The solutions to the book "Introduction to Algorithm, 3rd Edition"

Jian Li
Computer Science
Nanjing University, China

2011

I Foundations

Chapter 1

Problem 1-1 Comparison of running times

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\log n$	2^{10^6}	$2^{6\cdot10^7}$	$2^{36\cdot10^8}$	$2^{864\cdot10^8}$	$2^{2592\cdot10^9}$	$2^{94608\cdot10^{10}}$	$2^{94608\cdot10^{12}}$
$\sqrt{n}$	10^{12}	$36\cdot10^{14}$	$1296\cdot10^{16}$	$746496\cdot10^{16}$	$6718464\cdot10^{18}$	$8950673664\cdot10^{20}$	$8950673664\cdot10^{24}$
n	10^6	$6\cdot10^7$	$36\cdot10^8$	$864\cdot10^8$	$2592\cdot10^9$	$94608\cdot10^{10}$	$94608\cdot10^{12}$
$n\log n$	62746	2801417	$13\cdot10^7$	$27\cdot10^8$	$71\cdot10^9$	$80\cdot10^{10}$	$69\cdot10^{12}$
n^2	10^3	24494897	$6\cdot10^4$	293938	1609968	30758413	307584134
n^3	10^2	391	1532	4420	13736	98169	455661
2^n	19	25	31	36	41	49	56
$n!$	9	11	12	13	15	17	18

Chapter 2

Problem 2-1 Insertion sort on small arrays in merge sort

a. 对于 k 个元素插入排序的最坏时间为 $\Theta(k^2)$, 所以排序 n/k 个 sublists 的总最坏时间为 $\Theta(k^2 n/k) = \Theta(nk)$

b. 如果用 Merge Sort 原始的合并方法,即每次从 n/k 个 sublists 中找到最小的然后 Copy 到 return 的数组中,所需要的最坏时间为 $\Theta(n(n/k)) = \Theta(n^2/k)$
如果想在 $\Theta(n \log(n/k))$ 的时间内完成合并,我们可以对 n/k 个 sublists 进行一对一对的合并,直到最后合成一个表,所需要的时间即为 $\Theta(n \log(n/k))$

c. 当 $k = \Theta(\log n)$ 时,有 $\Theta(nk + n \log(n/k)) = \Theta(nk + n \log n - n \log k) = \Theta(2n \log n - n \log \log n) = \Theta(n \log n)$,即和 Standard Merge Sort 的渐进运行时间相同

d. 实际中,k 是最大的 list 长度,使得 Insertion Sort 比 Merge Sort 快

Problem 2-2 Correctness of bubblesort

a. 我们需要证明 A' 是 A 的一个排列

b. **Loop invariant:**
对于每次开始迭代时,$A[j] = \min\{A[k] : j \le k \le n\}$,并且,显然 $A[j..n]$ 是初始时 $A[j..n]$ 的一个排列
Initialization:
初始时,$j = n$,$A[j..n]$ 仅有一个元素 $A[n]$,循环不变式显然成立
Maintenance:
根据循环不变式,$A[j]$ 是 $A[j..n]$ 的最小元素;如果 $A[j-1] > A[j]$,那么通过交换 $A[j]$ 和 $A[j-1]$,使得 $A[j-1]$ 是 $A[j-1..n]$ 的最小值;有于在迭代开始之前,$A[j..n]$ 是初始时 $A[j..n]$ 的一个排列,而在迭代中仅仅是交换了 $A[j-1]$ 和 $A[j]$,所以 $A[j-1..n]$ 必然也是初始时 $A[j-1..n]$ 的一个排列;进入下一次迭代之前,j 的值减小为 $j-1$
Termination:
当 $j = i$ 时,循环结束;$A[i] = \min\{A[k] : i \le k \le n\}$,并且 $A[i..n]$ 是初始时 $A[i..n]$ 的一个排列

c. **Loop invariant:**
对于每次开始迭代时,$A[1..i-1]$ 包含初始时 $A[1..n]$ 的最小的 $i-1$ 个元素,并且 $A[1] \le A[2] \le .. \le A[i-1]$,而 $A[i..n]$ 包含初始时 A 中剩余的 $n-i+1$ 个元素

Initialization:
初始时,$i=1$,$A[1..i-1]$ 是空集,循环不变式显然成立
Maintenance:
根据循环不变式,$A[1..i-1]$ 包含初始时 $A[1..n]$ 的最小的 $i-1$ 个元素,并且 $A[1] \le A[2] \le .. \le A[i-1]$;而执行 lines 2-4 使得 $A[i]$ 是 $A[i..n]$ 中的最小的一个,所以现在 $A[1..i]$ 是初始时 $A[1..i]$ 中元素的非递减排列;$A[i+1..n]$ 包含初始时 A 中剩余的 $n-i$ 个元素
Termination:
当 $i=n$ 时,循环结束;$A[1..i-1]$ 即 $A[1..n-1]$ 是初始时 $A[1..n-1]$ 的非递减排列,且包含 A 中最小的 $n-1$ 个元素;显然,$A[n]$ 是 A 中最大的元素,所以 $A[1..n]$ 是初始 A 排序后的序列

d. Bubble Sort 最坏运行时间为 $\Theta(n^2)$,和 Insertion Sort 的运行时间一样

Problem 2-3 Correctness of Horner′s rule

a. $\Theta(n)$

b. 朴素算法的运行时间为 $\Theta(n^2)$

Naive-Polynomial-Evaluation$(P(x), x)$

```
1  y = 0
2  for i = 0 to n
3       t = 1
4       for j = 1 to i
5            t = t · x
6       y = y + t · a_i
7  return y
```

c.

$$y = \sum_{k=0}^{n} a_k x^k$$

(Omit!)

d. (Omit!)

Problem 2-4 Inversions

a. $(1,5),(2,5),(3,5),(4,5),(3,4)$

b. $\{n, n-1, n-2, .., 2, 1\}$
总共 $\binom{n}{2} = n(n-1)/2$ 对 inversions

c. Insertion-Sort 每交换一对元素,就消除一对原有序列中的 inversion,所以所以 Insertion-Sort 的运行时间和 inversions 的对数是同数量级的

d. N 个元素组成的序列,最坏运行时间为 $\Theta(n \log n)$,找出序列 inversions 的对数

Count-Inversions$(A, left, right)$

```
1  inversions = 0
2  if left < right
3      mid = ⌊(left + right)/2⌋
4      inversions = inversions + Count-Inversions(A, left, mid)
5      inversions = inversions + Count-Inversions(A, mid + 1, right)
6      inversions = inversions + Merge-Inversions(A, left, mid, right)
7  return inversions
```

Merge-Inversions$(A, left, mid, right)$

```
 1  n_1 = mid − left + 1
 2  n_2 = right − mid
 3  Let L[1 .. n_1 + 1] and R[1 .. n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[left + i − 1]
 6  for i = 1 to n_2
 7      R[i] = A[mid + i]
 8  L[n_1 + 1] = R[n_2 + 1] = ∞
 9  i = j = 1
10  inversions = 0
11  counted = False
12  for k = left to right
13      if counted = False and L[i] > R[j]
14          inversions = inversions + n_1 − i + 1
15          counted = True
16      if L[i] ≤ R[j]
17          A[k] = L[i]
18          i = i + 1
19      else A[k] = R[j]
20          j = j + 1
21          counted = False
22  return inversions
```

Answer = Count-Inversions(A, 1, n)
算法的最坏运行时间与 Merge-Sort 相同, 即 $\Theta(n \log n)$

Chapter 3

Problem 3-1 Asymptotic behavior of polynomials

(Omit!)

Problem 3-2 Relative asymptotic growths

A	B	$\mathcal{O}$	o	Ω	ω	Θ
$\log^k n$	n^ϵ	Yes	Yes	No	No	No
n^k	c^n	Yes	Yes	No	No	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No	No	No
2^n	$2^{n/2}$	No	No	Yes	Yes	No
$n^{\log c}$	$c^{\log n}$	Yes	No	Yes	No	Yes
$\log n!$	$\log n^n$	Yes	No	Yes	No	Yes

Problem 3-3 Ordering by asymptotic growth rates

a. $f(n)$ 增长率级别如下表:

$2^{2^{n+1}}$	2^{2^n}	$(n+1)!$	$n!$	e^n	$n \cdot 2^n$
2^n	$(3/2)^n$	$n^{\log\log n} = (\log n)^{\log n}$		$(\log n)!$	n^3
$4^{\log n} = n^2$		$n \log n$ and $\log(n!)$		$2^{\log n} = n$	
$(\sqrt{2})^{\log n}$	$2\sqrt{2\log n}$	$\log^2 n$	$\ln n$	$\sqrt{\log n}$	$\ln\ln n$
$2^{\log^* n}$	$\log^* n$ and $\log^*(\log n)$		$\log(\log^* n)$	$n^{1/\log n}$ and 1	

b.

$$f(n) = 2^{2^{(n+1)\cdot\cos n}}$$

Problem 3-4 Asymptotic notation properties

a. Wrong! 显然,$n = \mathcal{O}(n^2)$,但 $n^2 \neq \mathcal{O}(n)$

b. Wrong! 令 $f(n) = n^2, g(n) = n$,则 $\Theta(\min(f(n), g(n))) = \Theta(n)$,显然 $f(n) + g(n) = n^2 + n \neq \Theta(n)$

c. Right! 因为 $f(n) = \mathcal{O}(g(n))$ 且 $\log(g(n)) \geq 1, f(n) \geq 1$ when $n \to \infty$, 即 $f(n) \leq cg(n) \Rightarrow \log(f(n)) \leq \log(cg(n)) = \log c + \log(g(n))$, 令 $b = \log c + 1$, 则 $\log c + \log(g(n)) \leq b \log(g(n))$, 即 $\log(f(n)) = \mathcal{O}(\log(g(n)))$

d. Wrong! 令 $f(n) = 2n, g(n) = n$, 我们有 $f(n) = \mathcal{O}(g(n))$ 但 $2^{f(n)} = 2^{2n} \neq \mathcal{O}(2^{g(n)} = 2^n)$

e. Wrong! 如果 $f(n) < 1$

f. Right! 因为对于正实数 $c > 0, f(n) \leq cg(n) \Rightarrow 1/c \cdot f(n) \leq g(n)$

g. Wrong! 令 $f(n) = 2^n$, 则 $f(n) = 2^n \neq \Theta(f(n/2) = 2^{\sqrt{n}})$

h. Right! $f(n) + o(f(n)) = \Theta(\max(f(n), o(f(n)))) = \Theta(f(n))$

Problem 3-5 Variations on $\mathcal{O}$ and Ω

a.

$$f(n) = \begin{cases} \mathcal{O}(g(n)) \ \& \ \overset{\infty}{\Omega}(g(n)) & \text{if } f(n) = \Theta(g(n)) \\ \mathcal{O}(g(n)) & \text{if } 0 \leq f(n) \leq cg(n) \\ \overset{\infty}{\Omega}(g(n)) & \text{if } 0 \leq cg(n) \leq f(n), \text{ for infinitely many integers } n \end{cases}$$

