**第16节课时间: 2014年12月12，星期5，太平洋时间晚上7:00w**

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/join/163634853>

**Announcement:**

1. 班主任赵老师 email: [laioffer9@gmail.com](mailto:laioffer9@gmail.com) Tel 323-893-9079
2. 本班级QQ群 316871642 （加入群请使用真实姓名）
3. 上机课时间:

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

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

**Homework Solution**

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

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

**课程列表**

[Class 1 Array and Sorting Algorithms](#h.lb0mgtyfil2z)

[Class 2 Recursion and Binary Search](#h.ke6supc74d2x)

[Class 3 Stack & Linked List](#h.ixb8wyl2nbrx)

[Class 4 Binary Tree & Binary Search Tree](#h.rxxw9lcmmws3)

[Class 5 Heap & Graph + Search Algorithms](#h.2avqt3shgifm)

[Class 6 DFS & Hashtable](#h.1bx02ukgv8yu)

[Class 7 Midterm 1 (数据结构自测)](#h.tjhdtwb7bykb)

[Class 8 Bit representation of a number and bit operation](#h.db3ze5i6b7cz)

[Class 9 Object Oriented Design (1)](#h.m83bewnhdwmd)

[Class 10 Object Oriented Design (2)](#h.3bjbb3bw1tbp)

[Class 11 Dynamic Programming (1)](#h.8bf1kd5e6jrj)

[Class 12 String](#h.y8lr07hjqc4b)

[Class 13 Object Oriented Design (习题课)](#h.8unu1shm8p1d)

[Class 14 Dynamic Programming (2)](#h.4i2a2rvkcbv)

[Class 15 System design I (web applications)](#h.oamzmdz6ooa1)

Name List

|  |
| --- |
| **Andi Shen** |
| **Mengyu Liu** |
| **JIAHAO LIANG** |
| **Junwei Yuan** |
| **TIAN LUAN** |
| **Zishu Zhang** |
| **Mingjing Mao** |
| **JIAO CHEN** |
| **Fangli (Simon) Hou** |
| **Anqi Shao** |
| **Yunxi Jia** |
| **Runtian Liu** |
| **Fan Yang** |
| **Jenny ZHANG** |

**Xiaosong Wang**

**Ren Yu**

**Yongxiang Tang**

**Qian Ding**

**Yuanchao Xia**

**Hao Zhang**

**Boxia Feng**

**What** do you expect from this class?

1. 技术准备
   1. 基础知识: 算法，和其他cs知识
   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. **30 classes ＋ 1 codeLab/week**
2. 简历 (扬长避短)
   1. 自我的能力展现
   2. correctness, consistency, insights
   3. 精炼淘汰(什么该放，什么不该放)
   4. 如何通过hr，拿到面试的渠道
3. 什么时候知道自己ready
   1. mock interview (技术＋沟通)
   2. 冷冻期
   3. 未雨绸缪
4. 面试技巧和注意事项
   1. coding style
   2. 交流方式
   3. 碰上困难如何处理
   4. 如何准备自己的问题
5. 面试英文书信交流 / 如何negotiate offer
   1. We can back you up
   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](http://www.collabedit.com))

# Class 1 Array and Sorting Algorithms

**Data structure** is a particular way of organizing [data](http://en.wikipedia.org/wiki/Data_(computing)) in a computer so that it can be used [efficiently](http://en.wikipedia.org/wiki/Algorithmic_efficiency).

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];

void printArray(int\* a, int n) {

if (! a)

return;

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

std::cout << a[i] << std::endl;

}

}

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]);

}

E.g. int a[10];

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.

O(1)

**example:** int a[4] = {-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 }

**Time complexity** = O(1+2+3+ ... + n-1+n) = **O(n^2)**

**Space complexity** = O(1)

**2. Merge sort**

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

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

/ \

n/2 **13579** [line 08] breaking point 86420 O(1)

/ \ / \

n/4 **135**  79 864 20 O(2)

/ \ / \

n/8 **13 5** 79 86 4 2 0 O(4)....

**/ \**

**...1 1 3 5** 7 9 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; // mid is == 4

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 4 6 8

j

combine question

final solution = {0, 1, 2 …. 9}

**Time complexity** = **O(nlogn) Log\_2(2^64-1) → 64**

**O(64 \* n)**

**Space complexity** = O(n/2 + n/4 +... 1) == O(n);

**Discussion**:

Merge sort重点在于分而治之，找到中点，两边分别merge sort，递归到最底层

再一层层merge回来。要注意的是首先递归结束条件(left >= right)， 其次是merge

函数的写法，需要一个同样大小的helper array,先把初始的array拷贝到helper里面，

Merge的过程就是用两个指针在helper里面移动，哪个对应元素小就挑哪个放回原始

Array里面。

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

2) 什么面试中一个类型的题？

e.g., A1B2 | C3D4 -> ABCD1234 vs ABCD1234 → A1B2C3D4

A1 | B2

A 1 B 2

A1 B2

AB12 CD34

ABCD1234

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

New combine function to define the ordering of letter vs numbers

<http://www.tutorialspoint.com/java/character_isdigit.htm>

**3. Quick Sort** Average time = O(nlogn) , worst n^2

**1 | 2 3 4 5 n**

**p 2 3 4 5 n-1**

**p n-2**

**O(n^2)**

1 9 8 **5** 3

**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

13 **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,

What is the worst case scenario of Q-sort?

1 4 3 5 7

piv 4357

piv 457

**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-> unknown area to explore ← j**

**Discussion:**

1) 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

babbbb b cccbbccc

**E.g., abXxxxxxxxxxxxxxxxxxxxxxc**

i j → k [j, k] 为未知探索区域

**E.g., aaaaa** **bbbbbxxxxxxxc**cccc

[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).

a:12345

**Homeworks**

1. Selection Sort

2. MergeSort

3. QuickSort

public void quickSort(int[] array, int left, int right) {

if (left >= right) {

return;

}

int pivotPos = partition(array, left, right);

quickSort(array, left, pivotPos - 1);

quickSort(array, pivotPos + 1, right);

}

private int partition(int[] array, int left, int right) {

int pivotPos = pivotPos(left, right);

int pivot = array[pivotPos];

// swap the pivot element to the rightmost position first

swap(array, pivotPos, right);

int leftBound = left;

int rightBound = right - 1;

while (leftBound <= rightBound) {

if (array[leftBound] < pivot) {

leftBound++;

} else {

swap(array, leftBound, rightBound--);

}

}

# Quicksort重点在于这个partition函数，返回了pivot index以及在其左右放定了比它小和大的元素

要小心的是quicksort本身红色部分，因为是递归，所以一定要有结束条件！用partition函数

返回的pivot index把array分成前后两部分，再对每一部分进行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 (if we can resolve the same problem but with a smaller size, then what is left to do in the current problem size n)

**2. Base case:**  smallest problem to solve

Example problem: [Fibonacci sequence:](http://en.wikipedia.org/wiki/Fibonacci_number)

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) **breaking point..** **+ fibo**(n-2); // Recursive rule

08}

**F(n == 4)**

/ \

Line07 **F(3) breaking + F(2)**

/ \ / \

**F(2) breaking**+ **F(1)** F(1) + F(0) ⇒ 0 +1 = 1

/ \

**F(1) + F(0)**

|

immediately return 1

O(2^n)

**1+ 2^1 + 2^2 + …. + 2^(n-1)**

**1 + 2 + 4 + 8 + 16 ….**

**Principle: we only care about the total number of leaf nodes → total number of nodes in the tree**

**4 → 3 → 2 → 1**

**F(n) ??? O(n)**

Time complexity == O(2^n)

Space complexity == O(n)

**Q2 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)

E.g. b == 1000

for (int i = 0; i < 1000; i++) {

}

O(b)

Recursion:

1. Base case: b == 0 or b == 1
2. Recursive rule: a^b ~ a^(b/2) \* a^(b/2)

int **a\_power\_b**(int a, int b) {

if (b == 0) { // base case

return 1;

} else if (b == 1) {

return a;

}

int half\_result = **a\_power\_b**(a, b/2); // determine whether b is even or not

if (b % 2 == 0) {

return half\_result \* half\_result;

} else{

return half\_result \* half\_result \* a;

}

}

1000

|

500

|

250 O(log\_2(n));

.. how many levels are there??????? log\_2(1000)

Node total == 2^(level) = 2^(log\_2(1000)) == 1000

Time complexity = O(log(n))

O(n)

**2. Binary Search**

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

1. Array has to be sorted. ascending or descending 1 2 3 5 7 9 …

2. Problem to solve?

l=5 r=5

index 0 1 2 3 4 5

data **1 2**  5 **7 9**

target == 6

a[1] = {1} ; target == 6;

**// Classical Version**

// return any target element’s index

00 **int** **binary\_search**(int a[], int size, int target) {

int start = 0;

int end = size - 1;

**while (start <= end) {**

int mid = start + (end - start) / 2;

if (target == a[mid]) {

return mid;

}

if (target < a[mid]) {

**end = mid - 1;**

} else {

start = mid + 1;

}

}

return -1;

}

**// Variant 1.1**

// return the index of the **first** occurrence of an element, say 5

L=0 r=6

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

// 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 }

void main() {

int a[7] = {4, 5, 5, 5, 5, 5, 5};

int target = 4;

cout<<"1st occurrence of "<< target<< " is: " << BinarySearch(a, 0, 6, target);

}

第一变种是要找第一个相等的元素，所以基本原理和binarysearch一样，只是找到了

相等的不能停下，需要继续往左做binarysearch，一直到left = right – 1了

为止。这时有可能和target相等的只有left, right两个位置的元素，分别比较下，

如果都不相等就返回-1.这里要注意的细节是必须优先比较left,再比较right,因为我们

想要最左边的。

**// 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 }

这次换做找最后一个相等的元素，和上题类似，只是变成了如果找到了不能停，继续

向右边做binarysearch.另外最后要优先比较right,再比较left。

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

**Variant 1.3**  how to find an element in the array that is **closest** to the target number?

**Target == 4; L=2 R=3**

**index 0 1 2 3 4**

// e.g. int a[6] = {1, 2, **3**, 8, 9};

**Variant 1.3a** how to find closest k-elements from a target number in a sorted array.

**// 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.

Anqi:

M1:

A[2/1000] : try step = 2/1000;

CASE 1: A[1000] == target, return true;

CASE 2: A[1000] > target, good we already that the word can only exist in A[0] -- A[999] if any.

