight** is not the same:
     1. e.g. DP1 课后题： 沙子归并
     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)**

\_\_\_ \_\_\_**i \_\_\_\_\_\_\_\_k1 \_\_\_\_ k2 \_\_\_\_\_\_\_\_\_\_\_\_k3 \_\_\_\_\_\_\_j** \_\_\_\_\_

1. **(TODO： 稍微复杂)** 二维的original data (such as two words 求 longest common substring； 2D matrix 求最大sub-matrix 和最大),

\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ |\_\_ M[8] \* M[1] M[9]  **we must have M[1] M[2] M[3] …. M[8]**

\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_| \_\_ \_\_ M[7] \* M[2]

Q4.1 有一个长为L米的木料需要割开，需要切的位置在一个数组里A[0...N]，从一个地方切开的 cost是当前所切木料的长度。**按不同的顺序切割**，得到的total cost是不一样的，问怎么切cost最 小。 比如一个木料现在10米长，然后切的位置是2米处，4米处和7米处

（就是说arr A里A[1]是2，A[2]是4， A[3]是7）。那么比如先切2米，那么得到cost是

10（因为现在木料长度为10），然后切4米处，那么cost变成10 + 8(因为8是现在切的

时候木料的长度)。然后切7米处，cost变成10 + 8 + 6。那么这种切法总共的cost是24

**index**

**0 1 2 3 4**

**0**  1 2 3 4 5 6 7 8 9 10

**|\_\_ \_\_ | \_\_ \_\_| \_\_ \_\_ \_\_** | \_\_ \_\_ \_\_ |

[ i=1 j=4 ]

base case: (i == j-1) || ( i==j) M[i][j]== 0;

递进关系：

A[3] = { 2, 4, 7} // given an array that defines the possible places that we can cut the wood.

i j

**index 0 1 2 3 4**

**A[5]** = {0, **2, 4, 7**, 10} // 0 和 10为左右两端，后加进去的index的对应的米数，虽然不能被刀切。

**当我们从[i, j-1] ⇒ [i, j],**

minimal cost from A[i] to A[j].

**M[i][j]** = A[j]- A[i] + e.g i = 1 j= 3, A[i]=2, A[j]=7

cost of this cut

**min**{M[i][i+1] + **M[i+1][j]**,

M[i][i+2] + M[i+2][j],

M[i][i+3] + M[i+3][j],

....

M[i][j-1] + M[j-1][j]}

**(1)fill in the M[][] form from left to right**

**(2)fill in the M[][] form from bottom up.**

M[1][4] = A[4]-A[1] ==8

**+min**(**M[1][2] + M[2][4]**, **// case 1**, left\_section=1-2,

//right section=2-4)

**M[1][3] + M[3][4])** **// case 2,** left\_section=1-3,

//right section=3-4)

[i][j] i <= j

[i][i+1]

index 0 1 2 3 4

0 **0** **0** 4 11 x

1 x **0 0 5 x M[1][3] = A[3] - A[1] + min (M[1][2] + M[2][3]) = 5 + 0**

2 x x **0 0** 6 M[2][4] = A[4]-A[2] + min (M[2][3] + M[3][4]) = 6

3 x x x **0 0**

4 x x x **x 0**

M[i][i+1]=0

M[0][2]=4+min{M[0][1]+M[1][2]}=4

M[1][3]=5+min{M[1][2]+M[2][3]}=5

M[2][4]=6

M[0][3]=7+min{M[0][1]+M[1][3], M[0][2]+M[2][3]}=7+min{5, 4}=11

M[1][4]=8+min{M[1][2]+M[2][4], M[1][3]+M[3][4]}=8+min{6, 5}=13

M[0][4]=10+min{M[0][1]+M[1][4], M[0][2]+M[2][4], M[0][3]+M[3][4]}=10+min{13, 4+6, 11}=20

# Class 23 强化练习 3

**Q1. Common Element Problems**

**Q1.1** Find common elements in two arrays

A[m] = 1, 2,3 , 4, 5

B[n] = 4, 5, 6

sorted

1, 2, 3 , 4, 5

4, 5, 6

void findComm(int[] A, int[] B) {

assert A!=null && B!=null;

if (A.length==0 || b.length==0) {  
 return;

}

int indexA=0;

int indexB=0;

while(indexA<A.length && indexB <B.length) {  
 if (A[indexA]< B[indexB]) {  
 indexA++;

} else if(A[indexA]>B[indexB]]) {  
 indexB++;

} else{

System.out.println(A[indexA]);  
 indexA++;

indexB++;

}

}

**Q1.2** Find common elements in 3 sorted arrays p

1, 2, 3, 5, 10

3, 4, 6

3, 9

public void **commonElements**(int[] array1, int[] array2, int[] array3) {

int index1 = 0;

int index2 = 0;

int index3 = 0;

while (index1 < array1.length && index2 < array2.length && index3 < array3.length) {

if (same(array1[index1], array2[index2], array3[index3])) {

print(array1[index1], array2[index2], array3[index3]);

index1++;

index2++;

index3++;

} else if (array1[index1] <= array2[index2] && array1[index1] <= array3[index3]) {

index1++;

} else if (array2[index2] <= array1[index1] && array2[index2] <= array3[index3]) {

index2++;

} else {

index3++;

}

}

}

}

**Q2 String replacement problems**

**Q2.1** Replace all substrings s1 in a string s with s2

(with possible minimum memory allocation, in-place if possible)

s1 s2

Example: **“\_”** -> **“20%”**

url: [www.google.com?info=](http://www.google.com?info=)flower\_market..l….

**Now what if we do not know the size relationship between s1 and s2?**

Compare the lengths of s1 and s2:

(1) if s1.size < s2.size (then replace from the right hand side to the left hand side, discussed in the class)

(2) if s1.size >= s2.size (then replace from the left hand side to the right hand side)

s1.size == m

s2.size == k

s.size == n

use a strstr() helper function to identify the occurrence of s1 in the string s.

Time complexity: O(mn)

assume that there are x times s1 in s;

calculate the size change in s1: == x \*|m-k|

int strstr(const string& s, const int startIndex, const string& s1);

void Replace(string\* s, const string& s1, const string& s2) {

vector<int> index;

if (s1.size() < s2.size()) {

// find how many s1 in s.

int k = 0;

int find = strstr(s, 0, s1);

while (find != -1) {

k++;

index.push\_back(find);

find = strstr(s, find + s1.size(), s1);

}

// get newsize, replace from right to left.

} else {

// replace from left to right.

}

}

**Q2.2:**  (string) Given a string such as “a3b1c4d0” → “aaabcccc”

**Observation**:

if 数字为0 or 1则可以使得original string变短，

if 数字 >= 2, 则可以使得original string不变或者变长。

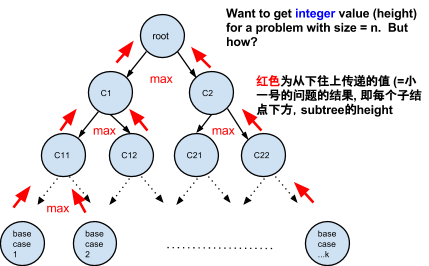
therefore, we need to deal with the two cases separately.

solution:

1. scan input, decide the output length, increase buffer size if necessary
2. process the input in two iterations, one for length 0/1/2, one for length greater than 2.

**Q3. Use recursion to return values needed in a bottom-up way in binary tree**

**Q3.1** Determine whether a binary tree is a balanced binary tree



public int **getHeight** (TreeNode root) {

if ( root == null ) { // base case

return 0;

}

return Math.max(**getHeight**(root.left), **getHeight**(root.right)) + 1;

}

public boolean isBalaced (TreeNode root) {

if (root ==null ) {

return true;

}

int diff = Math.abs(getHeight(root.left) - getHeight(root.right));

if (diff > 1) {

return false;

}

return isBalanced(root.left) && isBalanced(root.right);

}

Time complexity = **O(nlogn) // Taught in Class 3**

root (n)

/ \

lchild (n/2 for get height) rchild (n/2 for get height) → O(n) on this level

/ \ / \

n/4 n/4 n/4 n/4 for get height → O(n) on this level

log(n) level －> O(nlogn)

**Q3.2 Improve the solution to be O(N)?**

public boolean isBalanced(TreeNode root) {

return getHeight(root) != -1;

}

public int **getHeight**(TreeNode root) {

if (root == null) {

return 0;

}

int left = **getHeight**(root.left);

int right = **getHeight**(root.right);

if (left == -1 || right == -1 || Math.abs(left - right) > 1) {

return -1;

}

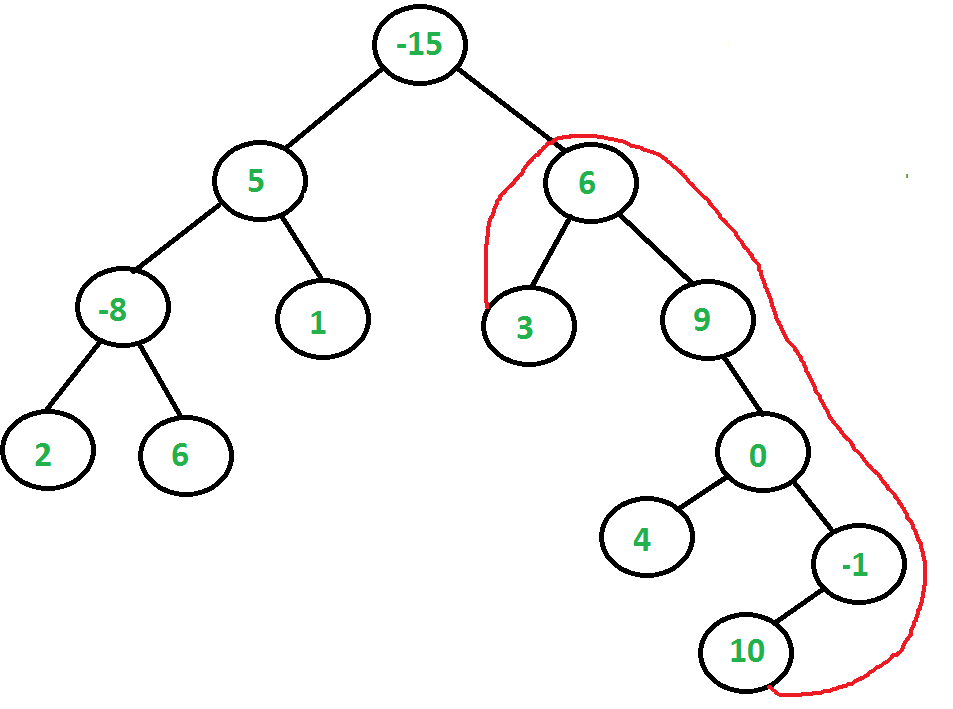
return Math.max(left, right) + 1;

}

**Q3.3 Midterm 2 question 2 (重复强调，简要复习)**

Given a binary tree in which each node element contains a number. Find the maximum possible sum of all nodes **from one leaf node to another**.

