

中山大学数据科学与计算机学院本科生实验报告

(2016 学年春季学期)

课程名称: Algorithm design

任课教师: 张子臻

年级	15	专业(方向)	软件工程(移动信息工程)
学号	15352408	姓名	张稼伟
电话	13531810182	Email	709075442@qq.com
开始日期	3.13	完成日期	3.14

目录

1.实验题目	3
Soj 1035 DNA matching	3
Soj 1198 Substring.....	3
Soj 1093 Air Express	3
Soj 1438 Shopaholic.....	3
Soj 1681 Matchsticks	3
Soj 10359 Valuable Jewellery	3
Soj 1405 length error	4
Soj 1620 SCVs and minerals.....	4
Soj 1783 Large is Better	4
Soj 2503 最长字符串	4
Soj 8536 Happy Camper	4
Soj 6771 Class Packing	5
2.实验目的	5
3.程序设计	6

Soj 1035 DNA matching	6
Soj 1198 Substring.....	7
Soj 1093 Air Express	8
Soj 1438 Sopaholic.....	9
Soj 1681 Matchsticks	10
Soj 10359 Valuable Jewellery	12
Soj 1405 length error	13
Soj 1620 SCVs and minerals.....	14
Soj 1783 Large is Better	15
Soj 2503 最长字符串	16
Soj 8536 Happy Camper	17
Soj 6771 Class packing	18
5.实验总结与心得	19
附录、提交文件清单	20

1.实验题目

贪心专题

Soj 1035 DNA matching

题意：给 n 个 DNA 单链，问最多能组成多少对 DNA 双链，每一个 DNA 单链只能使用一次，并且不能够翻转。两个 DNA 单链能够形成一对 DNA 双链的条件是对应位置 A/T，C/G 配对

约束： $1 \leq n \leq 100$ ，DNA 单链的长度 $m \leq 100$ 多组数据

Soj 1198 Substring

题意：给 n 个字符串，现在需要将他们拼接起来形成一个字典序最小的字符串

约束： $1 \leq n \leq 8$

Soj 1093 Air Express

题意：给出 4 个重量区间和各个区间的单位重量运输价，问对于一个背包，需要添加多少重量使得运输代价最小。

约束：所有的数都为正数，且不超过 1000

Soj 1438 Shopaholic

题意：有个购物狂在商场中购物，他想要买 n 个商品，商场在做促销，每买三个商品，最便宜的一个可以免费，现在问最多可以省多少钱。

约束： $n \leq 2 \times 10^4$ $price \leq 2 \times 10^4$

Soj 1681 Matchsticks

题意：拥有 n 根火柴，求出可以摆出的最大和最小的数。摆出 0-9

所需

火柴数分别为 6, 2, 5, 5, 4, 5, 6, 3, 7, 6。

Soj 10359 Valuable Jewellery

题意：有 N 个物品和 K 个背包，物品有质量 M_i 和价值 V_i 两个属性，

背

包有最大承受质量的属性，一个背包只能装一个物品，询问最大能够带走的价值。

Soj 1405 length error

题意：给出参数 n 和 k ， n 为人数， k 为所分组的每组的人数(保证 n 能被 k 整除，这样便得到 n/k 个组)。然后需要满足以下条件：组内所有人的姓名长度，和这个组的平均姓名长度的差距不大于 2(平均长度在这里可以是有小数的)，问是否存在这样的分组方法。

Soj 1620 SCVs and minerals

题意：一开始你有 N 个农民和 M 单位的资源，每个农民一秒钟可以得到 C 单位的资源，你可以使用 P 单位资源去立即完成生产一个农民，每次不限生产的农民数，只要你有足够的资源。周围资源没有限制，没有人口限制，而且农民采集资源的过程是离散的(例如不会在 0.5 秒的时候收入 0.5C)，问在 S 秒后所收集的最大资源是多少(不包括已经花出去的)。

Soj 1783 Large is Better

题意：给出一个数，你可以对这个数做出以下调整：交换任意两个相邻的数，没有次数限制，交换的两个数中不能有 0，求可以得到的最大的数。

Soj 2503 最长字符串

题意：要求你构造一个由字符 'A' , 'B' 组成的字符串，满足以下几个条件：

- 1) A 的个数 $\leq \text{countA}$
- 2) B 的个数 $\leq \text{countB}$
- 3) 连续的 A 的个数不可以超过 maxA .
- 4) 连续的 B 的个数不可以超过 maxB .
- 5) 这个字符串的长度最长.

