计算机算法设计与分析
习题解答

薛健

工程科学学院

Last Modified: 2016.5.7

主要内容

主要内容

主要内容

1. 习题解答
   - 递归程序设计
   - 元素平均移动次数
   - 多米诺骨牌问题的分治算法
   - 5 元素中位数和排序
   - 数组循环移位
   - 多米诺骨牌问题的动态规划算法
   - 最大相容线段集合问题
   - 邮局位置问题

递归程序设计

Exercise (2)

定义文件 xx.tar.gz 的产生方式如下：

- 以 xx 为文件名的文件通过 tar 和 gzip 打包压缩产生，该文件中以字符串的方式记录了一个非负整数；
- 或者以 xx 为名的目录通过 tar 和 gzip 打包压缩产生，该目录中包含若干 xx.tar.gz。

其中，x ∈ [0，9]。现给定一个根据上述定义生成的文件 00.tar.gz (该文件从课程网站下载)，请确定其中包含的以 xx 为文件名的文件个数以及这些文件中所记录的非负整数之和。

答案：文件数：6895；非负整数之和：34006822。

主要内容

问题及快速排序情况

Exercise (3)

试分析比较快速排序和归并排序在平均情况下元素移动次数。

- 快速排序
  - 每次比较，元素需要移动的可能性是多少？（是 1/2 吗？）
  - 元素移动发生在 Partition 阶段，而每次调用 Partition 时，子序列中元素两两之间都未经过比较，因此可以认为元素排列是随机的，从而 *pivot* 元素为第 k 小元素的可能性为 $1/n$，且 Partition 结束后 *pivot* 元素处于第 k 个位置
  - 每次 *ele* 与 *pivot* 比较后需要移动当且仅当 $ele < pivot$ 且位于序列位置的 $(k, n]$ 区间内或者 $ele \geq pivot$ 且位于序列位置的 $[1, k)$ 区间内
  - 当序列元素随机均匀分布时这两种情况的概率分别为 $\dfrac{n-k}{n-1}$ 和 $\dfrac{k-1}{n-1}$

快速排序情况 (cont.) 和归并排序情况

- 快速排序 (续)
  - 因此每次 Partition 的平均移动次数可计算如下:

$$M_{\text{ave}}(n) = \frac{1}{n}\sum_{k=1}^{n}\left(\frac{n-k}{n-1}\cdot(k-1)+\frac{k-1}{n-1}\cdot(n-k)\right)$$
$$= \frac{2}{n}\sum_{k=1}^{n}\frac{(n-k)(k-1)}{n-1} = \frac{n-2}{3} \approx \frac{n-1}{3}$$

  - 总的平均移动次数约为比较次数的 $1/3$，即大约 $0.462 n \lg n$!
- 归并排序
  - 使用最原始的归并程序，元素移动次数与子序列的元素数目相当
  - 总的移动次数约为 $n \lg n$，超过快速排序的两倍多!

补充讨论

- 归并程序的平均比较次数：

补充讨论

- 归并程序的平均比较次数：
  - m 个元素和 n 个元素归并可能产生的排列数：$C_{m+n}^{m} = C_{m+n}^{n}$

补充讨论

- 归并程序的平均比较次数：
  - m 个元素和 n 个元素归并可能产生的排列数：$C_{m+n}^{m}=C_{m+n}^{n}$
  - 比较次数：$C=m+n-S$，S 为剩余直接拷贝的元素数
  - 则 $S\geq s$ 的概率：

$$q_s=\begin{cases}\dfrac{C_{m+n-s}^{m}+C_{m+n-s}^{n}}{C_{m+n}^{m}}, & 1\leq s\leq m+n;\\ 0, & s>m+n.\end{cases}$$

补充讨论

- 归并程序的平均比较次数：
  - m 个元素和 n 个元素归并可能产生的排列数：$C_{m+n}^{m} = C_{m+n}^{n}$
  - 比较次数：$C = m + n - S$，S 为剩余直接拷贝的元素数
  - 则 $S \geq s$ 的概率：

$$q_s = \begin{cases} \dfrac{C_{m+n-s}^{m} + C_{m+n-s}^{n}}{C_{m+n}^{m}}, & 1 \leq s \leq m+n; \\ 0, & s > m+n. \end{cases}$$

  - 则 S 的均值为：$\mu_{mn} = q_1 + q_2 + \cdots = \dfrac{m}{n+1} + \dfrac{n}{m+1}$

补充讨论

- 归并程序的平均比较次数：
  - m 个元素和 n 个元素归并可能产生的排列数：$C_{m+n}^{m} = C_{m+n}^{n}$
  - 比较次数：$C = m + n - S$，S 为剩余直接拷贝的元素数
  - 则 $S \geq s$ 的概率：

$$q_s = \begin{cases} \dfrac{C_{m+n-s}^{m} + C_{m+n-s}^{n}}{C_{m+n}^{m}}, & 1 \leq s \leq m+n; \\ 0, & s > m+n. \end{cases}$$

  - 则 S 的均值为：$\mu_{mn} = q_1 + q_2 + \cdots = \dfrac{m}{n+1} + \dfrac{n}{m+1}$
  - 平均比较次数 $C_{\text{ave}} = m + n - \mu_{mn}$

补充讨论

- 归并程序的平均比较次数：
    - m 个元素和 n 个元素归并可能产生的排列数：$C_{m+n}^{m} = C_{m+n}^{n}$
    - 比较次数：$C = m + n - S$，S 为剩余直接拷贝的元素数
    - 则 $S \geq s$ 的概率：

$$q_s = \begin{cases} \dfrac{C_{m+n-s}^{m} + C_{m+n-s}^{n}}{C_{m+n}^{m}}, & 1 \leq s \leq m+n; \\ 0, & s > m+n. \end{cases}$$