如果只有有限多个 n 使得 $f(n) \geq cg(n) \geq 0$, 那么当 n 趋进 ∞ 时, 必有 $0 \leq f(n) \leq cg(n)$, 即 $f(n) = \mathcal{O}(g(n))$
显然, 用 Ω 替换 $\overset{\infty}{\Omega}$ 时不成立

b. Advantage: 可以刻画所有函数之间的关系
Disadvantage: 刻画得不精确

c. For any two functions $f(n)$ and $g(n)$, we have if $f(n) = \Theta(g(n))$ then $f(n) = \mathcal{O}'(g(n))$ and $f(n) = \Omega(g(n))$
But the converse is not true

d. $\widetilde{\Omega}(g(n)) = \{f(n) :$ there exist positive constants $c, k,$ and n_0 such that $0 \leq cg(n)\log^k(n) \leq f(n)$ for all $n \geq n_0\}$
$\widetilde{\Theta}(g(n)) = \{f(n) :$ there exist positive constants $c_1, c_2, k_1, k_2,$ and n_0 such that $0 \leq c_1 g(n)\log^{k_1}(n) \leq f(n) \leq c_2 g(n)\log^{k_2}(n)$ for all $n \geq n_0\}$
For any two functions $f(n)$ and $g(n)$, we have $f(n) = \widetilde{\Theta}(g(n))$ if and only if $f(n) = \widetilde{\mathcal{O}}(g(n))$ and $f(n) = \widetilde{\Omega}(g(n))$

Problem 3-6 Iterated functions

$f(n)$	c	$f_c^*(n)$
$n-1$	0	n
$\log n$	1	$\log^*(n)$
$n/2$	1	$\lceil \log n \rceil$
$n/2$	2	$\lceil \log n \rceil - 1$
$\sqrt{n}$	2	$\geq \log \log n$
$\sqrt{n}$	1	∞
$n^{1/3}$	2	$\geq \frac{\log \log n}{\log 3}$
$n/\log n$	2	$\mathcal{O}(\log n)$

Chapter 4

Problem 4-1 Recurrence examples

a. $T(n) = \Theta(n^4)$

b. $T(n) = \Theta(n)$

c. $T(n) = \Theta(n^2 \log n)$

d. $T(n) = \Theta(n^2)$

e. $T(n) = \Theta(n^{\log_2 7})$

f. $T(n) = \Theta(\sqrt{n} \log n)$

g. $T(n) = \Theta(n^3)$

Problem 4-2 Parameter-passing costs

a.
1. $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\log n)$
2. $T(n) = T(n/2) + \Theta(N) \Rightarrow T(n) = \Theta(N \log n)$
3. $T(n) = T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n)$

b.
1. $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$
2. $T(n) = 2T(n/2) + \Theta(N) \Rightarrow T(n) = \Theta(nN)$
3. $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$

Problem 4-3 More recurrence examples

a. $T(n) = \Theta(n^{\log_3 4})$

b. $T(n) = \Theta(n \log \log n)$

c. $T(n) = \Theta(n^2 \sqrt{n})$

d. $T(n) = \Theta(n \log n)$

e. $T(n) = \Theta(n \log \log n)$

f. $T(n) = \Theta(n)$

g. $T(n) = \Theta(\log n)$

h. $T(n) = \Theta(n \log n)$

i. $T(n) = \Theta(n/\log n)$

j. $T(n) = \Theta(n \log\log n)$

Problem 4-4 Fibonacci numbers

a. (Omit!)

b. (Omit!)

c. 无穷等比数列展开可得到

$$F(z) = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i)z^i$$

d. 因为 $|\hat{\phi}| < 1$,所以当 $i \to \infty$ 时,有 $\hat{\phi}^i \to 0$,所以 $F_i = \phi^i/\sqrt{5}$ for $i > 0$ 最接近于 Fibonacci 数

Problem 4-5 Chip testing

Note: 当我们找到一个好的芯片的时候,所有的芯片的好坏就可以全部知道

a. 我们假设有一个芯片是好的,那么因为坏的芯片数量超过一半,所以教授在测试这个芯片是否是好的时候,根据题意,所有的坏芯片可以联合起来共同欺骗教授,即说这是坏的,那么教授就会认为这是哥坏的芯片;所以当坏的芯片超过半数时,教授无法判断哪个芯片是好的

b. 可以对 n 个芯片进行成对测试,总共 $\lfloor n/2 \rfloor$ 对;我们将测试结果分成两组,一组是测试结果为 AB 同好坏的,我们称为 P 组;另一组是测试结果至少有一个是坏的,我们称为 Q 组;由于总体上 good > bad,且 Q 组必然 good $\leq$ bad,所以 P 组必是 good > bad,所以,为了找到一个好的芯片,我们可以舍弃 Q 组的所有芯片,而 P 组的芯片我们可以从每一对中任选出一个,作为下一次测试的对象,这样问题规模至少减少为原来的 $\frac{1}{2}$

c. $T(n) = T(n/2) + n/2 \Rightarrow T(n) = \Theta(n)$

Problem 4-6 Monge arrays

a. 由定义可证

b. $A[2,3] = 5$

c. 若存在 $f(i) > f(i+1)$,即存在 $A[i,j] + A[i+1,k] > A[i,f(i)] + A[i+1,f(i{+}1)]$,所以不满足 Monge array 的定义,所以不存在 $f(i) > f(i{+}1)$,即 $f(1) \leq f(2) \leq .. \leq f(m)$

d. 由 (c) 易得该时间为 $\mathcal{O}(m+n)$ 算法

e. $T(m) = T(m/2) + \mathcal{O}(m+n) \Rightarrow T(m) = \mathcal{O}(m + n \log m)$

Chapter 5

Problem 5-1 Probabilistic counting

a. $X_i(1 \le i \le n)$ 表示在第 i 次增加操作时计数器增加的值,V_i 表示在 i 次增加操作后计数器表示的值

$V_n = \sum_{i=1}^{n} X_i$

$E[V_n] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$

而 $E[X_i] = 0 \cdot (1 - \frac{1}{n_{i+1}-n_i} + (n_{i+1} - n_i) \cdot \frac{1}{n_{i+1}-n_i}) = 1$,即 $E[V_n] = n$

b. X_i 两两独立,所以 $Var[V_n] = Var[X_1] + Var[X_2] + .. + Var[X_n]$

$Var[X_i] = E[X_i^2] - E^2[X_i] = ((0^2 \cdot \frac{99}{100}) + (100^2 \cdot \frac{1}{100})) - 1^2 = 99$

所以 $Var[V_n] = 99n$

Problem 5-2 Searching an unsorted array

a. 算法如下:

RANDOM-SEARCH(A, x)

```
1  B = ∅
2  while B ≠ A
3       i = RANDOM(1, n)
4       if A[i] = x
5            return i
6       B = B ∪ A[i]
7  return Null
```

b. 期望在第 n 次找到 x

c. 期望在第 n/k 次找到 x

d. 期望在 $\Theta(n \log n)$ 次找到 A 中的所有元素

e. Average-Case: $\frac{n+1}{2}$, Worst-Case: n

f. Average-Case:

$$\sum_{i=1}^{n-k+1} \frac{i\binom{n-i}{k-1}}{\binom{n}{k}} = \frac{\binom{n+1}{k+1}}{\binom{n}{k}} = \frac{n+1}{k+1}$$

Worst-Case: $n-k+1$

g. Average-Case:n , Worst-Case:n

h. $K=0$: Worst-Case: n, Expected: n

$K=1$: Worst-Case: n, Expected: $\frac{n+1}{2}$

For $K \geq 1$, Worst-Case: $n-k+1$, Expected: $\frac{n+1}{k+1}$

i. Scramble-Search

II Sorting and Order Statistics

Chapter 6

Problem 6-1 Building a heap using insertion

a. No, counterexample: $1, 2, 3$

b. 很明显 Worst-Case 的上界为 $\mathcal{O}(n \log n)$, 所以我们主要来证明下界

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} \Theta(\lfloor \log i \rfloor) \\ &\geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \Theta(\lfloor \log i \rfloor) \\ &\geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \Theta(\lfloor \log(\lceil \frac{n}{2} \rceil) \rfloor) \\ &= \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \Theta(\lfloor \log n - 1 \rfloor) = \Omega(n \log n) \end{aligned}$$

所以, Worst-Case 的界为 $\Theta(n \log n)$

Problem 6-2 Analysis of d-ary heaps

a. index i, j-th child $(1 \leq j \leq d)$

D-ary-Parent(i)
1 **return** $\lfloor (i-2)/d + 1 \rfloor$

D-ary-Child(i, j)
1 **return** $d(i-1) + j + 1$

b. $\Theta(\log_d n)$

c. 和 Heap-Extract-Max 类似, 只是在 Max-Heapify 中, 由每次的比较 2 个元素找最大值, 变成比较 d 个元素找最大值, 所以运行时间为 $\mathcal{O}(d \log_d n)$

d. 和 Max-Heap-Insert 类似, 运行时间为 $\mathcal{O}(\log_d n)$

e. 和 Heap-Increase-Key 类似, 运行时间为 $\mathcal{O}(\log_d n)$

Problem 6-3 Young tableaus

a.

2	3	4	5
8	9	∞	∞
12	14	∞	∞
16	∞	∞	∞

b. (Omit!)

c. 类似用 Max-Heapify 的方法,用 $\mathcal{O}(m+n)$ 来维护 Young Tableau 的结构

d. 用 $\mathcal{O}(m+n)$ 的时间来找到一个不存在的元素,即 ∞;将元素插入到这个位置,然后用 $\mathcal{O}(m+n)$ 的时间来维护 Young Tableau 的结构

e. Young-Tableau-Insert:$\mathcal{O}(n^3)$
Young-Tableau-Extract-Min:$\mathcal{O}(n^3)$
Total:$\mathcal{O}(n^3)$

f. 注意 $m \times n$Young Tableau 的左下角的元素,以此元素为根,会形成一个类似 Binary Search Tree 的结构,算法和 BST 相同,所以运行时间即结构的高度 $\mathcal{O}(m+n)$

Chapter 7

Problem 7-1 Hoare partition correctness

a. (Omit!)

b. 因为当 Hoare-Partition 执行时,$p \le i < j \le r$ 恒成立,所以 i, j 不会访问 $A[p..r]$ 以外的元素

c. 显然,当 $i \ge j$ 时,Hoare-Partition 结束,所以 $p \le j < r$

d. 当 Hoare-Partition 结束时,有 $A[p..j] \le x \le A[j+1..r]$

e. the QUICKSORT procedure to use HOARE-PARTITION

```
Quicksort'(A, p, r)
1  if p < r
2      q = Hoare-Partition(A, p, r)
3      Quicksort'(A, p, q)
4      Quicksort'(A, q + 1, r)
```

Problem 7-2 Quicksort with equal element values

a. $\mathcal{O}(n \log n)$

b. two methods:

Method 1:

```
Partition'(A, p, r)
 1  x = A[p]
 2  i = p − 1
 3  j = r + 1
 4  while True
 5      repeat
 6          j = j − 1
 7      until A[j] < x
 8      repeat
 9          i = i + 1
10      until A[i] > x
11      if i < j
12          exchange A[i] with A[j]
13      else return j, i
```

Method 2:

Partition'(A, p, r)

```
1  i = j = Partition(A, p, r)
2  while i > 1 and A[i − 1] = A[i]
3        i = i − 1
4  while j < r and A[j + 1] = A[j]
5        j = j + 1
6  return i, j
```

take $\Theta(r-p)$ time

c. (Omit!)

d. 因为 Quicksort' 的比较次数一定少于 Quicksort,运行时间仍然为 $\mathcal{O}(n \log n)$

Problem 7-3 Alternative quicksort analysis

a. $E[X_i] = 1/n$

b.

$$E[T(n)] = E[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))]$$

c.

$$E[T(n)] = \frac{2}{n}\sum_{q=2}^{n-1} E[T(q)] + \Theta(n)$$

d.

