算法设计与分析

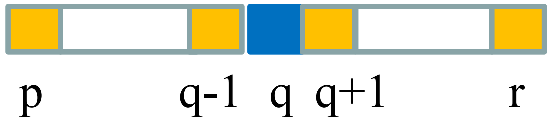
第2章 递归与分治策略
(3)

主要内容

- 快速排序
- 线性时间选择

快速排序

- 输入:数组a[p:r]
- 输出:已排序数组a[p:r]



- 快速排序算法设计：
  - 分: 以a[p]为基准元素将a[p:r]划分为三段
    - a[p:q-1]，a[q]，a[q+1:r]
    - a[p:q-1]任意元素<=a[q]
    - a[q+1:r]中任意元素>=a[q]
  - 递归求解：递归调用快排算法分别对a[p:q-1]和a[q+1:r]进行排序
  - 合：不需要执行任何计算,a[p:r]已排好序

a[p] ？

- 特点：重在**分**

快速排序

- 分：记录的比较和交换是从两端向中间进行
  - 关键字较大的记录一次就能交换到后面单元
  - 关键字较小的记录一次就能交换到前面单元
- 记录每次移动的距离较大，因而总的比较和移动次数较少

```
void QuickSort (Type a[], int p, int r){
    if (p<r) {
      int q=Partition(a,p,r);
      QuickSort (a,p,q-1); //对左半段排序
      QuickSort (a,q+1,r); //对右半段排序
    }
}
```

将小于a[p]的元素放在左半部分，大于a[p]的元素放在右半部分

快速排序

O(r-p)

```
int Partition (Type a[], int p, int r){
        int i = p, j = r + 1;
        Type x=a[p];
        // 将< x的元素交换到左边区域
        // 将> x的元素交换到右边区域
        while (true) {
           while (a[++i] <x);
           while (a[- -j] >x);
           if (i >= j) break;
           Swap(a[i], a[j]);
           }
        a[p] = a[j];
        a[j] = x;
        return j;
}
```

{6, 7, 5, 2, 5, 8} 初始序列 x=6

{6, 7, 5, 2, 5, 8} --j;

{6, 7, 5, 2, 5, 8} --j;

{6, 5, 5, 2, 7, 8} ++i;

{6, 5, 5, 2, 7, 8} ++i;--j

{2, 5, 5} 6 {7, 8} 完成

快速排序

- 复杂度分析
  - 基本操作：比较和移动

```
QuickSort (a[], int p, int r){
    if (p<r) {
      int q=Partition(a,p,r);
      QuickSort (a,p,q-1); //对左半段排序
      QuickSort (a,q+1,r); //对右半段排序
    }
}
```

T(n)

O(n)

? T(n/2)　? T(n-1)

快速排序-复杂度分析

- 快速排序算法运行时间与划分是否对称有关
- 最坏情况：
  - 两个区域分别包含n-1个元素和1个元素。
    - 例如：1，2，3，4，5 以1作为基准划分为（1），（2，3，4，5）两部分
  - 时间复杂性

$$T(n)=\begin{cases} O(1) & n\le 1 \\ T(n-1)+O(n) & n>1 \end{cases}$$

Partition的计算时间= O(n)

- $T(n)=O(n^2)$

快速排序-复杂度分析

- 最好情况
  - 每次划分都产生两个大小为n/2的区域。
  - 时间复杂性为：

$$T(n)=\begin{cases} O(1) & n\le 1 \\ 2T(n/2)+O(n) & n>1 \end{cases}$$

T(n)=O(nlogn)

快速排序

- 改进：
  - 快速排序算法的性能取决于划分的对称性
  - 选择一个好的划分元素，做到平均划分
- 做法：
  - 通过修改算法partition，设计出采用随机选择策略的快速排序算法
  - 在数组还没有被划分时，在a[p:r]中随机选出一个元素作为划分基准，可以使划分基准的选择是随机的，从而可以期望划分是较对称的。

```
int RandomizedPartition (Type a[], int p, int r)   {
        int i = Random(p,r);
        Swap(a[i], a[p]);
        return Partition (a, p, r);
}
```

快速排序

- We can say now:
  - “With high probability, randomized quicksort runs in (nlgn) time.”
- Where before, all we could say is:
  - “If you give me random input data, quicksort runs in expected (nlgn) time.”
- 随机化思想：
  - 被动变主动

