三 、基本的算法策略

4、贪婪算法

- 贪婪法又叫登山法，它的根本思想是逐步到达山顶，即逐步获得最优解。选择的贪婪策略要具有无后向性。某状态以后的过程不会影响以前的状态，只与当前状态或以前的状态有关，称这种特性为无后效性。

4、贪婪算法

- 4.1 可绝对贪婪问题
- 4.2 相对或近似贪婪问题
- 4.3 关于贪婪策略讨论

4.1 可绝对贪婪问题

- 若对于输入的任何数据贪婪策略都是适用的，则称为“可绝对贪婪问题”。
- 例：数列极差问题：在黑板上写N个正整数形成的一个数列，进行如下操作:每一次擦去其中的两个数a和b，然后在数列中加入一个新的数a×b+1，如此下去直至黑板上剩下一个数，该数列在所有按这种操作方式最后得到的数中，最大的记作max，最小的记作min，则该数列的极差定义为M=max-min。

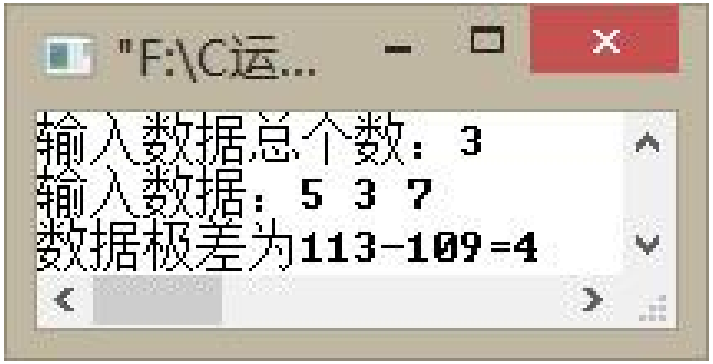
4.1 可绝对贪婪问题

- 问题分析：
  - 通过实例来认识题目中描述的计算过程。对三个具体的数据3，5，7讨论，可能有以下三种结果：

（3*5+1）*7+1=113

（3*7+1）*5+1=111

（5*7+1）*3+1=109



  - 由此可见，先运算小数据得到的是最大值，先运算大数据得到的是最小值。

4.1 可绝对贪婪问题

- 问题分析：
  - 再以三个数为例证明此题用贪心策略求解的合理性，假设：a<b<c，b=a+k1，c=a+k1+k2，其中k1，k2>0，则有以下几种组合计算结果：

(1) (a*b+1)*c+1=a^3+(2k1+k2)a^2+($k1^2$+k1k2+1)*a+k1+k2+1 ← 最大

(2) (a*c+1)*b+1=a^3+(2k1+k2)a^2+($k1^2$+k1k2+1)*a+k1+1

(3) (b*c+1)*a+1=a^3+(2k1+k2)a^2+($k1^2$+k1k2+1)*a+1 ← 最小

  - 可知此问题适合用贪婪策略，在求最大值时，要先选择较小的数操作。而求最小值时，要先选择较大的数操作。这是通过两次运用贪心策略解决的问题。

4.1 可绝对贪婪问题

- 算法设计：
  - 这个问题的解决方法和哈夫曼树的构造过程相似，不断从现有的数据中，选取最大的两个数和最小的两个数，计算后的结果继续参与运算，直到剩余一个数算法结束。
  - 选取最大的两个数和最小的两个数较高效的算法是用二分法完成，这里仅仅用简单的逐个比较的方法来求解。 由于找到的两个数将不再参与其后的运算，其中一个用它们的计算结果代替，另一个用当前的最后一个数据覆盖即可。所以不但要选取最大和最小，还必须记录它们的位置，以便将其覆盖。
  - 求max、min的过程必须独立，因为存在替代和覆盖的过程，必须用两个数组同时存储初始数据。

