算法导论读书笔记

1. 归并排序例程

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p+r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q+1, r)
5      MERGE(A, p, q, r)
```

归并排序改统计逆序对数量：只要归并的时候左大于右加上左边即可

2. 斯特林近似公式

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

中 e 是自然对数的底。正

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\lg(n!) = \Theta(n\lg n)$$

(3.19)。对所有 $n \geqslant 1$，下

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n}$$

3. 斐波那契数公式：指数增长

$$F_i = \left\lfloor \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \right\rfloor$$

4. 最大子数组的递归算法（跨越中间例程）

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4      sum=sum+A[i]
 5      if sum > left-sum
 6          left-sum = sum
 7          max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11      sum=sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2      return (low, high, A[low])                    // base case: only one
 3  else mid=⌊(low+high)/2⌋
 4      (left-low, left-high, left-sum) =
            FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5      (right-low, right-high, right-sum) =
            FIND-MAXIMUM-SUBARRAY(A, mid+1, high)
 6      (cross-low, cross-high, cross-sum) =
            FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8          return (left-low, left-high, left-sum)
 9      elseif rightr-sum ≥ left-sum and right-sum ≥ cross-sum
10          return (right-low, right-high, right-sum)
11      else return (cross-low, cross-high, cross-sum)
```

5. 矩阵相乘的 Strassen 算法 $T(n)=\Theta(n^{\lg 7})$
6. 代入法：先猜想在证明，可以减去一个低阶因子，换元法
7. 递归树：帮助猜想
8. 主方法

定理 4.1（主定理） 令 $a\geqslant 1$ 和 $b>1$ 是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将 n/b 解释为$\lfloor n/b\rfloor$或$\lceil n/b\rceil$。那么 $T(n)$有如下渐近界：

1. 若对某个常数 $\varepsilon>0$ 有 $f(n)=O(n^{\log_b a-\varepsilon})$，则 $T(n)=\Theta(n^{\log_b a})$。
2. 若 $f(n)=\Theta(n^{\log_b a})$，则 $T(n)=\Theta(n^{\log_b a}\lg n)$。
3. 若对某个常数 $\varepsilon>0$ 有 $f(n)=\Omega(n^{\log_b a+\varepsilon})$，且对某个常数 $c<1$ 和所有足够大的 n 有 $af(n/b)\leqslant cf(n)$，则 $T(n)=\Theta(f(n))$。 ■

证明：如果 $f(n)=\Theta(n^{\log_b a}\lg^k n)$，其中 $k\geqslant 0$，那么主递归式的解为 $T(n)=\Theta(n^{\log_b a}\lg^{k+1} n)$。为简单起见，假定 n 是 b 的幂。

9.

第五章 随机算法

1. 指示器随机变量

为了分析雇用问题在内的许多算法，我们采用指示器随机变量(indicator random variable)。它为概率与期望之间的转换提供了一个便利的方法。给定一个样本空间 S 和一个事件 A，那么事件 A 对应的**指示器随机变量** $\mathrm{I}\{A\}$ 定义为：

$$\mathrm{I}\{A\} = \begin{cases} 1 & \text{如果 } A \text{ 发生} \\ 0 & \text{如果 } A \text{ 不发生} \end{cases} \tag{5.1}$$

在一次抛掷硬币时，正面朝上的期望次数就是指示器变量 X_H 的期望值：

$$\begin{aligned} \mathrm{E}[X_H] &= \mathrm{E}[\mathrm{I}\{H\}] = 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{T\} \\ &= 1 \cdot (1/2) + 0 \cdot (1/2) = 1/2 \end{aligned}$$

因此抛掷一枚标准硬币时，正面朝上的期望次数是 1/2。如下面引理所示，一个事件 A 对应的指示器随机变量的期望值等于事件 A 发生的概率。

引理 5.1 给定一个样本空间 S 和 S 中的一个事件 A，设 $X_A = \mathrm{I}\{A\}$，那么 $\mathrm{E}[X_A] = \Pr\{A\}$。

证明 由等式(5.1)指示器随机变量的定义，以及期望值的定义，我们有

$$\mathrm{E}[X_A] = \mathrm{E}[\mathrm{I}\{A\}] = 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\overline{A}\} = \Pr\{A\}$$

2. 用指示器随机变量分析雇用问题

在第 6 行中，应聘者 i 被雇用，正好应聘者 i 比从 1 到 $i-1$ 的每一个应聘者优秀。因为我们已经假设应聘者以随机顺序出现，所以前 i 个应聘者也以随机次序出现。这些前 i 个应聘者中的任意一个都等可能地是目前最有资格的。应聘者 i 比应聘者 1 到 $i-1$ 更有资格的概率是 $1/i$，因而也以 $1/i$ 的概率被雇用。由引理 5.1，可得

$$\mathrm{E}[X_i] = 1/i \tag{5.3}$$

现在可以计算 $\mathrm{E}[X]$：

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] && \text{(根据等式(5.2))} && (5.4) \\ &= \sum_{i=1}^{n} \mathrm{E}[X_i] && \text{(根据期望的线性性质)} \\ &= \sum_{i=1}^{n} 1/i && \text{(根据等式(5.3))} \\ &= \ln n + O(1) && \text{(根据等式(A.7))} && (5.5) \end{aligned}$$

尽管我们面试了 n 个人，但平均起来，实际上大约只雇用他们之中的 $\ln n$ 个人。我们用下面的引

3. 随机排列算法

```
PERMUTE-BY-SORTING(A)
1  n = A.length
2  let P[1..n] be a new array
3  for i = 1 to n
4      P[i] = RANDOM(1, n^3)
5  sort A, using P as sort keys
```

```
RANDOMIZE-IN-PLACE(A)
1  n = A.length
2  for i = 1 to n
3      swap A[i] with A[RANDOM(i, n)]
```

[1, n^3]为了保证尽可能每个数字优先级都不同

4. 礼券收集者问题，b 种不同礼券需要 blnb 次才能收集齐

5. 抛掷一枚均匀的硬币 n 次，出现最长连续正面的期望是 O(lnn).

$\sum_{j=0}^{n} \Pr\{L_j\} = 1$，我们有 $\sum_{j=0}^{2\lceil \lg n \rceil - 1} \Pr\{L_j\} \leqslant 1$。因此，我们得到

$$
\begin{aligned}
\mathrm{E}[L] &= \sum_{j=0}^{n} j\Pr\{L_j\} = \sum_{j=0}^{2\lceil \lg n \rceil - 1} j\Pr\{L_j\} + \sum_{j=2\lceil \lg n \rceil}^{n} j\Pr\{L_j\} \\
&< \sum_{j=0}^{2\lceil \lg n \rceil - 1} (2\lceil \lg n \rceil)\Pr\{L_j\} + \sum_{j=2\lceil \lg n \rceil}^{n} n\Pr\{L_j\} \\
&= 2\lceil \lg n \rceil \sum_{j=0}^{2\lceil \lg n \rceil - 1} \Pr\{L_j\} + n \sum_{j=2\lceil \lg n \rceil}^{n} \Pr\{L_j\} \\
&< 2\lceil \lg n \rceil \cdot 1 + n \cdot (1/n) = O(\lg n)
\end{aligned}
$$

