算法分析
The Analysis of Algorithms

第三章　蛮力法

杨春明
西南科学技大学计算机学院

蛮力法

- 就像宝剑不是撬棍一样，科学也很少使用蛮力。——Edward Lytton
- 认真做事常常是浪费时间。——Robert Byrne

- *蛮力法*是一种简单直接地解决问题的方法，常常直接基于问题的描述和所涉及的概念定义。

蛮力法的优点

- 可以用来解决广阔领域的问题
- 对于一些重要的问题，它可以产生一些合理的算法
- 解决问题的实例很少时，它让你花费较少的代价
- 可以解决一些小规模的问题
- 可以作为其他高效算法的衡量标准

教学内容

- 选择排序和冒泡排序
- 顺序查找和蛮力字符串匹配
- 最近对核凸包问题的蛮力算法
- 穷举查找
- 要求
  - 掌握蛮力法的基本思想，了解排序和查找问题的蛮力算法。

选择排序

$$A_0 \le A_1 \le \ldots A_{i-1} \mid A_i, \ldots, A_{min}, \ldots, A_{n-1}$$

已经位于最终的位置上了　　最后的$n-i$个元素

算法　*SelectionSort*$(A[0..n-1])$

//该算法用选择排序对给定的数组排序
//输入：一个可排序数组 $A[0..n-1]$
//输出：非降序排列的数组 $A[0..n-1]$

for $i \leftarrow 0$ **to** $n-2$ **do**
　　$min \leftarrow i$
　　for $j \leftarrow i+1$ **to** $n-1$ **do**
　　if A[j] <A[min] $min \leftarrow j$
　　swap $A[i]$ and A $[min]$

89	45	68	90	29	34	**17**
17 |	45	68	90	**29**	34	89
17	29 |	68	90	45	**34**	89
17	29	34 |	90	**45**	68	89
17	29	34	45 |	90	**68**	89
17	29	34	45	68 |	90	**89**
17	29	34	45	68	89 |	90

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}$$

冒泡排序

$$A_0, \ldots, A_j \overset{?}{\leftrightarrow} A_{j+1}, \ldots, A_{n-i-1} \mid A_{n-i} \leq \ldots \leq A_{n-1}$$

已经位于最终的位置上了

算法 BubbleSort(A[0..n-1])

//该算法用冒泡排序对数组A[0..n-1]排序

//输入：一个可排序的数组A

//输出：非降序排列的数组

for i←0 to n-2 do

  for j←0 to n-2-i do

    if A[j+1]<A[j]

      swap A[j] and A[j+1]

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2-i)-0+1]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2)$$

89 ? 45 68 90 29 34 17

45 89 ? 68 90 29 34 17

45 68 89 ? 90 ? 29 34 17

45 68 89 29 90 ? 34 17

45 68 89 29 34 90 ? 17

45 68 89 29 34 17 |90

45 ? 68 ? 89 ? 29 34 17 |90

45 68 29 89 ? 34 17 |90

45 68 29 34 89 ? 17 |90

45 68 29 34 17 |89 90

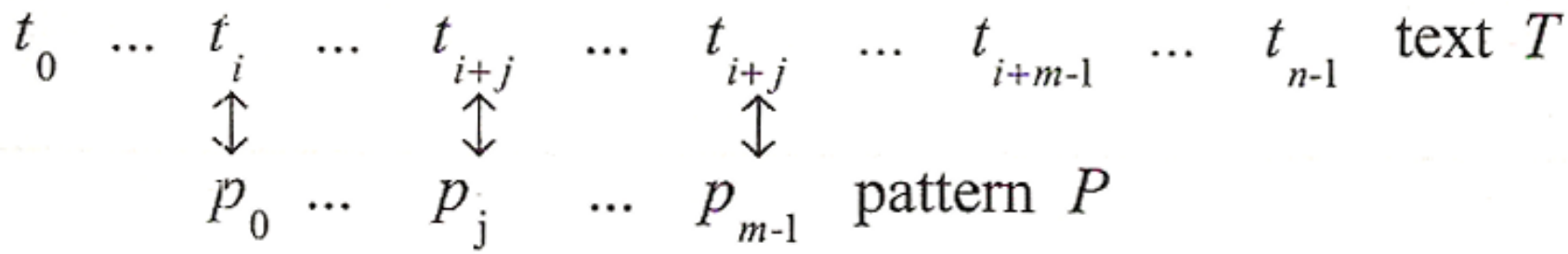
顺序查找

算法 *SequentialSearch2*$(A[0..n],K)$

//顺序查找的算法实现，它用了查找键来做限位器
//输入：一个 n 个元素的数组 A 和一个查找键 K
//输出：第一个值等于 K 的元素的位置，如果找不到这样的元素，返回-1