给你 countA , countB , maxA , maxB , 要求你输出字符串的最大长度.

Soj 8536 Happy Camper

题意：在连续的 P 天里可以有 L 天享受假期，这 P 天不能和另一个阶段的 P 天重用，问在 V 天里最多享受的假期数

Soj 6771 Class Packing

题意: 有年级为 0, 1, 2, 3, 4, 5, 6 共七个年级的孩子, 每个年级的孩子数量给出。现为这些孩子分配班级, 满足所分配班级的个数最小。其中, 分配班级的时候需要满足以下条件:

- 1: 一个班级只能有一个年级, 或者两个相邻年级的孩子
- 2: 拥有年级 0-2 的孩子的班级, 其人数最多为 20
- 3: 拥有年级 3-4 的孩子的班级, 其人数最多为 25
- 4: 拥有年级 5-6 的孩子的班级, 其人数最多为 30

2.实验目的

1. 加深对贪心算法设计的基本思想, 基本步骤, 基本方法的理解与掌握。
2. 提高运用课堂所学知识解决实际问题的能力。

3.程序设计

Soj 1035 DNA matching

分析:题目中说两个 DNA 单链能够形成一对 DNA 双链的条件是对应位置 A/T, C/G 配对, 这里也就告诉我们, 一个特定的 DNA 单链 D1, 只能与一个特定的 DNA 单链 D2 匹配, 不会出现 D1 可以同时和 D2、D3 匹配的情况(这里 D2、D3 是两种不同的 DNA 单链)。因此我们只要贪心地给每一个 DNA 单链寻找有没有还未配对的又能和它配对的另一个 DNA 单链即可。

解法:用一个 map 存放未匹配的 DNA 单链。从头开始循环, 对于每一个 DNA 单链, 询问 map 中是否还有与它配对的 DNA 单链, 若有, 则答案+1 并删去 map 中对应的 DNA 单链。否则, 将当前 DNA 单链存入 map 中。
时间复杂度 $O(nm\log n)$ 。

代码:

```
1 #include<iostream>
2 #include<string>
3 #include<map>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 int main() {
7     int T,ans,n;
8     string str,match_str;
9     map<char,string>match;
10    match['A']="T";
11    match['T']="A";
12    match['C']="G";
13    match['G']="C";
14
15    cin>>T;
16    rep(t,1,T) {
17        map<string,int>f;
18        ans=0;
19        cin>>n;
20        rep(i,1,n) {
21            cin>>str;
22            match_str="";
23            int len=str.length();
24            rep(j,0,len-1)
25                match_str+=match[str[j]];
26            int &ret=f[match_str];
27            if (!ret)f[str]++;
28            else {
29                ret--;
30                ans++;
31            }
32        }
33        cout<<ans<<endl;
34    }
35    return 0;
```

Soj 1198 Substring

分析:一个很容易的想法是将所有字符串按字典序排序之后顺序连接起来。然而,如果有字符串 b 和 ba,按照上述解法得到的解是 bba,但实际上 bab 更优。这里就启发我们,那我们排序的 A, B 两个字符串的时候,将简单的按照字典序比大小的方法改为连接起来比大小行不行呢?(即若 $AB < BA$ 则 $A < B$)。接下来我们证明这个比较方法具有传递性:

设 $A < B$ 当且仅当 $AB \leq BA$, 证明 $A < B, B < C$ 则 $A < C$ 。

由 $A < B, B < C$ 得 $AB \leq BA, BC \leq CB$

得 $BA+BC+AB \leq BA+CB+AB$

得 $A+B+A+C+A \leq A+C+A+B+A$

得 $AC \leq CA$

得 $A \leq C$

所以这个比较方法是可行的。

解法:读入所有字符串之后,按照 $S1+S2 < S2+S1$ 则 $S1 < S2$ 的比较方法将所有字符串从小到大排序,然后再顺次连接输出即可。
时间复杂度 $O(n \log n)$