The maximum sum path may or may not go through root. For example, in the following binary tree, the maximum sum is 27(3 + 6 + 9 + 0 – 1 + 10). Expected time complexity is O(n).



00 int maxPathSum(struct Node **\*root,** int &**result**) {

01 if (root == NULL) return 0; // Base case

02 // Find maximum sum in left and right subtree. Also find

03 // maximum root to leaf sums in left and right subtrees

04 // and store them in lcost and rcost

05 int lcost = **maxPathSum**(root->left, result);

06 int rcost = **maxPathSum**(root->right, result);

07 // Find the maximum path sum passing through root

08 int curr\_sum = lcost + rcost + root->data;

09 // Update res (or result) only when needed

10 if (**result** < curr\_sum && **root->left && root->right)** { //eg, 9 is not leaf to leaf

11 result = curr\_sum;

12 }

13 // Return the maximum (root to leaf path) cost

14 if (root->left == null){

15 return rcost + root->val;

16 } else if (root->right == null) {

17 return lcost + root->val;

18 ｝

19 return **max(lcost, rcost)** + root->data;

20 }

21 // The main function which returns sum of the maximum

22 // sum path between two leaves. This function mainly uses

23 // maxPathSum()

24 int **FindMaxPathCost**(Node \*root) {

25 int result = 0;

26 **maxPathSum**(root, result);

27 return result;

28 }

public int maxPathSum(TreeNode root, int[] result) {

if (root == null) {

return 0;

}

//这里的关键是先不找叶子到叶子，先找根到叶子的最大Path Sum

//这个函数是用来找根到叶子的最大Sum的，不是我们想要的

//我们想要的不是从这个函数返回值拿到，而是从result里，每次要update result

int lcost = maxPathSum(root.left, result);

int rcost = maxPathSum(root.right, result);

//左右孩子分别返回了从左右孩子为根到下面叶子节点最大的Sum值

int curr\_sum = lcost + rcost + root.key;

//再加上根本身的值，就构成了在当前层通过root从一个叶子到另一个叶子最大的Sum

if (curr\_sum > result[0] && root.left != null && root.right != null) {

result[0] = curr\_sum;

}

if (root.left == null) {

return rcost + root.key;

} else if (root.right == null) {

return lcost + root.key;

}

//更新result的值，如果变得更大

return Math.max(lcost, rcost) + root.key;

//向上一层返回本身值加左右max sum里面较大的一个

}

**Q4. Longest common substring/subsequence between two strings.**

**Q4.1 Longest common substring （solution 中字母必须连续）**

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

**A[] = sweden;**

**B[] = student;**

**First: Primitive idea**:

s w e d e n size = n

s t u d e n t size = m

1. for sweden, how many substrings are there????

O(n^2) substrings in sweden

(2) for each substring of sweden, we check whether this substring is in student. if YES, we check whether it is the longest so far.

index = 0 1 2 3 4 5 6 7

s t u d e n t

swe**d**en 1 == 1

s

d

**d** e=2

**d e** n=2+1

repeated computation: for each letter as the last letter of the substring, when we increase the substring by one, we need to the repeated comparison for all its prefix

**i**

**B[] = student dfddf;**

**A[] = sweden d;**

**j**

**For all DP problems, we care about**

1. **Base case**:
2. **Induction rule (subproblem size(n-1) → size (n)**:

**Main idea:** M[i][j] represents: use the letter at A[i] as the last letter of A[] and use the letter at B[j] as the last letter of B[], what is the length of the common substring in this case (including A[i] and B[j]).

M[i][j] = M[i-1][j-1] + 1 (if a[i] == b[j])

= 0 otherwise

**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 0 0 0 0 0 0 0**

2 **w** **0 0 0 0 0 0 0 0**

3 **e** **0 0 0 0 0 0 0 0**

4 **d** **0 0 0 0 0 0 0 0**

5 **e** **0 0 0 0 0 0 0 0**

6 **n** **0 0 0 0 0 0 0 0**

**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

**Q4.2 Longest common sub-sequence (字母可不连续)**

**A == s**tu**dent**

**i**

**B == s**we**den**asy**t**

j

m[i][j]represents the length of the longest common subsquence of A[1...i], B[1...j]

**M[i][j] = M[i-1][j-1] + 1 if (A[i]== B[j]);**

**max(M[i-1][j], M[i][j-1], M[i-1][j-1])** otherwise

i-1 i

XX **A**X **B**X **C**

j-1 j

YY A YY BxxxxCE **D**

**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 0 0 0 0 0 0 0**

2 **w** **0 0 0 0 0 0 0 0**

3 **e** **0 0 0 0 0 0 0 0**

4 **d** **0 0 0 0 0 0 0 0**

5 **e** **0 0 0 0 0 0 0 0**

6 **n** **0 0 0 0 0 0 0 0**

7 **a** **0 0 0 0 0 0 0 0**

8 **s** **0 0 0 0 0 0 0 0**

9 **y** **0 0 0 0 0 0 0 0**

**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 2 3 3 3

6 **n** **0** 1 1 1 2 3 **4** 4

7 **a** **0** 1 1 1 2 3 **4** 4

8 **s** **0** **1** 1 1 2 **3** 4 4

9 **y** **0** 1 1 1 2 3 **4** 4

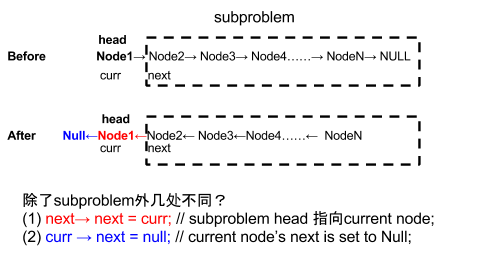
**ababc**

**abcab**

# Class 24 强化练习 4

**Q1. Reverse linkedlist questions**

**Q1.1 Reverse a singly linked list**



**Node {**

**Integer data;**

**Node next;**

**}**

public ListNode reverseRecursive(ListNode head) {

if (head == null || head.next == null) {

return head;

}

Node node1 = head;

Node node2 = head.next;

ListNode nodeN = reverseRecursive(node2);

node2.next = node1;

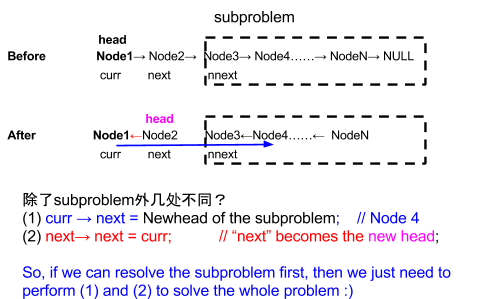
node1.next = null;

return nodeN;

}

* BASE CASE

**Q1.2 Reverse a linked list (pair by pair)**



public ListNode reverseRecursive(ListNode head) {

if (head == null || head.next == null) {

return head;

}

Node node1 = head;

Node node2 = head.next;

ListNode nodeN = reverseRecursive(node2.next);

node2.next = node1;

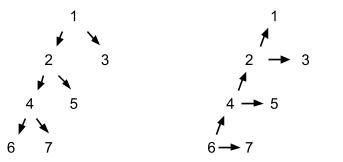
node1.next = nodeN;

return node2;

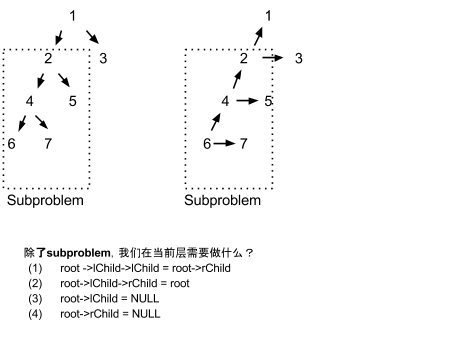
}

**Q1.3 Reverse a binary tree upside down**

Given a binary tree where all the right nodes are leaf nodes, flip it upside down and turn it into a tree with left leaf nodes. **For example**, turn these:



What do we need to do in each recursion level?



Node {

Integer data;

Node left, right;

}

**// does the right child node always exist?**

public Node **reverseRecursive**(Node head) {

if (head == null || head.left == null ) {

return head;

}

Node node2 = head.left;

Node node3 = head.right;

Node nodeN = **reverseRecursive**(node2);

node2.left = head.right;

node2.right=head;

head.left = null;

head.right = null;

return nodeN;

}

**DFS 基本方法：**

1. **what does it store on each level? (**每层代表什么意义？一般来讲解题之前就知道DFS要recurse多少层)
2. **How many different states should we try to put on this level? （**每层有多少个状态/case 需要try？**）**

**Q2.1 （之前课讲过）**Print all subset of a set

{a, b, c, d}

{},{a},{b},{c},{d},{ab},{ac},.... {a,b,c,d} 2^n-2

(pick/not pick)^n

Char[] elements;

Array<Char> solution; // the current state

public void search(int depth) {

if (depths == element.length) {

print(solution);

return;

}

// no pick

search(depth+1);

// recover previous state = do nothing

// pick

solution.append(elements[depths]);

search(depth+1);

// recover to previous state, but it doesn’t matter here

solution.pop\_back();

}

{d}

{c}

{c,d}

{b}

{b,d}

{b,c}

{b,c,d}

….