$$\begin{aligned}
\sum_{k=2}^{n-1} k\log k &\le \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k\log k + \sum_{k=\lceil \frac{n}{2} \rceil}^{n} k\log k \\
&\le \frac{(\frac{n}{2})^2}{2}\log\frac{n}{2} + \frac{(\frac{n}{2}+n)\frac{n}{2}}{2}\log n \\
&= \frac{n^2}{8}\log n - \frac{n^2}{8} + \frac{3}{8}n^2\log n \\
&= \frac{1}{2}n^2\log n - \frac{n^2}{8}
\end{aligned}$$

e. $E[T(n)] = \Theta(n \log n)$

Problem 7-4 Stack depth for quicksort

a. (Omit!)

b. 在 Worst-Case 的情况下,每次 $q = r$,所以对于 n 个元素的序列,Stack 的深度最多为 $\Theta(n)$

c. The worst-case stack depth is $\Theta(\log n)$, and the expected running time is $\mathcal{O}(n \log n)$

```
Tail-Recursive-Quicksort'(A, p, r)
1  while p < r
2      // Partition and sort the small subarray first
3      q = Partition(A, p, r)
4      if q − p < r − q
5          Tail-Recursive-Quicksort'(A, p, q − 1)
6          p = q + 1
7      else Tail-Recursive-Quicksort'(A, q + 1, r)
8          r = q − 1
```

Problem 7-5 Median-of-3 partition

a. $P_i = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$

b. 概率从原来的 $\frac{1}{n}$ 增长为

$$\lim_{n\to\infty} P_{\lfloor (n+1)/2 \rfloor} = \frac{3}{2} \cdot \frac{1}{n}$$

c.

$$\sum_{i=\frac{n}{3}}^{\frac{2n}{3}} P_i \approx \int_{\frac{n}{3}}^{\frac{2n}{3}} P_i = \frac{13}{27}$$

d. 每次 Partition 只增加 $\mathcal{O}(1)$ 的操作,所以总的复杂度依然是 $\Theta(n \log n)$, Median-of-3 method 只影响常数时间

Problem 7-6 Fuzzy sorting of intervals

a. A randomized algorithm for fuzzy-sorting n intervals

Find-Intersection(A, B, p, s, a, b)

```
 1  i = Random(p, s)
 2  exchange A[i] with A[s]
 3  exchange B[i] with B[s]
 4  a = A[s]
 5  b = B[s]
 6  for i = p to s − 1
 7       if A[i] ≤ b and B[i] ≥ a
 8            if A[i] > a
 9                 a = A[i]
10            elseif B[i] < b
11                 b = B[i]
```

Partition-Right(A, B, a, p, s)

```
1  i = p − 1
2  for j = p to s − 1
3       if A[j] ≤ a
4            i = i + 1
5            exchange A[i] with A[j]
6            exchange B[i] with B[j]
7  exchange A[i + 1] with A[s]
8  exchange B[i + 1] with B[s]
9  return i + 1
```

Partition-Left-Middle(A, B, b, p, r)

```
1  i = p − 1
2  for j = p to r − 1
3       if B[j] < b
4            i = i + 1
5            exchange A[i] with A[j]
6            exchange B[i] with B[j]
7  exchange A[i + 1] with A[r]
8  exchange B[i + 1] with B[r]
9  return i + 1
```

Fuzzy-Sort(A, B, p, s)

```
1  if p < s
2      a = b = 0
3      Find-Intersection(A, B, p, s, a, b)
4      r = Partition-Right(A, B, a, p, s)
5      q = Partition-Left-Middle(A, B, b, p, r)
6      Fuzzy-Sort(A, B, p, q − 1)
7      Fuzzy-Sort(A, B, r + 1, s)
```

b. The algorithm runs in expected time $\Theta(n \log n)$ in general, but runs in expected time $\Theta(n)$ when all of the intervals overlap. Since when all of the intervals overlap, the procedure Fuzzy-Sort just calls the procedure Find-Intersection only once and then return. Therefore, the algorithm runs in expected time $\Theta(n)$.

Chapter 8

Problem 8-1 Probabilistic lower bounds on comparison sorting

a. reachable leaves: each of the $n!$ possible permutations is the input with the probability $1/n!$
unreachable leaves: the probability is 0

b. Let $d_T(x) =$ depth of node x in tree T

$$\begin{aligned} D(T) &= \sum_{x \in leaves(T)} d_T(x) \\ &= \sum_{x \in leaves(LT)} d_T(x) + \sum_{x \in leaves(RT)} d_T(x) \\ &= \sum_{x \in leaves(LT)} (d_{LT}(x) + 1) + \sum_{x \in leaves(RT)} (d_{RT}(x) + 1) \\ &= \sum_{x \in leaves(LT)} d_{LT}(x) + \sum_{x \in leaves(RT)} d_{RT}(x) + \sum_{x \in leaves(T)} 1 \\ &= D(LT) + D(RT) + k \end{aligned}$$

c. A decision tree T with k leaves that the left tree has i leaves and the right tree has $k - i$ leaves.

$$d(k) = \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$$

d. Using derivative, it's easy to have the function $i \log i + (k-i) \log(k-i)$ is minimized at $i = k/2$
When $i = k/2$, using Master Theorem, we have $d(k) = \Omega(k \log k)$

e. Let $k = n!$, then $D(T_A) \ge d(k) = \Omega(k \log k)$, therefore $D(T_A) = \Omega(n! \log n!)$
Since the $n!$ permutations have equal probability $1/n!$, the expected time to sort n random elements is

$$\frac{\Omega(n! \log(n!))}{n!} = \Omega(\log n!) = \Omega(n \log n)$$

Problem 8-2 Sorting in place in linear time

a. Counting-Sort

b. Quicksort-Partition

c. Insertion-Sort

d. (a) Yes (b) No (c) No

e. use $\mathcal{O}(k)$ outside the input array

Counting-Sort(A, k)

```
1   // let C[0..k] be a new array
2   for i = 0 to k
3       C[i] = 0
4   for i = 1 to A.length
5       C[A[i]] = C[A[i]] + 1
6   // C[i] now contains the number of elements equal to i
7   p = 0
8   for i = 0 to k
9       for j = 1 to C[i]
10          p = p + 1
11          A[p] = i
```

Not stable, in place, in $\mathcal{O}(n + k)$

f. Extension: Is there exist an algorithm s.t. stable, in place, runs in $\mathcal{O}(n)$ time ?

Problem 8-3 Sorting variable-length items

a. Let m_i be the number of integers with i digits, for $i = 1, 2, \ldots, n$, we have $\sum_{i=1}^{n} i \cdot m_i = n$

* Counting every number's digits.....$\mathcal{O}(n)$
* Sort the integers by number of digits(Counting-Sort).....$\mathcal{O}(n)$
* Use Radix-Sort to sort each group of integers with the same length.....$\mathcal{O}(n)$

Therefore, the total running time is $\mathcal{O}(n)$

b.

* Use Counting-Sort to sort the strings by the first letter
* If the first letter of some strings are same, then put these strings into a group as well as remove the first letter
* Recursion sort until each group has only one string

The running time is $\mathcal{O}(n)$

Problem 8-4 Water jugs

a. n 个 Red jug 和 n 个 Blue jug 之间最多比较 $\Theta(n^2)$ 次

b. 用 Decision Tree 的模型, 对于树中的每个节点, 表示一个 Red jug 和 Blue jug 的比较; 从每个节点会引出三条边, 分别代表 Red jug 大、Blue jug 大、一样大, 树中的叶节点即表示一个确定的结果
对于 n 个 Red jug 和 n 个 Blue jug 之间最多有 $n!$ 种组合方式, 所以 Decision Tree 的高度为 $h = \log(n!) = \Omega(n \log n)$

c. the expected running time is $\mathcal{O}(n \log n)$, and the worst-case running time is $\mathcal{O}(n^2)$

Partition-Red(R, p, r, x)

```
1  i = p − 1
2  for j = p to r
3       if R[j] ≤ x
4            i = i + 1
5            exchange R[i] with R[j]
6  return i
```

Partition-Blue(B, p, r, x)

```
1  i = p − 1
2  for j = p to r
3       if B[j] ≤ x
4            i = i + 1
5            exchange B[i] with B[j]
```

Match-Jugs(R, B, p, r)

```
1  if p < r
2       k = Random(p, r)
3       q = Partition-Red(R, p, r, B[k])
4       Partition-Blue(B, p, r, A[q])
5       Match-Jugs(R, B, p, q − 1)
6       Match-Jugs(R, B, q + 1, r)
```

Problem 8-5 Average sorting

a. Ordinary Sorting

b. 2, 1, 4, 3, 6, 5, 8, 7, 10, 9

c. (Omit!)

d. Shell-Sort, i.e. We split the n-element array into k part. For each part, we use Insertion-Sort(or Quicksort) to sort in $\mathcal{O}(n/k \log(n/k))$ time. Therefore, the total running time is $k \cdot \mathcal{O}(n/k \log(n/k)) = \mathcal{O}(n \log(n/k))$

e. Using a heap, we can sort a k-sorted array of length n in $\mathcal{O}(n \log k)$ time. (The height of the heap is $\log k$, the solution to Exercise 6.5-9)

f. The lower bound of sorting each part is $\Omega(n/k \log(n/k))$, so the total lower bound is $\Omega(n \log(n/k))$. Since k is a constant, therefore $\Omega(n \log(n/k)) = \Omega(n \log n)$

Problem 8-6 Lower bound on merging sorted lists

a. $\binom{2n}{n}$

b. 建立一个长度为 2 的比较队列,每个 list 中的元素至少进入队列一次,出队列一次,所以至少 $2n - o(n)$ 次比较

c. 利用上述比较队列的模型,易知比较队列中的元素一定来自不同的队列,所以如果两个在已排好序列的连续的元素一定在在比较队列中出现过,所以它们之间比较过

d. 每个 list 中的元素至少进入队列一次,出队列一次,所以下界是 $2n - 1$

Problem 8-7 The 0-1 sorting lemma and columnsort

a. $A[p]$ 和 $A[q]$ 一定都在错误的位置,且由定义有 $A[p] < A[q]$,所以 $B[p] = 0, B[q] = 1$

b. 由于 $A[p]$ 是最小的错误位置元素,则 $q < p$,即 $(p > q, A[p] < A[q]) \Rightarrow (p > q, B[p] = 0 < B[q] = 1)$;又因为 Alogrithm X 是一个 oblivious compare-exchange algorithm,即 Algorithm X 排序 A 的操作指令和排序 B 时是一样的,所以 Algorithm X 不能正确的排序 B 序列

c. 不管用什么算法来排序,排序后的结果都和一个 oblivious compare-exchange algorithm 执行之后的结果相同,所以我们可以认为 column-sort 是一个 oblivious compare-exchange algorithm

d. 因为完成 step 1 的时候,所有的 1 都在底部,所以在完成 step 2, 3 之后,右边的列的 1 的个数一定大于等于左边的列的 1 的个数,而 dirty row 一定是由一些 0 和一些 1 组成;即 1 都连续的排在右边,0 都连续的排在左边,所以至多 (<) 有 s rows 是 dirty rows

e. 必然是从 clean area of 0s 开始,必然是在 clean area of 1s 结束;因为在 step 3 之后,至多有 s rows 是 dirty rows,所以中间的 dirty area 至多包含 s^2 个元素

f. step 5 的目的是保证每一列自身有序,step 6 的目的是将整个 array 首尾连接
因为 $r/2 \geq s^2$,又因为 dirty area 有至多 s^2 个元素,所以在 step 5 之后如果还未排好序,则一定是如下所示的这种情况

