第15节课时间: 2014年10月8日，星期三，太平洋时间晚上7:00

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

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

课程列表：

Sign in here please (reload the page if you do not have the edit permission):

==========
孙老师
Chunyung
Zaiqing
Nan Wang
Yuanliang Yu
He Jiang
Marcus Gao
Gu Jie
Yang YUAN
Hengshuo Zhang
Sun Yi
Ting Ma
Xiaosong Wang
Yanni Wang
Xin Jin
Xilan Ding
Liguo Yang
Hanchao
Xudong Bai
Shi Zong
Muzhi Wang

Class 1 Array and Sorting Algorithms

What do you expect from this class?

1. 技术准备
    1.1. 基础知识: 算法，和其他cs知识
    1.2. 动手能力coding训练 (不光是算法，重要是对语言的熟悉和运用)
        1.2.1. 30 classes + 1 codeLab/week
2. 简历 (扬长避短)
    2.1. 自我的能力展现
    2.2. correctness, consistency, insights
    2.3. 精炼淘汰(什么该放，什么不该放)
    2.4. 如何通过hr，拿到面试的渠道
3. 什么时候知道自己ready
    3.1. mock interview (技术 + 沟通)
    3.2. 冷冻期
    3.3. 未雨绸缪

4. 面试技巧和注意事项
    4.1. coding style
    4.2. 交流方式
    4.3. 碰上困难如何处理
    4.4. 如何准备自己的问题
5. 面试英文书信交流 / 如何negotiate offer
    5.1. We can back you up
    5.2. 来Offer :)

Why using Google doc but not ppt?

1. Better interaction (discuss a problem in real-time in a small class)
2. Mimic real interview scenario (www.collabedit.com)

Q&A after this class.

Data structure is a particular way of organizing data in a computer so that it can be used efficiently.

Common data structure

- Array (数组)
- Stack (堆栈)
- Queue (队列)
- Linked List (链表)
- Tree (树)
- Heap (堆)
- Graph (图)
- Hash table (散列表)

E.g. int a[10];

A**** 做题要求：

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.

```
for (int i = 0; i < n; i++) {
     System.out.println(array[i]);
}
```

time complexity: O(n)
Assume a problem's size is n (n elements) ，For example: print all the elments of this array.

Big O notation: algorithm complexity (time complexity, space complexity)
e.g., **time complexity** O(n).
e.g., **space complexity:** how much memory does it need to run this algorithm. O(n)
e.g., auxiliary space complexity: is the extra space or temporary space used by an algorithm.

example: int [i] a= {-1, -3, 4, 7} => {-3, -1, 4, 7} ascending order

1. Selection sort:

```
iteration 1: find global min -3   {-3, -1, 4, 7} insert -3 to the right place

iteration 2; find global min in the rest -3 {-1, 4, 7} => -3 -1 {4, 7}

iteration 3; find global min in the rest -3 -1 { 4, 7} => -3 -1 4 {7}

iteration 4; find global min and done.
```

```
// selection sort an array a[] with size n.
00 void SelectionSort(int a[], int n){
01    int global, temp;
02    for (int i = 0; i < n-1; i++) {//outer loop: how many iterations
03          global = i;
04          for (int j = i+1; j < n; j++) {//inner loop: find the
global min from the rest elements.
05                if (a[j] < a[global]) {
06                      //record the index of the smallest element.
07                      global = j;
08                }
09          }
10          // swap the global (a[index]) min with a[i];
11          temp = a[i];
12          a[i] = a[global];
13          a[global]=temp;
14    }
15 }
O(n^2)
```

时间复杂度分析：

```
for (int i = 0; i < n-1; i++) // outer
  for (int j = i+1; j < n-1; j++)  // inner
iteration i = 0: inner       (0..n-1)   = 4
iteration i = 1: inner n-1   (1..n-1)   = 3
```

```
iteration i = 2: inner n-2   (2..n-1)   = 2
iteration i = 3: inner n-2   (3..n-1)   = 1
1+2+3+4+..+n-1 = n(n-1)/2 -> n^2 → O(n^2)
```

2. Merge sort

a[10] = 1,3,5,7,9,8,6,4,2,0

O(nlog(n)) log_2(2^64-1) ~ 64 ===> O(64n)

1,3,5,7,9 ,8,6,4,2,0 a[10] -> a[0]...a[4] MERGE a[5]...a[9]

```
                 /          \
            13579           86420          O(n)
            /   \           /   \
         135    79         864  20         O(n)
         / \               / \
       13  5   79        86  4   2 0
       / \
      1  3  5   79       86  4   20   this level time complexity == O(n)
```

==========

```
       13  5   79       86  4   20   this level time complexity == O(n)
         \ /             \ /
        135  79           468  02     this level time complexity == O(n)
          \  /             \  /
         13579             02468     this level time complexity == O(n)
             \            /
              0123456789
```

```
// main function that calls merge_sort
// left:  the left index of the sub vector
// right: the right index of the sub vector
                                                    0          9
00 vector<int> mergesort (vector<int>& a, int left, int right) {
01    vector<int> solution;                  // store the final solution
02    if (left > right) return solution; // sanity check
03    if (left == right) {                   // base case
04          solution.push_back(array[left]);
05          return solution;
06    }
07    int mid = left + (right - left) / 2;
08    vector<int> solu_left = mergeSort(a, left, mid); //left:0 mid:4
????????????????????breaking point
09    vector<int> solu_right = mergeSort(a, mid + 1, right);//5    9
10    solution = combine(solu_left, solu_right);
11    return solution;
12 }
```

1,3,5,7,9

```
                  i
                 0 2 46 8
                  j
// This function is to combine two sorted integer vector v1 and v2
into one big vector and return it.
vector<int> combine(vector<int> v1, vector<int> v2) {
00    int ia = 0;
01    int ib = 0;
02    vector<int> solution;
03    while (ia < v1.size() && ib < v2.size()) {
04          if (v1[ia] < v2[ib]) {
05                solution.push_back(v1[ia]);
06                ia++;
07          } else {
08                solution.push_back(v2[ib]);
09                ib++;
10          }
11    }
12    while (ia < v1.size()) {
13          solution.push_back(v1[ia]);
14          ia++;
15    }
16    while (ib < v2.size()) {
17          solution.push_back(v2[ib]);
18          ib++;
19    }
20    return solution;
21  }
```

Discussion:

1) Could we use Merge Sort to sort a linked list? What is the time complexity if so?

2) 什么是面试中一个类型的题？

e.g., A1B2 | C3D4 -> ABCD1234

A1 B2

A 1 B 2

= = = = = = = = = =

A1 B2

AB12

AB12 CD34

AB12

i
CD34
j
ABCD1234

3. Quick Sort
O(n^2)
Average O(nlog(n))

1st Question: what is the final position of 5? 5 is randomly selected (5 is called pivot).

principle: iterate over the whole array, and put all numbers smaller than 5 to the left, then put 5 following (all numbers larger than 5 are already on 5's r-hand).
implementation details: first put 5 to the right most position (swap(5, 3)).

1 9 8 3 **5** current number: **1** . 1 < 5, so nothing changes, we will look at the next number 9
i j
1 **9** 8 3 **5** current number: **9**. 9 > 5, so put 9 to the number to the left of 5, ⇒ swap(9, 3)
i j
1 **3** 8 9 **5** current number: **3**. 3 < 5, so nothing changes, we look at the next number 8
i j
1 3 **8** 9 **5** current number: **8**. 8 > 5, so put 8 to the number to the left of 9
ij (9 was the left boundary of all numbers that are larger than 5)

1 3 **5** 9 **8** **finally, 5 is put to the right and FINAL position (by calling swap(5, 8))**
Recursive rule: Quicksort all numbers to the left of 5,
Quicksort all numbers to the right of 5,

两个挡板 i j ,三个区域 a) b) c) 的思想：
a) `[0...i)` ： i 的左侧（不包含i） 全部为比pivot小的数
b) `[i...j]`： i 和 j 之间为未知探索区域
c) `(j...n-1]` : j 的右侧(不包含j) 全部为比pivot大或等于的数字

`E.g. Pivot == 6`

index	0	1	2	3	4	5	6
A[7]	**1**	**2**	**9**	**7**	**5**	**8**	**6**
			i →		**← j**		

What is the worst case????????
1 2 3 4 5 6

p
n-1 + n-2 + n- 3 …… + 1 == O(n^2)

Discussion:

1) What is the worst case scenario for quick sort? Can you give an example?
2) Could we use quick sort to sort a linkedList?
3) 什么是面试中一个类型的题？
e.g., a) Rainbow sort (abcccabbcbbacaa → **aaaaa bbbbb** ccccc)
三个挡板，四个区域

E.g., aaaaa bbbbbXxxxxxCcccc

[i j → k [j, k] 为未知探索区域
initialization
i = 0; all letters to the left-hand side of i are all “a”s
j = 0; (j is actually the current index) all letters in [i j) are all “b”s ,
k = n-1 (all letters to the right-hand side of k are all “c”s).

unknown area is [j...k]

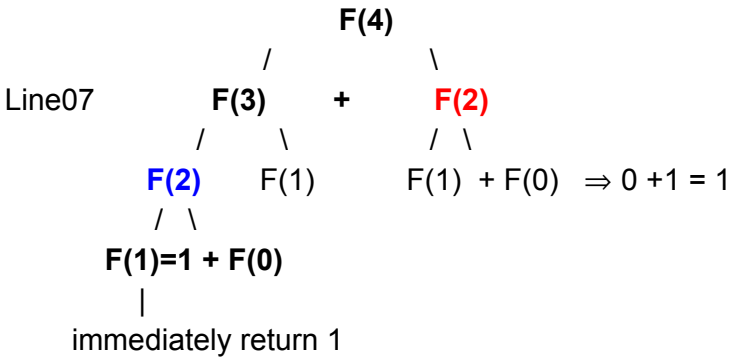
Homeworks
1. Selection Sort
2. MergeSort
3. QuickSort

Class 2 Recursion and Binary Search

1. Recursion.
需要掌握的知识点：
Boil down a big problem to smaller ones
1. recursive rule. how to make the problem smaller
2. base case
Example problem: Fibonacci sequence:
Base case: F(0) = 0; F(1) = 1;
Recursive rule: F(n) = F(n-1) + F(n-2);

```
// Calculating Fibonacci value
00 int fibo (int n) {
01   // Base case.
02   if (n == 0) {
```

```
03    return 0;
04  } else if (n == 1) {
05    return 1;
06  }
07  return fibo(n-1) break point   + fibo(n-2);  // Recursive rule
08}
```



Space O(n)

Time Complexity O(2^n)

What is the most popular interview questions for recursion?

Example question: how to calculate a^b (where a is an integer and b is also an integer, we do not care about the corner case where a = 0 or b < 0 for now)

```
int result = 1;
for (int i = 0; i < b; i++) {
     result = result * a;
}
return result;
```

1. base case

2. recursive rule????

```
2^7
2^3 * 2^3  *2
result = a^(b/2)
return result * result;
```

```
public double a_power_b(double a, int b) {
     if (b == 0) {
          return 1;
     }
     if (b == 1) {
          return a;
     }
     double result = a_power_b(a, b / 2);
```

```
        if (b % 2 == 0) {
            return result * result;
        } else  {
            return result * result * a;
        }
}
```

Time O(log(b))

2. Binary Search

what is **binary search** in the context of an array?

left right

1 3 5 7 9… 20. target number == 7

index = 0

l=r=0

7 target == 6

```
00 public int binary_search(int[] array, int left, int right, int
target) {
01    if (array == null) {
02        return -1;
03    }
04    int mid;
05    while (left <= right) {
06        mid = left + (right - left) / 2;
07        if (array[mid] == target) {
08            return mid;
09        }
10        else if (array[mid] < target) {
11            left = mid + 1;
12        }
13        else {
14            right = mid - 1;
15        }
16    }
17    return -1;
18 }
```

```
// Classical Version
// return any target element's index
00 int binary_search(int[] a, int target, int left, int right) {
```

```
01    while (left <= right) {
02          int mid = left + (right - left) / 2;    // 0
03          if(a[mid] == target) {
04              return mid;
05          } else if (a[mid] < target) {
06              left = mid + 1;
07          } else {
08              right = mid - 1;
09          }
10    }
      return -1;
}
// Variant 1.1
// return the index of the first occurrence of an element, say 5
           L=0         r=1
```

index	0	1	2	3	4	5	6
A[7]	**4**	**5**	**5**	**5**	**5**	**5**	**5**

Iteration 1: L = 0, R = 6, M = 3 A[M] == A[3] == 5 == target, so R = M = 3;
Iteration 2: L = 0, R = 3, M = 1 A[M] == A[1] == 5 == target, so R = M = 1;
因为左右边界相邻，我们terminate binary search， 再做post processing.

```
// e.g. int a[7] = {4, 5, 5, 5, 5, 5, 5};
// if target == 5;  then index 1 is returned;
// if target == 10; then -1 is returned;
// Termination condition: 当L和R 相邻的时候，跳出while 循环，再判断L和R究竟哪个是最终答案 (= post-processing)。
00 int BinarySearch(int a[], int left, int right, int target) {
01   int mid;
02   while (left < right - 1) {//if left neighbors right → terminate
03     mid = left + (right - left) / 2;
04     if (a[mid] == target) {
05         right = mid; // do not stop here, keep checking to left
06     } else if (a[mid] < target) {
07         left = mid;
08     } else {
09         right = mid;
10     }
11   }
12   if (a[left] == target) // check a[left] against target first
```

```
13      return left;
14    if (a[right] == target) // then check a[right] against target
15      return right;
16    return -1;
17  }

// Variant 1.2
// return the last occurrence of an element
// e.g. int a[6] = {4, 5, 5, 5, 5, 5};
// if target == 5;  then index 5 is returned;
// if target == 10; then -1 is returned;
00  int BinarySearch(int a[], int left, int right, int target) {
01    int mid;
02    while (left < right - 1) {//if left neighbors right → terminate
03      mid = left + (right - left) / 2;
04      if (a[mid] == target) {
05          left = mid; // do not stop here, keep checking to right
06      } else if (a[mid] < target) {
07          left = mid;
08      } else {
09          right = mid;
10      }
11    }
12    if (a[right] == target)
13      return right;
14    if (a[left] == target)
15      return left;
16    return -1;
17  }
```

Side-by-side comparison (difference is shown in red)

Variant 1.1 Find 1st element	Variant 1.2 Find last element
00 int BinarySearch(int a[], int left, int right, int target) { 01 int mid; **02 while (left < right - 1) {**//if left **neighbors** right → terminate 03 mid = left + (right - left) / 2; 04 if (a[mid] == target) { 05 **right = mid;** // do not stop here, keep checking to left 06 } else if (a[mid] < target) { 07 left = mid; 08 } else { 09 right = mid;	00 int BinarySearch(int a[], int left, int right, int target) { 01 int mid; **02 while (left < right - 1) {**//if left **neighbors** right → terminate 03 mid = left + (right - left) / 2; 04 if (a[mid] == target) { 05 **left = mid;** // do not stop here, keep checking to right 06 } else if (a[mid] < target) { 07 left = mid; 08 } else { 09 right = mid;

```
10      }
11    }
12    if (a[left] == target)
13        return left;
14    if (a[right] == target)
15      return right;
16    return -1;
17  }
```

```
10      }
11    }
12    if (a[right] == target)
13      return right;
14    if (a[left] == target)
15        return left;
16    return -1;
17  }
```

// Binary Search Variant 2.0:

Important Question: Given a sorted dictionary with unknown size, how to determine whether a word is in this dictionary or not.

Example: dictionary[x] = { 1 3 5 7 9 …...1000.. 10000000000…… }
target == 9999
assumption if a[index] == NULL then we know the size of dictionary is < index;

```
i = 0
if target =  x < 600
                        index??
1 3, 6, 8 ……500 ., 600 ,    999,   NULL…. NULL  NULL  NULL NULL
                  jump7                          jump8
```

Step 1: is to find the right border of the dictionary.
detail: use binary search **variant 1.2** to find the right-most element that is an integer [left = jump7, right = jump8]
Step 2: Given that we have found the right border: Classical Binary Search.
Optimization:
opt1: Step1, when jumping in Step 1, if A[jump_i] >= target number, then we can use jump_i as the right border of the dictionary. and use jump_{i-1} as the left index.
opt2: Step2, when we perform a classical binary search, we can use jump7 as the left border (instead of 0).