4. 在线雇用问题：面试并拒绝前 n/e 个，然后录用随后的第一个分数高过前 n/e 的人，有 1/e 的可能性录取到最好的面试者

```
ON-LINE-MAXIMUM(k, n)
1  bestscore = −∞
2  for i = 1 to k
3       if score(i) > bestscore
4            bestscore = score(i)
5  for i = k + 1 to n
6       if score(i) > bestscore
7            return i
8  return n
```

第六章：堆排序

1. 堆是根节点大于左右孩子的数据结构

a 维护最大堆的性质，时间复杂度 0(lgn)

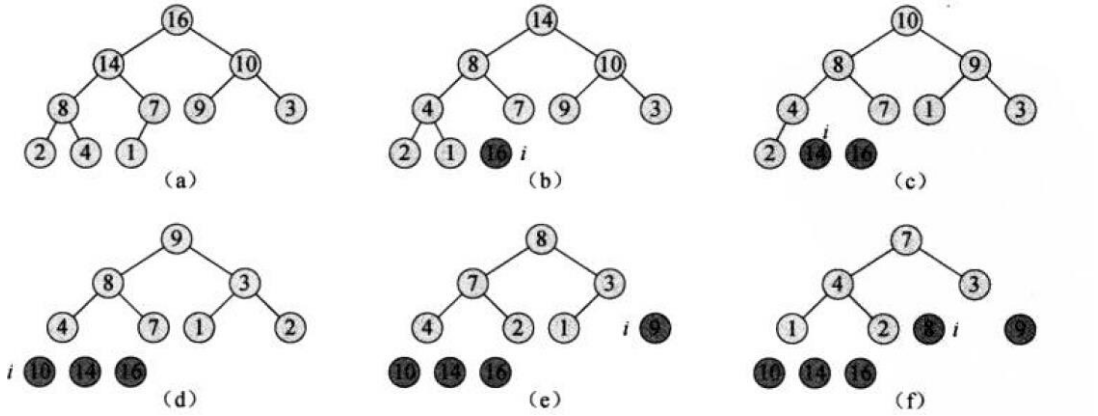
```
MAX-HEAPIFY (A, i )
 1  l= LEFT (i)
 2  r= RIGHT (i)
 3  if l ⩽ A. heap-size and A[l] > A[i]
 4       largest =l
 5  else largest = i
 6  if r ⩽ A. heap-size and A[r] > A[largest]
 7       largest = r
 8  if larget ≠ i
 9       exchange A[i] with A[largest]
10       MAX-HEAPIFY (A, largest )
```

b 建堆算法 0(N)

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

C 堆排序算法 O(nlgn)

```
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```



2．优先队列

优先队列(priority queue)是一种用来维护由一组元素构成的集合 S 的数据结构，其中的每一个元素都有一个相关的值，称为**关键字**(key)。一个**最大优先队列**支持以下操作：

INSERT(S, x)：把元素 x 插入集合 S 中。这一操作等价于 $S=S\cup\{x\}$。

MAXIMUM(S)：返回 S 中具有最大键字的元素。

EXTRACT-MAX(S)：去掉并返回 S 中的具有最大键字的元素。

INCREASE-KEY(S, x, k)：将元素 x 的关键字值增加到 k，这里假设 k 的值不小于 x 的原关键字值。

MAXIMUM(S)：O(1)

EXTRACT-MAX(S)：O(lgN)

INCREASE-KEY：O(lgN)

INSERT():O(lgN)

```
3  A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5      exchange A[i] with A[PARENT(i)]
6      i = PARENT(i)
```

```
MAX-HEAP-INSERT(A, key)
1  A.heap-size = A.heap-size + 1
2  A[A.heap-size] = −∞
3  HEAP-INCREASE-KEY(A, A.heap-size, key)
```

DELETE()：O(lgN)

把最后一个元素换上来，堆大小-1，查看元素是否大于 parent，是：相当于 INCREASE-KEY；否则相当于 MAX-HEAPIFY()

3. k 个链表的归并：取前 k 个元素组成堆，调用 EXTRACT-MAX(S)

第七章 快速排序

1. 时间复杂度非常好 0(nlgn)，常数因子很小，而且可以实现原值排序。

分解：数组 $A[p..r]$被划分为两个(可能为空)子数组 $A[p..q-1]$和 $A[q+1..r]$，使得 $A[p..q-1]$中的每一个元素都小于等于 $A[q]$，而 $A[q]$也小于等于 $A[q+1..r]$中的每个元素。其中，计算下标 q 也是划分过程的一部分。

解决：通过递归调用快速排序，对子数组 $A[p..q-1]$和 $A[q+1..r]$进行排序。

合并：因为子数组都是原址排序的，所以不需要合并操作：数组 $A[p..r]$已经有序。

下面的程序实现快速排序：

```
QUICKSORT(A, p, r)
1  if  p < r
2       q = PARTITION(A, p, r)
3       QUICKSORT(A, p, q-1)
4       QUICKSORT(A, q+ 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p-1
3  for j = p to r-1
4       if A[j]≤x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i+ 1] with A[r]
8  return i + 1
```

实质，i 从 p-1 开始，j 从 p 开始，不断把小于或者等于 x 的元素换到前面，最后把 x 换到 i+1 上。

1.5 最坏运行时间 0(n^2)

2. 快速排序的性质：划分为任意常数项（99:1）都能保证运行时间为 0(nlgn)

3. 随机化快速排序:更好的三数取中法，只影响常数因子

```
RANDOMIZED-PARTITION (A, p, r)
1  i =RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

$$\mathrm{E}[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}$$

在求这个累加和时，可以将变量做个变换($k=j-i$)，并利用公式(A.7)中给出的有关调和级数的界，得到：

$$\mathrm{E}[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}=\sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{k+1}<\sum_{i=1}^{n-1}\sum_{k=1}^{n}\frac{2}{k}=\sum_{i=1}^{n-1}O(\lg n)=O(n\lg n)\quad(7.4)$$

于是，我们可以得出结论：使用 RANDOMIZED-PARTITION，在输入元素互异的情况下，快速排序算法的期望运行时间为 $O(n\lg n)$。

第八章 线性时间排序算法

1. 最坏情况的比较排序

定理 8.1 在最坏情况下，任何比较排序算法都需要做 $\Omega(n \lg n)$ 次比较。

证明 根据前面的讨论，对于一棵每个排列都是一个可达的叶结点的决策树来说，树的高度完全可以被确定。考虑一棵高度为 h、具有 l 个可达叶结点的决策树，它对应一个对 n 个元素所做的比较排序。因为输入数据的 $n!$ 种可能的排列都是叶结点，所以有 $n! \leqslant l$。由于在一棵高为 h 的二叉树中，叶结点的数目不多于 2^h，我们得到：

$$n! \leqslant l \leqslant 2^h$$

对该式两边取对数，有

$$\begin{aligned} h &\geqslant \lg(n!) && \text{（因为 lg 函数是单调递增的）} \\ &= \Omega(n \lg n) && \text{（由公式(3.19)）} \end{aligned}$$