**Q2.3 （之前课讲过）**Print all valid combination of coins that can form a certain amount of money (99 cents)

1 5 10 25 cents

N

**1 what does it store on each level? (**每层代表什么意义？一般来讲解题之前就知道DFS要recurse多少层)

**2 How many different states should we try to put on this level? （**每层有多少个状态/case 需要try？**）**

combination(99, 0);

1. depth
2. solution
3. iterate
4. rember to recover the state if state is a global variable

ArrayList<int> solution;

int[] coins = {25,10,5,1};

public void combination (int money, int depth){

if(depth == 3){

solution.append(money);

// print solution

solution.remove(solution.length()-1);

return;

}

for(int i = 0; i <= money/coins[depth]; i++){

solution.append(i);

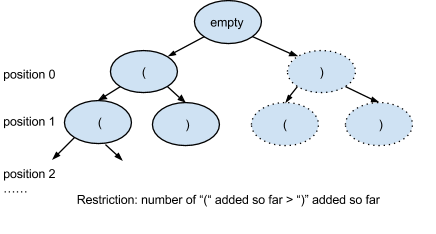
combination(money -coins[depth]\*i, depth + 1);

solution.remove(solution.length()-1);

}

}

**Q2.4 （之前课讲过）**Print all valid permutations of ()()() **(Class 6 : DFS 例题2)**



Given N pairs of ()

N=2

()() , (())

N=3

((())), ()()(), (())(), ()(())

1. depth -> 0..N-1 (each level is to decide to add (, ) )
2. solution is a string
3. base case? depths=N, print solution
4. restrictions durting iteration: count of “(“, count of “)”
5. remember to recover state (two counts)

public void printParentheses(int n) {

StringBuilder sb = new StringBuilder();

helper(n, n, sb);

}

private void helper (int left, int right, StringBuilder sb) {

// left is how many left buckect hasn’t been used

// right is how many right buckect hasn’t been used

if (left == 0 && right == 0) {

System.out.println(sb.toString());

return;

}

if (left>0) {

sb.append(‘(’);

helper(left-1,right,sb);

sb.deleteCharAt(sb.length() - 1);

}

if (right>0 && right > left) {

sb.append(‘)’);

helper(left, right-1, sb);

sb.deleteCharAt(sb.length()-1);

}

}

**Q2.5** Print all valid permutations of 3{ } 2[ ] 1( )

6 + 4 + 2

**DFS 基本方法：**

**1 what does it store on each level? (**每层代表什么意义？一般来讲解题之前就知道DFS要recurse多少层)

**6 + 4 + 2 == 12 levels**

2 **How many different states should we try to put on this level? （**每层有多少个状态/case 需要try？**）**

**6 states: ( ) [ ] { }**

**0 1 2 3 4 5**

**solu ( [ [ { } ] ] w**

**stack ( top**

depth -> 0..N-1 (each level is to decide to add (, ) )

1. solution is a string
2. base case? depths=N, print solution
3. restrictions during iteration: count of “(“, count of “)”
4. remember to recover state (two counts)

**Q2.6** Eight Queen

**0 1 0 0 0 0 0 0**

**0 0 0 1 0 0 0 0**

0 0 0 x0 0 0 0

0 0 0 1 0 0 0 0

0 1 0 0 1 0 0 0

0 0 0 1 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0

1. depth = one depth for each queen on each rows
2. solution is an array of size 8, each element is from 0..7
3. depth = 8, print out solution
4. restriction during iteration: (x1-x2)==(y1-y2), (x1=x2)=(y2-y1), or y1==y2
5. rememer to recover state (clean the occupence bit for columns and diagonal lines)

int[] solution = new int[8];

int[] column = new int[8], leftx = new int[15], rightx = new int[16];

// column[i] = 1 if that column is occupied, same for leftx and rightx

public void eight\_queen(int d) {

if (d == 8) {

// print solution;

return;

}

for (int i = 0; i < 8; ++i) {

if (!(column[i] || leftx[d+i] || rightx[8+d-i])) {

column[i] = leftx[d+i] = rightx[8+d-i] = 1; // some global state

solution[d] = i;

eight\_queen(d+1);

column[i] = leftx[d+i] = rightx[8+d-i] = 0;

}

}

}

**Q3: 2,3,4-SUM questions**

**Q3.1 2Sum**

Given an **unsorted/sorted** array, how to find two numbers from it that sum up to a target number x;

1. **unsorted** : use a hash\_table (be careful about the corner case e.g. 5+5 == 10)

A[i] == 3 the other element should be 7 target == 10

1. **sorted** array
   1. Binary search iterate over the whole array, A[i] == 3, we use binary search to find whether 7 is there or not. **O(nlog(n))**

**i j**

Example: A[] = { 1, 3, 5, 7, 9, …. 100 } target = 10, then return <1, 9>

// naive solution: try all possible pairs of i, j

for (i = 0; i < n; i++) {

for (j = i + 1; j < n; j++) {

A[i] + A[j] ??? target

}

}

**Use two indices:**

i = 0; j = n-1;

while (a[i] + a[j] != target && i<j) {

if (A[i] + A[j] > target), j--;

else if (A[i] + A[j] < target) i++;

}

O(n)

for (i = 0)

for (j = 0)

target = 5;

1 2 3 4 5 6 7 8 9

i j

O(n)

**Q3.2 3-SUM**: Find three numbers sum up to x (sorted)

Method: a + b + c == x;

put a to a fixed value, and the find a pair of b and c summed up to x - a

O(n^2)

**Q3.3 4-SUM**: Find three numbers sum up to x (sorted)

For 4 numbers: we permute all possible sums of a pair of numbers in the array

**i j**

Example: A[] = { 1, 3, 5, 7, 9, …. 100 } target = 10,

a + b + c + d == x

for (i

for (j {

get all possible sums formed by a pair of elements in the original array

}

Step1: use for for loop to store all possible sums formed by a pair of elements in the original array SUM[N^2].

Define each element to be < int **sum\_value**; int a\_index; int b\_index >

Step 2: sort the array SUM[N^2] → O(N^2 log(N^2)) → O(N^2 log(N));

If we use a hash\_table to store SUM[N^2] → **O(N^2) solution is here.**

Step 3:

Assume we have 1st pair <1, 3> == 4

2nd pair <1, 5> == 6

Sum of 1st and 2nd pair == 10

If we do not use a hash\_table in step 2, then **O(N^2 log(N));**

**Q4:** 石子归并：

设有N堆沙子排成一排，其编号为1,2,3,…,N(N<=100)。每堆沙子有一定的数量。现要将N堆沙子并成为一堆。归并的过程只能每次将相邻的两堆沙子堆成一堆（每次合并花费的代价为当前两堆沙子的总数量），这样经过N-1次归并后成为一堆，归并的总代价为每次合并花费的代价和。找出一种合理的归并方法，使总的代价最小。

例如：有3堆沙子，数量分别为13,7,8，有两种合并方案，

第一种方案：先合并1,2号堆，合并后的新堆沙子数量为20，本次合并代价为20，再拿新堆与第3堆沙子合并，合并后的沙子数量为28，本次合并代价为28，将3堆沙子合并到一起的总代价为第一次合并代价20加上第二次合并代价28，即48；

第二种方案：先合并2,3号堆，合并后的新堆沙子数量为15，本次合并代价为15，再拿新堆与第1堆沙子合并，合并后的沙子数量为28，本次合并代价为28，将3堆沙子合并到一起的总代价为第一次合并代价15加上第二次合并代价28，即43；

采用第二种方案可取得最小总代价，值为43。

**2D dynamic programming**

**Solution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **index** | **0** | **1** | **2** | **3** |
| **sand value** | 13 | 7 | 8 | 3 |

**M[i][j]** the optimal solution for merging the i-th sand all the way to the j-th sand altogether.

**M[i][j]**  = **min** ( M[i][i] + M[i+1][j] // case 1

M[i][i+1] + M[i+2][j] // caes 2

…… ...

M[i][j-1] + M[j][j]); // case j-1

* **SUM[i][j]**

**SUM[i][j]** stores the Sum of (A[i]..... A[j]) (i <= j)

j = 0 1 2 3

i=0 **13** 20 28 31 → fill in the SUM[i][j] from left to right

i=1 x  **7** 15 18

i=2 x x **8** 11

i=3 x x x  **3**

===============================================================

1. **fill in the following M[i][j] form bottom up**
2. **fill in the following M[i][j] form from left-hand side to the right**

j = 0 1 2 3

i=0 0 20 x x

i=1 x 0 15 31 = M[1][3] = min(M[1][1]+ M[2][3], M[1][2], M[3][3])+ SUM[1][3]

i=2 x x 0 11 min( 11 , 15 ) + **18**

i=3 x x x 0

# Class 25 强化练习 (Recursion 总结)

**Review:** What is recursion (again)?

**1. Recursion 与计算的结合**

**Q1.1** a^b

2^1000 2^500 2^250 2^ 125 ….