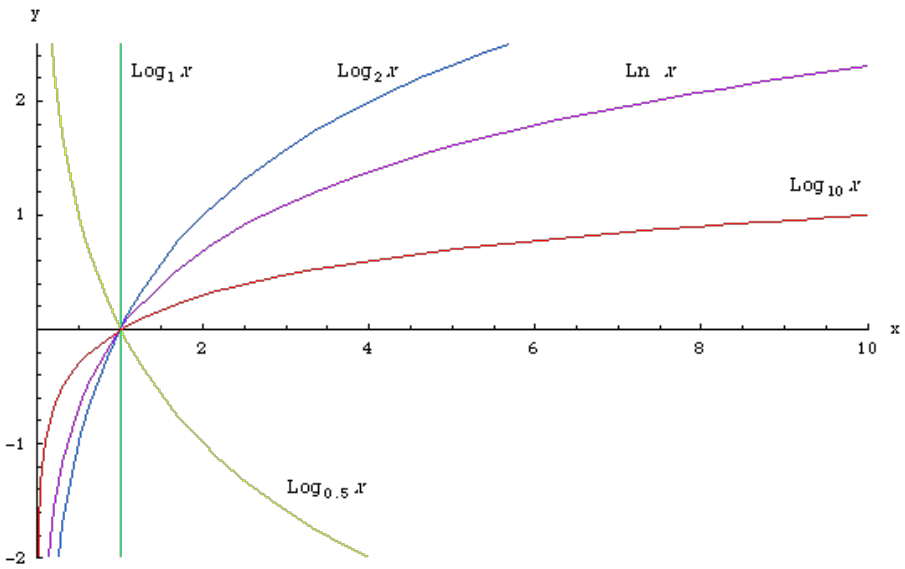
Further discussion about binary search:
Why not jump_step = jump_step * 10, instead of jump_step = jump_step *2

```
          LENGTH = n
_____________________________
 1 3, 6, 8 ……500 ., 600 ,      999,   NULL…. NULL  NULL  NULL NULL
                   jump7   index                        jump8
```

```
10times: jump8  == 10 * length
```

```
2 times: jump8' == 2  * length

length == n
```
jump_step = jump_step *2

```
10Times:  log_10 (n)          +    log_2(10n)
      time for jumping out        time for jumping in (classical)
2 Times:  log_2(n)            +    log_2(2n)
      time for jumping out        time for jumping in (classical)

10Times - 2Times:  [log_10(n) - log_2(n)]  + log_2(5)    < 0
                       < -2.5                      ~2.5
```



Homework1, Classical binary search + Variant 1.1 + Variant 1.2
Homework2, Given a sorted dictionary with unknown size, how to determine whether a word is in this dictionary or not.
Homework3, Given a sorted but shifted array (no duplicated elements), e.g., 123456789->shift to **891234567**, then how could we determine whether a target is in the array or not.
Homework4, Given a sorted array of integers, return the total number of duplicated target number. E.g., 1234**555555**67 **target = 5**, return 7.

Preview1: data structure Stack
Preview2: Linked list & pointer

Class 3 Stack & Linked List

1. Queue
   example: wait in a line, FIFO == **First in first out**
2. **Stack**
   LIFO Last in first out: like a box
   e.g., insertion order 1.2.3.4, then in the stack, it looks like
   4 <- top of the stack. **All operations can only be done to this element**
   3
   2
   1 <- bottom of the stack
   All operations available: **push**(), **pop**(), **top**()
   Implementation: popular data structure: array or vector

Two popular interview questions:

Question 1: How could we implement a queue by using two stacks?
Stack1 || :
Stack2 ||: 5 4 3 2 1

solution:
(1) we regard stack 1 as the stack to insert/push new element in
(2) we regard stack 2 as the stack to pop out element.
When popping out an element, check stack 2,
if not empty, then pop one element out
if empty, then **move** all data from stack 1 to stack 2 (special case, if both stack 1 and 2 are empty, then return NULL);
when push in, just push into stack 1;
Time complexity: push O(1)
pop() : **amortized** O(1)

Question 2: How to implement the **min()** function when using stack with time

1.1. stack 1 -- value ||-> 1 5
stack 2 -- || 1 1 1 3 3

什么问题要往Stack 上考虑？
Question 3: 从左到右linear scan 一个array/string时, 如果要不断回头看左边最新的元素时，往往要用到stack
1. Histogram 中找最大的长方形
2. reverse polish notation 逆波兰表达式的计算 a * (b+c) → abc+*

3. String的repeatedly deduplication. cabba → caa → c

2*(4+6) = 20

2 4 6+ * ⇒ 4+ 6 ⇒

stack || 2 10 2*10 ⇒ 20

Linked List:

Pointer basics (C/C++)

```
class  Node {
      int value;
      Node* next;
}
```

LinkedList uses pointer to point to its next node/struct

head

Node1->Node2-->Node3-->Node4……-->NodeN-->**NULL**

current

```
int CountNodeNumber(Node* head) {
      if (head == NULL) {
            return 0;
      }
      Node* current = head;
      int counter = 0;
      while(current != NULL) {
            counter++;
            current = current->next;
      }
      return counter;
}
```

常见考题： No.1 interview question on linkedlist: how to reverse a linked list

Node1→ Node2-->Node3-->Node4……-->NodeN-->**NULL**

head

reversed: NULL <--Node1 ← Node2 ← Node3 NULL

prev **curr**/ next

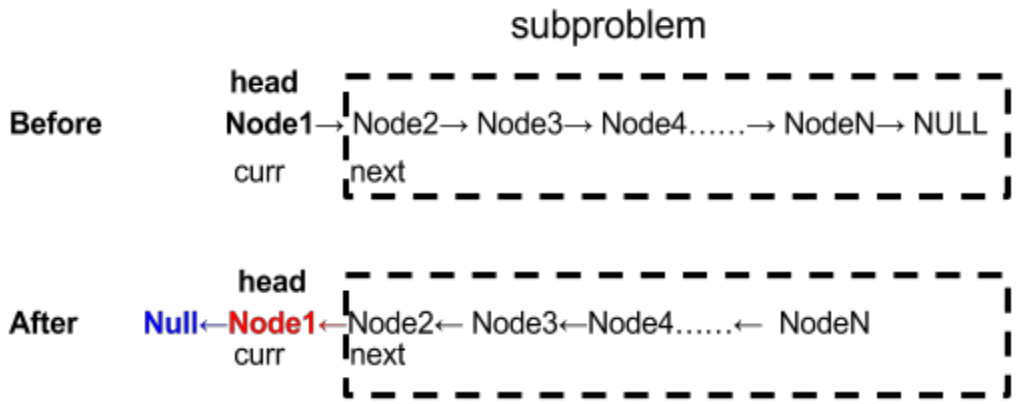
==========

Method1: Iterative Solution:

```
00 public ListNode reverse(ListNode head) {
01    if (head == null || head.next == null) {
02          return head;
```

```
03    }
04    ListNode prev = null;
05    ListNode cur = head;
06    while (cur != null) {
07          ListNode next = cur.next;
08          cur.next= prev;  // reverse action
09          prev = cur;      // move forward (for prev)
10          cur = next;      // move forward (for curr)
11    }
12    return prev;
13 }
```

Method 2: Using **recursion:**



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;

Recursion way: how to solve a bigger problem by boiling it down to smaller problems.

1. **Base case**: head == NULL || head->next == NULL (only 1 or 0 node in the linked list)
2. **Recursive rule**:
   1. subproblem = current node之外所有部分先翻转完毕
   2. next->next = current;
   3. current->next = NULL

==========

1st call R-func: **NULL <-- Node1 <--** Node2<-- Node3 NULL

head next newhead

2nd call R-func null<--Node2<-- Node3 NULL
head next

3rd call R-func Node3 → NULL
head = newhead

reversed: NULL <--Node1 ← Node2 ← Node3 NULL
prev

```
// Recursion
01 Node* ReverseList(Node* head) {
02    if (head == NULL || head->next == NULL) { // base case
03          return head;
04    }
05     Node* next = head->next;    // store the head of subproblem
06     Node* newhead = ReverseList(next);  // recursion call
07      next->next = head;
08     head->next = NULL;
09     return newhead;
10 }
```

optimized version

```
01 Node* ReverseList(Node* head) {
02    if (head == NULL || head->next == NULL) { // base case
03          return head;
04    }
05    Node* newHead = ReverseList(head->next); // recursive rule
06    head->next->next = head;
07    head->next = NULL;
08    return newHead;
09 }
```

常见考题：

1. How to find the middle node of a linked list?

N1 → N2 → N3 → N4 → N5 → N6 → NULL
← ---------- f
s

2. 用快慢指针判定一个linkedlist是否有环。

method to find the start node of the loop:

Step 1: quick and slow pointers need to meet together.

Step 2: move the slow pointer back to the head, and make slow and quick pointer move

together **(1 : 1),** when they meet again at Node_x, Nodex must the solution.

3. Insert a node in a sorted linked list (simple)
N1 → N2 → N3 → N4 → N4.5 → N5
prev
4. How to merge two sorted linkedlist into one long sorted linked list
N1 -> N2 -> N5 -> NULL
curr1
N1.5-> N3 -> N6 -> NULL
curr2

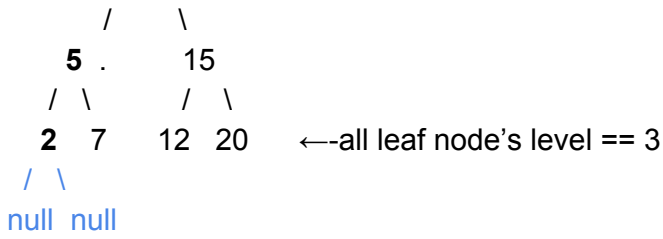
5. **N1 → N2 → N3 → N4 → N5 → N6 →…. → Nn**→ null **(convert to)**
(N1 → Nn) → (N2 → Nn-1)**→ …**
Step1: find the middle node of the linked list (q1)
N1 → N2 → N3 → N4 → || N5 → N6 →…. → Nn→ null
Step2: reverse the second half (with N5 as the head of the linked list)
Nn → Nn-1 → Nn-2 … .N5
Step3: Merge two linked list together
N1→ **Nn**→ **N2 → Nn-1 → N3**→ **Nn-2**

Homework1: reverse a linked list in an iterative way
Homework2: reverse a linked list in a recursive way
Homework3: 常见考题1
Homework4: 常见考题2
Homework5: 常见考题3
Homework6: 常见考题4
Homework7: 常见考题5

Homework8 (optional) Partition List:
Given a linked list and a target value x, partition it such that all nodes less than x are listed before the nodes larger than or equal to target value x. (keep the original relative order of the nodes in each of the two partitions).
For example,
Input: 1->6->3->2->5->2 and target x = 4,
result: **1->3->2->2->6->5.**

Class 4 Binary Tree & Binary Search Tree

Binary Tree and BST
Definition: at most two children nodes
10 == root

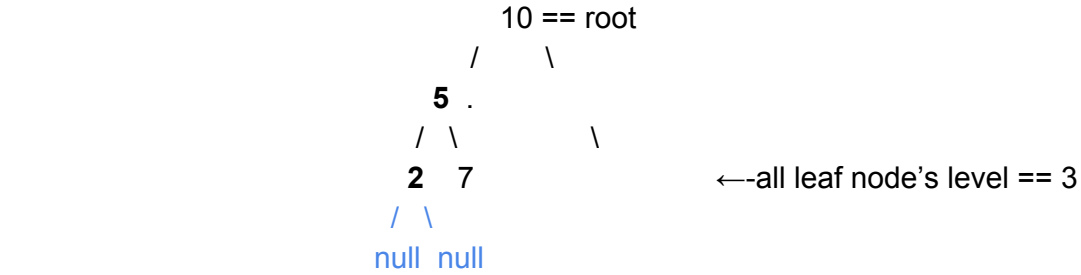


pre-order: 10 5 2 7 15 12 20
in-order : 2 5 7 10 12 15 20 (in an ascending order)
post-order 2 7 5 12 20 15 10

preorder traverse the binary tree and print each node's value on the console

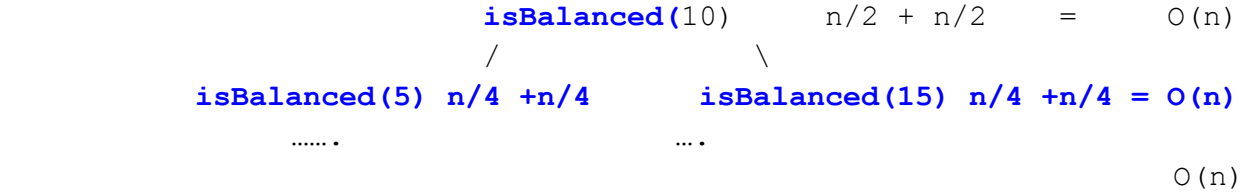
```
public void printBinaryTreePreOrder (Node root) {
      if ( root == null) {
            return;
      }
      System.out.println(root.val);
      printBinaryTreePreOrder(root.left);
      printBinaryTreePreOrder(root.right);
}
public void printBinaryTreeInOrder (Node root) {
      if ( root == null) {
            return;
      }
      printBinaryTreeInOrder(root.left);
      System.out.println(root.val);
      printBinaryTreeInOrder(root.right);
}
public void printBinaryTreePostOrder (Node root) {
      if ( root == null) {
            return;
      }
      printBinaryTreePostOrder(root.left);
      printBinaryTreePostOrder(root.right);
      System.out.println(root.val);
}
```

- **Balanced binary tree**: is commonly defined as a binary tree in which the depth of the left and right subtrees of **every node** differ by 1 or less
- **Complete binary tree**: is a binary tree in which every level, *except possibly the last*, is completely filled, and all nodes are as far left as possible




```
00 public boolean isBalanced(TreeNode root) {
01    if (root == null) {
02          return true;
03    }
04    int diff = getHeight(root.left) - getHeight(root.right);
05    if (Math.abs(diff) > 1) {
06          return false;
07    }
08    return isBalanced(root.left) && isBalanced(root.right);
09  }
```

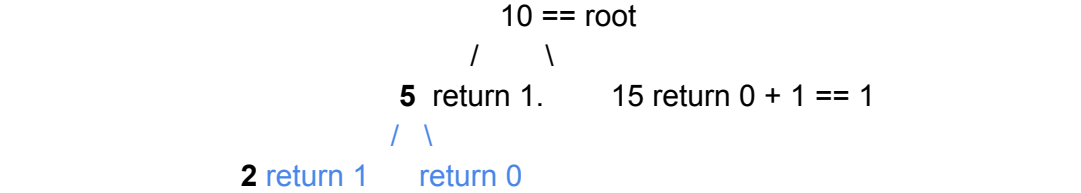
==========



==========

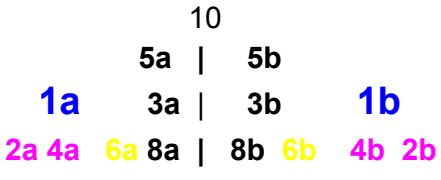
total log(n) layers, and for each layer the time complexity is O(n)
therefore, the total time complexity == **O(nlog(n))**

```
//get height of the tree
10 private int getHeight(TreeNode root = 15 ) {  // O(n) n is the
number of node in the subtree rooted at root
11    if (root == null) {
12          return 0;
13    }
14    return Math.max(getHeight(root.left)   0 // break point 1
,getHeight(root.right)   0  // break point 2) +1;
15 }
```



Complete binary tree: is a binary tree in which every level, ***except possibly the last***, is completely filled, and all nodes are as far left as possible

Q1 How to judge whether a binary tree is symmetric????



