蛮力法

1. 概述
2. 查找问题中的蛮力法
3. 排序问题中的蛮力法
4. 组合问题中的蛮力法
5. 图问题中的蛮力法
6. 几何问题中的蛮力法

1 蛮力法的设计思想

蛮力法是指采用遍历（扫描）技术，即采用一定的策略将待求解问题的所有元素依次处理一次，从而找出问题的解。

依次处理所有元素是蛮力法的关键，为了避免陷入重复试探，应保证处理过的元素不再被处理。

关于蛮力法思考

- ✓ 蛮力法（枚举法、穷举法，暴力法）要求设计者找出所有可能的情况，然后选择其中一种情况，若该情况不可行（或不是最优解）则试探下一种可能的情况。
- ✓ 蛮力法是一种直接解决问题的方法，常常直接基于问题的描述和所设计的概念定义（候选解）。
- ✓ “力”——指计算机的能力，而不是人的智力。
- ✓ 蛮力法常常是最容易应用的方法。
  - ➢ 用连续整数检测算法计算GCD(m, n)

关于蛮力法思考

- ✓ 蛮力法不是一个最好的算法（巧妙和高效的算法很少出自蛮力），但当我们想不出更好的办法时，它也是一种有效的解决问题的方法。
- ✓ 它可能是惟一一种几乎什么问题都能解决的一般性方法，常用于一些非常基本、但又十分重要的算法。

蛮力法的优点

- 逻辑清晰，编写程序简洁
- 对于一些重要的问题（比如：排序、查找、矩阵乘法和字符串匹配），可以产生一些合理的算法
- 解决问题的实例很少时，可以花费较少的代价
- 可以解决一些小规模的问题（使用优化的算法没有必要，而且某些优化算法本身较复杂）
- 可以作为其他高效算法的衡量标准

使用蛮力法的几种情况

- 搜索所有的解空间
- 搜索所有的路径

一个简单的例子——百元买百鸡问题

“百钱买百鸡问题”算法的分析与改进

周　铝

（云南农业大学经济管理学院 650201　云南昆明）

1.百钱买百鸡问题及计算机求解

中国古代数学家张丘建在他的《算经》中提出了著名的“百钱买百鸡问题”：鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一，百钱买百鸡，问翁、母、雏各几何？利用计算机来解决“百钱买百鸡问题”，是程序设计语言中的一个经典的例子。百钱买百鸡问题的解决，基本思想是采用穷举法，即列举各种可能的买鸡情况，从中选出鸡的总数为 100 只，且买鸡的钱也刚好是 100 元的公鸡、母鸡和小鸡数，但是采用不同的思路，可以得到不同的算法，其效率也不尽相同。下面给出五种不同的算法，在这五种算法中，用分别 Cock、Hen 和 Chicken 表示公鸡数、母鸡数和小鸡数。

2.百钱买百鸡问题的几种算法设计

2.1 算法一设计

采用穷举法，可能买到的公鸡数从 0 到 19 只，母鸡数从 0 到 33 只，小鸡数从 0 到 100 只，在各种鸡数目的不同组合中找出那些既使鸡的总数为 100，又使购买费用为 100 的组合。采用三重循环来实现，具体算法如下：

```
void bqbj( )
{ int Cock,Hen,Chicken;
  for (Cock=0;Cock<20;Cock++)
    for (Hen=0;Hen<34;Hen++)
    for (Chicken=0;Chicken<100;Chicken++)
      if ((Cock+Hen+Chicken==100) && (Cock*5+Hen*3+Chicken/3==100)
&& (Chicken%3==0))
        printf ("Cock=%d  Hen=%d  Chicken=%d\n",Cock,Hen,Chick-
en);
}
```

2.2 算法二设计

在算法一中，对三种鸡的数目穷举时并没有考虑到它们之间的相互关系，实际上能够购买母鸡的最大数目是随着已购买的公鸡数的变化而变化的，当购买公鸡的数目确定后，能够购买母鸡的最大数目也就确定了。所以，算法二中，将第二重循环的次数改为随着第一重循环动态改变，减少第二重循环的总次数。

```
void bqbj( )
{ int Cock,Hen,Chicken;
  for (Cock=0;Cock<20;Cock++)
    for (Hen=0;Hen<(100-Cock*5)/3;Hen++)
      for (Chicken=0;Chicken<100;Chicken++)
        if ((Cock+Hen+Chicken)==100) && ((Cock*5+Hen*3+Chicken/3)==
100) && (Chicken%3==0))
          printf ("Cock=%d  Hen=%d  Chicken=%d\n",Cock,Hen,Chick-
en);
}
```

2.3 算法三设计

在算法二中，小鸡的数目是从 0 到 100，实际上“鸡雏三，值钱一”，在百钱买百鸡问题中，钱一是最小单位，不能再划分，所以，小鸡的数量应该是 3 的倍数，穷举小鸡数时，可以跳过不是 3 的倍数的小鸡数，在算法中使小鸡的循环步长值改为 3。

```
void bqbj( )
{ int Cock,Hen,Chicken;
  for (Cock=0;Cock<20;Cock++)
    for (Hen=0;Hen<(100-Cock*5)/3;Hen++)
    for (Chicken=3;Chicken<100;Chicken+=3)
      if ((Cock+Hen+Chicken)==100) && ((Cock*5+Hen*3+Chicken/3)==
100) && (Chicken%3==0))
        printf("Cock=%d  Hen=%d  Chicken=%d\n",Cock,Hen,Chick-
en);
}
```