代码:

```
1 #include<iostream>
2 #include<string>
3 #include<algorithm>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 bool cmp(const string &s1,const string &s2){
7     return s1+s2<s2+s1;
8 }
9 int main(){
10     int T,n;
11     string str[9];
12     cin>>T;
13     rep(t,1,T){
14         cin>>n;
15         rep(i,1,n)
16             cin>>str[i];
17         sort(str+1,str+1+n,cmp);
18         rep(i,2,n)str[1]+=str[i];
19         cout<<str[1]<<endl;
20     }
21     return 0;
22 }
```

Soj 1093 Air Express

分析和解法:显然对于一个重量区间，这个区间的下界重量是运输总价格在这个区间里是最少的，所以对于每个背包，我们只要求出它在不增加重量的运输价格，以及加重量到之后的每个重量区间下界的运输价格，取一个最小值即可。

时间复杂度 $O(n)$

代码:

```
1 #include<iostream>
2 #include<cstdio>
3 #include<algorithm>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 struct node{
7     int low,up,rate;
8 }a[5];
9 int main(){
10     int x,T=0;
11     a[1].low=0;a[4].up=1000;
12     while (cin>>a[1].up>>a[1].rate){
13         rep(i,2,4){
14             if (i<4)cin>>a[i].up;
15             a[i].low=a[i-1].up+1;
16             cin>>a[i].rate;
17         }
18         printf("Set number %d:\n",++T);
19         while (cin>>x&&~x){
20             int ans,ansx=x;
21             rep(i,1,4){
22                 if (a[i].low<=x&&x<=a[i].up)
23                     ans=x*a[i].rate;
24                 else if (a[i].low>x){
25                     if (ans>a[i].low*a[i].rate){
26                         ans=a[i].low*a[i].rate;
27                         ansx=a[i].low;
28                     }
29                 }
30             }
31             printf("Weight (%d) has best price $%d (add %d pounds)\n",x,ans,ansx-x);
32         }
33         puts("");
34     }
35     return 0;
}
```


Soj 1438 Sopaholic

分析:要想节省更多的钱,则要想办法让尽可能多价格贵的商品可以免费。如果贵的商品和两个便宜的商品分在一起,显然贵的商品不会免费。所以我们只能将相邻价格的商品三个三个地捆绑在一起,才能节省最多的钱。

解法:我们只要把所有商品按价格大小从大到小排序,然后从头开始三个三个分一组,每一组的第三个即是免费商品。将每一组的第三个商品的价格加起来就是所能节省的最大钱数。

时间复杂度 $O(n\log n)$

代码:

```
1 #include<cstdio>
2 #include<algorithm>
3 using namespace std;
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
5 bool cmp(int a, int b){
6     return a>b;
7 }
8 int main(){
9     int t, n, a[20001], i, j, ans;
10    scanf("%d", &t);
11    while(t--){
12        ans=0;
13        j=3;
14        scanf("%d",&n);
15        rep(i,1,n)
16            scanf("%d", &a[i]);
17        sort(a+1,a+1+n,cmp);
18        while(j<=n){
19            ans+=a[j];
20            j+=3;
21        }
22        printf("%d\n", ans);
23    }
24    return 0;
25 }
```

Soj 1681 Matchsticks

分析和解法:如果数字要最大,则我们要摆的数字位数应该尽可能多,而这里数字 1 需要的火柴数最少,只有 2 根。所以我们应该摆尽可能多的 1,最后如果剩一根火柴,则将第一个 1 改为 7 即可。

相反,如果数字要最小,我们应优先考虑摆最少位数的数字,那么每位数字应使用最多的火柴,这里数字 8 用的火柴最多,有 7 根。则我们先摆多一些 8,到最后假设剩下 x 根火柴,我们再从左往右枚举,看能否找到最小的两个数字,使得他们的火柴数之和= $x+8$ 的火柴数,这样就可以将这两个数字替换掉开头的 8,以及用完剩余的火柴。这样做完仍不能保证数字最小,我们需要对之后的每两位都检验一下能否做如上的替换即可。