==========

level 1: for two nodes 5a and 5b, how many recursion function calls are triggered? two nodes ⇒ two recursion call || 1 node vs 1 call

level 2: for 1a and 1b, two recursion calls are called.
for 3a and 3b, two recursion calls are called || 1 node vs 1 call
Total runtime complexity = O(n)

==========

```
00 bool Symmetric(Node* root1 == 5a, Node* root2 == 5b) {
01    if(root1 == NULL && root2 == NULL) {  // base case
02          return true;
03    } else if (root1 == NULL || root2 == NULL) {
04          return false;
05    } else if (root1->value != root2->value) {
06          return false;
07    }
08    return Symmetric(root1->lChild, root2->rChild ) &&
09           Symmetric(root1->rChild, root2->lChild);//recursive rule
10 }

11 bool IsSymmetricMain(){
12    Node* root = xxx;// initialization
14    if (root == NULL) {
15          return true;
16    }
17    return Symmetric(root->lChild, root->rChild);
18 }
```

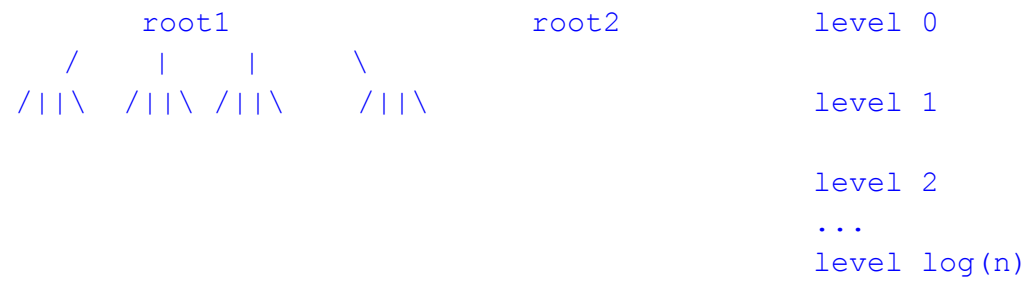
==========

Q2. Let's assume if we tweak the lchild with rchild of an arbitrary node in a binary tree, then the "structure" of the tree are not changed. Then how can we determine whether two binary

trees' structures are identical.

total number of layers in the original trees == log_2(n)

```
00 bool identicalHelper(Node* root1, Node* root2) {
01    if(root1 == NULL && root2 == NULL){  // base case
02         return true;
03    } else if (root1 == nullptr || root2 == nullptr) {
04         return false;
05    } else if (root1->value != root2->value) {
06         return false;
07    }
08    return (identicalHelper(root1->left, root2->left) &
identicalHelper(root1->right, root2 -> right))  // case 1
 || (identicalHelper(root1->left, root2->right) &&
identicalHelper(root1->right, root2 -> left));  // case 2
```



如果是个二叉树，有多少个node在二叉树里面？given 层数是log(n)？

2^log(n)

====> n node in the tree

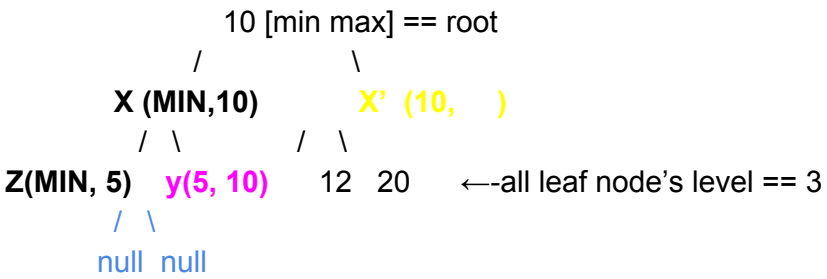
如果是个四叉树：

4^log(n) = 2^(2log(n)) = 2^(log(n^2)) = n^2

Total time complexity = O(n^2).

2. Binary Search Tree

经典例题1：How to determine whether a binary tree is a BST or not?

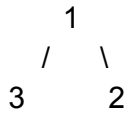



```
00 bool isBSTMain(TreeNode* root) {
01    int min = INT_MIN;
02    int max = INT_MAX;
03    return isBSTHelper(root, min, max);
04 }

05 bool isBSTHelper(TreeNode* root, int min = -infinity, int max = +
infinity) {
06    if( root == null ) return true;
07    if( root.val <= min || root.val >= max ) return false;
08    return (
      isBSTHelper(root.left, min , root.val) &&
      isBSTHelper(root.right , root.val , max) );
09 }
```

Homework1: preorder, inorder and postorder traverse a binary tree
Homework2: how to judge whether a binary tree is a binary search tree?
Homework3: how to judge whether two binary tree are identical?
Homework4: how to judge whether a binary tree is symmetric (Q1 in the class)
Homework5: Let's assume if we tweak the lchild with rchild of an arbitrary node in a binary tree, then the structure of the tree are not changed. Then how can we determine whether two binary trees' structures are identical. (Q2 in the class)
Homework6: Print BST in a given range.

Class 5 Heap & Graph + Search Algorithms

堆（英语：**heap**）亦被称为：**优先队列**（英语：**priority queue**）
Example



index 0	1	2	3	4	5
1	3	2	5	4	7

Heap: is an unsorted array but have special rules to follow

性质: 堆的实现通过构造二叉堆（binary heap）,这种数据结构具有以下性质

1. 任意节点小于它的所有后裔，最小元素在堆的根上（**堆序性**）。
2. 堆总是一棵完全树。complete tree
3. 将根节点最大的堆叫做MAX HEAP，根节点最小的堆叫做最小堆MIN HEAP
4. index of lChild = index of parent X 2 + 1
5. index of rChild = index of parent X 2 + 2
6. unsorted but follow rules above

支持的基本操作

1. insert：向堆中插入一个新元素； **时间复杂度O(log(n))**
2. update：将新元素提升使其符合堆的性质； **时间复杂度O(log(n))**
3. get：获取当前堆顶元素的值； **时间复杂度O(1)**
4. pop：删除堆顶元素； **时间复杂度O(log(n))**
5. **heapify**：从unsorted array 直接建成一个heap。 **时间复杂度O(n)**

Q1 Find **smallest** k elements from **an unsorted array** of size n.
Assume k == 7
M1:
Step1 heapify the first 7 elements from the array, and make it a **MAX** heap O(k)
Step2 iterate over the rest k-7 elements, for each element x, compare x with heap.top().
if (x < heap.top()), then pop heap and insert x into the heap.
O((n-k)log(k))
Total == O(k + (n-k)log(k)) = **O(nlog(k))**

M2:
selection sort
k iterations to find k times smallest element from rest of elements.
O(n k)

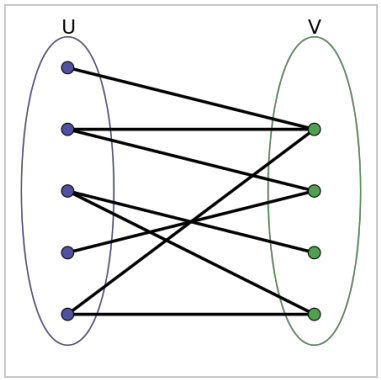
M3:
sort the whole array, and return A[0..k-1] **O(nlog(n))**

M4:

heapify the whole array to form a MIN_HEAP, and pop k-times
O(n + klog(n)) ===> O(n)

Graph

1. node
2. edge
3. directed vs undirected graph
4. **Bipartite** ： whether a graph's node can be divided into two group, such that the nodes in each group do not have direct edges between the nodes that belong to the same group.



1 (u) --- **2 (v)**
\ /
3 (v != u)

Expand 1(u) → generate 2 (v)
generate 3 **(v)**
queue == [2, 3]

Expand 2(v) → (re)generate 3**(u)**

```
class Node {
        int group;        // 0 : u ,   1: v
        vector<Node*> neighbors;
        bool has_expanded;
}
```

图里常用的search 算法

1. **Breath-First Search (BFS)** :

1 expand1 can generate 3 and 2 ---->
/ \
3 **2** **--->**

```
        /   \       /
       5     4     7    --->
```

FIFO queue, 1 queue == [1]
step 1: pop 1 out of the queue and **expand** 1, and **generate** 3 and 2, and insert them into the queue. queue= [3, 2]
step 2: pop 3 out of the queue and **expand** 3, and generate 5 and 4 and insert them into the queue, queue = [2, 5, 4]
step 3, pop 2, out of queue and **expand** 2, and generate 7 ……

```
struct Node {
        int value;
        Node* lChild;   // lChild == NULL by default
        Node* rChild;  // rChild == NULL by default
}
```

Termination condition: when the queue is empty.

BFS的操作过程 & How to describe a BFS's action during an interview?

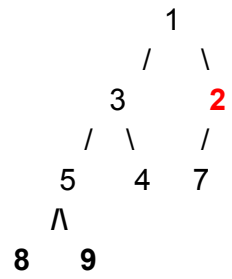
- **Definition 1**: **expand** a node s: **中文：延展**一个node **，e.g.** visit/print its value….
- **Definition 2: generate** s's neighbor node**:** reach out to its neighboring node (First, to generate Node 3, and then generate Node 2).
- **Data Structure**: Maintain a **FIFO queue**, put all generated nodes in the queue. e.g., 3 and then 2 into the queue (FIFO) queue head-> [3, 2] tail
- **Termination condition**: do a loop until the queue is empty
- **Process**:
    - step 1: queue size == 1 (only root in queue) → after expanding 1, then queue size == 2
    - step 2: ….

Classical Interview Question (BFS) :
经典例题1 : Determine whether a binary tree is a complete binary tree or not

```
            1
          /   \
        3       2 - - >1st element that misses one child
      /   \       \
     5     4       7
```

Solution: after detecting the first node that misses one child, then check whether all following nodes expanded to see whether they have any node generated (if any → then false)

经典例题2: 分层打印一个binary tree.
The key problem to print the binary tree layer by layer is to find the end of each layer.



1 == queue.size() for(i = 0; i < queue.size() == 1; i++) \\

32 size == 2 == queue.size() right after line change. for(i = 0; i < queue.size() == 2; i++) \\

547 size == 3 == queue.size() right after line change.

547

89

```
// param: root - the root of the tree
00 public void BFS(Node root) {
01    if (root == null)
02          return;
03    Queue<Node> q = new LinkedList<Node>();
04    q.offer(root);
05    while(!q.empty())  // termination condition of BFS
06    {
07          int size = q.size(); // size = # of generated nodes in the
next layer.
08          for(int i = 0; i < size; i++) {
09                Node n = q.remove();
10                if(n.left != null)
11                      q.offer(n.left);
12                if(n.right != null)
13                      q.offer(n.right);
14                System.out.print(n.val+" ");
15          }
16          System.out.println();
17    }
18 }
```

Discussion

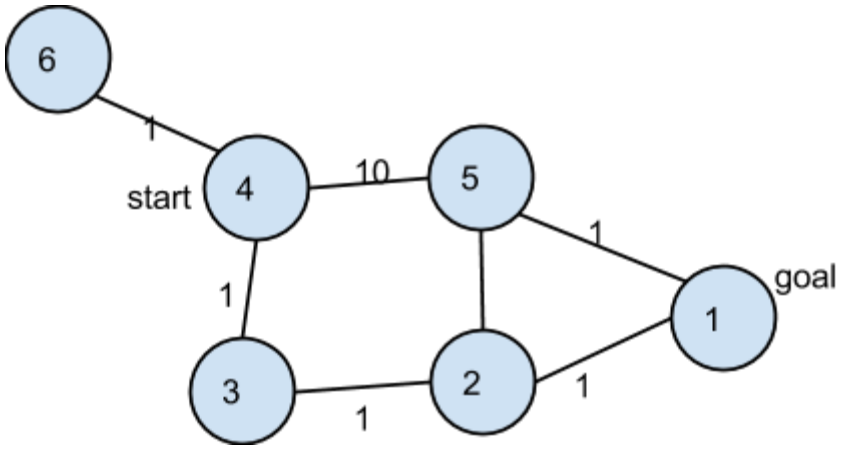
常见考题：

- 各种分层打印树的变体(zig-zag)
- 常见错误： BFS 不是找图上两点最短路径的算法！！！

2. Best First Search (BFS-2)

经典算法：Dijkstra's Algorithm

1. **Usages**: Find the shortest path cost from a single node (source node) to any other nodes in that graph (**点到面(==所有点)**的最短距离算法)
2. **Example problem**: 从北京到中国其他所有主要城市的最短距离是多少
3. **Data structure**: priority_queue (MIN_HEAP)
4. **解题思路**
   - 4.1. **Initial state (start node)**
   - 4.2. **Node expansion/Generation rule:**
   - 4.3. **Termination condition: 所有点都计算完毕才停止,也就是 p_queue 变空**
5. **Example**
   - 5.1. start node is **4**
   - 5.2. cost(node) = cost(parent of node) + c(parent of node, node)
     e.g. cost(2) = cost(3) + c(3, 2) = 1 + 1 = 2
   - **5.3. Cost[4, 5] = 10;**
   - 5.4. Cost for all the rest edges = 1;
   - 5.5.



   - 5.6. **Step 0 (initial state):** No nodes have been expanded, and only source node (== start node 4) is marked generated (which is inserted into the **priority_queque**)

     Step1: from start node (e.g., 4), we expand 4 and generate all its successors (5, 3, 6). with cost: cost(5) =10; cost(3) =1, cost(6) = 1, and inserts node 5,3.6 into the priority_queue.
     Step 2: (**expand node 6**, need to **pop node 6 out** of the p_queue) we pick one node from the priority_queue with the s**mallest cost, which is node 6**.
     -- generate NONE, priority queue = [3(1), 5(10)]

     Step3, expand node 3, generate node 2 with cost = cost (3) + c(3, 2) = 1 + 1 = 2, priority queue = [5(10), **2(2)**]

Step 4, expand node 2

-- generate node 1 with cost(1) = cost(2) + c(2, 1) = 2 + 1 == 3, insert node 1 into the priority_queue

-- generate node 5 for the **second time**, cost(5) = cost (2) + c(2, 5) = 2 + 1 = 3 < 10, so **update its order in the priority queue ， the priority queue == [5(3), 1(3)]**

termination condition: priority queue is empty

6. **properties (s)**
   - 6.1. one node can be expanded once and only once
   - 6.2. one node can be generated more than once. (cost can be reduced over time)
   - 6.3. all the cost of the nodes that are expanded are monotonically non-decreasing (**所有从priority queue里面pop出来的元素的值是单调非递减 - - > 单调递增**
   - 6.4. time complexity, for a graph with n node and the connectivity of the node is http://en.wikipedia.org/wiki/Dijkstra's_algorithm **O(nlog(n))**
   - 6.5. Whenever a node is popped out of the priority_queue for expansion, its cost is equal to the smallest path cost from the start node to it.

经典考题：(运用 Dijkstra's Algorithm的性质)

Given a matrix of size NxN, and for each row the elements are sorted in an ascending order. and for each column the elements are also sorted in an ascending order.

How to find the k-th smallest element in it?

```
12a 345
2b3456
34567
45678
56789
```

1. **Initial state (start node) == top left node**
2. **Node expansion/Generation rule:**

   if a node is being expanded, then we know the element

   -- to the right-hand side

   -- to the bottom

   are larger than itself, so we only generate the node's right-neighbor and right-neighbor

3. **Termination condition:** when the k-the element is popped out of the heap, then we just return this element as the final solution.

Trick to avoid duplicated element in the priority queue:

Using a hash_table (set) to store all the elements' coordinates that have been inserted into the priority_queue.

For example, when 3 is generated for the first time , we store 3's cooridnates [1][1] in the hash_table <**key = <x, y>,** value = false>. 3's coordinate is [1][1]

Homework1: Determine whether an undirected graph is Bipartite
Homework2: Find smallest k elements from an unsorted array
Homework3: Given a matrix of size NxN, and for each row the elements are sorted in an ascending order. and for each column the elements are also sorted in an ascending order. How to find the k-th smallest element in it?
Homework4: print the values of the nodes in a binary tree layer by layer.

Class 6 DFS & Hashtable

上一节课讲了两个BFS

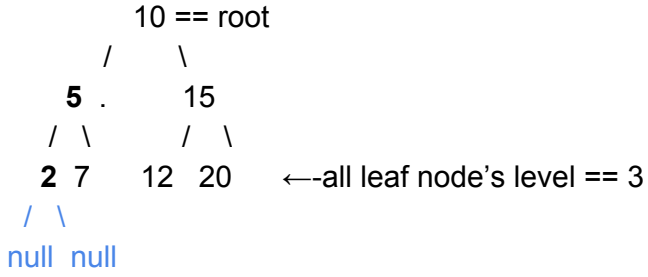
- Breadth First Search
- Best First Search

今天讲

Depth-First Search (DFS):

- Recall "using pre-order to traverse a tree".
- 实现方法： easy to use **recursion**
- 常见考题：
  - **DFS 例题1：**print all subsets of a set
  - **DFS 例题2：**print all valid permutations of （ ）（ ）（ ）
  - 如何生成一个随机的maze **（强化练习会讲）**
  - Various permutation/combination的题目： e.g., 凑硬币金额
    - **有1 分，5分，10分，25分coin，给定一个钱数99，有多少种组成方式，并打印出所有的可能组合？（强化练习会讲）**
    -

First, let's recall the pre-order traverse the binary tree code




```
public void printTreePreOrder(Node root) {
     if (root == null) {  // base case
```

```
            return;
        }
        System.out.println(root.val);
        printTreePreOrder(root.left); // case1: try left first
//////breaking point
        printTreePreOrder(root.right);// case2: try right then
}
```

pre-order: 10 5 2 7 15 12 20

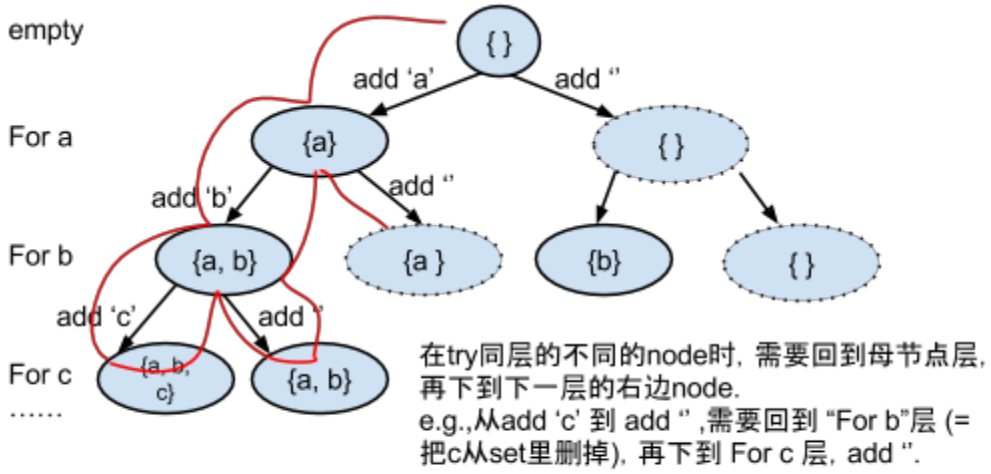
DFS 例题1 Print all subset of a set S = {'a', 'b', 'c'}