2.4 算法四设计

前三种算法中，采用的基本思想都是穷举三种鸡的各种不同数量的组合，然后用两个限定条件（鸡的总数量为 100 且钱的数目也为 100）来筛选符合条件的买法，所以三种算法都采用了三重量循环来实现，然而，由于三种鸡的总数量为 100，所以，当两种鸡的数量确定后，第三种鸡的数量也就确定了，这时再采用买鸡的钱为 100 这个条件来筛选满足条件的买法即可，所以算法四只需要两重循环即可完成。

```
void bqbj( )
{ int Cock,Hen,Chicken;
  for (Cock=0;Cock<20;Cock++)
    for (Hen=0;Hen<(100-Cock*5)/3;Hen++)
    { Chicken=100-Cock-Hen;
      if ((Cock*5+Hen*3+Chicken/3==100) && (Chicken%3==0))
        printf("Cock=%d  Hen=%d  Chicken=%d\n",Cock,Hen,Chicken);
    }
}
```

2.5 算法五设计

根据算法四的思路，“百钱买百鸡”最终要求解的是符合两个条件的三种鸡的数量，三种鸡的数量是三个未知数，两个条件可以得到两个关于三个未知数的方程，这两个方程可构成一个方程组。

$$\begin{cases} Cock + Hen + Chicken = 100 \\ Cock \times 5 + Hen \times 3 + Chicken \div 3 = 100 \end{cases}$$

如果某一种鸡的数量已知，则只剩下两个未知数，根据这两个方程就可以解出两个未知数的值，现假设公鸡数量已知，解方程组可得：

$$\begin{cases} Hen = 25 - \frac{7}{4} Cock \\ Chicken = 75 + \frac{3}{4} Cock \end{cases}$$

所以算法五只用一重循环来确定公鸡的数量，母鸡和小鸡的数量由方程组来计算，但是纯数学的方法计算出来的未知数可能是一个实型数据、甚至是一个负数，而鸡的数量不可能为实型，也不可能是负数，所以最后要判断母鸡和小鸡的数量是否同为正数，此外，算法中应用 C 语言中将实型数据赋值给整型变量的方法来保证母鸡和小鸡的数量都是整数，然而这种取整使得母鸡数和小鸡数加上公鸡数可能就会不是 100 只，所以还要判断三种鸡的总数是否为整数 100。

```
void bqbj( )
{ int Cock,Hen,Chicken;
  for (Cock=0;Cock<20;Cock++)
    { Hen=25-7/4.0*Cock;
      Chicken=75+3/4.0*Cock;
      if  (Hen>=0 && Chicken>=0 && Chicken%3==0 && Cock+Hen+Chicken==
100)
          printf("Cock=%d  Hen=%d  Chicken=%d\n",Cock,Hen,Chicken);
    }
}
```

3.几种算法效率的比较分析

算法的效率主要包括时间效率及空间效率。时间效率也叫时间复杂度，它指的是算法中原操作重复执行的次数，空间效率也叫空间复杂度，它指的是算法占用的辅助存储空间的多少。下面从空间复杂度和时间复杂度两个方面对五个算法进行比较分析。

3.1 空间复杂度比较

在算法的空间复杂度上，五种算法都只使用了

算法二，原操作也是 if 语句，它也包含在一个三重循环中，最外层和最内层循环执行的次数和算法一相同，分别是 20 次和 101 次，但中间的循环次数随着最外层循环而改变，当最外层循环的 Cock 取值为 0 时，第二重循环执行的次数最多，为 (100-Cock*5)/3+1=(100-0*5)/3+1=34 次，当 Cock 取值为 1 时，第二重循环执行的次数为 (100-Cock*5)/3+1=(100-1*5)/3+1=32 次，依此类推，当最外层循环的 Cock 取值为 19 时，第二重循环执行的次数最少，为(100-Cock*5)/3+1=(100-19*5)/3+1=2 次，若取中间值 16 作为平均次数，则语句频度为 32320(20*16*101)。

算法三，原操作也是 if 语句，它也包含在一个三重循环中，最外层和第二重循环的次数和算法二相同，最内层循环控制变量的取值是 3 的倍数，依次为 3,6,9,……,96,99，所以执行的次数为 33 次，则语句频度为 10560(20*16*33)。

算法四，原操作为 Chicken=100-Cock-Hen 赋值语句和 if 语句，它们包含在双重循环中，外循环的执行次数为 20，内循环的执行次数和算法二的第二重循环相同，所以，也取 16 作平均次数，为方便讨论，现假设赋值语句和 if 语句占用的 CPU 时间相同，则语句频度为 640(20*16+20*16)。

算法五，原操作为两个赋值语句和一个 if 语句，它们包含在一个单重循环中，其循环次数为 20，假设赋值语句和 if 语句占用的 CPU 时间相同，则语句频度为 60(20+20+20)。

通过分析，可以看出，从算法一到算法五，虽然空间复杂度相同，但时间复杂度却不断减小。

通过分析可知，虽然五种算法的空间复杂度是相同的，但是算法一的时间效率是最低的，而算法五的时间效率是最高的。

4.结束语

通过对“百钱买百鸡”问题算法的分析可知，同一个问题，由于解决的思路不同，算法的性能也不同，我们应该从不同的角度分析解决问题，努力提高算法的效率

蛮力法解题步骤

根据问题中的条件将可能的情况一一列举出来，逐一尝试从中找出满足问题条件的解。

但有时一一列举出的情况数目很大，如果超过了我们所能忍受的范围，则需要进一步考虑，排除一些明显不合理的情况，尽可能减少问题可能解的列举数目。

用蛮力法解决问题，通常可以从两个方面进行算法设计：

1）找出枚举范围：分析问题所涉及的各种情况。

2）找出约束条件：分析问题的解需要满足的条件，并用逻辑表达式表示。

思考下面问题：找出枚举范围和约束条件

求所有的三位数,它除以11所得的余数等于它的三个数字的平方和.

