第28节课时间: 2014年11月07日，星期五，太平洋时间晚上7:00

前15节教案链接

上课语音会议链接：
https://global.gotomeeting.com/meeting/join/163634853

C++ 上机课 语音会议链接:
https://global.gotomeeting.com/join/173379397
The class material google doc is here

Announcement:
1.上机课时间：
    Java 周六下午 太平洋时间1:00 - 3:00 pm (闫老师 laioffer.java@gmail.com)
    C++ 周六下午 太平洋时间3:00 - 5:00 pm (王老师 laioffer.cpp@gmail.com)
2. Homework Solution (Java version)
3. Homework Solution (C++ version)
4. 本班级QQ群 316871642

Class 29 期末考试 时间11月13日（周4），晚上7点

课程列表：

Class 16 Probability, Sampling, Randomization, etc.

Question 1: shuffling algorithm (OOD):

1.1. spades (♠),
1.2. hearts (♥),
1.3. diamonds (♦)
1.4. 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

Step 3: [0...49]

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)

then when t = k,

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

x(t= k-1) 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

```
int S[...];
int solu = S[1]; //unlimited data flow, which might not stay in the memory forever,
while (i++) {  // current element is the i-th element
      int r = random(1 , i); // randomly generate a number from 1 to i (inclusive);
      if (r == 1 ) {
            solu = S[i];  // the probability  of choosing i-th element as the final solution is 1/i;
      }
      // return a number “solu”  if someone asks for it.
   }
```

Reservoir sampling is a family of randomized algorithms 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.

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

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

1/25

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)

Better one??

Random(2):   0 vs 1
index 210

000
001
010
011
100
101
110
111

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

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

Random(5) → Random(10)
how many time do you need to call Random(5)   ~2
Random(10) → Random(1million)   6 * 2 ~ 12

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

example: **1 3 7 6 -2, 100............. n**
1 1&3 3 3&6

Idea: two heaps

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.

```
// 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
    (2) if n_min < n_max
        if x>= MIN_HEAP.top(), insert this element into MIN_HEAP
        else insert MAX_HEAP.top() into MIN_HEAP , and insert x into MAX_HEAP.
    (3)  if n_min > n_max
        if x<= MAX_HEAP.top()
O(nlog(n))
```

Question 5: Given 200 of urls, how to find 95-th **percentile** of all url’s length
http://en.wikipedia.org/wiki/Percentile

y-value	10	13	14	**5**		3
x-value	500	501	502 ….	**600**	|	750

The longest length of a url is 4010

Step 1: allocate an array of [4010 + 1], and each element i in this array represents the number of URLs with the length equal to i.
Step 2: count the number of URLs with each particular length by scanning each URL.
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

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 17 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。

123455555Xxxxxx 89
|
index

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 11223
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;
}
```

===

```
// 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];
}
```

1   1   1

```
slow - 1   slow    fast
```

```
index  0  1  2   3  4  5
input  1  1  1   2  2  3
          slow
          fast

output 1  1  2  2  3
```

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**

1 2 3 4 5 6 7 8
  i
small == 1
big == 1

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)

small[] = {1, 3, 5, 7}
large[] = {2, 4, 6, 8}
→ **1.5n**

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

binary reduction

```
                         <8,   [7, 6, 4]>
             <4, [3,2]>                         <8, [7, 6]>]
<2, [1]>           <4,[3]>          <6, [5]>       <8, [7>]
 12                    34                56              78
```

n + 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
```

```
void sprialprint(int[][] a, int offset == 1, int size, int counter) {
        if size <= 1
            print   // base case
        for(i = 0; i < size-1; i++) {   // size == 5;   upper row
                a[offset][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     // lower row (from right to left)
        for    //  left column
        …..
        //recursive rule
        spiralprint(a, offset + 1, size - 2, counter);
}
```

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

Q4.1 classical way

```
void pntBT (Node* root) {
     if (root == null) {
         return;
     }
     Queue<TreeNode> q = new LinkedList<TreeNode>();
     q.offer(root);
     while (!q.isEmpty()) {
         int size = queue.size();
         for (int index = 0; index < size; index++) {
```

```
        TreeNode tmp = q.poll();
        if(tmp.left != null) {
            q.offer(tmp.left);
        }
        if(tmp.right != null) {
            q.offer(tmp.right);
        }
        System.out.print(tmp.value);
    }
    System.out.println();
   }
}
```

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

```
          10
       5       15
   2   7     12     20
```

10
15 5
2 7 12 20

Case 1: in odd layer: 10, expand 10 and generate lChild 5 and rChild 15. Now we have 5 - **15** in the queue.

Case 2: in even layer: we expand every node **from right to left i**n the queue, and generate rChild and then lChild, and push them to the front (left end) to the queue, so we get
2 ⇐ 7 <=12   ⇐ 20

```
=====================================================
public void printZigzag(TreeNode root) {
    if (root == null) {
        return;
    }
    Queue<TreeNode> queue = new LinkedList<TreeNode>();
    boolean flag = false;
    queue.offer(root);
    while (!queue.isEmpty()) {
        int size = queue.size();
        for (int i = 0; i < size; ++ i) {
            if (!flag) {
                TreeNode temp = queue.poll();
                System.out.print(temp.val + “ ”);
                if (temp.left != null) {
                    queue.offer(temp.left);
                }
                if (temp.right != null) {
                    queue.offer(temp.right);
                }
            } else {
                TreeNode temp = queue.removeLast();
                System.out.println(temp.val + “ ”);
                if (temp.right != null) {
                    queue.addFirst(temp.right);
                }
                if (temp.left != null) {
                    queue.addFirst(temp.left);
                }
            }
        }
        System.out.println();
        flag = !flag;
    }
}
```

Q5 (Tree) Lowest Common Ancestor

```
                        10
                5                 15
        2=p       7          12  =q        20
               6   9
```

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
TreeNode* LCA(TreeNode* root, TreeNode* a, TreeNode* b) {
 if (root == NULL) {
    return NULL;
 }
 // One of a or b is root, so root is their LCA
if (root == a || root == b) {
     return root;
}
TreeNode* left = LCA(root->left, a, b); // recursive rule
TreeNode* right = LCA(root->right, a, b);
if (left != NULL && right != NULL) { // a and b are in different subtrees
    return root;
} else {
    return left ? left : right;
}
}
```

Q5.2: we have k nodes store in a **vector<Node*>**

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

```
                   10 ab
           5   a                15 b
       2         7 =a       12 =b        20
              6   9=a
```

Main idea 1: use a hashtable to store all ancestors of Node a ---> **7 5 10**
Iteratively go up from b to check whether the current node is in the
Node b→ **15 10 O(levels of the tree)**

Main idea 2:
9 is on the 3rd layer
12 is on the 2nd layer

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

2.2.3.

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             Fn-2                                     2
Fn-2   Fn-3      Fn-3Fn-4                                     4
                                                              8…
```

```
public int fibN(int n) {
   if (n == 1 || n == 2)   // base case
         return 1;
   return fibN(n-1) + fibN(n-2);    // recursive rule
}
```

index   0, 1   2   3 4 5 ...

M[n] = { 1, 1, 2, 3 5 8 ...   }

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.

```
Base case: 1 element only

index = 0 1 2 3 4 5 6 7 ....
A[] =  {7, 2, 3, 1, 5, 8, 9, 6}
M[] = { 1,1,2,1 2 3 4 x}

max_so_far = 1;
```

M[i] represents the length of the longest sub-array from left-hand side to the i-the element (including the i-the element)

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**).

base case: 1 meter-long rope

—|

a little larger (2m):

—|—

even larger (3m)

— — | —

 M[2]   M[1]

—|— —

M[1]   M[2]

even larger2 (4m)   in order to get M[4]

— — — | —   **Case1**

 M[3]   M[1]

— —| — —   **Case2 ⇒ maximize the product of Left * Right**

M[2]   M[2]

— |— — —   **Case3**

M[1] M[3]

```
public int getMaxProduct(int n) {
   int M[n+1];
   M[0] = 0;  // base cases
   M[1] = 0;
   for (int i = 1; i <= n; i++)  // For EVERY METER added
   {
         int max_val = 0;
         for (int j = 1; j <= i/2; j++) { // j meters in the right most (大段)
                int best_left = max(i-j, M[i-j]);
                int best_right = max(j, M[j]);
                max_val = Math.max(max_val, best_left * best_right);
         }
         M[i] = max_val;
```

```
    }
    return M[n];
}
```

Q1 Largest sum of a subarray

Given an unsorted array, find the **subarray** that has the greatest sum. Return the sum.

For example: If the input array is {**1, 2, 4,** -1, -2, -1}, the greatest sum is achieved by subarray {1, 2, 4}.

index = 0 1 2   3   4   5   6   7
A = {   1, 2, 4, -1, -2, 10, -50, 1000 }
M=   {   1 3 7   **6**   4   14   -36   1000   }

M[i] represents the largest sum from left-hand side to the i-th element (including the i-th element).

Q2 Dictionary word problem

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

Dictionary:

bob

cat

rob

Word input 1: bcoabt

Solution: False

Word input 2 : **bobcatrob** || catbob

Solution: True

Step1: assume we have hash all the words into a hash_table, so that we can determine whether a word is in the dictionary or not in O(1).

```
public boolean wordSolver(String word, HashSet<String> dict) {
    // store solutions to subproblems in array
    boolean[] canDo = new boolean[word.length() + 1];//include 0
    for (int i = 1; i <= word.length(); i++) { // 1 letter to n letters
        // If the word is in the dictionary, done
        if (dict.contains(word.subString(0,i))) {
            canDo[i] = true;
            continue;
        }
        // Otherwise, check the possible single splits
        for (int j = 1; j < i; j++) {
            // check subproblem and check the rest of the word
            if (canDo[j] && dict.contains(word.subString(j,i))){
                canDo[i] = true;
                break;
            }
        }
    }
    return canDo[word.length()];
}
```

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

```
public int editDistance(String word1, String word2) {
   // Base case
   if (word1.isEmpty()) return word2.length();
   if (word2.isEmpty()) return word1.length();

   // (a) Check what the distance is if the characters are equal
   // and we do nothing first
  int nothing = Integer.MAX_VALUE;
   if (word1.charAt(0) == word2.charAt(0)) {
        nothing = editDistance(word1.substring(1),
                               word2.substring(1))
  }
   // (b) Check what the distance is if we do a Replace first?
   int replace = 1 + editDistance(word1.substring(1),
                                 word2.substring(1))
   // (c) Check what the distance is if we do a Delete first?
   int delete = 1 + editDistance(word1.substring(1), word2);
   // (d) Check what the distance is if we do a Insert first?
   int insert = 1+ editDistance(word1, word2.substring(1));
   // Return best solution
   return min(nothing, replace, delete, insert);
}
```

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

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]

Case 2. replace c1 with c2: distance(s1r + c1, s2r + c2) = 1 + distance(s1r,s2r)
a
s
editDistance(1,1) -> 1 + editDistance(0,0)

Case 3. delete c1: distance(s1r+c1, s2r+c2) = 1 + distance(s1r, s2)
a
s
editDistance(1,1) -> 1 + editDistance(0,1)

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)

Subsolution[i][j] represents the subsolution of   s1[0….i] s2[0...j]   ⇒ the number of actions needed for converting s1[0..i] to s2[0..j]

```
       s2 s g h j
  ind   0 1 2 3 4
s1   0   0 1 2 3 4     x is the base case.
a    1   1 x=min(0+1, 1+1, 1+1        )? 2 3 4
s    2   2 1 2 3 4
d    3   3 2 2 3 4
f    4   4 3 3 3 4  == final solution
      s2   m a s d f
  ind   0 1 2 3 4 5
s1   0   0 1 2 3 4 5
a    1   1 1 1 x
s    2   2
d    3   3
f    4   4 == final solution
```

```java
public int editDistance(String word1, String word2) {
    int len1 = word1.length();
    int len2 = word2.length();

    // len1+1, len2+1, because we will return dp[len1][len2]
    int[][] dp = new int[len1 + 1][len2 + 1];
    // BASE CASES
    // fill in the first column (column 0)
    for (int i = 0; i <= len1; i++) {
        dp[i][0] = i;
    }
    // fill in the first row
    for (int j = 0; j <= len2; j++) {
        dp[0][j] = j;
    }

    //iterate through, and check last char
    for (int i = 1; i <= len1; i++) { // s1’s letters -->
        char c1 = word1.charAt(i-1);
        for (int j = 1; j <= len2; j++) { // s2’s letters-->
            char c2 = word2.charAt(j-1);

    //if last two chars equal, this is the best we can do
            if (c1 == c2) {
                dp[i][j] = dp[i-1][j-1]; // case 1
            } else {
                int replace = dp[i-1][j-1] + 1;//case2
                int insert = dp[i-1][j] + 1; // case3
                int delete = dp[i][j-1] + 1; // case4
                int min = Math.min(replace, insert);
                min = Math.min(min, delete);
                dp[i][j] = min;
            }
        }
    }
    return dp[len1][len2];
}
```

Class 18 System Design 2 (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              |....
```

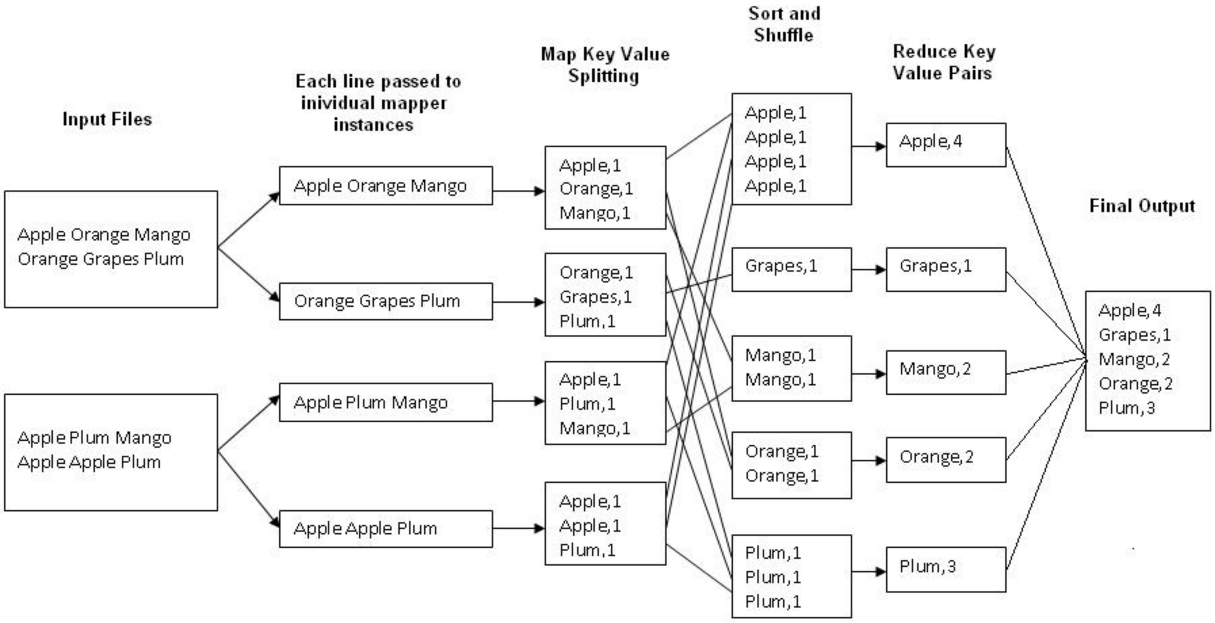
M1: Apple -1, Mango -1
M2: Plum-1, Orange-1
M3:.Apple-2, Plum-1

如何汇总结果?
Reduce
==================
M1, M2, M3 → M4
==================
m1
        m12

