算法设计与分析

Design and Analysis of Algorithms

第三章　蛮力法

Brute Force

主讲：　　朱东杰 博士、硕导

地点：　　M楼305

电话/微信：18953856806

Email：　　zhudongjie@hit.edu.cn

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
- 最近对和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，掌握排序和查找问题的蛮力算法。

选择排序

以第一个元素为基准，找到最小的元素，交换位置，然后以第二个为基准……

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

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

基本操作“比较”的执行次数：

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

冒泡排序

第一遍把最大元素沉到最后位置，第二遍把第二大元素沉下去……

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

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 $\overset{?}{\leftrightarrow}$	45	68	90	29	34	17
45	89 $\overset{?}{\leftrightarrow}$	68	90	29	34	17
45	68	89 $\overset{?}{\leftrightarrow}$	90 $\overset{?}{\leftrightarrow}$	29	34	17
45	68	89	29	90 $\overset{?}{\leftrightarrow}$	34	17
45	68	89	29	34	90 $\overset{?}{\leftrightarrow}$	17
45	68	89	29	34	17	|90
45 $\overset{?}{\leftrightarrow}$	68 $\overset{?}{\leftrightarrow}$	89 $\overset{?}{\leftrightarrow}$	29	34	17	|90
45	68	29	89 $\overset{?}{\leftrightarrow}$	34	17	|90
45	68	29	34	89 $\overset{?}{\leftrightarrow}$	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

查找键做“限位器”：避免每次while循环中都检查 *i* 是否到了表的末尾。

如果是有序表，则只需要找到一个大于或等于查找键的元素，算法就可以终止。

蛮力字符串匹配

$t_0 \quad \ldots \quad t_i \quad \ldots \quad t_{i+j} \quad \ldots \quad t_{i+j} \quad \ldots \quad t_{i+m-1} \quad \ldots \quad t_{n-1}$ text T

$\quad\quad\quad \updownarrow \quad\quad\quad \updownarrow \quad\quad\quad \updownarrow$

$p_0 \quad \ldots \quad p_j \quad \ldots \quad p_{m-1}$ pattern P

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

//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 – 例子:

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
```

如果文本的长度是n，模式的长度是m；最坏情况可能做n-m+1次模式匹配（例：n=6,m=3）的尝试，而1次模式匹配尝试在最坏情况下会做m次比较(最后一个字符不匹配)。

所以，最坏情况下的算法效率属于Θ(nm)，

平均效率属于Θ(m+n)=Θ(m)。

更多的蛮力算法例子:最近对

- 问题: 在k-维空间中的n个点中，找距离最近的点对;
- 蛮力算法: 计算每对点之间的距离;
- 效率:

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 $(n \geq 2)$ 个点的列表 P，$P_1=(x_1,y_1),\ldots,P_n=(x_n,y_n)$

//输出：两个最近点的下标，$index1$ 和 $index2$

$dmin \leftarrow \infty$

for $i \leftarrow 1$ **to** $n-1$ **do**

  for $j \leftarrow i+1$ **to** n **do** // 约束i<j，从而避免同一对点被比较两次。

    $d \leftarrow sqrt((x_i-x_j)^2+(y_i-y_j)^2)$ // $sqrt$ 是平方根函数

    if $d < dmin$

      $dmin \leftarrow d$; $index1 \leftarrow i$; $index2 \leftarrow j$

return $index1$, $index2$

不过不开平方根，只比较$(x_i\text{-}x_j)^2+(y_i\text{-}y_j)^2$，则算法得到简化。平方操作执行次数：

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

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

更多的蛮力算法例子:凸包

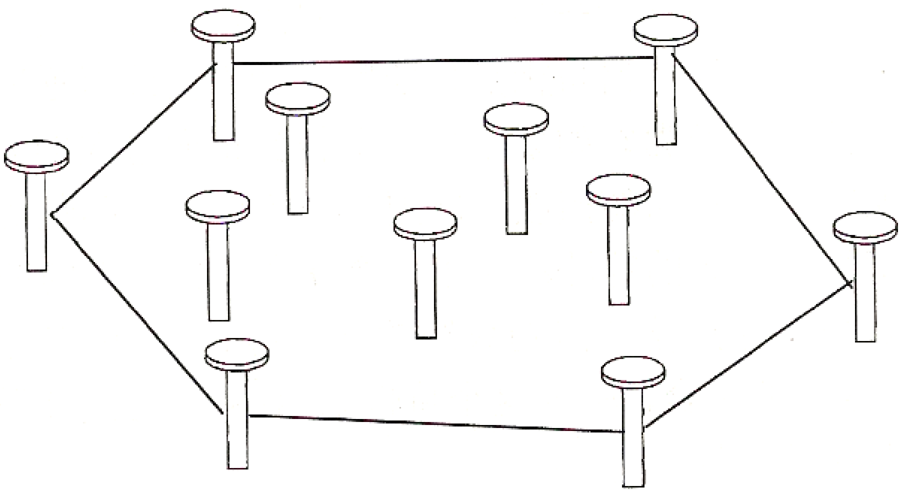
– 问题: 对于平面上n个点，找包围它们的最小凸多边形;

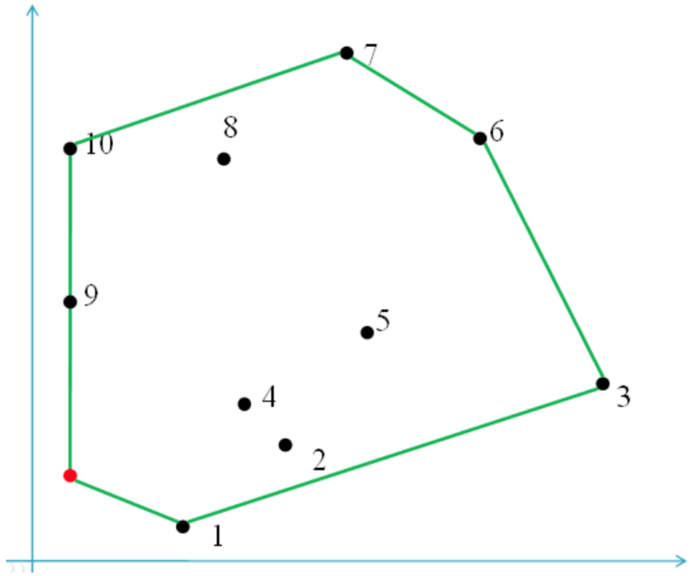
凸包问题

- 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。

想象，用橡皮筋去套一圈钉子

- 定义 一个点集合S的凸包是包含S的最小凸集合。





更多的蛮力算法例子:凸包

– 问题: 对于平面上n个点，找包围它们的最小凸多边形;

– 蛮力算法: 对于每对点 p_1和p_2， 判断是否所有其他点都在连接 p_1和p_2 的直线的同一侧;

– 效率:

凸包问题

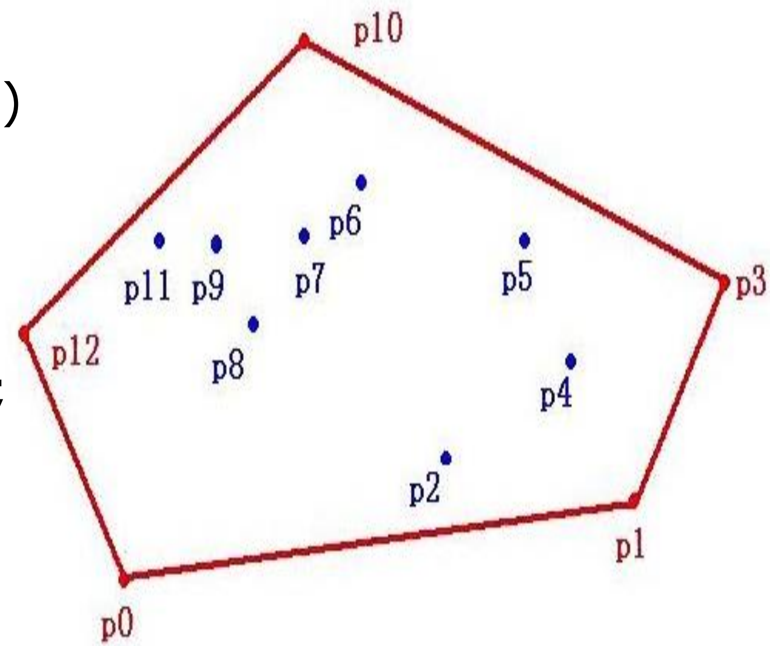
- 思路：两点确定一条直线，如果剩余的其它点都在这条直线的同一侧，则这两个点是凸包上的点，否则就不是。
- 步骤：
  1. 将点集里面的所有点两两配对，组成 n(n-1)/2 条直线。
  2. 对于每条直线，再检查剩余的 (n-2) 个点是否在直线的同一侧。
- 如何判断一个点 p3 是在直线 p1p2 的左边还是右边呢？（坐标：p1(x1,y1)，p2(x2,y2)，p3(x3,y3)）行列式求面积 (也就是我们常说的"叉积")