**Q1.2** Given a helper function **pow(a,b)** that can compute a^b, how to calculate x^x^x^x^x…^x (total number of x == n, n > 0)

2^2^ 2^2 → 4^4

int **calculate**(int x, int n) {

if(n ==1) {

return x;

}

int temp = calculate(x, n / 2);

if(n % 2 == 0) {

return pow(temp, temp);

} else {

return pow( **pow(temp, temp)**, x);

}

}

**2. Recursion 与1D or 2D Array的结合**

1. **1D array**: 二分法比较常见
   1. MergeSort , QSort
2. **2D array**:
   1. 逐层(row by row)递归: 8 queen
   2. 剥洋葱递归: Spiral Print 2D array

1 2 3 4 5

16 17 18 19 6

15 24 **25**  20 7

14 23 22 21 8

13 12 11 10 9

**void sprialprint**(**int[][] a**, int **offset** =0, int **size**, int **counter**) {

if (size <= 1)

print matrix // base case

for(i = 0; i < size-1; i++) { // size == 5; upper row

a[0+**offset**][0+**offset** + i] = counter++; // offset is the [x] and [y] cooridnates of the

upper-left cornner of the box

}

for(i = 0; i< size; i++) // right column

for // bottom row (from right to left)

for // left column

//recursive rule

**spiralprint**(a, offset + 1, size - 2, counter);

}

**3. Recursion 与LinkedList 的结合**

**Q3.1** Reverse a LinkedList (Class 3 review)

public void ListNode reverseRecursive(ListNode head) {

if (head == null || head.next == null) {

return head; // base case;

}

ListNode n1 = head;

ListNode n2 = head.next;

ListNode newHead = reverseRecursive(n2);

n2.next = n1;

n1.next = null;

return newHead;

}

**4. Recursion 与String的结合**

**Q0 recurse a string using recursion**

**abc d → cba d → d cba O(n^2)**

**output → dcba**

void reverse(string& input, int left , int right);

**Q1. A** word such as “book” can be abbreviated to 4, 1o1k, b3, b2k, etc. Given a string and an abbreviation, return if the string matches the abbreviation. Assume the original string only contains alphabetic characters. For example: “s**11**d” matches “s**ophisticate**d”.

book

i

b1o1k

j

**Case1** : if both are chars Match(string1, int index1, string2, index2)

**Case2** : if abbreviation word[index2] is not a char: keep reading all the digits out, and convert them to a integer value x;

sophisticated s11d

Match(string1, index1+x, string2, index2 + num of digits)

Abbreviation matching

// s1: student original string

// s2: s2d2t pattern

bool isDigit(char input) {

return input >= '0' && input <= '9';

}

00 bool **AbbrevMatch**(string s1, string s2) {

01 if (s1.size() == 0 && s2.size() == 0) {

02 return true; // base case;

03 } else if (s1.size() == 0 || s2.size() == 0) {

04 return false;

05 }

06 if (isDigit(s2.at(0))) { // Case1: s2.at(0) is a digit

07 int i = 0;

08 int num = 0;

09 while (**i < s2.size()** && isDigit(s2.at(i))) {

10 num = num\*10 + s2.at(i) - '0';

11 i++;

12 }

13 if (num > s1.size()) {

14 return false;

15 } else {

16 return **AbbrevMatch**(s1.substr(num), s2.substr(i));

17 }

18 } else { // Case2: s2.at(0) is **not** a digit

19 if (s1.at(0) != s2.at(0)) {

20 return false;

21 } else {

22 return **AbbrevMatch**(s1.substr(1), s2.substr(1));

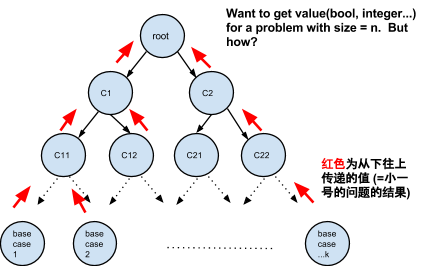
23 }

24 }

25 }

**5. Recursion 与 Tree的结合**

* Binary tree 往往是最常见的，和recursion 结合最紧密的面试题目类型。
* Reasons:
  + 每层的node 具备的性质，传递的值和下一层的性质 往往一致。比较容易定义**recursive rule**.
  + **Base case** (generally): null pointer under the leaf node
  + Example1: **int getHeight(Node root)**
  + Example2: 统计tree里边有多少个node？



**5.1 Tree + Recursion 第一类问题：从下往上返值 (int, bool, etc.)**

**Q5.1.1 (从下往上返值)** Review

int **getHeight**(Node root) {

if (root == null) {

return 0;

}

return Math.max(**getHeight**(root.left),

**getHeight**(root.right))+1;

}

最简单的当然就是这类题，题目本身就定义了递归规则，而且是和根相关的，直接

递归就可以了。

**Q5.1.2 (从下往上返值)** How to store how many nodes in each node’s left-subtree?

4(3)

/ \

2(1) 5(0)

/ \ \

1(0) 3(0) 6(0)

class TreeNode {

int value;

TreeNode left;

TreeNode right;

int **leftChildrenNum**;

}

**Way of thinking (Tricks)**

**1. What do you expect from your lchild / rchild? → (1) return type**

Total number of nodes in my left subtree **(1)**

Total number of nodes in my right subtree **(2)**

2. What do you want to do in the current layer? **(2) → if we want to maintain a global max/min value, we will have a global variable in the input argument list)**

store (1) in **current->lChildNum;**

**3. What do you want to report to your parent? (same as Q1 == Q3)**

4(x??? ← 3)

/ \

2(1) 5(0)

/ \ \

1(0) 3(0) 6(0)

这道题也是的，递归法则不能是左子树里的结点个数，因为和根无关，无法把规则传递

下去。最上一层（根那里）是要求左子树里节点个数，可是到了下一层就左右子树都要

考虑。所以只能定义递归法则为某根节点下面左右子树加起来的节点数目，在递归过程中

用一个变量存下当前结点左子树里节点个数。这样随着递归完毕，一直update这个值就好了

**int** **FindLeftChildenTotal(**TreeNode root) {

if (root == null) {

return 0; // base case

}

// what do you expect from your left/right child

int leftTotal = **FindLeftChildenTotal(**root.left);

int rightTotal = **FindLeftChildenTotal**(root.right);

// what we do in the current layer

**// root.leftChildrenNum = leftTotal;**

**root.rightChildrenNum = rightTotal;**

// what should we report to my current node’s parent?

**return leftTotal + rightTotal + 1;**

}

**Q5.1.3 (从下往上返值)** Find the node with the max difference in the total number descendents in its left subtree and right subtree

这个题也是一样，不能直接定义递归法则，因为和根节点无关。所以又是先定义递归法则

为某结点下的左右子树结点个数总和。这里递归是一个“不得不做”的事情，更重要的事情

是在偷偷跟着进行的。我们并不需要子树结点总和这个东西，可是为了实现目标不得不用它

来做迭代。递归的时候不断update左右子树结点个数的差。递归结束就找到最大差值了

int max = 0;

**int** **FindMaxDiff**(TreeNode root, TreeNode result, int max) {

if (root == null) {

return 0; // base case

}

// what do you expect from your left/right child

int leftTotal = **FindMaxDiff(**root.left);

int rightTotal = **FindMaxDiff**(root.right);

// what we do in the current layer

int diff = Math.abs(leftTotal - rightTotal);

if (diff > max) {

max = diff;

result = root;

}

// what should we report to my current node’s parent?

**return leftTotal + rightTotal + 1;**

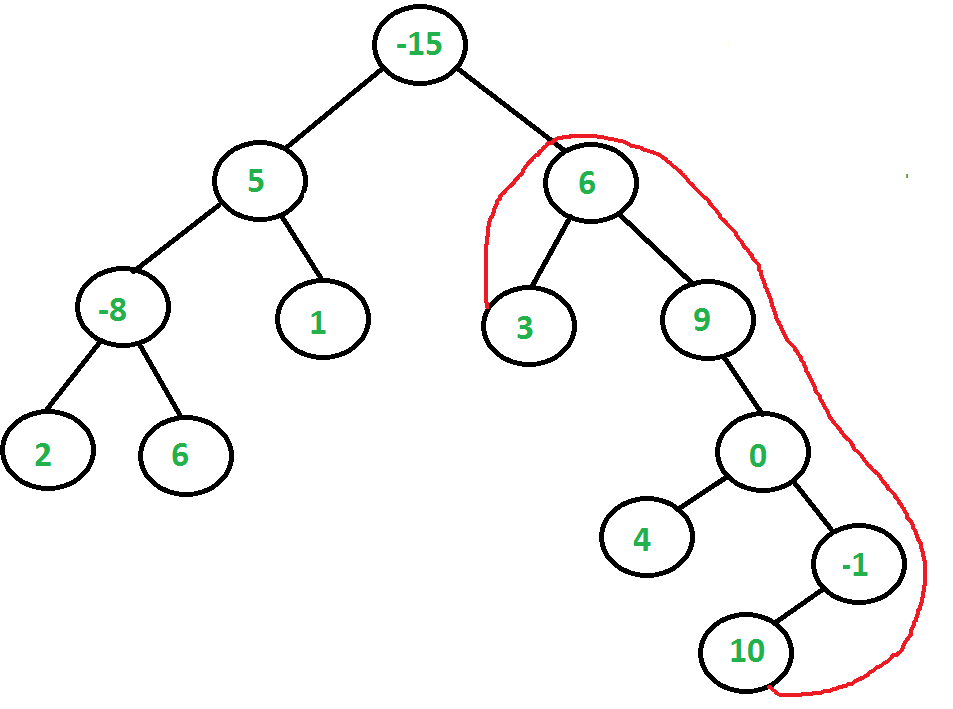
}

**Q5.1.4 (再次复习， 人字形path 问题）**

**Laicode.com Class 20 (Maximum Path Sum Binary Tree I)**

Given a binary tree in which each node element contains a number. Find the maximum possible sum **from one leaf node to another**.

The maximum sum path may or may not go through root. For ex ample, in the following binary tree, the maximum sum is 27(3 + 6 + 9 + 0 – 1 + 10). Expected time complexity is O(n).