快速排序

- 最坏时间复杂度：$O(n^2)$
- 平均时间复杂度：$O(n\log n)$
- 辅助空间：$O(n)$或$O(\log n)$

线性时间选择

- 问题来源：网球公开赛
  - 怎么通过最少的比赛场次公平地决出各个选手的名次（前提是假设比赛中的输赢完全由选手的水平决定）
  - 最简单问题：就是找出序列中的最大值和最小值。
    - 遍历序列，并且用一个额外的空间保存当前为止的最大值或最小值，共需要进行n-1次比较，
- 网页排序问题：只需要top K
  - 比较排序，最优也要O(nlogn)
- 有72位同学参加期末考试，要找出考分排在第15名的同学
  - 将试卷排序， 从最高分试卷往下数直到排名第15的试卷
  - 时间复杂度是O(nlogn)
  - ？有没有更快的方法

线性时间选择---第k选择问题

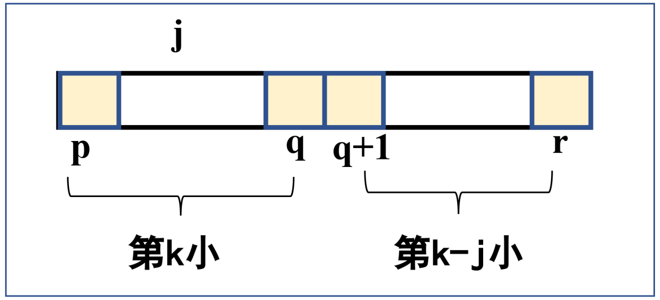
- 元素选择问题
  - 给定无序序列集a中n个元素和一个整数k，$1 \le k \le n$，要找出这n个元素中第k小的元素x.
  - 例如：a=[6,9,-2,-9,12,1, 15], select(a, 3)=1
- 分析
  - k=1 找最小元素： O(n)
  - k=n 找最大元素： O(n)
  - k=(n+1)/2 找中位数?
- 解决方案
  - 排序
  - 快排思想

一般的选择问题也可以在O(n)时间内解决

线性时间选择---第k选择问题

- Blum,Floyd,Pratt,Rivest和Targan等于1973年发明了时间复杂度为O(n)的算法
- 设计思路
  - 利用PARTITION过程。如果划分元素在A(i)的位置上，则有i-1个元素小于或等于A(i)，且有n-i个元素大于或等于A(i)。此时，
    - 若k=i，则A(i)即是第k小元素；
    - 若k<i，则第k小元素将出现在A(1:i-1)中；
    - 若k>i，则第k小元素将出现在A(i+1:n)中。

线性时间选择---第k选择问题



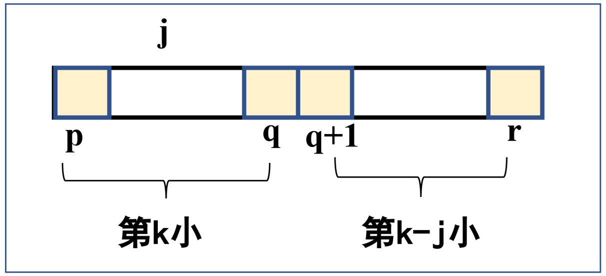
- 输入：数组a, p, r, k，p≤k≤r
- 输出：第k小元素x
- 设计
  - 分: 选择划分基准元素a[p], 将a[p:r]分为a[p:q]和a[q+1:r] 两个子数组。
    - a[p:q]任意元素<= a[q+1:r]中任意元素
  - 解：
    - 递归调用选择算法对a[p:q]或a[q+1:r]中1个子问题求解
    - 只有一个元素时, 找到第k小元素，
  - 合：不需要执行任何操作
- 注意
  - 左半部分子问题是求解a[p:q]中第k小元素
  - 右半部分子问题是求解a[q+1:r]中第k-j小元素(j=q-p+1)

线性时间选择---第k选择问题

- 输入：a, k;
- 输出：第k小元素;

和快排算法有什么不同??