```
m2
                    m1234
m3
        m34
m4

m5
        m56         m56
m6
==================
```

继续利用所有的机器?

Shuffle intermediate results



Map → **Shuffle** → **Reduce**

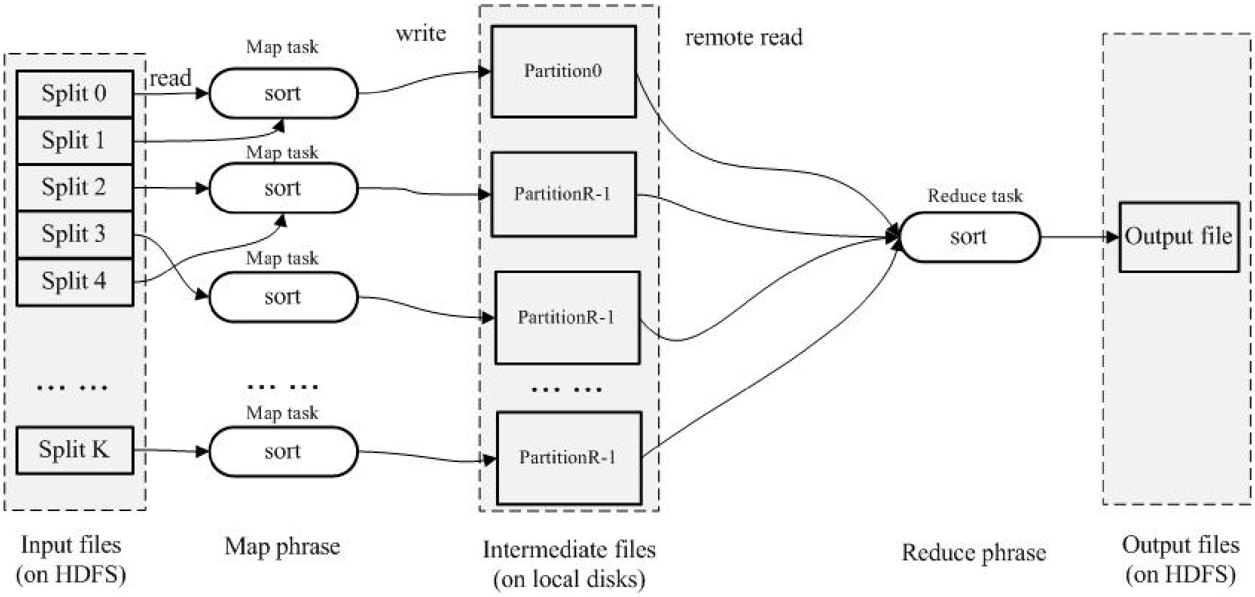
II. Terasort

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

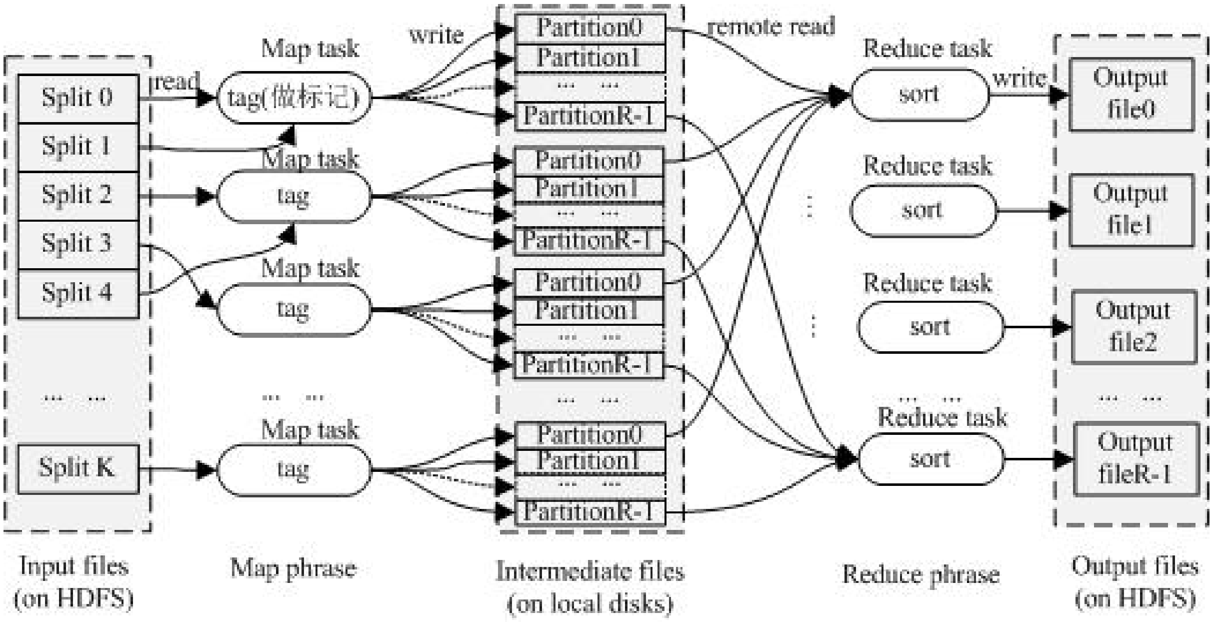
Solution 2:
1) 在map阶段, 每个map task都会将数据划分成R个数据块(R为reduce task个数), 其中第i(i>0)个数据块的所有数据都会比第i+1个中的数据大.

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...

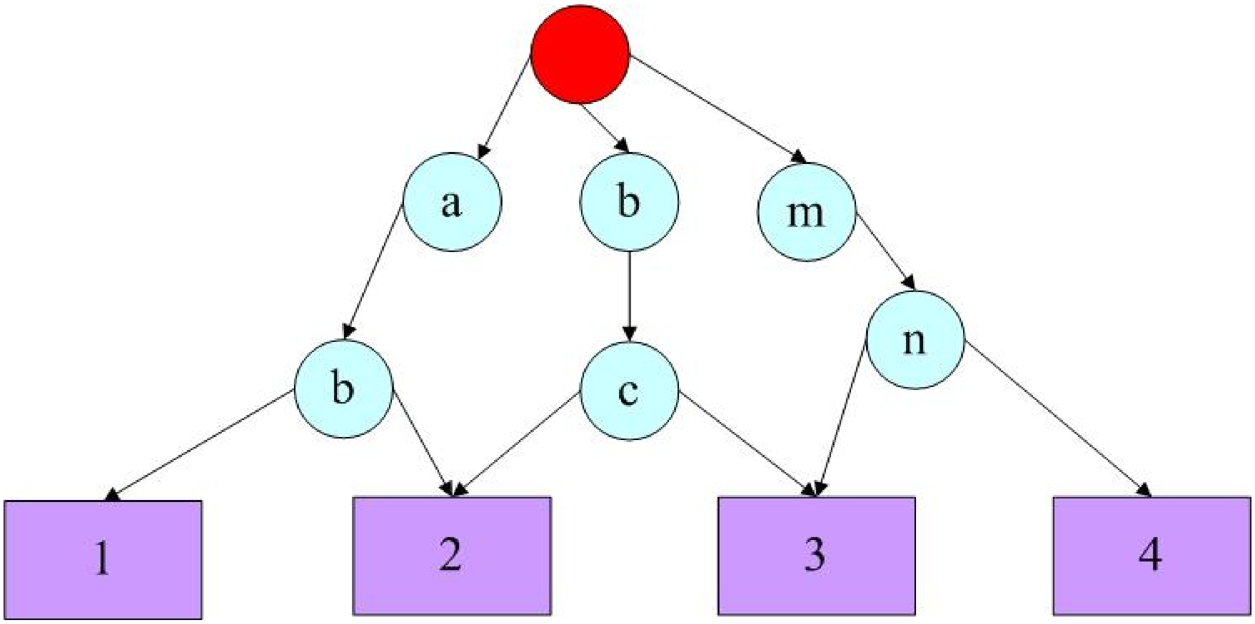
Mapper_4
input: rrr, mnk, qfdk
mnk... || qfdk... || rrr…

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 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.

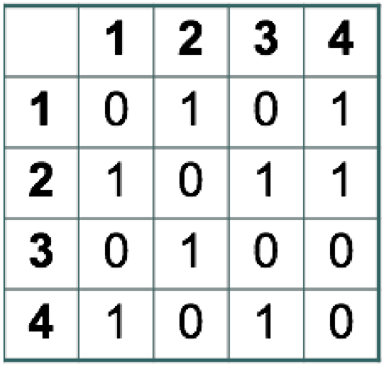
III. Parallel Breadth-First Search

图的定义

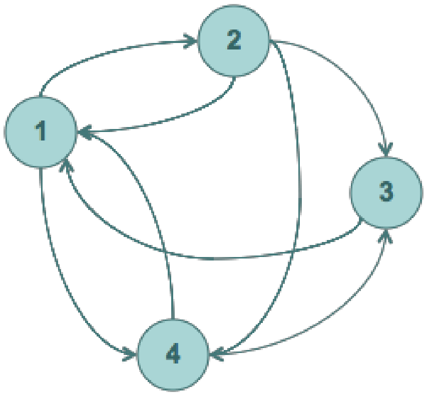
Graph G=(V, E)
V: represents the set of vertices (nodes)
E: represents the set of edges (links)
Both vertices and edges may contain additional information. E.g., distance

图的表示

1) Adjacency Matrices
Represent a graph as an n x n square matrix M:
n = |V| (点的个数)
M_{ij} = 1 means a link from node i to j

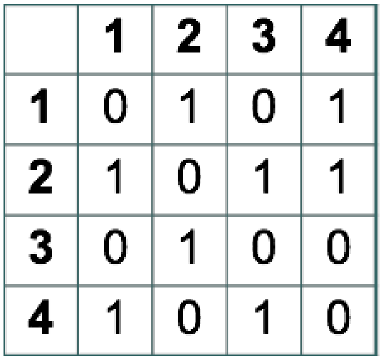


	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	0
4	1	0	1	0



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



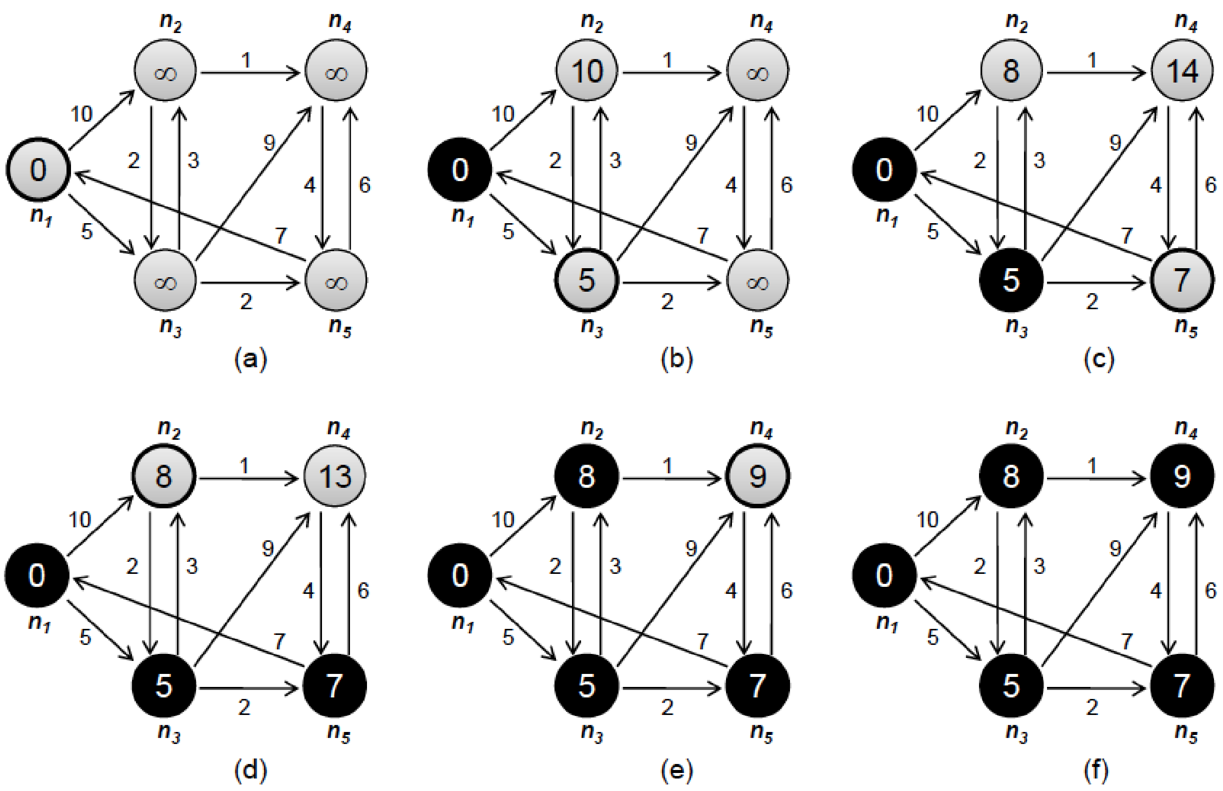
	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	0
4	1	0	1	0

→

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

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:

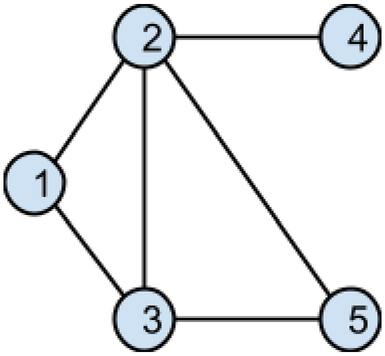
1) A map task receives

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, 1)=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)

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 new finding between two steps

Class 19 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
 --------------->
      --------------------------->
```

```
public  class ListNode{
     int value;
     ListNode next;
     ListNode forward;
     ListNode (int value){
        this.value = value;
     }
}
```

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

```
N1 -> N2 -> N3 -> N4 ->  N5 ->NULL
--------------->
|      |      |     |       |
N1’ -> N2’ -> N3’ -> N4’ -> N5 ->NULL
---------------->
```

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, **(avoid duplication if you want to unify iteration 1 and 2 into a single iteration**),

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

}

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

Method: build a hash_map to avoid duplication when copying a node.

```
 N1 --  N2            N1'   --- N2'
  \       /                \       /
    N3                    N3'
```

If we use **BFS**
Step1: Expanding N1: make a copy of N1 copy(N1 , N1') and insert <N1 --> N1'> into the hashtable

-- generate N2, since N2 is not in hashtable yet, new(N2') , insert <N2-->N2'> into the hash table.

-- generate N3, since N3 is not in hashtable yet, new(N3') , insert <N3-->N3'> into the hash table.

Step2: Expanding N2:

-- generate N3, since N3 is already in the hashtable, we do not need to new(N3'), we only need to find the 1:1 mapping between N3 and N3', that is N2'-> N3'

Step3: Expanding N3:

-- generate nothing since all its neighbor have been expanded…..

Q2 : **k-way merge problems**
Q2.1 How to merge **k sorted array** into one big sorted array?