CASE 3: A[1000] < target, double the step

step1 1000

step2 2000

step3 3000

step4 4000

**mid = (jump6+jump7)/2**

1 3, 6, 8 **……[500** ., 600 , 999, null NULL…. **NULL]** NULL NULL NULL

**Jump6** index=777 **Jump7**

LEFT RIGHT

**Further discussion about binary search:**

Why not jump\_step = jump\_step \* 10, instead of jump\_step = jump\_step \*2

Step by step == double

Step by step == 10\* times

**Assume the size of the dictionary is (n); // n is unknown**

**size == n**

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

**log\_2(n) > log\_10(n)**

**\_\_**

**\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| overshot index <= 2n**

Worst case: For jumping out

**For 2 times method:**

**log\_2(n)**

**For 10 times method:**

**log\_10(n)**

**=============================================**

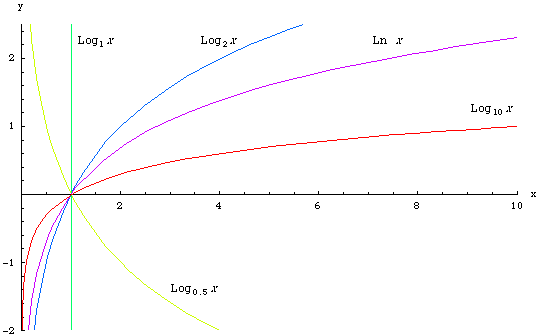
Worst case: For jumping in

**For 2 times method:**

**log\_2(2n)**

**For 10 times method:**

**log\_2(10n)**



**10 times - 2times =**

**(log\_10(n) + log\_2(10n)) - (log\_2(n) + log\_2(2n)) < 0**

**1 2 3 4**

**log\_10(n) - log\_2(n) + (log\_2(10n) - log\_2(2n)**

**1 3 2 4**

**-10000 log\_2(5) ~ 2.5**

**1-3 2-4**

**Homework1,** Classical binary search + Variant 1.1 + Variant 1.2 ＋ Variant 1.3

**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**5555555**67 **target = 5**, return 7.

# Class 3 Stack & Linked List

1. Queue

example: wait in a line, FIFO == **First in first out**

1. **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

X X X XX XX X

||--> current

**Two popular interview questions:**

**Question 1**: How could we implement a queue by **using two stacks**?

E.g. input order 1 2 3 4 →

output order 1 2 3 4

solution:

Stack1: insert → statck 1

Stack2: when popout,

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 for each action

push() O(1)

pop() **amortized** O(1)

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

stack 1 -- value ||-> -1 ……

stack 2 -- value ||-> -1

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+\*

a \* (b+c) → abc+\*

abc+ \*

i

a b c + d \*

(b + c) → d

a \* d → final ouput

**Linked List:**

Pointer basics (C/C++)

class Node {

int value;

Node\* next;

}

**head**

1 → 3 → 5 → 7 → 10 → NULL

f

LinkedList uses pointer to point to its next node/struct

**head**

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

prev curr next

int **CountNodeNumber**(Node\* head) {

if (head == NULL) {

return 0;

}

Node\* ptr = head;

int count = 0;

while (ptr ! = NULL) {

count ++;

ptr = ptr->next;

}

return count;

}

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

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

**head**

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

**head**

**Method1: Iterative Solution:**

**null ← Node1←**  Node2 ← Node3 **NULL**

**prev cur/next**

Node\* Reverse(Node\* **head**) {

if (head == NULL || head->next == NULL) {

return head;

}

Node\* **prev** = NULL;

Node\* **cur** = head;

**while** (cur != NULL) {

Node\* next = cur->next;

cur->next = prev;

prev = cur;

cur = next;

}

return **prev**; / / new head after reversing the whole linkedlist

}

链表反转就从头开始一个一个反过来，用curr和prev两个指针分别指向当前结点和

当前结点即将指向的结点，要注意的就是每次都要事先用next保存当前结点的下个结点，

因为即将改变这个next指针指向prev，如果不保存就没法update curr指针了。

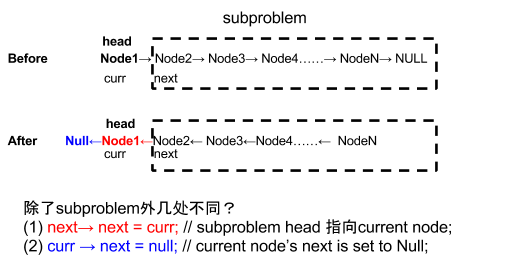
链表题通常都要考虑prev, curr, 和next指针。

**Method 2: Recursive Solution:**

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

1. **Base case**: head == NULL || head->next == NULL (only 0 or 1 node in the linked list)

2. **Recursive rule**: n-node linked list → (n-1) node linked list find the recursive rule!!



**// Recursion**

01 public Node **ReverseList**(Node head) {

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

03 return head; // base case;

04 }

05 Node **nextNode** = head.next;

06 Node **newHead** = **ReverseList**(**nextNode**); // recursive rule

07 nextNode.next = head;

**08 head.next = null;**

09 return **newHead**;

10 }

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

head next

**2nd call R-func: NULL← Node1 NULL<-** Node2← Node3  **return Node3**

head next

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

**3rd call R-func: NULL← Node1 NULL<-** Node2 Node3→ NULL **return Node3**

head next

**1st call R-func: Node1 <--** Node2 ← Node3 NULL return newHead(**Node3**)

head next

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

**2nd call R-func: Node1 →** Node2 <--> Node3 NULL return newHead(**Node3**)

head next

**==============================================**

**3rd call R-func: Node1 →** Node2 → **Node3** → NULL **// new head == Node3**

head

**常见考题：**

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

N1 → N2 → **N3** → N4 → N5 → N6 → NULL

slow

fast

M1: 1st iteration is to find the total number of node in the list.

2nd iteration is to find the n/2 node.

M2: fast/slow pointers

2. 用快慢指针判定一个linkedlist是否有环。

N1 → N2 → **N3** → N4 → N5

← -------------

Termination conditions (1) fast == NULL return false;

1. slow == fast return true;

3. Insert a node in a sorted linked list (simple)

x 1 → 3 → 6 → 9 → NULL

cur next 4

insert target 4

cur -> next = 4;

4-> next = nextNode

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 → N**n-1)**→ …**

**Step 1: find the middle node of the linkedlist**

**Step 2: reverse the 2nd half of the linked list**

**Step 3: merge two linked list**

**N1 → N2 → N3 → N4 → N5 → N6 →NULL**

**m1**

**send half : N6→ N5 → N4**

**merge: N1 → N2 → N3**

**6.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.**

**Anqi:**

two linked list to store all the nodes

(1)

One linked list is to store all the nodes whose values are **smaller** than the target value;

The other linked list is to store all the nodes whose values are **larger** than the target value;

(2)connect the two linked lists

(3) set tail->next = NULL;

**1 -> 3 -> 2 -> 5 -> 2 → 6 → 5**

**pre cur tail**

fakeSmallHead(0) → 1

smallTail

fakeLargeHead(0)

largeTail

public ListNode **partition**(ListNode head, int target) {

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

return head;

}

ListNode fakeHeadSmall = new ListNode(0);

ListNode fakeHeadLarge = new ListNode(0);  
 ListNode **smallTail** = fakeHeadSmall;  
 ListNode **largeTail** = fakeHeadLarge;

ListNode curr = head;

while (curr != NULL) {

if (curr.value <= target) {

smallTail.next = curr;

smallTail = smallTail.next;

} else {

largeTail.next = curr;

largeTail = largeTail.next;

}

curr = curr.next;

}

smallTail.next = fakeHeadLarge.next;

largeTail.next = NULL;

return fakeHeadSmall.next;

}

**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: 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 BS

Definition: at most two children nodes

class Node {

int value;

Node\* lChild;

Node\* rChild;

// Node\* parent;

}

10 == root

/ \

**5**  . 15

/ \ / \

**2** 7 12 20 ←-all leaf node’s level == 3

/ \

null null

**Pre-order** (for root/current node): 10 5 2 7 15 12 20

**In-order**: 2 5 7 10 12 15 20

**Post-order**: **2 7 5 12 20 15 10**

void **printBinaryTreePreOrder**(Node\* root) {

if (root == NULL) return;

cout << root->value << endl;

**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);

}

preorder, inorder, postorder都是利用堆栈的性质以某种方式push,pop以得到需要的

顺序。其中preorder是一路以根节点的right, left顺序压栈，以保证弹出的时候顺序是

Left, right。Inorder是一路先沿左子树到最底下一路压栈，然后一路弹出同时压入right。

Postorder需要两个栈，第一个栈就是为了调整顺序弹出，第二个栈接受第一个栈弹出的元素，

然后装满后把第二个栈元素依次弹出。因为弹出顺序是左右中，所以要保证入第二个栈顺序是

中右左。那么弹出第一个栈顺序就是左右。第一个栈放入根节点，弹出压入第二个栈，然后把

根节点左右孩子放入第一个栈，如此进行。。。

* **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; // base case;

03 }

04 if (Math.abs(**getHeight**(root.left) - **getHeight**(root.right)) > 1){

05 return false;

06 }

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

08}

10 root (2 times getHeight n/2 + n/2)-->O(n)

/ \

5(2 times getHeight n/2) 15(n/2) -->O(n)

total level = log(n) level

for each level O(n)

So, the total time complexity = **O(nlog(n))**

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

//get height of the tree

10 private **int** **getHeight**(TreeNode Node) { // time complexity = **O(n)**

if (Node == null) return 0; // base case

// recursive rule.