■

2. 计数排序：确定每个数之前有几个

   时间复杂度 O(k+n)

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]]=C[A[j]]+1
 6  //C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i]=C[i]+C[i-1]
 9  //C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]]=A[j]
12      C[A[j]]=C[A[j]]-1
```

3. 基数排序

```
RADIX-SORT(A, d)
1  for i =1 to d
2      use a stable sort to sort array A on digit i
```

引理 8.3 给定 n 个 d 位数，其中每一个数位有 k 个可能的取值。如果 RADIX-SORT 使用的稳定排序方法耗时 $\Theta(n+k)$，那么它就可以在 $\Theta(d(n+k))$ 时间内将这些数排好序。

引理 8.4 给定一个 b 位数和任何正整数 $r \leqslant b$，如果 RADIX-SORT 使用的稳定排序算法对数据取值区间是 0 到 k 的输入进行排序耗时 $\Theta(n+k)$，那么它就可以在 $\Theta((b/r)(n+2^r))$ 时间内将这些数排好序。

把一个二进制数拆成多个 N 进制数排序

4. 稳定排序：计数排序，基数排序，插入排序，归并排序。

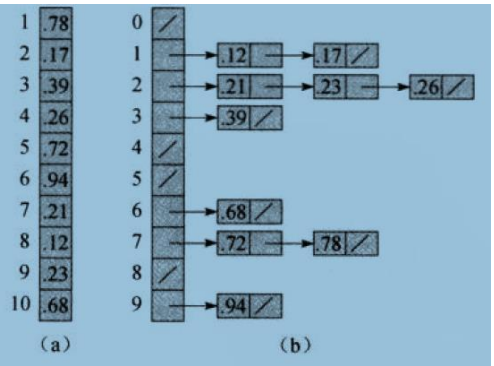
   不稳定：快排，堆排序（以 index 作为第二关键字）

5. 桶排序：假设输入序列均匀，时间复杂度 O(N)

即使输入数据不服从均匀分布，桶排序也仍然可以线性时间内完成。只要输入数据满足下列性质：所有桶的大小的平方和与总的元素数呈线性关系，那么通过公式(8.1)，我们就可以知道：桶排序仍然能在线性时间完成。

的一些基本操作)。

```
BUCKET-SORT(A)
1  n = A.length
2  let B[0..n−1] be a new array
3  for i = 0 to n−1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊nA[i]⌋]
7  for i = 0 to n−1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0],B[1],…,B[n−1] together in order
```



$$E[T(n)] = E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \quad (利$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \quad (利$$

$$\Theta(n) + n \cdot O(2 - 1/n) = \Theta(n)$$

$$E[n_i^2] = 2 - 1/n$$

第九章 顺序统计量

1. 成对处理元素，则 3n/2 时间内就可以同时找到最小值和最大值
2. 寻找第二小元素：期望 0(N)

```
RANDOMIZED-SELECT (A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q−p+1
5  if i == k          // the pivot value is the answer
6       return A[q]
7  else if i < k
8       return RANDOMIZED-SELECT(A, p, q−1, i)
9  else return RANDOMIZED-SELECT(A, q+1, r, i−k)
```

第十章 基本数据结构

1. 栈和队列

栈和队列都是动态集合，且在其上进行 DELETE 操作所移除的元素是预先设定的。在**栈**(stack)中，被删除的是最近插入的元素：栈实现的是一种**后进先出**(last-in，first-out，LIFO)策略。类似地，在队列(queue)中，被删去的总是在集合中存在时间最长的那个元素：队列实现的是一种**先进先出**(first-in，first-out，FIFO)策略。在计算机上实现栈和队列有几种有效方式。本

2. 队的实现

```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
1  x = Q[Q.head]
2  if Q.head = Q.length
3        Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

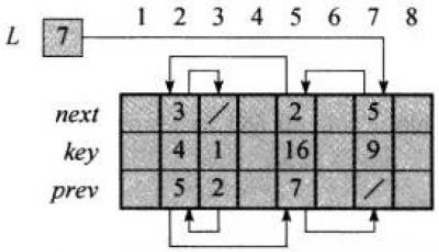
```
LIST-DELETE(L, x)
1  if x.prev ≠ NIL
2        x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5        x.next.prev = x.prev
```

3. 两个栈实现队列/两个队列实现栈：来回倒
4. 链表：插入/删除 0(1)

5. 有哨兵的双向循环链表：简化常数因子

哨兵(sentinel)是一个哑对象，其作用是简化边界条件的处理。例如，假设在链表 L 中设置一个对象 $L.nil$，该对象代表 NIL，但也具有和其他对象相同的各个属性。对于链表代码中出现的每一处对 NIL 的引用，都代之以对哨兵 $L.nil$ 的引用。如图 10-4 所示，这样的调整将一个常规的双向链表转变为一个**有哨兵的双向循环链表**(circular, doubly linked list with a sentinel)，哨兵 $L.nil$ 位于表头和表尾之间。属性 $L.nil.next$ 指向表头，$L.nil.prev$ 指向表尾。类似地，表尾的 $next$ 属性和表头的 $prev$ 属性同时指向 $L.nil$。因为 $L.nil.next$ 指向表头，我们就可以去掉属性 $L.head$，并把对它的引用代替为对 $L.nil.next$ 的引用。图 10-4(a)显示，一个空的链表只由一

6. 没有指针的链式数据结构



)-5 用数组 key、$next$ 和 $prev$ 表示图 10-3(a)中的链表。每一列数组项表示一个单一的对象。数组内存放的指针对应于上方所示的数组下标；箭头给出其形象表示。浅阴影的位置存放的是表内元素。变量 L 存放表头元素的下标

在不支持显式的指针数据类型的编程环境下，我们可以采用同样的策略来实现对象。图 10-6 举例说明了如何用单个数组 A 存储图 10-3(a)和图 10-5 所示的链表。一个对象占用一段连续的子数组 $A[j..k]$，对象中的每个属性对应于从 0 到 $k-j$ 之间的一个偏移量，指向该对象的指针就是下标 j。在图 10-6 中，对应于属性 key、$next$ 和 $prev$ 的偏移量分别为 0、1 和 2。给定一个指针 i，要读取 $i.prev$ 的值，只需在指针的值 i 上加上偏移量 2，所以要读取的是 $A[i+2]$。

```
FREE-OBJECT(x)
1  x. next = free
2  free = x
```