Method1: Use a min-heap

```
//da jia hao!
class Element {
        double value;
        int index; // for kth array
        int pos; // position at original array
```

}

O(k) + **O(km *log(k))**

extra space: heap + final_solution**(size = mk)**

M2: binary-reduction

```
(1)                      13579…
               i
(2)    (12)              13478…
                         j
(3)        (1234)        23456…
                         x
(4)    (34)              56789…
                         y
```

O(log(k) km)
space complexity: O(2* mk)

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   best=3
     5          15  best=2        target ==  13
  2   7      12=c  best=1    20
           /  \
              null
```

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;

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

```
        10
```

```
     5          15        target ==  13
  2   7       12    20
```

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

Method: three cases

(1) root.val = target
   a) 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
   b) else replace root with (1) non-null child is any (2) null

(2) root.val > target go to left subtree by recursion.

(3) root.val < target go to right subtree by recursion.

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

```
      10
   /      \
  5        15
 / \      /   \
2   7   12    20
```

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..)
   1.1. if the **weight** of each smallest element in the original data is identical/similar
      1.1.1. e.g. **identical**: 1 meter of rope
      1.1.2. e.g. **similar**: a letter, a number

   Then this kind of problem is usually simple:

   Linear scan and look back to the previous element(s)

   For example:

   Longest Ascending Subarray (when at i, look back at i-1)

   Longest Ascending Subsequence (when at i, look back at 1….i-1)

   1.2. If the **weight** is not the same:
      1.2.1. e.g. DP1 课后题： 沙子归并
      1.2.2. e.g. 强化练习题： 切木头

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

2. **(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),

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[0]是2，A[1]是4， A[2]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  <--- 切割点 index
0   1   2   3   4   5   6   7   8   9   10  <--- 木材

|______ | ______ | _  _  _ | _ __ _  _ |
 [ i=1                          j=4 ]
```

index 0 1 2 3 4

0 **0 0** x x x

1 x **0 0** 5 13=? C(1,4) = MIN (C(1,2) + C(24) , C(1,3)+C(34)) + L(14)

2 x x **0 0** 6 c(2,4) = c(2,3) + c(3,4) + L(2,4)

3 x x x **0 0**

4 x x x x **0**

```
(1)   Fill in the form from the left hand side to right
(2)   Fill in the form from bottom up
(3)   care about right up half of the matrix
```

```
                                        i< k < j
C(i,j) for i<k<j,   C(I,J)    =C(i,k)+C(k,j)+L(i,j), C(i,j) = min(S(k))

-----
i=2,j=4,k=3
C(2,4)=C(2,3)+C(3,4)+L(2,4)=0+0+6
-----


C(0,4)=min(
  C(0,1)+C(1,4)+10,
  C(0,2)+C(2,4)+10,
  C(0,3)+C(3,4)+10,
)

C(0,1)=0
C(0,2)=4
C(0,3)=min(
  C(0,1)+C(1,3)+7, // 12
  C(0,2)+C(2,3)+7, // 11
) = 11

C(1,4)=min(
  C(1,2)+C(2,4)+8, // 14
  C(1,3)+C(3,4)+8, // 13
) = 13

C(1,2)=0
C(1,3)=5
C(2,3)=0
C(2,4)=6
C(3,4)=0
```

Class 20 Midterm 2

请同学们打开我跟每个人share 的1:1 google doc

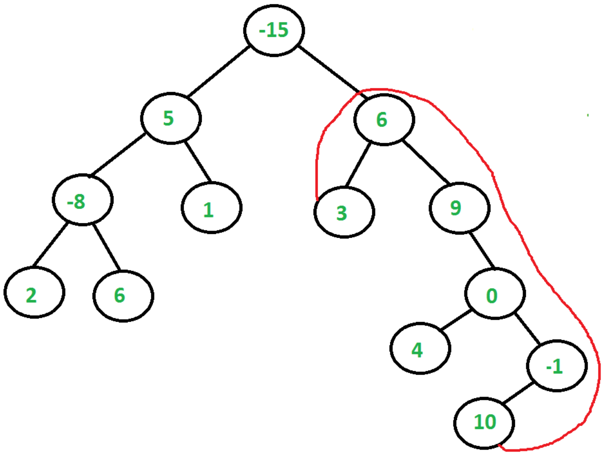
A complete answer will include the following:
1. Document your assumptions
2. Explain your approach and how you intend to solve the problem
3. Provide code comments where applicable
4. Explain the big-O run time complexity of your solution. Justify your answer.
5. Identify any additional data structures 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

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 example, in the following binary tree, the maximum sum is 27(3 + 6 + 9 + 0 – 1 + 10). Expected time complexity is O(n).



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

example:

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

M[i] = { 0, 1 x }

a

a | b

a b **(reuse subsolution)** | o **(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**)

0

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^3).

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

i = 7 [0, 7] [1, 7] … [6, 7]

```
public class test {
    public int minCut(String s) {
        int min = 0;
        int len = s.length();
        boolean[][] matrix = new boolean[len][len]; // optimization for speed up the procedure to determine whether a string is a palindrome.
        int cuts[] = new int[len+1];

        if (s == null || s.length() == 0)
```

```
            return min;
        //初始化cuts里面的值为最坏情况的值
        for (int i = 0; i < len; ++i){
            cuts[i] = len - i;
        }
        //dp过程
        for (int i=len-1; i>=0; --i){
            for (int j=i; j<len; ++j){
                if ((s.charAt(i) == s.charAt(j) && (j-i < 2))
                        || (s.charAt(i) == s.charAt(j) && matrix[i+1][j-1]))
                {
                    matrix[i][j] = true;
                    cuts[i] = getMinValue(cuts[i], cuts[j+1]+1);
                }
            }
        }
        min = cuts[0];
        return min-1;
    }

    public int getMinValue(int a, int b){
        return a > b ? b : a;
    }

    public static void main(String[] args) {
        System.out.println(new test().minCut("ababbbabbaba"));
    }
}
```

DP 的解题常用方法：

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

    Then this kind of problem is usually simple:

    Linear scan and look back to the previous element(s)

    For example:

    Longest Ascending Subarray (when at i, look back at i-1)

    Longest Ascending Subsequence (when at i, look back at 1….i-1)

M[i] represents the minimum number of cuts needed to make all sub-strings of [0… i] all palindrome.

Class 21 Process, Thread and Concurrency

目标:
1. 了解Process, Thread, 互斥(Mutual Exclusion), 死锁(deadlock)等基本概念
2. 理解通过P/V原语解决互斥同步问题
3. 通过经典例子掌握互斥同步原理

I. Thread/Process基本概念
Thread of execution: Single sequence of instructions executed by the processor

Thread vs. Process
Both processes and threads are independent sequences of execution. The typical difference is that **threads** (of the same process) run in a **shared memory space**, while **processes** run in **separate memory spaces**. A thread is a subset of a process. A process contains one or more threads.

Threads in the same process **share** memory and open files
– BUT with **separate** program counter, registers, and stack
– Shared memory includes the heap and global/static data
– **No** memory protection among the threads

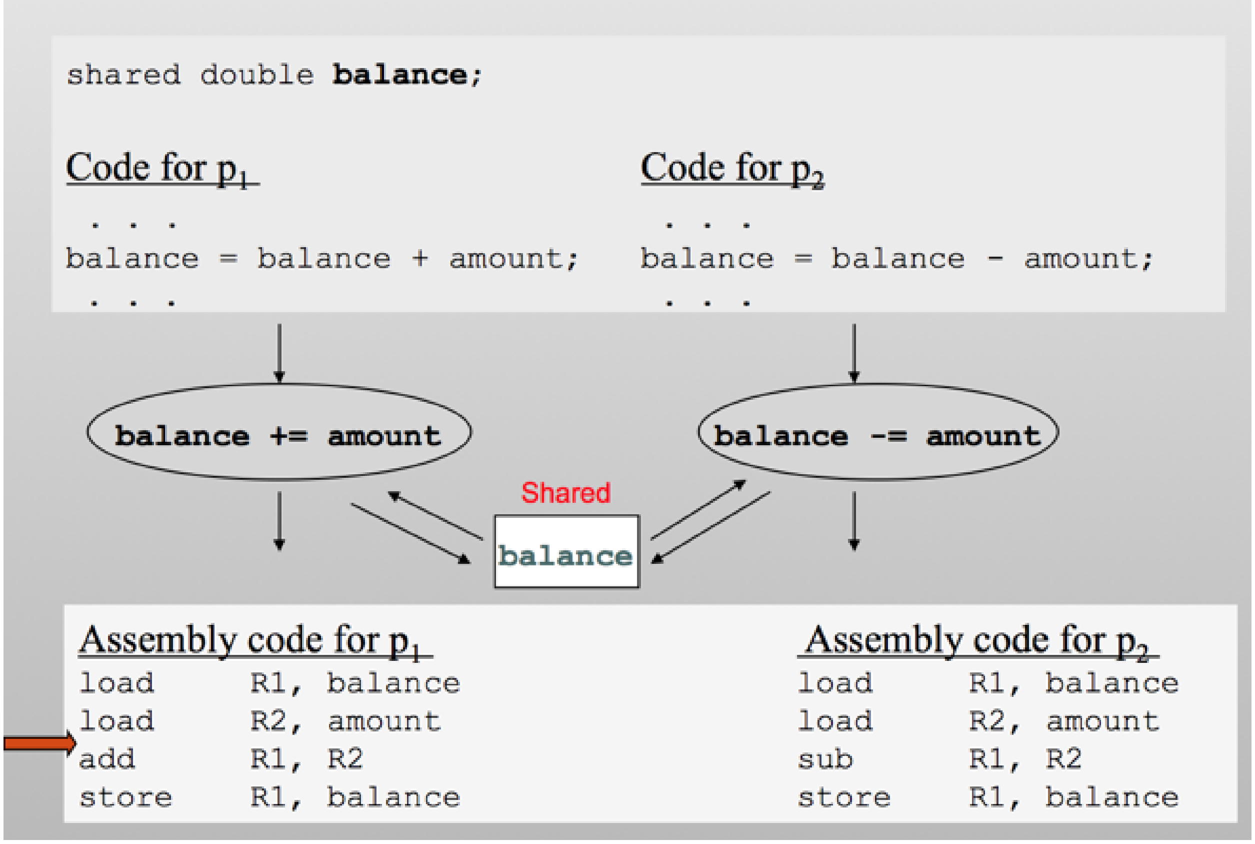
Threads share:
• Text segment (instructions)
• Data segment (static and global data)
• BSS segment (uninitialized data) http://en.wikipedia.org/wiki/Data_segment#BSS
• Open file descriptors
• Signals
• Current working directory
• User and group IDs

Threads do not share:
• Thread ID
• Saved registers, stack pointer, instruction pointer
• Stack (local variables, temporary variables, return addresses)
• Signal mask
• Priority (scheduling information)

线程的优点:
1) More efficient
– Much less overhead to create: no need to create new copy of memory space, file descriptors, etc.
2) Sharing memory is easy (automatic)
– No need to figure out inter-process communication mechanisms
3) Take advantage of multiple CPUs – just like processes
– Program scales with increasing # of CPUs
– Take advantage of multiple cores

II. Critical Sections & Mutual Exclusion

例子:
两个process p1, p2



```
shared double balance;
```

Code for p_1

```
 . . .
balance = balance + amount;
 . . .
```

Code for p_2

```
 . . .
balance = balance - amount;
 . . .
```

Assembly code for p_1

```
load    R1, balance
load    R2, amount
add     R1, R2
store   R1, balance
```

Assembly code for p_2

```
load    R1, balance
load    R2, amount
sub     R1, R2
store   R1, balance
```

Mutual exclusion (互斥): Only one process can be in the critical section at a time. Without mutual exclusion, results of multiple execution are not consistent

When there is a ***race*** to execute critical sections, need an OS mechanism so programmer can resolve:
1) Disable interrupts
2) Software solution – locks, semaphores

利用**锁机制(Lock)**:

```
Code for p1                             Code for p2
 . . .                                   . . .
get(lock);                              get(lock);
 <execute critical section>;             <execute critical section>;
release(lock);                          release(lock);
```

注意: get(lock) and release(lock) operations must be **atomic (原子操作)**

对锁机制实现的要求:

1. 互斥: 同一时间只能有一个process访问critical section
Mutual exclusion: Only one process at a time in the Critical Section (CS)

2. 避免死锁: **NO DEADLOCK**
Example:
P1: holding Lock L1, trying to acquire L2
P2: holding L2, trying to acquire L1

```
 Code for p1                             Code for p2
 . . .                                   . . .
 get(lock1);                             get(lock2);
 . . .                                   . . .
 /* Enter CS */                          /* Enter CS */
 get(lock2);                             get(lock1);
 . . .                                   . . .
 release(lock2);                         release(lock1);
 . . .                                   . . .
 release(lock1);                         release(lock2);
 . . .                                   . . .
```

避免死锁: 保证拿锁顺序一致!

```
void f1() {                ← p1 is running f1
  get(lock1);
  // do sth.                ← p1 comes here
  get(lock2);
  // do sth.
  release(lock2);
  release(lock1);
}

void f2() {                 ← p2
```

```
  get(lock2);
  // do sth.                    ← p2 comes here
  get(lock1);
  // do sth.
  release(lock1);
  release(lock2);
}
```

3. 避免process饥饿: **NO STARVATION**
Once a process attempts to enter its CS, it should not be postponed indefinitely

III. Semaphore, P()/V()

1) Conceptual OS mechanism, with no specific implementation defined (could be get()/release())
2) Basis of all contemporary OS synch. mechanisms

A **semaphore**, s, is a nonnegative integer variable that can only be changed or tested by these two **atomic** (indivisible / uninterruptable) functions:

P(s): [while(s == 0) {wait}; s = s – 1;] (get(Lock), pthread_mutex_lock())
V(s): [s = s + 1;] (release(Lock), pthread_mutex_unlock())

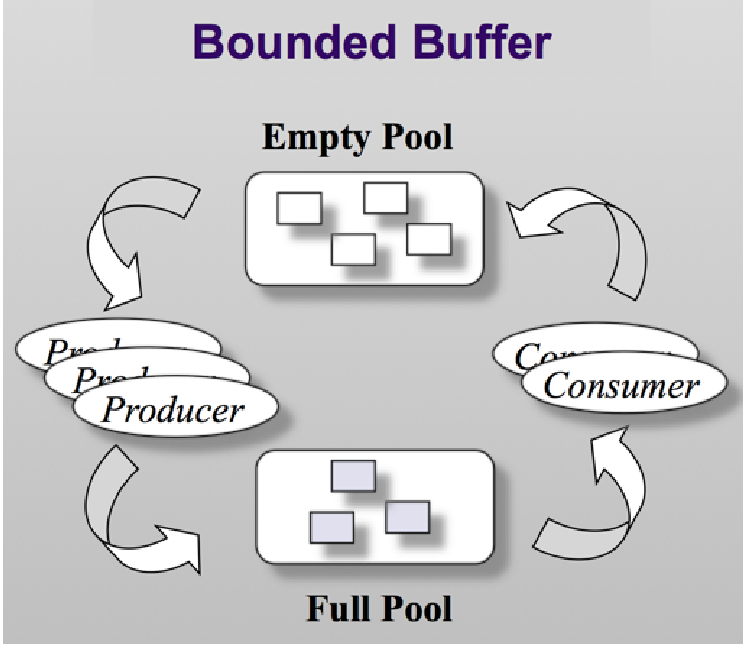
3.1 Account Balance Example

```
semaphore mutex = 1;
P0() {
  . . .
  /* Enter the CS */
  P(mutex);
    balance += amount;
  V(mutex);
  . . .
}

P1() {
  . . .
  /* Enter the CS */
  P(mutex);
    balance -= amount;
  V(mutex);
  . . .
}
```