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

}

10 == root

/ \

**5**  . 15

/ \ / \

**2** 7 12 20 ←-all leaf node’s level == 3

/ \

time complexity of **getHeight** == O(n) // n here is the total number of node in this **sub-tree**

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

10

**5a | 5b**

**1a**  **3a** | **3b** **1b**

**2a 4a 6a 8a | 8b 6b 4b 2b O(n)**

void main() {

Node\* = root;

bool result = **Symmetric**(root->lChild, root->rChild);

}

**public bool Symmetric(Node root1 == 5a, Node root2 == 5b) {**

if(root1 == null && root2 == null) {

return true; // case 1

} else if( root1 == null || root2 == null) {

return false;

} else if(root1.val != root2.val) {

return false;

} // base case 3

return **Symmetric**(root1.left, root2.right) **&& Symmetric**(root1.right, root2.left);

}

}

Time complexity == O(n) // n is total number of node in the binary tree

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 (== not changed).

**8**

**/ \**

**4 5**

**|**

**3**

case1. 8a 8b case 2 8a 8b --2boo

/ \ / \ / \  **/ \**

**4a**  **5a**   **4b**  **5b**  **4a**  **5a** **5b 4b**

**| | | |**

**3 3 3 2 3**

**root1 root2**

**/ | | \**

**R x x x**

**/ | | \**

**R x x x**

public boolean **identicalHelper**(Node root1**=O(n)**, Node root2 **= O(n)**) {

if (root1 == null && root2 == null) {

return true;

}

if (root1 == null || root2 == null) {

return false;

}

if (root1.val != root2.val) {

return false;

}

// recursive rule

return ( (**identicalHelper**(root1.left, root2.left) && **identicalHelper**(root1.right, root2.right))  **||**

(**identicalHelper**(root1.left, root2.right) && **identicalHelper**( .right, root2.left))

);

}

**root1 (n) root2(n) log\_2(n) layers**

**/ | | \**

**/||\ /||\ /||\ /||\**

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\_2 (n) = 2^(2log\_2 (n)) = 2^(log\_2 (n^2)) = O(n^2)**

**2. Binary Search Tree**

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

10 [min = -infinity, max = + infinity] == root

/ \

**5**  . 15 (**min** = parent.value =10, **max** = parent.max ]

/ \ / \

**2** 7 xxx(**min** = parent.min, **max** = parent.value = 15) 20 ←-all leaf node’s level == 3

/ \

null null

Definition: for all nodes belong to the left-subtree of sub-root, they should be smaller than the value of the root node.

public boolean isBST(TreeNode root) {

return isBSTHelper(root,Integer.MIN\_VALUE, Integer.MAX\_VALUE);

}

private boolean **isBSTHelper**(TreeNode **root**, int **min**, int **max**) {

if (root == null) {

return true;

}

if (root.val <= min || root.val >= max) {

return false;

}

return **isBSTHelper**(root.left,min,root.val) && **isBSTHelper**(root.right,root.val,max);

}

**Homework1**: preorder, inorder and postorder traverse a binary tree (recursion + **iterative**)

**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)

Preorder顺序是中左右，打印结果就是一直沿着左边打，然后打出左子树所有右

节点（从下往上），右边类似，只要记得打印完根节点总是先往左走一直到不能再左

为止就行了。

Inorder顺序是左中右， 打印结果是左子树一直打印三角形，从最左下底层开始，一直

打到根节点，然后右子树斜向上45度打左中，左中，左中。。。

Postorder顺序是左右中， 打印结果是左子树从最底层开始左右中，一直到根节点的左

子树根，然后右子树所有左结点斜向下45度打到底，再斜向上45度打所有右结点，

一直到根节点的右子树根，最后打印根节点。

# Class 5 Heap & Graph + Search Algorithms

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

Example

1

/ \

3 7

/ \ /

5 4 8

**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](http://zh.wikipedia.org/w/index.php?title=%E5%AE%8C%E5%85%A8%E6%A0%91&action=edit&redlink=1)。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**: sort the whole array, and return A[0..k-1] **O(nlog(n))**

**M2:** Step 1 heapify the whole array to be a **MIN\_HEAP**

keep poping k elements out. Done!  **O(n) + O(k log(n)) → O(n) +O(7logn)**

**M3**: quick sort to find a pivot that is located at the k-1th location O(n^2)

**M4**: **max\_heap**????

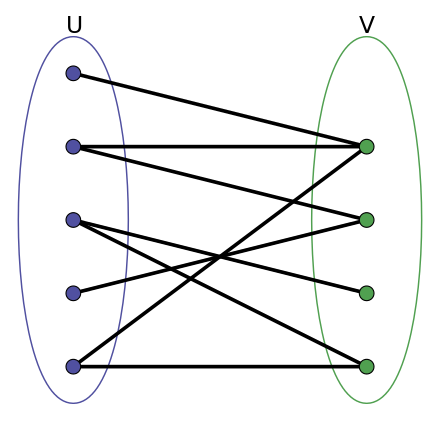
Step1 heapify the first 7 elements from the array, and make it a **MAX** heap O(k)

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(k) + (n-k)log(k) → 7 + n log(7)**

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 )

BFS

**图里常用的search 算法**

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

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

/ \

**3**   **2 -->**

/ \ /

4 7 -->

1 3 2 5 4 7 (BFS)

FIFO queue

Initialization: only one node is inserted in the queue, that is 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 4 and 7 and insert them into the queue, queue = [2, 4, 7]

step3, pop 2 out of the queue and **expand 2,** and generate …..

Termination condition: ??? queue is empty, done!!!

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 null

Solution:

1) if one node misses left child only (right child is there), → false

2)after detecting the first node that misses one right 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 size == 1

/ \

3  **2 size == 2**

/ \ /

5 4 size == 3

public void BFS(Node root) {

if (root == null){

return;

}

Queue<Node> queue= new LinkedList<Node>();

queue.offer(root);

while(!**queue**.isEmpty()){

**int size = queue.size();**

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

Node temp=queue.pop();

if(temp.left!=null){

queue.offer(temp.left);

}

if(temp.right!=null){

queue.offer(temp.right);

}

System.out.print(temp.value);

}

**System.out.println(); // change line**

}

}

**Discussion**

**常见考题**：

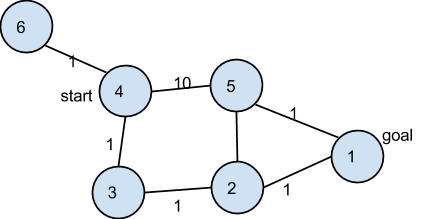
* 各种分层打印树的变体(zig-zag)
* **常见错误**： BFS 不是找图上两点最短路径的算法！！！

**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**
   1. **Initial state (start node)**
   2. **Node expansion/Generation rule:**
   3. **Termination condition: 所有点都计算完毕才停止,也就是 p\_queue 变空\**
5. **Example**
   1. start node is **4**
   2. cost(node) = cost(parent of node) + c(parent of node, node)

e.g. cost(2) = cost(3) + c(3, 2) = 1 + 1 = 2

* 1. **Cost[4, 5] = 10;**
  2. Cost for all the rest edges = 1;
  3. 
  4. **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. q\_queue = [5(10), 3(1), 6(1))]**

**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

1. **properties（s）**
   1. one node can be expanded once and only once
   2. one node can be generated more than once. (cost can be reduced over time)
   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**
   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))**
   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?

**12345**

**23456**

**34567**

**45678**

**56789**

**O(klog(k))**

assuming we are finding the 4-th smallest **value**

1 2 2 3 3 3 4 4 4 4 5 5 5 5

i

previous == 4

counter = 4

1. **initial state** [0][0]
2. **Expansion/Generation rule**: expand[0][0] → generate [0][1] and [1][0]
3. **Termination condition**: when the k-th element is popped out of the heap, then we are done. just return it.

**De-Duplication: 3**

**ttxxx**

**txxxx**

**xxxxx**

**xxxxx**

**xxxxx**

**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

**F(4)**

/ \

Line07 **F(3) +**   **F(2)**

/ \ / \

**F(2) =1** + **F(1)** F(1) + F(0) ⇒ 0 +1 = 1

/ \

**F(1) + F(0)**

**| |**

**1 0**

**in essence: traverse the whole tree (= whole search space)**

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

10 == root

/ \

**5**  . 15

/ \ / \

**2** 7 12 20 ←-all leaf node’s level == 3

/ \

null null

00 public void **printTreePreOrder**(Node root **== 10**) {

01 if (root == null ) {

02 return; **// base case**

03 }

04 System.out.println(root.value); **// what we do in current layer**

05 **printTreePreOrder(root.left); // try left node = state 1**

05b **printTreePreOrder(root.middle); // if we have 3 children nodes, then we should try middle node = state 1b**

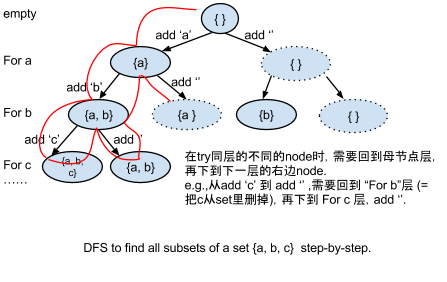
**06 printTreePreOrder(root.right); // try right node = state 2**

**07 }**

**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？**）**

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



**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)

00 public void **FindSubSet** (String **input**, int **index**, StringBuilder **solution**){

01 if (input.length == index) {

02 System.out.println(solution);

03 return;

04 }

05 // **Case 1**. Add string.at(index) to subset.

06 solution.**append**(input.charAt(Index));

**07 FindSubSet**(**input**, **index + 1**, solution); // breaking point

08 solution.**removeLast**(); **//recover the state before trying case 2**

09 // **Case 2**. Do not add string.at(index) to subset.