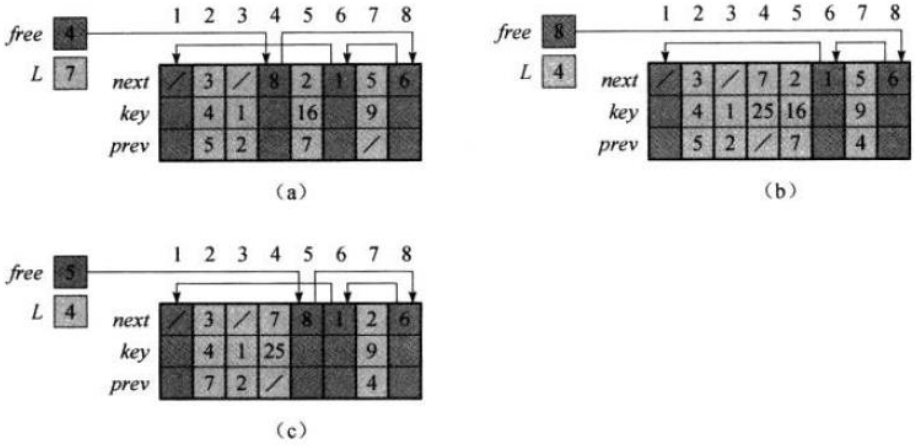


图 10-7　过程 ALLOCATE-OBJECT 和 FREE-OBJECT 的执行结果。(a)图 10-5 中的链表(浅阴影部分)和自由表(深阴影部分)。箭头标示自由表的结构。(b)调用 ALLOCATE-OBJECT()(返回下标 4)、将 $key[4]$设为 25、再调用 LIST-INSERT(L，4)处理的结果。新自由表的头为原自由表中 $next[4]$所指的对象 8。(c)执行 LIST-DELETE(L，5)，然后调用 FREE-OBJECT(5)。对象 5 成为新自由表的表头，对象 8 紧随其后

7. 任意节点树保存法：左孩子右兄弟

所幸的是，有一个巧妙的方法可以用来表示孩子数任意的树。该方法的优势在于，对任意 n 个结点的有根树，只需要 $O(n)$的存储空间。这种**左孩子右兄弟表示法**(left-child，right-sibling representation)如图 10-10 所示。和前述方法类似，每个结点都包含一个父结点指针 p，且 $T.root$ 指向树 T 的根结点。然而，每个结点中不是包含指向每个孩子的指针，而是只有两个指针：

1. $x.left\text{-}child$ 指向结点 x 最左边的孩子结点。
2. $x.right\text{-}sibling$ 指向 x 右侧相邻的兄弟结点。

如果结点 x 没有孩子结点，则 $x.left\text{-}child$ = NIL；如果结点 x 是其父结点的最右孩子，则 $x.right\text{-}sibling$ = NIL。

第十一章 散列表

1. 散列表的 search 时间平均为 O(1),最坏情况为 O(N)
2. 直接寻址表

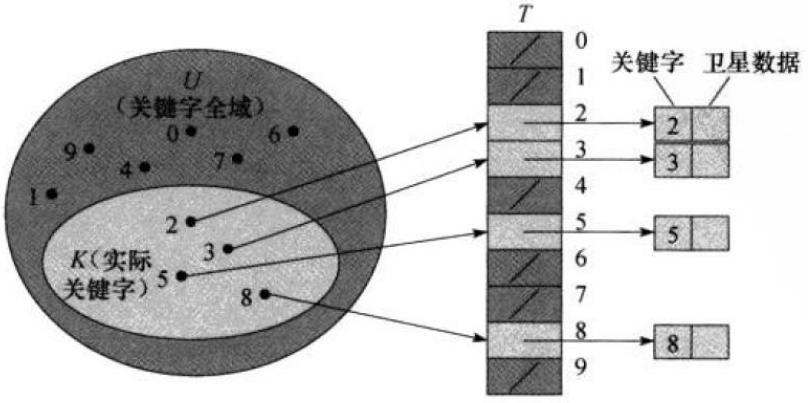


图 11-1　如何用一个直接寻址表 T 来实现动态集合。全域 U={0，1，…，9}中的每个关键字都对应于表中的一个下标值。由实际关键字构成的集合 K={2，3，5，8}决定表中的一些槽，这些槽包含指向元素的指针。而另一些槽包含 NIL，用深阴影表示

3. 给定 40 亿个不相同的整数，判定某个数是否在其中/判断是否有重复：申请 512MB 的 bit 数组，直接寻址表
4. 散列表

的全域 U 映射到**散列表**(hash table) $T[0..m-1]$ 的槽位上：

$$h:U\to\{0,1,\cdots,m-1\}$$

这里散列表的大小 m 一般要比 $|U|$ 小得多。我们可以说一个具有关键字 k 的元素被**散列**到槽 $h(k)$ 上，也可以说 $h(k)$ 是关键字 k 的**散列值**。图 11-2 描述了这个基本方法。散列函数缩小了数组下标的范围，即减小了数组的大小，使其由 $|U|$ 减小为 m。

5. 解决冲突方法一：链接法 插入、删除：最坏 O(1)。查找：最坏 O(N)

在链接法中，把散列到同一槽中的所有元素都放在一个链表中，如图 11-3 所示。槽 j 中有一个指针，它指向存储所有散列到 j 的元素的链表的表头；如果不存在这样的元素，则槽 j 中为 NIL。

6. 简单均匀数列

散列方法的平均性能依赖于所选取的散列函数 h，将所有的关键字集合分布在 m 个槽位上的均匀程度。11.3 节将讨论这些问题，现在我们先假定任何一个给定元素等可能地散列到 m 个槽中的任何一个，且与其他元素被散列到什么位置上无关。我们称这个假设为**简单均匀散列**(simple uniform hashing)。

对于 $j=0,1,\cdots,m-1$，列表 $T[j]$ 的长度用 n_j 表示，于是有

$$n=n_0+n_1+\cdots+n_{m-1} \qquad (11.1)$$

并且 n_j 的期望值为 $E[n_j]=\alpha=n/m$。

定理 11.1 在简单均匀散列的假设下，对于用链接法解决冲突的散列表，一次不成功查找的平均时间为 $\Theta(1+\alpha)$。

平均查找时间也是 O(1+a)，说明链接法插入、删除和查找的平均时间都是 O(1)

7. 除法散列法

散列函数为：

$$h(k)=k \bmod m$$

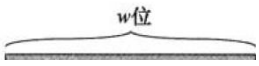
8. 乘法散列法

构造散列函数的乘法散列法包含两个步骤。第一步，用关键字 k 乘上常数 $A(0<A<1)$，并提取 kA 的小数部分。第二步，用 m 乘以这个值，再向下取整。总之，散列函数为：

$$h(k)=\lfloor m(kA \bmod 1)\rfloor$$

这里“$kA \bmod 1$”是取 kA 的小数部分，即 $kA-\lfloor kA\rfloor$。

乘法散列法的一个优点是对 m 的选择不是特别关键，一般选择它为 2 的某个幂次($m=2^p$，p 为某个整数)，这是因为我们可以在大多数计算机上，按下面所示方法较容易地实现散列函数。