思路:

枚举范围：100—999，共900个。

约束条件：设三位数的百位、十位、个位的数字分别为x，y，z。则有$x^2+y^2+z^2\leq 10$,进而$1\leq x\leq 3$, $0\leq y\leq 3$, $0\leq z\leq 3$。

解：所求三位数必在以下数中：

100，101，102，103，110，111，112，

120，121，122，130，200，201，202，

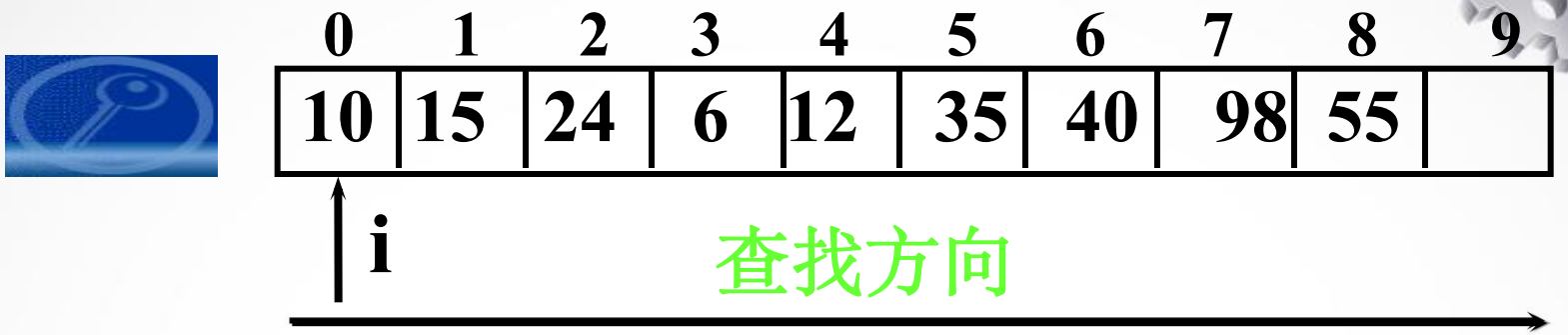
211，212，220，221，300，301，310。

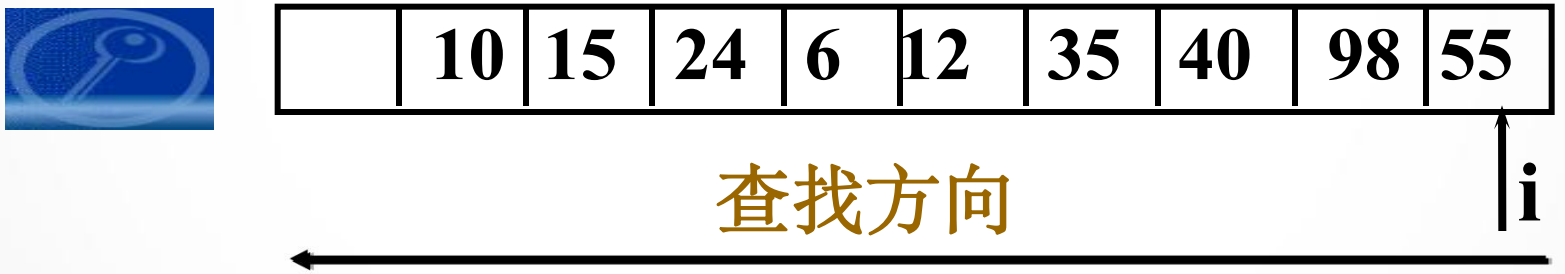
不难验证只有100，101两个数符合要求。

2 查找问题中的蛮力法—顺序查找

思路：顺序查找从表的一端向另一端逐个将元素与给定值进行比较，若相等，则查找成功，给出该元素在表中的位置；若整个表检测完仍未找到与给定值相等的元素，则查找失败，给出失败信息。

2 查找问题中的蛮力法—顺序查找





2 查找问题中的蛮力法—顺序查找

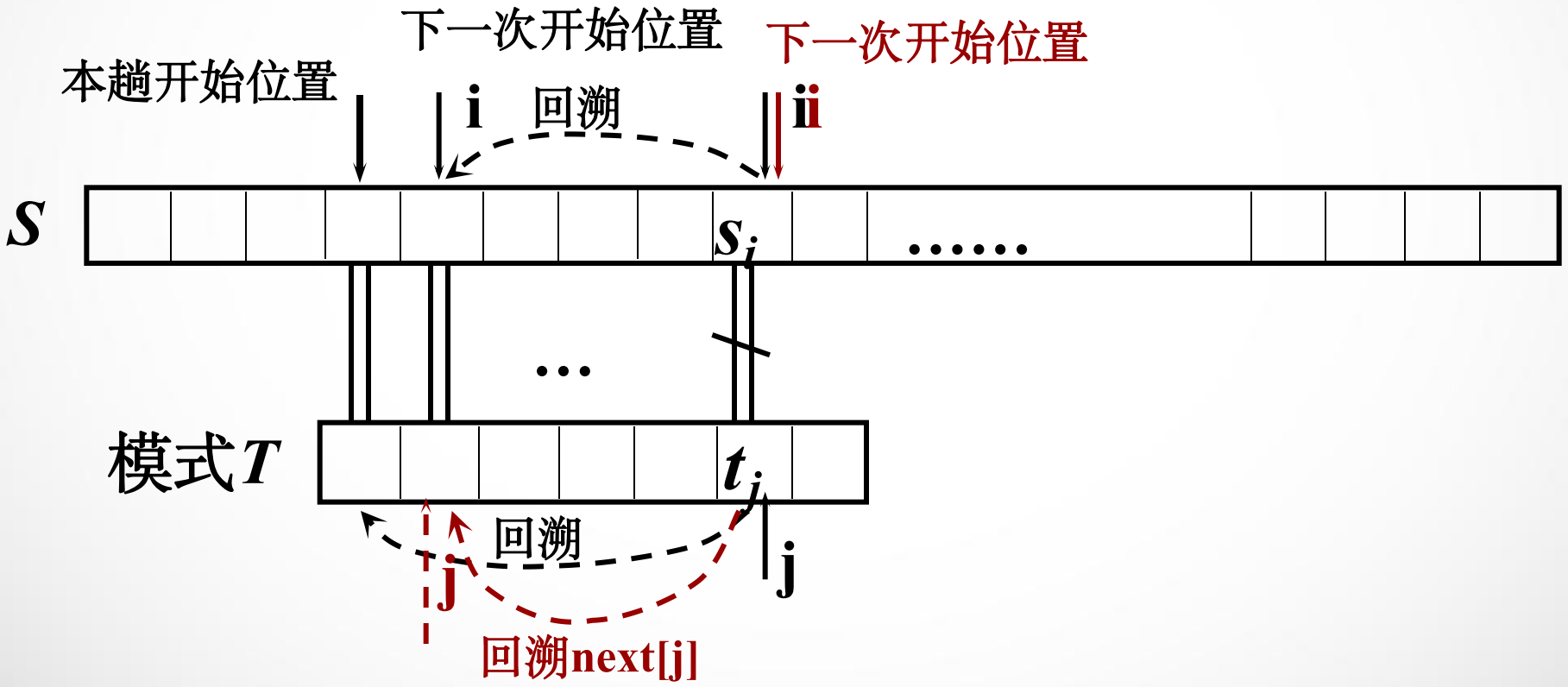
```
int SeqSearch1(int r[ ], int n, int k)
{   i=n;
    while (i>0 && r[i]!=k)
    i--;
   return i;
}
```