1	2	3	4	5	6
10	4	3	1132	111	1

最大s1= 1 4 4

次大s2= 2 1 5

```
a[s1]=a[s1]*a[s2]+1;
a[s2]=a[n];
n--;
```

1 2 3 4 5

110 4 3 113331 1

最大s1= 1 4

次大s2= 2 1

```
a[s1]=a[s1]*a[s2]+1;

a[s2]=a[n];

n--;
```

4.1 可绝对贪婪问题

- 算法实现：

```
int s1,s2;
max2(int a[],int n)
{
   int j;
   if(a[1]>=a[2]) //前两个数中最大数下标记录到s1，次大数下标记录到s2
   {
        s1=1;
        s2=2;
   }
   else
   {
        s1=2;
        s2=1;
   }
```

4.1 可绝对贪婪问题

```
for(j=3;j<=n;j++)  //从第三个数开始逐个比较，最大数下标记录到s1，次大数下标记录到s2
        if (a[j]>a[s1])
        {
                s2=s1;
                s1=j;
        }
        else
        {
                if(a[j]>a[s2])
                        s2=j;
        }
}
```

4.1 可绝对贪婪问题

```
int calmin(int a[],int n)
{
   int j;
   while(n>2)
    {
        max2(a,n);
        a[s1]=a[s1]*a[s2]+1;  //a×b+1的结果代替a
        a[s2]=a[n];   //最后一个数代替b
        n--;  //数据个数减一（忽略最后一个数）
   }
   return(a[1]*a[2]+1);  //只剩两个数时返回计算结果
}
```

4.1 可绝对贪婪问题

```
min2(int a[],int n)
{
   int j;
   if(a[1]<=a[2])
   {
        s1=1;
        s2=2;
   }
   else
   {
        s1=2;
        s2=1;
   }
```

4.1 可绝对贪婪问题

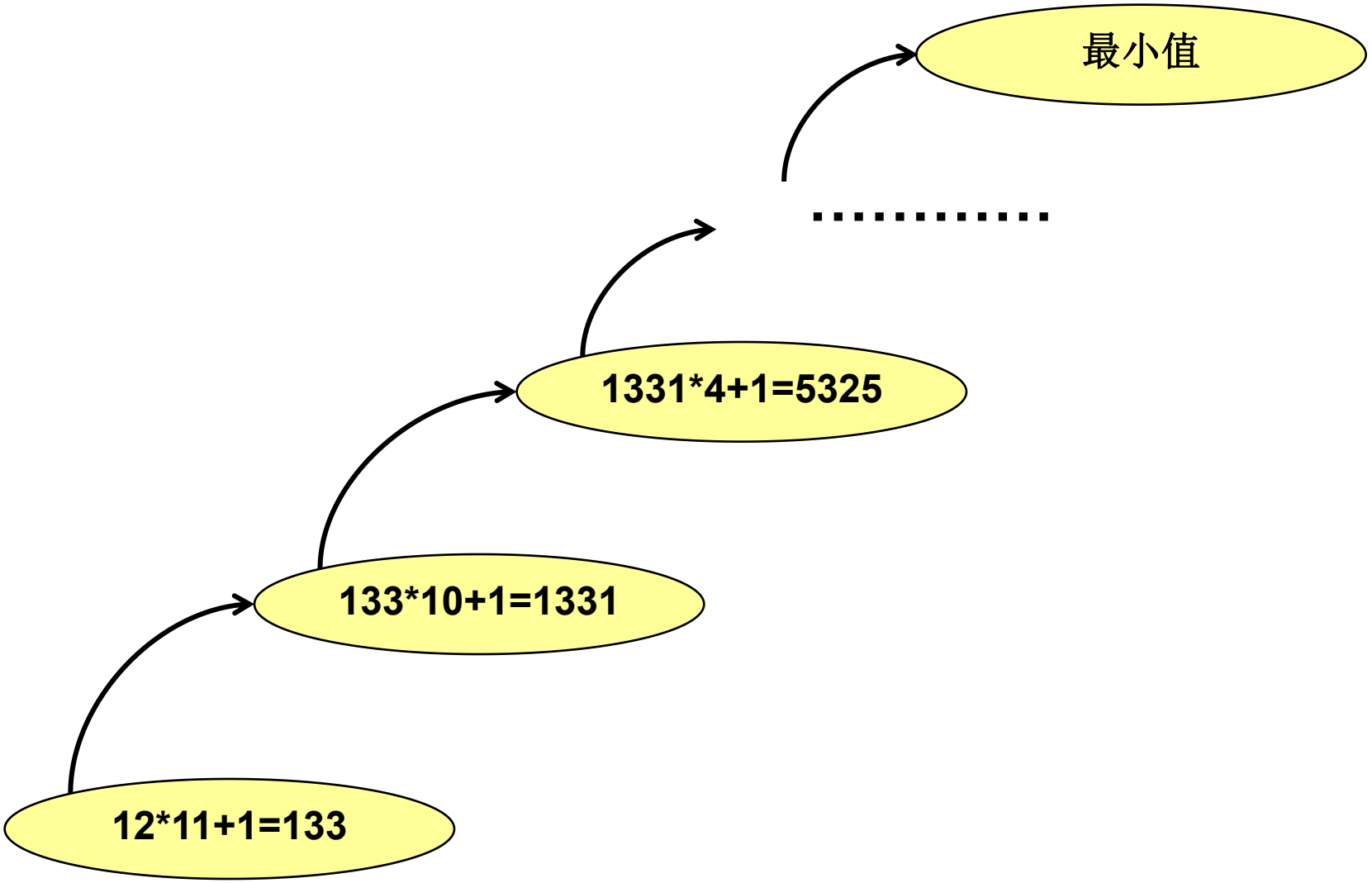
```
    for (j=3;j<=n;j++)
        if (a[j]<a[s1])
        {
                s2=s1;
                s1=j;
        }
        else
        {
                if(a[j]<a[s2])
                        s2=j;
        }
}
```