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{matrix}$$

即存在两个相邻的列,左边的列是以 1 结尾,而右边的列以 0 开头;所以在 step 7, 8 之后,所有的 array 中元素都按 column-major order 排好

g. 如果 s 不能整除 r,即对于 every column 在做变换的时候,至多产生 2 dirty rows,即中间的 0、1 过度区域产生一个 dirty row,最后一行产生一个 dirty row;在 step 3 再排序之后,dirty cows 的个数等于最多 1 的列和最少 1 的列相差 1 的个数,而对于 1 column 变换,即至多产生 2 dirty rows,这 2 rows 中 column 之间的个数至多差 1 个,所以 step 3 之后 dirty cows 至多为 s
$r \geq 2s^2$

h. 如果 $r \geq 2s^2$ 不成立,则可以通过填充无意义的元素 (example. ∞) 来使得 $r \geq 2s^2$ 成立

Chapter 9

Problem 9-1 Largest i numbers in sorted order

a. $\Theta(n \log n)$

b. $\Theta(n + i \log n)$

c. $\Theta(n + i \log i)$

Problem 9-2 Weighted median

a. Since all the element has same weight $w_i = 1/n$, therefore the median of $x_1, x_2, .., x_n$ is the weighted median of the x_i for $i = 1, 2, .., n$

b.
* Sorting the n element into increasing order by x_i values.....$\mathcal{O}(n \log n)$
* Locate the weighted median of n elements.....$\mathcal{O}(n)$

Therefore, the total running time is $\mathcal{O}(n \log n)$

c. a linear-time median algorithm

Weighted-Median(X, p, r)

```
1  q = Randomized-Partition(X, p, r)
2  WL = sum_{i=p}^{q-1} X[i]
3  WR = sum_{i=q+1}^{r} X[i]
4  if WL < 1/2 and WR ≤ 1/2
5       return q
6  if WL ≥ 1/2
7       w[q − 1] = w[q − 1] + WR
8       return Weighted-Median(X, p, q − 1)
9  if WR > 1/2
10      w[q + 1] = w[q + 1] + WL
11      return Weighted-Meidan(X, q + 1, r)
```

Analyse:
The recurrence for the worst-case running time of Weighted-Median is $T(n) = T(n/2 + 1) + \Theta(n)$. The solution of the recurrence is $T(n) = \Theta(n)$

d. Let p be the weighted median. For any point x, let $f(x) = \sum_{i=1}^{n} w_i \, |x - p_i|$. We want to find a point x such that $f(x)$ is minimum.

Assume $f(p)$ is minimum, we will prove that for any other point x, $f(p) \leq f(x)$. For any p and x such that $x > p$, we have

$$f(x) - f(p) = \sum_{i=1}^{n} w_i(|x - p_i| - |p - p_i|)$$

* When $p_i \leq p < x$, we have $|x - p_i| - |p - p_i| = x - p$
* When $p < p_i \leq x$, we have $|x - p_i| - |p - p_i| \geq p - x$
* When $p < x \leq p_i$, we have $|x - p_i| - |p - p_i| = p - x$

Therefore,

$$\begin{aligned} f(x) - f(p) &= \sum_{i=1}^{n} w_i(|x - p_i| - |p - p_i|) \\ &\geq \sum_{p<p_i} w_i(p - x) + \sum_{p\geq p_i} w_i(x - p) \\ &= (x - p)(\sum_{p\geq p_i} w_i - \sum_{p<p_i} w_i) \\ &\geq 0 \end{aligned}$$

For $p > x$, we have the same result. Thus, $f(x) \geq f(p)$, i.e. the weighted median is a best solution for the 1-dimensional post-office location problem.

e. Find a point $p(x, y)$ to minimize the sum

$$f(x, y) = \sum_{i=1}^{n} w_i(|x - x_i| + |y - y_i|)$$

Since $d(a, b) = |x_a - x_b| + |x_b - y_b|$, it is obvious that

$$\min_{x,y} f(x, y) = \min_{x} g(x) + \min_{y} h(y)$$

Therefore, the best solution for the 2-dimensional post-office location problem reduce to the best solution for the 1-dimensional post-office location problem that x and y are independent.

Problem 9-3 Small order statistics

a. **1.** divide all the elements into two group:

Group 1: a_1 a_3 a_5 $\cdots$
Group 2: a_2 a_4 a_6 $\cdots$

2. pairwise compare the elements of the two group, and put the bigger of the two elements into a new group, called bigger group, and put the others into another group, called smaller group ..... $\lfloor \frac{n}{2} \rfloor$
3. recursively find the i-th smallest element in the smaller group ..... $U_i(\lceil \frac{n}{2} \rceil)$
4. meanwhile, we can get the smallest i numbers in the smaller group and their corresponding elements in the bigger group
5. the i-th smallest elements must be in these $2i$ elements, then find it ..... $T(2i)$

Therefore, we have

$$U_i(n) = \begin{cases} T(n) & \text{if } i \geq n/2 \\ \lfloor \frac{n}{2} \rfloor + U_i(\lfloor \frac{n}{2} \rfloor) + T(2i) & \text{otherwise} \end{cases}$$

b. the depth of recursion is $k = \mathcal{O}(log\frac{n}{i})$ therefore, if $i < n/2$, then $U_i(n) = n + \mathcal{O}(T(2i) \log \frac{n}{i})$

c. since i is a constant less than $n/2$, $U_i(n) = n + \mathcal{O}(\log n)$

d. If $k > 2$, then $U_i(n) = n + O(T(2n/k) \log k)$
If $k = 2$, i.e. $\log k = 1$, then

$$\begin{aligned} U_i(n) &= T(n) \\ &= n + (T(n) - n) \\ &\leq n + (T(2i) - n) \\ &= n + (T(2n/k) - n) \\ &= n + (T(2n/k) \log k - n) \\ &= n + \mathcal{O}(T(2n/k) \log k) \end{aligned}$$

Problem 9-4 Alternative analysis of randomized selection

a.

$$E[X_{ijk}] = \begin{cases} \frac{2}{j-i+1} & \text{if } i \leq k < j \text{ or } i < k \leq j \\ \frac{2}{k-i+1} & \text{if } i < j \leq k \\ \frac{2}{j-k+1} & \text{if } k \leq i < j \end{cases}$$

b.

$$
\begin{aligned}
E[X_k] &\leq \sum_{i\leq k\leq j} \frac{2}{j-i+1} + \sum_{j<k} \frac{2}{k-i+1} + \sum_{k<i} \frac{2}{j-k+1} \\
&= \sum_{i=1}^{k}\sum_{j=k}^{n} \frac{2}{j-i+1} + \sum_{i=1}^{k-2}\sum_{j=i+1}^{k-1} \frac{2}{k-i+1} + \sum_{i=k+1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-k+1} \\
&= 2(\sum_{i=1}^{k}\sum_{j=k}^{n} \frac{1}{j-i+1} + \sum_{j=k+1}^{n} \frac{j-k-1}{j-k+1} + \sum_{i=1}^{k-2} \frac{k-i-1}{k-i+1})
\end{aligned}
$$

c. $E[X_k] \leq 4n$

d. the expected running time is $T(n) = E[X_k] + \mathcal{O}(n) = \mathcal{O}(n)$

III Data Structures

Chapter 10

Problem 10-1 Comparisons among list

	unsorted, singly linked	sorted, singly linked	unsorted, doubly linked	sorted, doubly linked
Search(L, k)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Insert(L, x)	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Delete(L, x)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Successor(L, x)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Predecessor(L, x)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Minimum(L, x)	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Maximum(L, x)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$

Problem 10-2 Mergeable heaps using linked lists

Binomial heap，是 mergeable heap 且用 linked list 实现
Make-Heap: $\mathcal{O}(1)$
Insert: $\mathcal{O}(\log n)$
Minimum: $\mathcal{O}(\log n)$
Extract-Min: $\mathcal{O}(\log n)$
Union: $\mathcal{O}(\log n)$

Problem 10-3 Searching a sorted compact list

a. 因为在 Compact-List-Search 和 Compact-List-Search' 中 RANDOM$(1, n)$ 产生的序列相同，所以 Compact-List-Search' 的返回值与 Compact-List-Search 相同且执行 for, while 循环的总次数至少为 t

b. The expected runnning time of Compact-List-Search'(L, n, k, t) is $\mathcal{O}(t + E[X_t])$

c.

$$\begin{aligned} E[X_t] &= \sum_{i=0}^{n} i \cdot Pr\{X_t = i\} \\ &= \sum_{i=1}^{n} Pr\{X_t \geq i\} \\ &\leq \sum_{i=1}^{n} (1 - r/n)^t \end{aligned}$$

d.

$$\sum_{r=0}^{n-1} r^t \leq \int_0^{n-1} r^t dr \leq n^{t+1}/(t+1)$$

e.

$$E[X_t] \leq \sum_{i=1}^{n}(1-r/n)^t = \frac{1}{n^t}\sum_{r=0}^{n-1} r^t \leq n/(t+1)$$

f. Compact-List-Search'(L, n, k, t) runs in $\mathcal{O}(t+E[X_t]) = \mathcal{O}(t+n/(t+1)) = \mathcal{O}(t+n/t)$ expected time

g. Compact-List-Search runs in $\mathcal{O}(\max(t, n/t)) = \mathcal{O}(\sqrt{n})$ expected time

h. 如果有相同的 key, 随机会大量重复判断一些无用的值, 使得渐进时间复杂度没有改进

Chapter 11

Problem 11-1 Longest-probe bound for hashing

a. 因为 $n \le m/2$,所以 $\alpha = n/m \le 1/2$,Random Variable X 表示不成功查找所要 probe 的次数,所以 $Pr\{X > k\} = Pr\{X \ge k+1\} \le \alpha^{(k+1)-1} = 2^{-k}$

b. 利用 (a) 中的结论, 令 $k = 2\log n$, 第 i 次 Insertion 需要多于 $2\log n$ probes 的概率为 $\mathcal{O}(2^{-k}) = \mathcal{O}(2^{-2\log n}) = \mathcal{O}(1/n^2)$

c.

$$Pr\{X > 2\log n\} \le \sum_{i=1}^{n} Pr\{X_i > 2\log n\} \le n \cdot \frac{1}{n^2} = \frac{1}{n}$$

d.

$$\begin{aligned}
E[X] &= \sum_{k=1}^{n} k \cdot Pr\{X = k\} \\
&= \sum_{k=1}^{\lceil 2\log n \rceil} k \cdot Pr\{X = k\} + \sum_{k=\lceil 2\log n \rceil + 1}^{n} k \cdot Pr\{X = k\} \\
&\le \sum_{k=1}^{\lceil 2\log n \rceil} \lceil 2\log n \rceil \cdot Pr\{X = k\} + \sum_{k=\lceil 2\log n \rceil + 1}^{n} n \cdot Pr\{X = k\} \\
&= \lceil 2\log n \rceil \sum_{k=1}^{\lceil 2\log n \rceil} Pr\{X = k\} + n \sum_{k=\lceil 2\log n \rceil + 1}^{n} Pr\{X = k\} \\
&\le \lceil 2\log n \rceil \cdot 1 + n \cdot (1/n) \\
&= \lceil 2\log n \rceil + 1 \\
&= \mathcal{O}(\log n)
\end{aligned}$$

Problem 11-2 Slot-size bound for chaining

a.

$$Q_k = (\frac{1}{n})^k (1 - \frac{1}{n})^{n-k} \binom{n}{k}$$

b. Let random variable X_i denote the number of keys hash to slot i. Frome (a), we have $Pr\{X_i = k\} = Q_k$

$$\begin{aligned} P_k &= Pr\{M = k\} \\ &= Pr\{(\max_{1 \le i \le n} X_i) = k\} \\ &\le \sum_{i=1}^{n} Pr\{X_i = k\} \\ &= nQ_k \end{aligned}$$

c.