$\sum_{i=1}^{n} 1$

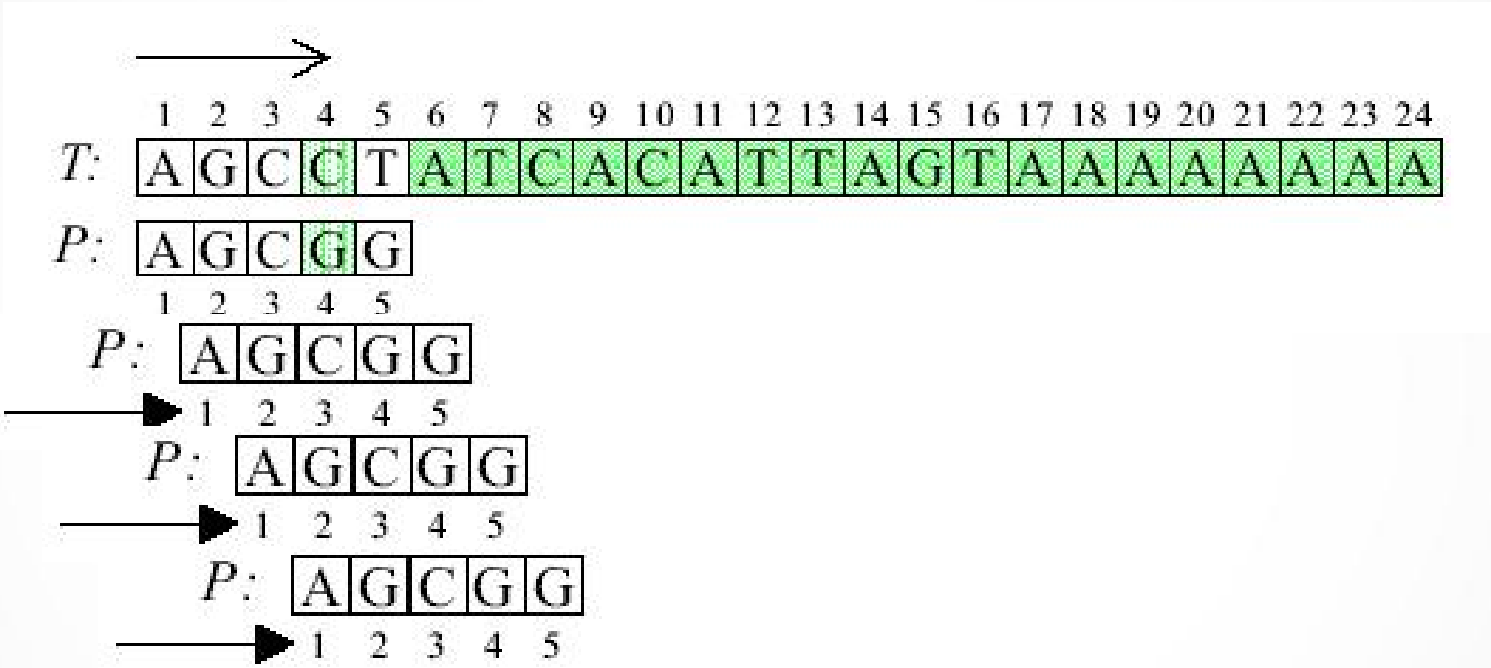
```
  int SeqSearch2(int r[ ], int n, int k)
  { r[0]=k;
     i=n;
    while (r[i]!=k)
    i--;
    return i;
}
```