4.1 可绝对贪婪问题

```
int calmax(int a[],int n)
{
   int j;
   while(n>2)
   {
        min2(a,n);
        a[s1]=a[s1]* a[s2]+1;
        a[s2]=a[n];
        n--;
   }
   return(a[1]*a[2]+1);
}
```

4.1 可绝对贪婪问题

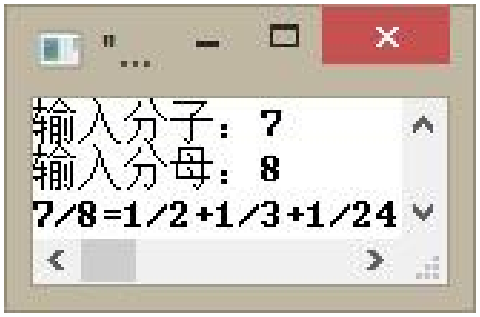
```
int main()
{
   int j,n,a[100],b[100],max,min;
   printf("输入数据总个数：");
   scanf("%d",&n);
   printf("输入数据：");
   for(j=1;j<=n;j++)
   {
        scanf("%d",&a[j]);
        b[j]=a[j];
   }
   min=calmin(a,n);
   max=calmax(b,n);
   printf("数据极差为%d-%d=%d\n", max,min,max-min);
   return 0;
}
```

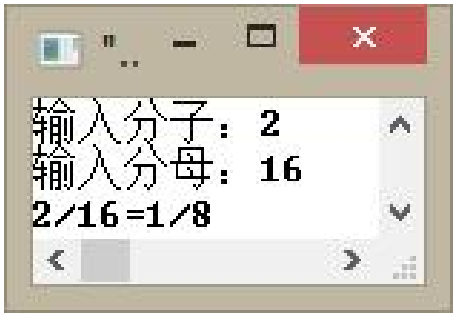


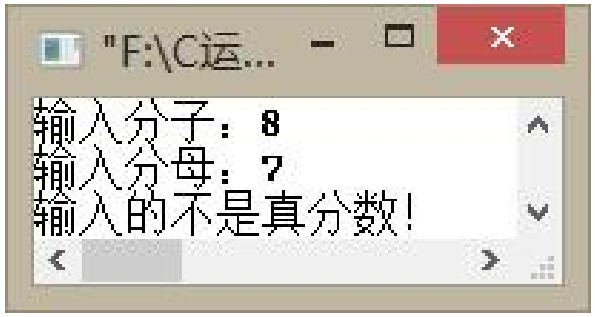
- 贪婪法（登山法）：逐步到达山顶, 即逐步获得最优解。
- 无后效性：某状态只与当前状态或以前的状态有关 。