$A[n] \leftarrow K$
$i \leftarrow 0$
while $A[i] \neq K$ **do**
    $i \leftarrow i+1$
if $i < n$ **return** i
else return -1

蛮力字符串匹配

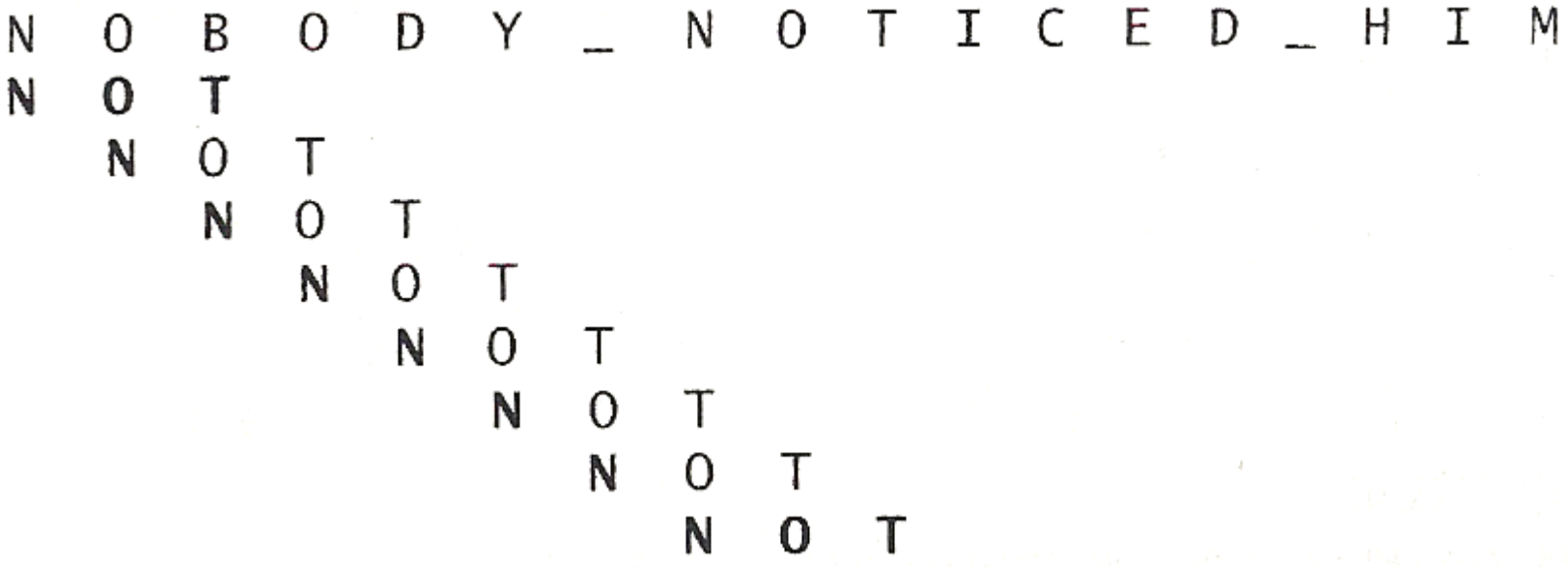


BruteForceStringMatch($T[0..n-1]$,$P[0..m-1]$)

//该算法实现了蛮力字符串匹配
//输入：一个 n 个字符的数组 $T[0..n-1]$代表一段文本
//　　一个 m 个字符的数组 $P[0..n-1]$代表一个模式 、
//输出：如果查找成功的话，返回文本的第一个匹配子串中第一个字符的位置
//　　否则返回-1

for $i \leftarrow 0$ **to** $n-m$ **do**
　　$j \leftarrow 0$
　　while $j < m$ **and** $P[j] = T[i+j]$ **do**
　　$j \leftarrow j+1$
　　if $i = m$ **return** i
return -1