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？ **)**

For this question:

1. how many elements/letters total in the full set (therefore, there are three levels in the DFS tree)
2. whether this letter should be inserted into the sub-set or not (therefore there are two states to be tried for each node)



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

```
public void FindSubSet (String input, int index, StringBuilder
solution){
        if (input.length == index) {
                System.out.println(solution);
                return;
```

```
        }
        // Case 1. Add string.at(index) to subset.
        solution.append(input.charAt(Index));
        FindSubSet(input, index + 1, solution); // breaking point
        solution.removeLast(); //recover the state before trying case 2

        // Case 2. Do not add string.at(index) to subset.
        FindSubSet(input, index + 1, solution);
}
```

DFS 例题2 ()()() find all valid permutation using the parenthesis provided.

```
invalid:  )(
          ( ) )
valid:    (((   )))
          (() ())
```

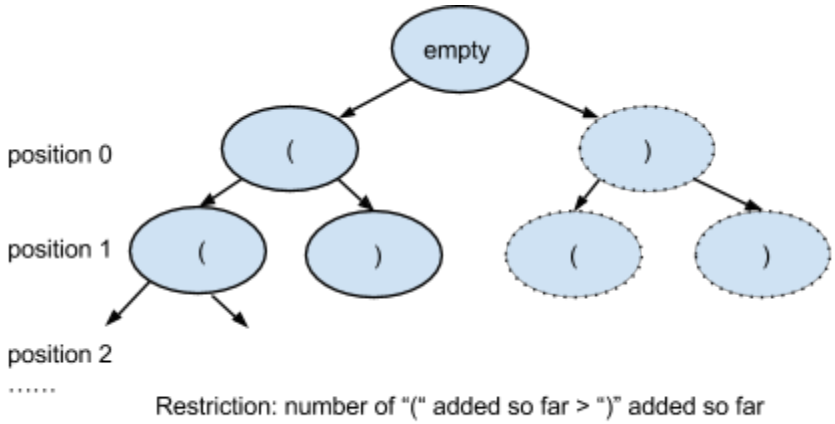
Restriction: whenever we insert a new ")" we need to make sure the number of "(" added so far is larger or equal to the number of ")" added so far.

DFS 解法：

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) **six-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？ **)** **two nodes/cases**

index 0 1 2 3 4 5

(( )
))



```
// n stores total number of "pair of ()" need to add. So total levels
// == 2*n.
// l stores the number of left parenthesis "(" added so far.
// r stores the number of right parenthesis ")" added so far.
// solu_prefix: solution so far
public void DFS (int n, int l, int r, StringBuilder solu_prefix) {
      if (l + r == 2n) {     // base case
            System.out.println(solu_prefix);
            return;
      }
      if (l < n) {  // case 1.  add "("
            solu_prefix.append("(");
            DFS (n, l + 1, r, solu_prefix);
            solu_prefix.removeLast("(");
      }
      if (r < l) {  // case 2.  add ")"
            solu_prefix.append(")");
            DFS(n, l, r + 1, solu_prefix);
            solu_prefix.removeLast(")");
      }
}
```

Hash Table

1. hash_map
2. principle **<key == name, value == counter++>** e.g., **<string, int>**

syntax: declare→ **hash_map <string, int> map1;**
key value

仔细阅读： http://en.wikipedia.org/wiki/Hash_table

1. hash collision
   1.1. chaining
   1.2. open address (probe + rehash)

常见考题：

Q1, For a composition with different kinds of words, try to find the **top k frequent** words from the composition.

Step1: parse the whole composition, and store each word into the hash_table

<string = word, int =counter>
e.g. <apple, 7>
<banana, 6>
…….

Step 2: iterate over the hash_table, and heapify the **first k words** to form a min_heap
Step 3, iterate over the hash_table from k+1-th element to the n-th element, to get the top-k element from the in_heap.
Principle: combine priority_queue with hash_table.

Q2. If there is only one missing number from 1 to n in an unsorted array. How to find it in O(n) time? **size of the array is n-1**.

M1: use hash_table,
Step1: iterate over the whole array, and store each element in the hash_table <key = int, value = bool>
Step2: for(int = 1; i <=n; i++) {
check whether i is in the hash_table or not.
}

M2: what if we have additional restriction, do not use O(n) space . let us say we use O(1) space.
Bucket sort?????

i =0

Index	0	1	2	3	4	5	6	7
value	null	1	2	3	4	5	6	7

M3
1 2 3 4 5 ……n
(1+n)/2 *n == sum (1….n) == s
sum(1..n miss one number) == x
s-x == target that is missing

Q3 Find common numbers between two **sorted** arrays a[N], b[M], N, M
a[N] = {1, 3, 5, 7, …..}
b[M] = {1, 2, 4,,,,.....}

M1: use hash_table

Step 1: build one hash_table for a[N]

Step 2: Iterate over each element in b[M], and check against the hash_table to see whether the element is in the hash_table, done!

Time Complexity = O(m + n)
Space Complexity = O(n)

M2: (what if N <<<<<< M)

A= {1, 5000} N
B = {1……. 1 billion} M

Run two binary searches on B , with target = 1, and 5000, respectively.
O(nlog(m)) == 64*n = 128
O(n+m) == 1 biliion

M3: merge
A= 1 3 5 6 7 8 9
i
B = 2 4 6 7 8 9
j

6 7 8 9

O(m+n)

Homework:
Q1: DFS 例题1：print all subsets of a set
Q2: DFS 例题2：print all valid permutations of （ ）（ ）（ ）
Q3: Various permutation/combination的题目： e.g., 凑硬币金额
有1 分，5分，10分，25分coin，给定一个钱数99，有多少种组成方式,并打印出所有的可能组合
Q4: For a composition with different kinds of works, try to find the **top k frequent** words from the composition.
Q5: If there is only one missing number from 1 to n in an unsorted array. How to find it in O(n) time? **size of the array is n-1**. (Using hash_table)
Q6: Find common numbers between two **sorted** arrays a[N], b[M], N, M
M-1 2 3.
==========

99 cent
**/ \ **

25 0 (99 remain) 1 (74 remain) 2 (49 remain) 3 ()
/ \\\\\ /|
10 0 1 2 3 4 … 9 0 1 2 3...7

5

Class 7 Bit representation of a number and bit operation

Assume we have a number 10 , how do you convert it to a binary representation?

10 /2 == 5 10 mod 2== 0
5 /2 == 2 5 mod 2 == 1
2/2 == 1 2 mod 2 == 0
½ == 0 1 mod 2 == 1

000000000000000 **1010**

== 0*2^30 + 0*2^29…. +1*2^3 + 0*2^2 + 1 * 2^1 + 0*2^0 = 10
8 + 0 + 2 + 0 = 10

注意**最左边一位**是符号位，0: positive number; 1: negative number
“2” ***0***000 0000 0000 0000 0000 0000 0000 00**1**0
== 0*2^30 + 0*2^29…. + **1*2^1** + 0*2^0= 1

对于负数 **“-1”: 1 变反加1，**
1== ***0***000 0000 0000 0000 0000 0000 0000 000**1**
变反 1111 1111 1111 1111 1111 1111 1111 1110
加一 1111 1111 1111 1111 1111 1111 1111 1111

- & AND if (true1 && true2)
  - 1 & 1 == 1
  - 1 & 0 == 0
  - 0 & 1 == 0
  - 0 & 0 == 0
  - For instance, working with a byte (the char type):

```
  1100 1110
& 1001 1000
= 1000 1000
```

- | OR

- ~ NOT (1->0 and 0>1 for each bit)
- ^ XOR
    - 00->0
    - 11->0
    - 01 or 10 ->1

- << left shift **右侧补充零**
    - x = 0001
    - x << 1 → 0010
- >> right shift 左侧补充零 (positive number): e.g int x = 1; x >>1，将x 向右移动一位，左侧补充零
  If the variable ch contains the bit pattern 01100101, then **ch >> 1** will give the output as **0**01110010**1**, and **ch >> 2** will give **00**01110011**0**.

  if we have a **negative** number , after right-shirt by 1 bit, then what do we get?
  11111111111111111111111 (after right shift, the sign bit will fill in)

(1) Given a number x, how to set x's k-th bit to 1? . e.g. x == 000**0** 0000, and k = 4;
result: x == 000**1** 0000
Step 1: a = 1; // a == 0000 0000 0000 0000 0000 0000 0000 0001
a << 4; // a == 0000 0000 0000 0000 0000 0000 000**1** 0000
x == 00000000101001010101101010100**0**x 0010

step 2: x = x | a for short **x |=a**
x = x + 1 x+=1

(2) Given a number x, how to set x's k-th bit to 0? . e.g. x == 000**0** 0000, and k = 4;

x == 110**0** 0010
0001 0000
1110 1111
Step 1: a = 1; // a == 0000 0000 0000 0000 0000 0000 0000 0001
a << 4; // a == 0000 0000 0000 0000 0000 0000 000**1** 0000
Step 2: a = ~a;
a == 1111 1111 1111 1111 1111 1111 111**0** 1111
Step3: x = x & a for short x &=a

Question 1: determine whether a number x is a power of 2 (x == 2^n)

2^6 == target == 0**1**00 0000 >>1
target' 0000 0**0**0**1**

实质上，是问这个数的二进制里，是否只有一个bit为1.

```
00 bool isPowerOfTwo(int target) {
01    if (target <= 0) {
02          return false;
03    }
04    while (target & 1 == 0) {
05          target >>= 1;
06    }
07    return (target == 1);
08 }
```

M2:

Assume x is a positive number, then this method works.

what if x can be 0?

```
return ((x & (x - 1)) == 0 && x != 0);
```

x == 0000 0000 0000 1000

x-1 == 0000 0000 0000 0111

x == 0000 0000

x-1 = -1 1111 1111

Question 2: How to determine the number of bits that are different between two positive integers?

x = 3; 0011

y = 2; 0010

x^y == 0001

Step 1: result = x ^ y;

Step 2: count how many 1s in the result;

```
public int checkTwoNumbersBits(int a, int b) {
      int result = a ^ b;
      int count = 0;
      while (result > 0) {
            if(result & 1 != 0)
                  count ++;
```

```
            result >> = 1;
        }
        return count;
}

int checkTwoNumbersBits(int a, int b) {
  int count = 0;
  for (int c = a ^ b  (initialization); c != 0 (termination
condition); c = c >> 1 (action after each iteration)) {
        count += c & 1;
  }
  return count;
}
```

Question 3: What happens if we assign a negative number to an unsigned integer?

```
00 int a = -1;      // a = 1111 1111 1111 1111
01 unsigned int b;
02 b = a;           // b = 1111 1111 1111 1111
03 cout << "b == "<< b; // b is VERY LARGE positive number
```

principle: when we assign a negative number to an unsigned integer, the binary representation does NOT change. However, we will calculate the value of the unsigned integer by using the current binary representation.

Question 4: determine whether a word contains all unique letters.
ASCII 256 letters possible

word = Aacdwer$#;

M1: **Hash_table** = <key == letter, value = bool>
M2: use an array of size bool a[256]
Advantage is that is easy to implement, **Space complexity** == 256 Byte

M3: bit vector

Assume we have a 32 bit machine, for one integer
int x; // x actually occupies 32 bit == 4Byte.

256 letters if we can use only one bit to represent each letter, then we can save memory by 8 times.

Simplified question, let assume we only have 26 letters "abc…..xyz"

int x = 0; // x actually has 32 bit

gfe dcba

0000 0000 0000 0000 0100 0000 0000 0011

string s = "banana"

i

1<<1

bit vector == essentially, it is an array of integer
256/32 = 8

int a[8] to represent 256 letters, should that be enough?

a[0] = 0000 0000 0000 0000 0000 0000 0000 0000 // first 32 bits [0--31] bit
a[1] = 0000 0000 0000 0000 0000 0000 0000 0000 // 2nd 32 bits [32 - 63] bit
a[2] =

what if the letter that we are scanning is $ (sequence number = 87)
step 1: we need to find which element in the array should we set bit in?
87/ 32 = 2 ⇒ we should set bit in a[2]
step 2: which bit should we set to "1" in a[2]
87 % 32 = 23

Question5 Given a number x, how to get the hexadecimal representation of the number in string type? E.g 29 ⇒ "0x1D" 0X15 VS 15

10
0x10 = 1* 16^1 + 0 * 16 ^0

10 进制 = >0 1 2 3 4 5 6 7 8 9 10 **11** 12 **13** 14 15
16 进制 => 0 1 2 3 4 5 6 7 8 9 A B C **D** E F

static char **a[17]** = {'0', ….'F'};

```
string convert(int number) {
     if (number==0)
       return "0x0";
     string result = "";
     while (number){
          int digit = number % 16;
```

```
            result = a[digit] + result;
            number >>4;
        }
        return ("0x"+result);
}
```

Homework:
Question1-5

Class 8 Midterm 1 (Data Structure)

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.

Question 1. How to judge whether a binary tree is a binary search tree?

Question 2. Given a string **with no duplicated letters**, how to print out all permutations of the string.
e.g., string input = "abc" output: abc acb bac bca cab cba
Step 1: know how many layers in the recursion tree (string = "abc" → 3 layers)
Step 2: know how many states should we try on each node ()

position 0: a (rest = bc) b (rest = ac) c (rest = ab)
/ \ / \
position 1: b(rest = c) c(rest = b) a(rest =c) c (rest=a)
| |
position 2: c b

postion 3??????????

???????????

```
// swap two letters in input string [i] and [j].
00 void swap(string& input, int i, int j) {
01     assert(i < input.size() && j < input.size());
02     if (i == j) {
03         return;
04     }
05     char temp = input[i];
06     input[i] = input[j];
07     input[j] = temp;
08 }

// index is the current layer that we are tyring
09 void permutation(string& input, int index) {
10     if (index == input.length()) { // print solution and return;
11         cout << input << endl;
12         return;
13     }
14     // put each letter in the index-th position of the input str.
   index = 0  1  2
   input = a  b  c

15     for(int i = index = 0; i < input.size(); i++) {
                           0    0,1,2
16         swap(input, index, i);
                 swap(00)  swap(01)  swap(02)
                    abc     /bac      /cba
                                      /cab
                   bc/cb
17         permutation(input,          index+1);  // c b a
18         swap(input, index, i);
19     }
20 }
```

1 3 3 4 5

```
     1                    3     4     5
  3  3 4    5            1345  1335  1334
 345  335  334
45
```

==========

```
21 int main() {
```

```
22    string input = "abc";
23    permutation(input, 0);
24    return 0;
25 }
```

Question 3. Given two **sorted** arrays A and B, with their sizes to be m and n, respectively. We can pick one element a from A and the other element b from B, and their sum s is defined to be s = a + b. How to find k-th smallest s from all possible values of s.

(Assumption, no duplicated elements from A and B).

A = {1, 3, 4, 5, ….}
B = {2, 3, 6, 8….}

Best-First Search
state <A[i], B[j]>

1. Initial state <A[0], B[0]>
2. Expansion/Generation Rule: <A[i], B[j]>
   2.1. --generate <A[i+1], B[j]>
   2.2. --generate <A[i], B[i+1]>
3. Termination condition: k-the state is popped out of the queue.

Deduplication: A[5][5] → A[4][5]
                A[5][4]

??????????????????
index 0 1 2 3 4
A = { 1 3 5 7 9 … }
B = { 2 4 6 8 ….}

<0 0>

Class 9 String

5类常考问题:（**和array的有些问题相似，往往需要2个index来完成操作。**）

1. 相邻字母de-duplication aaaabbbb_ccc ->ab_c
2. replace e.g. replace empty space “ ” with “20%”

2.1. www.yahoo.com?info=flower_lsdkjfk

3. reverse (swap) e.g. **I love yahoo** -> yahoo love I
4. substring → strstr
   4.1. regular method
   4.2. Robin-Carp (hash based string matching) & KMP (Knuth–Morris–Pratt)
5. other operations
   5.1. move letters around e.g. abc123 -> a1b2c3
   5.2. permutation (use DFS)
   5.3. concatenation

"apple"
Popular representations of characters:

1. ASCII representation of a letter: A == 65, a == 97
For example, in c++

```
char x = 'A'
printf("%d", x)  // print 65
```

2. **Unicode** : the latest version of Unicode contains a repertoire of more than 110,000 characters covering 100scripts and various symbols.