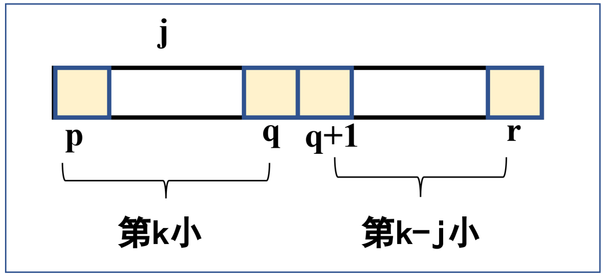
```
Type RandomizedSelect(Type a[],int p,int r,int k)
{
    if (p==r) return a[p];//只有1个元素
    int q=RandomizedPartition(a,p,r),
    j=q-p+1;          //a[p:q]中元素个数j
    if (k<=j) return RandomizedSelect(a,p,q,k);
    else return RandomizedSelect(a,q+1,r,k-j);
}
```



线性时间选择---第k选择问题

- 输入：a, k;
- 输出：第k小元素;

```
SELECT(A,n,k)  {       //在数组A(1:n)中找第k小元素，并将之放在A(k)中
    integer n,k,p,r,j;
    p←1;r ←n+1;A(n+1) ←+∞;       //A(n+1) 置为大值，用于限界
    loop                          //在进入循环时，1≤m≤k≤r≤n+1
        q ← Partition (A, p, r)      //返回q,它使得A(q)是第q小的值
        case
            k=q:  return;
            k<q:  r←q;       //q是新的上界
            else: p←q+1;  //k>q, q+1是新的下界
        endcase
    repeat
}
```



线性时间选择---第k选择问题

- 算法分析的两点假设
  - ① A中的元素互异
  - ②随机选取划分元素，且选择A中任一元素作为划分元素的概率相同
- 第k选择问题分析
  - 分：每次调用Partition(A,p,r),所需的元素比较次数是O(r-p+1)。
  - 解：在执行一次Partition后，或者找到第k小元素，或者将在缩小的子集(A(p,k)或A(k+1,r))中继续查找。缩小的子集的元素数将至少比上一次划分的元素数少1。
  - 合：无需合

启示：实验的完备性，各种特征的数据集

线性时间选择

- 与快速排序算法的异同点?
  - 基本思想相同
    - 对输入数组进行递归划分
  - 不同
    - 线性时间选择只对划分出的子数组之一进行递归处理

在最坏情况下，算法**randomizedSelect**需要$O(n^2)$计算时间
但可以证明，算法**randomizedSelect**可以在$O(n)$平均时间内找出n个输入元素中的第k小元素。

线性时间选择

- (1) 最坏情况：递归算法

  例如: 1,2,3,4,5 分成1, (2,3,4,5)

  $T(n)=T(n-1)+O(n) =O(n^2)$

```
Type RandomizedSelect(Type a[],int p,int r,int k)
{
      if (p==r) return a[p];//只有1个元素
      int q=RandomizedPartition(a,p,r),
      j=q-p+1;          //a[p:q]中元素个数j
      if (k<=j) return RandomizedSelect(a,p,q,k);
      else return RandomizedSelect(a,q+1,r,k-j);
}
```

T(n)

O(n)

? T(n-1) **? T(n/2)**

线性时间选择

```
SELECT(A,n,k){ //在数组A(1:n)中找第k小元素，并将之放在A(k)中
    integer n,k,p,r,j;
    p←1;r ←n+1;A(n+1) ←+∞ ;
      //A(n+1)被定义，并置为一大值，用于限界
    loop  //在进入循环时，1≤p≤k≤r≤n+1
       j ←r ;
         //将剩余元素的最大下标加1后置给j
       j ← Partition(A, p , r) ;
         //返回j,它使得A(j)是第j小的值
       case
            k=j:  return;
            k<j:  r←j;       //j是新的上界
            else: p←j+1;  //k>j, j+1是新的下界
       endcase
    repeat
}
```

(1) 最坏情况-递推算法:

最坏情况时间是$O(n^2)$

- A中的元素已经按照递增的顺序排列，且k=n
- 此时，需要n次调用Partition过程，且每次返回的元素位置是子集中的第一个元素，子集合的元素数一次仅减少1，而j值不变。
- 则, n次调用的时间总量是

$$O\left(\sum_{1}^{n}(i-1)\right)=O(n^2)$$

元素比较次数

线性时间选择

- (2) 最好情况：每次划分都是一半

$T(n)= T(n/2)+ n = T(n/2^2)+(n/2)+n=\ldots$

$=T(n/2^i)+n/2^{i-1}+\ldots+n$

$=T(1) + n * \sum_{i=0}^{\log n-1)} (\frac{1}{2})^i$

$\leq T(1) + \underline{n * \sum_{i=0}^{\infty} (\frac{1}{2})^i}$

$=1+2n$

有$T(n)=O(n)$

```
Type RandomizedSelect(Type a[],int p,int r,int k){
    if (p==r) return a[p];//只有1个元素
    int q=RandomizedPartition(a,p,r),
    j=q-p+1;        //a[p:q]中元素个数j
    if (k<=j) return RandomizedSelect(a,p,q,k);
    else return RandomizedSelect(a,q+1,r,k-j);
}
```