$$A\approx(\sqrt{5}-1)/2=0.618\,033\,988\,7\cdots$$

9. 全域散列：universal hashing

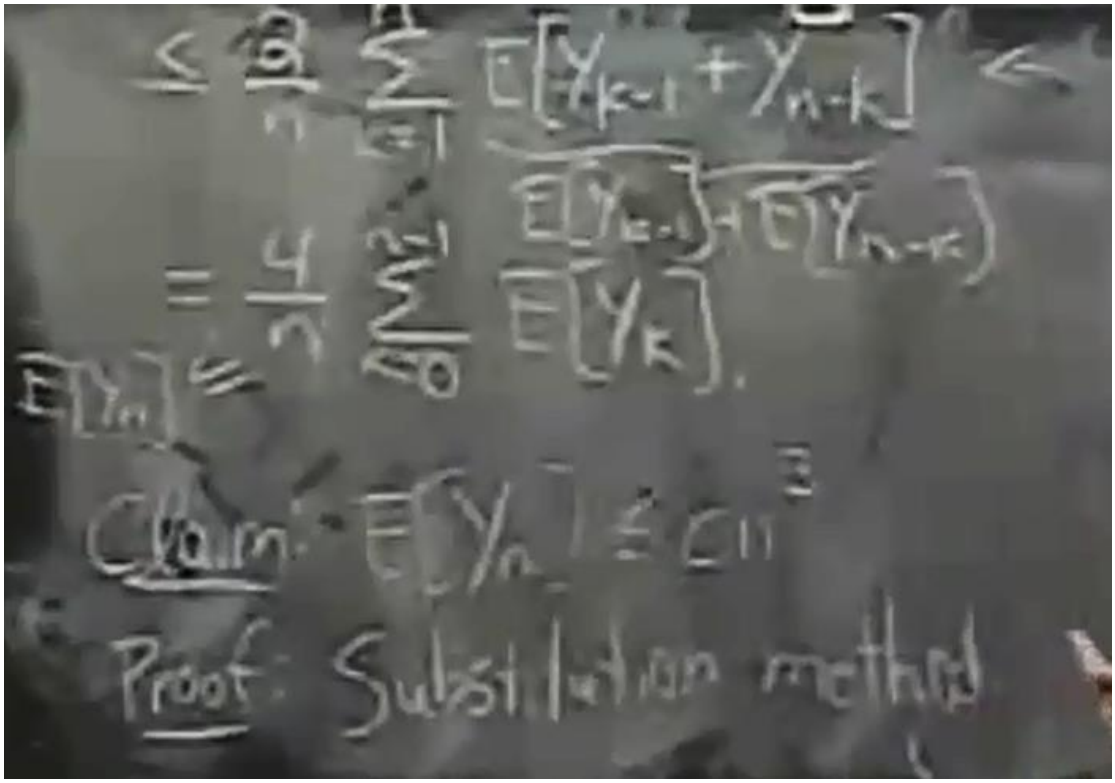
现这种令人恐怖的最坏情况。唯一有效的改进方法是随机地选择散列函数，使之独立于要存储的关键字。这种方法称为**全域散列**(universal hashing)，不管对手选择了怎么样的关键字，其平均性能都很好。

设 $\mathcal{H}$ 为一组有限散列函数，它将给定的关键字全域 U 映射到 $\{0,1,\cdots,m-1\}$ 中。这样的一个函数组称为**全域的**(universal)，如果对每一对不同的关键字 $k,l\in U$，满足 $h(k)=h(l)$ 的散列函数 $h\in\mathcal{H}$ 的个数至多为 $|\mathcal{H}|/m$。换句话说，如果从 $\mathcal{H}$ 中随机地选择一个散列函数，当关键字 $k\neq l$ 时，两者发生冲突的概率不大于 $1/m$，这也正好是从集合 $\{0,1,\cdots,m-1\}$ 中独立地随机选择 $h(k)$ 和 $h(l)$ 时发生冲突的概率。

BSD search

1. 与快速排序基本是一样的

随机生成的二叉搜索树，其高度期望为 logn



平衡二查搜索树：所有操作在 nlogn 完成

红黑树

1. 红黑树首先都是二叉搜索树
2. 红黑树把红节点和他的孩子合并起来，所有的孩子都是一样高的，高度是黑高，并且变成了 2,3,4 树

一棵红黑树是满足下面**红黑性质**的二叉搜索树：

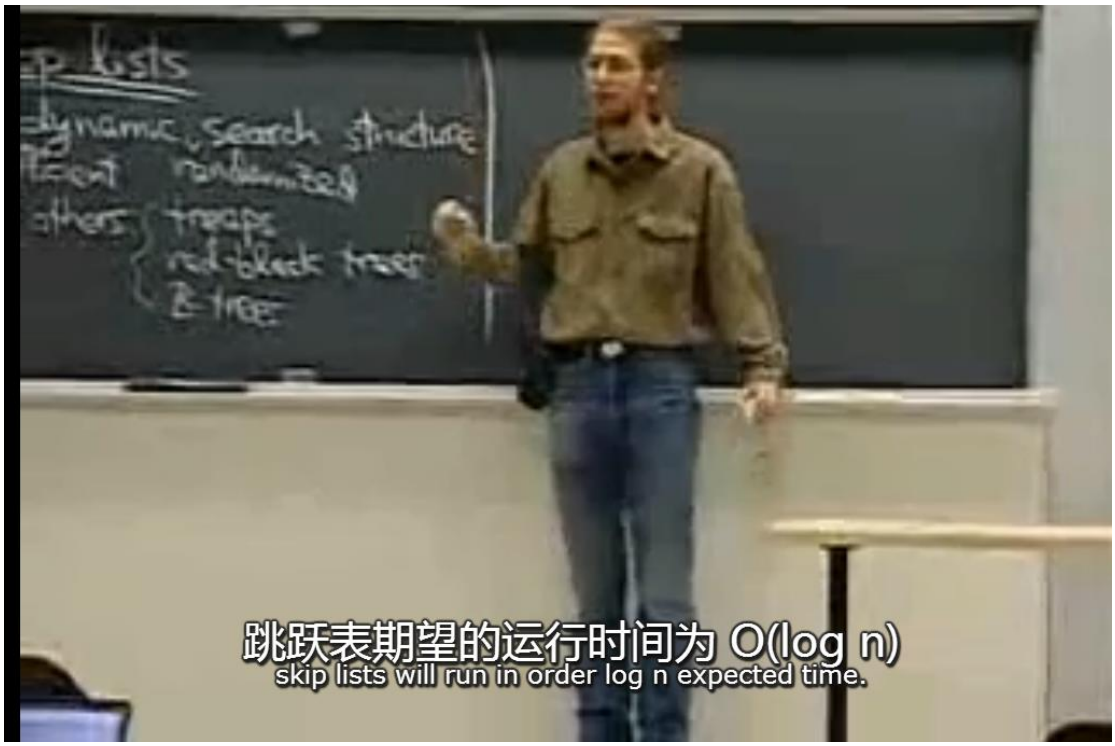
1. 每个结点或是红色的，或是黑色的。
2. 根结点是黑色的。
3. 每个叶结点(NIL)是黑色的。
4. 如果一个结点是红色的，则它的两个子结点都是黑色的。
5. 对每个结点，从该结点到其所有后代叶结点的简单路径上，均包含相同数目的黑色结点。