    - 则 S 的均值为：$\mu_{mn} = q_1 + q_2 + \cdots = \dfrac{m}{n+1} + \dfrac{n}{m+1}$
    - 平均比较次数 $C_{\text{ave}} = m + n - \mu_{mn}$
    - 更深入的分析请参考：
      Donald E. Knuth. The Art of Computer Programming (Volume 3: Section 5.2.4)

主要内容

问题描述

Exercise (4)

现有 n 块“多米诺骨牌” $s_1, s_2, \cdots, s_n$ 水平放成一排，每块骨牌 s_i 包含左右两个部分，每个部分赋予一个非负整数值，如下图所示为包含 6 块骨牌的序列。骨牌可做 180 度旋转，使得原来在左边的值变到右边，而原来在右边的值移到左边，假设不论 s_i 如何旋转，$L[i]$ 总是存储 s_i 左边的值，$R[i]$ 总是存储 s_i 右边的值，$W[i]$ 用于存储 s_i 的状态：当 $L[i] \leq R[i]$ 时记为 0，否则记为 1，试采用分治法设计算法求 $\sum_{i=1}^{n-1} R[i] \cdot L[i+1]$ 的最大值，以及当取得最大值时每个骨牌的状态。

5 | 8	4 | 2	9 | 6	7 | 7	3 | 9	11 | 10
s_1	s_2	s_3	s_4	s_5	s_6

问题描述

Exercise (4)

现有 n 块“多米诺骨牌” $s_1, s_2, \cdots, s_n$ 水平放成一排，每块骨牌 s_i 包含左右两个部分，每个部分赋予一个非负整数值，如下图所示为包含 6 块骨牌的序列。骨牌可做 180 度旋转，使得原来在左边的值变到右边，而原来在右边的值移到左边，假设不论 s_i 如何旋转，$L[i]$ 总是存储 s_i 左边的值，$R[i]$ 总是存储 s_i 右边的值，$W[i]$ 用于存储 s_i 的状态：
当 $L[i] \leq R[i]$ 时记为 0，否则记为 1，试采用分治法设计算法
求 $\sum_{i=1}^{n-1} R[i] \cdot L[i+1]$ 的最大值，以及当取得最大值时每个骨牌的状态。

s_1	s_2	s_3	s_4	s_5	s_6
5 | 8	4 | 2	9 | 6	7 | 7	3 | 9	11 | 10

解题思路：

5 | 8	4 | 2	9 | 6	7 | 7	3 | 9	11 | 10

问题描述

Exercise (4)

现有 n 块“多米诺骨牌” $s_1, s_2, \cdots, s_n$ 水平放成一排，每块骨牌 s_i 包含左右两个部分，每个部分赋予一个非负整数值，如下图所示为包含 6 块骨牌的序列。骨牌可做 180 度旋转，使得原来在左边的值变到右边，而原来在右边的值移到左边，假设不论 s_i 如何旋转，$L[i]$ 总是存储 s_i 左边的值，$R[i]$ 总是存储 s_i 右边的值，$W[i]$ 用于存储 s_i 的状态：
当 $L[i] \leq R[i]$ 时记为 0，否则记为 1，试采用分治法设计算法
求 $\sum_{i=1}^{n-1} R[i] \cdot L[i+1]$ 的最大值，以及当取得最大值时每个骨牌的状态。

5 | 8	4 | 2	9 | 6	7 | 7	3 | 9	11 | 10
s_1	s_2	s_3	s_4	s_5	s_6

解题思路：

$$\overbrace{\boxed{0}\quad \boxed{5\,|\,8}\quad \boxed{4\,|\,2}\quad \boxed{9\,|}}^{S_1}\overbrace{\boxed{|\,6}\quad \boxed{7\,|\,7}\quad \boxed{3\,|\,9}\quad \boxed{11\,|\,10}\quad \boxed{0}}^{S_2}$$

算法描述

Algorithm StoneLargest$(L[], R[], W[], f, l)$

if $f < l$ **then**
  $mid \leftarrow (f + l)/2$;
  $L1 \leftarrow$ StoneLargest$(L, R, W, f, mid - 1)$;
  $R1 \leftarrow$ StoneLargest$(L, R, W, mid + 1, l)$;
  $(Lt, Rt, Wt) \leftarrow (L[f..l], R[f..l], W[f..l])$;
  Swap$(L[mid], R[mid])$; $W[mid] \leftarrow -W[mid]$;
  $L2 \leftarrow$ StoneLargest$(L, R, W, f, mid - 1)$;
  $R2 \leftarrow$ StoneLargest$(L, R, W, mid + 1, l)$;
  if $L1 + R1 < L2 + R2$ **then** $m \leftarrow L2 + R2$;
  else $(L[f..l], R[f..l], W[f..l]) \leftarrow (Lt, Rt, Wt)$; $m \leftarrow L1 + R1$;
else if $f == l$ **then**
  $m \leftarrow m1 \leftarrow R[f - 1] * L[f] + R[f] * L[f + 1]$;
  $m2 \leftarrow R[f - 1] * R[f] + L[f] * L[f + 1]$;
  if $m1 < m2$ **then**
    Swap$(L[f], R[f])$; $W[f] \leftarrow -W[f]$; $m \leftarrow m2$;
  end
else $m \leftarrow R[f - 1] * L[f]$;
return m;