**10 FindSubSet**(input, index + 1, solution);

11 }

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

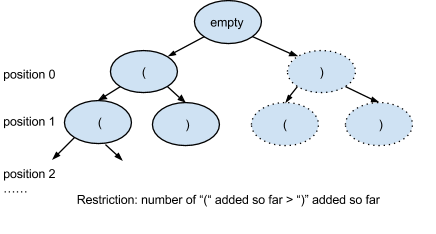
() () ()

)()()( whenever we input a right parenthesis, there must be a left parenthesis to match it (in front of it)

((( )))

**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DF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？**）**
3. how many levels???? 6 layers, xxx xxx
4. how many different states should we try???? **two** states we could try



// **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 **= 3**, int l, int r, StringBuilder **solu\_prefix**) {

if (l + r == 2 \* n) {

System.out.println(**solu\_prefix**); // base case;

return;

}

// try case 1: add left parenthesis “(”

if (l < n) {

solu\_prefix.append(“(”);

**DFS**(n, **l + 1**, r, solu\_prefix);

solu\_prefix.removeLast(“(”);

}

if (r < l) {

solu\_prefix.append(“)”);

DFS(n, l, r + 1, solu\_prefix);

solu\_prefix.removeLast(“)”);

}

}

Time complexity = **O(2^n)**

**经典例题3**  Print all valid combination of coins that can form a certain amount of money

99 cents

25 10 5 1 cent

3 x 25 cents

2 x 10 cents

0 x 5 cents

4 x 1 cents

**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DF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？**）**

**99** cents

/ | | \

level 1: 25 cents 0 x 25 **(99 remain)**  1 x 25 (74 rem) 2 x 25 (49 rem) 3 x 25 (24 rem)

/|| \ ???????\ /||||\

level 2: 10 cents 0x10,1x10….9x10 0 x 10, **1x10 (64 remain)** ... 7 x10

/||||\

level 3: 5 cents 0 x 5  **0x5 ,1x5...total 13 states under this path**

/||\

**level 4: 1 cents**

level 5 NULL

4 level

O (t**otal\_money/5**)^4

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

00 void **FindCombination**(int **money\_left**, int level, int sol[]) {

if (level == 3) {

sol[level] = monety\_left;

print;

}

// money value on this level == **coin[level]**;

for(int i = 0; **i <= money\_left / coin[level]**; ++i){

**sol[level] = i;**

**Findcombination**(money\_left - i \* coin[level], level+1, sol[]);

} // end for

}

**Hash Table**

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

**syntax: declare→ map <string, string> map1;**

key value

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

1. hash collision
   1. chaining
   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

Step 1: use hash\_table to count the occurrence of each word.

<key = word, value = counter>

Step 2a: sort the word according to its counter (assuming n words, )

O(nlog(n))

Step 2b: **MIN\_HEAP** of size k

iterate over the words from index k+1 to n

if the frequency of the word is > MIN\_HEAP.top(), then remove the top element from the MIN\_HEAP and insert the new word into the heap.

**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: hash\_table to count the number of occurrence of each number

M2: non hash\_table,

1 + 2 + 3 +4 …+n = n(n-1)/2 sum1

a[0] +a[1] +.... a[n-2] == sum 2

sum1 - sum2 == the missing number.

**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:** hash\_table, we insert all numbers in array a into the hash\_table,

then we iterate over all numbers in array b, if the element is in the hash\_table, then add it to the solution.

**M2**: 谁小移谁

a[] = 1 3 5 7 9

i

b[] = 2 3 5 6 7

j

3 5 7

O(n+m)

**M3**: N =3, M = 1 billion

binary search

O(nlog(M))

**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 words, 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.

# Class 7 Midterm 1 (数据结构自测)

**请大家各自打开自己的google doc （老师已经share给每位同学）**

**Question 1**. Reverse a singly LinkedList.

**Question 2**: print a binary tree layer by layer

**Question 3.** How to judge whether a binary tree is a binary search tree?

**Question 4.** Given a string **with no duplicated letters**, how to print out all permutations of the string

example: string input = “abcd”;

**DFS**

1. **how many levels and for each level, what does it store?**
2. **how many states can we try in each state on each level**

position 0 1 **2** 3

a b c d

a c d b

level 0 a (rem: bcd) b(rem: acd) c(rem: abd) d (rem: abc)

/ | \ / | \

level 1 b(cd) c(bd) d(bc) a c d ….

/ \

level 2 c d

| |

level 3 d c

// 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 0 1 2 3**

**input = “a b c d”**

a1 a2 c d

a1 a2 cd

a2 a1 cd

// index is the current layer number

09 void **permutation**(string& input, int **index == 1**) {

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.

15 for (int i = **index**; i < **input.size()**; i++) {

**root == a b c d**

i = 0, i = 1 , i = 2 , i = 3

level 0 **a** (bcd) **b**(acd) **c**(bad) **d** (bca)

/ | \ /|\

i=1,b i=2,c i=3(d) i=1 i=2 i =3

level 1 **b**(cd) **c**(bd) **d**(cb) x x x

16 swap(input, index, i);

17 **permutation**(input, index + 1);

18 swap(input, index, i);

19 }

20 }

# Class 8 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

10^2 10^1 10^0

1 0 1 0

→ least significant bit

2^2 2^1 2^0

binary representation of 10 is : **1 0 1 0**

**1\*2^3 + 0\*2^2 + 1\*2^1 + 0 \*2^0 = 8 + 0 + 2 + 0 = 10**

**32 bit machine**

int a = 10; // signed integern bc dfc dc

**0**000 0000 0000 0000 0000 0000 0000 1010

0: positive

1: negative

how do we represent a negative number, and what is the relationship between the positive representation

1 vs -1;

0000000000000000000000000 1

what if for “-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)

1111

1010

\_\_\_\_\_

1010

* 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 if (condition1 **||** condition2)

1111

1010

\_\_\_\_\_

1111

~ NOT (1->0 and 0->1 for each bit)

~ 1010

\_\_\_\_\_\_

0101

* ^ **XOR**
  + 00->0
  + 11->0
  + 01 or 10 ->1

0101

1100 ^

\_\_\_\_\_\_\_

1001

* << left shift (**右侧补充零)**
  + x = 0001
  + x << 1 → 001**0**
* >> 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 00000001 >>1 == 00000000000

If the variable ch contains the bit pattern 01100101, then **ch >> 1** will give the output as **0011100101**, and **ch >> 2** will give **00011100110**.

if we have a **negative** number , after right-shirt by 1 bit, then what do we get?

original number == **1**000000000000111111 >> >> >>

**111111111111111111111111** (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  
 Step 2: a = a << 4 (for short a<<=4)

a == 0000 0000 0000 0000 0000 0000 0001 0000

Step 3: x = x | a (for short **x |=a)**

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

**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

Step 3: x = x & a

设置某位为1就和0000010000或一下

设置某位为0就黑11111101111和一下

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

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

2^1 == 0000 0010

M1: count how many 1 digits in the binary representation of the number

00 bool isPowerOfTwo(int target) {

int a = 1;

int counter = 0;

while (target > 0) {

if (target & 1 == 1) {

counter ++;

}

target >>= 1;

}

return (counter == 1);

}

**M2:**

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

**what if x can be 0?**

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

**0000 1000 0000**

0000 0111 1111 &

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0000 0000 0000

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

0000 0101 0100

1010 1010 1010 XOR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 010 1111 1110

Step 1: take XOR

Step 2: count how many 1s in the result

int FindDiff(int a, int b) {

int temp = a ^ b;

int count = 0;

while (temp != 0) {

if (temp & 1 != 0) {

count++;

}

temp >>= 1;

}

return count;

}

for (A: **i = 0** (initialization); B:  **i < 1000**; C **i++** ) {

}

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 unsigned int a;**

01 a = b; // b’s value == -1, **1**111 1111 1111 1111 111...

02 cout << a; // what does a look like in the console?

// a == “a is a very large (largest 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.

2^64 - 1

**Question 4**: determine whether a word contains all unique letters.

e.g. student book abc → true

M1: hash\_table

boolean IsAllUnique(String input) {

Hashset<Character> sol = new Hashset<Character>();

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

if (sol.contains(input.charAt(i))) {

return false;

} else {

sol.add(input.charAt(i));

}

}

return ture;

M2: binary (26 letters)

**string1: s t u d e n t**

**int x;**

**hgfe dcba**

**0000 0000 0000 0000 0000 0000 0000 0000**

char == ‘h’

the bit to set is ‘h’ - ‘a’ = 7

we need to set the 7-th bit to be “1”

temp = 0000 x000 0000 // check whether x is == 1 or 0

public boolean IsAllUnique(String input) {

int temp = 0;

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

int temp2 = input.charAt(i)-’a’; // the 7-th bit

if (temp **&** (1 << temp2) > 0 ) {

return false;

}

temp |= (1 << temp2);

}

return **true**;

}

**Extension:**

what if we have a language that has 100 unique letters total

bit vector

int temp // we can represent 32 bits

How many integers can we use to represent 1000 bits?

100 / 32 + 1 == 4 integers

**int bitvec[4]**; // can represent 32 x 4 = 128 bit/letter

let’s assume we have a letter with its index to be 77;

‘a’’s index is still 0;

public boolean **IsAllUnique**(String input) {

**int bitvec[4]** = 0;

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

int temp2 = input.charAt(i)-’a’; // the 7-th bit

int index = temp2 / 32 ;

temp2 = temp2 % 32;

if (**bitvec[index]** **&** (1 << temp2) > 0) {

return false;

}

**bitvec[index]** |= (1 << temp2);

}

return **true**;

}

long

int

int32

int64

**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 **map[17]** = {‘0’, …..’F’};

10CD

// what if number == 0;

string convert(int **number**) {

if (number == 0) {

return “0x0”;

}

string result;

while (number > 0) {

int index = number % 16;

result = map[index] + result;

number = number / 16;

}

string = “0X” + string;

return string;

}

**Homework**:

**Questions 1 - 5**

# Class 9 Object Oriented Design (1)

I. Motivation

II. Basic Concepts

III. OOD in practice

IV. OOD questions in interview

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++, C#, Python, ...)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;

static {

// static field initialization

}

**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(); // bonusRate is 0.1

Employee.bonusRate = 0.2;

Employee e2 = new Employee("LiSi", 30, "222-22-2222", 6);

e2.printInfo(); // for e1 and e2, bonusRate has changed to 0.2

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**).

package test;

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;

}

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

public void printInfo();

}

public class Manager implements Employee {

private final String name;

private long id;

// ….

public Manager(String name) {

this.name = name;

//...