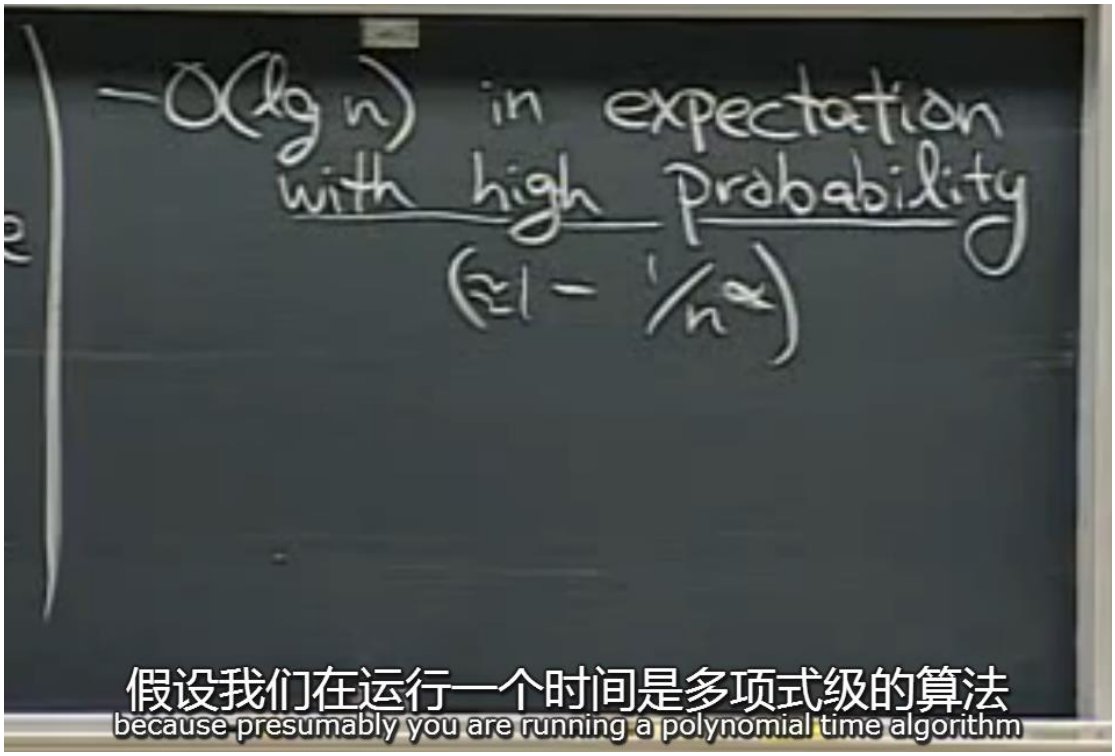
从某个结点 x 出发(不含该结点)到达一个叶结点的任意一条简单路径上的黑色结点个数称为该结点的**黑高**(black-height)，记为 $\mathrm{bh}(x)$。根据性质 5，黑高的概念是明确定义的，因为从该结点出发的所有下降到其叶结点的简单路径的黑结点个数都相同。于是定义红黑树的黑高为其根结点的黑高。

下面的引理说明了为什么红黑树是一种好的搜索树。

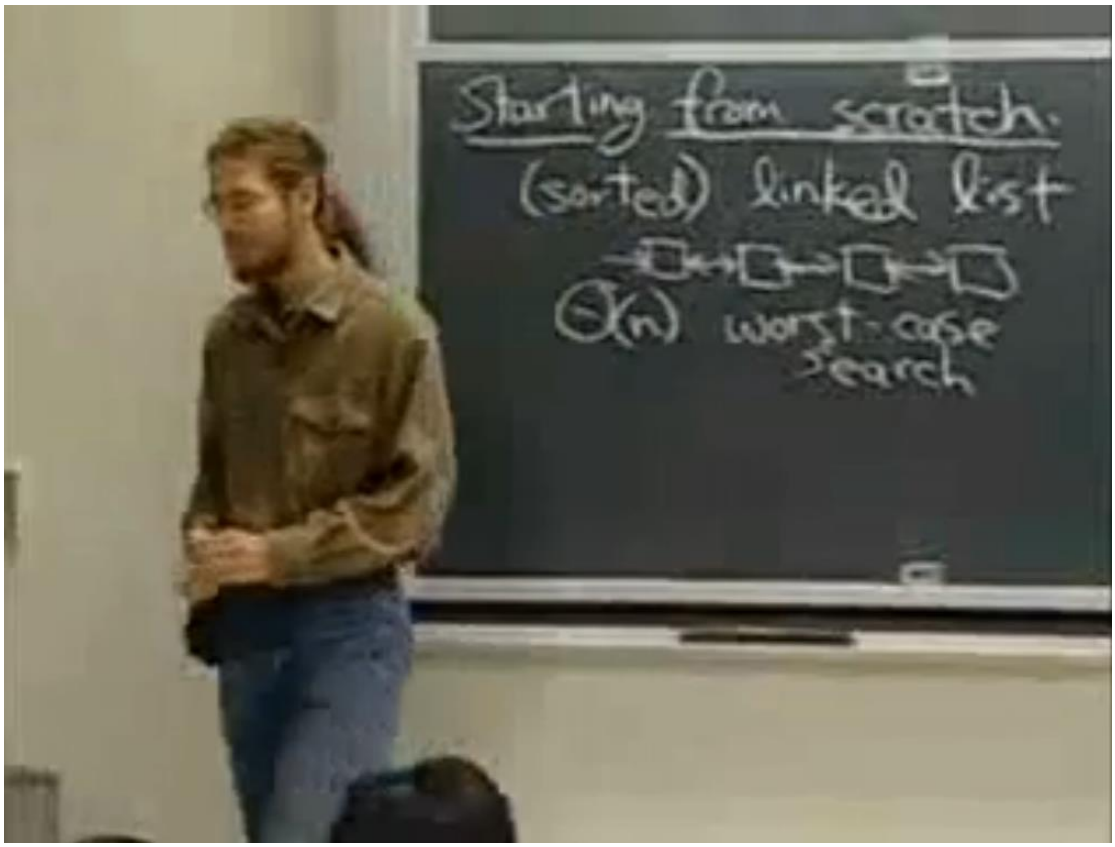
引理 13.1 一棵有 n 个内部结点的红黑树的高度至多为 $2\lg(n+1)$。