$$\begin{aligned} Q_k &= (\frac{1}{n})^k (1 - \frac{1}{n})^{n-k} \frac{n!}{k!(n-k)!} \\ &< \frac{n!}{n^k k!(n-k)!} \\ &< \frac{1}{k!} \\ &< \frac{e^k}{k^k} \end{aligned}$$

d. (Omit!)

e.

$$\begin{aligned} E[M] &= \sum_{k=0}^{n} k \cdot Pr\{M = k\} \\ &= \sum_{k=0}^{k_0} k \cdot Pr\{M = k\} + \sum_{k=k_0+1}^{n} k \cdot Pr\{M = k\} \\ &\le \sum_{k=0}^{k_0} k_0 \cdot Pr\{M = k\} + \sum_{k=k_0+1}^{n} n \cdot Pr\{M = k\} \\ &= k_0 \sum_{k=0}^{k_0} Pr\{M = k\} + n \sum_{k=k_0+1}^{n} Pr\{M = k\} \\ &= k_0 \cdot Pr\{M \le k_0\} + n \cdot Pr\{M > k_0\} \end{aligned}$$

因为 $Pr\{M \le k_0\} \le 1$ 且

$$Pr\{M > k_0\} = \sum_{k=k_0+1}^{n} Pr\{M = k\} < \sum_{k=k_0+1}^{n} \frac{1}{n^2} < n \cdot \frac{1}{n^2} = \frac{1}{n}$$

所以，

$$E[M] \leq k_0 \cdot 1 + n \cdot \frac{1}{n} = k_0 + 1 = \mathcal{O}(\log n / \log \log n)$$

Problem 11-3 Quadratic probing

a. The probe sequence is <$h(k), h(k)+1, h(k)+1+2, h(k)+1+2+3, \cdots$>

$$h'(k,i) = (h(k) + \frac{1}{2}i + \frac{1}{2}i^2) \mod m.$$

b. We show that for any key k and for any probe number i and j such that $0 \leq i < j < m$, we have $h'(k,i) \neq h'(k,j)$.
If assume that $h'(k,i) = h'(k,j)$, it yields a contradiction. Therefore, this algorithm examines every table position in the worst case.

Problem 11-4 Hashing and authentication

Chapter 12

Problem 12-1 Binary search trees with equal keys

a. $\mathcal{O}(n^2)$

b. $\mathcal{O}(n \log n)$

c. $\mathcal{O}(n)$

d. Worst-Case:$\mathcal{O}(n^2)$, Expected runnning time:$\mathcal{O}(n \log n)$

Problem 12-2 Radix trees

因为 n 为所有 strings 的长度和,所以为所有 strings 构造一个 Radix tree 的时间复杂度为 $\Theta(n)$,排序所有 strings 即对这棵 Radix tree 进行 Preorder-Traverse,所以总时间复杂度为 $\Theta(n)$

Problem 12-3 Average node depth in a randomly built binary search tree

a. 根据定义,有

$$\sum_{x \in T} d(x, T) = P(T)$$

所以

$$\frac{1}{n} \sum_{x \in T} d(x, T) = \frac{1}{n} P(T)$$

b. $P(T) = P(T_L) + P(T_R) + n - 1$

c. Randomly built binary search tree,随机一个节点作为树根,所以根据定义,有

$$P(n) = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n - i - 1) + n - 1)$$

d.

$$P(n) = \frac{2}{n} \sum_{k=1}^{n-1} P(k) + \frac{(n-1)^2}{n} = \frac{2}{n} \sum_{k=1}^{n-1} P(k) + \Theta(n)$$

e. 由 Problem 7-3 Alternative quicksort analysis 的分析,可得 $P(n) = \mathcal{O}(n \log n)$

f. Binary Search Tree 中的所有非 Leaves 节点对应着 Quicksort 中的 Pivot

Problem 12-4 Number of different binary trees

a. 根据定义,空树的个数为 1,即 $b_0 = 1$,且通过选取树中 n 个节点作为树的根,可得

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k} \quad (n \geq 1)$$

b. 由

$$B(x) = \sum_{n=0}^{\infty} b_n x^n$$

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k}$$

可得

$$B(x) = xB(x)^2 + 1$$

解得

$$B(x) = \frac{1}{2x}(1 - \sqrt{1-4x})$$

c. 通过对 $\sqrt{1-4x}$ 进行 Taylor 展开,可得

$$\sqrt{1-4x} = 1 - \frac{2}{1!}x - \frac{4}{2!}x^2 - \frac{24}{3!}x^3 - \frac{240}{4!}x^4 - \cdots$$

所以,

$$\begin{aligned} B(x) &= \frac{1}{2x}(\frac{2}{1!}x + \frac{4}{2!}x^2 + \frac{24}{3!}x^3 + \frac{240}{4!}x^4 + \cdots) \\ &= \frac{1}{1!} + \frac{2}{2!} + \frac{12}{3!}x^2 + \frac{120}{4!}x^3 + \cdots \\ &= \sum_{n=1}^{\infty} \frac{(2n-2)!}{n(n-1)!(n-1)!}x^{n-1} \\ &= \sum_{n=0}^{\infty} \frac{1}{n+1}\binom{2n}{n}x^n \\ &= \sum_{n=0}^{\infty} b_n x^n \end{aligned}$$

即

$$b_n = \frac{1}{n+1}\binom{2n}{n}$$

d. 通过 Stirling‘s approximation

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))$$

可得

$$b_n = \frac{1}{n+1}\binom{2n}{n} = \frac{4^n}{\sqrt{\pi}n^{\frac{3}{2}}}(1+\mathcal{O}(\frac{1}{n}))$$

Chapter 13

Problem 13-1 Persistent dynamic sets

a. Insertion: 仅改变新插入的节点和其所有 ancestors
Deletion: 至多改变被删除的节点和其 successor 的 ancestors

b. 调用 Persistent-Tree-Insert$(T.root, k)$,返回 T'

Persistent-Tree-Insert(t, k)

```
 1  if t = NIL
 2      x.key = k
 3      x.left = x.right = NIL
 4  else x.key = t.key
 5      x.left = t.left
 6      x.right = t.right
 7      if k < t.key
 8          x.left = Persistent-Tree-Insert(t.left, k)
 9      else x.right = Persistent-Tree-Insert(t.right, k)
10  return x
```

c. 时间复杂度、空间复杂度都为 $\mathcal{O}(h)$

d. 若有 parent 指针,则每插入、删除一次,则所有点都要 copy 一次,所以时间复杂度、空间复杂度都为 $\Omega(n)$

e. We can still find the parent pointers we need in $\mathcal{O}(1)$ time without using parent pointers by using a stack to record the ancestors of the node.
We can also maintain the properties of Persistent Red-Black Tree in $\mathcal{O}(\log n)$ time.

Problem 13-2 Join operation on red-black trees

a. (Omit!)

b. 因为 $T_1.bh \geq T_2.bh$,根据 Red-Black Tree 的性质,从 T_1 的 Root 开始,若有 Right-Child,则向 Right-Child 走,否则向 Left-Child 走,必然存在一个 Black 节点 y 使得 $y.bh = T_2.bh$,且保证 y 是所有满足条件的节点中 key 最大的
算法的时间复杂度:$\mathcal{O}(\log n)$

c. 算法时间复杂度:$\mathcal{O}(1)$

RB-Join'(T_y, x, T_2)

```
1  z.left = T_y
2  z.right = T_2
3  z.parent = T_y.parent
4  z.key = x
5  if T_y.parent.left = T_y
6        T_y.parent.left = z
7  else T_y.parent.right = z
```

d. Red
因为树 T_y, T_2 的 Root 的颜色都为 Black, 如果 x 的 parent 的颜色也为 Red, 则将 x 的颜色置为 Black, 且将 T_y, T_2 的 Root 的颜色置为 Red, 递归向下调整, 保证 Black-Height 不变
时间复杂度: $\mathcal{O}(\log n)$

e. 同理, 若 $T_1.bh \leq T_2.bh$, 则用上述算法对称处理

f. The running time of RB-Join is $\mathcal{O}(\log n)$

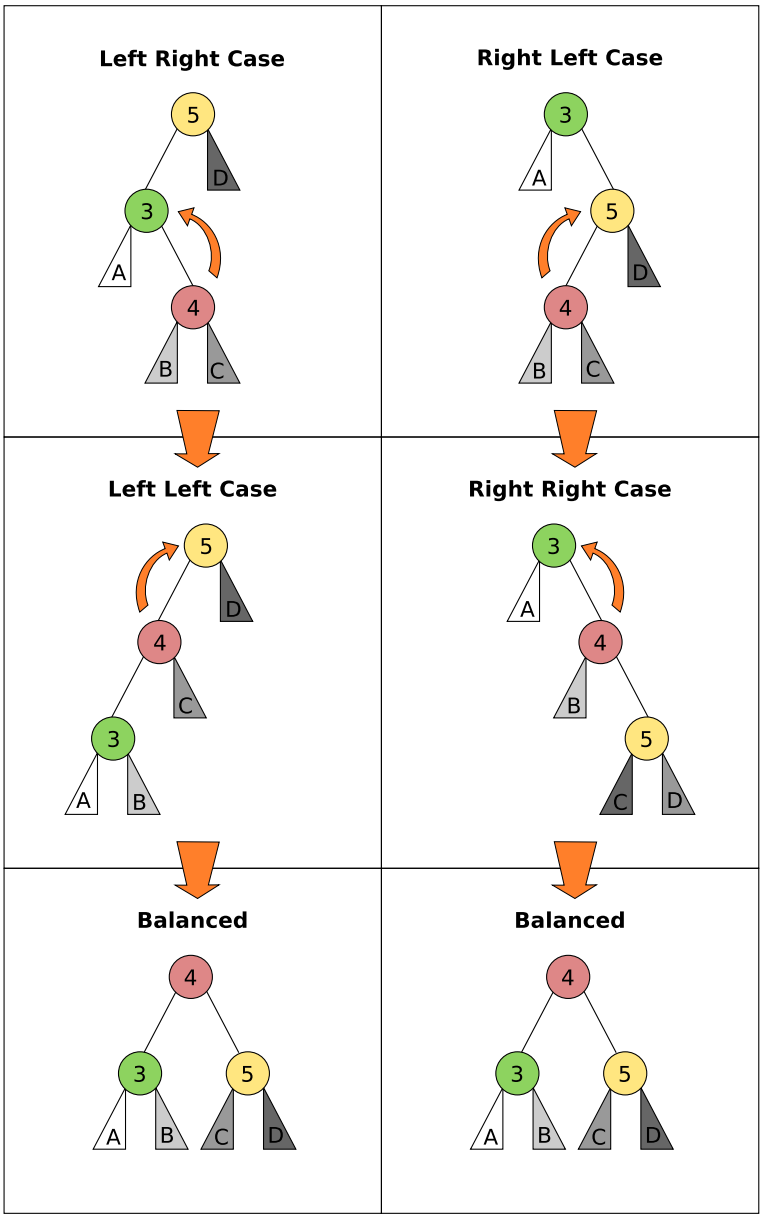
Problem 13-3 AVL trees

a. 设高度为 h 的 AVL tree 含有至少(严格)G_h 个节点, 则有

$$
\begin{aligned}
&G_h = G_{h-1} + G_{h-2} + 1 \quad n \geq 2 \\
&G_0 = 1 \\
&G_1 = 2
\end{aligned}
$$