Question 1, how to iterate through a string

1.1 c-style `char* p3 = "apple\0";` // what is the last element in this string? '\0'

```
char *p3 = "apple";
while (*p3 != '\0') {
    cout<<*p3 <<endl;
    p3++;
}
```

1.2 c++ style

```
int i = 0;
string aa = "apple2\0";      // '\0' is following the last element.
while (aa[i] != '\0') {    // or you can use i < aa.size().
    cout<< aa[i++];
}
cout<<endl;
cout<<"size of aa == "<<aa.size()<<endl;  //size is 6 or 7?
```

1.3. How to convert from C style to C++ style, that is (**char* → string**)

```
constructor1 string (const string& str);  //1
constructor2 string (const char* s);      //2
```

E.g.,

```
char *p1 = "abc"; // p1 is a pointer (== address) that points a char string

string s1(p1) ;
string a = "apple";
string b(a);
```

1.4. How to convert from C++ style to C style, that is (**string → const char* or char***)

E.g.,

1.4.1 string → const char*

```
string s2 = "abc";
const char* p2 = s2.c_str();
```

1.4.2 string → char* (rarely used)

```
// using C++ new/delete operators.
string s2;
char *c3;
c3 = new char[s2.size() + 1];
memcpy(c3, s2.c_str(), s2.size() + 1); // destination, source, size);
```

Question 2, remove **duplicated** and **adjacent** letters (leave only one letter in each duplicated section) in a string.

E.g.,

```
index        0 1 2 3 4 5 6 7……
string s = "a a a a a b b b b b _ c c c c  a ";    -> "ab c"
           i           j
             s[++i] = s[j++]

             i++;
             s[i] = s[j];
             j++;
```

```
public String deDuplicate(String input) {
        if (input == null || input.length() <= 1) {
                return null;
        }
        char[] inputChar = input.toCharArray();
        int global = 0;
        for (int index = 1; index < inputChar.length; index++) {
                if ( inputChar[global] != inputChar[index]) {
```

```
                inputChar[++global] = inputChar[index];
                index++;
            } else {
                continue;
            }
        }
        return inputChar.sub(0, global + 1);
}
void deduplication(string& input) {
  int i = 0;
  int j = 1;
  while (j < input.size()) {
      while (input[j] == input[i] && j < input.size()) {
          j++;
      }
      if (j < input.size()) {
         input[++i] = input[j++];
      }
  }
  input = input.substr(0, i + 1);
  return;
}
```

Question 3 (a). in URL encoding, whenever we see an empty space " ", then we need to replace it with "20%", so in this context, how could we perform this encoding for a given string?

For Java user: assume the input is in CharArray

yahoo/?q=flo**X**wer**X**market → yahoo/?q=flo**20%**wer**20%**market

Step 1: traverse the whole string and find how many occurrence of " ".
Step 2: Calculate the size of the new string : new_size = old_size + 2 * times_of_X
Step 3: yahoo/?q=flo**X**wer**X**market__← 20%market

p1 → p1' (become longer)
p2 → p2' (become shorter) ab → a

Question 4.
(4.1)

```
apple    elppa      → elppa
i    j       ij
```

(4.2) I love yahoo -> **yahoo love I**

M1: use stack, but it requires O(n) space complexity.
Step 1: delimiter : " " // 分隔符 → find each word, and reverse it?
yahoo
love
I
yahoo love I

M2: I love yahoo -> **yahoo love I**
i j
Step 1: reverse each word
i evol oohay
Step 2: reverse the whole sentence
yahoo love I

Question 5. **substring problem:** how to determine whether a string is a substring of another string.

full string
sub-string

corner case: if full string's length < sub-string return false;

```
s1 = "a b c d e";
          i

s2 = "c d";
       j
```

Method1:
Step 1 define a helper function to determine whether two words are identical
bool checkIdentical(string s1, int index1 , string s2)

Step 2, for each index i (i = 0), check the substring of s1, starting with index i to determine whether this substring is identical to s2 (by using this helper function)

O(m*n) where m and n are sizes of string s1 and string s2, respectively

Method2:

Robin-Carp (**hash based st**ring matching)

```
s1 = "a b c d e";
       a b   if we can(can we?)compose a hash function that is O(1)
         b c
           c d
             d e

s2 = "c d"; → 45678
```

Assumption1: if we can hash s2 to an integer that is unique compared to any other string with two letters. e.g., hash("cd") → 45678

Assumption2: let's assume we only have 26 type of letters in the string. **'a' -- 'z' (26进制)**

"c d" - - > 2*26^1 + 3*26^0

```
s1 = "a b c d e";
       a b   if we can(can we?)compose a hash function that is O(1)
         b c
           c d
             d e

s2 = "c d"; → 45678
```

ab ->bc when we increment index by 1 to the right, a is removed, and b's index changes from 0 to 1, and then we add c to the right most position

a b

1*26^1 + 2*26^0

b c

2*26^1 + 3*26^0

1. remove the leftmost item from the polynomial function

2. all the rest items of (ab's hashed value) x 26

3. add new item c

Question 6 Repeatedly de-duplicate adjacent repeated letters :

```
 0 1 2 3 4 5 6 7
 a b b b b a z x→ abbbbaz → z
```

```
             i
```

Use a stack to store the last letter that is known (so far) to be non-duplicated.

```
i = 0,  push stack → ||a
i = 1,  push stack → ||a b
i = 2,  b == stack.top() :i++  || a b
i = 3,  b == stack.top() :i++  || a b
i = 4,  b == stack.top() :i++  || a b
i = 5,  a != stack.top().   pop stack (b is out ),
        a == stack.top(). i++  || a,
i = 6,  z != stack.top()   pop stack (a is out). z is in the stack.
i = 7,
```

output == (keep popping all letters out of the stack and **reverse** the order.)

```
00 void removeDuplicate(string& s) {
01    if(s.length() <= 1)
02         return;
03    vector<char> st;
04    int i = 1;
05   st.push_back(s[0]);
06   while(i < s.size()) {
07         char c = s[i];
08         if (st.size() > 0 && s[i] == st[st.size()-1]) {
09              while (i < s.size() && c == s[i]) {
10                i++;
11              }
12              st.pop_back();
13         }
13         else {
14              st.push_back(s[i]);
15               i++;
16         }
17   }
18   s.clear();
19   for (int j = 0; j < st.size(); j++) {
20      s += st[j];
21   }
22 }
```

Homework:
Question 2
Question 3
Question 4
Question 5 (method 1)
Question 6 (use stack) (todo: Yuan Liang)

Class 10 System design I (web applications)

Design a web crawler

Purpose: write a server (crawler) to follow links on each web page and download all the content on the web pages.

The problem is how to explore the link network (e.g., do not download content from one web page twice).

Single machine solution: Depth first search to follow links and mark visited to already visited pages.

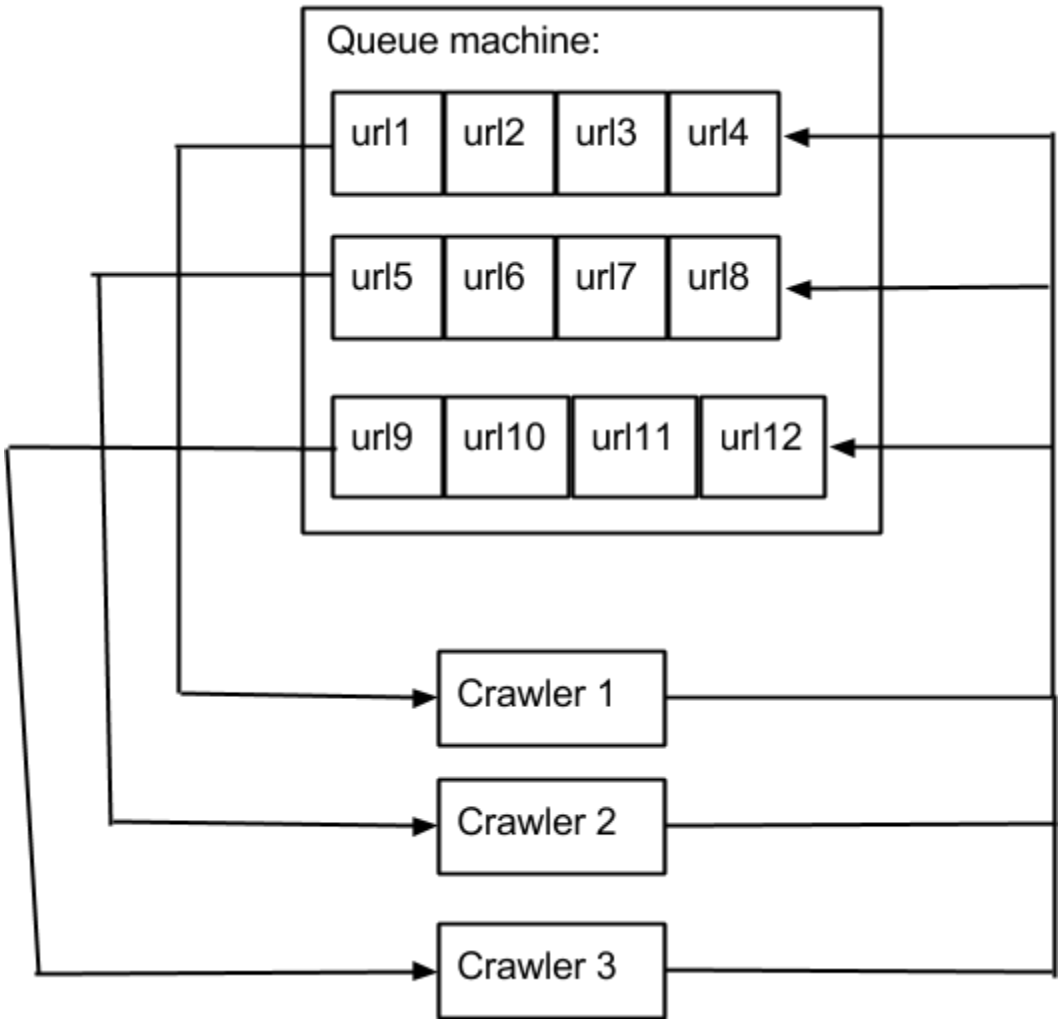
Pseudo-code:

```
hash_set visited;
void crawl(current_url) {
   if (visited.count(current_url) > 0) {
      return;
   }

   visited.insert(current_url);
   parse current_url;

   for each url on current_url {
       crawl(url);
   }
}
```

Bottleneck:
- network transmission is slow
- locally store all visited information for all web pages need a lot of space.

can be more scalable and speed up a lot by distributed system:

The problem for a distributed crawler system is how to maintain the information and synchronize with each other. e.g., one machine has visited one page, how does the rest of the machines know?



The queue is sharded by url. One url is deterministically going to one of the queues. Each crawler corresponds to one of the queues, i.e., it will only claim url to crawl from one of the queues. Then, the crawled page will produce more urls as candidates to crawl, the crawler will enqueue those candidates to their corresponding queues based on the same url sharding.

Each crawler also maintain a local visited page copy for pages it has visited and check to see if it needs to crawler the claimed url.

Design a feed product

Requirements:
1. each user could have 1- ~1000 friends, can get stories from your friends in real time.
2. scalability:
    - millions of query per second (qps)
    - a lot of data to store, cannot be stored on one machine, needs distributed system
3. can do ranking in real time

Two models:
1. push model
2. pull model

Push model:

when there is a story happened, it's pushed to all of the friends.

e.g., we have u1, u2, u3, u1 is friend of u2 and u3.

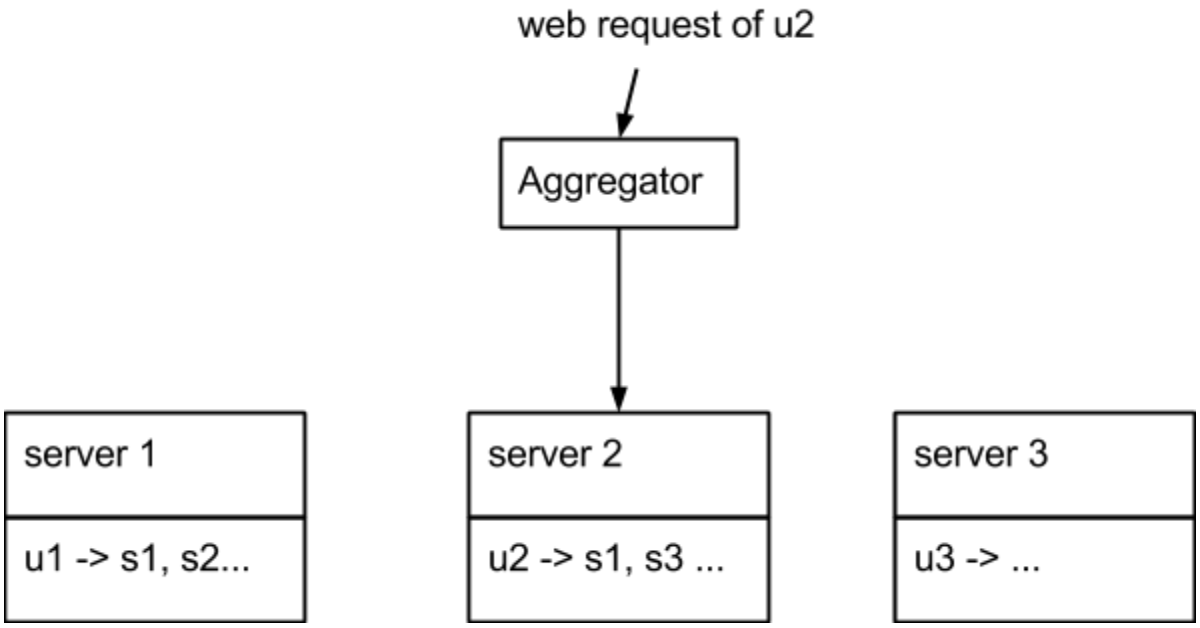
Each user maintain a list of friends' stories that **are eligible to** be displayed to himself:

u1 -> s1, s2
u2 -> s1, s3, s4
u3 -> s3, s5

We call this structure an **inverted index**. Also needs a **forward index** to store information for each story, e.g., click rate for each story, etc., mainly for the purpose of ranking:

s1 -> info for s1
s2 -> info for s2
s3 -> info for s3
s4 -> info for s4

The architecture looks like:



now u1 has a story (e.g., posted a status update), we will push this story, s6, to u2 and u3:

u1 -> s1, s2
u2 -> s1, s3, s4, s6
u3 -> s3, s5, s6

Pros:

- real time update
- For scalability: can shard by user: e.g., u1's index goes to machine 1, u2's index goes to machine 2….
- easy to implement and understand

Cons:

- duplicate storage for each story: one story needs to be stored n times (n = number of friends)
- not very each to scale, you have to duplicate the forward index on all machines
- one user's request is handled by only one machine (not parallel), so speed is slow

Pull model:

Each user maintains a list of stories of **his own**

For example, u1 has story s1, s2, s3 of his own, u2 has s4, s5, s6 of his own, u3 has s7, s8, s9 of his own, the inverted index will look like:

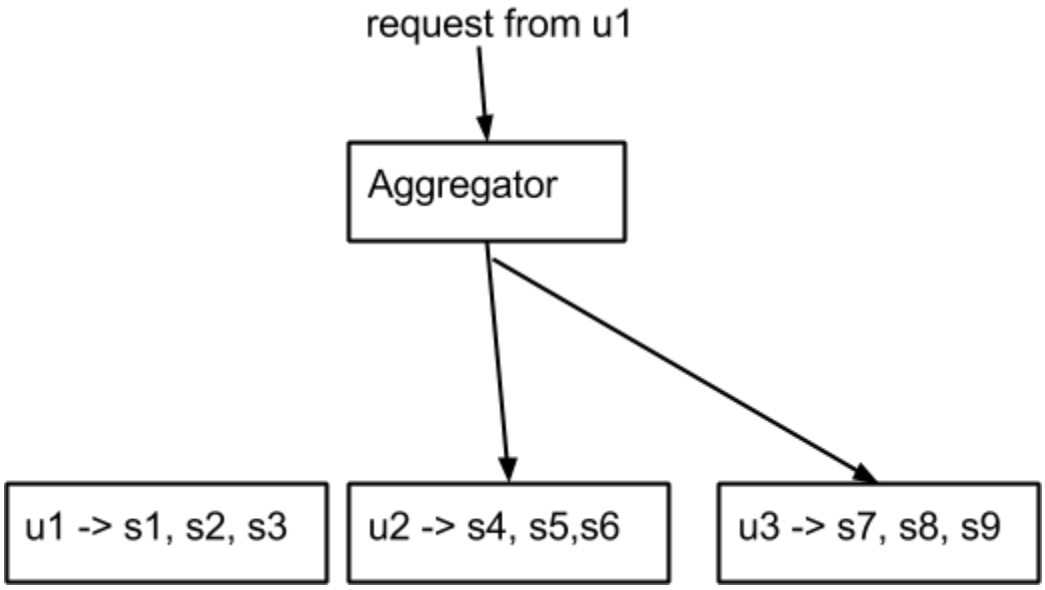
u1 -> s1, s2, s3 ...
u2 -> s4, s5, s6 ...
u3 -> s7, s8, s9 ...