蛮力字符串匹配



$\Theta(n+m)=\Theta(n)$

最近对问题

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

算法 *BruteForceClosestPoints*(*P*)

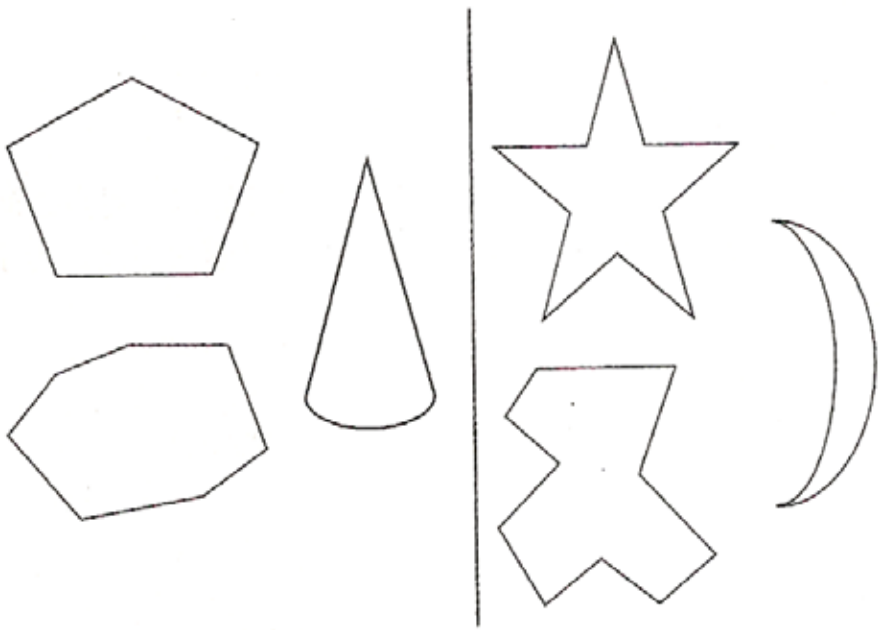
```
//输入：一个 n (n≥2)个点的列表 P，  P1 = (x1, y1),..., Pn = (xn, yn)
//输出：两个最近点的下标，index1 和 index2
dmin ← ∞
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        d ← sqrt((xi − xj)^2 + (yi − yj)^2) // sqrt 是平方根函数
        if d < dmin
            dmin ← d; index1 ← i; index2 ← j
return index1, index2
```

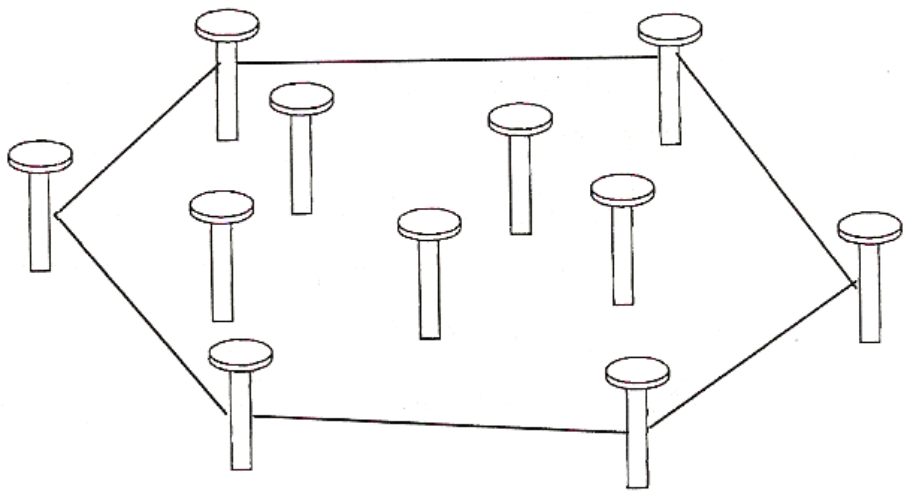
$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2\sum_{i=1}^{n-1} (n-i)$$

$$= 2[(n-1) + (n-2) + \ldots + 1] = (n-1)n \in \Theta(n^2)$$

凸包问题

- **定义** 对于平面上的一个点集合（有限或无限），如果以集合中任意两点P和Q为端点的线段都属于这个集合，则这个集合是*凸*的。
- **定义** 一个点集合S的凸包是包含S的最小凸集合。