算法描述

Algorithm StoneLargest$(L[], R[], W[], f, l)$

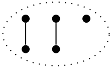
if $f < l$ **then**
  $mid \leftarrow (f + l)/2$;
  $L1 \leftarrow$ StoneLargest$(L, R, W, f, mid - 1)$;
  $R1 \leftarrow$ StoneLargest$(L, R, W, mid + 1, l)$;
  $(Lt, Rt, Wt) \leftarrow (L[f..l], R[f..l], W[f..l])$;
  Swap$(L[mid], R[mid])$; $W[mid] \leftarrow -W[mid]$;
  $L2 \leftarrow$ StoneLargest$(L, R, W, f, mid - 1)$;
  $R2 \leftarrow$ StoneLargest$(L, R, W, mid + 1, l)$;
  if $L1 + R1 < L2 + R2$ **then** $m \leftarrow L2 + R2$;
  else $(L[f..l], R[f..l], W[f..l]) \leftarrow (Lt, Rt, Wt)$; $m \leftarrow L1 + R1$;
else if $f == l$ **then**
  $m \leftarrow m1 \leftarrow R[f-1] * L[f] + R[f] * L[f+1]$;
  $m2 \leftarrow R[f-1] * R[f] + L[f] * L[f+1]$;
  if $m1 < m2$ **then**
    Swap$(L[f], R[f])$; $W[f] \leftarrow -W[f]$; $m \leftarrow m2$;
  end
else $m \leftarrow R[f-1] * L[f]$;
return m;