3.2 Producer and Consumer

生产者和消费者问题:

One or more producers are generating data and placing these in a buffer
Each consumer is taking items out of the buffer one at time



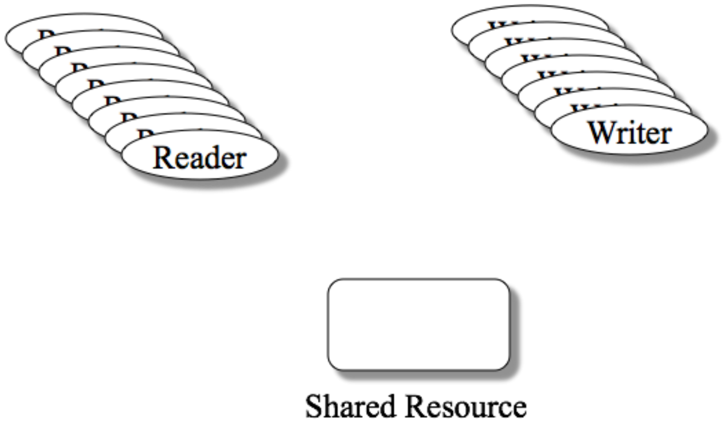
一种解法:

```
Semaphore s = 1; // 用来作为对buffer操作进行保护的lock
Semaphore n = 0; // 用来指示buffer里available的item
Semaphore e = sizeOfBuffer; // 用来指示buffer的剩余空间

void procuder() {
  while (true) {
    produce();     // 生产, 但还没放入buffer
    semWait(e);  // P(e), 等待可用空间
    semWait(s);  // P(s), 拿锁保证原子操作
    appendBuffer();  // 放入buffer
    semSignal(s);  // V(s), 放锁
    semSignal(n);  // V(n), buffer里多了一个available的item
  }
}

void consumer() {
  while (true) {
    semWait(n);  // P(n), 等待buffer里available的item
    semWait(s);  // P(s), 拿锁保证原子操作
    takeFromBuffer();  // 从buffer里取
    semSignal(s);  // V(s)
    semSignal(e);  // V(e), buffer里多了一个空
    consume();      // 可以消费这个取出的item了
  }
}
```

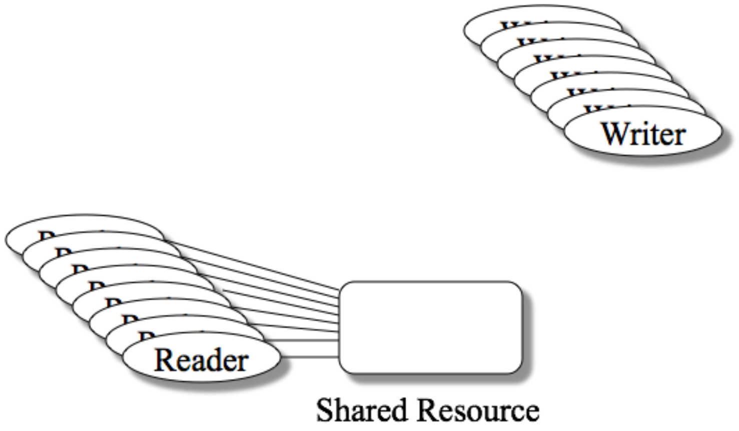
Example: java blockqueue
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html

3.3 Reader/Writer

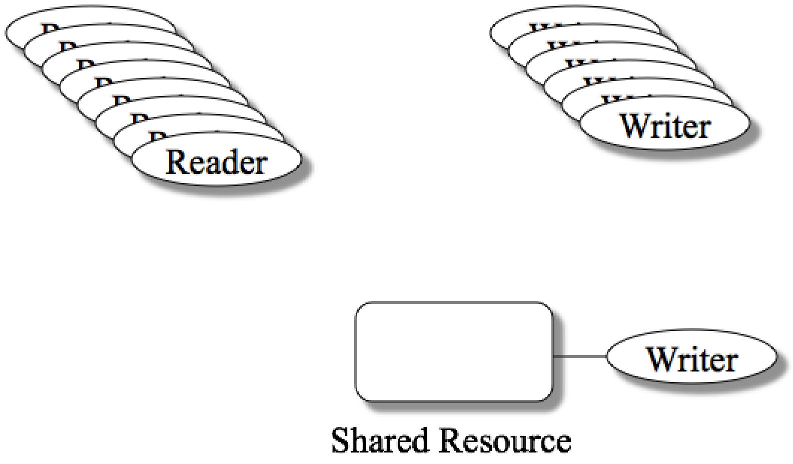
读者, 写者, 共享资源



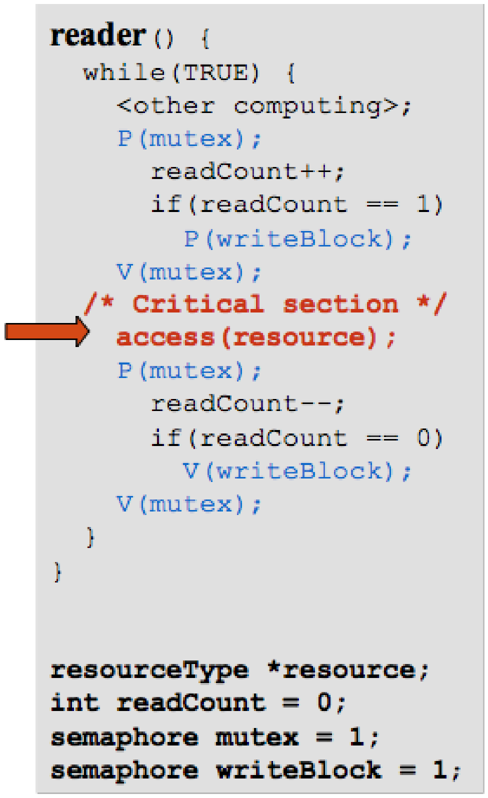
多个读者可以共享



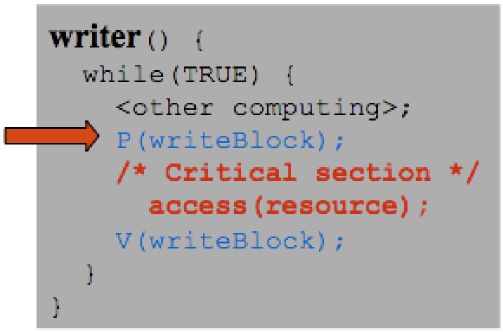
写者独享资源



解法1:



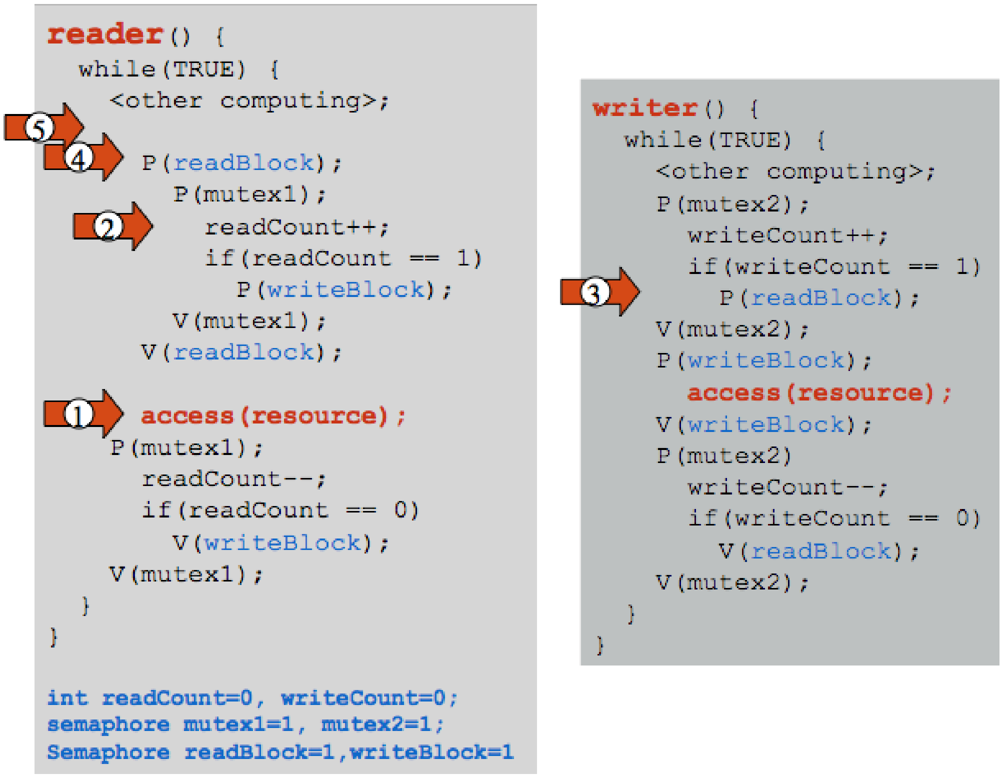
```
reader() {
  while(TRUE) {
    <other computing>;
    P(mutex);
      readCount++;
      if(readCount == 1)
        P(writeBlock);
    V(mutex);
  /* Critical section */
    access(resource);
    P(mutex);
      readCount--;
      if(readCount == 0)
        V(writeBlock);
    V(mutex);
  }
}

resourceType *resource;
int readCount = 0;
semaphore mutex = 1;
semaphore writeBlock = 1;
```



```
writer() {
  while(TRUE) {
    <other computing>;
    P(writeBlock);
    /* Critical section */
      access(resource);
    V(writeBlock);
  }
}
```

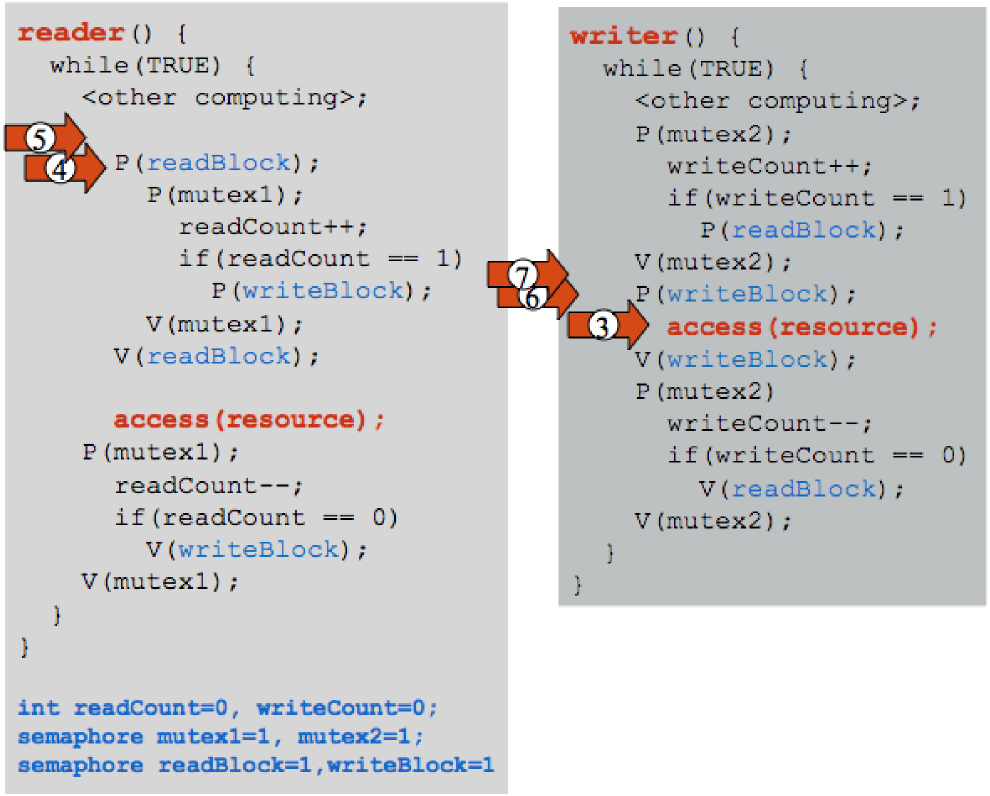
1) First reader competes with writers
2) Last reader signals writers
3) Any writer must wait for all readers
4) Readers can starve writers, and “Updates” can be delayed forever

解法2:



```
reader() {
  while(TRUE) {
    <other computing>;

      P(readBlock);
        P(mutex1);
          readCount++;
          if(readCount == 1)
            P(writeBlock);
        V(mutex1);
      V(readBlock);

      access(resource);
    P(mutex1);
      readCount--;
      if(readCount == 0)
        V(writeBlock);
    V(mutex1);
  }
}

int readCount=0, writeCount=0;
semaphore mutex1=1, mutex2=1;
Semaphore readBlock=1,writeBlock=1
```

```
writer() {
  while(TRUE) {
    <other computing>;
    P(mutex2);
      writeCount++;
      if(writeCount == 1)
        P(readBlock);
    V(mutex2);
    P(writeBlock);
      access(resource);
    V(writeBlock);
    P(mutex2)
      writeCount--;
      if(writeCount == 0)
        V(readBlock);
    V(mutex2);
  }
}
```



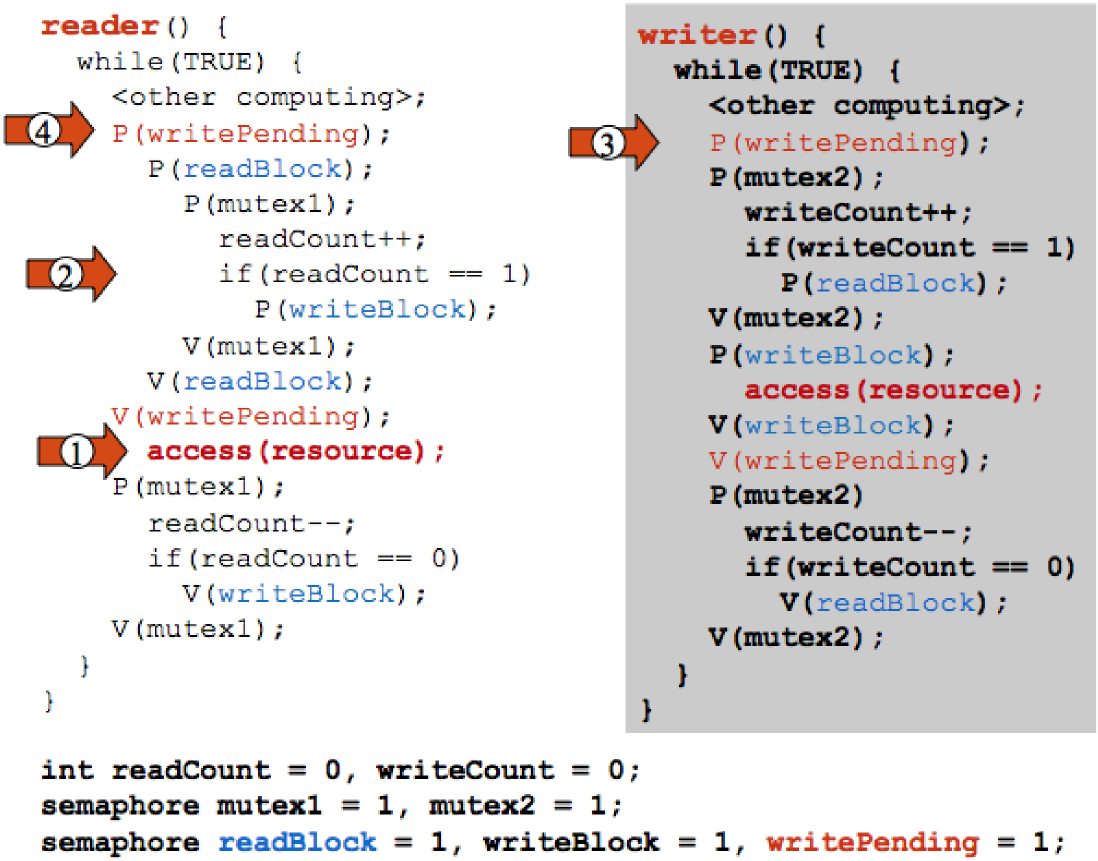
```
reader() {
  while(TRUE) {
    <other computing>;

      P(readBlock);
        P(mutex1);
          readCount++;
          if(readCount == 1)
            P(writeBlock);
        V(mutex1);
      V(readBlock);

      access(resource);
    P(mutex1);
      readCount--;
      if(readCount == 0)
        V(writeBlock);
    V(mutex1);
  }
}

int readCount=0, writeCount=0;
semaphore mutex1=1, mutex2=1;
semaphore readBlock=1,writeBlock=1
```

```
writer() {
  while(TRUE) {
    <other computing>;
    P(mutex2);
      writeCount++;
      if(writeCount == 1)
        P(readBlock);
    V(mutex2);
    P(writeBlock);
      access(resource);
    V(writeBlock);
    P(mutex2)
      writeCount--;
      if(writeCount == 0)
        V(readBlock);
    V(mutex2);
  }
}
```