线性时间选择

- (3)　平均情况
  - 设 $T_A^k(n)$是找A(1:n)中第k小元素的平均时间。
  - $T_A(n)$是SELECT的平均计算时间，则有

$$T_A(n) = \frac{1}{n}\sum_{1\le k\le n} T_A^k(n)$$

- 并定义
  - $R(n) = \max\{T_A^k(n)\}$

- 则有：T(n)≤R(n)

线性时间选择

- 定理　SELECT的平均计算时间$T_A(n)$ 是O(n)
- 证明：
  - Partition和SELECT中，case语句的执行时间是O(n)。
  - 在随机等概率选择划分元素时，首次调用Partition中划分元素v刚好是A中第i小元素的概率为1/n，1≤i≤n。
  - 则，存在正常数c，c>0,有，
    - $T_A^k(n) \le cn + \frac{1}{n}(\sum_{1\le i\le k} T_A^{k-i}(n-i) + \sum_{k\le i\le n} T_A^k(i-1))$　　n≥2
    -
    - 且有, $R(n) \le cn + \frac{1}{n} max_k\{\sum_{1\le i<k} R(n-i) + \sum_{k<i\le n} R(i-1)\}$
    - $= cn + \frac{1}{n} max_k\{\sum_{n-k+1}^{n-1} R(i) + \sum_{k}^{n-1} R(i)\}$　　n≥2

线性时间选择

令c≥R(1)。利用数学归纳法证明，对所有n≥2，有R(n)≤4cn.

① 当n=2时，由上式得： $R(n) \le 2c + \frac{1}{2}\max\{R(1), R(1)\} \quad \le 2.5c < 4cn$

② 假设对所有的n, 2≤n<m，有R(n)≤4cn

③ 当n = m时，有， $R(n) \le cm + \frac{1}{m}\max_{k}\{\sum_{m-k+1}^{m-1} R(i) + \sum_{k}^{m-1} R(i)\}$

由于R(n)是n的非降函数，故在当m为偶数而k=m/2，或当m为奇数而k=(m+1)/2时，

$\sum_{n-k+1}^{n-1} R(i) + \sum_{k}^{n-1} R(i)$ 取得极大值。因此，

若m为偶数，则 $R(m) \le cm + \frac{2}{m}\sum_{m/2}^{m-1} R(i) \quad \le cm + \frac{8c}{m}\sum_{m/2}^{m-1} i < 4cm$

若m为奇数，则 $R(m) \le cm + \frac{2}{m}\sum_{(m+1)/2}^{m-1} R(i) \quad \le cm + \frac{8c}{m}\sum_{(m+1)/2}^{m-1} i < 4cm$

由于$T_A(n)$≤R(n)，所以$T_A(n)$≤4cn。 故，$T_A(n)=O(n)$

线性时间选择

- 问题：
  - 虽然采用了分治策略,但上面的算法在最坏情形下的时间复杂度仍然是$O(n^2)$阶的.
- 原因：
  - 划分算法无法避免划分元素位于数组的两端.
- 改进思路（O(n)阶选择算法的思路）
  - 解决问题的关键是：能否花费O(n)代价改进划分算法,保证每次划分的分点不接近两端.换句话说,即使不能恰好在中间,也要保证较小的一半占有一定的比例.
    - 两点考虑：划分的方法，划分的代价

$T(n)=T(n-1)+O(n) =O(n^2)$ → $T(n)=T(n/2)+O(n)$

线性时间选择

- 例如
  - 假设算法丢弃1/3并对剩余的2/3部分递归，则在第2次调用中，元素的个数变为2n/3个，第3次调用中为4n/9个，第4次调用中变为8n/27等等。
  - 假设在每次调用中，算法对每个元素耗费的时间不超过一个常数，则耗费在处理所有元素上的全部时间产生一个**几何级数**:
  - $cn+(2/3)cn+(2/3)^2cn+\ldots+(2/3)^icn+\ldots$
  - $<= \sum_{j=0}^{\infty} cn(2/3)^i = 3cn = \theta(n)$

cn

T(n/3) 2cn/3)

4cn/9)