所以, $G_h \geq F_h$(the h-th Fibonacci number), 即一棵含有 n 个节点的 AVL tree 的高度为 $\mathcal{O}(\log n)$ (golden ratio)

b. 仅四种情况 left-left, left-right, right-left, right-right, Balance(x) 的过程如图所示



c. the procedure AVL-Insert(x, z) consist of BST-Tree-Insert(T, z) and Balance(x)

d. BST-Tree-Insert(T, z): $\mathcal{O}(\log n)$
Balance(x): $\mathcal{O}(\log n)$
AVL-Insert(x, z) Total: $\mathcal{O}(\log n)$

Problem 13-4 Treaps

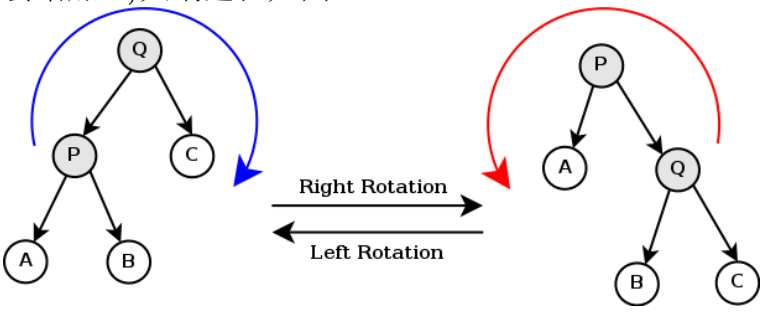
a. 对于一棵 BST，可以对每个节点进行 Rotation，所有这些 BST 与原始的 BST 都是同构的，然而在 Treap 中，这棵树要满足 Heap 的性质，所以不能进行 Rotation，即这棵 BST 是确定唯一的

b. 对于所有节点的 priority 序列，每一种排列都唯一的对应一种 Treap，即所有节点以一种排列顺序插入所形成的 BST，因为所有的节点的 priority 都是随机产生的，所以 Treap 的本质就是 Randomly built binary search trees，所以 the expected height of a treap is $\Theta(\log n)$

c. the procedure Treap-Insert consist of BST-Tree-Insert and Heap-Increase-Key by Rotation

d. BST-Tree-Insert: $\mathcal{O}(\log n)$
Heap-Increase-Key: $\mathcal{O}(\log n)$ (Rotation: $\mathcal{O}(1)$)
Treap-Insert Total: $\mathcal{O}(\log n)$

e. 对于一个 leaf node x，可以归纳，每次对于 x 进行一次 Rotation，其 $C+D$ 会增加 1，归纳过程如图



对于 Right-Rotation，设节点 x 为 P，A, B 分别为 x 的左右两棵子树，Q 是 P 的 parent，C 是 Q 的右子树，原始的 $C + D = left\_spine(B) + right\_spine(A)$，经过 Right-Rotation 后，$C' + D' = right\_spine(A) + 1 + left\_spine(B)$
同理，Left-Rotation 亦是如此，即对于插入树中的节点 x，其 Rotation 的次数等于 $C + D$

f. (Omit!)

g.

$$Pr\{X_{ik} = 1\} = \frac{(k-i-1)!}{(k-i+1)!} = \frac{1}{(k-i+1)(k-i)}$$

h.

$$E[C] = \sum_{i=1}^{k-1} Pr\{X_{ik} = 1\} = \sum_{i=1}^{k-1} \frac{1}{(k-i+1)(k-i)} = \sum_{j=1}^{k-1} \frac{1}{j(j+1)} = 1 - \frac{1}{k}$$

i. 由对称性,令 $k = n - k + 1$ 即可,即

$$E[D] = 1 - \frac{1}{n-k+1}$$

j. 因为 $E[C + D] = E[C] + E[D] < 2$,所以当向 Treap 中插入一个节点时,期望的 Rotation 次数小于 2

Chapter 14

IV Advanced Design and Analysis Techniques

Chapter 15

15-1 Longest simple path in a directed acyclic graph

Let $f(i)$ be the longest distant from vertex s to vertex i
Recursive formulation:

$$f(i) = \max_{(j,i)\in E}\{f(j) + d_{ji}\}$$

initial case:

$$f(s) = 0$$

Visit every vertex in Graph G by topological order
Running time: $\mathcal{O}(V + E)$

15-2 Longest palindrome subsequence

Let $f(i, j)$ be the length of the longest palindrome subsequence from the ith character to the jth character
Recursive formulation:

$$f(i,j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i > j \\ f(i+1, j-1) + 2, & \text{if } a_i = a_j \\ \max\{f(i+1,j), f(i,j-1)\}, & \text{otherwise} \end{cases}$$

Running time: $\mathcal{O}(n^2)$

15-3 Bitonic euclidean traveling-salesman problem

First, sort the points by x-coordinate, <$P_1, P_2, .., P_n$> $(x_{P_i} < x_{P_{i+1}})$
Let $P_{i,j}$ $(i < j)$ denote the bitonic path: $P_i \xrightarrow{left} P_1 \xrightarrow{right} P_j$ (a vertex-disjoint path)
It means that the bitonic path $P_{i,j}$ $(i < j)$ includes all vertices $P_1, P_2, \cdots, P_j$
Let $f(i, j)$ $(i \le j \le n)$ be the length of the shortest bitonic path $P(i, j)$
Recursive formulation:

$$\begin{cases} f(i,j) = f(i,j-1) + |P_{j-1,j}| & \text{if } (i < j-1) \\ f(i-1,i) = \min\limits_{1\le k<i-1}\{f(k,i-1) + |P_kP_i|\} & \\ f(i,i) = f(i-1,i) + |P_{i-1}P_i| & \end{cases}$$

initial case:

$$f(1,2) = |P_1P_2|$$

Running time: $\mathcal{O}(n^2)$

15-4 Printing neatly

Let $cost(i,j)$ denote the number of extra space characters at the end of the line which includes the $i \cdots j$-th word
Therefore,

$$cost(i,j) = \begin{cases} 0, & \text{if } j = n \text{ and } M - j + i - \sum_{k=i}^{j} l_k \geq 0 \\ \infty, & \text{if } M - j + i - \sum_{k=i}^{j} l_k < 0 \\ (M - j + i - \sum_{k=i}^{j} l_k)^3, & \text{otherwise} \end{cases}$$

Let $f(i)$ be the minimum number of extra space characters after placing first ith words
Recursive formulation:

$$f(i) = \min_{0 \leq j < i} \{f(j) + cost(j+1, i)\}$$

initial case:

$$f(0) = 0$$

Running time: $\mathcal{O}(n^2)$
Space: $\mathcal{O}(n)$

15-5 Edit distance

a. Let $X_i = x[1..i]$ and $Y_i = y[1..y]$ and $f(i,j)$ denote the minimum cost of transformating X_i to Y_j
Recursive formulation:

$$f(i,j) = \min \begin{cases} f(i-1, j-1) + cost(copy), & \text{if } x[i] = y[j] \\ f(i-1, j-1) + cost(replace), & \text{if } x[i] \neq y[j] \\ f(i-1, j) + cost(delete) & \\ f(i, j-1) + cost(insert) & \\ f(i-2, j-2) + cost(twiddle), & \text{if } i,j \geq 2, x[i] = y[j-1], \text{and } x[i-1] = y[j] \\ \min_{0 \leq k < m} \{f(k,n)\} + cost(kill), & \text{if } i = m \text{ and } j = n \end{cases}$$

initial case:

$$f(0,0) = 0$$

Running time and Space: $\mathcal{O}(nm)$

b.

$$cost(copy) = -1$$
$$cost(replace) = 1$$
$$cost(delete) = 2$$
$$cost(insert) = 2$$

maximize $\rightarrow$ minimize

15-6 Planning a company party

Let $f(x,0)$ denote the maximum sum of the conviviality ratings of the guests when the emplyee x doesn't attend the party and $f(x,1)$ denote the maximum sum of the conviviality ratings of the guests when the emplyee x attends the party
Recursive formulation:

$$f(x,0) = \sum_{y \in Son\{x\}} \max\{f(y,0), f(y,1)\}$$
$$f(x,1) = \sum_{y \in Son\{x\}} f(y,0) + rating(x)$$

initial case: if x is a leaf,

$$f(x,0) = 0$$
$$f(x,1) = rating(x)$$

Running time: $\mathcal{O}(V+E)$

15-7 Viterbi algorithm

a. Let $f(x,i)$ denote whether there exist a path in G that begins at x and has a sequence $<\sigma_1, \sigma_2, \cdots, \sigma_i>$ as it label
Recursive formulation:

$$f(x,i) = \bigvee_{y \text{ s.t. } \sigma(y,x)=\sigma_i} f(y, i-1)$$

initial case:

$$f(x,0) = true$$

Running time: $\mathcal{O}(kn^2)$

b. Let $f(x,i)$ denote the maximum probability of a path in G that begins at x and has a sequence $<\sigma_1, \sigma_2, \cdots, \sigma_i>$ as it label
Recursive formulation:

$$f(x,i) = \max_{y \text{ s.t. } \sigma(y,x)=\sigma_i} \{f(y,i-1) \cdot p(y,x)\}$$

Running time: $\mathcal{O}(kn^2)$

15-8 Image compression by seam carving

a. $\mathcal{O}(3^m)$

b. Let $f(i,j)$ denote the seam with the lowest disruption measure of $i \times j$ array $A[1..i, 1..j]$
Recursive formulation:

$$f(i,j) = \min\{f(i-1,j-1), f(i-1,j), f(i-1,j+1)\} + d[i,j]$$

The answer is find the i such that minimize $f(m,i)$
Running time: $\mathcal{O}(nm)$

Problem 15-9 Breaking a string

Let $L[0] = 0$ and $L[m+1] = n$
Let $f(i,j)$ denote the lowest cost of breaks that breaking every break points in $L[i..j-1]$
Recursive formulation:

$$f(i,j) = \min_{i<k<j} \{f(i,k) + f(k,j)\} + (L[j] - L[i])$$

initial case:

$$f(i,i+1) = 0$$

The answer is $f(0,m+1)$ Running time: $\mathcal{O}(m^3)$

Problem 15-10 Planning an investment strategy

a. (Explicit)

b. Let $f(k,i)$ denote in k years and we inveset the ith investment at the kth year, the maximum amount of the monney
Recursive formualation:

$$f(k,i) = \max_{j \neq i}\{f(k-1,i) - f_1, f(k-1,j) - f_2\} + d \cdot r_{ik}$$

c. Running time: $\mathcal{O}(years \cdot inverstments^2) = \mathcal{O}(n^2)$

d. (Omit!)