代码:

```
1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 string finmin(int x){
5     string s="";
6     int num[10] = {6, 2, 5, 5, 4, 5, 6, 3, 7, 6};
7     if (x<=7){
8         rep(i,1,9)
9             if (x==num[i]){
10                 s+=char(i+48);
11                 break;
12             }
13     }
14     else {
15         int eight=x/7;
16         x%=7;
17         if (x){
18             int tem;
19             if (x==1) tem=9;
20             else{
21                 rep(i,1,9)
22                     if (x==num[i]){
23                         tem=i;
24                         break;
25                     }
26             }
27             bool start=true;
28             while (eight){
29                 bool flag = true;
30                 int i;
31                 if (start) i = 1;
32                 else i = 0;
33                 start=false;
34                 for (i;i<=tem;i++){
```

```

35         rep(j,0,7)
36         if (num[i]+num[j]==x+num[8])
37             s+=char(i+48);
38             tem=j;
39             flag=false;
40             eight--;
41             break;
42     }
43     if (!flag)break;
44 }
45 if (flag){
46     s+=char(tem+48);
47     break;
48 }
49 x=num[tem];
50 if (!eight){
51     s+=char(tem+48);
52     break;
53 }
54 }
55 }
56 rep(i,1,eight)
57     s+="8";
58 }
59 return s;
60 }
61 string finmax(int x){
62     string s="";
63     if (x%2==1){
64         x-=3;
65         s="7";
66     }
67     rep(i,1,x/2) s+="1";
68     return s;
69 }
70 int main(){
71     int T,n;
72     cin>>T;
73     rep(t,1,T){
74         cin>>n;
75         string minn=finmin(n);
76         string maxx=finmax(n);
77         cout<<minn<<" "<<maxx<<endl;
78     }
79     return 0;
80 }

```

Soj 10359 Valuable Jewellery

分析: 由于一个背包只能装一个物品，要求最大能带走的质量，我们最直观的想法就是尽量装价值高的物品，而为了尽量利用好每个背包，我们对每一个价值高的物品，应寻找一个刚好能装下它的背包去装。如果不这样做，很有可能导致一些本来能带走的物品最后不能带走。例如有两个背包，A 能容纳 10kg，B 能容纳 3kg，两个物品，一个 3kg，一个 6kg。假如我们用 A 去装 3kg 的物品，那么我们就只能带走一个物品，而实际上能带走两个物品。

解法: 将物品按照价值从大到小排序。将背包存入一个可重复集合 multiset 中。按价值从大到小遍历物品，用 multiset 中的 lower_bound 函数寻找刚好能装下当前物品的背包。若能找到，则答案加上这个物品的价值并删去 multiset 中的相应背包；若找不到，则放弃带走该物品。
时间复杂度 $O(n\log n + n\log k)$

代码:

```
1 #include<iostream>
2 #include<cstdio>
3 #include<algorithm>
4 #include<set>
5 using namespace std;
6 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
7 #define N 300010
8 struct item{
9     int m,v;
10 }a[N];
11 bool cmp(const item &A,const item &B){
12     return A.v>B.v;
13 }
14 multiset<int>bag;
15 int n,k,x;
16 long long ans=0;
17 int main(){
18     scanf("%d%d",&n,&k);
19     rep(i,1,n)
20         scanf("%d%d",&a[i].m,&a[i].v);
21     sort(a+1,a+1+n,cmp);
22     rep(i,1,k){
23         scanf("%d",&x);
24         bag.insert(x);
25     }
26     rep(i,1,n){
27         multiset<int>::iterator it;
28         it=bag.lower_bound(a[i].m);
29         if (it==bag.end()) continue;
30         ans+=a[i].v;
31         bag.erase(it);
32     }
33     cout<<ans<<endl;
34     return 0;
35 }
```

Soj 1405 length error

分析: 我们知道, 要使一组数的平均数与这组数的每一个数字的差值尽量小, 则这组数字应尽量接近。同理, 将 n 个人分 k 组, 组内所有人的姓名长度, 和这个组的平均姓名长度的差距要尽量小, 则这组内每一个人的姓名的长度应该尽量接近。

解法: 根据上述分析, 我们将 n 个人按照其姓名长度排序, 然后从左到右每 k 个人分为一组即可。再判断每一组内是否满足组内所有人的姓名长度, 和这个组的平均姓名长度的差距不大于 2。由于平均姓名长度可能为小数, 所以我们可以用乘法代替直接比较。设某个人的姓名长度为 x , 总姓名长度为 k 。则 $|x - \text{sum}/k| \leq 2$ 可化为 $|x*k - \text{sum}| \leq 2*k$ 。关于 sum , 我们在遍历每一组人的时候可以顺便求得, 而我们不需要对组内每个人都进行上述比较, 只需用组内最长姓名和最短姓名去比较即可。
时间复杂度 $O(n \log n)$ 。