}

public int calculateSalary(double performanceScore) {

// return 100000;

}

public void printInfo() {

// even do nothing is ok

}

}

public static void main(String[] argv) {

// Employee e = new Employee(); **Wrong!**

**Employee e = new Manager(); // correct!**

}

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 calculateSalary(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 calculateSalary 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 class Programmer extends Employee {

// …

@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);

Employee e2 = new Manager(...);

e2.calculateSalary(0.85);

public void doSomthing(List<String> list) { // list → ArrayList/Vector/Stack/LinkedList/….

// list.add

// list.get

// list.remove

}

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

public class Employee {

// ….

public Employee() {

this.salary = calculateSalary(); // this is wrong!!

}

public int calculateSalary() {

//….

}

}

public class Manager extends Employee {

private List<..> historyPerformance;

public Manager() {

super(); // call Employee’s constructor

// do sth. else

// initialize historyPerformance

}

@Override

public int calculateSalary() {

// use historyPerformance to do the calculation

}

}

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

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. Classes, Functionality of each class, their relationships

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interface (public API): 别人怎么调用你的程序?

不要关心function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

public boolean canPark(Vehicle v) {

// TODO

}

}

class ParkingSpot {

boolean canPark(Vehicle) {

// check size

}

}

public interface Vehicle {

// private data field size; xxx

// size() method

}

car class, bus, … implements Vehicle

4. 定义类内部的其他方法和数据

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;

}

}

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 10 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 firstName, String lastName) {  
 this.firstName = firstName;

this.lastName = lastName;  
 }

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: 需要多个调用才能完成object creation, 并且无法确定一个User object什么时候构建完成

**使用Builder Pattern:**

public class User {

private **final** String firstName; // required, and immutable

private **final** String lastName; // required

private int age;

private String phone;

private String address;

**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; // this.age → UserBuilder object’s 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();**

// …..

User.UserBuilder b = **new User.UserBuilder(“San”, “Zhang”)  
 .age(25)  
 .phone(“1234567890”)  
 .address(“Fake address”);**

**User user = b.build();**

User.UserBuilder b = **new User.UserBuilder(“San”, “Zhang”);  
 b.age(25).phone(“1234567890”).address(“Fake address”);**

**// b.age(25); b.phone(“12345...”); b = b.address(“....”);**

**User user = b.build();**

**===============================================**

**User user = new User(firstName, lastName);**

**// user.setAge(25);**

**// do sth. to get the address and verify the address**

**------> use user to do sth.**

**user.setAddress(...);**

**// do some work to retrieve the phone number from a database**

**user.setPhone(....);**

**===============================================**

}

}

<http://stackoverflow.com/questions/1816458/getting-hold-of-the-outer-class-object-from-the-inner-class-object>

<http://stackoverflow.com/questions/70324/java-inner-class-and-static-nested-class>

public class User {

private String firstName;

private String lastName;

static class UserPrinter {

void print(User user) {

System.out.println(“This is user ” + user.**firstName** + “ ” + user.**lastName**);

}

}

class UserPrinter2 {

void print() {

System.out.println(“This is user ” + **firstName** + “ ” + **lastName**);

}

}

}

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

**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 room2 = makeRoom();

MagicMazeGame game2 = new MagicMazeGame();

game2.createGameWithTwoRooms(); // 2 magic rooms

**}**

}

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

// without factory method

public class MazeGame {

public void createGameWithTwoRooms() {

Room room1 = new OrdinaryRoom();

Room room2 = new OrdinaryRoom();

room1.connect(room2);

// we have more complicated process here….

this.addRoom(room1);

this.addRoom(room2);

}

}

public class MagicMazeGame extends MazeGame {

@Override

public void createGameWithTwoRooms() {

**Room room1 = new MagicRoom();**

**Room room2 = new MagicRoom(); // this is the only difference**

// this is the part we want to reuse

room1.connect(room2);

// we have more complicated process here….

this.addRoom(room1);

this.addRoom(room2);

}

}

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

**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

Example:

public interface GUIFactory {

public Button createButton();

}

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

class WinFactory implements GUIFactory {

public Button createButton() {

return new WindowsButton(); // suppose WindowsButton is a derived class of Button

}

}

class OSXFactory implements GUIFactory {

public Button createButton() {

return new MacButton(); // suppose MacButton is a derived class of Button

}

}

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 {

Application(GUIFactory factory) { // input: the GUIFactory factory used to create buttons

Button button = factory.**createButton**();

**button.paint();**

}

}

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() { // to make sure no one can call “new Singleton();”

// do something here...

}

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**

private 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);

}

public double getCost() { // Overriding methods defined in the abstract superclass

return super.getCost() + 0.5;

}

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());

}

}

# Class 11 Dynamic Programming (1)

Dynamic Programming is not a specific algorithm. It’s the name of a cluster of algorithms, which divide big problems into smaller problems to solve it efficiently.

Fibonacci:

Fn = Fn-1 + Fn-2

F1 = F2 = 1

given N, output Fn

public int f(int n) {

if(n == 1 || n == 2) {

return 1;

}

return f(n-1) + f(n-2);

}

f(1), f(2), …, f(n)

iteration:

public int f(int n) {

int a = new int[n+1]; 0..n-1

a[0] = 1;

a[1] = 1;

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

a[i] = a[i-1] + a[i-2];

}

int d = a[n]; <- out of range if array length is n

delete a;

return d;

}

O(1) space <- most efficient algorithm

a = 1; // F(n-2)

b = 1; // F(n-1)

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

c = a + b; F(n)

a = b;

b = c;

}

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.

basic solution:

start position, end position

enumerate all possible solution.

maxLength = 0;

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

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

// now check the subarray starting from i and ending at j

bool asc = true;

for (int k =i+1; k <=j; ++k) {

if (array[k] <= array[k-1]) { asc = false; break; }

}

if (asc) {

if (j-i+1 > maxLength) {

maxLength = j-i+1;

}

}

}

return maxLength;

}

S(i,j) (longest subarray from i to j)

i = 1, j = 3 k=1..3 satisfy a[k-1]<a[k]

i =1, j = 4, k=4

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

for (int j = i+1; j < array.length; ++j) {

if (array[j] > array[j-1]) {

// the subarray from i to j is a valid ascending array

maxLength = max(maxLength, j-i+1);

} else {

break;

}

}

}

return maxLength;

=====

i=0;

for (int j = 1; j < array.length; ++j) {

if (array[j] > array[j-1]) {

// the subarray from i to j is a valid ascending array

maxLength = max(maxLength, j-i+1);

} else {

i = j;

}

}

return maxLength;

[2,4,5,6,7,3,4,5,1]

i=0, j=1,2,3,4,5 [2,4,5,6,7] 3

i=5, j=6,7,8,9 [3,4,5] 1

i=9, break

O(n)

=======

public static int findlength(int[] array) {

int length = 1;

int maxLength = 0;

for(int i = 1; i <= array.length; i++) {

if(array[i] < array[i - 1]) {

length ++;

if (length > maxLength) { maxLength = length; }

} else {

length =1;

}

}

return maxLength;

}

**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 given by the problem**.

**——| —— —— —— —— —— —— ——**

**—— ——| —— —— —— —— —— ——**

**integer N = p0+p1+..pm-1, maximize p0\*p1\*..pm-1**

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

brute force:

recursion (Febnacci)

什么事大问题，什么是小问题:

MaxProduct(N,m)= 长度为N的rope切M刀

6切三刀的最优解 MaxProduct(6,3), 1..5

MaxProduct(6,3) = max {

1\*MaxProduct(6-1, 2),

2\*MaxProduct(6-2, 2),

3\*MaxProduct(6-3, 2),

4\*MaxProduct(6-4, 2),

5\*MaxProduct(6-5, 2)}

MaxProduct(5,2)= max {

1\*MaxProduct(5-1, 1),

2\*MaxProduct(5-2, 1),

3\*MaxProduct(5-3, 1),

4\*MaxProduct(5-4, 1)}

MaxProduct(N,m)= max{

1 \*MaxPRoduct(N-1,m-1),

2 \* MaxProduct (N-2,m-1),...

(N-1) \* MaxProduct (1,m-1),

}

MaxProduct(N,1)=N

**Recursion Solution:**

int maxProduct(int n, int m) {

if (m==1) {

return n;

}

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

int x = maxProduct(n-i,m-1)\*i;

if (x > maxP) {

maxP = x;

}

}

return maxP;

}

int solution = 0;

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

solution = max (solution, maxProduct(n,m));

}

**Iteration Solution:**

**S(N) = max{1\*S(n-1), 2\*S(n-2), … (n-1)\*S(1), N}**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S=1** | **S(2)** | **S(3)** | **S(4)** | **5** | **….11** |
| **1** | **max{1\*S(1),2}=2** | **max{1\*S(2),2\*S(1),3}=3** | **max{1\*S(3),2\*S(2),3\*S(1),4}=4** | **max{1\*S(4),2\*S(3),3\*S(2),4\*(S1)}=6** |  |
|  |  |  |  | **2** | **5** |

**11->5->2**

6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S(N,m) | 1 | 2 | 3 | 4 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 2 | 1\*1=1 | 0 | 0 |
| 3 | 3 | S(2,1)\*S(1,1)=2 | 1\*S(2,2) | 0 |
| 4 | 4 | max(1\*S(3,1),2\*S(2,1),3\*S(3,1)) | max(1\*S(3,2),2\*S(2,2)) |  |
| 5 | 5 | max(i\*S(5-i,1)) i=1..4 |  |  |

S[i][j]=0 (i,j=1..n)

S[1][j]=j (j=1..n)

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

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

S[i][j] = max(S[i-k][j-1]\*k, for k = 1..i-1)

}

}

return S[k][n]

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.

Define big problem, small problem

S(4) = (S(3) and A[3]>=1) or (S(2) and A[2]>=2) or (S(1) and A[1]>=3) (S(0) and A[0]>=4)

= ((S(2) and A[2]>=1) or (S(1) and A[1]>=2)) and A[3]>=1) or …...