1) Writers can starve readers

2) “Reads” can be delayed forever

解法3:



```
reader() {
  while(TRUE) {
    <other computing>;
    P(writePending);
      P(readBlock);
        P(mutex1);
          readCount++;
          if(readCount == 1)
            P(writeBlock);
        V(mutex1);
      V(readBlock);
    V(writePending);
      access(resource);
    P(mutex1);
      readCount--;
      if(readCount == 0)
        V(writeBlock);
    V(mutex1);
  }
}
```

```
writer() {
  while(TRUE) {
    <other computing>;
    P(writePending);
    P(mutex2);
      writeCount++;
      if(writeCount == 1)
        P(readBlock);
    V(mutex2);
    P(writeBlock);
      access(resource);
    V(writeBlock);
    V(writePending);
    P(mutex2)
      writeCount--;
      if(writeCount == 0)
        V(readBlock);
    V(mutex2);
  }
}
```

```
int readCount = 0, writeCount = 0;
semaphore mutex1 = 1, mutex2 = 1;
semaphore readBlock = 1, writeBlock = 1, writePending = 1;
```

1) P(writeBlock)保证reader/writer互斥语义
2) mutex1/mutex2 保证对readCount/writeCount的读写同步语义
3) P(writePending)保证了不会出现reader/writer starvation
4) 有没有死锁: 检查锁顺序

Test case
1. R1, W1
2. R1, R2, R3, W1
3. R1, R2, W1, R3
4. R1, R2, W1, W2, R3
….

Class 22 强化练习 3

Q1. Common Element problems
Q1.1 Find common elements in two arrays
A[m]

B[n]
sorted

solution: keep two pointers, one per array, compare the pointee and increase the pointer

```
void findComn(int* A, int* B, int m, int n) {
        for (int i = 0, j=0; i < m, j < n; /**/) {
                if (A[i] == B[j]) {
                        cout << A[i] << " ";
                        ++i;
                        ++j;
                } else if (A[i] > B[j]){
                        ++j;
                } else {
                        ++i;

                }
        }
}
```

Q1.2 Find common elements in 3 sorted arrays

```
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) {
      // for (int i=0, j=0, k=0; i<m, j<n, k<p; ) {
        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)

我们曾经在课上讲过 如何把一个短的string1 替换成一个长的string2
s1   s2
Example: **"_"** -> **"20%"**
url: www.google.com?info=flower_market..l….

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

Solution:
Step 1: 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)

Step 2: 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| (for case 1)

Step 3: iterate through string s, to replace the string s1 with s2;
s = aaaaa ⇒ baa
s1= aaa
s2 = b

```
void Replace(string* s, const string& s1, const string& s2);
// C++
string Replace(string s, string s1, string s2) { // const string&
        string ret = "";
        int len = s.length();
        int len1 = s1.length();
        vector<int> index;
        int i = 0;
        while(i < len){
                // s="aaaa", s1="aaa", s2="x"      s-> "xa"
                if(i + len1 <= len && s.substr(i,len1) == s1) {
                        index.push_back(i);
```

```
            i += len1;
        }
        else{
            i++;
        }
    }
    int start = 0;
    for(int i = 0; i < index.size(); ++i){
        ret += s.substr(start, index[i] - start);
        ret += s2;
        start = index[i] + len1;
    }
    return ret;
}
```

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:
Step1: we deal with all numbers that can **shorten** the string. In the meantime, we count all numbers > 2.
Step2: we deal with all numbers that can **not shorten** the string…..

“a3b1c4d0” → “aaabcccc”

we cannot make it longer first and then shorten it

1. Make it longer

a3b1c4d0” for the numbers >= 2
+1 +2 → total +3
aaab1ccccd0

a1b3
bbb -> size + 1
Step 1: longer
a1bbb

a1b1 -> ab → return substr(0,2)

a1b3
a3b1

run-length encoding

input=a3b1c4d0
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.

input could be a resizable buffer, e.g. StringBuffer

aaabccc -> a3b1c4

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

```
// Java
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);
}

public int getHeight (TreeNode root) {
      if ( root == null ) {
          return 0;
      }
      return Math.max(getHeight(root.left), getHeight(root.right)) + 1;
}
```

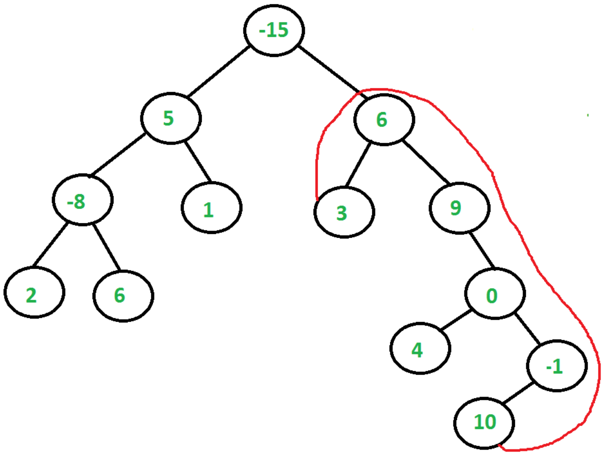
Q3.2 Improve the solution to be O(N)?

```
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).



```
int maxPathSum(struct Node *root == 6, int &result) {
    if (root == NULL) return 0;   // Base case
    // Find maximum sum in left and right subtree. Also find
    // maximum root to leaf sums in left and right subtrees
    // and store them in lLPSum and rLPSum
    int lcost = maxPathSum(root->left, res);
    int rcost = maxPathSum(root->right, res);

    // Find the maximum path sum passing through root
    int curr_sum = lcost + rcost + root->data;
    // Update res (or result) only when needed
    if (result < curr_sum &&
        (root->left && root->right || !root->left && !root->right)) {
        result = curr_sum;
    }
    // Return the maximum (root to leaf path) cost
    return max(lcost, rcost) + root->data;
}

// The main function which returns sum of the maximum
// sum path between two leaves.  This function mainly uses
// maxPathSum()
int FindMaxPathCost(Node *root) {
    int result = 0;
    maxPathSum(root, result);
    return result;
}
```

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;

For all DP problems, we care about

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

First: Primitive idea:

sweden size = n
student 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.

sweden size == n
student size == m

```
index = 0 1 2 3 4 5 6 7
          s t u d e n t
sweden    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

B[] = student;

A[] = sweden;

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

Base case: empty string

repeated computation (subproblem): When end_index + 1;from A[start_index] to A[end_index], we have to repeatedly check A[start_index, end_index] is a substring of B[] or not : such as de->den de is checked more than once.

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
```

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

A == student
i

B == swedenasyt
j

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

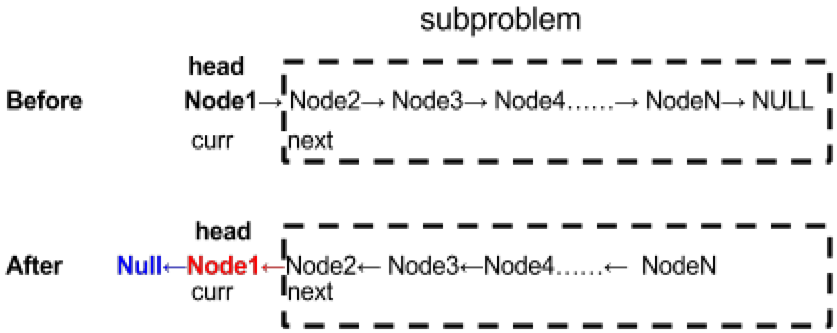
```
M[i][j] = M[i-1][j-1] + 1                 if (A[i]== B[j]);
          max(M[i-1][j], M[i][j-1])       otherwise
```

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

Class 23 强化练习 4

Q1. Reverse linkedlist questions


subproblem
Before
head
Node1→Node2→ Node3→ Node4……→ NodeN→ NULL
curr
next
After
head
Null←Node1←Node2← Node3←Node4……← NodeN
curr
next

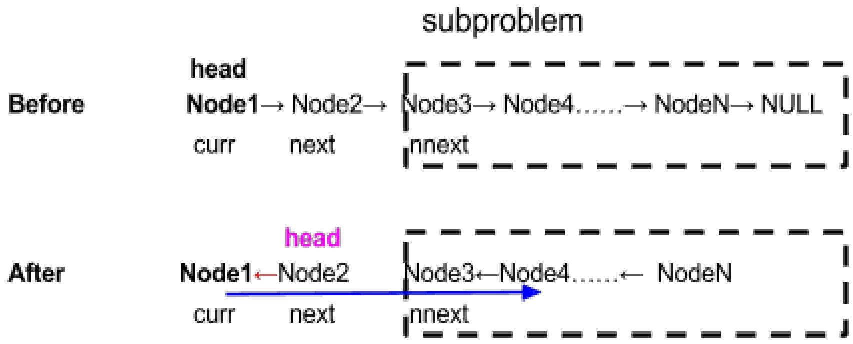
除了subproblem外几处不同？
(1) next→ next = curr; // subproblem head 指向current node;
(2) curr → next = null; // current node’s next is set to Null;

```
public ListNode reverseRecursive(ListNode head) {
     if (head == null || head.next == null) {
          return head;
     }
     ListNode newHead = reverseRecursive(head.next);
     head.next.next = head;
     head.next = null;
     return newHead;
}
```

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

Example 1-> 2 -> 3 -> 4 -> 5 → NULL
prev cur next

output : 2 -> 1 -> 4 -> 3 -> 5→ NULL;


subproblem
Before
head
Node1→ Node2→ Node3→ Node4......→ NodeN→ NULL
curr
next
nnext
After
head
Node1←Node2 Node3←Node4......← NodeN
curr
next
nnext

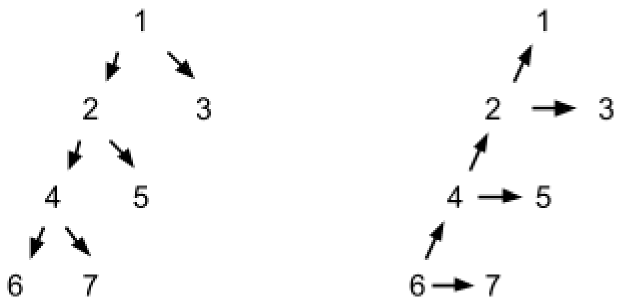
除了subproblem外几处不同？
(1) curr → next = Newhead of the subproblem; // Node 4
(2) next→ next = curr; // "next" becomes the new head;

So, if we can resolve the subproblem first, then we just need to perform (1) and (2) to solve the whole problem :)

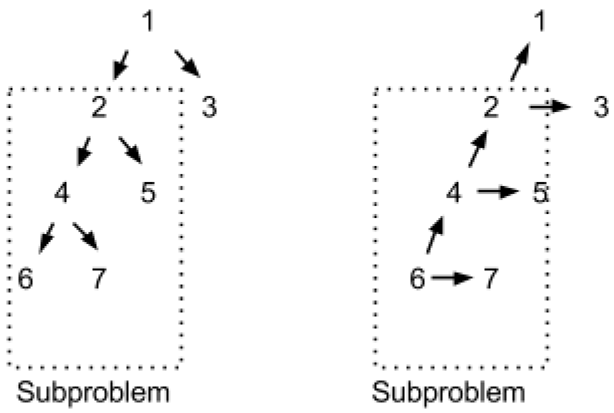
```
Node* PairReverse(Node* head) {
    if (head == NULL || head->next ==NULL) {
        return head; //base case
    }
    Node* rest = PairReverse(head->next->next);
    Node* sec = head->next;
    head->next = rest;
    sec->next = head;
    return sec;
}
```

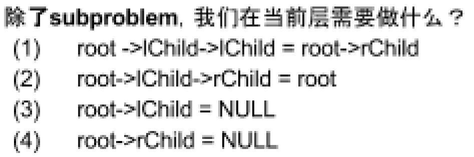
Q1.3 Reverse a binary tree upside down

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?




除了subproblem，我们在当前层需要做什么？

(1) root ->lChild->lChild = root->rChild
(2) root->lChild->rChild = root
(3) root->lChild = NULL
(4) root->rChild = NULL

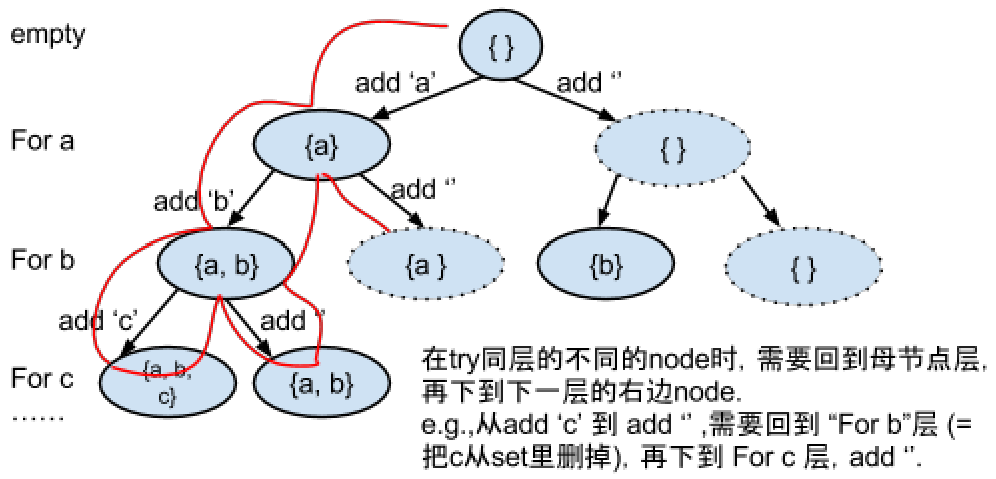
```
Node* convert(Node* root) {
  if (root == NULL || root->left == NULL)
    return root;     // base case
  Node* newRoot = convert(root->lChild);
  root->lChild->lChild = root->rChild;
  root->lChild->rChild = root;
  root->lChild = NULL;
  root->rChild = NULL;
  return newRoot;
}
```

Q2. DFS

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 Print all subset of a set



在try同层的不同的node时，需要回到母节点层，再下到下一层的右边node。
e.g.,从add 'c' 到 add '' ,需要回到 "For b"层 (=把c从set里删掉)，再下到 For c 层，add ''.

