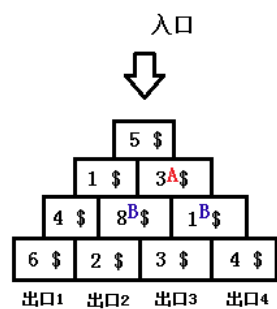


# 算法设计与分析 钻石金字塔问题

班级：2015211312  
学号：2015211484  
姓名：刘佳鑫

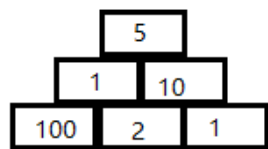
## ● 问题描述



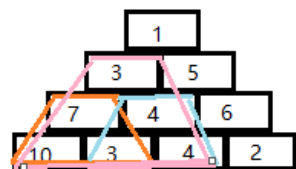
现在你的任务是从金字塔的顶端向金字塔的底端收集钻石，并且尽可能收集价值高的钻石，但是只能从一块砖斜向左下或斜向右下走到另一块砖上，如从上图从用红色 A 标记的砖走向用蓝色 B 标记的砖上。富翁希望 heimengnan 找到一个收集最高价值钻石的路线，并且把可能收集的最大价值告诉富翁。

## ● 算法描述及思考过程

拿到矿工问题，我第一个想法就是贪心算法，从上往下，每次都取大的。但是遇到以下状况时，贪心算法取不到最优解。最优解应该为  $5+1+100=106$ ，但是贪心算法求的是  $5+10+2=17$ 。



第二个想法是先遍历一遍，找到该金字塔中价值最大的位置，再根据它往上往下找，但是这样复杂度太高，而且也不能保证取得最优解。所以想到了用分治法。



如图，比如现在要填价值为“3”这个位置的和，则判断下它的左边和右边哪个价值和更大，则选择哪边，那么3这个位置的和则为3+更大的一方。即  $SUM[i]_{max} = A[i] + \max\{SUM[left], SUM[right]\}$ ；//SUM 为存储此位置能拿到的最大价值，A 为原金字塔数组。

**代码如下：**

```
int firstSolution(int A[],int top,int nf,int n,int sf,int sum[])
/*A[]-金字塔原数组， top-当前处于塔顶元素的序号， nf-当前 top 元素所处层数， n-金字塔元素总数， sf-金字塔总层数， sum-当前求得和*/
{
    int left,right;
    if(top>=n-sf+1&&top<=n)//top 元素位于最后一行
        return(sum[top]=A[top]);
    else
    {
        left=firstSolution(A,top+nf,nf+1,n,sf,sum);
        right=firstSolution(A,top+nf+1,nf+1,n,sf,sum);
        //和等于 top 处元素和左右之间大的那个相加
        if(left>right)
        {
            return(sum[top]=left+A[top]);
        }
        else
        {
            return(sum[top]=right+A[top]);
        }
    }
}
```

因为还要寻找路径， 所以根据 sum 数组的值， 查找路径。

**代码如下：**

```
void findway(int sum[],int A[],int sf)
{
    int *way;
    way=new int[sf+1];
    int k=1;
    for(int i=1;i<=sf;i++)//i 为当前层数
    {
        if(sum[k+i]>=sum[k+i+1])//左边大
        {
            way[i]=A[k+i];
            k+=i;
        }
        else
        {
            way[i]=A[k+i+1];
            k+=i+1;
        }
    }
}
```

```

    }

}

way[0]=A[1];//第一个必须经过
cout<<"路径"<<endl;
for(int j=0;j<sf;j++)
{
    cout<<way[j]<<" ";
}
cout<<endl;
}

```

运行结果如下：

```

请您输入金字塔数字层数
4
    1
   2 3
  4 5 6
 7 8 9 10

分治法
所用时间:0
路径
1 3 6 10
20

```

结果是正确的。

通过之前几次测试也可知，递归算法在相同时间复杂度时所耗费的时间远大于非递归算法，所以要想办法消除递归。

分治法的递归是从头向下走，递归深入金字塔，再把值返回来比较，所以要消除递归，我们可以从底下往上算。此时的复杂度为  $O(n)$ 。

代码如下：

```

int secondSolution(int A[],int n,int sf,int sum[]) //消除递归，从底往上走
//*A[]-金字塔原数组，n-金字塔总元素总数，sf-金字塔总层数，sum-当前求得和*/
{
    int i;
    int nf;//当前层数

    for(i=(sf*(sf-1)/2)+1;i<=n;i++)//最后一行
    {
        sum[i]=A[i];
    }
    nf=sf-1;
    for(i=sf*(sf-1)/2;i>=1;i--)