S[k=0] = true

S[k=1]

S[k=2] = S[1], S[0]

S[k=3] = S[2],S[1],S[0]

S[k=N] = S[n-1], S[n-2]...S[0]

for (int k = 1; k <=N; ++k) {

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

S[k] = S[k-i] && A[k-i] >= i；

if (S[k]) break;

}

}

return S[N];

# Class 12 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%”
   1. www.yahoo.com?info=flower\_lsdkjfk
3. reverse (swap) e.g.  **I love yahoo** -> yahoo love I
4. substring → strstr
   1. regular method
   2. [Robin-Carp](http://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm) (hash based string matching) & [KMP](http://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm) (Knuth–Morris–Pratt)
5. other operations
   1. move letters around e.g. abc123 -> a1b2c3
   2. permutation (use DFS)
   3. concatenation

“apple”

Popular representations of characters:

**1.** [**ASCII**](http://www.asciitable.com/) representation of a letter: A == 65, a == 97

1 byte = 8 bits

For example, in c++

char x = ‘A’

char y = ‘C’

printf(“%d”, x) // print 65

printf(“%d”, y - x) // print 2

## 

2. [**Unicode**](http://en.wikipedia.org/wiki/Unicode) : the latest version of Unicode contains a repertoire of more than 110,000 [characters](http://en.wikipedia.org/wiki/Character_(computing)) covering 100[scripts](http://en.wikipedia.org/wiki/Script_(Unicode)) 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.

for (i = 0; i < aa.size(); i++) {

cout << aa[i];

}

0 … aa.size() - 1

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? 6

JAva

String a = “abc”;

a.length() // 3

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**) ; //s1 == “abc”

string a = “apple”;

string b(a); // b == “apple”

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()**;

1.5 in JAva, you can use toCharArray

String a = “abc”;

char[] b = a.toCharArray();

**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 b \_ a a b b b b b** \_ c c c c a ";  **-> “ab\_ca”**

i j

s[++i] = s[j++]

i++;

s[i] = s[j];

j++;

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;

}

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);

}

**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?

abc20%def20%gh

a b c d e f

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

string input;

// step 2

count = …; // from step 2

input.resize(count);

“xyz abc ”

input.size() == 9

// step 3

public static String transfer (String input) {

StringBuilder sb = new StringBuilder();

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

if(input.CharAt(i) == “ ”) {

sb.append(“2”);

sb.append(“0”);

sb.append(“%”);

} else {

sb.append(input.CharAt(i));

}

}

return sb;

}

z

**Question** 4.

(4.1)

apple elppa

i j ij

apple

eppla

i j

elppa

reverse( string& input)

(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

reverseWord(string& input) {

reverse(input); // oohay evol i

// find word one by one

for each word

reverse(word); // 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 e c d** e”; n

i

s2 = “**c d**”; m

j

**Method1 brute force**:

Step 1 define a helper function to determine whether two words are identical

bool checkIdentical(string s1, int index1 , string s2)

s1.substr(index1, index1 + s2.size()) == 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

worst case

aaaaaaaaaaaaa

aaaaaab

**Method2**:

[Robin-Carp](http://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm) (**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**

**hash(“cd”) => 45678**

**hash(“ab”) => 23432**

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)**

**a -> 0**

**b -> 1**

**c -> 2**

**d -> 3**

**….**

**z -> 25**

**b[cde]fgh**

hash(“b **c d** ”) －－> 1 \* 26^2 + 2\*26^1 + 3\*26^0

hash(“c **d e**”) －－> (hash(“bcd”) - 1 \* 26^2) \* 26 + 4 \* 26 ^0

**1. remove the leftmost item from the polynomial function // cd**

**2. all the rest items of (ab’s hashed value) x 26 // move cd to the left**

**3. add new item c // cde**

**O(n) because hash takes constant time**

**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 x → zx

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)

# Class 13 Object Oriented Design (习题课)

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

**1) Design a generic deck of cards**

**2) Design the blackjack game (focus on cards, deck ….)**

**3) Simulate the blackjack game**

**1) Design a generic deck of cards**

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

Assumption: standard 52-card set

Card, Deck, Hand

Card: 1) Value 2) Suit

Deck: 1) number of cards 2) Card[], List<Card> --- class composition

Hand: Card[]

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

Card

* getValue and getSuit

Deck

* shuffle
* getRemainingNum
* dealCards

Hand

* getScore
* getCardsNum
* addCards

class card {

enum{Heart, Spade, Diamond, Club} Suit;

private:

Suit suit;

enum{ A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K} value;

}

public enum Suit {

Club (0, “black”),

Diamond (1, “red”),

Heart (2, “red”),

Spade (3, “black”);

private final int value;

private final String color;

private Suit(int v, String c) {

value = v;

color = c;

}

public int getValue() {

return value;

}

public String getColor() {

return color;

}

public static Suit getSuitFromValue(int value) {

switch (value) {

case 0:

return Suit.Club;

case 1:

return Suit.Diamond;

case 2:

return Suit.Heart;

case 3:

return Suit.Spade;

default:

return null;

}

}

}

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

public **abstract** class Card {

/\* number or face that's on card - a number 2 through 10,

\* or 11 for Jack, 12 for Queen, 13 for King, or 1 for Ace

\*/

protected int faceValue;

protected Suit suit;

public Card(int c, Suit s) {

faceValue = c;

suit = s;

}

**public abstract int value();**

public Suit suit() {

return suit;

}

}

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

public class Deck<T extends Card> {

private static final Random random = new Random();

private ArrayList<T> cards = new ArrayList<T>();

private int dealtIndex = 0; // marks first undealt card

public void setDeckOfCards(ArrayList<T> deckOfCards) {

cards = deckOfCards;

}

public void **shuffle**() {

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

int j = random.nextInt(cards.size() - i - 1) + i;

T card1 = cards.get(i);

T card2 = cards.get(j);

cards.set(i, card2);

cards.set(j, card1);

}

}

private int remainingCards() {

return cards.size() - dealtIndex;

}

public Hand<T> **dealHand**(int number) {

if (remainingCards() < number) {

return null;

}

Hand<T> hand = new Hand<T>();

while (hand.size() < number) {

T card = dealCard();

if (card != null) {

hand.addCard(card);

}

}

return hand;

}

public T **dealCard**() {

if (remainingCards() == 0) {

return null;

}

T card = cards.get(dealtIndex);

dealtIndex++;

return card;

}

public void print() {

for (Card card : cards) {

card.print();

}

}

}

public class Hand<T extends Card> {

protected ArrayList<T> cards = new ArrayList<T>();

public int **score**() {

int score = 0;

for (T card : cards) {

score += card.value();

}

return score;

}

public void **addCard**(T card) {

cards.add(card);

}

public void print() {

for (Card card : cards) {

card.print();

}

}

public int size() {

return cards.size();

}

}

**2) How to extend the current design to support blackjack?**

BlackJackCard extends Card

* int value()

blackjack card rules:

* 1~10 scores its face value
* J, Q, and K (face card) scores 10
* A scores either 1 or 11

BlackJackHand extends Hand

* score: choose the best among all possible values
* isBlackJack: boolean

class BlackJackCard : Card {

public:

int \*getScores() { } // option 1

int getAScore() { } // option 2

}

class BlackJackHand : Hand {

public:

int \*getPosibleScores() { }

int getFinalScore() { } // check busted.

bool isBlackJack() { }

}

public class BlackJackCard **extends** Card {

public BlackJackCard(int c, Suit s) {

super(c, s);

}

@Override

public int **value**() {

if (faceValue >= 11 && faceValue <= 13) { // Face card

return 10;

} else { // Number card or Ace

return faceValue;

}

}

public int **minValue**() {

if (isAce()) { // Ace

return 1;

} else {

return value();

}

}

public int **maxValue**() {

if (isAce()) { // Ace

return 11;

} else {

return value();

}

}

public boolean isAce() {

return faceValue == 1;

}

public boolean isFaceCard() {

return faceValue >= 11 && faceValue <= 13;

}

}

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

public class BlackJackHand extends Hand<BlackJackCard> {

@Override

public int **score**() {

ArrayList<Integer> scores = possibleScores();

int maxUnder = Integer.MIN\_VALUE;

int minOver = Integer.MAX\_VALUE;

for (int score : scores) {

if (score > 21 && score < minOver) {

minOver = score;

} else if (score <= 21 && score > maxUnder) {

maxUnder = score;

}

}

return maxUnder == Integer.MIN\_VALUE ? minOver : maxUnder;

}

**private** ArrayList<Integer> possibleScores() {

ArrayList<Integer> scores = new ArrayList<Integer>();

if (cards.size() == 0) {

return scores;

}

for (BlackJackCard card : cards) {

addCardToScoreList(card, scores);

}

return scores;

}

**private** void addCardToScoreList(BlackJackCard card, ArrayList<Integer> scores) {

if (scores.size() == 0) {

scores.add(0);

}

int length = scores.size();

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

int score = scores.get(i);

scores.set(i, score + card.minValue());

if (card.minValue() != card.maxValue()) {

scores.add(score + card.maxValue());

}

}

}

public boolean busted() {

return score() > 21;

}

public boolean is21() {

return score() == 21;

}

public boolean **isBlackJack**() {

if (cards.size() != 2) {

return false;

}

BlackJackCard first = cards.get(0);

BlackJackCard second = cards.get(1);

return (first.isAce() && second.isFaceCard())

|| (second.isAce() && first.isFaceCard());

}

}

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

**3) How to simulate the game, i.e., design a blackjack game automator?**

棋牌游戏类OOD

* 对游戏道具和游戏状态的描述
* 游戏规则和游戏流程

game status s1 ---> rules + actions ---> s2 → check end → continue…..

1. Understand the rules

2. Capture game status

3. Model game procedure

How to describe the current game status

* Hand[]
* Deck

Hand h1, Hand h2, Shuffled Deck

{} {} {c1, c2, …, c52}

deal cards

{c1, c2} {c3, c4} {c5, c6, …, c52}

apply rules: if there is/are blackjack

action: h1/h2 decides if continue, and if so call “addCard”

h1 h2 Deck

{c1, c2, c5} {c3, c4, c6} {c7, c8, ...}

apply rule: check if h1/h2 busted

action…..