**Way of thinking (Tricks)**

**1. What do you expect from your lchild / rchild? → (1) return type**

Total number of nodes in my left subtree **(1)**

Total number of nodes in my right subtree **(2)**

2. What do you want to do in the current layer? (**(2) → if we want to maintain a global max/min value, we will have a global variable in the input argument list)**

store (1) in **current->lChildNum;**

**3. What do you want to report to your parent? (same as Q1 == Q3)**

要用树能递归下去，返回的东西必然要和根有关。一切都要从根开始想，怎么能

迭代下去。如果题目本身要问的不涉及根，那么只能间接解决这个问题，因为不涉及

根的无法递归下去，所以通常都是用和根有关的一个递归规则partially解决这个问题，

然后想办法update一个真正想要的东西，在递归的过程中顺便解决我们实际希望得到

的东西。这个题就是一个典型的这样的题目，findMax并没有直接找到从一个叶子到另

一个叶子的最大Sum，而是从当前根到另一个叶子的最大Sum。我们真正需要的被存在

了max这个数组里，随着递归不断被update。

所以这里第1,3步从“下层”得到什么以及向“上级”回报什么和递归有关，属于

“例行公事”，真正想要的“私事”是在第2步当前层“偷偷执行的”。。。

**int** **findMax** (TreeNode root, int[] max) {

if (root == null) {

return 0;

}

int leftSum = findMax(root.left, max);

int rightSum = findMax(root.right, max);

// 2. what we do in the current layer

int tmp = leftSum + rightSum + root.val;

if (tmp > max[0] && root.left != null && root.right != null) {

max[0] = tmp;

}

// 3. what do we want to report to my parent ???

if (root.left == null) {

return rightSum +root.val;

}

if (root.right == null) {

return leftSum + root.val;

}

return Math.max(leftSum, rightSum) + root.val;

}

**Q5.1.6 (人字形path 问题）**

**Laicode.com Class 20 (Maximum Path Sum Binary Tree II)**

Get Maxim sum of the path cost from **any node to any node** (not necessarily the leaf to leaf)

**Solution:**

00 int **helper**(TreeNode\* root, int& max) {

01 if (root == NULL) {

02 return 0;

03 }

04 int left = **helper**(root->left, max);

05 int right = **helper**(root->right, max);

06 left = left < 0 ? 0 : left;

07 right = right < 0 ? 0 : right;

08 max = max > (root->value + left + right) ? max : (root->value + left + right);

09 return left > right ? (left + root->value) : (right + root->value);

10 }

**5.2 Tree + Recursion 第二类问题：**

Carry a **直上直下** path prefix (非人字形) while traversing the tree:

**Discussion:**

Note that: Tree 相关问题，路径种类可以分为两大类

**Class 1**: 人字形path, 这类题一般需要从下往上传integer value (E.g.,**Q5.1.5 above**)

**Class 2**：从root 往下 （**直上直下**）path

**10**

/ \

**5** 15

/ \

2  **7**

\

**8 [end]**  current node

path\_prefix = {10 5 7 8 }

←

8

78

578

10 5 7 8

**Q 5.2.1** (laicode.com Class 20) **Binary Tree Path Sum To Target**

Given a binary tree in which each node contains an integer number. Determine if there exists a path from any node to any node (the two nodes can be the same node and they can only be on the path from root to one of the leaf nodes)**,** the sum of the numbers on the path is the given target number.

**Examples**

5

/ \

2 11

/ \

6 14

If target = 17, There exists a path 11 + 6, the sum of the path is target,

If target = 10, There does not exist any paths sum of which is target.

carry a prefix of the path from root to the current node while traversing the binary tree.

**10**

/ \

**5**  15

/ \

2  **7**

\

**8 [end]**  current node

int solve(){

vector<int> res;

int max = 0;

helper(root, res, max);

return max;

}

// extend to the original problem **the largest path cost** from any node to any node (along the path from root to a leaf node).

**res[] = {10 5 7 8} current value == 8**

void helper(TreeNode\* root **= 5**, vector<int> &res, int &max){

if(root == NULL) return;

**res.push\_back(root->value);**

int sub\_sum = 0;

for(int i=res.size()-1; i>=0; i--){

sub\_sum += res[i];

if(sub\_sum > max)

max = sub\_sum;

}

helper(root->left, res, max);

// **res.pop\_back(root\_value);**

// **res.push\_back(root->value);**

helper(root->right, res, max);

**res.pop\_back(root\_value)**;

return;

}

**5.3 Tree + Recursion 第三类问题 (Advanced Topic)：**

利用某些特种树(e.g., complete tree, BST)的性质，由一个1D 数组的数据，构建一个binary tree / BST

**Q5.3.1** Re-construct a complete binary tree from its level-order traversal sequence only.

10

5 15

2 7 12 20

1

**e.g., level\_order[] = { 10 5 15 2 7 12 20 1 }**

current node’s index = i;

left child index = 2\*i +1;

rigtt child index = 2\*i +2;

public TreeNode **construct** (int[] level\_order, int index) {

if (index >= **level\_order**.length)

return null;

TreeNode root = new TreeNode(**level\_order**[index]);

root.left = **construct** (**level\_order**, index\*2+1);

root.right = **construct** (**level\_order**, index\*2+2);

return root;

}

**Q5.3.2:** Re-construct the BST, given only the **preorder** traversal sequence.

**Method 1 ( O(n^2) time complexity)**

The first element of preorder traversal is always root. We first construct the root. Then we find the index of first element which is greater than root. Let the index be ‘i’. The values between root and ‘i’ will be part of left subtree, and the values between ‘i+1′ and ‘n-1′ will be part of right subtree. Divide given pre[] at index “i” and recur for left and right sub-trees.

For example in {10, 5, 1, 7, 15, 20}, 10 is the first element, so we make it root. Now we look for the first element greater than 10, we find 40. So we know the structure of BST is as following.

**10**, 5, 1, 7, 15, 20

10  
 / \  
 / \  
 5 15

/ \ / \

{1} {7} {-} {20}

**TreeNode {**

**int value;**

**vector<Node\*> children;**

**}**

# Class 26 强化练习5

**Q1 xxxxx type**

**Q1.1** 一个字典有给一系列strings，要求找两个string,使得它们没有共同字符，并且**长度乘积 最大**. (Assumption: all letters in the word is from ‘a-z’ in ASCII)

**Example:**

w1 abcde size = 5

w2 adzz size = 4

w3 abd size = 3

w4 fgz size = 3;

solution: abcde x fgz = 5 x 3 == 15

**Best First Search (BFS2):**

1. **Initial state (start state)**
2. **Expansion/generation**
3. **Termination condition**
4. **Deduplication (for the same state when generated)**

1.Initial state: **<w1, w2>**

2.Expansion/Generation rule <W1, W2> → <W2, W2>, <W1, W3>.

state<wi, wj> → <wi+1, wj> or <wi, wj+1>. be careful about the caes like **<wi, wi>**

3**.Termination condition**: whenever a state <wi, wj> is popped out of the max\_heap, and wi and wj do not share any common chars, then terminate.

**<w2 w3> !=???? <w3 w2>**

**Q1.2** How to find the **k-th smallest number** in the f(x,y,z) = 3^x \* 5^y \* 7^z **(int x > 0, y>0, z>0).**

1.Initial state <x=1, y=1, z=1>

2.Expansion/Generation rule: <i, j, k > → <i+1, j k> <i, j+1, k> <i, j , k+1>

3.Termination condition: when the k-th state is popped out of the p\_queue, then terminate and return the state.

4.**Deduplication (for the same state when generated)**

we need to be careful. Maintain a hash\_map to store all the states that have been generated. (Avoid inserting the same state into the p\_queue for the 2nd time.)

**Q1.3** Given three arrays with numbers in ascending order. Pull one number from each array to form a coordinate <x,y,z> in a 3Dspace. (1) How to find the coordinates of the points that is k-th closest to <0,0,0>?

A1[m] = { 1, 3, 5, 7, 9, ,... } -> X

i

A2[n] = { 2, 3, 4, 6, 8, ….} -> Y

j

A3[L] = { 1, 3, 4, 5, 6, ...} -> Z

k

1.Initial state: <0, 0, 0>

2.Expansion/Generation rule: <i, j, k > → <i+1, j k> <i, j+1, k> <i, j , k+1>

3.Termination condition: when the k-th state is popped out of the p\_queue, then terminate and return the state.

4.**Deduplication (for the same state when generated)**

we need to be careful. Maintain a hash\_map to store all the states that have been generated. (Avoid inserting the same state into the p\_queue for the 2nd time.)

**Q1.4** Given a gym with k equipments, and some **obstacles**. Let’s say we bought a chair and wanted to put this chair into the gym such that the sum of the shortest path cost from the chair to the k equipments is minimal.

N X N Gym → N^2 states **O(k\* N^2 lgN) = I need to run k times Dijskstra’s algorithm**

**k equipments**

**class Cell {**

**int x;**

**int y;**

**vector<int> path\_cost;**

**}**

432 1/5/3=9**e1**xxxxxxx

xxxxxxx**e2**xxxx

xxxx**e3**xxxxxxx

35 33xxxxxxxxxx

Better solution: Run Dijkstra’s algorithm from each equipment e\_i,

Worse solution:

N^2 times of DIjstra O(N^2 \* **N^2 lgN)**

1. **best first search is the only general algorithm to find the shortest path in any arbitrary graph.**
2. **Special case, in 4-connected grids with edge cost all equal to one, then you can use BFS (Breath-first search) to find a shortest path (special case).**

**Q2 (Design)** Given a single computer with a single CPU and a single core, which has 2GB of memory and **1GB available** for use, it also has two 100GB hard drives.

How to sort 80GB integers of 64 bits.?