Problem 15-11 Inventory planning

Let $D(x) = \sum_{i=1}^{x} d_i$ and $D(0) = 0$, therefore, $D(n) = D$
Let $f(i, j)$ denote the minimum cost of first ith month that manufactured j machines
Recursive formulation:

$$f(i,j) = \min_{j \geq k \geq D(i-1)} \{f(i-1,k) + \max\{0, j-k-m\} \cdot c\} + h(j - D(i)), \text{ for } D \geq j \geq D(i)$$

The answer is $f(n, D)$
Running time: $\mathcal{O}(nD^2)$

Problem 15-12 Signing free-agent baseball players

For every free-agent player x, let $pos(x)$ denote the player's position, $cost(x)$ denote the amount of money it will cost to sign the player, and $vorp(x)$ denote the player's VORP
Let $f(i, j)$ denote the maximum total VORP of the players signed for first i position and used j money
Recursive formulation:

$$f(i,j) = \max_{k \text{ s.t. } pos(k)=i} \{f(i-1,j), f(i-1, j-cost(k)) + vorp(k)\}$$

The answer is $f(N, X)$
Running time: $\mathcal{O}(\max(N, P) \cdot X)$

Chapter 17

Problem 17-1 Bit-reversed binary counter

a. (Trivial!) The running time of the bit-reversal permutation on an array of length $n = 2^k$ in $\mathcal{O}(nk)$ time

b. It is the same as the procedure Increment for a binary counter. Therefore, the bit-reversal permutation on an n-element array to be performed in a total of $\mathcal{O}(n)$ time

c. Yes

Problem 17-2 Making binary search dynamic

a. Using binary search to search each sorted array one by one, until find the element that we search
The running time of the worst case:

$$\begin{aligned} T(n) =& \Theta(\log 1 + \log 2 + \log 2^2 + \cdots + \log 2^{k-1}) \\ =& \Theta(0 + 1 + \cdots + (k-1)) \\ =& \Theta(\frac{1}{2}k(k-1)) \\ =& \Theta(\log^2 n) \end{aligned}$$

Therefore, the running time of the worst case is $\Theta(\log^2 n)$

b.
1. create a new sorted array of size 1, A'_0, containing the new element to be inserted
2. if A_0 is empty, then we replace A_0 by A'_0; Otherwise, we merge the two sorted array into another sorted array, A'_1
3. repeat these step, until for some i that A_i is empty

The amortized running time of the worst case:

We use accounting method to analyse the running time. We can charge k to insert an element. 1 pays for the insertion, and we put $(k-1)$ on the inserted from to pay for it being involved in merges later on.
And it can move to a higher-indexed array at most $k-1$ times, so the $k-1$ on the item suffices to pay for all the times it will never be involved in merges.
Therefore, the amortized running time of the worst case is $\mathcal{O}(\log n)$

c. **1.** find the smallest j for which the array A_j with 2^j elements is full
Let y be the last element of A_j

2. find x, assume it is in A_i

3. delete x from A_i and insert y into A_i with correct place

4. divide A_j(left $2^j - 1$ elements) into $A_0, A_1, \cdots, A_{j-1}$

The running time is $\Theta(n)$

Problem 17-3 Amortized weight-balanced trees

a. **1.** list all the elements in the subtree rooted at x by inorder tree walk, i.e. a sorted list

2. rebuild the subtree so that it becomes $1/2$-balanced

The running time of the algorithm is $\Theta(x.\textit{size})$ and use $\mathcal{O}(x.\textit{size})$ auxiliary storage

b. Since the height of the α-balanced binary search tree is $\mathcal{O}(\log_{\frac{1}{\alpha}} n) = \mathcal{O}(\log n)$
Therefore, performing a search in an n-node α-balanced binary search tree takes $\mathcal{O}(\log n)$ worst-case time

c. (Trivial!) Any binary search tree has nonnegative potential and that a $1/2$-balanced tree has potential 0

d.

$$c = \frac{1}{2\alpha - 1}$$

e. After insert or delete a node from an n-node α-balanced tree, we use $\mathcal{O}(1)$ amortized time to rebalance the tree

Problem 17-4 The cost of restructuring red-black trees

(From CLRS Solution)

a. (Omit!)

b. All cases except or case 1 of RB-INSERT-FIXUP and case 2 of RB-DELETE-FIXUP are terminating

c. Case 1 of RB-INSERT-FIXUP decreases the number of the red nodes by 1. Hence, $\Phi(T') = \Phi(T) - 1$

d. Line 1-16 of RB-INSERT-FIXUP causes one node insertion and a unit increase in potential.
The nonterminating case of RB-INSERT-FIXUP causes there color changing and decreases the potential by 1.
The terminating case of RB-INSERT-FIXUP causes one rotation each and do not affect the potential.

e. By the assumption that 1 unit of potenital can pay for the structural modifications performed by and of the three cases of RB-INSERT-FIXUP, we can conclude that the amortized number of structural modifications performed by any call of RB-INSERT is $\mathcal{O}(1)$

f. $\Phi(T') \leq \Phi(T) - 1$ for all nonterminating cases of RB-INSERT-FIXUP.
The amortized number of structural modifications performed by any call of RB-INSERT-FIXUP is $\mathcal{O}(1)$

g. $\Phi(T') \leq \Phi(T) - 1$ for all nonterminating cases of RB-DELETE-FIXUP.
The amortized number of structural modifications performed by any call of RB-DELETE-FIXUP is $\mathcal{O}(1)$

h. Since the amortized number of strutural modification in each operation is $\mathcal{O}(1)$, any sequence of m RB-INSERT and RB-DELETE operations performs $\mathcal{O}(m)$ structural modifications in the worst case

Problem 17-5 Competitive analysis of self-organizing lists with move-to-front

a. (Trivial!) The worst-case costs for H on an access sequence σ is $C_H(\sigma) = \Omega(mn)$

b. (Trivial!) By the defination of the $rank_L(x)$ and c_i, we have

$$c_i = 2 \cdot rank_L(x) - 1$$

c. (Trivial!) By the defination of the L_i^*, t_i^* and c_i^*, we have

$$c_i^* = rank_{L_{i-1}^*}(x) + t_i^*$$

d. Since an adjacent transposition change only one inversion, and $\Phi(L_i) = 2q_i$, therefore, a transposition either increases the potential by 2 or decreases the potential by 2

e. (Trivial!) $rank_{L_{i-1}}(x) = |A| + |B| + 1$ and $rank_{L_{i-1}^*} = |A| + |C| + 1$

f. For analysing the potential function of list L_i, i.e. $\Phi(L_i)$, we first assume that $L_i^* = L_{i-1}^*$, therefore, $\Phi(L_i) - \Phi(L_{i-1}) = 2(|A| - |B|)$. Then, the transposition from list L_{i-1}^* to L_i^*, it at most increases potential by $2 \cdot t_i^*$. Thus, we have

$$\Phi(L_i) - \Phi(L_{i-1}) \leq 2(|A| - |B| + t_i^*)$$

g.

$$\begin{aligned}
\hat{c_i} =& c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
=& 2(|A| + |B| + 1) - 1 + \Phi(L_i) - \Phi(L_{i-1}) \\
\leq& 2(|A| + |B| + 1) - 1 + 2(|A| - |B| + t_i^*) \\
=& 4\,|A| + 1 + 2t_i^* \\
\leq& 4(|A| + |C| + 1 + t_i^*) \\
=& 4c_i^*
\end{aligned}$$

h. (Trivial!) The cost $C_{MTF}(\sigma)$ of access sequence σ with move-to-front is at most 4 times the cost $C_H(\sigma)$ of σ with any other heuristic H, assuming that both heuristics start with the same list

V Advanced Data Structures

VI Graph Algorithms

Chapter 22

Problem 22-1 Classifying edges by breadth-first search

a.

1. 对一个无向图进行 BFS 的时候,若有 back edge 或是 forward edge,利用 BFS 的性质和无向图的性质,其在 BFS Tree 中都是 tree edge
2. 对于一条 tree edge (u, v),我们有 $v.\pi = u$,即利用 BFS 性质,可得 $v.d = u.d + 1$
3. 若边 (u, v) 是 cross edge,则当我们访问 u 的时候,v 必然已经进入队列,否则 (u, v) 为 tree edge;由 Lemma 22.3,我们有 $v.d \leq u.d+1$;由 Corollary 22.4,我们有 $v.d \geq u.d$,即 $v.d = u.d$ or $v.d = u.d + 1$

b.

1. 在 BFS 过程中 forward edge 即为 tree edge
2. 与无向图相同
3. 在 BFS 过程中,对于任意边 (u, v) 均满足 $v.d \leq u.d + 1$
4. 易得对于所有 v,有 $v.d \geq 0$,因为 (u, v) 是 back edge,即 v 是 u 的祖先,所以有 $v.d \leq u.d$,即 $0 \leq v.d \leq u.d$

Problem 22-2 Articulation points, bridges, and biconnected components

a. 若 G_π 的 root 只有一个儿子,在删除 root 之后,图的连通度不会增加,即不是 articulation points;若 G_π 的 root 有多于两个儿子,在删除 root 之后,图的连通度会增加,即 G_π 是 articulation points

b. 若 v 为 articulation points,则 v 的儿子中必然有一个儿子,使得其没有 back edge 连接到 v 的祖先;否则若 v 没有这样的儿子,由于是无向图,当删除结点 v 之后,图依然保持连通,即 v 不是 articulation points;反之亦然

c. 利用 DFS,可以在 $\mathcal{O}(V + E) = \mathcal{O}(E)$ 的时间内维护每个结点的 low 值

d. Vetex v is an articulation points iff. ($v.low \geq v.d$) or (v is the root of G_π and v has at least two children in G_π). The running time of the algorithm is $\mathcal{O}(V + E) = \mathcal{O}(E)$

e. 设边 (u, v) 是图 G 的 bridge,若边 (u, v) 在一个图 G 的一个 simple cycle 中,那么当删除边 (u, v) 之后,必然存在一条 u 到 v 的 path,使得 u 和 v 依然保持连通,与边 (u, v) 是 bridge 矛盾,所以边 (u, v) 必然不在一个 simple cycle 之中;反之亦然

f. Edge $e = (u, v)$ is a bridge iff. (e is a tree edge) and ($u.d < v.low$). The running time of the algorithm is $\mathcal{O}(V + E) = \mathcal{O}(E)$

g. Equivalence relation: $e_1 \sim e_2$ iff. ($e_1 = e_2$) or (e_1 and e_2 lie on some common simple cycle). Therefore, the biconnected components of G partition the nonbridge edges of G.

h. 在 DFS 过程中,利用一个 stack 来存放所访问的边,若遇到 articulation point 或 bridge,则将之后的所有的边 pop,并将 bcc 标记成相同标号,运行时间 $\mathcal{O}(V + E) = \mathcal{O}(E)$

Problem 22-3 Euler tour

a. 充分性易证,我们来证必要性:若对于任意结点 v,均满足 in-degree(v) = out-degree(v),那么对于任意结点 v,必然存在一个 cycle 包含结点 v;我们可以利用上述引理,来扩展构造整个图的 Euler tour

b. 利用上述引理,首先对一个结点找到一个包含它的 cycle,然后从图中删除这个 cycle 上的所有边,继续找,直到图中不含有任何边;找每个环的公共结点,合并所有的环,构成原图的 Euler tour

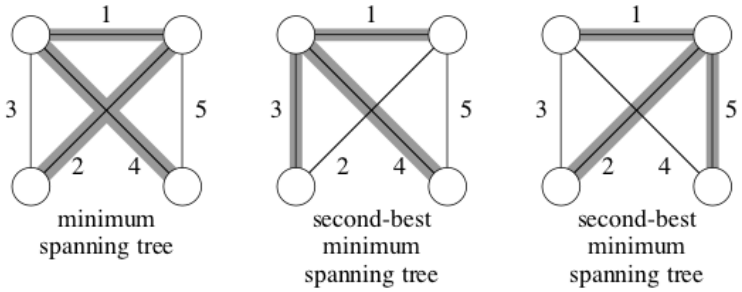
Problem 22-4 Reachability

计算 G^T,以 $L(v)$ 值递增顺序对 G^T 进行 DFS,若结点 v 在 DFS Tree 中的 root 为 u,那么 min(v) $= u$;算法运行时间为 $\mathcal{O}(V + E)$