DFS to find all subsets of a set {a, b, c} step-by-step.

Q2.2 Print all permutations of a string (with/without duplicated letters)

string input = "abcde";

```
level 1       a                 b             c                d   e
             /| | \           / | | \        /|\\
level 2      bcde              acde         abde

level 3
…

level 5
```

Q2.3 Print all valid combination of coins that can form a certain amount of money (99 cents)
1 5 10 25 cents

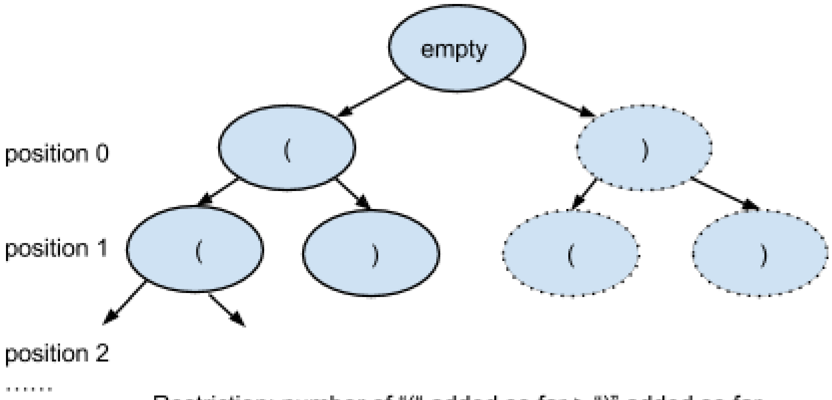
```
                                              99
                               /           /        \           \
(level 0)      0 x25 (99 remaining)       1 x 25(74 remaining)          2 x25       3 x25
          ////  /\\\\\\ (10 branches)                              ///\\\(7 branches)
(level 1) 0x10(99)    1x10 (89)….         9 x10 (9 remaining)
          //
(level 2: 5 cents)
```

(level 3: 1 cent)

Sample solution:

```
static int coin[4] = {25, 10, 5, 1};

void FindCombination(int money_left, int level, int sol[]) {
    if (level == 3) {
        sol[level] = money_left;    // base case
        printsolution(sol);
        return;
    }
    for (int i = 0; i * coin[level] <= money_left; i++) {
        sol[level] = i;
        FindCombination(money_left - coin[level]*i, level+1, sol);
    }
}

int main() {
    int solution[5];
    FindCombination(99, 0, solution);
    return 0;
}
```

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



Restriction: number of "(" added so far > ")" added so far

index 0 1 2 3 4 5

(( )

))

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: ( ) [ ] { }

Review question 2.4,

(1) Restriction 1: there was only one restriction: when putting a new) parentheses, we need to worry about the number of (vs).

(2) Observation (**restriction** 2), whenever we want to put a new right parenthesis)] }, we need to figure out the latest left parenthesis added to the prefix- of the solution (using a stack)

Use a stack to store all left parenthesis added so far

([[{   x   1(2 [3 {   4)5]6 } }]

???????

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

8 levels total

8 states per parent state to try

base case: we finished the 8th row;

recursive rule: iff position(i, j) valid → go to the next row: (i+1)

index   0 1 2 3...

```
// int A[N] stores the current solution on each row, A[N] =  { 1, 3, 5, 7..}
// current_row,  the current row we are inserting a new queen
void EightQueen (int A[N], int current_row) {
        if (current_row == N) {   // base case
                // print A[N];
        }
        for (int i = 0; i < N; i++) { // we can try N columns to insert a new queen on this row
                A[current_row] = i;
                // check whether this configuration is valid or not.
                // using a helper function to check whether A[0. ….current_row-1] conflicts the
                current queen inserted or not;
                if (pass the check) {
                        EightQueen(A[N], current_row + 1);   // recursive rule
                }
        }
```

Q3: 2,3,4-SUM questions

Find X-elements from a (unsorted/sorted) array such that their sum is equal to a target value.

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

(2) **sorted** array

a) 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))**

b) any better idea?

```
                    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;

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

O(n);

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)

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。

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 24 强化练习 5

Q1 Best First Search

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

Define a state **<s_i, s_j>**.

Best First Search:
Initial state: <s1, s2>;
expansion / generation rules: <s_i+1, s_j> OR <s_i, s_j+1>
Termination condition: once we found the first state popped out of the pq is valid

deduplication when generating a new state
<5, 4> -> <5,5> → <5, 6> <6, 5>

<4, 5> -> <5,5>

Helper function to determine whether two words share any common letter

```
bool CheckCommonLetters(const string& s1, const string& s2) {
        // for each letter in s1 we set bit vector 1 according;
        // for each letter in s1 we set bit vector 2 according;
        apple
        abc
0000 0000  0000 0000  0000 0100  0000 1000  0000 0000  0000 0000  0000 0000  0001
0001
&
0000 0000  0000 0000  0000 0000  0000 0000  0000 0000  0000 0000  0000 0000  0000
0111
--------------
==  OR !=0
    return true if the result is 0  , and return false otherwise;
}
```

Q1.2 How to find the k-th **smallest** value of f in the f(x,y,z) = 3^x * 5^y * 7^z (int x > 0, y>0, z>0).

Best First Search:

(1) Initial state: <x=1, y=1 , z=1>
(2) expansion / generation rules: expand<i, j, k> → generate <i+1, j , k>, <i, j+1, k>, <i, j, k>
(3) Termination condition: when the k-th state is popped out of the heap, STOP!

Deduplication (using hash_table) is needed

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 3D space. (1) How to find the coordinates of the points that is k-th closest to <0,0,0>?

Best First Search:

(1) Initial state: <x=A[0], y=B[0] , z=C[0]>
(2) expansion / generation rules: expand<i, j, k> → generate <i+1, j , k>, <i, j+1, k>, <i, j, k>
(3) Termination condition: when the k-th state is popped out of the heap, STOP!

Deduplication (using hash_table) is needed

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.

Assumption: n x n 2d array to represent the gym,
Given the coordinates of each equipments and obstacles;

000**e**0000**5/12/5**

000011000

0**e**0001000

0000010**e**0

M1: perform Dijsktra's algorithm from each empty cell
perform how many times of Dijkstra's algorithm O(n^2)
Dijkstra's algorithm time complexity on grids O(n^2 log(N))
Total time complexity == O(n^4 log(N))

M2: perform Dijstra's algorithm from each equipment
perform how many times of Dijkstra's algorithm : k
Dijkstra's algorithm time complexity on grids O(n^2 log(N))
finally, we sum up all values in each cell O(k n^2)

Total time complexity = O(k n^2 log(N) + **k n^2**) = **O(k n^2 log(N))**

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: what is the data range.
if [0-100] bucket sort
Assumption 3: ascending or descending???

Simon:

Step1: read 0.5GB data by 0.5 GB
Step2: sort (**quicksort**) 0.5 GB data using 0.5 GB memory and write it back the HD
Step3: you have now 160 chunks of data, sorted.
Step4, use a k-way merge (heap) to read in data one by one.

chunk1 (0.5GB) [5MB]
chunk2 (0.5GB) [5MB]--> MIN_Heap → buffer [5MB] the element in an arrayList→ write to HD in one shot
…

chunk160 [5MB]
===

Chunk 1() Chunk 12()
Chunk 2() Chunk 14()
FINAL Chunk(A)
Chunk 3() Chunk 34()
Chunk 4()

Q3 (string conversion)

Q3.1 “A1B2 C3D4” ==> “ABCD1234”

A1B2 C3D4 AB12 CD34

A1 B2

A 1

A1 B2

AB12

Q3.2 **“**ABCD1234” ==> “A1B2C3D4”

I LOVE YAHOO problem is here!!

Hint: “AB | **CD | 12** | 34” ==> “**A1B2**C3D4”

four chunks 1 2 3 4

AB **DC** 21 34

AB 12 | **CD** 34

log(n) row

A1 B2

ABC | **12**3

C 21

AB12 | C3

Critical details: guarantee size of Chunk 1 == Chunk 3;

O(nlog(n))

M2: “AB | **CD | 12** | 34”

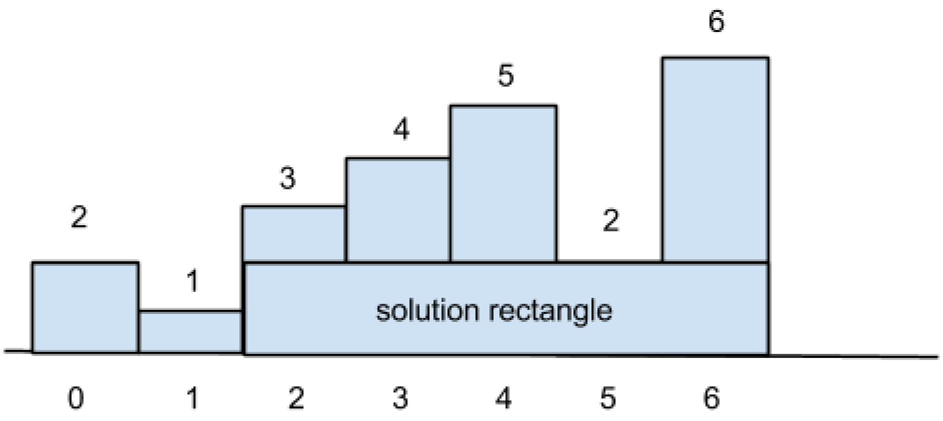
A 123 BCD 4

A1 BC 23 D4

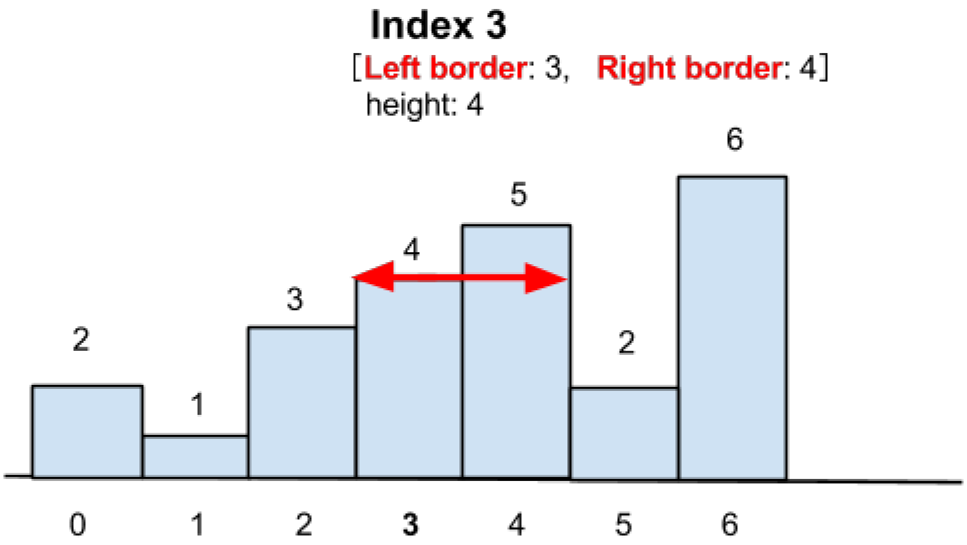
A1B 2 C3 D4 **O(n) row** n → **O(n^2)**

Q4 Histogram questions (直方图问题)

Q4 .1直方图中找最大矩形



Primitive idea: O(n*n) ⇒ O(N^2) 中心开花



Optimal solution: O(n)??

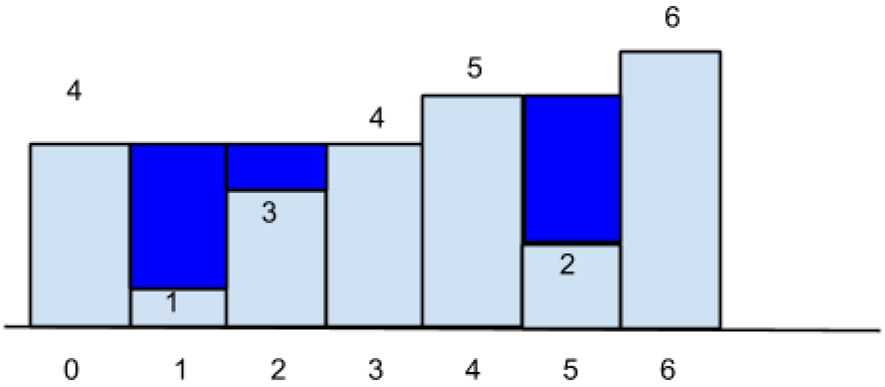
Use a stack to store all the indices of the columns that form an ascending order

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);

Q4 .2 直方图下雨接水问题



index =   0 1, 2,;
LEFT_MAX[N] = { 4 4 4   4 }

Primitive solution: 中间开花 **O(n^2)**
木桶理论：盯住最短板
Naive solution: **O(n^2)** 中间开花，从x 左右走， 找到x 左右两侧的最高值 left_max and right_max,   then the water can be stored at x's top is **min(left_max, right_max) - A[x]**

Solution 1 (Good):

For each index, calculate what the tallest point in the histogram preceding that index is. This is easy and O(n) because **max_left[n] = max (max_left[n-1], histogram[n])**. Now, also for each index, calculate a similar **max_right** array (calculate from the back using **max_right[n] = max (histogram[n], max_right[n+1])**). Now, for each index n**, water_height[n] = min (max_left[n], max_right[n]).** At this point, loop over the indices i = 0...n-2 and do:

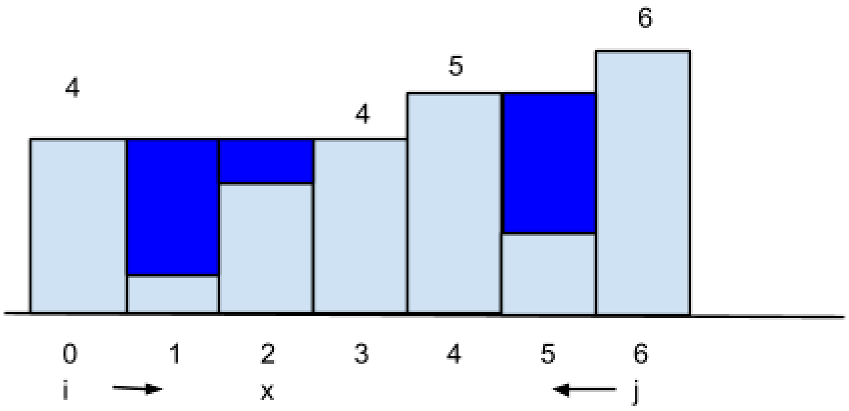
Time = O(2N)

Space = O(N)

Solution 2 (better):

time O(1*n)

space O(1)



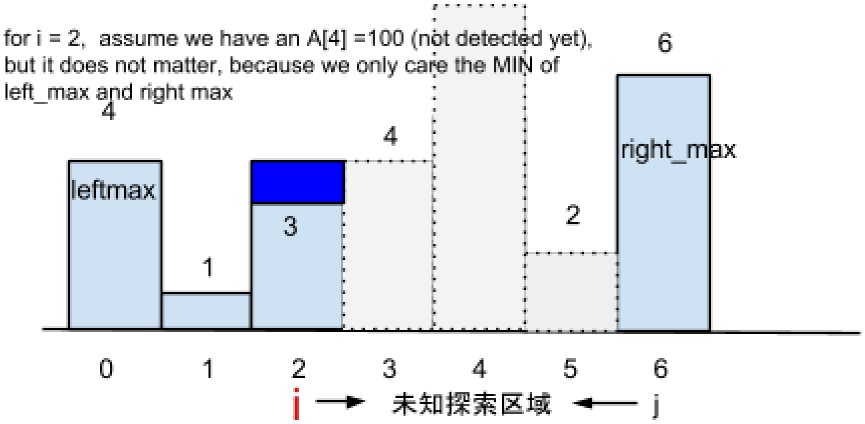
For an index x, how much water can we store in x-th indexed place?

It depends on **min** (highest bar on the left of x,
highest bar on the right of x) - **height of x**

Initially, we set left index i = 0; right index j = n-1, 左右对进。

initialization:

left_max = A[i],
right_max = A[n-1]


for i = 2, assume we have an A[4] =100 (not detected yet), but it does not matter, because we only care the MIN of left_max and right max

For an index i, how much water can we store in i-th indexed place?

It depends on **min** (highest bar on the left of i (including i),
highest bar on the right of i (including i))
- **height of i**

Initially, we set left index i = 0; right index j = n-1, 左右对进。

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， 从i → i+1 分析问题
case 1: if left_max < right_max, we already know the water stored 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 --;

Code:
```
public int trap(int[] A){
  int i = 0;
  int j = A.length - 1;
  int max = 0;
  int leftmax = 0;
  int rightmax = 0;
  while (i <= j) {
     leftmax = Math.max(leftmax, A[i]);
     rightmax = Math.max(rightmax,A[j]);
     if (leftmax < rightmax){
         max += (leftmax - A[i]);
         i++;
     } else{
         max += (rightmax - A[j]);
         j--;
     }
  }
  return max;
}
```

Class 25 强化练习 6

Q1. Binary Search Related Problems, variants of BS
1) two sorted integer arrays, how to find the k-th smallest element from them.

Example Input:

```
A[] = {2, 5, 7, 10, 13}
B[] = {1, 3, 4, 13, 20, 29}
k = 5
```

Output: 5

M1: merge sort, stop at the kth element, time O(k), space O(1)

A[i], B[j]
case 1) B[j-1]<A[i]<B[j], i+j=k-1, i=k/2, j=k-k/2-1 // this step is not strictly necessary, but could improve efficiency of the algorithm in certain cases.
case 2) A[i]>B[j], recursively select k/2th element from A[0..n] and B[k-k/2...m]

A[i]<B[j-1]<B[j] ⇒ B[j]>A[i]
A[i-1]<B[j]<A[i]

Time: O(logK) Space: O(1)

iteration 1) k=5, i=2, j=2, A[i] > B[j] (A[2]=7 B[2]=4 B[1]=3)
=> we know that B[0]...B[2] won't be the 5th element, then we can continue search for k/2=2nd smallest element in B[3...m], A[0...n]
iteration 2) k=2 A={2,5,7,10,13} B={13,20,29} (B[3...m] of original array), i=1, j=0
=> A[0]=2, A[1]=5, B[0]=13, A[1] < B[0], A[1] is the result

```
// How to find the kth smallest element from two sorted arrays.
//
// To reduce coding complexity, we omit the optimization discussed in case 1) above.
// Instead, we simply keep eliminating k/2 elements in each iteration, and end the iteration
// either when k==1 or one of the sub-array is empty.
int imax = std::numeric_limits<int>::max();
int findKthSmall(const vector<int>& a, int a_left,
                 const vector<int>& b, int b_left, int k) {
 if (k == 1)
     return min(a[a_left], b[b_left]);  // base case 1
 if(a_left >= a.size())
     return b[b_left + k - 1];   // if nothing left in a;
 if(b_left >= b.size())
     return a[a_left + k - 1];   // if nothing left in b;
 // We set array length to k/2 no matter k is odd or even. We simply eliminate k/2 elements in
 // each iteration.
 int a_kth_value = a_left + k/2 - 1 < a.size() ? a[a_left + k/2 - 1]
```

```
    : imax;  // why set to imax?
             // because in such case, a[] and b[] does not have enough length to cover k elements.
             // we want to eliminate as much elements as possible in each iteration, which is the
             // whole k/2 elements in b[].
             //
             // This is an optimization to speed up the search process.
int b_kth_value = b_left + k/2 - 1 < b.size() ? b[b_left + k/2 - 1]
    : imax;
if (a_kth_value < b_kth_value) {
    return findKthSmall(a, a_left + k/2, b, b_left, k - k/2);
    // Notice that, we use k-k/2 instead of k/2 here since k/2 may not equal to (k-k/2).
} else {
    return findKthSmall(a, a_left, b, b_left + k/2, k - k/2);
}
}
```

Main:

```
int kthSmallest(const vector<int>& a, const vector<int>& b, int k) {
    return findKthSmall(a, 0, b, 0, k);
}
```

Q2 Given a number x, how to get the **hexadecimal** representation of the number in string type?

E.g **29 ⇒ 0x1D**

```
string hexadecimal_representation(int num){
        const char[] chars = “0123456789abcdef”;
        string s = “”;
        if(num == 0){
                return ”0x0”;
        }
        while (num > 0) {
                s += chars[num % 16];
                num = num / 16;
        }
        return s;
}
```

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

array size: N, window size K

M1: sort all numbers in the window, and return the result
Time: O(K*N)

M2: Time: O(N)
1 3 2

3 2 **5**

stack
queue
deque

[]
[] + 1 => [1]
[1] + 3 => [] + 3 => [3]
[3] + 2 => [3, 2]
[3, 2] + 5 => [] + 5 => [5]
[5] + 8 => [] + 8 => [8]
[8] + 9 => .. => [9]
[9] + 4 => [9, 4]
7→ 2
[9, 4] + 2 => [9, 4, 2]

[9, 4] + 7 => [9] + 7 => [9, 7]
[9, 7] => [7], [7] + 3 => [7, 3]   // the index of 9 is already out of range

Notice that, the elements in the deque is always sorted.

Q4 How to design a LRU cache?

double linked list + hash table

Case 1: cache miss: put the new data in the front of the queue. evicted the end of the queue.
Case 2: cache hit: put this data hit in the front of the queue.

E.g., each data cached is a URL

Cache content (in doubled linked list):
Url8 <->Url2< -> Url3…. <-> **Url7**<-->Url 9….
prev next

Hash_table <key = url, value = pointer to the element in the double linked list>
www.cnn.com & url8

```
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;
        }
    }
};
```

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？

M1: hash table Time: O(N), Space: worst case O(N), best case O(1)
M2: sort, Time: O(NlogN), Space: worst case O(N), average O(logN)
heap sort, in place
M3: Time: O(N), Space: O(1)

Example input:

```
1 1 1 3 3 7 7 3 3 3 7 3 3
```

Output:
3

use a counter to record the count of the current candidate.

<current_candidate, current_candidate’s count>

```
Whenever we scan a new number x,
        if x == current_candidate:
                current_candidate’s count ++
        else:
                current_candidate’s count--
                if  current_candidate’s count == 0:
                        current_candidate = x
                        current_candidate’s count = 1;
```

```
1 1 1 3 3 7 7 3 3 3 7 3 3
1 <1, 1>
1 <1, 2>
1 <1, 3>
3 <1, 2>
3 <1, 1>
7 <1, 0> => <7, 1>
7 <7, 2>
3 <7, 1>
3 <7, 0> => <3, 1>
3 <3, 2>
7 <3, 1>
3 <3, 2>
3 <3, 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”

A = ab
B = cd
C = acbd / bcad

DP:

M[i][j] represents we have tried to use first i letters of A[] and first j letters of B[] to form the first i+j letter of C, that is `C[1]` to `C[i+j]`

```
M[i][j] =  M[i-1][j] && A[i] == C[i+j] ||  (同一列，B[ ]不取新字母)
            M[i][j-1] && B[j] == C[i+j]     (同一行， A[ ]不取新字母)
```

string C = “acde abcd”

```
index 0 1 2 3 4
   B=   a c d e
A
0     t t t t t
1 a   t f f f t
2 b   f f f f t
3 c   f f f f t
4 d   f f f f t
```

```
Q7: Design a RateLimiter.
Protect a limited resource from being overloaded.

1 1 1 | 5 5 3 |
```

Class 26 强化练习 7

Q1 Reconstruct a BST by using xxx-order and in-order traversal sequences

Q1.1 How to reconstruct a BST 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**

Base case : only 1 element left.

Solution:
Get the first element from pre-order (=10), and find the index of 10 in in-order sequence. Assume its inorder index = mid;

10   3

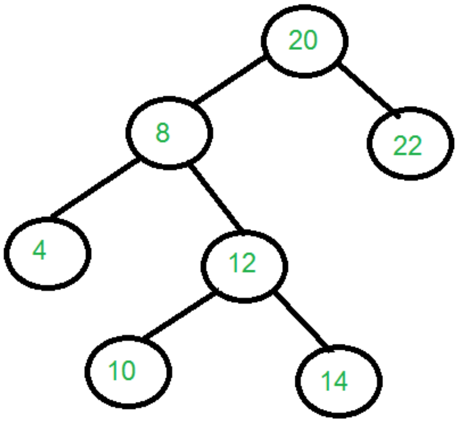
optimization: we can build a **hash_table<value, index_in_inorder_aray>** for in-order sequence, so that we can quickly find the index of an element in the in-order sequence.

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 BST with **post-order** and **in-order** sequences of all nodes.

Q1.3 Construct a 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};

8   xx   4 12   10 14

in-order left [] = {4, 8, 10, 12, 14,}   8 is the subtree root node
in-order right[] = {22}

level-order left[] = { 8, 4, 12, 10, 14 };   8 is the subtree root nod
level-order right[] = { 22   }

O(n^2)

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>
P……

n points
Runtian M1:

p1:   iterate over p2….pn
hash_table <key = **slope**, value = **counter**>
count how many points can form the same line as p1.

p2:   iterate over p3… pn
hash_table.clear();
hash_table <key = **slope**, value = **counter**>

p3…..
corner case: **slope can be infinity**

M2:   Method: y = **ax** + **b**
for for loops   for every pair of points.

record how many points share the same line function defined by   <a, b> by using a hash_table

hash_table <key = <a, b>, value = set of points>

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.

<x1 y1> <x2, y2> <x3, y3> …   <xn, yn>

x1 y1   x2 y2

(y2 - y1) / (x2 - x1) > 0, given that x2 > x1,   we must have y2 > y1.

Step1, sort all points in an ascending order based on their x-coordinates
Step2: look for the **longest increasing subsequence** based their y-coordinates.

xi =   in an ascending order
yi =   1 3 5 7 9 ….

Q3 How to design a search suggestion system.

E.g., **football** -> ticket
season
player
……

System Design: what should we ask instead of giving answers at the very beginning.
Make assumptions / ask questions:

Scalability: how many users are going to use this system. (10 million users)
Who are the users? USA Germany Japan (Localization??)
How much data do we have?
language/country football (GB→ zuqiu, USA→ gan lan qiu)
personalized → signed in / out

Xie Xin:

1. how often should we update the system.
2.

What is the most important factor/metrics to order all the words follow the key word?
Proposal 1: <football **ticket**> 1.8 billion
<football **season**> 1.2 billion
…..

Assume we have 3 days of data /logs from **Germany** 10000 clusters.

mapreduce to count what????:
<key= key word "football", value = " < f1= subsequent word , second = frequency > " >

how to represent the search box????
football → 1. ticket 2. season 3.….

use pre-fix tree (trie) to index all the keyword
http://en.wikipedia.org/wiki/Trie

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.)

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
```

1 : wall cell
2:   empty cell

Search algorithms: BFS **DFS** Best-First Search
Jiang He: use some kind of search algorithms to generate a tree on the maze.

```
0 0 0 1 0
1 1 0 1 0
0 1 0 1 0
0 1 0 1 0
0 0 0 0 0
```

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 2 2 2 2 2 1 1 3 3 1 1
1 1 2 1 1 1 2 1 1 3 1 1 1
2 2 2 1 0 1 2 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
```

n x n matrix
BFS:

convex shape
Optimization: binary search variant 2.0

Q6 Given a 2D array A[8][8] with all positive numbers if we take a number a[i][j], then we cannot take its 8 neighboring cells. How should we take these numbers to make their sum as **large** as possible.

```
0 1 2 1 7 -1 8 9    config9:   0 0 0 0 0 0 1 1 → take 8 and 9 values out
x x x x x x x x
g f e 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 1 1 1 1
1 1 1 1 1 1 1 1

n == 8
O(2^n)
  1001 0101
  0100 1010
>> and & ____________
           0


M[I][J] represents  the max sum for the configuration j at row i,
config 0:   0000 0000
config 1:   0000 0001
config 2:   0000 0010
config 3:   0000 0100
….
config 50;
```

M[0][0], **M[0][1]** **M[0][2]……** **M[0][49]**

\ | / ...

M[1][0] **M[1][1] = c[1][1] + max(M[0][i])** **(for all valid configuration i that is not conflicting with config [1])**

……

M[2][7] = C[2][7] + max(M[1][i]) (for all valid config i that is not conflicting with config[7])

configuration nunber == k (50)

line number n (8)

Finally, we just need to return max **(M[7][i])**

Time complexity = O(nk k n)

g f e 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 1 1 1 1
1 1 1 1 1 1 1 1

1. **DP**
2. Best First Search

Subproblem is defined as follows:
Let’s define a **configuration** for a row
0 0 0 0 0 0 0 0   How many configurations per row? 2^8 = 256 different configuration.

0 0 0 0 0 0 1 1   invalid configuration
1 0 0 0 0 0 1 0   valid configuration

~50 valid configuration for each row: e.g.
valid:
config 0:   0000 0000
config 1:   0000 0001
config 2:   0000 0010
config 3:   0000 0100
….
config 50;

If we define the **element** to be the valid **configuration** for each row
M[8] [50]

For the **first** row
Configuration at row 0, there are 50 valid configurations on this row
C00 C01 C02……… C0,49 → **M[0][0] , M[0][1]... M[0][49].**

for the 2nd row

Class 27 OO-Design 习题课

I. Design a generic deck of cards. Explain how you would subclass the data structure to implement blackjack.

Q. Start from designing **a generic deck of cards:**

Step 1. Understand the question. Figure out what classes we need to define and their relationships.

棋牌游戏类OOD
- 游戏道具
- 游戏状态
- 游戏规则
- 游戏流程

Card:
- Value
- Suit

Deck
- Card[] or List<Card>

Hand
- Card[] or List<Card>

Step 2. Functionalities of main classes and their public interfaces.