代码:

```
6 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
7 bool cmp(const string &s1,const string &s2){
8     return s1.length()<s2.length();
9 }
10 int n,k;
11 int main(){
12     int T=0;
13     string s[1001];
14     while (cin>>n>>k,n>0){
15         if (++T>1)cout<<endl;
16         cout<<"Case "<<T<<" ";
17         rep(i,1,n)cin>>s[i];
18         sort(s+1,s+1+n,cmp);
19         int sum=0,minn=100,maxx=0,cnt=0,flag=1;
20         rep(i,1,n){
21             cnt++;
22             sum+=(int)s[i].size();
23             minn=min(minn,(int)s[i].size());
24             maxx=max(maxx,(int)s[i].size());
25             if (cnt==k){
26                 if (!(sum-minn*k<=2*k && maxx*k-sum<=2*k)){
27                     flag=0;
28                     break;
29                 }
30                 cnt=sum=0;
31                 maxx=0;minn=100;
32             }
33         }
34         if (flag)cout<<"yes"<<endl;
35         else cout<<"no"<<endl;
36     }
37     return 0;
38 }
```


Soj 1620 SCVs and minerals

分析:显然,在每一秒,都有两种选择,一种是买农民,另一种是什么都不做等收益。而决定我们是否买农民取决于买了这个农民之后到最后能否回本。若能回本,则买尽量多的农民。那能否暂时不买或者买一些,过一些时间再买一些呢?我们设原有 x 个农民,当前在第 k 秒,当前资源为 m ,最多可以买 y 个农民,假设我们以后不再买新的农民,那么在最后的收益为 $m-y*p+(x+s-k+1)*c$ 。假如我们现在不买 y 个农民,改为买 z 个 ($z < y$),在 $k+L$ 秒买 $y-z$ 个,则最后收益为 $m-y*p+(x+s-k+1)*c-(y-z)*c*L$,显然不如一开始买 y 个收益高。这里又有一个疑问,在 $k+L$ 秒买比 $y-z$ 更多的农民会不会更优,显然,在 $k+L$ 秒买更多的农民,是因为第 k 秒买的农民回本之后带了充足的资源,所以,第 k 秒买的农民越多,第 $k+L$ 秒能买的农民就更多,收益也越大。这证明了我们的贪心策略是对的。

解法:在当前第 i 秒判断当前买农民最后是否能回本且带来更多的收益,即满足 $c*(s-i+1) \geq p$ 。若能,则买最多能买的农民,然后更新当前秒的资源总数。

时间复杂度 $O(S)$ 。

代码:

```
1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 int main(){
5     int T,N,M,C,P,S;
6     cin>>T;
7     while (T--){
8         cin>>N>>M>>C>>P>>S;
9         rep(i,1,S){
10             if (M>=P&&C*(S-i+1)>=P){
11                 N+=M/P;
12                 M%=P;
13             }
14             M+=N*C;
15         }
16         cout << M << endl;
17     }
18     return 0;
19 }
```

Soj 1783 Large is Better

分析:由于交换的数中不能有 0，那么我们将 0 当做分隔符，将原数列分成若干段。每一段的数字从大到小排序后接在一起输出即可。

解法:读入字符串 s，从头开始遍历，用另一个字符串 str 存储当前段的数字，遇到 0 时，将 str 中的数字从大到小排序后输出。然后将 str 清空，重复上述步骤。

时间复杂度 $O(n\log n)$ 。

代码:

```
1 #include<iostream>
2 #include<cstring>
3 #include<algorithm>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 int main(){
7     int T;
8     string s;
9     cin>>T;
10    rep(t,1,T){
11        cin>>s;
12        int len=s.length();
13        string str="";
14        rep(i,0,len-1){
15            str+=s[i];
16            if (s[i]=='0'){
17                sort(str.begin(),str.end(),greater<char>());
18                cout<<str;
19                str="";
20            }
21        }
22        if (str!=""){
23            sort(str.begin(),str.end(),greater<char>());
24            cout<<str;
25        }
26        cout<<endl;
27    }
28    return 0;
29 }
```