设计题，先问interviewer，搞清楚模糊的点：

1. **objective**: system是干什么的
2. **functionality**： 具体实现什么功能
3. **scalability**: 要处理多大的问题size, 有多少机器/cluster

**Assumption 1**: Let’s assume that all data is stored in one hard drive, with the other one totally empty.

**Assumption 2:**  after sorting, we store the data into one hard drive only

**Assumption 3:** **Ascending** order or Descending?

**Assumption 4**: what is the data of the range. any value is possible.

160 chunks 0.5GB / Chunk Can we use **QSort**()

**Next step**: for each chunk, there are n elements.

1

2 12

1234 (=2GB)

3

4 34

12345678

5

6 56

5678

7

8 78

How many levels of merging  **log(160) \* 80GB**

**160-Way merge**

1

2

3

4 **MIN\_HEAP** → solution:  **80GB \* log(160)**

5

…

160

**Q3** (string conversion)

**Q3.1** “A1B2 C3D4” ==> “ABCD1234”

A1 B2

A 1 B 2

A1 B2

A1

i

B2

j

AB12

**Q3.2 “**ABCD1234” ==> “A1B2C3D4”

I love yahoo 变种

AB | CD | 12 | 34

DC 21

12 CD

AB12 | CD34

log(n) times to call recursion function, and O(n) for each call

so total O(nlogn)

**Critical details**: guarantee size of Chunk 1 == Chunk 3;

**关键问题**： 注意n/2 == 奇数情况

index = 0 1 2 3 4 5 6 | 7 8 9 0 1 2 3

A B C **D E F G** | **1 2 3** 4 5 6 7

lm m rm

size = 14

mid = left + size/2 = 7

leftmid = ¼ \* size = 3

rightmid = left+ ¾ size = 10

00 void **convert**(char a[], int left, int right) {

01 if (right - left <= 1)

02 return;

03 int size = right - left + 1;

04 int mid = left + size/2;

05 int leftmid = left + size/4;

06 int rightmid = left + size \* 3/4;

07 Reverse(a, leftmid, mid-1);

08 Reverse(a, mid, rightmid-1);

09 Reverse(a, leftmid, rightmid-1); // DE123 -> 123DE

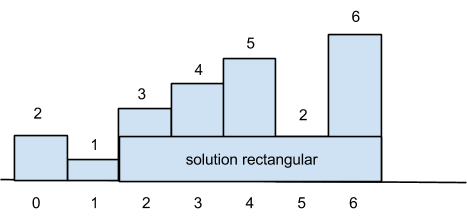
10 **convert**(a, left, left+2\*(leftmid-left)-1);

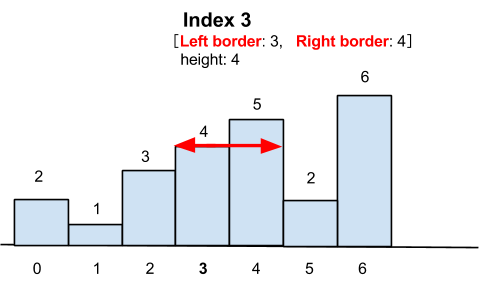
11 **convert**(a, left+2\*(leftmid-left), right);

12 }

**Q4 Histogram questions** (直方图问题)

**Q4 .1**直方图中找最大矩形





**Use a stack to store all the indices of the columns that form an ascending order**

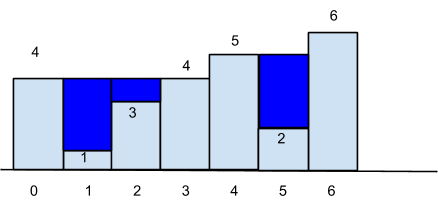
**stack that stores the indices in ascending order Bottom|| [1, 2, 3, 4,**

**When scanning the element with index = 5, M[5] == 2 < M[4] == 5, so we keep checking left column of index 5, and calculate the area of index 4, 3, 2, and pop them out of the stack, after this step , the stack is Bottom||[1, 5**

**Principle: to maintain the stack to make sure the columns whose indices are stored in the stack form an ascending order. (细节：When popped an element out of the stack, the element’s right border == the current index - 1, the left border of the element = the index of the element on top of the statck + 1);**

**O(n)**

**Q4 .2**直方图下雨接水问题



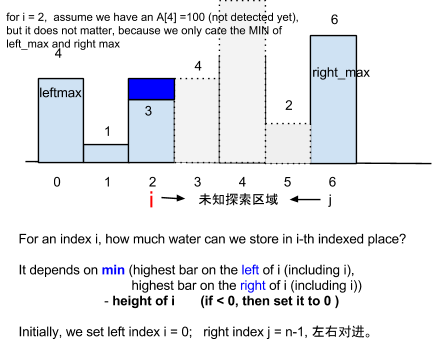
How to find the global\_max\_left and global\_max right for each index i????

Naive solution: 中心开花 → O(n^2)

Better solution: O(2n) → O(n)

1. From left handside to right keep the global\_max\_left with O(n)
   1. M1[i] = {4, 4, 4, 4, 5, 5, 6}
2. From righthandside to left, keep the global\_max\_right with O(n)
   1. M2[i] ={6 6 6 6 6 6 6}

Optimal solution: O(1\* n)



**Example: left\_max [ i, …….. j ] right\_max**

Step 1: i = 0, j = n-1 min(A[0], A[6]) = min(3, 6) = 3, 最短板在 i = 0, so

water height at index i == min(left\_max, right\_max) - A[i] = 3 - 3 = 0;

i++; //**left\_max** vs **right\_max**  **哪边小就移动哪边**；why?

**木桶理论：盯住最短板，谁小移动谁**

1. **if** left\_max 小就移动i (i++) ，
2. **else** 移动 j (j--)

**induction 推理 （proof）， 从i → i+1 分析问题**

**case 1: if left\_max < right\_max,**  we already know the water stores in i+1 can be calculated safely. so i++ ;

Why?

Case 1.1: **A[i+1] <= left\_max**, then left\_max is not updated, left border is still valid and does not change, so water at i+1 == left\_max - A[i+1];

Case 1.2:  **A[i+1] > left\_max** then left max is updated to A[i+1], water at i+1 == new\_left\_max - A[i+1] = A[i+1] - A[i+1] = 0;

**case 2: else, j --;**

**Q4.3.** (Discuss) Skype Line

Given n houses on the ground with each house represented by a rectangle. The i-th rectangle is represented as **[start\_i, end\_i, height\_i]**, where 0 <= i < n. The rectangles may overlap with each other. How can we calculate the total area that these rectangles cover.

Example: input = {<1,3,1>, <2,4,2>}, output = 5.

# Class 27 强化练习 6

**Q1. Binary Search Related Problems**:

还有好多变种，用到binary search 的各种variant. e.g.,

1) two sorted integer arrays, how to find the **k-th** smallest element from them.

2) two sorted integer arrays, how to find the **median** of the two arrays. （和1是一个题）

Example Input:

A[] = {2, 5, 7, 10, 13}

B[] = {1, 3, 4, 13, 20, 29}

k = 5

**Output**: 5

**M1: Primitive method:**

A[] = {2, 5, 7, 10, 13}

i

B[] = {1, 3, 4, 13, 20, 29}

j

**谁小移谁** → O(k)

**M2: Better method O(log(k))**

**(High Level) 核心思想是什么：** 把A[N]和前B[M]的各自前k/2比较，以每次删除k/2。

**(Details) How to delete k/2 ？** A[k/2 - 1] 和 B[k/2 - 1]**谁小就删谁**的前k/2

**(Proof) Why is it correct?**  Because result (=k-th smallest) cannot be among A[0] -- A[k/2 -1]

**反证法：** Assume the k-th smallest element is in A[0]...A[k/2-1], such as A[3] in our example, but we remove A[0]...A[k/2-1].

**index 0 1 2 3 = j 4 … k/2 -1**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A[N]** | **1** | **3** | **5** | **7=k-th** | **9...** |  | A[k/2-1] 小 | **...** |
| **B[M]** | **2** | **4** | **6** | **8** |  |  | B[k/2-1] 大 | **...** |

1. The fact is there are total **k/2 + k/2 == k** elements from A[0] to A[k/2 -1] plus B[0] to B[k/2 -1].
2. **Assume** that A[k/2 - 1] < B[k/2 -1], and there is an element A[j] **(0<= j <= k/2-1)** is the k-th smallest element in. E.g., **A[3] is the k-th smallest**.
3. Since A[3] <= A[k/2-1] < B[k/2-1], then **所有比k-th element小的数字，都只可能在红色或者蓝色区域， 但两个区域加起来所有元素的和也没有k个 (和(1)矛盾)**

**// How to find the kth smallest element from two sorted arrays.**

**// a\_left is the A[]’s left border to consider from**

**// b\_left is the B[]’s left border to consider from**

00 int **findKthSmall**(vector<int> a, int **a\_left**,

vector<int> b, int **b\_left**, int k) {

01 if (k == 1)

02 return **min**(a[a\_left], b[b\_left]); // base case 1

03 if(a\_left >= a.size())

04 return b[b\_left + k - 1]; // base case2: if nothing left in a;

05 if(b\_left >= b.size())

06 return a[a\_left + k - 1]; // base case3: if nothing left in b;

// Since index is from 0, so the k/2-the element should be = left +k/2 - 1

**// why is correct?** if a.size too small, then remove elements from b first.

07 int **a\_half\_kth** = a\_left + k/2 **- 1** < a.size() ? a[a\_left + k/2 - 1] : **imax**;

08 int **b\_half\_kth** = b\_left + k/2 **- 1** < b.size() ? b[b\_left + k/2 - 1] : imax;

09 if (**a\_half\_kth** **<** **b\_half\_kth**) {

10 return **findKthSmall**(a, a\_left + k/2, b, b\_left, k - k/2);

11 } else {

12 return **findKthSmall**(a, a\_left, b, b\_left + k/2, k - k/2);

13 }

14 }

Example 2

Example Line 07, e.g. **k = 10,** so **k/2 = 5**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A[2]** | **1** | **3** | **null** |  |  |  |  |  |  |
| **B[12]** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16= result** |  |

The result cannot be in the grey area, since the total size of A[2] is smaller than k/2 = 5. Thus, when A.size < k/2, we can simply remove the first k/2 of B[M],.