$$W(n) = 4W(n/2) + \Theta(1) \Rightarrow W(n) \in \Theta(n^2)$$

主要内容

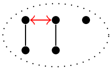
习题 5(a)：最多用 6 次比较找 5 个元素的中位数

第 3 次比较



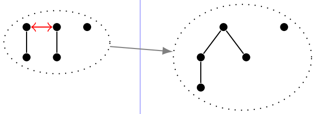
习题 5(a)：最多用 6 次比较找 5 个元素的中位数

第 3 次比较



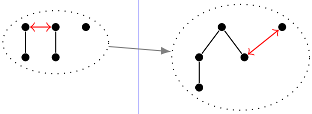
习题 5(a)：最多用 6 次比较找 5 个元素的中位数

第 3 次比较 第 4 次比较

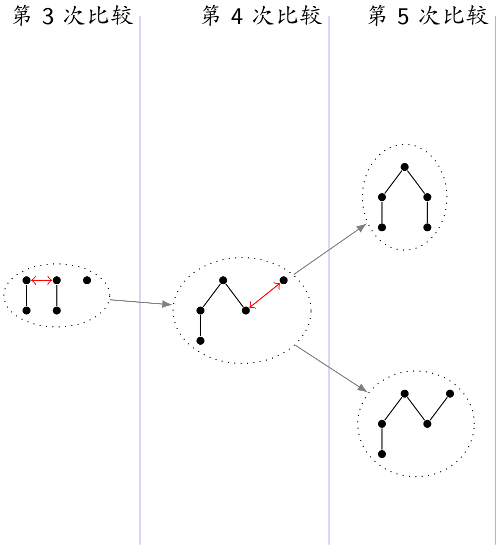


习题 5(a)：最多用 6 次比较找 5 个元素的中位数

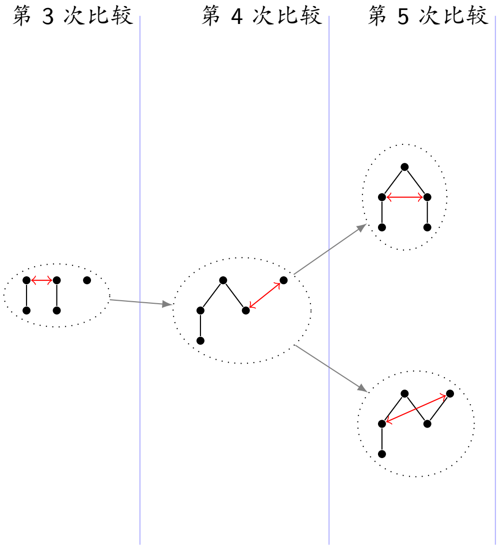
第 3 次比较 第 4 次比较



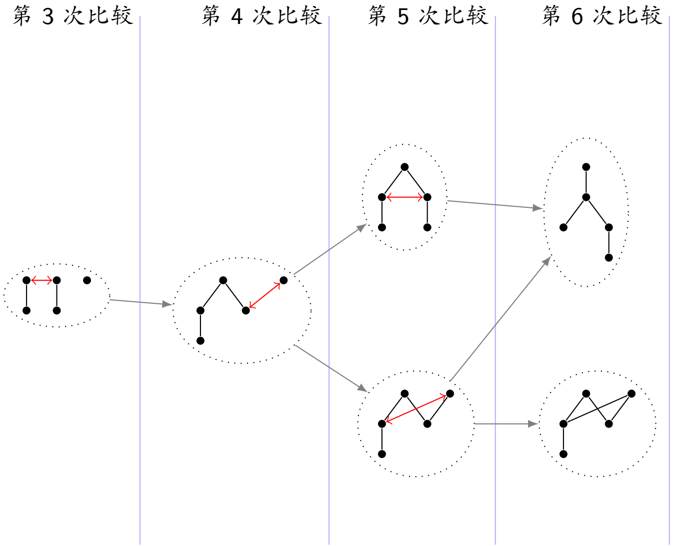
习题 5(a)：最多用 6 次比较找 5 个元素的中位数



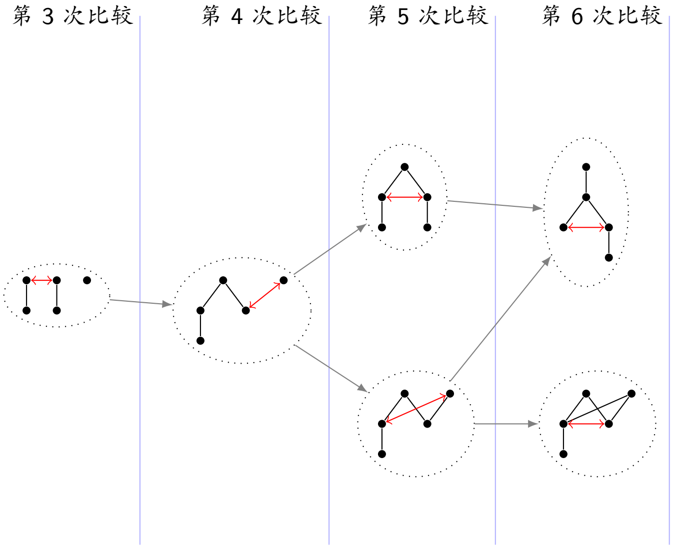
习题 5(a)：最多用 6 次比较找 5 个元素的中位数



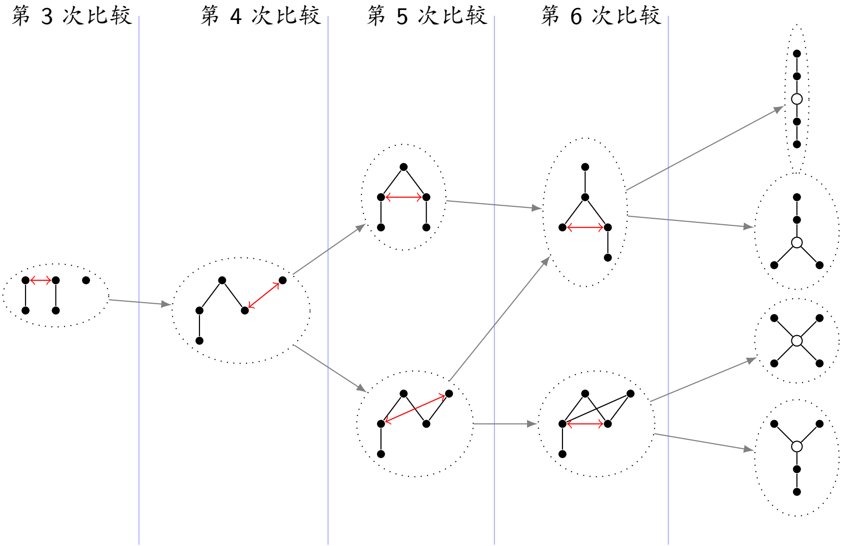
习题 5(a)：最多用 6 次比较找 5 个元素的中位数



习题 5(a)：最多用 6 次比较找 5 个元素的中位数

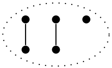


习题 5(a)：最多用 6 次比较找 5 个元素的中位数



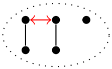
习题 5(b)：最多用 7 次比较对 5 个元素排序

第 3 次比较



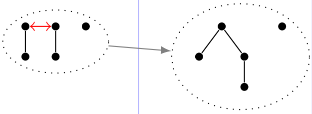
习题 5(b)：最多用 7 次比较对 5 个元素排序

第 3 次比较



习题 5(b)：最多用 7 次比较对 5 个元素排序

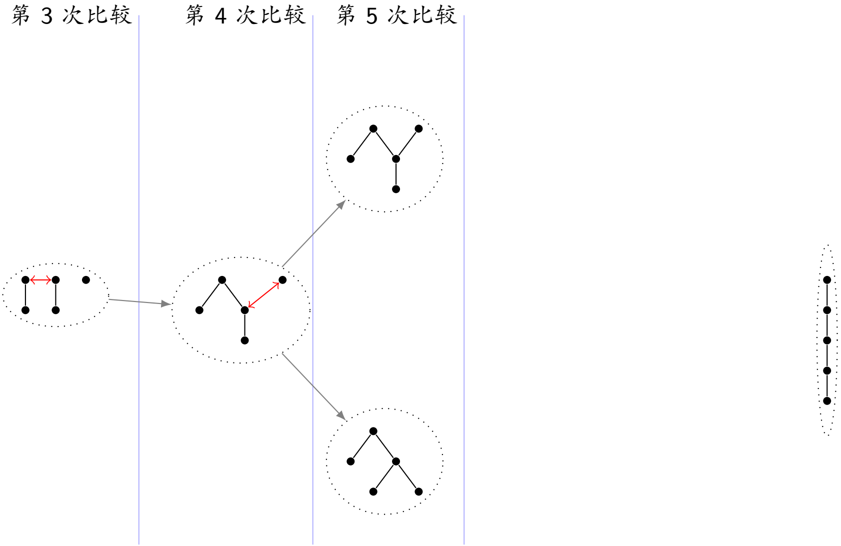
第 3 次比较　第 4 次比较



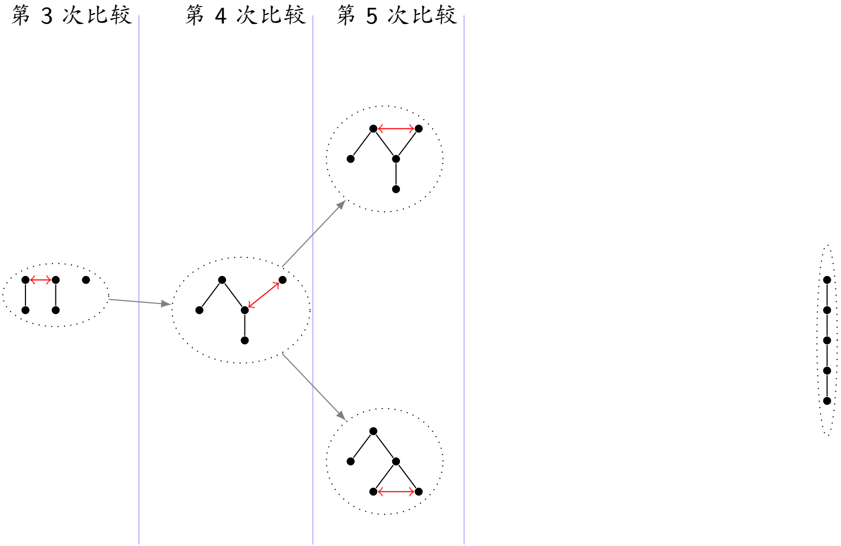
习题 5(b)：最多用 7 次比较对 5 个元素排序

第 3 次比较 | 第 4 次比较