Soj 2503 最长字符串

分析:我们可以把其中一个字符串看成基串，另一个字符串拆成若干个分隔符去分割基串，由于要得到最长字符串，所以基串分割的每一段都尽量达到最大长度，而作为分割符的那个串，每一个分隔符都应尽量长。

解法:首先求出 A 和 B 串最少可以分成几段(每一段尽量达到 $\max A$ 或 $\max B$ 的长度)，记为 blocka 和 blockb 。显然，接下来有三种情况：

1. $\text{countA} < \text{blockb}$ ，即能将 A 串拆成 countA 个分隔符插入到 B 串中，得到 $\text{countA}+1$ 个 B 的连续子串，每个子串长度为 $\max B$ 。此时答案为 $\text{countA} + (\text{countA}+1) * \max B$ 。
2. $\text{countB} < \text{blocka}$ ，即能将 B 串拆成 countB 个分隔符插入到 A 串中，得到 $\text{countB}+1$ 个 A 的连续子串，每个子串长度为 $\max A$ 。此时答案为 $\text{countB} + (\text{countB}+1) * \max A$ 。
3. 若不满足以上两种情况，显然此时一个串可以将另一个串分隔且每一段不超过最大值而且没有剩下的字符。因此此时答案为 $\text{countA} + \text{countB}$ 。

此题还有注意单独处理 $\max A=0$ 或 $\max B=0$ 或 $\max A$ 和 $\max B$ 同时为 0 的情况。时间复杂度 $O(1)$ 。

代码：

```
1 #include<iostream>
2 #include<algorithm>
3 #include<cmath>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 int main(){
7     int countA,countB,maxA,maxB,ans;
8     cin>>countA>>countB>>maxA>>maxB;
9     if (maxA==0 && maxB==0) cout<<0<<endl;
10    else if (maxA==0) cout<<min(countB,maxB)<<endl;
11    else if (maxB==0) cout<<min(countA,maxA)<<endl;
12    else {
13        int blocka=ceil((double)countA/maxA);
14        int blockb=ceil((double)countB/maxB);
15        ans=0;
16        if (countA<blockb) ans+=countA+(countA+1)*maxB;
17        else if (countB<blocka) ans+=countB+(countB+1)*maxA;
18        else ans=countA+countB;
19        cout<<ans<<endl;
20    }
21    return 0;
22 }
```


Soj 8536 Happy Camper

分析:此题是一个简单的数学题。在连续的 P 天里可以有 L 天享受假期，这 P 天不能和另一个阶段的 P 天重用，问在 V 天里最多享受的假期数。显然，在每个 p 天里都享受 L 天假期，这样子享受的假期数是最多的。所以将 V 天分成若干个 p 天，每个 p 天都享受 L 天假期，这样前面一共享受了 $(V/P)*L$ 天假期。剩下的 $V\%P$ 天中再享受尽量多的假期，即 $\min(L, V\%P)$ 天，两者相加即为答案。

时间复杂度为 $O(1)$

解法: 读入数据后直接计算即可。答案为 $(V/P)*L+\min(L, V\%P)$ 。

代码:

```
1 #include<iostream>
2 using namespace std;
3 int main() {
4     int T=0,L,P,V;
5     while (cin>>L>>P>>V,V>0)
6         cout<<"Case "<<T++<<": "<<(V/P)*L+min(L,V%P)<<endl;
7     return 0;
8 }
```

Soj 6771 Class packing

分析:这道题首先明确一点，每个孩子都要分配到一个班级。由于不同年级的班级人数限制不一样，显然将同一个年级的孩子尽量分到一个班是最好的选择。考虑年级 0 的孩子，如果人数多，将他们尽量分配到同一个班的结果是理想的，不然会影响高年级的班级容量。如果人数有剩下，则从高一年级中分配人过来填满一个班，这样既不浪费班级容量，又可以使高年级再后面的分配中少分班。对于其他年级，同样如上处理。











解法:一重循环从低年级到高年级遍历， $a[i]$ 记录 i 年级孩子人数， $num[i]$ 记录 i 年级的班级容量。对每个年级，首先分配 $a[i]/num[i]$ 个班，剩余 $a[i]\%num[i]$ 个孩子再分一个班，如果 $i+1$ 年级有孩子，则调配 $\min(a[i+1], num[i]-a[i]\%num[i])$ 个孩子过来填满这个班。时间复杂度 $O(n)$ ， n 为年级数量。