…

线性时间选择

- 如果能在线性时间内找到一个**划分基准**，使得按这个基准所划分出的2个子数组的长度都至少为原数组长度的ε倍(0<ε<1是某个正常数)，那么就可以在**最坏情况**下用O(n)时间完成选择任务。

例如，若ε=9/10，算法递归调用所产生的子数组的长度至少缩短1/10。所以，在最坏情况下，算法所需的计算时间T(n)满足递归式T(n)≤T(9n/10)+O(n) 。由此可得T(n)=O(n)。

线性时间选择

- 线性时间选择问题：
  - 在一个具有n个元素的集合中，能够在最坏情况下用线性时间找到中项或通常意义上的第k小元素。
- 基本思想: 在线性时间找一个**好划分基准**
  - 最好：二分 →1/2

$$T(n)=T(n/2)+O(n)$$

  - 泛化：问题的规模以**几何级数递减**，也就是在每个调用过程中，问题的规模以一个常因子被减小。 → 1/x

$$T(n)=T(n/x)+O(n)$$

线性时间选择

- 采用两次取中的规则选取划分元素
  - 最坏情况是O(n)的选择算法

- 中位数
  - 一个中位数是它所在集合的“中点元素”
  - n为奇数时，中位数唯一
  - n为偶数时，中位数有两个，分别在位置n/2和n/2+1处

线性时间选择

- 两次取中选取划分基准的方法

- 取中位数问题实例：
  - 给定以下25个元素，选择第13小元素。
  - A[1..25]， 中位数k=$\lceil 25/2 \rceil$=13

8，33，17，51，57，49，35，11，25，37，14，3，2，13，52，12，6，29，32，54，5，16，22，23，7

线性时间选择-实例

- 第1步: 分组, n=25, 每组元素r=5,分成5组

8，33，17，51，57，49，35，11，25，37，14，3，2，13，52，
12，6，29，32，54，5，16，22，23，7

实现关键：原问题：A[1..25]
分组：A[1..5], A[6...10],A[11...15],A[16...20],A[21...25]

8，33，17，51，57，
49，35，11，25，37，
14，3，2，13，52，
12，6，29，32，54，
5，16，22，23，7

→

8，17，33，51，57，
11，25，35，37，49，
2，3，13，14，52，
6，12，29，32，54，
5，7，16，22，23

第1次取中

- 第2步: 第1次取中

必须排序吗？

如何保存中位数？

线性时间选择-实例

- 第3步：第2次取中

8，	33，	17，	51，	57，
49，	35，	11，	25，	37，
14，	3，	2，	13，	52，
12，	6，	29，	32，	54，
5，	16，	22，	23，	7

8，	17，	33，	51，	57，
11，	25，	35，	37，	49，
2，	3，	13，	14，	52，
6，	12，	29，	32，	54，
5 ，	7 ，	16，	22，	23

13, 16, **29**, 33, 35

第2次取中

如何实现？ 递归调用

线性时间选择-实例

- 第4步：以29作为划分点，重新划分数组

13, 16, **29**, 33, 35

8，33，17，51，57，49，35，11，25，37，14，3，2，13，52，12，6，29，32，54，5，16，22，23，7

↓

8，17，11，25，14，3，2，13，12，6， 5，16，22，23，7, 29，33， 51， 57，49，35， 37， 52， 32， 54

- A1={8,17,11,25,14,3,2,13,12,6, 5,16,22,23,7}
- A2={29}
- A3={33,51,57,49,35,37,52,32,54}

线性时间选择-实例

- 目标：利用线性时间选择算法找出中位数（找⌈n/2⌉小元素）

  8，33，17，51，57，49，35，11，25，37，14，3，2，13，

  52，12，6，29，32，54，5，16，22，23，7

  - A[1..25]
  - **中位数**k=⌈25/2⌉=13

- 第一轮划分基准29：
  - A1={8,17,11,25,14,3,2,13,12,6,5,16,22,23,7}, A2={29}
  - A3={33,51,57,49,35,37,52,32,54}
- 判断：|A1|=15>13,故在A1中**继续递归查找（比考虑A2和A3）**：A=A1
  - A={8,17,11,25,14,3,2,13,12,6,5,16,22,23,7}，进行下一轮递归
    - 分组：{8,17,11,25,14}, {3,2,13,12,6}, {5,16,22,23,7}
    - 1次取中：{14, 6,16}
    - 2次取中：得14
    - 重新划分为：A1={8,11,3,2,13,12,6,5,7}，A2={14}, A3={17,25,16,22,23}
    - |A1|+|A2|=10<13,故在A3中继续递归查找第3小元素(13-10=3)
      - 返回22