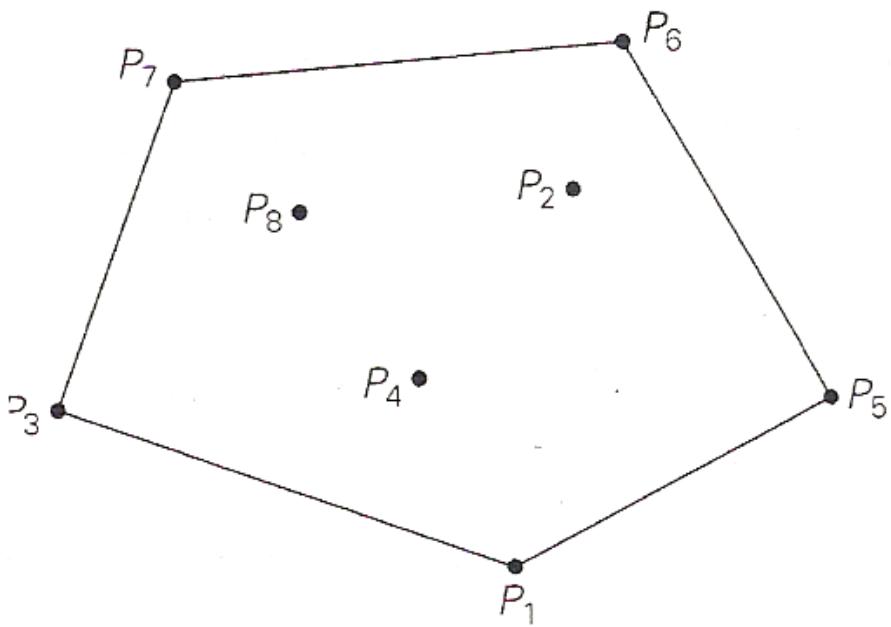




凸包问题

- 定理 任意包含n>2个点（不共线）的集合S的凸包是以S中的某些点为顶点的凸多边形。
- 凸包问题是为一个n个点的集合构造凸包的问题。
- 极点：对于任何一集合中的点为端点的线段来说，它们不是这种线段的中点。

对于一个n各点集合中的两个点P_i和P_j，当且仅当该集合中的其他点都位于穿过这两点的直线的同一边时它们的连线是该集合凸包边界的一部分。

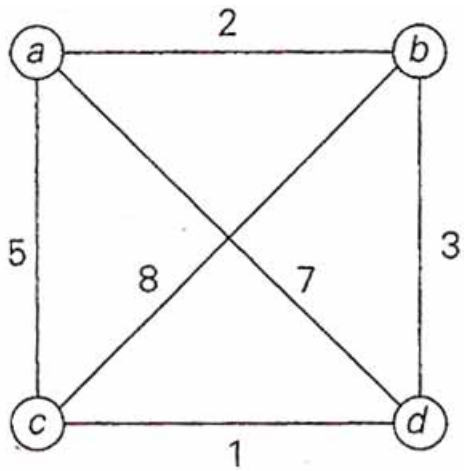


直线方程：**ax+by=c**

$a=y_2-y_1$, $b=x_1-x_2$, $c=x_1y_2-y_1y_2$

穷举查找

- 旅行商问题
- 背包问题
- 分配问题



路线	旅程	
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$l = 2 + 8 + 1 + 7 = 18$	
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$l = 2 + 3 + 1 + 5 = 11$	最佳
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$l = 5 + 8 + 3 + 7 = 23$	
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$l = 5 + 1 + 3 + 2 = 11$	最佳
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$l = 7 + 3 + 8 + 5 = 23$	
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$l = 7 + 1 + 8 + 2 = 18$	

小结

- 蛮力法是一种简单直接地解决问题的方法，通常直接基于问题的描述和所涉及的概念定义。
- 蛮力法的主要优点是它广泛的适用性和简单性；它的主要缺点是大多数蛮力算法的效率都不高。
- 除了相关问题的一些规模非常小的实例，穷举查找法几乎是不实用的。