4.1 可绝对贪婪问题

- 贪婪策略不仅可以应用于最优化问题中，有时在解决构造类问题时，用这种策略可以尽快地构造出一组解。
- 例：设计一个算法，把一个真分数表示为埃及分数之和的形式。埃及分数是指分子为1的形式，如：7/8=1/2+1/3+1/24。







4.1 可绝对贪婪问题

- 问题分析：
  - 基本思想是逐步选择分数所包含的最大埃及分数，这些埃及分数之和就是问题的一个解。

    如：7/8中最大埃及分数为1/2，

    7/8-1/2中最大埃及分数为1/3，

    7/8-1/2-1/3=1/24分解完毕。

4.1 可绝对贪婪问题

- 数学模型：
  - 真分数F=A/B，则B/A运算后，商为D，余数为K,0＜K＜A，则：
  - B=A*D+K --→ B/A=D+K/A 因为D+K/A＜D+1
    --→ B/A＜D+1
    --→ A/B＞1/(D+1)
  - 设C=D+1，则A/B＞1/C， 1/C就是真分数F所包含的最大埃及分数。
  - 进一步计算：A/B-1/C=（A*C-B）/（B*C）
  - 则算法继续要解决的是有关分子为A=A*C-B,分母为B=B*C的问题。

4.1 可绝对贪婪问题

- 算法设计：
  - (1)设某个真分数的分子为A(A≠1），分母为B；
  - (2)把B除以A的商的整数部分加1后的值作为埃及分数的一个分母C;
  - (3)输出1/C；
  - (4)将A*C-B作为新的A；
  - (5)将B*C作为新的B；
  - (6)如果A≠1且能整除B,则最后一个分母为B/A；
  - (7)如果A＝1,则最后一个分母为B;否则转步骤(2)。