线性时间选择

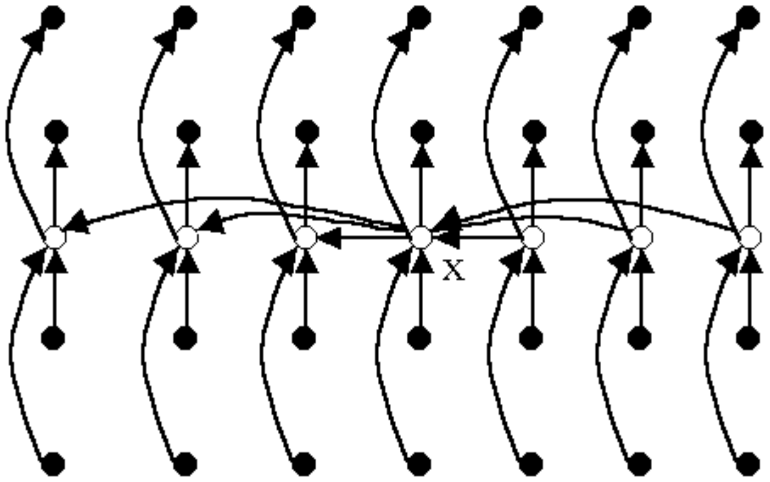
- 两次取中规则的步骤
  - 第1步 将参加划分的n个元素分成$\lceil n/r \rceil$ 组，每组有r个元素(r≥1)。
  - 第2步 对这$\lceil n/r \rceil$组里每组的r个元素进行查找, **找出其中间元素m_i**,$1 \le i \le \lceil n/r \rceil$，共得$\lceil n/r \rceil$个中间值。
  - 第3步 对这$\lceil n/r \rceil$个中间值进行查找，**找出其中间值mm**。
  - 第4步 划分：**将mm作为划分元素**执行划分

线性时间选择

- 线性时间选择问题：在一个具有n个元素的集合中，能够在最坏情况下用线性时间找到中项或通常意义上的第k小元素。
- 设计
  - 分:　采用二次选中规则选择划分基准元素，将a[p:r]分为a[p:q],,a[q+1:r]两个子数组。
  - 解：
    - 元素个数足够少时直接调用排序算法找第k小元素
    - 递归调用选择算法对a[p:q]或a[q+1:r] 1个子问题求解
  - 合：不需要执行任何操作