Similarly, we still need the forward index to store information for each story

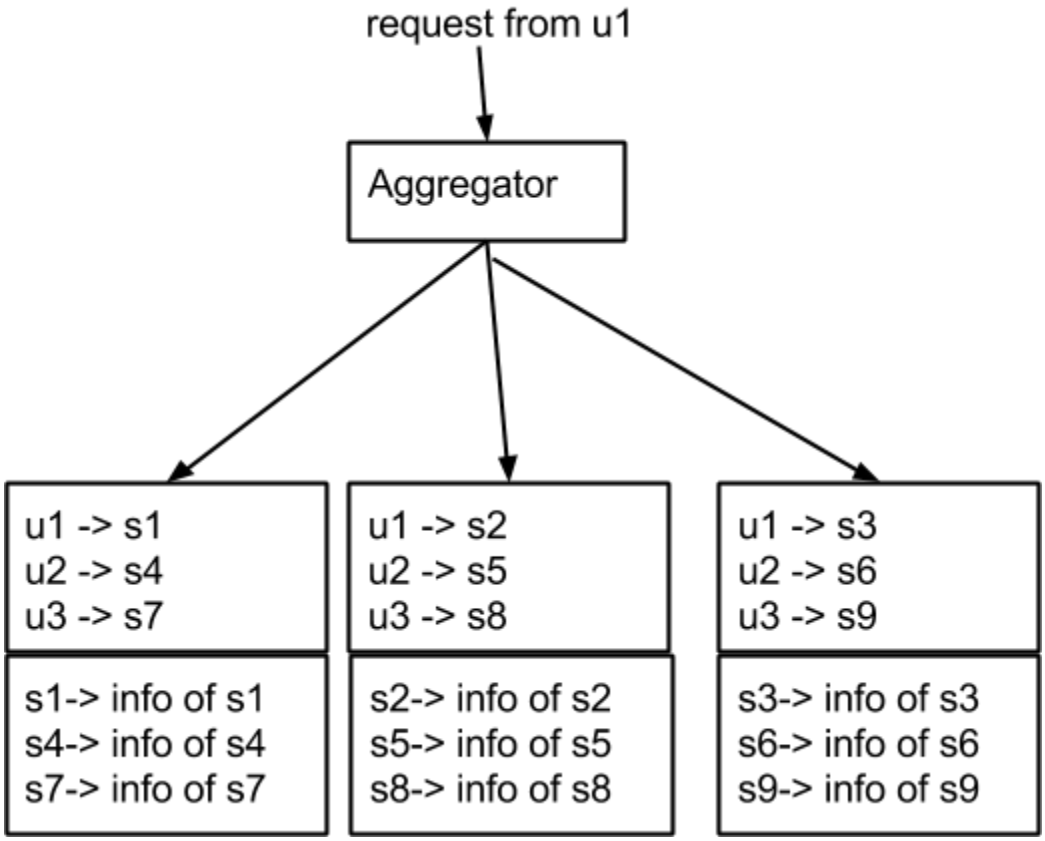
s1 -> info for s1
s2 -> info for s2
s3 -> info for s3
s4 -> info for s4

Now, u1 comes to the site, and he is friend of u2 and u3. We will send request to fetch stories of his friend, u2 and u3. In the example, we will get s4, s5 and s6 from u2, and s7, s8 and s9 from u3. Then we merge them into s4, s5, s6, s7, s8, s9 to display.

The architecture looks like:



The good thing is that, now, you can do (partial) ranking in parallel on each index server, then, each index server returns a ranked list to aggregator, and aggregator will do a merge sort for the final ranked list. E.g., u2 returns s5, s4, s6, and u3 returns s8, s9, s7, we just need a two way merge sort of (s5, s4, s6) and (s8, s9, s7).



Now, each index server only need to store all information of stories sharded to itself. e.g., index server 1 only store s1, s4, s7. After getting request from aggregator to fetch stories for u2 and u3 (who are u1's friend), index server 1 do 3 things:
1. get s4 and s7 from inverted index,
2. fetch ranking info for s4 and s7 and rank them, say, the rank result is (s7, s4)
3. return (s7, s4) to aggregator

Aggregator receives the partially ranked list from index server 1,2,3, it will do a merge sort finally and return it to browser to display.

Cons:
- more complicated to implement and understand

pros:
- no duplicate information
- ranking can be done in parallel
- real time update is easier

Design a friend suggestion service

Requirements:
1. be able to recommend not-connected friends based on your current friend list

2. real time adjustment of candidate list: especially important for new users, e.g., you just added one friend, we should be able to adjust our recommendation candidates based on the newly added friend.

3. accuracy: recommended friends should be your true friends, we also want to recommend at the right moment measured by the highest conversion rate.

4. diversity: not only a few candidates rotating.

5. scalable: be able to handle millions of requests per second, and store various data for ranking purpose.

Set up an index: key is user, value is a list of existing friends

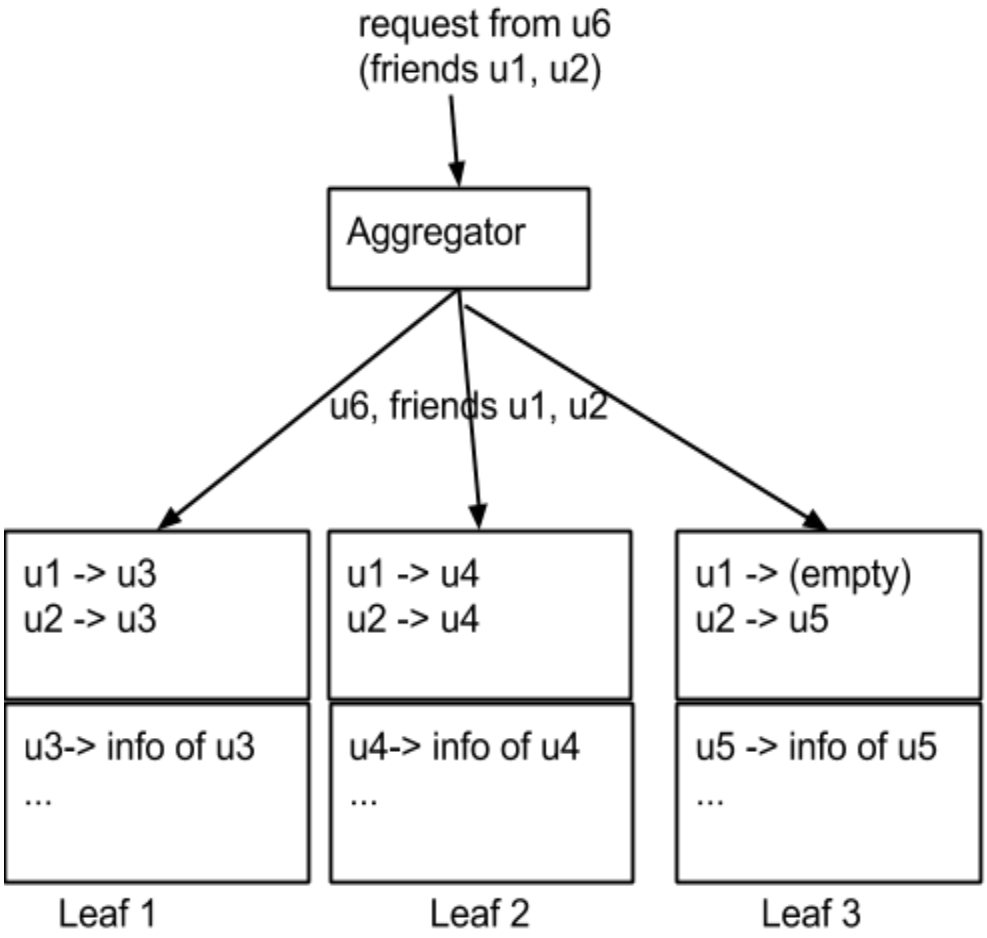
For example:

u1 -> u3, u4
u2 -> u3, u4, u5
…

And similar aggregator-leaf architecture as for feed service.

Aggregator: take request of a given user id, along with the friend list of this user.

Leaf: stores this index, shard by friend ids. and another forward index with user's information for ranking purposes.



Aggregator sends this user friend graph to all leaves and look for the second hop for candidate with mutual friends.

For example, u6 comes to the site, it has u1 and u2 as friends. this information is duplicated to leaf 1, 2, 3. On each leaf, it looks for the current friends of u1 and u2, and compute mutual friends in real time. E.g., I know u6 and u3 has 2 mutual friends. The sharding schema is key here, since one candidate's information is all on one shard, so you can do ranking in parallel in each of the leaf. Aggregator only needs to aggregate partially ranked candidates after all leaves finish.

Real time update:

If a user added a new friend, we will immediately update the graph stored in the leaf for this user. also, we can fetch the most updated friend graph each time a request is generated (for the friend list in request).

For example, if u1 added a new friend, u7. Then, its list becomes:

u1 -> u3, u4, u7

u6's request will see u7 as a candidate immediately.

Question, what do you do for user's without any friends?

Answer: use pending friend request, etc.

ranking signals:

- user specific information: e.g., total number of friends (representing how active the user is). will be stored in the forward index for each user.

- number of mutual friends: can be computed in real time during the request.

- timing information for each existing connection: store them in the index, i.e., along each edge, e.g., u1 has u3 (1 year ago), u4 (1 month ago), u5 (today) as friends.

homework:

Question: given billions of users, how to compute number of mutual friends for each pair?

hint: use map reduce

input:
u1 has friends u3, u4, u5
u2 has friends u3, u4

output:
(u3, u4) 2
(u4, u5) 1
(u3, u5) 1

Class 11 Object Oriented Design (1)

==========

I. Motivation
II. Basic Concepts
III. OOD in practice

目标:
1) 了解掌握Object Oriented Programming/Design的基本概念
2) 对于一个实际/面试设计问题,能够按照正确的步骤给出面向对象的设计
3) 通过课后的学习和练习, 深入了解掌握一种面向对象的程序语言(Java, C++, ...)的语言特性, 能够把OOD的基本概念对应到具体的语言特性上

I. Motivation of Object Oriented Programming
1.1 Structured programming: code + data
1) 如果一组data总是同时出现,同时总是调用一组function对这些data进行各种操作?
例子: List: data + operations on data
2) 如果只想暴露有限的接口和数据给**"调用方"**?
暴露(让别人调用): add(E element), get(int index), remove(int index), size()
隐藏: 如何组织存储数据, 如何track size的变化
3) 如果想重用部分程序,同时根据不同的情况进行扩展?
List → Stack, Vector
4) 如何更好的进行模块化测试?

1.2 OOP的一些优点 (http://en.wikipedia.org/wiki/Object-oriented_programming):
Modular, reusable software systems: 模块化,可重用
1) Increased understanding
通过class/object的定义来描述系统, 体现对系统的理解
2) Ease of maintenance
数据封装 (Data Encapsulation), 隐藏实现细节
3) Ease of evolution
通过继承(Inheritance)重用已有代码,同时针对不同的用例提供特定的实现

==========

II. Basic Concepts
2.1 Class and Object
Class: a blueprint for a data type, scheme
Object: a specific realization of any class

```
public class Employee {
  static double bonusRate = 0.1;
```

```
  final String name;
  int age;
  String id;
  int salary;
  int level;

  Employee(String name, int age, String id, int level) {
//    this.name = name;
    this.age = age;
    this.id = id;
    this.level = level;
  }

  void printInfo() {
    System.out.println("Name: " + name + "; Age: " + age + "; ID: " + id);
  }

  void setId(String id) {
    this.id = id;
  }

  void setAge(int age) {
    this.age = age;
  }

  public int calculateSalary(double performanceScore) {
    // calculate salary based on the employee's level and performance score
  }

  public static void main(String[] args) {
    Employee e1 = new Employee("ZhangSan", 20, "111-11-1111", 4);
    e1.printInfo();
    Employee e2 = new Employee("LiSi", 30, "222-22-2222", 6);
    e2.printInfo();

   Employee.bonusRate…. // correct
   e1.bonusRate… // javac warning
  }
}
```

2.2 Data Encapsulation: Data Abstraction and Access Labels

1) Providing only essential information to the outside world and hiding their background details

2) Separate interface and implementation
3) Access Labels: public, private, protected, default

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

4) Advantages:
Class **internals are protected** from inadvertent user-level errors, which might corrupt the state of the object.
The class implementation may **evolve** over time in response to changing requirements or bug reports without requiring change in user-level code (easier to maintain the **compatibility**).

```
public class Employee {
  private final String name;
  private final String id;
  private int age;
  private int salary;
  private int level;

  public Employee(String name, int age, String id, int level) {
    this.name = name;
    this.age = age;
    this.id = id;
    this.level = level;
  }

  public void printInfo() {
    System.out.println("Name: " + name + "; Age: " + age + "; ID: " + id);
  }

  public int calculateSalary(double performanceScore) {
    // calculate salary based on the employee's level and performance score
    // We can change the salary calculation method details here without affecting other
    // code
  }

  public String getName() {
    return name;
```

```
  }

  public String getId() {
    return id;
  }

  public int getSalary() {
    return salary;
  }

  public int getLevel() {
    return level;
  }

  public static void main(String[] args) {
    Employee e1 = new Employee("ZhangSan", 20, "111-11-1111", 4);
    // get the performance score of ZhangSan
    double score1 = …..;
    // wrap the salary calculation logic inside of Employee instead of here
    System.out.println("The salary of " + e1.getName() + " is " + e1.calculateSalary(score1));
    Employee e2 = new Employee("LiSi", 30, "222-22-2222", 6);
    // get the performance score of LiSi
    double score2 = …..;
    System.out.println("The salary of " + e2.getName() + " is " + e2.calculateSalary(score2));
  }
}
```

2.3 Inheritance, Interface, Abstract Class

2.3.1 Inheritance(继承): Base class (父类), Derived class (子类)

Employee → Manager/Programmer

2.3.2 Interface and Abstract Class

C++: Abstract class

Java: Interface and Abstract class

```
public interface Employee {
  // no data fields
  public int calculateSalary(double performanceScore); // no method implementation
                                                       // before Java8
}
```

OR

```
public abstract class Employee {
  // we can have some data fields here
  // constructor method OK
  // methods with implementation also OK

  /** calculate salary based on the employee's level and performance score */
  public abstract int setSalary(double performanceScore);
}
```

不能直接instantiate abstract class或者interface:

```
Employee e = new Employee();  Wrong!

public class Manager extends Employee { // Employee is an abstract class
   // TODO: provide setSalary method implementation
}
Employee e2 = new Manager();  Correct!
```

2.4 Polymorphism and Overriding

Override: 子类重写父类方法

```
public class Manager extends Employee {
  // …

  @Override
  public int calculateSalary(double performanceScore) {
    salary = getBaseSalary(level) * (1 + 0.25 * performanceScore);
    return salary;
  }
  // …

  public void manage() {
     //
  }
}

public class Programmer extends Employee {
  // …

  // method defined only in Programmer
  public void program() {
     // ….
  }
```

```
  @Override
  public int calculateSalary(double performanceScore) {
    // salary = getBaseSalary(level) * (1 + 0.15 * performanceScore)
  }
  // ...
}
```

Polymorphism (多态): A call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Example:

```
Employee e1 = new Programmer(...);
e1.calculateSalary(0.95);
e1.program(); // wrong!!!
((Programmer) e1).program(); // bad behavior
((Manager) e1).program(); // run time error

Programmer e3 = new Programmer();
e3.program(); // correct

Employee e2 = new Manager(...);
e2.calculateSalary(0.85);
```

课后阅读:
1. 如何实现Polymorphism?
(http://mortoray.com/2011/08/09/how_polymorphism_works_part1/,
http://mortoray.com/2011/08/12/how_polymorphism_works-part2-virtual-table/,
https://www.seas.upenn.edu/~cit595/cit595s10/lectures/polymorphism.pdf)

III. OOD in practice
经典考题: 运用OOD设计一个Parking Lot

步骤:
1. 明确这个程序/系统是干什么的 (分析用户需求)
用途: 描绘parking lot的建筑,还是专注于车辆停泊的监控?
决定了系统的Interface (public API)

(对于给定的车辆, 提供整个Parking Lot里可用的泊位的数量)

其他问题:
一层还是多层?
能停多种车辆? 怎么定义车辆?

是否需要关心车辆的大小?
是否需要记录现在停着的车?
…

2. 确定类和类关系 (classes and their relationships)
Vehicle, Parking Spot, Level, Parking Lot…

常用的类关系
Association: a general binary relationship that describes an activity between two classes.