2 查找问题中的蛮力法—串的匹配

BF算法　　**KMP算法**

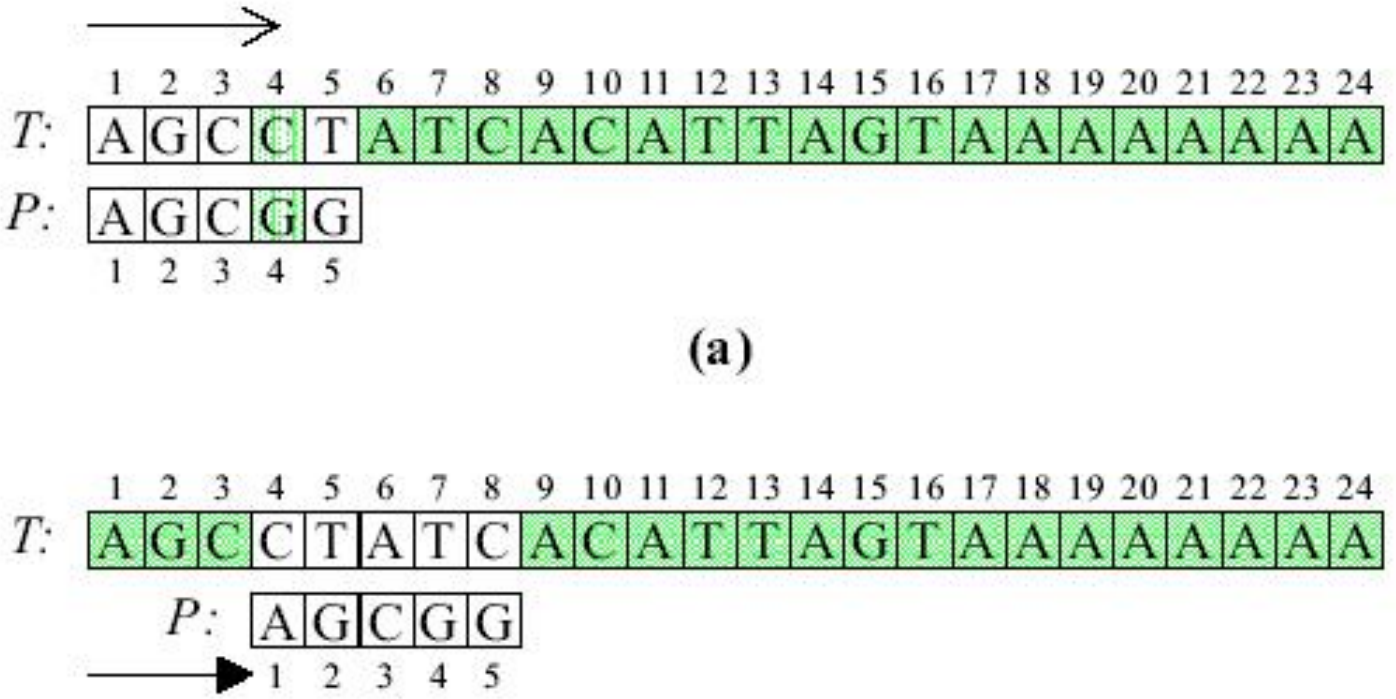


BF 算法


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
T: A G C C T A T C A C A T T A G T A A A A A A A A
P: A G C G G
1 2 3 4 5
P: A G C G G
1 2 3 4 5
P: A G C G G
1 2 3 4 5
P: A G C G G
1 2 3 4 5

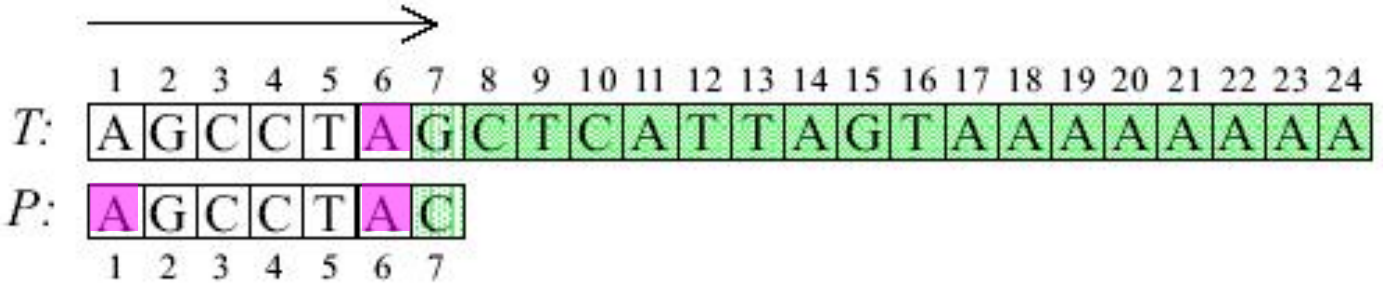
KMP 算法：第一种情况

- The first symbol of P does not appear in P again.
- We can slide to T_4, since $T_4 \neq P_4$ in (a).

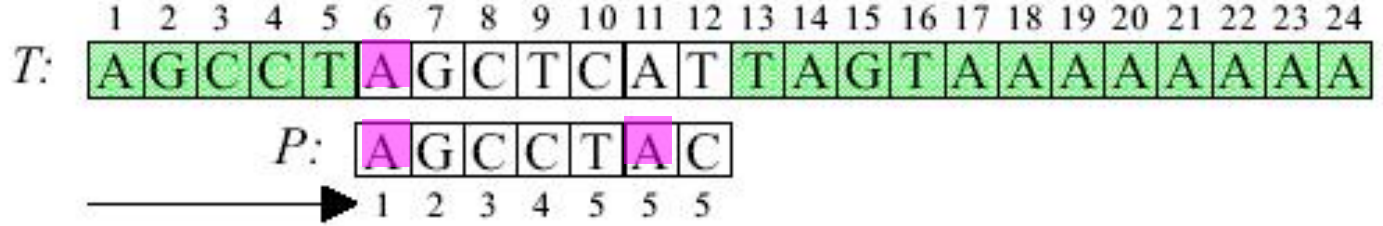


KMP 算法：第二种情况

- The first symbol of P appears in P again.
- $T_7 \neq P_7$ in (a). We have to slide to T_6, since $P_1 = P_6 = T_6$.