…..

apply rules to check/compare scores

public class BlackJackGameAutomator {

private Deck<BlackJackCard> deck;

private BlackJackHand[] hands;

private static final int HIT\_UNTIL = 16;

public BlackJackGameAutomator(int numPlayers) {

hands = new BlackJackHand[numPlayers];

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

hands[i] = new BlackJackHand();

}

}

public boolean dealInitial() {

for (BlackJackHand hand : hands) {

BlackJackCard card1 = deck.dealCard();

BlackJackCard card2 = deck.dealCard();

if (card1 == null || card2 == null) {

return false;

}

hand.addCard(card1);

hand.addCard(card2);

}

return true;

}

public ArrayList<Integer> getBlackJacks() {

ArrayList<Integer> winners = new ArrayList<Integer>();

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

if (hands[i].isBlackJack()) {

winners.add(i);

}

}

return winners;

}

public boolean playHand(int i) {

BlackJackHand hand = hands[i];

return playHand(hand);

}

public boolean playHand(BlackJackHand hand) {

while (hand.score() < HIT\_UNTIL) {

BlackJackCard card = deck.dealCard();

if (card == null) {

return false;

}

hand.addCard(card);

}

return true;

}

public boolean playAllHands() {

for (BlackJackHand hand : hands) {

if (!playHand(hand)) {

return false;

}

}

return true;

}

public ArrayList<Integer> getWinners() {

ArrayList<Integer> winners = new ArrayList<Integer>();

int winningScore = 0;

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

BlackJackHand hand = hands[i];

if (!hand.busted()) {

if (hand.score() > winningScore) {

winningScore = hand.score();

winners.clear();

winners.add(i);

} else if (hand.score() == winningScore) {

++++ winners.add(i);

}

}

}

return winners;

}

public void initializeDeck() {

ArrayList<BlackJackCard> cards = new ArrayList<BlackJackCard>();

for (int i = 1; i <= 13; i++) {

for (int j = 0; j <= 3; j++) {

Suit suit = Suit.getSuitFromValue(j);

BlackJackCard card = new BlackJackCard(i, suit);

cards.add(card);

}

}

deck = new Deck<BlackJackCard>();

deck.setDeckOfCards(cards);

deck.shuffle();

}

public void printHandsAndScore() {

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

System.out.print("Hand " + i + " (" + hands[i].score() + "): ");

hands[i].print();

System.out.println("");

}

}

public static void main(String[] args) {

int numHands = 5;

BlackJackGameAutomator automator = new BlackJackGameAutomator(numHands);

automator.initializeDeck();

boolean success = automator.dealInitial();

if (!success) {

System.out.println("Error. Out of cards.");

} else {

System.out.println("-- Initial --");

automator.printHandsAndScore();

ArrayList<Integer> blackjacks = automator.getBlackJacks();

if (blackjacks.size() > 0) {

System.out.print("Blackjack at ");

for (int i : blackjacks) {

System.out.print(i + ", ");

}

System.out.println("");

} else {

success = automator.playAllHands();

if (!success) {

System.out.println("Error. Out of cards.");

} else {

System.out.println("\n-- Completed Game --");

automator.printHandsAndScore();

ArrayList<Integer> winners = automator.getWinners();

if (winners.size() > 0) {

System.out.print("Winners: ");

for (int i : winners) {

System.out.print(i + ", ");

}

System.out.println("");

} else {

System.out.println("Draw. All players have busted.");

}

}

}

}

}

}

**II. Design an in-memory file system.**

Main functionalities:

1. files, directories, A directory contains files/sub-directories
2. metadata of files/directories: name, creation time, access time, modification time, etc.
3. A FileSystem can then be modeled as a tree consisting of files/directories
   1. Leaf nodes: files or empty directories
   2. Q: how to count the total number of files/directories ?

/

/foo /foo2 /bar

/foo/baz ….

class Entry {

String name;

long creationTime;

long accessTime;

Directory parent;

}

class file {

long size;

}

class directory{

List<Entry> children;

}

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

public abstract class Entry {

protected Directory parent;

protected long created;

protected long lastUpdated;

protected long lastAccessed;

protected String name;

public Entry(String n, Directory p) {

name = n;

parent = p;

created = System.currentTimeMillis();

}

public boolean delete() {

if (parent == null) {

return false;

}

return parent.deleteEntry(this);

}

public abstract int size();

public String getFullPath() {

if (parent == null) {

return name;

} else {

return parent.getFullPath() + "/" + name;

}

}

public long getCreationTime() {

return created;

}

public long getLastUpdatedTime() {

return lastUpdated;

}

public long getLastAccessedTime() {

return lastAccessed;

}

public void changeName(String n) {

name = n;

}

public String getName() {

return name;

}

}

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

public class File extends Entry {

private String content;

private int size;

public File(String n, Directory p, int sz) {

super(n, p);

size = sz;

}

public int size() {

return size;

}

public String getContents() {

return content;

}

public void setContents(String c) {

content = c;

}

}

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

public class Directory extends Entry {

protected ArrayList<Entry> contents;

public Directory(String n, Directory p) {

super(n, p);

contents = new ArrayList<Entry>();

}

protected ArrayList<Entry> getContents() {

return contents;

}

public int size() {

int size = 0;

for (Entry e : contents) {

size += e.size();

}

return size;

}

public int **numberOfFiles**() {

int count = 0;

for (Entry e : contents) {

if (e instanceof Directory) {

count++; // Directory counts as a file

Directory d = (Directory) e;

count += d.numberOfFiles();

} else if (e instanceof File) {

count++;

}

}

return count;

}

public boolean deleteEntry(Entry entry) {

return contents.remove(entry);

}

public void addEntry(Entry entry) {

contents.add(entry);

}

}

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

public class FileSystem {

private final Directory root;

public FileSystem() {

root = new Directory(“/”, null);

}

public List<Entry> resolve(String path) {

// TODO: write program to resolve path like “/foo/bar/baz”

}

public void mkdir(String path) {

// TODO: create a new directory with the given path

}

public void createFile(String path) {

// TODO: create a new file with the given path

}

public void delete(String path) {

// TODO: delete the file/directory with the given path

}

public Entry[] list(String path) {

// TODO: list all the immediate children of the directory specified by the given path

return null;

}

public int count() {

// TODO: return the total number of files/directories in the FileSystem

}

}

**III. Design a chat server**

Functionalities:

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

User, User status, User management

Message, Conversation

Private Chat, Group Chat

Q: how to model a user’s basic information

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

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

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

Q: how to manage all the users?

* UserManager
  + id-user mapping
  + current online users

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

https://github.com/gaylemcd/ctci/tree/master/java/Chapter%208

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

# Class 14 Dynamic Programming (2)

**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?

1. how to reduce big problem into small problems. a.k.a define recursive formula
2. how to iteratively calculate solutions from small problems to big problem.

S(n-1) - minimimal number of jumps from 0 to n-1, using array A[0..n-1]

S(n) = min{ S(0)+1 if A[0]>=n and S(0) is set before// if the opimal solution is from 0 jump to n directly

S(1)+1 if A[1]>=n-1 // ……………………….is from 1 jump to n directly

....

S(n-1)+1 if A[n-1]>=1 }

what if A[0]<n, A[1]<n-1, .. A[n-1]<1? S(n)=?

//using -1 is not a good idea

//S(3) = -1 A[3]=1

//S(4) = S(3)+1=0

S(0) = 0

**Recursion Solution**

int **minimal\_jump**(int n, vector<int>& a) { // n is the length to jump to reach the target

if (n==0) {

return 0;

}

int minjump = n+1;

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

if (a[i] >= n-i) {

minjump = min(**minimal\_jump(i, a)**+1, minjump);

}

}

return minjump;

}

index = 0 1 2 3 4 5 6 n-1

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

m2:

int **min\_jump**(vector<int>& a, int **cur\_index = 0**) {

if (cur\_index >= n-1) { // base case

return 0;

}

int jum\_num = INT\_MAX;

for (int i = 1; i <= a[cur\_index]; i++) {

int temp = **min\_jump**(a, cur\_index + i) + 1;

if (temp < jum\_num) {

jum\_num = temp;

}

}

return jum\_num;

}

O(2^n)

][n, n-1, n-2, …,2, 1

[n, n-1, n-2, … 3, 1]

[n, n-1, n-2, …..4,1]

C(n,n-1)

+

C(n,n-2)

[n-1, n-2, ..1]

2^n -1 = C(n,1)+C(n,2)+...+C(n,n-1)+C(n,n)

int answer = minimal\_jump(n, a);

if (answer == n+1) {

printf(“no answer”);

} else {

...

}

**DP Solution:**

**public int jump(int[] array){**

**if(array.length < 0 ) {**

**return -1;**

**}**

**int[] S = new int[array.length];**

**S[0] = 0;**

**for (int i =1; i< array.length; i++){**

**// calculate S[i] = min { S[j]+1, if A[j]>=i-j, j=0..i-1}**

**S[i] = array.length+1;**

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

**if (A[j]>=i-j) {**

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

**}**

**}**

**}**

**return S[array.length-1]; // array.length+1 if no solution**

**}**

**Greedy Algorithm:**

int left = 0;

int right = 0;

int rst = 0;

while (left <= right) {

if (right >= num.length - 1) {

return rst;

}

int newRight = right;

for (int i = left; i <= right; i++) {

newRight = Math.max(newRight,num[i]+i);

}

left = right+1;

right = newRight;

rst++;

}

return -1;

**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

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

inHashtable(W) = true if W is in the dictionary, false if not O(1) ( O(n) )

S(n) - string 0..n-1 can be composed by words in the dictionary

S(i) - strng 0..i-1 can be composed by words in the dictionary

S(n) = S(i) && inHashtable(substring(i,n-i) ) , i=0..n-1

for i=0 it is S(0) && inHashtable(substring(0,n-1)), which is checking entire string in dictionary or not. therefore S(0) must be true.