Vehicle -- Parking Spot

Aggregation/Composition: a special form of association, which represents an ownership relationship between two classes. (has-a)

Level -- Parking Spot
Parking Lot -- Level

Inheritance

Vehicle -- Car, Truck

3. 对于定义出清晰的API: 别人怎么调用你的程序?
不要关心实现细节!!!

functionality → 能不能把一个给定的车park进去? // 还有几个位置? // 在哪一层有几个位置?

```
public class ParkingLot {
  private Level[] levels;

  /** given a vehicle, tell me whether I can park it? */
  public boolean canPark(Vehicle v) {
    // TODO: 遍历level, 对每一个level, call Level#canPark(Vehicle)
  }
}

public class Level {
  // tracking Parking Spots
  // canPark
}

class ParkingSpot {
  // boolean canPark(Vehicle): check size
```

```
   // API to track whether the spot is available
}

public interface Vehicle {
  // API: getSize() method
}

car class, bus, … implements Vehicle
```

4. 定义类内部的其他方法和数据

```
Assumption:
1. multiple levels
2. Vehicle size

API: boolean canPark(Vehicle v); // parking lot level

classes:
ParkingLot, Level, Vehicle, ParkingSpot, Car, Truck

public enum VehicleSize {
  Compact, Large;
}

public interface Vehicle {
  public VehicleSize getSize();
}

// Car class
public class Car implements Vehicle {
   public VehicleSize getSize() {
     return VehicleSize.Compact;
   }
}

// Truck class
public class Truck implements Vehicle {
   public VehicleSize getSize() {
     return VehicleSize.Large;
   }
}
```

```
public class ParkingSpot {
  private VehicleSize size;
  private Vehicle currentVehicle; // null if no vehicle is parked inside

  public boolean canPark(Vehicle v) {
    if (currentVehicle == null) {
      // check size:
      return canFit(this, v));  // use VehicleSize.value
    }
    return false;
  }

  // record a vehicle is parked in by updating the currentVehicle field
  public void park(Vehicle v) {
    // check canPark again to make sure it is available: optional
    currentVehicle = v;
  }

  public void leave() {
    currentVehicle = null;
  }
}

public class Level {
  private final List<ParkingSpot> spots;

  public Level(int numOfSpots) {
    // do nothing will cause javac error since spots needs to be initialized
    spots = new ArrayList<ParkingSpot>(numOfSpots);
  }

  public boolean canPark(Vehicle v) {
    for (ParkingSpot s : spots) {
      if (s.canPark(v)) {
        return true;
      }
    }
    return false;
  }

  public boolean park(Vehicle v) {
    for (ParkingSpot s : spots) {
      if (s.canPark(v)) {
```

```
        s.park(v);
        return true;
      }
    }
    return false;
  }
}
```

Homework:
1. 根据你最常用的语言(e.g., C++ or Java), 对每一个基本OOP概念整理出相关的语言规范
2. 运用OOD设计一个in-memory File System (重点在API的定义)
3. 设计实现HashMap/Hashtable, 并阅读Java/C++的实现源码
4. 完成Parking Lot的例子

Class 12 Object Oriented Design (2)

Simple Design Patterns
目标:
1) 了解几种常见的设计模式, 深刻理解它们的motivation
2) 能够在平时的编程中用到其中的2,3种设计模式

==========

Builder (http://en.wikipedia.org/wiki/Builder_pattern)
An object creation software design pattern.

例子:
class User中包括有多个optional的data field:

```
public class User {
  private final String firstName;    //required
  private final String lastName;    //required
  private int age;                  //optional
  private String phone;         //optional
  private String address;       //optional

  public User(String firstName, String lastName) {
    this(firstName, lastName, 0);
  }

  public User(String firstName, String lastName, int age) {
```

```
    this(firstName, lastName, age, '');
  }

  public User(String firstName, String lastName, int age, String phone) {
    this(firstName, lastName, age, phone, '');
  }

  public User(String firstName, String lastName, int age, String phone, String address) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.phone = phone;
    this.address = address;
  }
}
```

问题: 对于每一种成员变量的组合, 都需要提供一个相应的constructor来赋初值.

解决方案一: 不提供constructor, 只提供相应的setter/getter.

```
public class User {
  private final String firstName; // required
  private final String lastName; // required
  private int age; // optional
  private String phone; // optional
  private String address;  //optional

  public User(String last, String first) {
    lastName = last;
    firstName = first;
  }

  public String getFirstName() {
        return firstName;
  }
  public String getLastName() {
        return lastName;
  }
  public int getAge() {
        return age;
  }
  public void setAge(int age) {
        this.age = age;
  }
```

```
  public String getPhone() {
        return phone;
  }
  public void setPhone(String phone) {
        this.phone = phone;
  }
  public String getAddress() {
        return address;
  }
  public void setAddress(String address) {
        this.address = address;
  }
}
```

缺点:
1. 无法确定一个User object什么时候构建完成
2. 没法定义immutable data fields (每一个data field都可以被set任意多次)

使用Builder Pattern:

```
public class User {
  private final String firstName; // required, and immutable
  private final String lastName; // required
  private final int age; // optional, and immutable
  private final String phone; // optional
  private final String address; // optional

  private User(UserBuilder builder) {
        this.firstName = builder.firstName;
        this.lastName = builder.lastName;
        this.age = builder.age;
        this.phone = builder.phone;
        this.address = builder.address;
  }
  public String getFirstName() {
        return firstName;
  }
  public String getLastName() {
        return lastName;
  }
  public int getAge() {
        return age;
  }
  public String getPhone() {
```

```
        return phone;
}
public String getAddress() {
        return address;
}

public static class UserBuilder {
        private final String firstName; // these two are required!
        private final String lastName;
        private int age = 0;  // default value is 0
        private String phone = “”; // default value is an empty string
        private String address; // default value is null

        public UserBuilder(String firstName, String lastName) {
                this.firstName = firstName;
                this.lastName = lastName;
        }

        // all the following methods are used to set values for optional fields
        public UserBuilder age(int age) {
                this.age = age;
                return this;
        }
        public UserBuilder phone(String phone) {
                this.phone = phone;
                return this;
        }
        public UserBuilder address(String address) {
                this.address = address;
                return this;
        }
        public User build() {
                return new User(this);
        }
}

public static void main(String[] argv) {
  User user = new User.UserBuilder(“San”, “Zhang”)
                        .age(25)
                        .phone(“1234567890”)
                        .address(“Fake address”)
                        .build();
  // …..
```

```
  }
}
```

==========

Factory method (http://en.wikipedia.org/wiki/Factory_method_pattern)
A creational pattern which uses factory methods to deal with the problem of creating objects without specifying the exact class of object that will be created.

Example:

```
public class Room {
   // data fields
}
public class OrdinaryRoom extends Room {
    // …
}
public class MagicRoom extends Room {
    // …
}

// Some other Room implementations maybe...

public class MazeGame {
   public void createGameWithTwoRooms() {
      // Call factory method makeRoom for object creation, instead of using
      // constructors. In this way createGameWithTwoRooms() does not need to know
      // the details about what types of Room will be used, thus can be directly reused
      // by derived classes.
      Room room1 = makeRoom();   // Room room1 = new Ordin…();
      Room room2 = makeRoom();   // Room room2 = new OrdinaryRoom();

      room1.connect(room2);
      this.addRoom(room1);
      this.addRoom(room2);
   }

   /** Factory method. return type is base class or interface */
   protected Room makeRoom() {
      return new OrdinaryRoom();
   }
}

public class MagicMazeGame extends MazeGame {
   /** Override the makeRoom method in the base class */
```

```
    @Override
    protected Room makeRoom() {
        return new MagicRoom();
    }

    // MagicMazeGame can directly use createGameWithTwoRooms defined in base
class

 main() {
   MazeGame game1 = new MazeGame();
   game1.createGameWithTwoRooms();  → 2 ordina..
   MagicMazeGame game2 = new Magic….

   game2.createGameWithTwoRooms();  → 2magic rooms
}
}
```

==========

Abstract Factory (http://en.wikipedia.org/wiki/Abstract_factory_pattern)
The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes.

Example:
http://en.wikipedia.org/wiki/Abstract_factory_pattern#Pseudocode

Factory interface (or abstract class): 定义了接口

```
interface GUIFactory is
  method createButton()
      output:  a button
```

实现Factory interface的子类实现了具体的object creation过程

```
class WinFactory implementing GUIFactory
class OSXFactory implementing GUIFactory
```

调用factory interface的程序只需要知道相应的output的type, 不需要知道具体是怎样生成这个output的

```
class Application is
  constructor Application(GUIFactory factory) is
      input:  the GUIFactory factory used to create buttons
    Button button := factory.createButton()
    button.paint()     // defined in the Button base class
```

传入的factory object根据具体的OS环境或者configuration在运行时确定.

```
main() {
   Read the configuration file
   If the OS specified in the configuration file is Windows, then
     Construct a WinFactory
     Construct an Application with WinFactory
   else
     Construct an OSXFactory
     Construct an Application with OSXFactory
}
```

==========

Singleton (http://en.wikipedia.org/wiki/Singleton_pattern)
Ensure a class has only one instance, and provide a global point of access to it.

Example:

```
public class Singleton {
   private static final Singleton INSTANCE = new Singleton();

   private Singleton() {
      // do sth….
   }

   public static Singleton getInstance() {
      return INSTANCE;
   }
}
```

==========

Delegation (http://en.wikipedia.org/wiki/Delegation_pattern)
In software engineering, the delegation pattern is a design pattern in object-oriented programming where an object, instead of performing one of its stated tasks, delegates that task to an associated helper object. The delegation pattern is one of the fundamental abstraction patterns that underlie other software patterns such as composition (also referred to as aggregation), mixins and aspects.

Example:

```
interface I {
   void f();
   void g();
}

class A implements I {
   public void f() { System.out.println("A: doing f()"); }
```

```
    public void g() { System.out.println("A: doing g()"); }
}

class B implements I {
    public void f() { System.out.println("B: doing f()"); }
    public void g() { System.out.println("B: doing g()"); }
}

class C {  // 这里实际就是之前我们提到的 class composition
    I i = null;
    // delegation
    public C(I i){ this.i = i; }
    public void f() { i.f(); }
    public void g() { i.g(); }

    // normal attributes
    public void to(I i) { this.i = i; }
}

public class Main {
    public static void main(String[] args) {
        C c = new C(new A());
        c.f();     // output: A: doing f()
        c.g();     // output: A: doing g()
        c.to(new B());
        c.f();     // output: B: doing f()
        c.g();     // output: B: doing g()
    }
}
```

==========

Decorator (http://en.wikipedia.org/wiki/Decorator_pattern)

1. abstract class A 定义接口

```
// The abstract Coffee class defines the functionality of Coffee implemented by decorator
public abstract class Coffee {
    public abstract double getCost(); // Returns the cost of the coffee
    public abstract String getIngredients(); // Returns the ingredients of the coffee
}
```

2. A的简单实现 B

```
// Extension of a simple coffee without any extra ingredients
public class SimpleCoffee extends Coffee {
```

```
    public double getCost() {
        return 1;
    }

    public String getIngredients() {
        return "Coffee";
    }
}
```

3. abstract class Decorator D, 一般也是A的子类, 它的实现一般是基于A的一个delegator

```
// Abstract decorator class - note that it extends Coffee abstract class
public abstract class CoffeeDecorator extends Coffee {
    protected final Coffee decoratedCoffee;
    protected String ingredientSeparator = ", ";

    public CoffeeDecorator (Coffee decoratedCoffee) {
        this.decoratedCoffee = decoratedCoffee;
    }

    public double getCost() { // Implementing methods of the abstract class
        return decoratedCoffee.getCost();
    }

    public String getIngredients() {
        return decoratedCoffee.getIngredients();
    }
}
```

4. concrete decorator E, F, G, 在D的基础上进行各种添加

```
// Decorator Milk that mixes milk with coffee.
// Note it extends CoffeeDecorator.
class Milk extends CoffeeDecorator {
    public Milk (Coffee decoratedCoffee) {
        super(decoratedCoffee);
    }

    @Override
    public double getCost() { // Overriding methods defined in the abstract superclass
        return super.getCost() + 0.5;
    }

    @Override
    public String getIngredients() {
```

```
        return super.getIngredients() + ingredientSeparator + "Milk";
    }
}

// Decorator Whip that mixes whip with coffee.
// Note it extends CoffeeDecorator.
class Whip extends CoffeeDecorator {
    public Whip (Coffee decoratedCoffee) {
        super(decoratedCoffee);
    }

    public double getCost() {
        return super.getCost() + 0.7;
    }

    public String getIngredients() {
        return super.getIngredients() + ingredientSeparator + "Whip";
    }
}

// Decorator Sprinkles that mixes sprinkles with coffee.
// Note it extends CoffeeDecorator.
class Sprinkles extends CoffeeDecorator {
    public Sprinkles (Coffee decoratedCoffee) {
        super(decoratedCoffee);
    }

    public double getCost() {
        return super.getCost() + 0.2;
    }

    public String getIngredients() {
        return super.getIngredients() + ingredientSeparator + "Sprinkles";
    }
}

使用时可以在B上同时添加E,F,G
public class Main {

    public static final void main(String[] args) {
            Coffee c = new SimpleCoffee();
            System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());
```

```
        c = new Milk(c);
        System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());

        c = new Sprinkles(c);
        System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());

        c = new Whip(c);
        System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());

        // Note that you can also stack more than one decorator of the same type
        c = new Sprinkles(c);
        System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());
    }
}
```

Homework1: 用java/C++实现Builder里的example
Homework2: 用java/C++实现Abstract Factory里的example

Last lesson homework:
OOD -- in-momery file system

File, Directory,
Tree

/

/User /Application /Library

/User/zhangsan /User/lisi /Application/app1 ….

…….

class Entry: name, id, access time, creation time, owner,...., Folder parent
class File extends Entry: type, ...,
class Folder extends Entry: List<Entry> children

Class 13 Dynamic Programming 1

hey, everyone.

Fibonacci:
Fn = Fn-1 + Fn-2
F1 = F2 = 1

Recursion Solution: O(2^N)

```
00public int fibN(int n) {
01    if (n == 1 || n == 2)  // base case
02          return 1;
03    return fibN(n-1) + fibN(n-2);    // recursive rule
04}
```

Fib(5) = Fib(4)+Fib(3) = (Fib(3)+Fib(2)) + (Fib(2)+Fib(1)) = ((Fib(2)+Fib(1))+Fib(2)) + (Fib(2)+Fib(1))

`

Iteration Solution: O(n)

```
public int fibN(int n) {
      int[] fibsFound = new int[n+1];    // sub-solutions
      fibsFound[1] = 1;
      fibsFound[2] = 1;
      for (int i = 3; i <= n; i++) {
            fibsFound[i] = fibsFound[i-1] + fibsFound[i-2];
            //  fib[2] ⇒ f[0] + f[1]
            //  fib[3] =  f[1] + f[2]
            …...
      }
      return fibsFound[n];
}
```

Iteration Solution: O(1) space

```
public int fibN(int n) {
      int x = 1;        // i-2
      int y = 1;        // i-1
      for (int i = 3; i <= n; i++) {
            int z = x + y;
            x = y;
            y = z;
      }
```

```
        return y;
}
```

Always describe your algorithm before writing on the white board

DP principle:

1. **Subproblems (Repeated calculations)** - allow more efficient calculation of larger problems
2. **Base case** - size == 1 (usually)

2. **Longest Ascending Subarray**

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

```
00 public int getLongest(int[] s) {
01    if (s == null || s.length < 1) {
02          return 0;
03    }
04    int max = Integer.MIN_VALUE;
05    for (int i = 0; i < s.length; i++) {// Subarray beginning index
06          for (int j = i+1; j < s.length; j++) { // end index
07                if (check(s, i, j)) {
08                      max = Math.max(max, j - i);
09                }
10          }
11    }
12    return max;
13 }
O(n^3)
```

```
for (int i = 0; i < s.length; ++i) {
  int l;
  for (int j = i+1; j < s.length; ++j) {
     if (s[j] < s[j-1]) {
        break;
    }
    l++;
 }
```

```
 if (l > max) {
   max = l;
  }
}
return max;
O(n^2)
```

3 **Longest Ascending Subsequence**

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

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

{1} => {1} length = 1
{1,2} => {1,2} length = 2
{1,2,4] => {1,2,4} length = 3
{1,2,4,3} => {1,2,3} length = 3

{1,2,4,3,7}  S(5) =   max { S(1) if a(1)<a(5),
                            S(2) if a(2)<a(5),
                            S(3) if a(3)<a(5),
                            S(4) if a(4)<a(5) }
```

化大问题为小问题

```
S(7) {1,2,4,3,7,6,4} = max { S(i) if (a(i) < a(7) }
S(1)=1 S(2)=2 S(4)=3
S(7) = 4

O(n^2)
native solution O(2^n)



100 200 300 400 1 2 500 3 <-
            S(4)=4  S(6)=2

int longestASeq(int []a, int n) {
int S[] = new int[n];
for (int i = 0; i < n; ++i) {
```