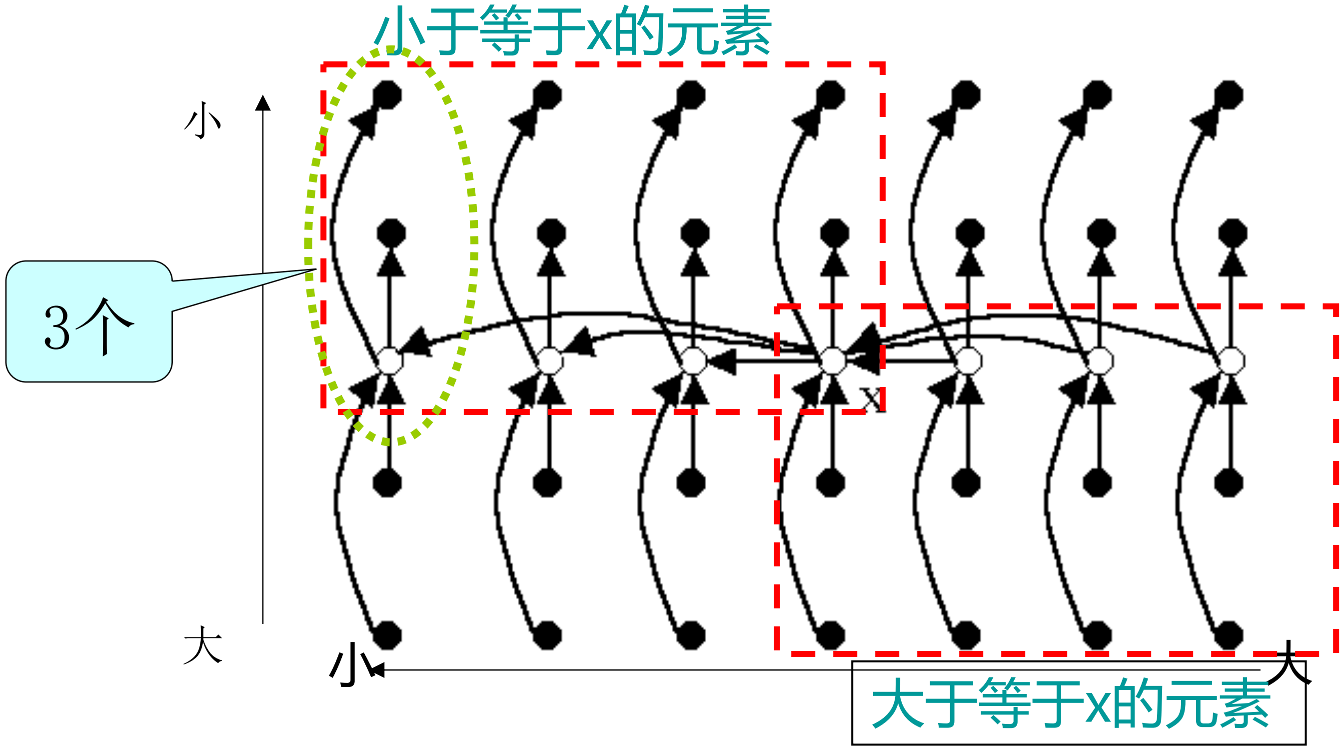
线性时间选择

- 二次取中的实现
  - 选择中位数的算法：select(a,k)
  - 输入：数组a，包含n个元素
  - 输出：第k小元素（中位数即k=n/2）
  - 设每组包含的元素个数 r=5
  - 步骤：
    - 将n个输入元素划分成⌈n/5⌉个组，每组5个元素，只可能有一个组不是5个元素
    - 第1次取中　用任意一种排序算法，将每组中的元素排好序，并取出每组的中位数，共⌈n/5⌉个
    - 第2次取中　递归调用select来找出这⌈n/5⌉个元素的中位数。
      - 如果⌈n/5⌉是偶数，就找它的2个中位数中较大的一个。以这个元素作为划分基准



线性时间选择

- 为什么能实现线性选择？



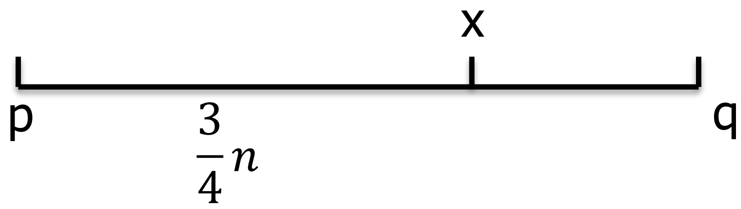
设所有元素互不相同。

(1)每一组中有2个元素小于本组的中位数

(2) 共有$\frac{n-5}{5}$组，这些组中又有一半的中位数小于x，即$\frac{n-5}{10}$组

(3)这一半组中有3个元素小于x

$\Rightarrow$ x至少比$3 * \frac{n-5}{10}$个元素大



•因为在每一组中有2个元素小于本组的中位数，而n/5个中位数中又有(n-5)/10个小于基准x, 在此情况下，找出的基准x至少比**3(n-5)/10**个元素**大**

•同理，基准x也至少比**3(n-5)/10**个元素**小**。而当n≥75时，3(n-5)/10≥n/4

•所以,按此基准划分所得的2个子数组的长度都**至少缩短1/4**。

线性时间选择

输入：A, n, k　　　　　//对特定的r分析SELECT2:选取r=5

输出：第k小元素

SELECT2（A,n,k){　　　　//假定A中的元素各不相同　　　**T(n)**

① 若n≤r，则采用插入法直接对A分类并返回第k小元素　　→O（1）

② 把A分成大小为r的$\lfloor n/r \rfloor$ 个子集合 ,忽略多余元素　　→O（n）

③ 设M={m1,m2,...m }是$\lfloor n/r \rfloor$个子集合的中间值集合　　→O（n）

④ v←SELECT2(M, $\lfloor n/r \rfloor$, $\lceil \lfloor n/r \rfloor/2 \rceil$)　　→T(n/5)