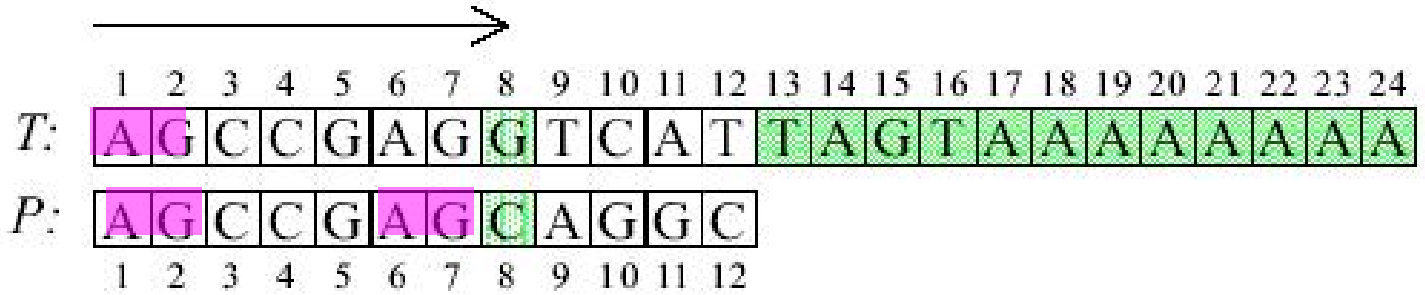
(a)



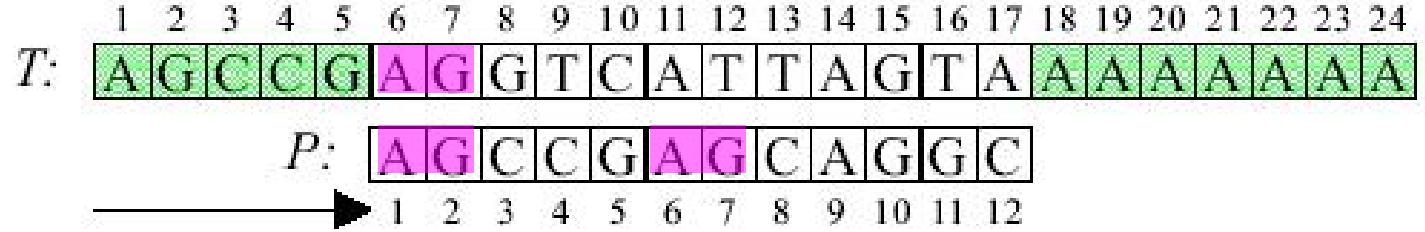
(b)

KMP 算法：第三种情况

- The prefix of P appears in P again.
- $T_8 \neq P_8$ in (a). We have to slide to T_6, since $P_{1,2}=P_{6,7}=T_{6,7}$.



(a)



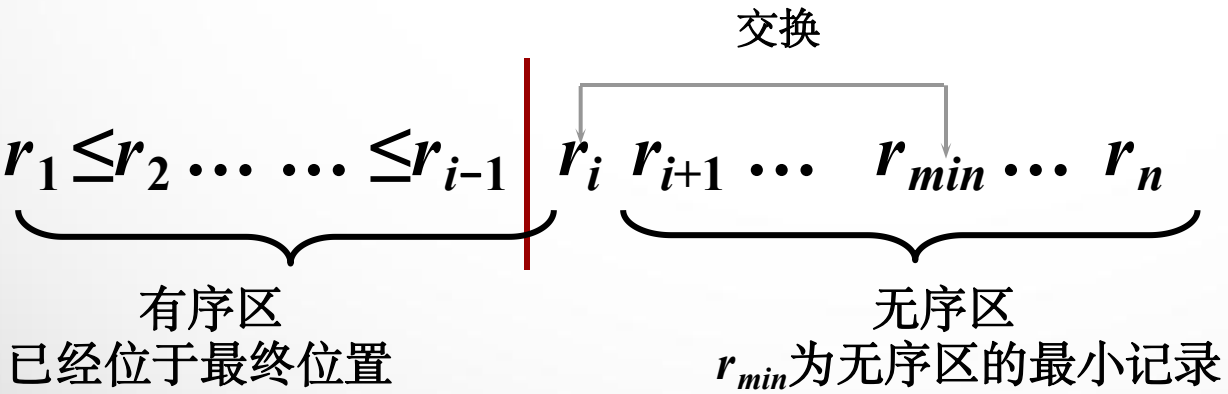
(b)

3 排序问题中的蛮力法—选择排序

选择排序开始的时候，扫描整个序列，找到整个序列的最小记录和序列中的第一个记录交换，从而将最小记录放到它在有序区的最终位置上，

然后再从第二个记录开始扫描序列，找到n-1个序列中的最小记录，再和第二个记录交换位置。

一般地，第i趟排序从第i个记录开始扫描序列，在$n-i+1$（$1 \le i \le n-1$）个记录中找到关键码最小的记录，并和第i个记录交换作为有序序列的第i个记录。



3 排序问题中的蛮力法—选择排序

```
void SelectSort(int r[ ], int n)
{
    for (i=1; i<=n-1; i++)
    {
      index=i;
      for (j=i+1; j<=n; j++)
            if (r[j]<r[index]) index=j;
      if (index!=i) r[i]←→r[index];
    }
 }
```