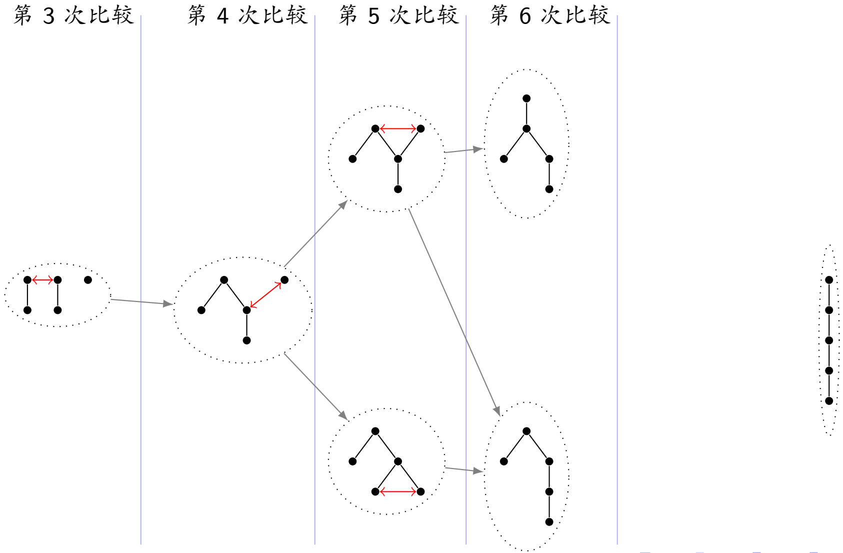
习题 5(b)：最多用 7 次比较对 5 个元素排序



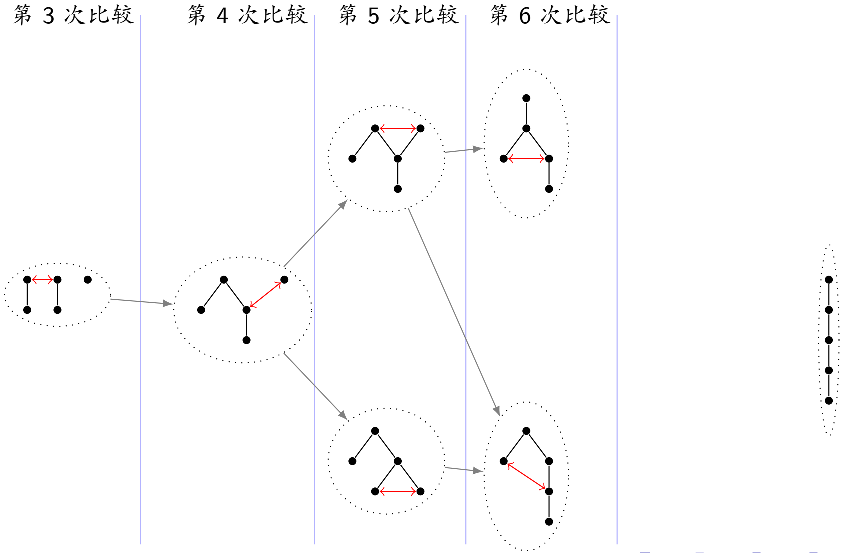
习题 5(b)：最多用 7 次比较对 5 个元素排序



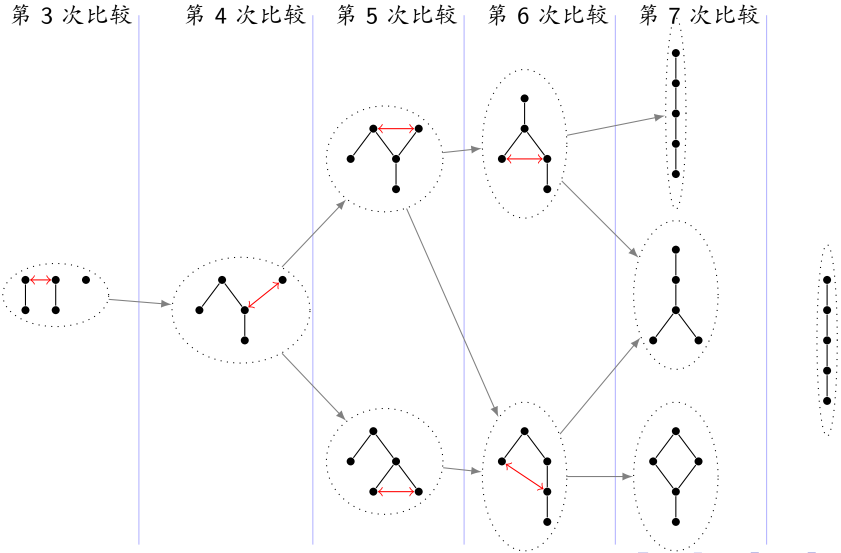
习题 5(b)：最多用 7 次比较对 5 个元素排序



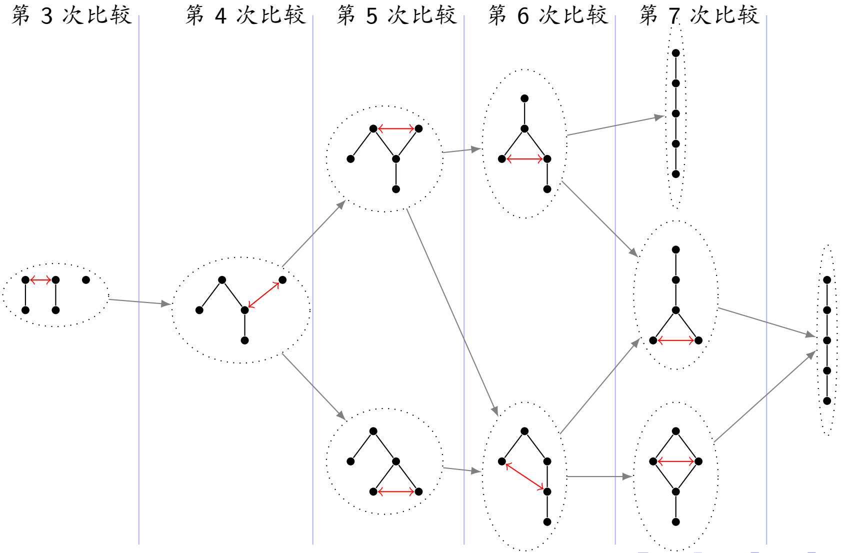
习题 5(b)：最多用 7 次比较对 5 个元素排序



习题 5(b)：最多用 7 次比较对 5 个元素排序



习题 5(b)：最多用 7 次比较对 5 个元素排序



主要内容

问题描述及简单算法

Exercise (1)

已知一个长度为 n 的数组和一个正整数 k，并且最多只能使用一个用于交换的附加空间单元，试设计算法得到原数组循环右移 k 次的结果并分析算法的时间复杂度。

问题描述及简单算法

Exercise (1)

已知一个长度为 n 的数组和一个正整数 k，并且最多只能使用一个用于交换的附加空间单元，试设计算法得到原数组循环右移 k 次的结果并分析算法的时间复杂度。

- 常见算法设计：

Algorithm ShiftRight($A[], n, k$)

```
k ← k mod n;
for i ← 1 to k do
    temp ← A[n − 1];
    for j ← n − 1 to 1 do
        A[j] ← A[j − 1];
    end
    A[0] ← temp;
end
```

问题描述及简单算法

Exercise (1)

已知一个长度为 n 的数组和一个正整数 k，并且最多只能使用一个用于交换的附加空间单元，试设计算法得到原数组循环右移 k 次的结果并分析算法的时间复杂度。

- 常见算法设计：

Algorithm `ShiftRight`$(A[], n, k)$

```
k ← k mod n;
for i ← 1 to k do
    temp ← A[n − 1];
    for j ← n − 1 to 1 do
        A[j] ← A[j − 1];
    end
    A[0] ← temp;
end
```

- 当 $k \bmod n > n/2$ 时可以转换为循环左移 $n - k \bmod n$ 次
- 时间复杂度 $W(n) \in \Theta(kn) \Rightarrow \Theta(n^2)$