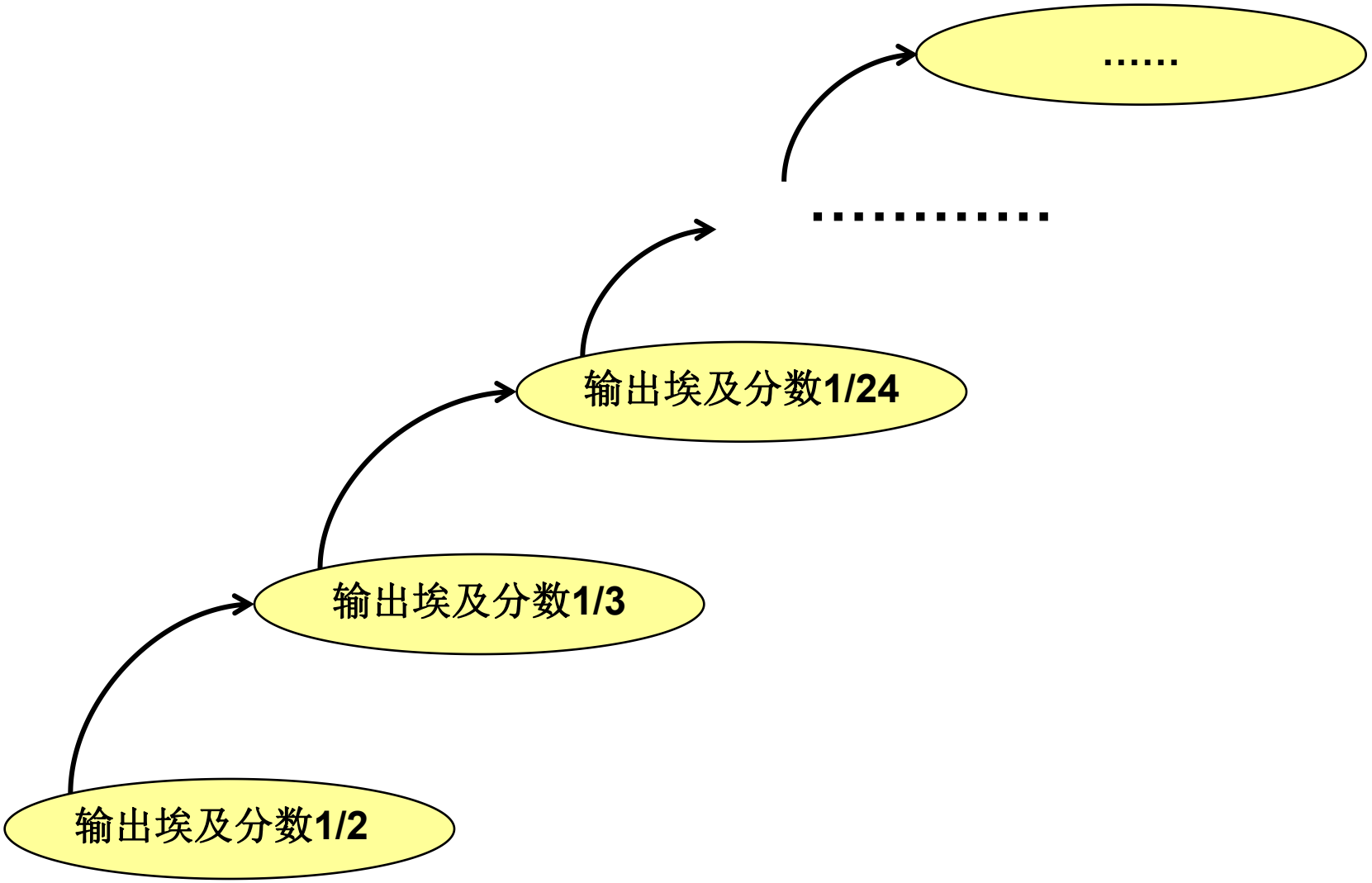
4.1 可绝对贪婪问题

- 算法设计：
  - 举例验证真分数（A/B）7/8:
  - 第一个埃及分数分母C=B/A+1=8/7+1=2，　输出：1/2

    新的A=A*C-B=7*2-8=6，新的B=B*C=8*2=16；
  - 第二个埃及分数分母C=B/A+1=16/6+1=3，　输出：1/3

    新的A=A*C-B=6*3-16=2，新的B=B*C=16*3=48；
  - A≠1且能整除B,最后一个埃及分数分母C=B/A=48/2=24；

    输出：1/24



- 贪婪法（登山法）：逐步到达山顶, 即逐步获得最优解。
- 无后效性：某状态只与当前状态或以前的状态有关 。

4.1 可绝对贪婪问题

- 算法实现：

```
main()
{
   int a,b,c;
    printf("输入分子：");
    scanf("%d",&a);
    printf("输入分母：");
    scanf("%d",&b);
    if(b<a)
         printf("输入的不是真分数！\n");
    else
        if(a==1||b%a==0)
                printf("%d/%d=1/%d\n",a,b,b/a);
```

4.1 可绝对贪婪问题

```
        else
        {
                printf("%d/%d=",a,b);
                while(a!=1)
                {
                        c=b/a+1;
                        a=a*c-b;
                        b=b*c;
                        printf("1/%d",c);
                        if(b%a==0)
                        {
                                printf("+1/%d",b/a);
                                a=1;
                        }
                        if(a>1)
                                printf("+");
                }
        }
}
```

4.2 相对或近似贪婪问题

- 例：币种统计问题：某单位给每个职工发工资（精确到元）。为了保证不要临时兑换零钱，且取款的张数最少，取工资前要统计出所有职工的工资所需各种币值(100,50,20,10,5,2,1元共七种)的张数。请编程完成。

4.2 相对或近似贪婪问题

- 算法设计：
  - 从键盘输入每人的工资。
  - 对每一个人的工资，用“贪婪”的思想，先尽量多地取大面额的币种，由大面额到小面额币种逐渐统计。
  - 利用数组应用技巧,将七种币值存储在数组B。七种币值就可表示为B[i],i=1,2,3,4,5,6,7。为了能实现贪婪策略，七种币应该从大面额的币种到小面额的币种依次存储。
  - 利用数组技巧,设置一个有7个元素的累加器数组S。