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

- *当上式结果为正时，p3在直线 p1p2 的左侧；当结果为负时，p3在直线 p1p2 的右边。*

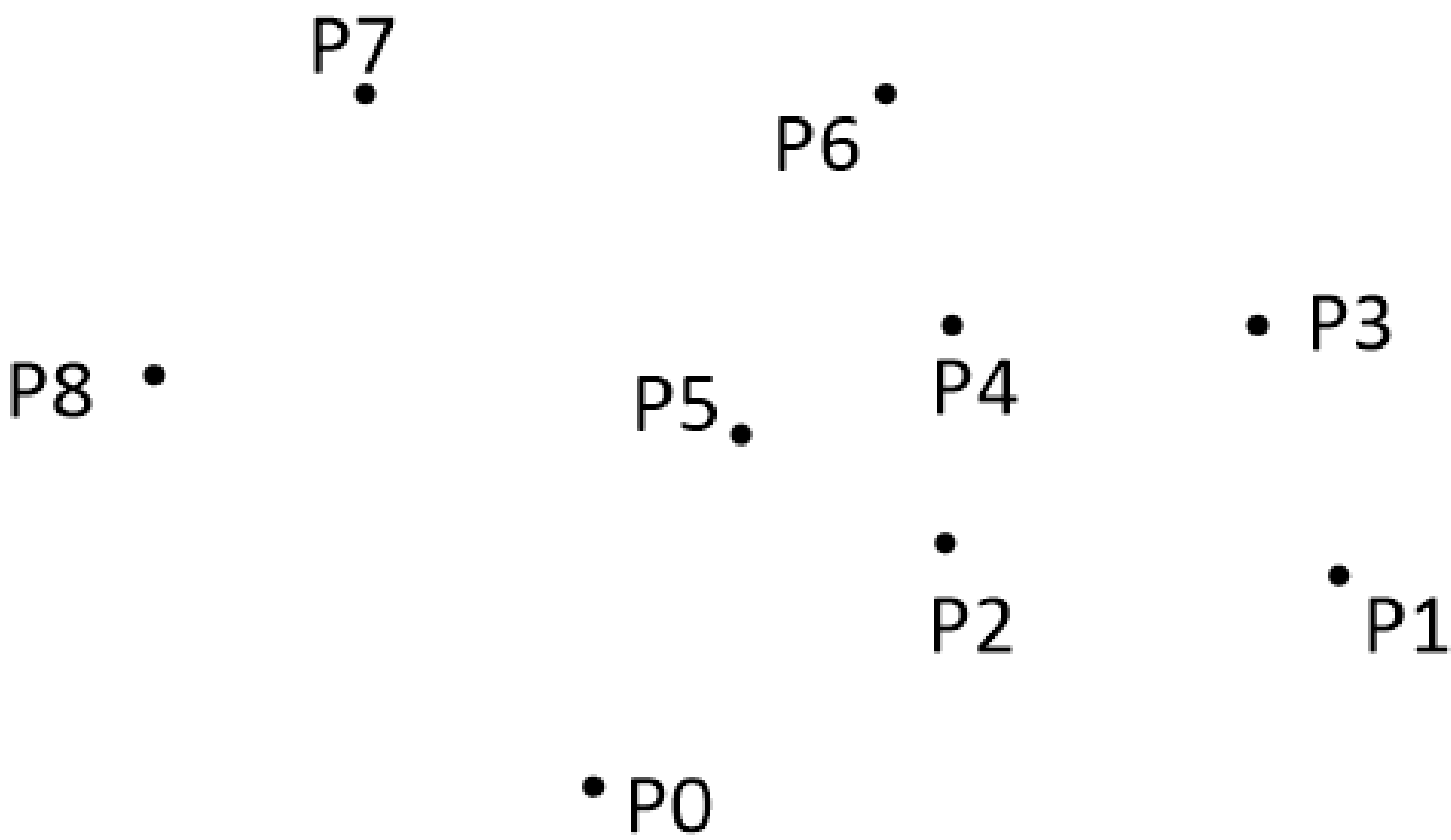
凸包问题

蛮力法解决凸包问题：
BruteForceConvexHull（P）
//输入：一个n个（n≥2）的点的列表P，$P_i=(X_i, Y_i)$
//输出：能够组成凸包的点的列表$Q_i=(X_i, Y_i)$
for i <— 1 to n-1 do
for j <— i+1 to n do
sign1 <— 0;sign2 <— 0;
$a = y_j - y_i;b = x_i - x_j;c = x_i*y_j - y_i*x_j;$
for k <— 1 to n do
if k=i||k=j continue
if axk+byk-c≥0 sign1++；
if axk+byk-c≤0 sign2--;
if sign2=2-n || sign1=n-2 record
the pole
return OK



蛮力算法的时间效率属于$O(n^3)$。如何降低为$O(n\log n)$?

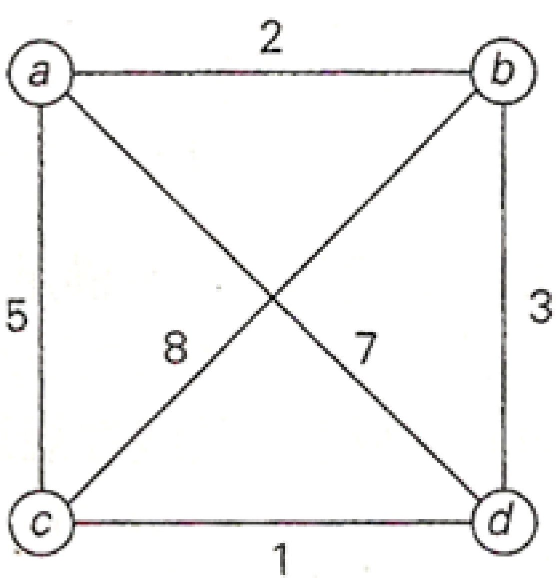
凸包问题



Graham扫描法

穷举查找

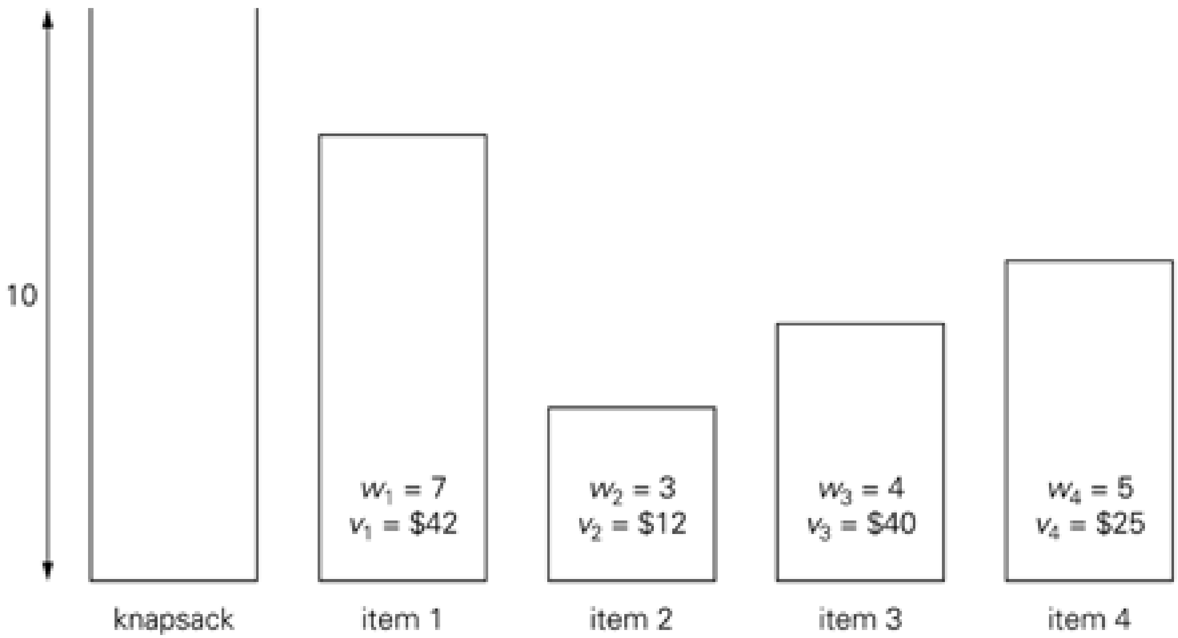
- 旅行商问题（Travelling salesman problem, TSP）