```

```

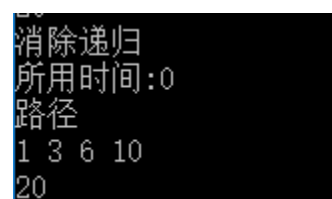
{
    if(sum[i+nf]>=sum[i+nf+1])
    {
        sum[i]=A[i]+sum[i+nf];
    }
    else
    {
        sum[i]=A[i]+sum[i+nf+1];
    }

    if(i==(nf*(nf-1)/2)+1)
    {
        nf--;//到了每行开头，则行数-1，因为下一个元素就是上一行的了
    }
}
return sum[1];

}

```

运行结果如下：



```

消除递归
所用时间:0
路径
1 3 6 10
20

```

运行结果正确。

此算法和张舒阳同学在课上讲的 betterpyramid 算法基本一致，所以借鉴她之后的优化，可以降低空间复杂度，即直接把算出来的 sum 存到 A 数组中，因为不会再回头，所以 A 中用过的元素不会再用第二遍，所以可以直接替换掉。大大降低了空间复杂度，也省去了填充最后一行的 sum 数组，时间复杂度变为  $O(n-sf)$

代码如下：

```

int thirdSolution(int A[],int n,int sf)//直接存 A 数组中，不用 sum 数组
{
    int i,nf;
    nf=sf-1;
    if(sf==1)//只有一层
    {
        return A[1];
    }

    else
    {

```

```

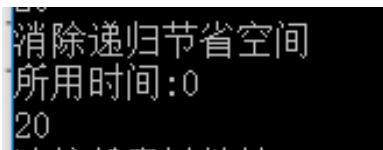
        for(i=n-sf;i>=1;i--)
        {
            if(A[i+nf]>=A[i+nf+1])
            {
                A[i]=A[i]+A[i+nf];
            }
            else
            {
                A[i]=A[i]+A[i+nf+1];
            }
            if(i==(nf*(nf-1)/2)+1)
            {
                nf--;//到了每行开头，则行数-1，因为下一个元素就是上一行的了
            }
        }

    }

    return A[1];
}

```

运行结果如下：



```

消除递归节省空间
所用时间:0
20

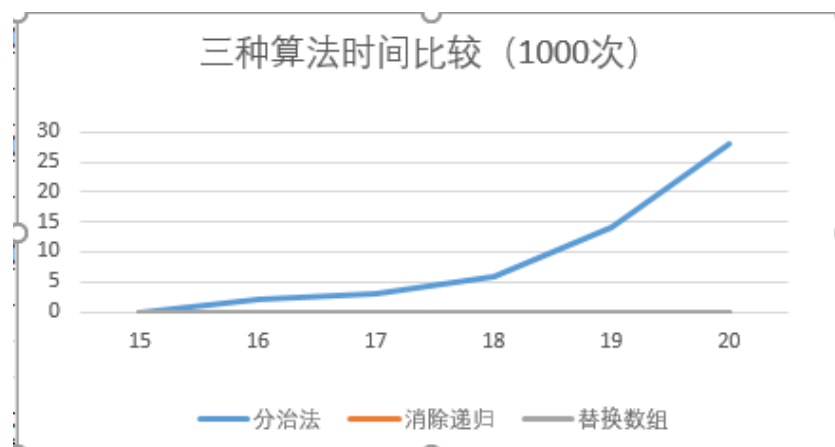
```

运行结果正确。

## ● 时间复杂度测试：

	层数	15	16	17	18	19	20
分治法	Iterations(K)	1000	1000	1000	1000	1000	1000
	Duration(sec)	0	2	3	6	14	28
消除递归	Iterations(K)	1000	1000	1000	1000	1000	1000
	Duration(sec)	0	0	0	0	0	0
替换数组	Iterations(K)	1000	1000	1000	1000	1000	1000
	Duration(sec)	0	0	0	0	0	0

根据数据得出以下折线图：



经几次测试发现，在层数逐渐增大时，分治法所用时间明显增加，斜率也十分大。可见其时间复杂度之高。

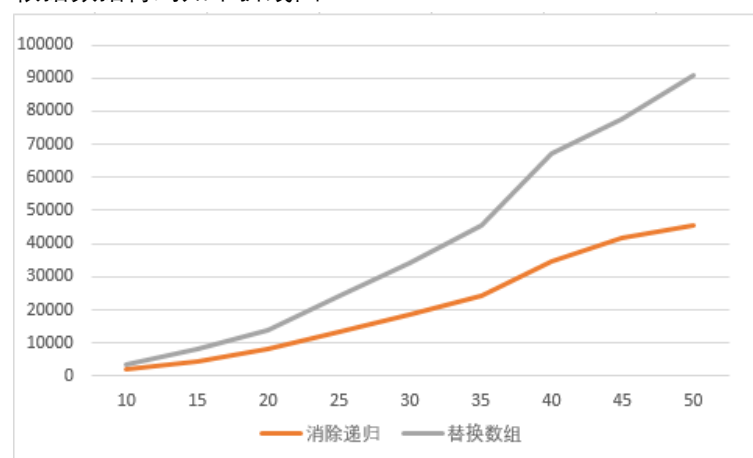
于是抛弃分治法，用了更高的重复次数和更多的层数测试后两个算法之间的差距。

测试了几组，发现因为差距不大，用 s 作为单位观察不出差距，所以把单位换成 ms。

得到数据如下：

层数		10	15	20	25	30	35	40	45	50
消除递归	Iterations(K)	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
	Duration(ms)	2111	4418	8082	13302	18740	24252	34511	41632	45288
替换数组	Iterations(K)	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
	Duration(ms)	1577	3569	5769	10742	15553	21459	32632	35939	45682