3 排序问题中的蛮力法—起泡排序

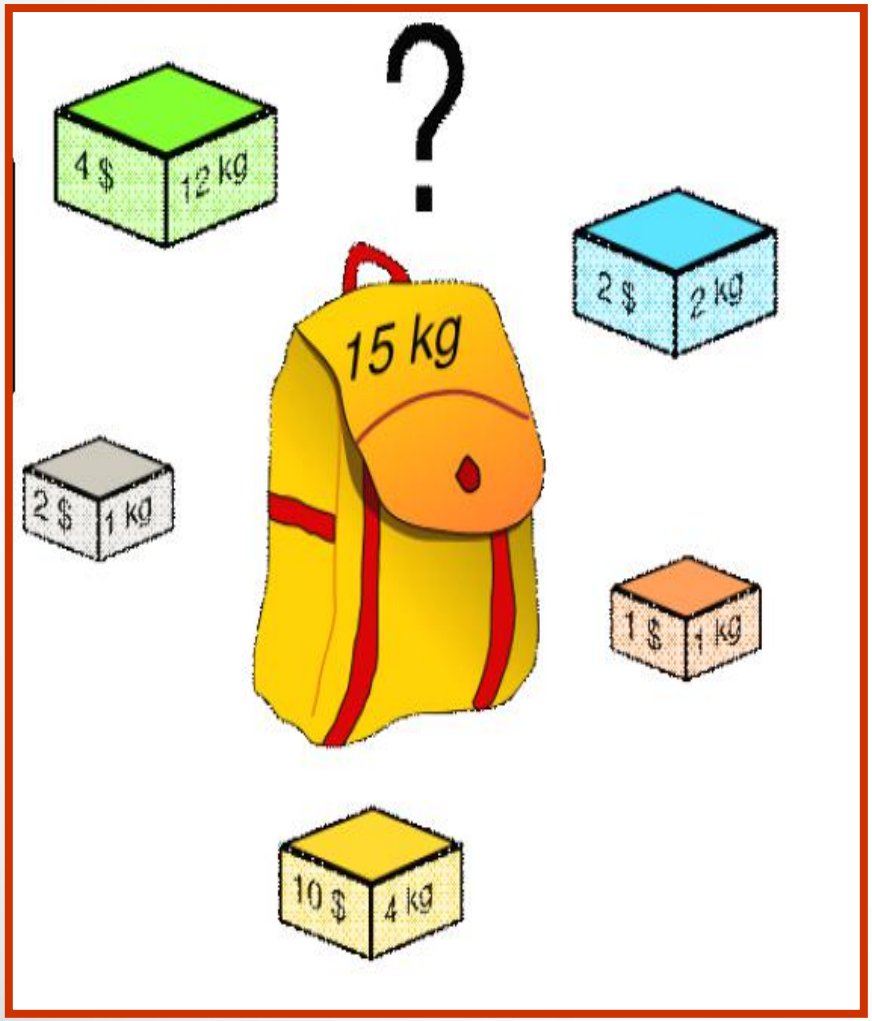
```
void Bubble1(int r[ ], int n)
{
    for (i=1; i<=n-1; i++)
        for (j=1; j<=n-i; j++)
            if (r[j]>r[j+1])
                r[j]←→r[j+1];
}
```

```
void Bubble3(int r[ ], int n)    {
    exchange=n;
    while (exchange)
    {
        bound=exchange;
        exchange=0;
        for (j=1; j<bound; j++)
            if (r[j]>r[j+1])
            {
                r[j]←→r[j+1];
                exchange=j;
            }
    }
}
```

蛮力法一般观点

用蛮力法设计的算法，一般来说，经过适度的努力后，都可以对算法的第一个版本进行一定程度的改良，改进其时间性能，但只能减少系数，而数量级不会改变。

4 组合问题中的蛮力法—0/1背包问题



问题描述：给定n个重量为$\{w_1, w_2, \cdots, w_n\}$、价值为$\{v_1, v_2, \cdots, v_n\}$的物品和一个容量为$C$的背包，求这些物品中的一个最有价值的子集，且要能够装到背包中。

4 组合问题中的蛮力法—0/1背包问题

对于一个具有n个元素的集合,其子集数量是2^n,所以,不论生成子集的算法效率有多高,蛮力法都会导致一个$\Omega(2^n)$的算法.

序号	子集	重量	价值	序号	子集	重量	价值
1				17			
2				18			
3				19			
4				20			
5				21			
6				22			
7				23			
8				24			
9				25			
10				26			
11				27			
12				28			
13				29			
14				30			
15				31			
16				32			

4 组合问题中的蛮力法—任务分配问题

问题描述：假设有n个任务需要分配给n个人执行，每个任务只分配给一个人，每个人只分配一个任务，且第j个任务分配给第i个人的成本是$C[i,j]$（$1 \leqslant i, j \leqslant n$），任务分配问题要求找出总成本最小的分配方案。

	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

4 组合问题中的蛮力法—任务分配问题

	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

可以用一个 n 元组$(j_1, j_2, \ldots, j_n)$来描述任务分配问题的一个可能解，其中第 i 个分量j_i $(1 \le i \le n)$ 表示在第 i 行中选择的列号，因此用蛮力法解决任务分配问题要求生成整数1~n的全排列，然后把成本矩阵中相应元素相加来求得每种分配方案的总成本，最后选出具有最小和的方案。

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

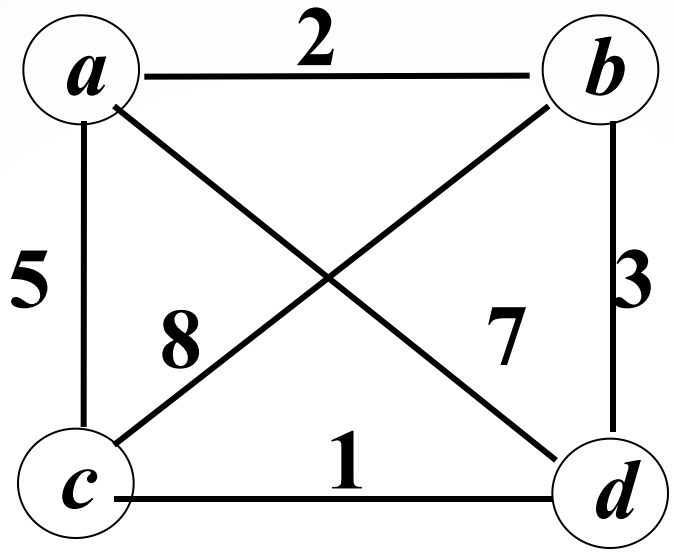
<1, 2, 3, 4> cost = 9 + 4 + 1 + 4 = 18
<1, 2, 4, 3> cost = 9 + 4 + 8 + 9 = 30
<1, 3, 2, 4> cost = 9 + 3 + 8 + 4 = 24
<1, 3, 4, 2> cost = 9 + 3 + 8 + 6 = 26
<1, 4, 2, 3> cost = 9 + 7 + 8 + 9 = 33
<1, 4, 3, 2> cost = 9 + 7 + 1 + 6 = 23

5 图问题中的蛮力法—TSP问题

- TSP问题是指旅行家要旅行n个城市然后回到出发城市，要求各个城市经历且仅经历一次，并要求所走的路程最短。该问题又称为货郎担问题、邮递员问题、售货员问题，是图问题中最广为人知的问题。
- 用蛮力法解决TSP问题，可以找出所有可能的旅行路线，从中选取路径长度最短的简单回路。
- 求解: 一个加权连通图中的最短哈密顿回路问题。

TSP问题有着简单的表述、重要的应用、以及和其他NP完全问题的重要关系，它在近100年的时间里强烈地吸引着计算机科学工作者.