  - 一个商品推销员要去若干个城市推销商品，该推销员从一个城市出发，需要经过所有城市后，回到出发地。应如何选择行进路线，以使总的行程最短。



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

穷举查找

- 背包问题
  - 给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高？



Subset	Total weight	Total value
∅	0	\$ 0
{1}	7	\$42
{2}	3	\$12
{3}	4	\$40
{4}	5	\$25
{1, 2}	10	\$36
{1, 3}	11	not feasible
{1, 4}	12	not feasible
{2, 3}	7	\$52
{2, 4}	8	\$37
{3, 4}	**9**	**\$65**
{1, 2, 3}	14	not feasible
{1, 2, 4}	15	not feasible
{1, 3, 4}	16	not feasible
{2, 3, 4}	12	not feasible
{1, 2, 3, 4}	19	not feasible

穷举查找

- 分配问题
  - N个任务分配给n个人，任务j分配给人i的成本是C[i,j]

	任务1	任务2	任务3	任务4
人员1	9	2	7	8
人员2	6	4	3	7
人员3	5	8	1	8
人员4	7	6	9	4

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

任务:1, 2, 3, 4

人:<1, 2, 3, 4> cost = 9 + 4 + 1 + 4 = 18

<1, 2, 4, 3> cost = 9 + 4 + 8 + 9 = 30

<1, 3, 2, 4> cost = 9 + 3 + 8 + 4 = 24

<1, 3, 4, 2> cost = 9 + 3 + 8 + 6 = 26

<1, 4, 2, 3> cost = 9 + 7 + 8 + 9 = 33

<1, 4, 3, 2> cost = 9 + 7 + 1 + 6 = 23

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。

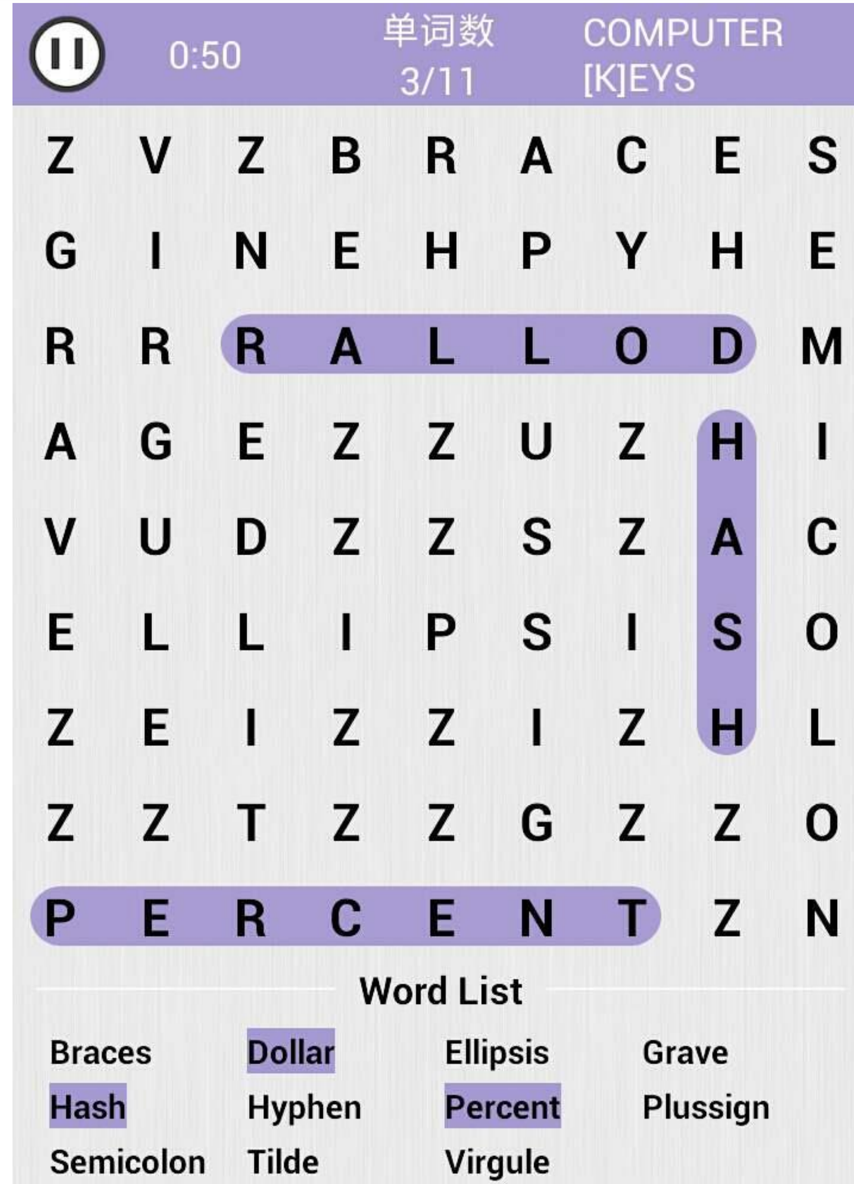
课下练习

- 找词游戏
- 幻方

找词游戏

找词”游戏是在美国流行的一种游戏，它要求游戏者从一张填满字母的正方形表中，找出包含在一个给定集合中的所有词。这些词可以竖着读（向上或向下）、横着读（从左或从右），或者沿45度对角线斜着读（4个方向都可以），但这些词必须是由表格中邻接的连续的单元格组成。遇到表格的边界时可以环绕，但方向不得改变，也不能折来折去。表格中的同一单元格可以出现在不同的词中，但在任一词中，同一单元格不得出现一次以上。

以下是一具体游戏事例，请结合这个实例为该游戏设计一个蛮力算法



幻方

幻方是我国古代的一种智力游戏，它是在N × N 的矩阵方格中填入1...N^2 的正整数， 使得其每行每列及对角线的和相等。

很明显，这个和对某一阶幻方是一个定值。设此值为S_N ， 则不难解出：$S_N = N^2 \cdot (N^2+1)/2N = N \cdot (N^2+1)/2$。

4	9	2
3	5	7
8	1	6

2	9	4
7	5	3
6	1	8