代码:

```
1 #include<iostream>
2 #include<algorithm>
3 using namespace std;
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
5 int main(){
6     int a[7];
7     int num[7]={20,20,20,25,25,30,30};
8     while (cin>>a[0]){
9         int flag=a[0],ans=0;
10        rep(i,1,6) cin>>a[i],flag|=a[i];
11        if (!flag)break;
12        rep(i,0,6){
13            if (!a[i])continue;
14            ans+=a[i]/num[i];
15            a[i]%=num[i];
16            if (!a[i])continue;
17            ans++;
18            if (i<6&&a[i+1]>0)
19                a[i+1]-=min(a[i+1],num[i]-a[i]);
20        }
21        cout<<ans<<endl;
22    }
23    return 0;
24 }
```

4.程序运行与测试

12 道题均通过成功运行并通过 sicily 的测试。

Run ID		User Name	Problem	Language	Status	Run Time	Run Memory	Code Length	Submit Time
5054768		smie15352408	6771	C++	Accepted	0.01sec	308 KB	586 Bytes	2017-03-14 12:10:45
5054758		smie15352408	8536	C++	Accepted	0sec	308 KB	175 Bytes	2017-03-14 11:56:33
5054753		smie15352408	2503	C++	Accepted	0sec	308 KB	665 Bytes	2017-03-14 11:43:55
5054752		smie15352408	2503	C++	Accepted	0sec	308 KB	669 Bytes	2017-03-14 11:43:27
5054750		smie15352408	2503	C++	Accepted	0sec	308 KB	815 Bytes	2017-03-14 11:40:10
5054741		smie15352408	1783	C++	Accepted	0.02sec	308 KB	620 Bytes	2017-03-14 11:21:04
5054730		smie15352408	1620	C++	Accepted	0sec	308 KB	371 Bytes	2017-03-14 11:10:09
5054470		smie15352408	1405	C++	Accepted	0sec	308 KB	968 Bytes	2017-03-13 23:54:00
5054443		smie15352408	10359	C++	Accepted	0.84sec	9648 KB	690 Bytes	2017-03-13 23:35:41
5054429		smie15352408	1681	C++	Accepted	0sec	308 KB	1896 Bytes	2017-03-13 23:23:44

Run ID		User Name	Problem	Language	Status	Run Time	Run Memory	Code Length	Submit Time
5054245		smie15352408	1438	C++	Accepted	0.05sec	288 KB	485 Bytes	2017-03-13 20:30:27
5054240		smie15352408	1093	C++	Accepted	0sec	308 KB	947 Bytes	2017-03-13 20:25:45
5053955		smie15352408	1198	C++	Accepted	0sec	308 KB	444 Bytes	2017-03-13 14:57:21
5053946		smie15352408	1035	C++	Accepted	0sec	308 KB	724 Bytes	2017-03-13 14:49:54

5.实验总结与心得

通过这次贪心专题的训练，我加深了对贪心算法的理解。贪心算法的中心思想是局部最优解能导致产生全局最优解。设计贪心算法一般来说分为以下几步：1.建立数学模型来描述问题。

2.把求解的问题分成若干子问题。

3.对每个子问题求解，得到每个子问题的最优解。

4.将子问题的最优解组合起来就得到了全局的最优解。

要注意，要应用贪心算法一定要分析解决问题的过程是否满足无后效性，若有后效性则不能用贪心算法。

看到上面对贪心算法设计步骤的分解，我们发现其与动态规划非常相像。但实际上，贪心与动态规划的区别在于，贪心的子问题之间是相互独立的，子问题的最优解组合起来得到全局的最优解；动态规划的子问题之间不一定是相互独立的，有可能子问题 B 的最优解需要用到子问题 A 的最

优解才能求出，全局的最优解则通过子问题之间的关系层层递推而得到，而不是单纯地组合起来。

附录、提交文件清单

实验报告.pdf

1035.cpp

1093.cpp

1198.cpp

1405.cpp

1438.cpp

1620.cpp

1681.cpp

1783.cpp

2503.cpp

6771.cpp

8536.cpp

10359.cpp