4.2 相对或近似贪婪问题

- 算法实现：

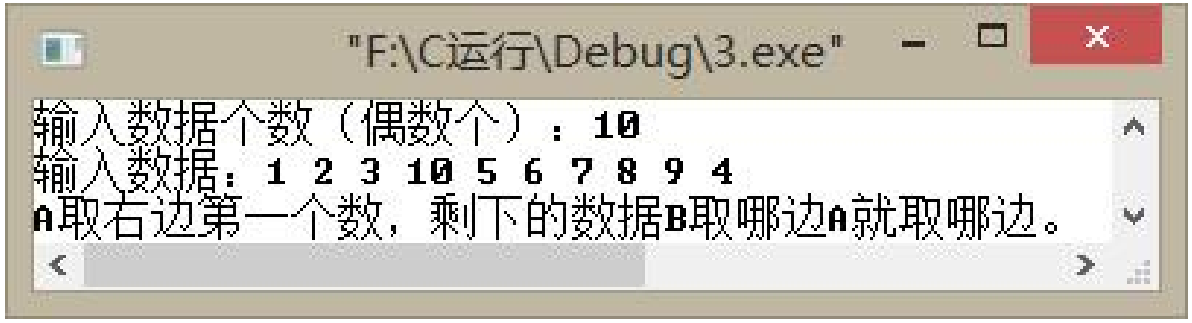
```
main()
{   int i,j,n,GZ,A;
    int B[8]={0,100,50,20,10,5,2,1},S[8]={0};
    printf("输入总人数：");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {     printf("输入第%d个人的工资：",i);
          scanf("%d",&GZ);
          for(j=1;j<=7;j++)
          {  A=GZ/B[j];
             S[j]=S[j]+A;     //S[1]——S[7]数组存储7种币值的张数
             GZ=GZ-A*B[j];     }
    }
    for(i=1;i<=7;i++)
        printf("%d元的%d张\n",B[i],S[i]);
}
```

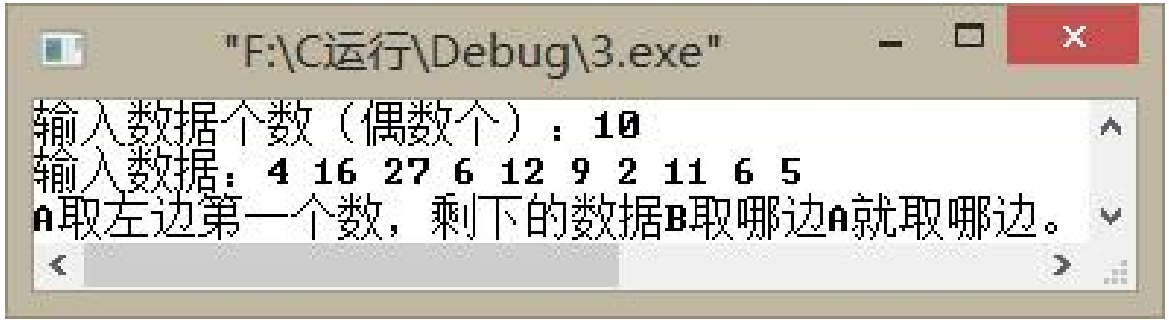
4.2 相对或近似贪婪问题

- 以上问题的背景是在我国，这样的币种正好适合使用贪婪算法(感兴趣可以证明这个结论)。若某国的币种共9种：100, 70, 50, 20, 10, 7, 5, 2, 1，再用贪婪算法就行不通了。比如某人工资是140，按贪婪算法140=100*(1张)+20*(2张)共需要3张, 而事实上, 只要取2张70面额的是最佳结果, 这类问题可以用动态规划算法来解决。
- 因此在用贪婪算法策略时，需要用数学方法证明每一步的策略是否能保证得到最优解。

4.2 相对或近似贪婪问题

- 例：取数游戏：有2个人轮流取2n个数中的n个数，取数之和大者为胜。假设取数者能看到全部2n个数，且每次只能取两边的数据。编写算法，让先取数者胜，模拟取数过程。