根据数据得到如下折线图：



发现两个算法增长的斜率几乎一致，但是仍在层数增大时花费时间差增大，因为差的是  $O(sf)$ ，和层数有关。

## ● 测试截图：

1. 包括分治法，重复次数为 1000 次

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000
请您输入金字塔数字层数
15

分治法
所用时间:0
680
消除递归
所用时间:0
680
消除递归节省空间
所用时间:0
680
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000
请您输入金字塔数字层数
16

分治法
所用时间:2
816
消除递归
所用时间:0
816
消除递归节省空间
所用时间:0
816
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000
请您输入金字塔数字层数
17

分治法
所用时间:3
969
消除递归
所用时间:0
969
消除递归节省空间
所用时间:0
969
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000
请您输入金字塔数字层数
18

分治法
所用时间:6
1140
消除递归
所用时间:0
1140
消除递归节省空间
所用时间:0
1140
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000
请您输入金字塔数字层数
19

分治法
所用时间:14
41330
消除递归
所用时间:0
51330
消除递归节省空间
所用时间:0
81330
请按任意键继续. . .
```



```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000
请您输入金字塔数字层数
20

分治法
所用时间:28
1540
消除递归
所用时间:0
1540
消除递归节省空间
所用时间:0
1540
请按任意键继续. . .
```

2.不算分治法，重复次数 1000000 次

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
10

消除递归
所用时间:2111
220
消除递归节省空间
所用时间:1577
220
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
15

消除递归
所用时间:4418
1680
消除递归节省空间
所用时间:3569
1680
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
20
消除递归
所用时间:8082
1540
消除递归节省空间
所用时间:5769
1540
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
25
消除递归
所用时间:13302
2925
消除递归节省空间
所用时间:10742
2925
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
30
消除递归
所用时间:18740
4960
消除递归节省空间
所用时间:15553
4960
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
35

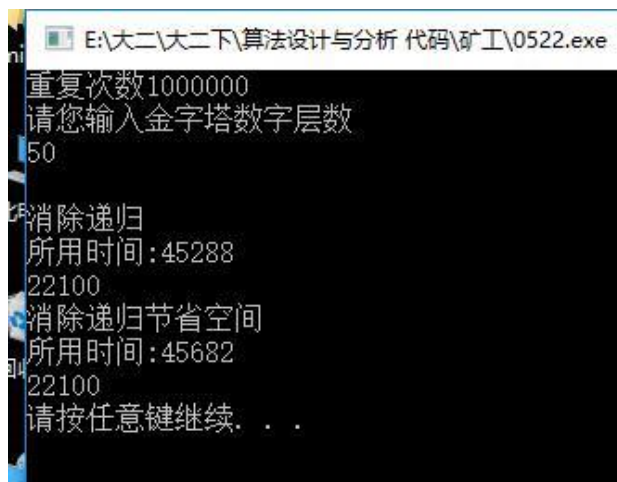
消除递归
所用时间:24252
7770
消除递归节省空间
所用时间:21459
7770
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
40

消除递归
所用时间:34511
11480
消除递归节省空间
所用时间:32632
11480
请按任意键继续. . .
```

```
E:\大二\大二下\算法设计与分析 代码\矿工\0
重复次数1000000
请您输入金字塔数字层数
45

消除递归
所用时间:41632
16215
消除递归节省空间
所用时间:35939
16215
请按任意键继续. . .
```



```
E:\大二\大二下\算法设计与分析 代码\矿工\0522.exe
重复次数1000000
请您输入金字塔数字层数
50
消除递归
所用时间:45288
22100
消除递归节省空间
所用时间:45682
22100
请按任意键继续. . .
```

## ● 实验总结：

这次探索的过程充满了惊喜。虽然在课堂上同学已经提供了好多种思路，但是真正自己开始思考问题时也是很有收获的。

借鉴张舒阳同学的思路，第一个分治法还可以进化一下，不用重复计算 sum 数组里的值，在计算之前判断一下里面有没有值，不过因为我直接从消除递归思路想的，所以跳过了这一步优化

通过测试也发现这几种算法时间复杂度还是有一定差距，所以优化算法非常有必要。不过测试仍是在相对的基础上，因为每次重复使用算法之前都要先初始化 A 数组或者 sum 数组，也要耗一定时间。而且最后一个算法虽然时间复杂度和空间复杂度有提高，但是不便于寻找路径。有利也有弊。