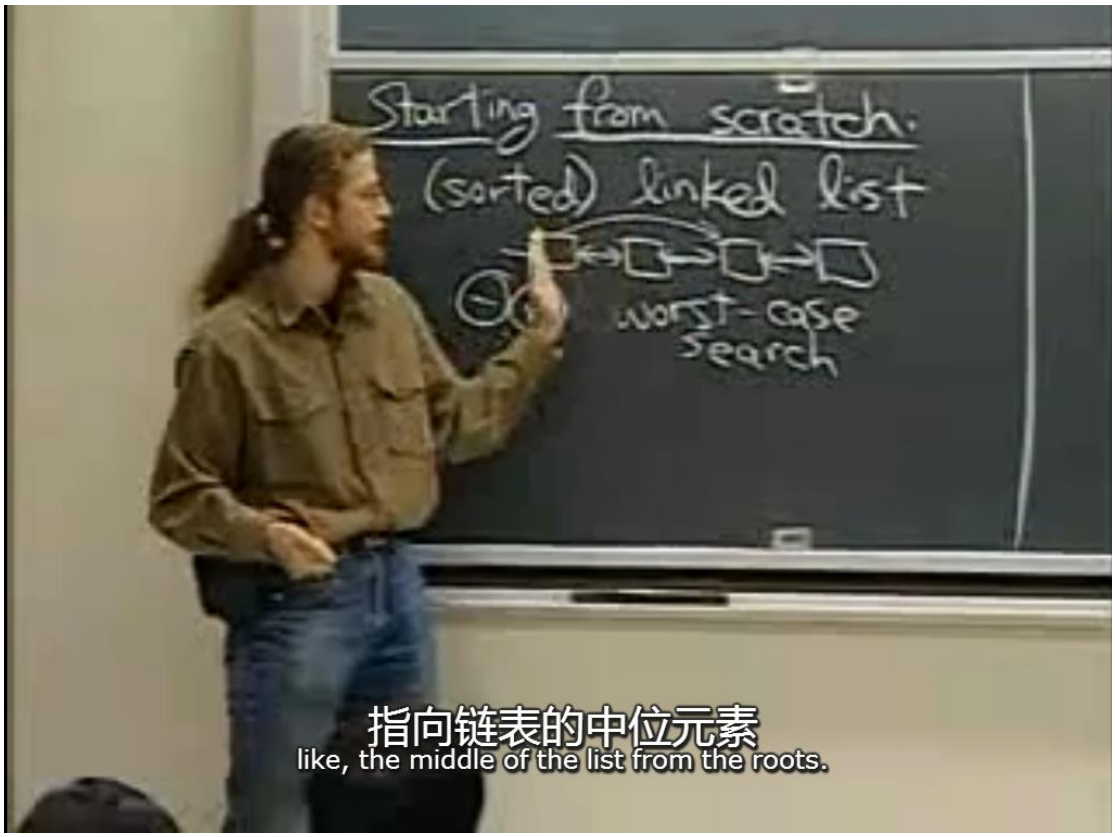
证明 先证明以任一结点 x 为根的子树中至少包含 $2^{\mathrm{bh}(x)}-1$ 个内部结点。要证明这点，对 x 的高度进行归纳。如果 x 的高度为 0，则 x 必为叶结点($T.nil$)，且以 x 为根结点的子树至少包含



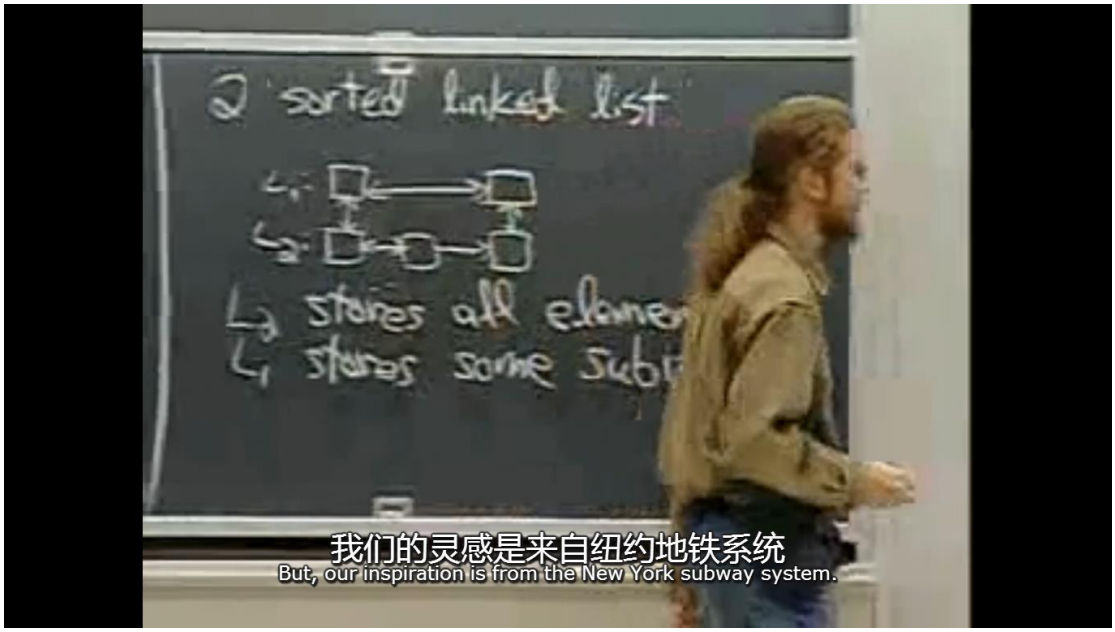


假设我们在运行一个时间是多项式级的算法
because presumably you are running a polynomial time algorithm