```
  // the current problem we want solve S(n)
  S[i] = 1;
  for (int j = 0; j < i; ++j) {
// the subproblem we want to check if can provide a good solution
    if (a[j] < a[i] && S[j]+1 > S[i]) {
      S[i] = S[j]+1;
    }
  }
}
int m = 0;
for (int i = 0; i < n; ++i) {
  m = max(m, S[i]);
}
return m; //max(S.begin(), S.end());
```

// 精髓就是化大问题为小问题，不重复计算
// 先写递推公式

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**).

—| — — — — — — —
— —| — — — — — —

一个数n，拆成m个数，使得乘积最大

```
6 = 3+3, 3*3 = 9
6 = 2+2+2, 2*2*2 = 8
6 = 1+1+4 or 1+1+2+2
max = 9
```

大问题是什么，小问题是什么

```
max of f(n,m) =
 max ((n-1, m-1)*1, (n-2, m-1)*2......(1, m-1)*(n-1))
recursion solution:

Complexity?
exponential!
O(N^M)

f(100,100) = f(99,99) , f(98,99)
```

要算50^50 次 f(50,50)

Iteration O(N*M):

```
for (int i = 1; i < m; ++i) {
  for (int j = 1; j < n; ++j) {
    D[i][j] = max(D[i-k][j-1]*k, for k = 1..i-1)
  }
}
return D[n][m];
```

Recursion O(N^M)

```
int f(int n,int m) {
  int X = 0;
  for (int i = 1; i < n; ++i) {
    X = max(X, f(n-i,m-1)*i);
  }
  return X;
}
```

Memorization: O(N*M)

```
int D[][] = {-1};
int f(int n,int m) {
  if (D[n][m] >= 0) {
    return D[n][m];
  }
  int X = 0;
  for (int i = 1; i < n; ++i) {
    X = max(X, f(n-i,m-1)*i);
  }
  D[n][m] = X;
  return m;
}
```

4. **Jump Game**

Given an array of non-negative integers, you are initially positioned at the first index of the array. **Each element in the array represents your maximum jump length at that position.** Determine if you are able to reach the last index.

For example:
index 0 1 2 3 4
A = [2,3,1,1,4], return true.
B = [3,2,1,0,4], return false.

化大问题为小问题

```
[2,3,1,1,4]
S(4) <- 能跳到 A[4]
? = {
S(3) if A[3]>=1
S(2) if A[2]>=2
S(1) if A[1]>=3
S(0) if A[0]>=4
}
S(n) ??=??? S(n-i)
```

Iteration Solution: O(N^2)

```
S[0..n-1] = false; // S[i] position i is reachable from position 0
S[0] = true; // position 0 is always reachable - base case
for (int i = 1; i < n; ++i) {
  for (int j = 0; j < i; ++j) {
    S[i] |= S[j] && A[j] >= i-j;
  }
}
return S[n-1];
```

Memorization Solution: (O(n^2))

int X[]; { -1 - 没算过， 0 - false， 1 - true }

```
int S(int n) {
  if (X[n] >= 0) {
    return X[n] == 0 ? false : 1;
  }
  for (int i = 0; i < n; ++i) {
    if (S(i) && A[i] >= n-i) {
      X[n] = 1;
      break;
    }
  }
  X[n] = 0;
  return X[n];
}
```

Recursion Solution: O(N!), O(2^N) - Exponential

炒股票：

array of stock prices, each number is the price for a given day. Find out what's the max money you can make by buying and selling stocks. You can only buy/sell one unit of stock on each day.

```
input: [5, 3, 6, 2, 7, 8, 5, 1]
           b  s  b     s
solution: 3+6 = 9
```

石子归并（供学有余力的同学）：
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。

Class 14 Dynamic Programming 2

`Greedy Algorithm` - 贪心

Previous question: Given an array of non-negative integers, you are initially positioned at the first index of the array. **Each element in the array represents your maximum jump length at that position.** Determine if you are able to reach the last index.

For example:

index 0 1 2 3 4
A = [2,3,1,**1**,4], return true.
B = [3,2,1,0,4], return false.

Q0 Minimum Number of Jumps
Given the same setup as the Jump problem, can you return the minimum number of jumps needed to reach the end instead of just whether or not it is possible to reach the end?

BFS (breadth-first-search)

DP (dynamic programming)

Reduce big problem to smaller problem

A0,A1,...An-1

S(n) - minimum number of jumps from position 0 to position N-1

S(1), S(2), … S(n-1) - subproblem

S(n) = ? S(1), S(2), … S(n-1)

```
   V  V
A = [2,3,1,1,4]
```

```
S(3) = minimum number of jumps from position 0 to position 2
S(1)=0
S(2)=1
for S(3), you have two options:
S(1)+1 because A(0)=2 >= 2-0, so you can go from 0 to 2 directly
S(2)+1 because A(1)=3 >= 2-1, so you can go from 1 to 2 directly
S(3) = min{S(1)+1, S(2)+1}
     = 1

S(n) = min { S(n-i)+1 if A(n-i-1) >= i, i=1..n-1 }
S(n) = min ( a(i-1) >= (n - i ) ? s(i) + 1 : MAX_VALUE, i=x...x )
```

Iteration Solution: (O(N^2))

O(1+2+3+4+...+N) = O (N(N+1)/2) = O(N^2)

```
for (int k = 0; k <= n; ++k) {
  S[k] = -1;
}
S[1] = 0;
for (int k = 2; k <= n; ++k) {
  for (int i = 1; i < k; ++i) {
    if (A[i-1]>=k-i && (S[k]==-1 || S[i]+1 < S[k]) && S[i]>=0) {
      S[k] = S[i] + 1;
    }
  }
}
if (S[n] >= 0) { return S[n]; }
return NO_ANSWER;
```

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}.

```
Problem:
S(n) = max sum of a subarray if A[n] is picked (subarray ends at
A[n])
S(0) = 1
S(1) = 3
S(2) = S(1)>0? S(2)=S(1)+A(2)=7
S(n) = ? S(0),S(1)...S(n-1) ?
S(n) = max( S(n-1) + A(n), A(n) )
                  ^         ^-------- the length of solution is one
                  |_________ the length is one plus the length of best
solution at n-1
Base Case: ?
S(0)= a[0]
Final Solution: ?
return max (S[0]...S[n-1])

A  = {-1,-2,-3,1,2,3,-1,-1, 2, -3}
S  = {-1 -2 -3 1 3 6, 5, 4, 6,  3}

Iteration Solution: O(n) space -> O(1)
S[0] = A[0];
m = 0;
for (int i = 1; i < n; ++i) {
  S[i] = max(S[i-1]+A[i], A[i]); //S[i-1]is negative then you pick
A[i]
  m = max(m, S[i]);
}
return m;

S(n) is max sum of any subarray between 0 and n-1  ← wrong direction
try to limit the freedom of the state so that you can write down the
iterative formula
```

Q2 Dictionary word problem

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

bob
cat
rob
Word: bcoabt
Solution: False
Word: **bobcatrob** || catbob
Solution: True

Hash Set:
{bob, cat, rob}
any lookup of a string in the dictionary is O(1)

Big Problem:
A[0..n-1]
S(N) = A.substring(0, N) can be concatenated by words in the dictionary
solution: return S(N-1)

S(N) = { true if S(i) is true and A.substring(i+1, N) is in the dictionary, i=0..N-1}
or { A.substring(0,N) is in the dictionary) }

Iteration Solution:

```
for (int i = 0; i < n; ++i) {
  S[i] = dict.contains(A.substring(0,i+1));
  for (int j = 0; j < i; ++j) {
     if (S[j] && dict.contains(A.substring(j+1,i-j)) {
        S[i] = true;
        break;
     }
  }
}
return S[n-1]
```

O(n^3) because substring() is linear
there is a O(N^2)

Q3. Minimal 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

Recursion Solution: O(4^N) -> Memoriazation Solution: O(N^2)

```
00 public int editDistance(String word1, String word2) {
     if ( S[word1.length][word2.length] 算过 ) { return
S[word1.length][word2.length]; }

     // Base case
01   if (word1.isEmpty()) return word2.length();
02   if (word2.isEmpty()) return word1.length();

     // (a) Check what the distance is if the characters are equal
     // and we do nothing first
03   int nothing = Integer.MAX_VALUE;
04   if (word1.charAt(0) == word2.charAt(0)) {
```

```
05          nothing = editDistance(word1.substring(1),
06                                 word2.substring(1))
07    }
      // (b) Check what the distance is if we do a Replace first?
08    int replace = 1 + editDistance(word1.substring(1),
                                     word2.substring(1))
      // (c) Check what the distance is if we do a Delete first?
09    int delete = 1 + editDistance(word1.substring(1), word2);
      // (d) Check what the distance is if we do a Insert first?
10    int insert = 1+ editDistance(word1, word2.substring(1));
      // Return best solution
11    S[word1.length][word2.length]= min(nothing, replace, delete,
insert);
      return S[word1.length][word2.length];
}
```

Problem Definition:

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]

S[i][j] = { 4 situations:

```
            S[i-1][j-1]    S1[i]==S2[j],
            S[i-1][j-1]+1  S1[i]!=S2[j],
            S[i][j-1]+1    insert a S2[j],
            S[i-1][j]+1    insert a S1[i] }
```

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

      // len1+1, len2+1, because we will return dp[len1][len2]
03    int[][] dp = new int[len1 + 1][len2 + 1];
      // BASE CASES
      // fill in the first column (column 0)
04    for (int i = 0; i <= len1; i++) {
05          dp[i][0] = i;
06    }
      // fill in the first row
07    for (int j = 0; j <= len2; j++) {
```

```
08          dp[0][j] = j;
09      }

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

        //if last two chars equal, this is the best we can do
14              if (c1 == c2) {
15                  dp[i][j] = dp[i-1][j-1]; // case 1
16              } else {
17                  int replace = dp[i-1][j-1] + 1;//case2
18                  int insert = dp[i-1][j] + 1; // case3
19                  int delete = dp[i][j-1] + 1; // case4
20                  int min = Math.min(replace, insert);
21                  min = Math.min(min, delete);
22                  dp[i][j] = min;
23              }
24          }
25      }
26      return dp[len1][len2];
27  }
```

		s2	**s**	**g**	**h**	**j**
	ind	0	1	2	3	4
s1	0	**0**	**1**	**2**	**3**	**4**
a	1	**1**	**1**	**2**	3	4
s	2	**2**	1	2	3	4
d	3	**3**	2	2	3	4
f	4	**4**	3	3	3	4

x is the base case.

x= min(from left + 1, from topleft+1, from top+1)

dp[1][1] = max { dp[0][0]+1, dp[0][1]+1, dp[1][0]+1} = dp[0][0]+1 = 1

do[1][2] =

Q4 Largest square of 1's in a binary matrix

What is the edge length of the largest square of 1's in a given binary matrix. In this case, your solution should return 3 (3x3 square).

```
   cl     cr
0  0  0  0  0    rt 0
1  1  1  1  0              1
1  1  1  1  0
1  1  1  0  0    rb 3
1  1  1  0  0              4
```

Dimension Reduction: 2D -> 1D

Brute Force (蛮力算法) :

O(n^3) - (Xtop, Ytop), length; O(n^2) - scan all elements; O(N^5)

Solution A: O(N^3)

(Xtop, Ytop), length 0..n-1

S [N] [N][N]

S[1][N][N] = A // base case

maxLength = 0

For Length =2 To N:

```
  bool found = False;
  For TopX = 0 To N-Length {
     For TopY=0 To N-Length {
        S[Length][TopX][TopY] =
      S[Length-1][TopX][TopY] &&
      S[Length-1][TopX+1][TopY] &&
      S[Length-1][TopX][TopY+1] &&
      S[Length-1][TopX+1][TopY+1];

      found |= S[Length][TOpX][TopY];
    }
 }
 if (found) { maxLength = Length; }
}
return MaxLength;
```

Solution B: O(N^2)

```
0  0  0  0  0
1  2  3  2  0     S[i][j] = min(S[i-1][j], S[i-1][j-1], S[i][j-1])+1
1  2  3  1  0
1  2  2  0  0
1  1  1  0  0
```

Class 15 Dynamic Programming 3 训练课

请同学们打开各自期中考试的google doc， 上训练课用。

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

= =

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** are not the same:
        - 1.2.1. e.g. DP1 课后题： 沙子归并
        - 1.2.2. e.g. 强化练习题： 切木头

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

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),

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

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

index 0 1 2 3 4 5 6
A[i] **1, 2, 4,** -1, -2, -1 -100 **98** 10000 |||| -1 10
S[i] 1 3 7 6 4 3 -97 98 10098

base case: M[0] = A[0]
M[i] : resprents the largest sum of subarray which end with A[i];
M[i] = M[i - 1] + A [i] if (M[i - 1] > 0)
A [i] if(M[i - 1] <=0)

Question 1. 一个 unsorted一维数组最长连续1的问题。

A[n] = 00110011101111**1**10010100111111111
M[n] = 00120012301234500101001234567**89**

1. Base case: one element M[0] = 1 if A[0] == 1; M[0] =1 otherwise;
2. Induction rule:
   2.1. M[i] represents longest consecutive 1s ends at i, from left-hand side.
   M[i] = 0 if A[i] = 0
   M[i] = M[i-1] + 1 if A[i] = 1;

Question 2a. Given a Matrix that contains only 1s and 0s, how to find the largest cross which contains only 1s, with the same arm lengths and the two arms join at the central point of each other.

Exmaple:

```
0100
1111        for the pink 1--->   min(2, 3, 3, 2) ==>2
0100
0100
```

width of the matrix == n
Naive solution: starting from each "1" expand it to four directions altogether.
O(n^3)

step1: for each row, from left to right, call Q1 O(n)
since we have n row, then == > O(n x n)
step 2: for each row, from right to left, call Q1
⇒ O(n^2)
Step 3: for each column
Step 4……
O(4* n^2)

Step 5, iterate over M[i][j] ⇒ min of all 4 direction for each M[i][j] ===> **O(n^2)**

```
0100
1111        for the pink 1--->   min(2, 3, 3, 2) ==>2
0100
0100
```

```
M[i][j]   first direction: from left → right
0100
1234
0100
0100
```

```
M[i][j]   second direction: from right → left
0100
4321
0100
0100
```

```
M[i][j]   third direction: from up → bottom
0100
1211
0300
0400
```

```
M[i][j]   fourth direction: from bottom → up
```

Question 2b. Given a Matrix that contains only 1s and 0s, how to find the largest **X** which contains only 1s, with the same arm lengths and the two arms join at the central point of each other.

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

```
1111111     1111
2222222     2222
3333333     3333

44444444444

size == nxn
```

Naive solution:

1. how many sub-matrix in the big matrix?

C_n2 * Cn^2 → O(n^4)

2. for each sub-matrix, O(n^2) to calculate the sum of each sub-matrix;

Total time == **O(n^6).**

```
1111111     1111
2222-> [1,4]22     2222
3333333 ->[2,6]    3333
```

```
if we know the M[i][j]:represents the sum of all elements in a
sub-matrix, whose upleft corner is [0][0], and bottom-right corner is
[i][j].
```

sum of (222,333) = M[2][6] - M[0][6] - M[2][3] + M[0][3]

generalized coordinates top-left corner at [i,j], and bottom corner at [k][t]

sum of [i][j]-[k][t] = M[k][t] - M[i-1][t] - M[k][j-1] + M[i-1][j-1]

==========

Then the question is converted to how to get M[i][j] (with 00 as the top-left, and i,j as the bottom right)

Example:

original data(matrix)

1 2 3

1 2 3

1 2 3

M[i][j] = M[i-1][j] + sum_from_the_left_in_this_row[i]
1 3 6 | |
2 6 x = 6 + 6 == 12
3 9 18
sum_so_far = 6

填M[i][j] 表格，需要O(N^2)

O(Cn2 * Cn2) = O(n^4)

==========
Optimal solution: O(N^3)
Question 0 (most popular DP question) **Largest sum of a subarray O(N)**

Example

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

2 3 4 -6 3 2 1 1 4 row_upper =1
1 2 3 -4 5 6 7 8 9 row_bottom=2
| | | | | | | | |
3 5 7 -10 8 8 8 9 13 = sum of row1 && row2

3 5 7 -10 8 8 8 9 13
2 3 4 6 3 2 1 1 4

Example: original matrix
2 3 4
1 2 3 top row == 1
3 5 7 bottom row ==2
2 3 4
---
sum of each column **M[i][j]** **O(N^2)**
2 3 4
3 5 7
6 10 14
8 13 18

6-2 10-3 14-4
\ | /
4 7 10 → as input array = {4 7 10} for Q0

O(N * N^2) ⇒ O(N^3)

Step1: fill in the table M[i][j] O(N^2)
Step 2: choose up and bottom rows O(N^2)
for each pair of up-bottom row
O(n) + O(n)
| |
拍扁 call Q0

How many ways to choose the upper row and bottom row of an arbitrary sub-matrix
O(n^2)
So total time = O(n^2) + O(N^3) → O(N^3)

Homework:
Q1
Q2a and Q2b
Q3a and Q3b