**Q2**  Given a number x, how to get the **hexadecimal** representation of the number in string type?

E.g  **4444429 ⇒ 0x1D**

**16 + 13**

10 进制 ＝>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

16 进制 => 0 1 2 3 4 5 6 7 8 9 A B C D E F

string **toHex**(int number){

string result = “”;

**while** (number != 0) {

int digit = number % 16

string char = array\_map[digit];

result = char + result;

number = number / 16;

}

return “0x” + result;

}

**Q3:**  (Array) Sliding window of size k, always return the **max element** in the window size.

**1 3 2 5** 8 9 4 7 3, WINDOW size k == 3

\_\_\_\_\_

3

\_\_\_\_\_

5

\_\_\_\_\_

8

array size: N, window size K

Naive method: for n for k

**O(nk)**  each time we ask for max\_element, O(k)

*Max\_Heap? if heap, how to remove the element out of window?*

**O(nlog(k))** difficult to implement.

Any better idea:

**1 3 2 5** 8 9 4 7 3,

\_\_\_\_\_

3 2

3 2 5

5 8

8 **9** 4 7 3

SOME data\_structure: ⇒ Deque 双向操作元素

*trick: when we add A[i] to the window, scan from right to the left of deque and remove all elements smaller than A[i], insert A[i] to the right side of deque.*

*Check the leftmost element of deque, if it is out of the size, remove it.*

*Rule: the deque maintains a* ***non-increasing*** *sequence.*

***Q4* Doubly LinkedList (DLL) + HashTable**

**Q4.1** How to design a **LRU** cache? (Least Recently Use)

**Cache content**

**latest** *Urlyyy →* **Url7** Url8 <->Url2< -> Url3<-->…7777.…-->NULL **oldest**

prev next

Array[N=1000] NONONO, because expensive to remove an element from an array

**Doubly LinkedList (DLL):** Yes O(1) move , delete….

*But only using doubly linked list cannot find the element in the cache efficiently O(n);*

*Therefore, we use a* ***hash\_map*** *<key = <a,b>, value = <Node’s reference in the DLL>>*

**template** <class T>

class LRU {

private:

map<string, T\*> table; // hashtable

int size; // the current number of elements in the cache

const int MAX\_CAPACITY; // the total size of the cache

public:

void **Insert**(T\* t) {

if(size >= MAX\_CAPACITY)

**Remove()**;

if(table.count(T->name) == 0) { // check whether cache missed or not

T\* nNode = new T(t);

table[t.name] = nNode; // <name, address> is inserted into hash\_table

nNode->next = head;

if(head)

head->prev = nNode;

head = nNode;1

if(size == 0) {

tail = nNode;

tail->prev = NULL;

tail->next = NULL;

}

}

size++;

return;

}

T\* **Get**(string& s) {

if(table.count(s) == 0) {

return NULL;

} else {

if(table[s] != head) {

T\* Prev = table[s]->prev;

T\* Next = table[s]->next;

table[s]->next = head;

head->prev = table[s];

head = table[s];

if(Prev)

Prev->next = Next;

if(Next)

Next->prev = Prev;

}

return(table[s]);

}

}

void **Remove**() {

// remove the last element in the linkedlist when inserting a new element

if(tail) {

tail = tail->prev;

delete tail->next;

if(tail->next)

tail->next = NULL;

size--;

if(size == 0)

head = NULL;

}

}

};

**Q4.2** Given a stream of characters, find the **first non-repeating** character from the stream. You need to tell the first non-repeating character in O(1) time at any moment.

**(Similar the use of LRU, which is implemented by Doubly Linked List + hash\_table)**

***a b c a c b***  *d e →*

*a a a b b NULL d*

*DLL : NULL*

*head*

*??????????????*

*Map<String, ListNode> name= new HashMap<String, ListNode>();*

*Hash\_table <****key*** *= char ,* ***value****= address of the char (if any) in the DDL, NULL otherwise >*

1. *<a, NULL>*
2. *<b, NULL>*
3. *<c, NULL>*
4. *<d, address>*

**Q5. 给一个integer array，允许duplicates，而且其中某个未知的integer的 duplicates的个数占了整个array的一大半( > 50%)。如何有效的找出这个integer？**

***Method1****:*

*Hash\_table <key= integer, value = counter(integer)>*

***Time = O(n)****, Space = O(n)*

*Space = O(1)*

***Method2****: Sort Time =O(nlogn), Space = O(1)*

*Return A[n/2]*

***Method3****:*

*O(1) : maintain a counter for the current candidate:*

*1 3 5 7 5 5 5 9 5*

*5 3 5 3 5 3 5 3 5*

*5(3)*

**Q6**  How to determine whether an array C[] can be merged by A[] and B[], while reserving the relative order of the letters in the original arrays A[] and B[]?

string A = “abcd”

string B = “acde”

string C = “acde abcd”

***Method1: DP***

***Base case: M[0][0] = true;***

***Induction Rule****: M[i][j]*

*The i+j-th letter can be from either A[i] or B[j]*

***M[i][j] represents*** *what: Whether the prefix of the C[], that is C[0...i+j] can be represented by A[i] and B[j].*

*M[i][j] =* ***M[i-1]M[j] && A[i] == C[i+j]***  *if the i+j-th letter is from A[]*

*||*  ***M[i]M[j-1] && B[j] == C[i+j]***  *else*

**A :aab B: ba**

**C= ababc**

**B= b a**

**index 0 1 2**

**A**

**0 T f f**

**1 a T T T**

**2 a F T T**

**3 b F T F**

***O(m\*n)***

***Method2: Recursion***

1. *Base case: When one string is empty, just check whether the other string’s suffix is identical to the C string’s suffix*
2. *Recursive rule:*
   1. *A[i], B[j] vs C[i+j]*

*Case1 A.substring(i+1) B.substring(j) C.substring(i+j+1]), when only A[i] == C[i+j]*

*Case2 A.substring(i) B.substring(j+1) C.substring(i+j+1]),, when only B[j] == C[i+j]*

*Case3, if both A[i] == C[i+j] && B[j] == C[i+j] A.substring(i+1) B.substring(j) C.substring(i+j+1]), || A.substring(i) B.substring(j+1) C.substring(i+j+1]),*

*Case4, return false;*

Method 1: recursion

00 bool **Combineable**(int index\_a, int index\_b, int index\_c ) {

01 if (index\_a == A.size() && index\_b == B.size()) {

02 return index\_c == C.size(); // base case

03 }

// case 3.

04 if (A[index\_a] == C[index\_c] && B[index\_b] == C[index\_c]) {

05 return **Combineable**(index\_a + 1, index\_b, index\_c +1) ||

06 **Combineable**(index\_a, index\_b + 1,index\_c + 1);

// case1.

07 } else if (index\_a < A.size() && A[index\_a] == C[index\_c]) {

08 return **Combineable**(index\_a + 1, index\_b, index\_c + 1);

// case2.

09 } else if (index\_b < B.size() && B[index\_b] == C[index\_c]) {

10 return **Combineable**(index\_a, index\_b + 1, index\_c + 1);

// case 4.

11 } else {

12 return false;

13 }

14 }

**Second way of writing code with recursion.**

bool result1 = false;

bool result2 = false;

if(index\_a < A.size()) && a[index\_a]== C[index\_c]{

result1 = Combineable(index\_a+1，index\_b,index\_c-1);

}

if(index\_B < B.size()) && b[index\_b]== C[index\_c]{

result2 = Combineable(index\_a+1，index\_b,index\_c-1);

}

return result1 || result2;

Worst Case Time = O(2^(m+n))

**Best Case Time = O(m+n)**

15 int main() {

16 bool result = **Combineable**(0, 0, 0);

17 if (result) {

18 cout << "Yes!!!" <<endl;

19 } else {

20 cout << "No!!!" <<endl;

21 }

22 return 0;

23 }

# Class 28 强化练习 7

**Q1 Reconstruct a Binary Tree by using xxx-order and in-order traversal sequences**

**Q1.1** How to reconstruct a Binary Tree with **pre-order** and **in-order** sequences of all nodes.

10

/ \

5 15

/ \ / \

2 7 12 20

**Index 0 1 2 3 4 5 6**

Preorder: **10**, 5, 2, 7, 15, 12, 20

Inorder: 2, 5, 7, **10**, 12, 15, 20

**Index 0 1 2 3 | 4 5 6**

Preorder: **10** 5 2 7 | 15 12 20

Inorder:  **2 5 7 10 |**  **12 15 20**

Get the first element from pre-order (=10), and find the index of 10 in in-order sequence. Assume its inorder index = mid;

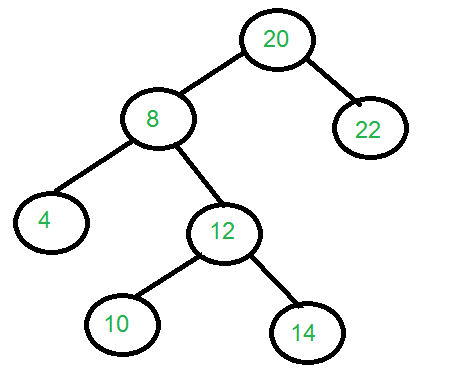
We divide the whole problem into two parts, inorder’s left part = [0, mid], right part = [mid+1, right]

**Q1.2** How to reconstruct a Binary Tree with **post-order** and **in-order** sequences of all nodes.

(Same as pre-order + in-order)

**Q1.3** Construct a Binary Tree from **Inorder** and **Level** order traversals

Given inorder and level-order traversals of a Binary Tree **(you can assume all unique numbers in the tree)**, construct the Binary Tree. Following is an example to illustrate the problem.