4.2 相对或近似贪婪问题

- 问题分析：
  - 这个游戏若假设取数者只能看到2n个数中两边的数。
  - 若一组数据为：6,16,27,6,12,9,2,11,6,5。用贪婪策略每次两人都取两边的数中较大的一个数,先取者胜。
  - 以A先取为例，取数结果为：

A　6+27+12+5+11=61　胜

B　16+6+9+6+2=39

  - 但若选另一组数据：16,27,7,12,9,2,11,6。仍用贪婪算法，先取者A败，取数结果为：

A　16+7+9+11=43

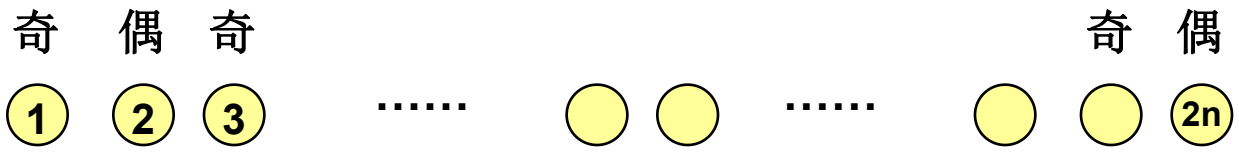
B　27+12+6+2=47　胜

4.2 相对或近似贪婪问题

- 问题分析：
  - 若我们只能看到两边的数据，则此题无论先取还是后取都无必胜的策略。这时一般的策略是近似贪婪算法。
  - 但若取数者能看到全部2n个数，则此问题可有一些简单的方法，有的虽不能保证所取数的和是最大，但确是一个先取者必胜的策略。

4.2 相对或近似贪婪问题

- 数学模型：
  - 2N个数排成一行，编号依次为1，2，…，2N，A先取数，B后取数，所以一开始A既可以取到一个奇编号的数（最左边编号为1的数），又可以取到一个偶编号的数（最右边编号为2N的数）。



  - A能够控制让B自始自终只取一种编号的数。只需比较奇编号数之和与偶编号数之和谁大，以决定最开始A是取奇编号数还是偶编号数即可。
  - 如果奇编号数之和与偶编号数之和同样大，A第一次可以任意取数，因为当两者所取数和相同时，先取者为胜。

4.2 相对或近似贪婪问题

- 算法设计：
  - 只需分别计算一组数的奇数位和偶数位的数据之和，则先取数者就可以确定必胜的取数方式了。
  - 以下面一排数为例：

位序	1	2	3	4	5	6	7	8	9	10
	3	2	1	10	9	8	5	6	7	4

  - 奇编号数之和为25（=3+1+9+5+7），小于偶编号数之和为30（=2+10+8+6+4）。A第一次取4，之后B取哪边的数A就取哪边的数（如果B取3，A就取2；如果B取7，A就取6）。这样可以保证A自始自终取到偶编号的数，而B自始自终取到奇编号的数。

4.2 相对或近似贪婪问题

- 算法实现：

```
main()
{   int i,s1=0,s2=0,data,n;
    printf("输入数据个数（偶数个）：");
    scanf("%d",&n);
    printf("输入数据：");
    for(i=1;i<=n;i++)
    {     scanf("%d",&data);
          if (i%2==0)
                    s2=s2+data;
          else
                    s1=s1+data;
    }
    if(s1>s2)
          printf("A取左边第一个数，剩下的数据B取哪边A就取哪边。\n");
    else
          printf("A取右边第一个数，剩下的数据B取哪边A就取哪边。\n");
}
```

4.3 关于贪婪策略讨论

- 贪婪算法适用的问题：
  - 贪婪算法对问题只需考虑当前局部信息就要做出决策，也就是说使用贪婪算法的前提是“局部最优策略能导致产生全局最优解”。
  - 该算法的适用范围较小，若应用不当，不能保证求得问题的最佳解。一般情况下通过一些实际的数据例子(当然要有一定的普遍性)，就能从直观上就能判断一个问题是否可以用贪婪算法。更准确的方法是通过数学方法证明问题对贪婪策略的通用性。