(next page)

00public boolean **wordSolver**(String word, HashSet<String> dict) {

01

02 if (word.length() == 0) {

03 return true;

04 } else {

05 for (int i = 0; i < word.length()-1; i++) {

// Check if substring(i,n-i) in dictionary and substring(0,i+1) can be solved recursively

06 if(**dict.contains**(word.subString(i, n-i)) &&

07 **wordSolver**(word.subString(0, i), dict)){

08 return true;

09 }

10 }

11 }

12 return false;  
13}

O(2^n)

**bob**

**cat**

**rob**

Word: bcoabt

Solution: False

Word: **bobcatrob** || catbob

Solution: True

starting form small problem to big problem

**bobcatrob**

S(1) = b <- S(0) && b is in dictionary = false = false

S(2) = bo = S(0) && bo in dictionary = false, or S(1) && o in dictionary = false = false

S(3) = bob = S(0) && bob in dictionary = true, or S(1) && ob, or S(2) && b = true

S(4) = bobc = S(0) && bobc in dictionary = false, S(1) && obc, S(2) && bc, S(3) && c = false

..S(6) = bobcat = S(0) && bobcat = false, S(1) && obcat, S(2) && bcat, S(3) && cat = True, S(4) && at, S(5)&t = True

..

S(9) = …. S(6) && rob = True

String::substring(int start, int length) {}

// not String::substring(int start, int end) {}

S[0] = True;

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

// 求解 S(i), word[0..i-1] can be composed by dictionary

S[i] = false; // 假设S(i) 无解

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

// 利用 S(j) 求解 S(i)

if (S[j] && dict.contains(word.substring(j,i-j))) {

S[i] = true; break;

}

}

}

return S[n-1];

O(n\*n) = O(n^2) or O(n^3)

**Q3. Edit Distance**

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

**Example:**

s1 = “asdf”

s2 = “sghj”

s1 == c1 | s1r ← rest of s1

s2 == c2 | s2r ← rest of s2

Example:

s1= a | sdf

s2= s | ghj

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

**s** sdf

s ghj

editDistance(sdf, ghj) + 1

**(2) Delete:**

\_sdf

sghj

editDistance(sdf, sghj) + 1

**(3) Insert:**

s asdf

s ghj

editDistance(asdf, ghj) + 1

Solution = min{ (1), (2), (3) }

**Recursive Solution:**

int editDistance(String s1, String s2) {

if (s1.empty()) { return s2.length(); }

if (s2.empty()) { return s1.length(); }

return min(

editDistance(s1[1..n1-1],s2[1..n2-1]) +1,

editDistance(s1[1..n1-1],s2[0..n-2-1])+1

editDistance(s1[0..n1-1],s2[1..n2-1])+1

);

}

**S[i][j]** represents the subsolution of s1[0….i-1]

s2[0...j-1] ⇒ the number of actions needed for converting s1[0..i-1] to s2[0..j-1]

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

ind 0 1 2 3 4

**s1** 0 **0 1 2 3 4 x is the base case.**

**a** 1 **1** **1** **2** 3 4 x= min(from left + 1, from topleft+1, from top+1)

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

**d** 3 **3** 2 2 3 4

**f** 4 **4** 3 3 3 4

s[1][1] = min { S[0][0]+1 // replacement, s[1][0] + 1 // deletion, s[0][1] + 1 // insertion }

a -> sg

s[1][2] = min { s[0][1] (“”->s) + 1(a->g, replacement),

s[0][2] (“” ->sg) + 1 (a->””, delete a),

s[1][1] (a->s) + 1 (insert g) }

00public 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 }

**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

S(n,m) = 以n,m为右下角的所有正方形当中的最大值

largest square of 1’s, max { S(i,j) , i =0..n-1, j=0..m-1}

S(n,m) =

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 **1** **1** S(4,1) … S(4,5)

1 1 1 **1** 1 S(5,5) = 5?

S(5,1)..S(5,4)

S(5,5) <= S(4,4)+1

S(5,5) <= S(5,4)+1

S(5,5) <= S(4,5)+1

S(5,5) = min { S(4,4)+1, S(5,4)+1, S(4,5)+1 } if matrix[5][5]=1

= 0 if maxtrix[5][5]=0

S(n,m) = min { S(n-1,m)+1, S(n-1,m-1)+1, S(n,m-1)+1} if matrix[n][m]=1,

= 0 if matrix[n][m] = 0

S[0][i] = S[i][0] = 1 (if matrix[0][i]/[i][0] = 1)

**Matrix**

**0 0 0 0 0**

**1 1 1 1 0**

1 1 1 1 0

1 1 1 0 0

1 1 1 0 0

S[i][j]= // 以i,j为右下角的最大全1正方形

**0 0 0 0 0**

**1 1 1 1 0** **?** = min([0][0], [0][1], [1][0]) + 1;

1 2 2 2 0 **x** = min([2][0], [1][1], [1][0]) + 1 == 2;

1 2 3 0 0

1 2 **3** **0** 0

S[1][1] = min(S[0][1], S[1][0],S[0][0]) + 1 = 1

S[1][2] = S[1][3] = 1

S[1][4] = 0 because matrix[1][4] = 0

S[2][1] = min(S[1][0],S[1][1],S[2][0]) + 1 = 2

**DP Solution:**

s[0][i] = m[0][i] (i=0..m-1)

s[i][0] = m[i][0] (i=0..n-1)

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

for (int j = 1; j < m; ++j) {

if (m[i][j] == 1) {

s[i][j] = min(s[i-1][j],s[i][j-1],s[i-1][j-1]) + 1;

} else {

s[i][j] = 0;

}

}

}

int maxa = 0;

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

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

maxa = max(maxa, s[i][j]);

}

}

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 15 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);

}

}

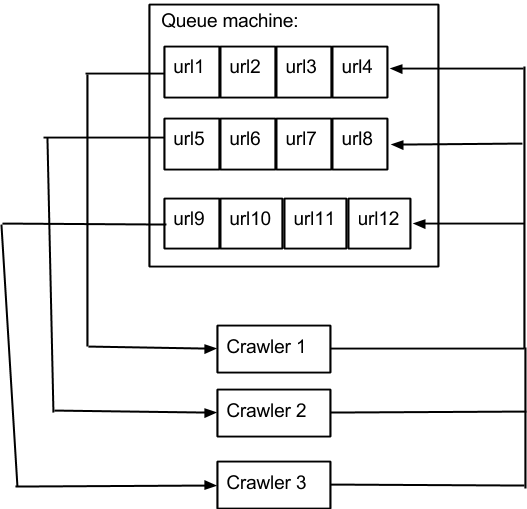
Bottleneck:

- 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:

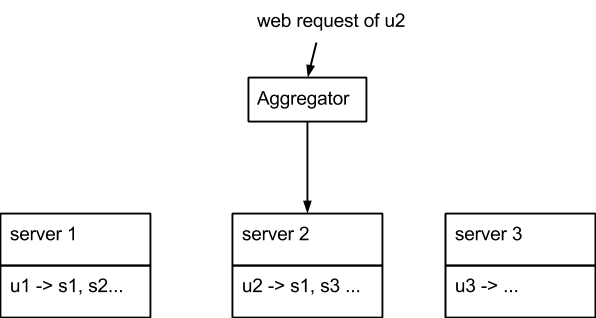
s1 -> info for s1

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

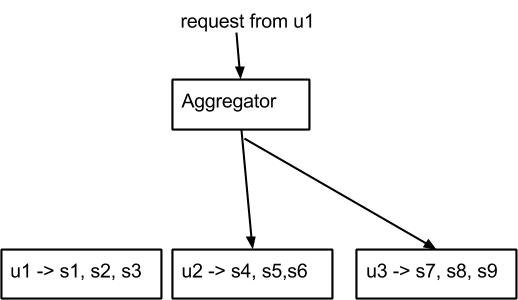
s2 -> info for s2

s3 -> info for s3

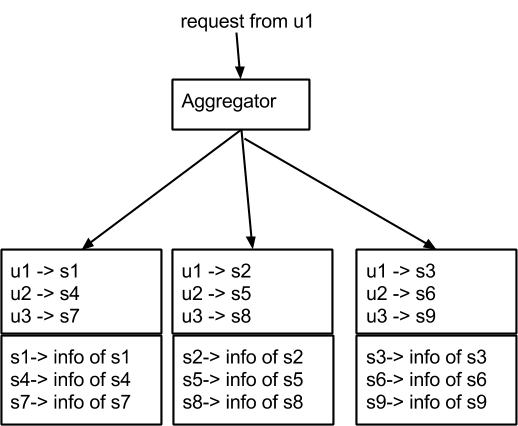
s4 -> info for s4

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

leaf 1 : (s4, score4), (s7, score7)

leaf 2: (s8, score8), (s5, score5)

leaf 3: (s9, score9), (s6, score6)

=> aggregator, merge sort

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:

* uid as key is duplicated

- more complicated to implement and understand

pros:

- no duplicate information of stories

- 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

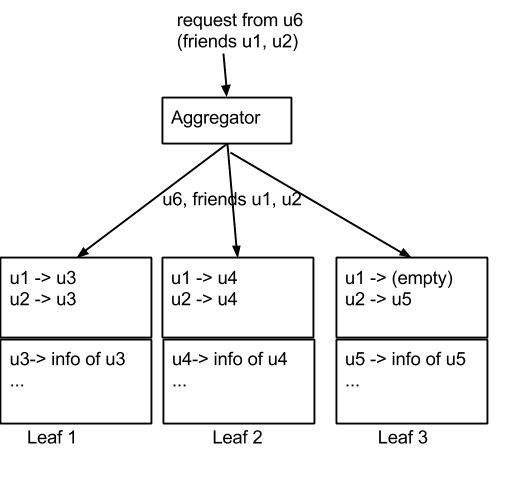
u2 -> u3, u4, u5

…

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.

u1 -> u3, u4

u2 -> u3, u4, u5

U6->U1,U2

U7->u3

U8->u1,u2,u3

u1, u3 -> go to leaf1

u2, u4 -> go to leaf2

u5, u7 => goto leaf 3

leaf 1:

u1 -> u3

u2 -> u3

U6 -> u1

u7 -> u3

u8 -> u1

leaf 2

u1 -> u4

u2 -> u4

u6 -> u2

u8 -> u2

leaf 3

u1 -> u7

u2 -> u5

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.