Chapter 23

Problem 23-1 Second-best minimum spanning tree

a. Since a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut(Exercise 23.1-6). Therefore, For a graph that all edge weights are distinct, the minimum spanning tree is unique.
But the second-best minumum spanning tree need not be unique. Here is an example,



b. 只需要证明,若替换 MST 中的两条或两条以上的边,我们得不到一个 Second-best Minimum Spanning Tree
设 T 是图 G 的 Minimum Spanning Tree,存在 Second-best Minimum Spanning Tree T',其中至少有两条边和 T 中不同;设边 (u,v) 是 $T-T'$ 中权值最小的边,将边 (u,v) 加入 T' 中,则必然形成一个 cycle,cycle 中包含一些 $T'-T$ 中的边 (x,y),则 $w(u,v)<w(x,y)$
(若 $w(u,v)>w(x,y)$,则将边 (x,y) 加入 T,我们得到一个 cycle,在 cycle 中包含一些 $T-T'$ 中的边 (u',v'),设 $T''=T-\{(u',v')\}\cup\{(x,y)\}$ 是一棵 Spanning Tree,因为 T 是 MST,所以 $w(x,y)>w(u',v')$,即 $w(u,v)>w(x,y)>w(u',v')$,与我们对于边 (u,v) 的选择矛盾)
所以对于 Spanning Tree $T'-\{(x,y)\}\cup\{(u,v)\}$,其权值比 $w(T')$ 小,且其和 MST T 不同,即 T' 不是 Second-best Minimum Spanning Tree

c. For each vertex u, using DFS to find max$[u,v], v\in V$. The running time is $\mathcal{O}(V^2)$

d.
1. Find Minimum Spanning Tree of G.....$\mathcal{O}(E+V\log V)$
2. Compute max$[u,v]$ for all $u,v\in V$.....$\mathcal{O}(V^2)$
3. Find an edge $(u,v)\notin T$, that minimizes $w(u,v)-w(\text{max}[u,v])$.....$\mathcal{O}(E)$

The Second-best minimum spanning tree is $T-\{\text{max}[u,v]\}\cup\{(u,v)\}$
The running time of the algorithm is $\mathcal{O}(V^2)$

Problem 23-2 Minimum spanning tree in sparse graphs

a. 因为 A 是图 G' 的 MST,而 T 中的所有边都是两端结点之间最短的边,所以 $T \cup \{(x,y).orig' : (x,y) \in A\}$ 是图 G 的 MST

b. Obviously, $|G'.V| \leq |V|/2$

c. Using disjoint set, we can implement MST-REDUCE so that it runs in $\mathcal{O}(E)$ time

d. ???

e. ???

f. ???

Problem 23-3 Bottleneck spanning tree

a. 我们证明原命题的逆否命题 (contrapositive), 即无向图 G 中的一棵 Spanning Tree T 不是一棵 Bottleneck Spanning Tree, 那么 T 亦不是 G 的 Minimum Spanning Tree
设边 e 是 T 中权值最大的边, 且设边 e 两端连接的子树为 T_1, T_2, 因为 T 不是一棵 Bottleneck Spanning Tree, 所以必然存在一条边 e' 连接 T_1, T_2, 且 $w(e') < w(e)$, 所以 T 亦不是 G 的 Minimum Spanning Tree

b. 从图 G 中删除所有权值大于 b 的边; 若图依然连通, 则存在一棵最大权值不大于 b 的 Bottleneck Spanning Tree, 否则不存在

c. 对 b 的值进行二分答案(在所有边的权值中), 利用 Part(b) 中的算法来验证是否存在 Bottleneck Spanning Tree; Running Time: $\mathcal{O}(E + \frac{E}{2} + \frac{E}{4} + ...) = \mathcal{O}(E)$
Any better solution ???

Problem 23-4 Alternative minimum-spanning-tree algorithms

a. Correct!
By the conclusion of Exercise 23.1-5
The running time of the algorithm is $\mathcal{O}(E(E+V))$

b. Wrong!
The running time of the algorithm is $\mathcal{O}(E\alpha(V))$

c. Correct!
By the conclusion of Exercise 23.1-5
The running time of the algorithm is $\mathcal{O}(VE)$

Any better solution ???

Chapter 24

Chapter 25

Problem 25-1 Transitive closure of a dynamic graph

a. Let the edge (u, v) is inserted into G. For all $i, j \in V$, if $t_{iu} \bigwedge t_{vj} = 1$, then we have $t_{ij} = 1$
The running time is $\mathcal{O}(V^2)$

b. Consider a example, the original graph is a chain, i.e. $G : v_1 \to v_2 \to ... \to v_n$. The matrix of the connectivity T is a upper-triangular matrix. Now we insert an edge (v_n, v_1), then the matrix of the connectivity T is an all one matrix. The changing value is at least $\Omega(V^2)$.
Therefore, a graph G and an edge e such that $\Omega(V^2)$ time is required to update the transitive closure after the insertion of e into G, no matter what algorithm is used.

c. The algorithm:

Insert(u, v)
```
1  for i = 1 to |V|
2      if t_iu = 1 and t_iv = 0
3          for j = 1 to |V|
4              if t_vj = 1
5                  t_ij = 1;
```

Since the condition of $t_{iv} = 0$ is true at most $\mathcal{O}(V^2)$ times, therefore, the running time of the algorithm is $\mathcal{O}(V^3)$

Problem 25-2 Shortest paths in ϵ-dense graphs

a. In a d-ary min-heap, the asymptotic running time of
Insert: $\Theta(\log_d n)$
Extract-Min: $\Theta(d \log_d n)$
Decrease-Key: $\Theta(\log_d n)$

If we choose $d = \Theta(n^\alpha)$ or some constant $0 < \alpha \leq 1$, the asymptotic running time of
Insert: $\Theta(\frac{1}{\alpha})$
Extract-Min: $\Theta(\frac{d}{\alpha}) = \Theta(\frac{n^\alpha}{\alpha})$
Decrease-Key: $\Theta(\frac{1}{\alpha})$

b. $d = V^\epsilon$

c. To solve the all-pairs shortest-paths problem on an ϵ-dense directed graph $G = (V, E)$ with no negative-weight edges, we can run $|V|$ times Dijkstra's algorithms described above, the total running time of the algorithm is $\mathcal{O}(VE)$

d. Using Johnson's algorithm to reweight the graph $G = (V, E)$, and then execute the above algorithm. The running time of the algorithm is $\mathcal{O}(VE + VE) = \mathcal{O}(VE)$

Chapter 26

Problem 26-1 Escape problem

a. For all vertices v in graph $G = (V, E)$, we can split it into two corresponding vertices v_0, v_1 and there is a edge between two vertices (v_0, v_1)
Now we rebuild the graph G
$V' = \{v_0, v_1 : \text{if } v \in V\}$ and
$E' = \{(v_0, v_1) : \text{if } v \in V\} \cup \{(u_1, v_0) : \text{if } (u, v) \in E\}$.
We also define the capacity function, $c'(u, v)$ for all $(u, v) \in E'$

$$\begin{cases} c'(v_0, v_1) = c(v), \text{ if } v \in V \\ c'(u_1, v_0) = c(u, v), \text{ if } (u, v) \in E \\ c'(u, v) = 0, \text{ otherwise} \end{cases}$$

Thus, the vertex capacities problem is reduced to an ordinary maximum-flow problem on a flow network

b. The graph $G = (V, E)$, defined as
$V = \{s, t\} \cup \{v_{i,j} : \text{for every } i, j = 1, 2, \cdots, n\}$ and
$E = \{(s, v) : \text{if } v \text{ is a starting point}\} \cup \{(v, t) : \text{if } v \text{ is a boundary point}\} \cup \{(u, v) : \text{if } u, v \text{ are adjacent points on the grid}\}$
The capacity function for edges is $c(u, v) = 1$, for all $(u, v) \in E$

It's easy to find that $|V| = \mathcal{O}(n^2)$ and $|E| = \mathcal{O}(n^2 + m)$

A good implement algorithm for network flow can solved this problem in $\mathcal{O}(n^6)$ time

Problem 26-2 Minimum path cover

a. To find a minimum path cover of a directed acyclic graph $G = (V, E)$, we construct the graph $G' = (V', E')$, where
$V' = \{x_0, x_1, \cdots, x_n\} \cup \{y_0, y_1, \cdots, y_n\}$,
$E' = \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i, j) \in E\}$,
and all edge in E' have an unit capacity. Then the maximum-flow of graph G' is $|f|$, and the minimum path cover of the graph G is $|V| - |f|$
The running time of the algorithm is $\mathcal{O}(VE)$

b. The above alogorithm doesn't work for directed graphs that contain cycles

Problem 26-3 Algorithmic consulting

a. Since the cut (S, T) of G is a finite-capacity, then there is no such edge where the capacity is infinite, i.e. edge $(A_i, J_j) \notin (S, T)$ for all $i = 1, 2, \cdots, n$, $j = 1, 2, \cdots, m$

b. The maximum net revenue is $\sum_i P_i - |c|$

c.
1. find the maximum flow of the graph G, and obtain the residual network G_f, $\mathcal{O}((n+m)^2 \cdot (n+m+r))$
2. find the minimum cut set (S, T) by DFS the residual, $\mathcal{O}(n+m+r)$
3. if edge $(s, A_i) \in (S, T)$, then we hire expert i
   if edge $(J_i, t) \notin (S, T)$, then we accept the job i
   $\mathcal{O}(n+m)$

The total running time is determined by the implemention of the maximum flow algorithm

Problem 26-4 Updating maximum flow

a. Just execute one more iteration of Ford-Fulkerson algorithm.
Argument:
If the edge (u, v) does not cross a minimum cut, then incresing the capacity of the edge (u, v) does not change the capacity of the minimum cut as well as maximum flow.
If the edge (u, v) does cross a minimum cut, then it increases the capacity of the minimum cut as well as maximum flow at most 1.
The both cases need only find augmenting path one more time.

The running time of the algorithm is $\mathcal{O}(V + E)$

b.
1. the new flow $f'(u, v) = f(u, v) - 1$
2. if there is a augmenting path from u to v, then augment an unit flow along the augmenting path
   otherwise, find augmenting paths from u to s and from t to v, decreasing an unit flow along the augmenting path

The running time of the algorithm is $\mathcal{O}(V + E)$

Problem 26-5 Maximum flow by scaling

a. Since $C = \max_{(u,v)\in E} c(u, v)$, and there are at most $|E|$ edges in graph G, therefore, a minimum cut of G has capacity at most $C\,|E|$

b. Just ignore the edges whose capacity less than K when finding an augmenting path.
It takes $\mathcal{O}(V + E) = \mathcal{O}(V + E)$ time

c. Since when $K = 1$, the procedure Max-Flow-By-Scaling(G, s, t) is the typical Ford-Fulkerson algorithm. Since the Ford-Fulkerson algorithm returns a maximum flow, therefore, Max-Flow-By-Scaling returns a maximum flow

d. Since there are at most $|E|$ edges and each time line 4 is executed, the capacity of the residual network G_f is at most $2K$. Thus, the capacity of a minimum cut of the residual network G_f is at most $|E| \times 2K = 2K\,|E|$

e. By part **d**, the capacity of a minimum cut of the residual network G_f is at most $2K\,|E|$ each time line 4 is executed, i.e. the maximum flow of G_f is at most $2K\,|E|$. And each time the inner while loop finds an augmenting path of capacity at least K, the flow increases by at least K.
Thus, the inner while loop of lines 5-6 executes $\mathcal{O}(2K\,|E|)/K = \mathcal{O}(E)$ times for each value of K

f. By above arguments, we can easily obtain the running time of the procedure Max-Flow-By-Scaling is $\mathcal{O}(E^2 \log C)$