Card
- getValue
- getSuit

Deck
- 洗牌: shuffle
- 发牌: dealCard

Hand
- 决定是否抓牌: continuePlaying
- 抓牌: addCard
- 得到当前有什么牌: getCards
- 得分: score

Step 3. Define classes

```
public enum Suit {
    Club (0, “black”),
    Diamond (1, “red”),
    Heart (2, “red”),
    Spade (3, “black”);

    private final int value;
    private final String color;

    private Suit(int v, String c) {
        value = v;
        color = c;
    }

    public int getValue() {
        return value;
    }

    public String getColor() {
        return color;
    }

    public static Suit getSuitFromValue(int value) {
        switch (value) {
        case 0:
            return Suit.Club;
        case 1:
            return Suit.Diamond;
        case 2:
            return Suit.Heart;
        case 3:
            return Suit.Spade;
        default:
                return null;
        }
    }
}

public abstract class Card {
    /* number or face that's on card - a number 2 through 10,
     * or 11 for Jack, 12 for Queen, 13 for King, or 1 for Ace
     */
    protected int faceValue;
    protected Suit suit;
```

```
        public Card(int c, Suit s) {
                faceValue = c;
                suit = s;
        }

        public abstract int value();

        /*
        public int value() {
          return faceValue;
        }
        */

        public Suit suit() {
                return suit;
        }
}

class Deck {
  private final List<Card> cards;

  public Deck() {
    // TODO: generate 52 cards, insert them into #cards
  }

  public void shuffle() {
      // TODO
  }

  public Card dealCard() {

  }

  public List<Card> dealCards() {
  }
}

public abstract class Hand {  // Hand will be extended to BlackJackHand, or
TexasHoldmHand
  private List<Card> cards;  // cards in hand

  public void addCard(Card card) {
    cards.add(card);
  }

  public abstract int score();
}
```

==

Q. Blackjack?

1. list rules
2. capture game status
3. model game procedure

game status s1 --- action + rule ---> s2

BlackJackGame
BlackJackHand extend Hand
BlackJackCard extend Card

How to describe the current game status

- Hand[]
- Deck

Hand h1,	Hand h2,	Deck
{}	{}	{full….}
		shuffle
{c1, c2}	{c3, c4}	{full - c1~4}

apply rule: if there is/are blackjack
action: h1/h2 decide if continue, and if so call addCard

{c1, c2, c5} {c3, c4, c6} {full - c1~6}

apply rule: check if h1/h2 busted
action…..

…..

apply rule to check/compare scores

```
public class BlackJackCard extends Card {
  public BlackJackCard(int c, Suit s) {
    super(c, s);
  }

  @Override
  public int value() {
    if (faceValue >= 11 && faceValue <= 13) { // Face card
      return 10;
    } else { // Number card or Ace
      return faceValue;
    }
  }

  public int minValue() {
    if (isAce()) { // Ace
      return 1;
    } else {
      return value();
    }
  }

  public int maxValue() {
    if (isAce()) { // Ace
      return 11;
    } else {
      return value();
    }
  }

  public boolean isAce() {
    return faceValue == 1;
  }

  public boolean isFaceCard() {
    return faceValue >= 11 && faceValue <= 13;
  }
}
-----------------------------------------------------------------------------
public class BlackJackHand extends Hand<BlackJackCard> {
  @Override
  public int score() {
    ArrayList<Integer> scores = possibleScores();
    int maxUnder = Integer.MIN_VALUE;
    int minOver = Integer.MAX_VALUE;
    for (int score : scores) {
      if (score > 21 && score < minOver) {
        minOver = score;
      } else if (score <= 21 && score > maxUnder) {
        maxUnder = score;
      }
    }
    return maxUnder == Integer.MIN_VALUE ? minOver : maxUnder;
  }

  private ArrayList<Integer> possibleScores() {
    ArrayList<Integer> scores = new ArrayList<Integer>();
    if (cards.size() == 0) {
      return scores;
    }
    for (BlackJackCard card : cards) {
      addCardToScoreList(card, scores);
    }
    return scores;
  }

  private void addCardToScoreList(BlackJackCard card, ArrayList<Integer> scores) {
    if (scores.size() == 0) {
      scores.add(0);
    }
    int length = scores.size();
    for (int i = 0; i < length; i++) {
      int score = scores.get(i);
      scores.set(i, score + card.minValue());
      if (card.minValue() != card.maxValue()) {
        scores.add(score + card.maxValue());
      }
    }
  }

  public boolean busted() {
    return score() > 21;
  }

  public boolean is21() {
    return score() == 21;
  }
```

```
  public boolean isBlackJack() {
    if (cards.size() != 2) {
      return false;
    }
    BlackJackCard first = cards.get(0);
    BlackJackCard second = cards.get(1);
    return (first.isAce() && second.isFaceCard())
        || (second.isAce() && first.isFaceCard());
  }
}
```

==

Q. How to design a blackjack game automator?

BlackJackGameAutomator

- **Deck<BlackJackCard> deck**
- **BlackJackHand[] hands // multiple players**
- rules
    - Everybody gets 2 cards initially
    - If there is blackjack, declare winner(s)
    - Otherwise continue. In each round, each player continue play before its score reaches or exceeds 16.

game procedure

deck initialization (shuffle) → initial deal → check blackjack → play hands → check winner

define a method for each step of the game procedure

```
public class BlackJackGameAutomator {
  private Deck<BlackJackCard> deck;
  private BlackJackHand[] hands;
  private static final int HIT_UNTIL = 16;

  public BlackJackGameAutomator(int numPlayers) {
    hands = new BlackJackHand[numPlayers];
    for (int i = 0; i < numPlayers; i++) {
      hands[i] = new BlackJackHand();
    }
  }

  public boolean dealInitial() {
```

```
  for (BlackJackHand hand : hands) {
    BlackJackCard card1 = deck.dealCard();
    BlackJackCard card2 = deck.dealCard();
    if (card1 == null || card2 == null) {
      return false;
    }
    hand.addCard(card1);
    hand.addCard(card2);
  }
  return true;
}

public ArrayList<Integer> getBlackJacks() {
  ArrayList<Integer> winners = new ArrayList<Integer>();
  for (int i = 0; i < hands.length; i++) {
    if (hands[i].isBlackJack()) {
      winners.add(i);
    }
  }
  return winners;
}

public boolean playHand(int i) {
  BlackJackHand hand = hands[i];
  return playHand(hand);
}

public boolean playHand(BlackJackHand hand) {
  while (hand.score() < HIT_UNTIL) {
    BlackJackCard card = deck.dealCard();
    if (card == null) {
      return false;
    }
    hand.addCard(card);
  }
  return true;
}

public boolean playAllHands() {
  for (BlackJackHand hand : hands) {
    if (!playHand(hand)) {
      return false;
    }
  }
  return true;
}

public ArrayList<Integer> getWinners() {
  ArrayList<Integer> winners = new ArrayList<Integer>();
  int winningScore = 0;
  for (int i = 0; i < hands.length; i++) {
    BlackJackHand hand = hands[i];
    if (!hand.busted()) {
      if (hand.score() > winningScore) {
        winningScore = hand.score();
        winners.clear();
        winners.add(i);
      } else if (hand.score() == winningScore) {
        winners.add(i);
      }
    }
  }
  return winners;
}

public void initializeDeck() {
  ArrayList<BlackJackCard> cards = new ArrayList<BlackJackCard>();
  for (int i = 1; i <= 13; i++) {
    for (int j = 0; j <= 3; j++) {
      Suit suit = Suit.getSuitFromValue(j);
      BlackJackCard card = new BlackJackCard(i, suit);
      cards.add(card);
    }
  }

  deck = new Deck<BlackJackCard>();
  deck.setDeckOfCards(cards);
  deck.shuffle();
}

public void printHandsAndScore() {
  for (int i = 0; i < hands.length; i++) {
    System.out.print("Hand " + i + " (" + hands[i].score() + "): ");
    hands[i].print();
    System.out.println("");
  }
}
}
```

==

Last Step: Write a main function to demonstrate your code

```
  public static void main(String[] args) {
      int numHands = 5;

      BlackJackGameAutomator automator = new BlackJackGameAutomator(numHands);
      automator.initializeDeck();
      boolean success = automator.dealInitial();
      if (!success) {
        System.out.println("Error. Out of cards.");
      } else {
        System.out.println("-- Initial --");
        automator.printHandsAndScore();
        ArrayList<Integer> blackjacks = automator.getBlackJacks();
        if (blackjacks.size() > 0) {
          System.out.print("Blackjack at ");
          for (int i : blackjacks) {
            System.out.print(i + ", ");
          }
          System.out.println("");
        } else {
          success = automator.playAllHands();
          if (!success) {
            System.out.println("Error. Out of cards.");
          } else {
          System.out.println("\n-- Completed Game --");
          automator.printHandsAndScore();
          ArrayList<Integer> winners = automator.getWinners();
          if (winners.size() > 0) {
            System.out.print("Winners: ");
            for (int i : winners) {
              System.out.print(i + ", ");
            }
            System.out.println("");
          } else {
            System.out.println("Draw. All players have busted.");
          }
        }
      }
    }
  }
```

II. Design an in-memory file system.

class Node

- name
- Folder parent
- last modification time
- creation time

class Folder extends Node

- Node[] children

class File extends Node

- type

class FileSystem

- resolvePath (/foo/bar --> Node)
- createFile
- mkdir
- delete

```
public abstract class Entry { // Node
  protected Directory parent;
  protected long created;
  protected long lastUpdated;
  protected long lastAccessed;
  protected String name;

  public Entry(String n, Directory p) {
    name = n;
    parent = p;
    created = System.currentTimeMillis();
  }

  public boolean delete() {
    if (parent == null) {
      return false;
    }
    return parent.deleteEntry(this);
  }

  public abstract int size();

  public String getFullPath() {
    if (parent == null) {
      return name;
    } else {
      return parent.getFullPath() + "/" + name;
    }
  }

  public long getCreationTime() {
    return created;
  }

  public long getLastUpdatedTime() {
    return lastUpdated;
  }

  public long getLastAccessedTime() {
    return lastAccessed;
  }

  public void changeName(String n) {
    name = n;
  }

  public String getName() {
    return name;
  }
}
--------------------------------------------------------------------------------
public class File extends Entry {
  private String content;
  private int size;

  public File(String n, Directory p, int sz) {
    super(n, p);
    size = sz;
  }

  public int size() {
    return size;
  }

  public String getContents() {
    return content;
  }

  public void setContents(String c) {
    content = c;
  }
}
-------------------------------------------------------------------------------
public class Directory extends Entry {
  protected ArrayList<Entry> contents;

  public Directory(String n, Directory p) {
    super(n, p);
    contents = new ArrayList<Entry>();
  }

  protected ArrayList<Entry> getContents() {
    return contents;
  }

  public int size() {
    int size = 0;
    for (Entry e : contents) {
      size += e.size();
    }
    return size;
  }

  public int numberOfFiles() {  // mapping to tree
    int count = 0;
    for (Entry e : contents) {
      if (e instanceof Directory) {
        count++; // Directory counts as a file
        Directory d = (Directory) e;
        count += d.numberOfFiles();
      } else if (e instanceof File) {
        count++;
      }
    }
    return count;
  }

  public boolean deleteEntry(Entry entry) {
    return contents.remove(entry);
  }

  public void addEntry(Entry entry) {
    contents.add(entry);
  }
}

--------------------------------------------------------------------------------
public class FileSystem {

  private final Directory root;

  public FileSystem() {
    root = new Directory(“/”, null);
  }

  public List<Entry> resolve(String path) {
    // TODO: write program to resolve path like “/foo/bar/baz”
    assert path.startsWith("/");
    String[] components = path.substring(1).split("/");
    List<Entry> entries = new ArrayList<Entry>(components.length + 1);
    entries.add(root);

    Entry entry = root;
    for (String component : components) {
      if (entry == null || !(entry instanceof Directory)) {
        throw new IllegalArgumentException("invalid path: " + path);
      }
      if (!component.isEmpty()) {
        entry = ((Directory) entry).getChild(component);
      }
    }
    return entries;
  }

  public void mkdir(String path) {
    // TODO: create a new directory with the given path
    List<Entry> entries = resolve(path);
    if (entries.get(entries.size() - 1) != null) {
      throw new IllegalArgumentException("Directory already exists: " + path);
    }
    String[] components = path.split("/");
    final String dirName = components[components.length - 1];
    final Directory parent = (Directory) entries.get(entries.size() - 2);
    Directory newDir = new Directory(dirName, parent);
    parent.addEntry(newDir);
  }

  public void createFile(String path) {
    // TODO: create a new file with the given path
  }

  public void delete(String path) {
    // TODO: delete the file/directory with the given path
  }

  public Entry[] list(String path) {
    // TODO: list all the immediate children of the directory specified by the  given path
    return null;
  }

  public int count() {
    // TODO: return the total number of files/directories in the FileSystem
  }
}
```

III. Design a chat server

Functionalities:
Use your experience using QQ, MSN, GTalk, 微信!

User, User status (enum), User management
Message, Conversation
Private Chat, Group Chat

Q: how to model a user’s basic information

- id: 用来在系统内部区分用户
- full name
- account name
- contact list

Q: how to describe a user involving in chat(s)?

- conversation
    - participants: User[]
    - Existing messages: Message[]
    - private chats: only 2 participants
    - group chats: >= 2 participants

Q: how to manage all the users?

- UserManager
    - id-user mapping
    - current online users

Class 28 System Design 3

1 Design a site similar to tinyurl.com (URL shortener service)

http://bit.ly/aBc1d2e
http://goo.gl/
http://tinyurl.com/abcde =>
https://www.google.com/search?q=laioffer&oq=laioffer+&aqs=chrome..69i57j69i60l3j0l2.2208j0j1&sourceid=chrome&es_sm=93&ie=UTF-8

Features:
o long URL => shortened URL
o shortened URL => long URL

o uniqueness? the same long URL always maps to the same shortened URL
    o without uniqueness requirement, we do not need to check if the long URL exists
    o with it, we need to maintain the mapping from long URL (e.g. its hash) to short URL

o how many different URLs? 10B
o how many bits we need in the integer?
    o 2^32, 2^48, 2^64
long URL => integer (hash, sequence num) => short string

seq num / integer => short string
    o pre-built a series of short URLs, i.e. permutations
        o compute chunk by chunk rather than compute all at once (hundreds of millions!)
        o save internal states of permutation generator to storage, and read it back when computing next chunk
    o BaseXX encoding
        o hex: 0-9a-f
        o Base62: [a-zA-Z0-9]
        o Random Base56 encoding
    o Randomly select a char for each position, then check if there is a collision

o how many chars? [a-zA-Z0-9]
    o exclude il1oO0, 62-6=56

o 56^6=30B

Extended features
o 301 or 302 redirection?
o track / provide stats of the click? may not work very well due to 301 redirection
o custom / readable URL? e.g. tiny.cn/man-in-the-middle-attack
o path / prefix in the URL, e.g. goo.gl/maps/Ae9DQ
o recycle unused URLs
o bad words detection
o XSRF protection

Continue: Design the serving for the URL shortener site

10B (shortened URL, long URL) pairs
o shortened URL => long URL

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 latency requirement?
o 200ms max
o serving from disk is enough
o set up index in memory to quickly locate the data on disk, one seek
o use bloomfilter to check if a shortened URL exists on disk before really hit disk

o 20ms @95, 5M qps
o 500 machines serving all from ram, usually 1ms latency,
o 5M/500=10Kqps x 200 bytes = 2MiB, network is not the bottleneck
o ram + disk, where is the bottleneck
o one hard drive, 100 seeks per second, 400 seeks total

o incoming request rate?
o 10 QPS (query per second)
o 200 bytes x 10 = 2KiB
o 10 seeks
o 5M QPS
o 200 bytes x 5M = 1GiB, minimum 8 machines, network bounded
o 5M / 400 = 12K machines, worst case

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

Class 29 Final Exam

请大家打开个人期中考试的doc

Q1. Given that we already have a helper function pow(a, b) that is a^b, how to calculate x^x^x^x^x^x … (total number of x == n) Assuming a, b, x and n are all positive integers.

2^2^2^2
2^2^ (2^2)
4^4
pow(half_result, half_result);

Q2. 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;

Q3 Shifting “ABCDEF” to the right by K letters, example, if k == 2, then output == “EFABCD”. Assuming k < size of the letter

Q4. k-way merge k sorted array with integers. (using heap)

Q5 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