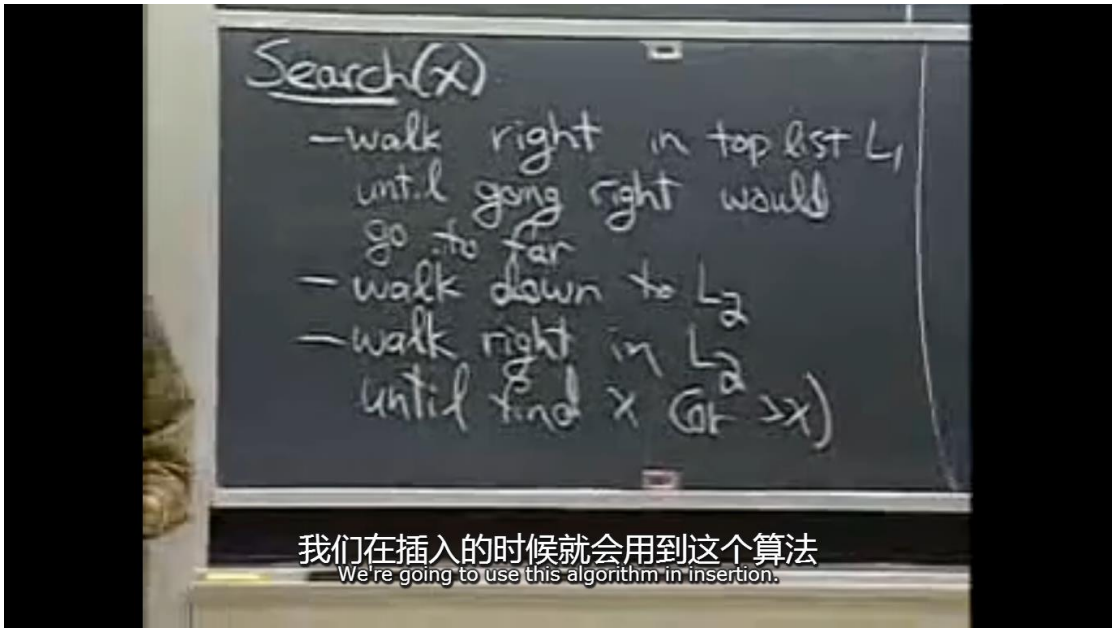




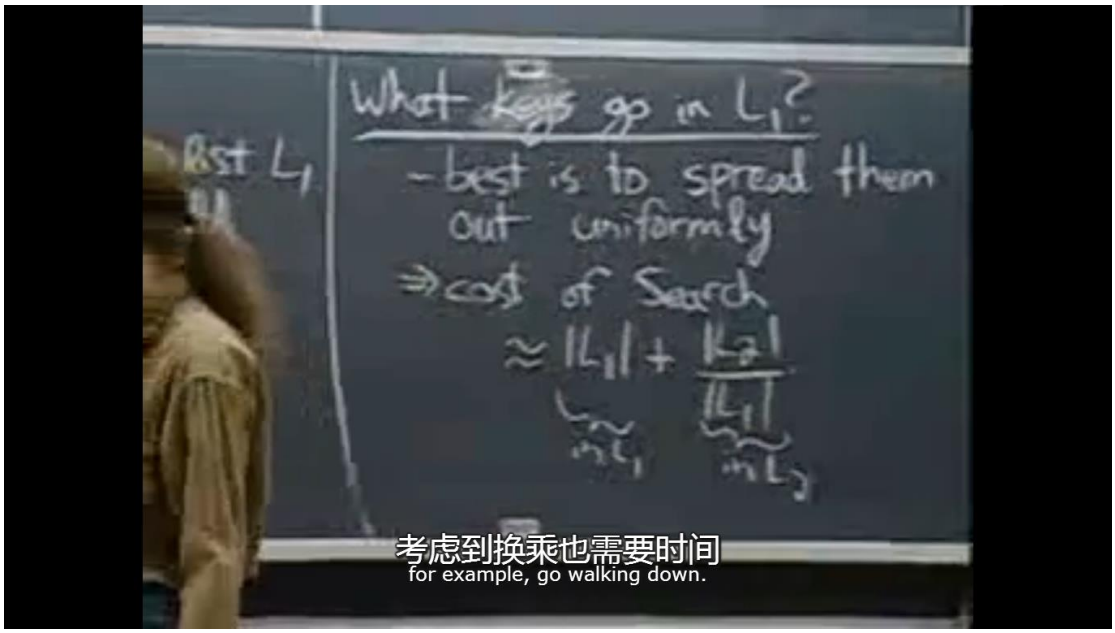
指向链表的中位元素
like, the middle of the list from the roots.



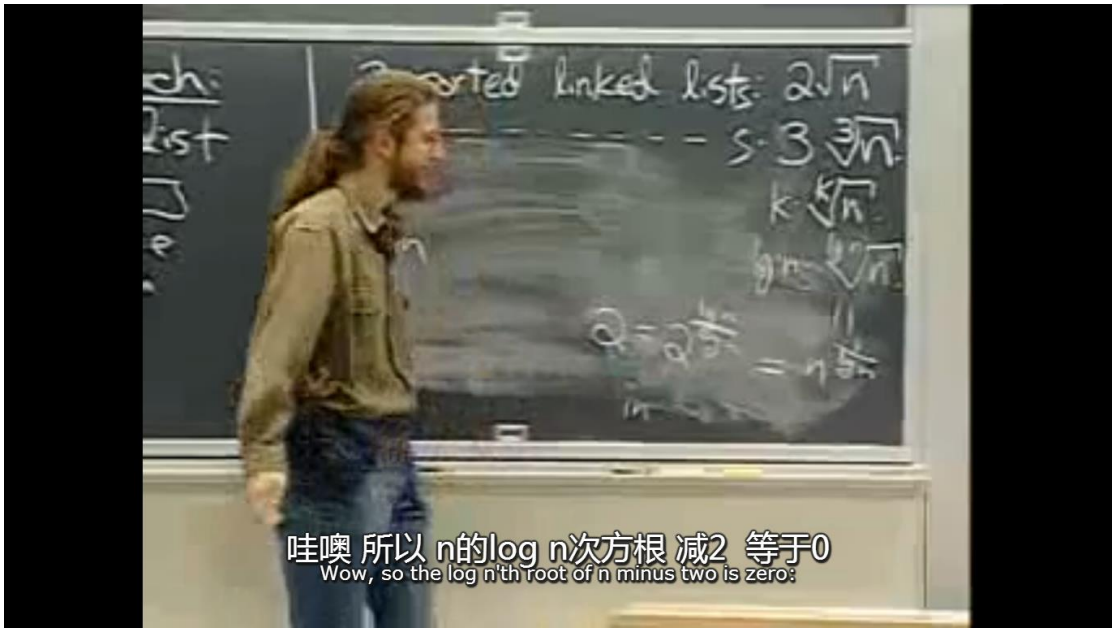
我们的灵感是来自纽约地铁系统
But, our inspiration is from the New York subway system.

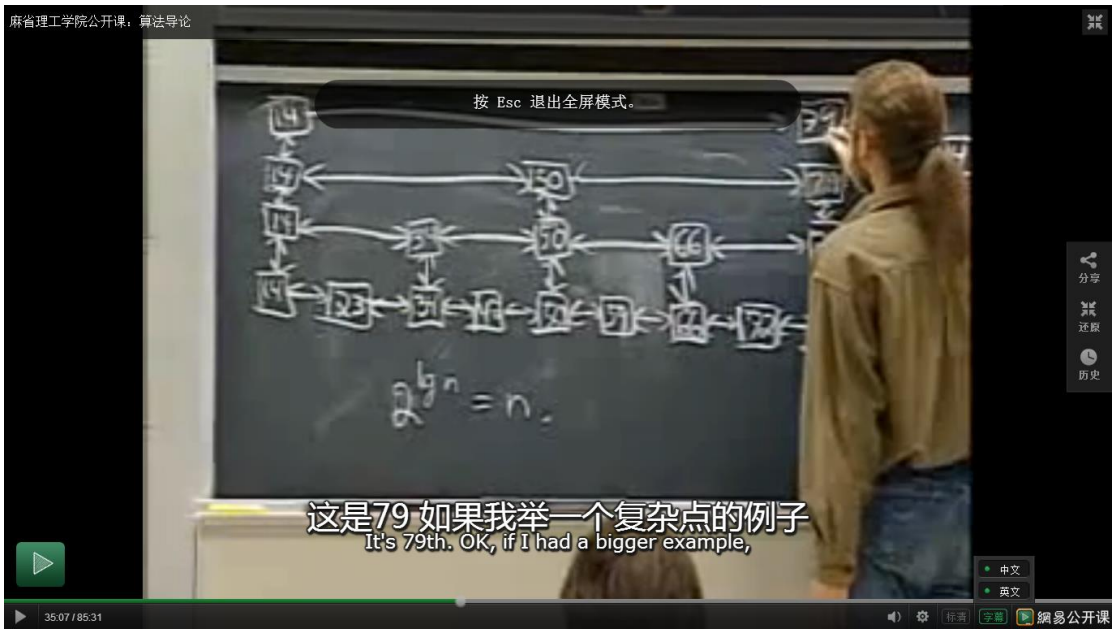


我们在插入的时候就会用到这个算法
We're going to use this algorithm in insertion.



考虑到换乘也需要时间
for example, go walking down.


哇噢 所以 n的log n次方根 减2 等于0
Wow, so the log n'th root of n minus two is zero:


麻省理工学院公开课：算法导论
按 Esc 退出全屏模式。
这是79 如果我举一个复杂点的例子
It's 79th. OK, if I had a bigger example,
分享
还原
历史
中文
英文
35:07 / 85:31
網易公开课