线性复杂度算法

1. 直接把元素放到右移 k 以后的位置，这样每个元素只需要移动一次，可保证时间复杂度为 $\Theta(n)$

线性复杂度算法

1. 直接把元素放到右移 k 以后的位置，这样每个元素只需要移动一次，可保证时间复杂度为 $\Theta(n)$

Algorithm `ShiftRightFast`$(A[], n, k)$

```
if n = 1 then return;
d ← GCD(n, k);
m ← n div d;
for i ← 0 to d − 1 do
    temp ← A[(i + (m − 1) ∗ k) mod n];
    for j ← m − 1 down to 1 do
        curr ← (i + j ∗ k) mod n;
        prev ← (i + (j − 1) ∗ k) mod n;
        A[curr] ← A[prev];
    end
    A[i mod n] ← temp;
end
```

线性复杂度算法 (cont.)

2. 将原数组分割为两部分 AB，其中 B 的长度为 $k \bmod n$，那么最终我们需要的结果就是 BA，要得到这一结果，可用下面的思路：先将 A 逆序，再将 B 逆序，再将整个数组逆序，即可得到 BA
   - 复杂度：$W(n) = A(n) = 2n$
   - C++ STL 中 `std::rotate` 所使用的算法

线性复杂度算法 (cont.)

2. 将原数组分割为两部分 AB，其中 B 的长度为 $k \bmod n$，那么最终我们需要的结果就是 BA，要得到这一结果，可用下面的思路：先将 A 逆序，再将 B 逆序，再将整个数组逆序，即可得到 BA
   - 复杂度：$W(n) = A(n) = 2n$
   - C++ STL 中 `std::rotate` 所使用的算法

3. 当 $k \bmod n < n/2$ 时，可将原数组分割为 $A_l A_r B$，其中 A_l 的长度与 B 的长度相同均为 $k \bmod n$，则需要的最终结果是 $BA_l A_r$，该结果可通过两步得到，先将 A_l 与 B 交换，再将 A_l 与 A_r 交换位置 (注意这时二者长度不同)；若 $k \bmod n \geq n/2$，则可将原问题转化为循环左移 $n - k \bmod n$ 次，思路一样。
   - 关键是长度不同的两个数组如何交换位置
   - 复杂度可以达到 $\Theta(n)$
   - 分治法？