Input: Two arrays that represent Inorder and level order traversals of a **Binary Tree**  
**in-order[]** = {4, 8, 10, 12, 14, **20**, 22};  
**level-order[]** = {**20**, 8, 22, 4, 12, 10, 14};

**Method** : hash in-order sequence in each recursion function.

**Step1**: for the 1st element in level[]. pick it up, and look for its index x in inorder[].

form left tree by using inorder [0.. x-1] [x+1,...n-1]

**Step2**: iterate over the level[]. divide it into two sub-arrays,

when scanning an element level[i], if level[i] is in inorder [0.. x-1], then insert it into left part

Else, right part.

**Q 1.4** xxx-order, in-order persist → file; how to reload the binary tree from the file to memory

same with Q 1.1~1.3

**Q 1.5** how about binary search tree?

1. BST with pre-order
2. BST with level-order

**Q 1.6** how to persist a Tree into a file, then reload the Tree back into memory?

**Q2 Most number of points in 2D space problems**

**Q2.1** Given an array of coordinates of points, how to find largest number of points that can be crossed by a same line in 2D space?

Point 1 <x1, y1>

Point 2 <x2, y2>

Point 3 <x3, y3>

……

Point n <xn, yn>

n points

Method: y = ax + b

a: slope

b: intercept

**Hash\_table1** <**key=<a, b>**, value = Set() >

P1: P2~Pn, <a, b> → Hashtable1

P2: P3~Pn, ….

….

Pn-1: Pn …

Corner case: what is the slope is infinity??????

<x1, y1> <x2, y2> iff (x1 == x2)

When the slope is infinity (x1 == x2)

**Hash\_table2** <key = x coordinate, value = Set()>

**Q2.2** Given an array of coordinates of points, how to find the largest number of points that can form a set such that for any pair of points in the set can form a line with positive slope.

o

o

**o**

o

o

**Step 1, sort by x-coordinate and put it in an array A**

Step 2, in A find the **longest increasing subsequence** by y-coordinate.

**Review of LIS problem (Class 13: DP1)**

3 5 4 8 1 6

**Q3 How to design a search suggestion system.**

E.g., **football** -> ticket

season

player

……

Functionalities?

Components?

**Scalability**: how many users/

how much data we have.

**Who are the users**? USA Germany Japan (Localization??)

**Time interval**? how often do we need to update the system

**Typo correction**: fotball or fooootabll or futable ---> football

**Q4** Given an NxN matrix, how to **randomly** generate a maze whose corridor and wall’s width are both 1 cell. In the meantime, for each pair of cells on the corridor, there must exist a path between them. (**Randomly** means that the solution is generated randomly, and whenever the program is executed, the solution can be different.)

**1 1 1 1 1**

**1 1 1 1 1**

**1 1 1 1 1**

**1 1 1 1 1**

**1 1 1 1 1**

**0 0 0 1 0**

**1 1 0 1 0**

**0 1 0 0 0**

**0 1 1 1 0**

**0 0 0 0 0**

**DFS, but for each iteration we move two steps.**

**0 0 0 1 1**

**1 1 0 1 1**

**0 1 0 1 1**

**0 0 0 1 1**

**1 1 1 1 1**

**Q5**  In a 2D black image there are some disjoint white objects with arbitrary shapes, find the number of **disjoint** white objects in an efficient way.

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 0 0 0 0 0 1 1 **0 0**  1 1

1 1 0 0 0 0 0 0 1 **0** 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 **0 0 0 0** 1 1 1 1 1

1 1 1 1 1 **0**  1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

BFS: scan the image row by row, find a 0: then we run BFS from a 0,

counter = 2;

How to quickly detect that a 0 has been visited before (i.e., included in a visited white objects already) ?

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1  **2 2 2 2 2** 1 1 **3 3**  1 1

1 1 **2 2 2 2 2** 1 1 **3** 1 1 1

**2 2 2** 1 1 1 1 1 1 1 1 1 1

1 1 1 1  **4 4 4 4** 1 1 1 1 1

1 1 1 1 1 **4**  1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

if the size of the matix is nxn, then O(n^2)

# Class 29 System Design 3

**1 Design a site similar to tinyurl.com (URL shortener service)**

Example:

<https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=laioffer>

→ <http://tinyurl.com/llmnkn4>

Features:

o long URL => shortened URL

o shortened URL => long URL

o uniqueness?

o how many different URLs? 10B x 200 bytes = 2000 GiB = 2 TiB

o long URL -> integer -> shortened URL

o hash / seq num

o if hash is not stable, i.e. same input may result in different output, we need to store the long URL

o integer encoding.

o [a-zA-Z0-9] 62 chars, il1oO0, 56

o 56^x >= 10B, 56^6=30B

**Continue: Design the serving for the URL shortener site**

shorten URL -> long URL:

[shortened URL -> integer] ← decoding

[integer -> long URL] ← 48 bits, 64 bits

10B URLs

o key: string 5-6 bytes, seq num: 1-5 bytes (varint encoding), 50 GiB

o value: 200 bytes (URL is plaintext, compression), 2 TiB

o machine configuration? ram? cpu? disk? network bandwidth?

o ram 4GiB, cpu 4 core 1.6GHz arm, hard drive 4x600GiB, 1Gib / s = 125 MiB/s

o seek time: 5-10ms, bandwidth: 100MiB/s x 4 = 400 MiB/s

o 4x100 seek per second

capacity of one machine:

o 4 disk x 100 seek / s = 400 seek / s

o 10 ms / seek

o 10 ms latency, 400 qps

o 100 ms latency, 4K qps

QPS: 100

latency: 100ms @95ile

config: serve from disk

QPS: 10K

latency: 100ms @95tile

config: 10K/4K = 3 machine, serve from disk

QPS: 10K

latency: 15ms @95ile, 100ms @98ile, 1s @99ile

config: use ram as cache if we know the access pattern works with our cache schema, 1 machine might be enough. But, if we do not have a good cache schema, we need to serve the data from ram, and need more machines.

response size x qps = bandwidth usage

expected QPS: 10K (network? 10K x 200 bytes = 2MiB/s << bandwidth limit)

latency: 100ms

o how to layout the data in memory? which data structure? compact & fast

o for string keys: search tree

o for seq num (dense): offset + contiguous storage

o hash table, memory overhead

**Q2 How to design a search suggestion system.**

E.g., football -> football ticket

football tv schedule …

This is in fact the problem of how to rank a set of queries sharing a common prefix.

From system designs perspective, here are the tasks:

o what information need to be collected together with the queries

o which data structure to store the queries (and their scoring data)

o *how to make the serving efficient enough, i.e. scalable*

o how to rank the queries

Rank query:

o collaborative filtering.

o users have similar queries in the past

o users with similar geo location

o users clicked similar pages in the past

o etc..

o content-based filtering

o queries share the same time / language / geo location

o queries lead to high CTR

o queries lead to high rank pages

Suggest:

o find the set of candidate queries by looking up index [prefix -> queries]

e.g. one entry in the index:

“football” -> [“football ticket”, “football tv schedule”, “football player” ….]

o index size:

o English words in total: 1M, assume valuable query num: 6M

o average length of query: 15 chars

o estimate max num of prefix: 6M x 15 = 90M (upper bound), reduce to 5M (e.g. de-duplication, importance of the prefix, etc)

o prefix bytes: 5 bytes x 3

o value bytes: [query 1K bytes] x 2K candidates (upper bound) = 2MiB

o 5M x 2MiB = 10 TiB

o rank candidate queries (see above)

How to build candidate set in offline processing?

o cluster queries lead to similar set of result pages / clicked pages

o for each query, collect all relevant information to compose a document representing it

o feedback from users to promote / demote a query. E.g. a suggestion was accepted by the user

# Class 30 Final Exam

**请大家打开个人期中考试的doc**

**Q1.** Determine whether a linked list is a palindrome. Space = O(1);

Example:

Input: a → b→ c → b → a return yes

Input: a → b return false;

**Q2.** Shifting “ABCDEF” to the right by K letters, example, if k == 2, then output == “EFABCD”.

Assuming k < size of the letter

DCBA FE

EFABCD

**Q3.**  Given a Binary Tree, print Right view of it. Right view of a Binary Tree is set of nodes visible when tree is viewed from Right side.

Right view of following tree is 1 3 7 8 11  
  
 1  
 / \  
 2 3 level == 2  
 / \ / \  
 4 5 6 7 level == 3 ...  
 / \ ⇐======== people  
 9 8 level = 4

/ \

10 11 level 5

private void rightView(TreeNode root, int depth) {

if (root == null) {

return;

}

if (depth > maxDepth) {

maxDepth = depth;

System.out.println(root.val);

}

rightView(root.right, depth+1);

rightView(root.left, depth+1);

}

**Q4** Given an array of strings, find if **all the strings** can be chained to form a circle

**Input**: arr[] = {"aaa", "bbb", "baa", "aab"};  
Output: Yes, the given strings can be chained. The strings can be chained as "aaa", "aab", "bbb"   
and "baa"Output: Yes  
  
**Input**: arr[] = {"aaa", "bbb"};  
Output: No

**Input**: arr[] = {"aaa"};  
Output: Yes

**Q5** Given two values k1 and k2 (where k1 < k2) and a root pointer to a Binary Search Tree. Print all the keys of tree in range k1 to k2. i.e. print all x such that k1<=x<=k2 and x is a key of given BST. **Print all the keys in an increasing order.**

1) If value of root’s key > k1, then recursively call in left subtree.

2) If value of root’s key is in range, then print the root’s key.

3) If value of root’s key < k2, then recursively call in right subtree.