⑤ j←Partition(A，v)　　→O（n）

⑥ case　　→T(3n/4),　$n \geqslant n_0$

k=j: return(v)

k<j: 设S是A(1:j-1)中元素的集合;　　return(SELECT2(S, j-1,k))

else: 设R是A(j+1:n)中元素的集合;　　return(SELECT2(R, n-j,k-j))

endcase

}

线性时间选择

```
Type Select(Type a[], int p, int r, int k){
    if (r-p<75) {
        用某个简单排序算法对数组a[p:r]排序;
         return a[p+k-1];
     };
    for ( int i = 0; i<=(r-p-4)/5; i++ )        //i表示第几组
        将a[p+5*i]至a[p+5*i+4]的第3小元素与a[p+i]交换位置;
    //找中位数的中位数，r-p-4即上面所说的n-5
    Type x = Select(a, p, p+(r-p-4)/5, (r-p-4)/10);
    int i=Partition(a,p, r, x),
    j=i-p+1;
    if (k<=j) return Select(a,p,i,k);
    else return Select(a,i+1,r,k-j);
}
```

返回第k小元素

第一次找到$\lceil n/5 \rceil$个中位数

第二次找**1**个中位数

线性时间选择

	cost
Type **Select**(Type a[], int p, int r, int k){	T(n)
if (r-p<75) {	
用某个简单排序算法对数组a[p:r]排序;	
return a[p+k-1];	
};	
for (int i = 0; i<=(r-p-4)/5; i++)	
将a[p+5*i]至a[p+5*i+4]的第3小元素与a[p+i]交换位置;	O(1)*n/5
//找中位数的中位数，r-p-4即上面所说的n-5	
Type x = Select(a, p, p+(r-p-4)/5, (r-p-4)/10);	T(n/5)
int i=Partition(a,p,r, x),	O(n)
j=i-p+1;	
if (k<=j) return Select(a,p,i,k);	T(3n/4)
else return Select(a,i+1,r,k-j);	
}	

有： $T(n)=O(n)+ T(n/5)+T(3n/4)$

线性时间选择

复杂度分析

$$T(n) \le \begin{cases} C_1 & n < 75 \\ C_2 n + T(n/5) + T(3n/4) & n \ge 75 \end{cases}$$

第一次选中时间O(1)*n/5
加上Partition时间O(n)

解得： T(n)=**O(n)**

上述算法将每一组的大小定为5，并选取75作为是否作递归调用的分界点。这2点保证了T(n)的递归式中2个自变量之和n/5+3n/4=19n/20=εn，0<ε<1。这是使T(n)=O(n)的关键之处。当然，除了5和75之外，还有其他选择。

线性时间选择

- 故有，

$$T(n)=\begin{cases} cn & n<n_0 \\ T\left(\frac{n}{5}\right)+T\left(\frac{3n}{4}\right)+cn & n\geq n_0 \end{cases}$$

- 用归纳法可证：　　T(n)≤20cn

  即，T(n) = O(n)

线性时间选择

- 定理　设b,c1,c2是非负常数，那么递推式

$$f(n)=\begin{cases} 0 & n=0 \\ b & n=1 \\ f(\lfloor c_1 n \rfloor)+f(\lfloor c_2 n \rfloor)+bn & n\ge 2 \end{cases}$$

的解是

$$f(n)=\begin{cases} O(n\log n) & c_1+c_2=1 \\ \Theta(n) & c_1+c_2<1 \end{cases}$$

线性时间选择

- 思考题
  - 在算法select中，输入元素被分为每组5个元素，如果它们被分为每组7个元素，该算法仍然以线性时间工作吗？如果分成每组3个元素呢？

线性时间选择

- 讨论
  - 改进算法的基本思路是对原有算法进行分析,在其次要部分付出一定代价,使得算法能在主要部分获得收益.
    - 快排的改进
      - 最坏情况：$O(n^2)$
    - 划分基准的寻找->线性时间
      - 最坏情况：$O(n)$

思考

- 第k选择的算法和快速排序算法的共同点？
  - 依赖于使用随机的划分元素，性能的保证也来自于概率
- 线性时间选择算法的关键是什么？
- 讨论：排序有哪些应用？

小结

- 重点：
  - 快排算法
  - 线性选择第k小元素算法

- 线性时间选择问题
  - 算法设计思想：
    - 快排 + 二次取中确定划分基准，递归实现
  - 分治算法设计：分、解、合
  - 分治算法描述
  - 算法复杂度分析

小结

- 难点
  - 算法的基本思想和时间复杂度
  - 快排的划分方法
  - 二次取中规则

- 线性选择问题注意
  - 改进算法重点在于改进分解过程，提高算法性能
  - 利用二次取中方法，在线性时间内找到一个划分基准，保证子问题规模以小于1的常数因子缩小