主要内容

问题描述

Exercise (6)

现有 n 块“多米诺骨牌” $s_1, s_2, \cdots, s_n$ 水平放成一排，每块骨牌 s_i 包含左右两个部分，每个部分赋予一个非负整数值，如下图所示为包含 6 块骨牌的序列。骨牌可做 180 度旋转，使得原来在左边的值变到右边，而原来在右边的值移到左边，假设不论 s_i 如何旋转，$L[i]$ 总是存储 s_i 左边的值，$R[i]$ 总是存储 s_i 右边的值，$W[i]$ 用于存储 s_i 的状态：当 $L[i] \le R[i]$ 时记为 0，否则记为 1，试设计时间复杂度为 $O(n)$ 的动态规划算法求 $\sum_{i=1}^{n-1} R[i] \cdot L[i+1]$ 的最大值，以及当取得最大值时每个骨牌的状态。

s_1	s_2	s_3	s_4	s_5	s_6
5 | 8	4 | 2	9 | 6	7 | 7	3 | 9	11 | 10

动态规划算法设计

- 问题分割：
  - 子问题 (s, i)：当第 i 块骨牌状态为 s 时求 $\max(\sum_{k=1}^{i-1} R[k]L[k+1])$
  - 用二维数组 M 记录子问题结果，显然 M 的大小为 $2 \times n$
  - 原问题求解目标：$\max(M[0, n], M[1, n])$
- 是否具有最优子结构性质：
  - 设 $(L[1..n], R[1..n])$ 为某一个最优解状态，若第 n 块骨牌状态确定，则需在 $L[n]$ 确定的情况下求 $(\sum_{k=1}^{n-2} R'[k]L'[k+1]) + R'[n-1]L[n]$ 的最大值，可以肯定 $(L'[1..n-1], R'[1..n-1]) = (L[1..n-1], R[1..n-1])$，否则用 $(L'[1..n-1], R'[1..n-1])$ 代替 $(L[1..n-1], R[1..n-1])$ 将得到一个更优的解，与假设矛盾
- 建立子问题求解的递推方程：
  - 设第 i 块骨牌 s_i 上的两个数为 a_i, b_i，且 $a_i \leq b_i$，则：

$$M[0,1] = M[1,1] = 0$$
$$M[0, i+1] = \max\{(M[0,i] + b_i \cdot a_{i+1}), (M[1,i] + a_i \cdot a_{i+1})\}$$
$$M[1, i+1] = \max\{(M[0,i] + b_i \cdot b_{i+1}), (M[1,i] + a_i \cdot b_{i+1})\}$$

算法描述

Algorithm `StoneLargest`$(L[], R[], W[])$

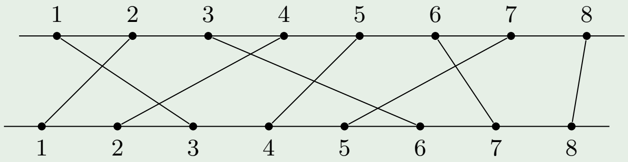
$M[0,1] \leftarrow M[1,1] \leftarrow 0$;
for $i \leftarrow 1$ **to** $n-1$ **do**
    $(a_i, b_i) \leftarrow (W[i]=0)\ ?\ (L[i], R[i])\ :\ (R[i], L[i])$;
    $(a_{i+1}, b_{i+1}) \leftarrow (W[i+1]=0)\ ?\ (L[i+1], R[i+1])\ :\ (R[i+1], L[i+1])$;
    if $(M[0,i] + b_i \cdot a_{i+1}) > (M[1,i] + a_i \cdot a_{i+1})$ **then**
        $M[0,i+1] \leftarrow M[0,i] + b_i \cdot a_{i+1}$; $P[0,i+1] \leftarrow 0$;
    else $M[0,i+1] \leftarrow M[1,i] + a_i \cdot a_{i+1}$; $P[0,i+1] \leftarrow 1$;
    if $(M[0,i] + b_i \cdot b_{i+1}) > (M[1,i] + a_i \cdot b_{i+1})$ **then**
        $M[1,i+1] \leftarrow M[0,i] + b_i \cdot b_{i+1}$; $P[1,i+1] \leftarrow 0$;
    else $M[1,i+1] \leftarrow M[1,i] + a_i \cdot b_{i+1}$; $P[1,i+1] \leftarrow 1$;
end
if $M[0,n] > M[1,n]$ **then** $M \leftarrow M[0,n]$; $W[n] \leftarrow 0$;
else $M \leftarrow M[1,n]$; $W[n] \leftarrow 1$;
for $i \leftarrow n$ **down to** 2 **do**
    $W[i-1] \leftarrow W[i] = 0\ ?\ P[0,i]\ :\ P[1,i]$;
end
return $(W[], M)$;

主要内容

问题描述

Exercise (7)

在两条相互平行的直线上分别有按顺序排列的 n 个点，标有 $1, 2, \cdots, n$，如下图所示，上面直线上的每一点 i 分别与下面直线上唯一一点 $\pi(i)$ 相连，反之亦然，也就是说需要 n 条线段 $(i, \pi(i))$ 来连接这 n 对点。



其中，对于任意两条线段 $(i, \pi(i))$ 和 $(j, \pi(j))$，若 $i < j$ 且 $\pi(i) > \pi(j)$，或者 $i > j$ 且 $\pi(i) < \pi(j)$，则这两条线段必然相交。不满足上述条件，即不相交的线段称为相容线段。试设计一个时间复杂度为 $O(n^2)$ 的动态规划算法找到这 n 条线段中的最大相容线段集合，即该集合中线段互不相交且线段条数最多。例如上图中 $\{(1,3),(3,6),(6,7),(8,8)\}$ 即为一个相容线段集合 (但不一定是最大相容线段集合)。