5 图问题中的蛮力法—TSP问题



序号	路径	路径长度	是否最短
1	$a \to b \to c \to d \to a$	18	否
2	$a \to b \to d \to c \to a$	11	是
3	$a \to c \to b \to d \to a$	23	否
4	$a \to c \to d \to b \to a$	11	是
5	$a \to d \to b \to c \to a$	23	否
6	$a \to d \to c \to b \to a$	18	否

蛮力法求解TSP问题存在的问题

注意到图中有3对不同的路径，对每对路径来说，不同的只是路径的方向，因此，可以将这个数量减半，则可能的解有$(n-1)!/2$个。随着n的增长，TSP问题的可能解也在迅速地增长。

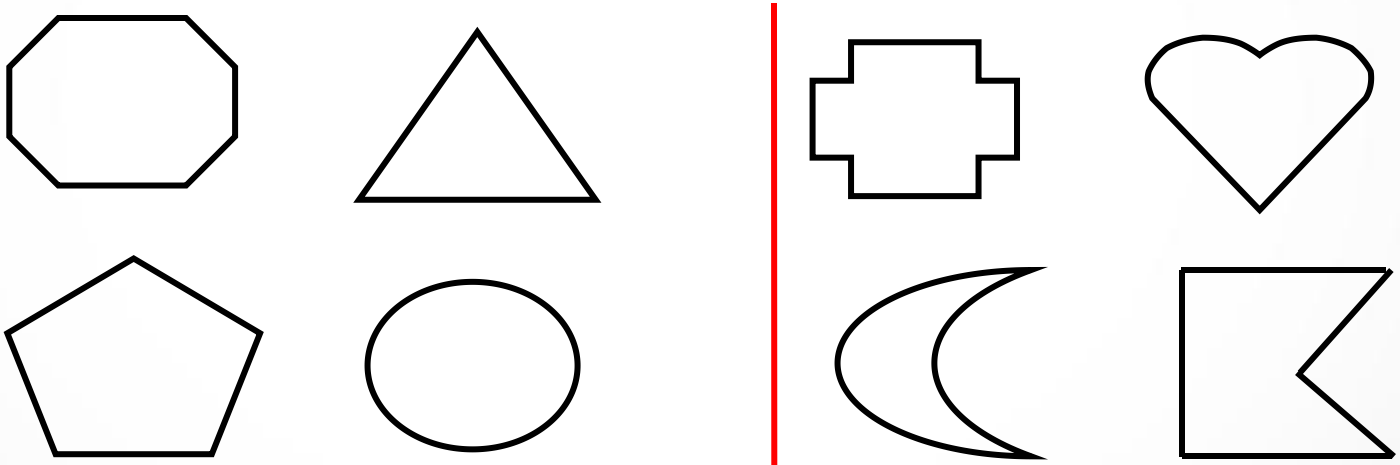
- 一个10城市的TSP问题有大约180,000个可能解。
- 一个20城市的TSP问题有大约

  60,000,000,000,000,000个可能解。
- 一个50城市的TSP问题有大约10^{62}个可能解，而一个行星上也只有10^{21}升水。

蛮力法求解TSP问题，只能解决问题规模很小的实例。

6 几何问题中的蛮力法—凸包问题

对于平面上的一个点的有限集合，如果以集合中任意两点P和Q为端点的线段上的点都属于该集合，则称该集合是凸集合。



一个点集*S*的凸包是包含*S*的最小凸集合，其中，最小是指*S*的凸包一定是所有包含S的凸集合的子集。

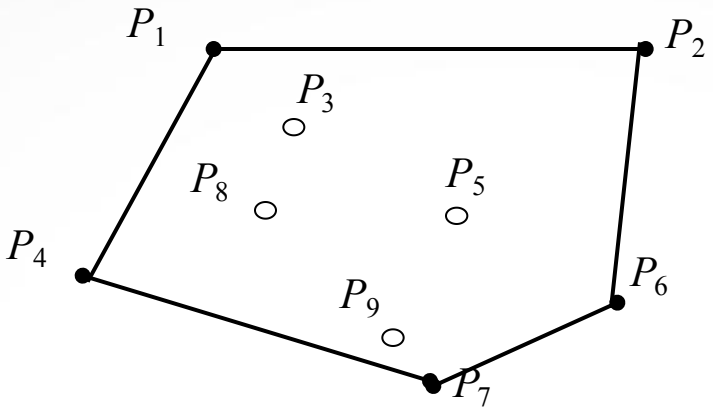
凸包问题的相关定理

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；如果所有点都位于一条直线上，则凸多边形退化为一条线段。

凸包问题是为一个具有n个点的集合构造凸多边形的问题。

极点：对于任何以凸集合中的点为端点的线段来说，它不是这种线段中的点。

6 几何问题中的蛮力法—凸包问题



对于一个由n个点构成的集合S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（假定不存在三点同线的情况），他们的连线是该集合凸包边界的一部分。对每一对顶点都检验一遍后，满足条件的线段构成了该凸包的边界。

在平面上，穿过两个点(x_1, y_1)和(x_2, y_2)的直线是由下面的方程定义的：

$$ax + by = c \quad (\text{其中，} \ a=y_2-y_1, b=x_1-x_2, c=x_1y_2-y_1x_2)$$