问题分析

- 可以发现在每个相容线段集合 $\{(i_1, \pi(i_1)), (i_2, \pi(i_2)), \cdots, (i_k, \pi(i_k))\}$ 中，若对 i_k 排序，即 $i_1 < i_2 < \cdots < i_k$，则必有 $\pi(i_1) < \pi(i_2) < \cdots < \pi(i_k)$；
- 反过来说也就是满足上述条件的线段集合必为相容线段集合；
- 因此原问题就等价于在序列 $\pi(1)\pi(2)\cdots\pi(n)$ 中寻找最长的递增子序列（注意，子序列中的元素只需保持在原序列中的相对位置而不必保持连续，这与子串不同）
- 将 $\pi(1)\pi(2)\cdots\pi(n)$ 存放于序列 A 中，设 $A[i] = \pi(i)$，$L[i]$ 为结束于 $A[i]$ 的最长递增子序列的长度，则有如下关系成立：

$$L[i] = 1 + \max\{L[k] \mid A[k] \leq A[i] \text{ and } k < i\}$$

- 为恢复出所找到的子序列，将每个 $L[i]$ 对应的 k 保存在 $F[i]$ 中

算法描述

Algorithm LongestSubsequence$(A[], n)$

$L[1] \leftarrow 1;\ F[1] \leftarrow -1;$
for $i \leftarrow 2$ **to** n **do**
    $L[i] \leftarrow 1;\ F[i] \leftarrow -1;$
    for $k \leftarrow 1$ **to** $i-1$ **do**
        if $A[k] \leq A[i]$ **and** $L[i] < L[k]+1$ **then**
            $L[i] \leftarrow L[k]+1;\ F[i] \leftarrow k;$
        end
    end
end
$index \leftarrow 1;$
for $i \leftarrow 2$ **to** n **do**
    if $L[index] < L[i]$ **then** $index \leftarrow i;$
end
$length \leftarrow L[index];$
for $i \leftarrow length$ **down to** 1 **do** $B[i] \leftarrow A[index];\ index \leftarrow F[index]$;
return $B[], length;$

算法描述

Algorithm `LongestSubsequence`$(A[], n)$

$L[1] \leftarrow 1;\ F[1] \leftarrow -1;$
for $i \leftarrow 2$ **to** n **do**
    $L[i] \leftarrow 1;\ F[i] \leftarrow -1;$
    for $k \leftarrow 1$ **to** $i-1$ **do**
        if $A[k] \leq A[i]$ **and** $L[i] < L[k]+1$ **then**
            $L[i] \leftarrow L[k]+1;\ F[i] \leftarrow k;$
        end
    end
end
$index \leftarrow 1;$
for $i \leftarrow 2$ **to** n **do**
    if $L[index] < L[i]$ **then** $index \leftarrow i;$
end
$length \leftarrow L[index];$
for $i \leftarrow length$ **down to** 1 **do** $B[i] \leftarrow A[index];\ index \leftarrow F[index]$;
return $B[], length;$

$$W(n) \in O(n^2)$$

主要内容

问题描述及解题思路

Exercise (8)

在一条街上有 n 所房子，$H[i]$ $(1 \le i \le n)$ 是第 i 所房子离街道起点处的距离 (以米为单位)，假定 $H[1] < H[2] < \cdots < H[n]$。目前该街道上还没有一所邮局，现计划新建若干所邮局，使得每所房子到最近的邮局距离在 100 米以内。试设计一个时间复杂度为 $O(n)$ 的算法，计算出新建邮局的位置，即每所新建邮局离街道起点处的距离 $P[j]$ $(1 \le j \le m)$，同时确保新建邮局个数 m 最小。

- 贪心方法思路：
  - 首先确定第一所邮局位置应为 $P[1] = H[1] + 100$；
  - 其后从 $H[2]$ 开始检查每个 $H[i]$，若 $H[i] > P[m] + 100$，则 m 增 1，$P[m] = H[i] + 100$，否则继续检查下一所房子位置 $H[i+1]$；
  - 最后，如果 $P[m] > H[n]$，则令 $P[m] = H[n]$

算法描述

Algorithm `PostOffice`$(H[], n)$

```
P[1] ← H[1] + 100;
m ← 1;
for i ← 2 to n do
    if H[i] > P[m] + 100 then
        m ← m + 1;
        P[m] ← H[i] + 100;
    end
end
if P[m] > H[n] then P[m] ← H[n] ;
return (P, m);
```

算法的正确性

反证法.

- 假设存在某种更优的邮局位置设置 $P'[1..k]$，使得所需新建的邮局数目更少为 $k < m$
- 根据 $P[i]$ 的位置选择有 $P'[i] \leq P[i]$，否则：若有 $P'[a-1] \leq P[a-1]$ 而 $P'[a] > P[a]$，设 $P[a] = H[b] + 100$，则 $P'[a] - H[b] > 100$ 而 $P'[a-1] - H[b] \leq P[a-1] - H[b] < -100$，即房子 $H[b]$ 未被 P' 覆盖；
- 因此 $P'[k] \leq P[k]$，且由于 $m > k$，必存在 $p \geq 1$ 使 $H[j+p] > P[k] + 100$，所以 $P'[k] < H[j+p] - 100$，即存在房子 $H[j+p]$ 未被 P' 覆盖，与 P' 是原问题的解矛盾。

$\square$