收徒

2016年5月7日　　22:33

题目来源：https://biancheng.love/contest-ng/index.html#/34/problems

题目描述

Nova君想要找个徒弟和他一起玩游戏机，然而，Nova君是个要求很多的人，游戏能力在他之上的或者和他一样强的都不要，要了岂不是很没面子ヽ(ゞ▽ゝ)ノ能力太差的也不要_Orz__ 所以Nova君决定在游戏能力在他之下的人里挑选能力最强的。世界那么大，很可能有一些人能力值一样。Nova君在想，到底有多少人有资格呢？请来帮帮他找机友。

输入

多组测试数据（组数不超过10），对于每组数据，输入两行，第一行为两个正整数N和M，分别表示所有候选人的个数以及Nova君的能力值；第二行包含N个正整数，表示N个候选人的能力值（已经按非降序排列好）。N<=1000000

输出

对于每组数据，输出一行，输出有资格的人的个数。

输入样例

```
7 9
2 4 6 8 8 8 9
```

输出样例

```
3
```

HINT

请用二分实现

解题思路：

根据题目要求，我们需要做的就是找到能力不超过自己的能力最大的人。也就是第二大数有多少个。

要求使用二分实现。

二分实现的一些函数lower_bound upper_bound.

推荐博客：http://www.cnblogs.com/cobbliu/archive/2012/05/21/2512249.html

现在给出代码：

```
#include <bits/stdc++.h>
#define MAX 10000010
using namespace std;
int a[MAX];
int main()
{
    int n,m,k;
    int *f,*p,*l;
    while(~scanf("%d%d", &n,&m))
    {
        for(int i=0;i<n;i++)
            scanf("%d",&a[i]);
        p=lower_bound(a,a+n,m);
        k=*(p-1);
```

```
        f=lower_bound(a,a+n,k);
        l=upper_bound(a,a+n,k);
        printf("%d\n", l-f);
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5037546>

零崎的悠哉日常 I

2016年5月7日　　22:34

题目来源：https://biancheng.love/contest-ng/index.html#/34/problems

题目描述

最近，零崎发现自己的基友们都很闲，于是提议大家来玩一个叫“跑得快”的游戏吧，输的人就去操场跑三千米。经过一番苦战，除了跑的最快的jhljx，其他人都要去跑三千米了……

关于长跑，零崎有一个无耻的习惯，那就是紧跟着第一的人跑（如果前面没人就干脆不跑）。因为学院路这边的操场上每天都有很多人跑步锻炼身体，零崎肯定是逃不掉这个三千米的，跑完不过是时间问题。

为了简化问题，假设这些人跑步都是匀速的，零崎会跟在第一个人后面跑，如果被更快的人超过，零崎会加速跟在那个人后面。那么零崎到底多久能跑完？

输入

多组输入数据。

每组输入数据第一行为一个整数N，为操场上跑步的总人数。

接下来N行每行两个数Vi,Ti为第i个人的速度(km/h)和出发时间(s)（负数为在零崎之前出发，不考虑超过零崎一圈的情况）

每组输入数据中至少有一组出发时间非正的数据和一组出发时间非负的数据。

输出

对于每组数据，输出一行，为零崎跑完三千米的时间，以秒为单位向上取整。

输入样例

```
4
20 0
25 -155
27 190
32 200
```

输出样例

```
538
```

Hint

double ceil(double x) 头文件 math.h / cmath

解题思路：

根据题目要求，不考虑超过一圈的情况，这样可以把跑道看做一条直线。每个人都是匀速直线地跑步，而你却是跟着超过你的人跑步。当没有人在你前面的时候你会觉得没有动力所以也就站在起点一动不动，直到有人超过你。这时你就跟着这个人一起跑。突然又有一个人超过了你们两个，所以你决定追上他，跟着新的人跑。按照这个规律下去，直到到达终点。我们现在要做的就是求出到达终点也就是跑3000米需要的时间。

仔细想想，提前出发的人也就是出发时间为负值的人，我们没有必要把他们考虑进去。第一种情况：如果你能超过他，那么说明你的速度比他快也就是你能更快的到达终点，不需要跟着他，也就相当于不考虑他了。第二种情况：你没有追上先出发的那些人，说明你没有资格跟在他们后面，为什么没有资格呢，是因为后出发的人当中没有人能够超过他们，也就是你跑完3000米的时间跟他们没有关系。因此可以将出发时间为负值的那些人当做路人甲，不用管他们。

现在我们需要求出到达终点的时间。因为每次你都能跟着超过你的人，所以直到终点你一直都是

最快的，（这里提示一下，你不是全程都跟着最快的人跑，你这是在路途中不断地更换最快的人），那么我们求的时间就是谁最先到达终点，你也就跟着他到达了终点。也就是意味着不用考虑跑步过程中究竟是先跟着谁然后跟着谁，决定你跑步时间的是最先到达终点的（提示一下，这里最先到达终点的所属集合是出发时间为非负数，也就是不提前出发的人）。

通过分析很容易得到代码：

```
#include <bits/stdc++.h>
using namespace std;
#define max_size 10000
struct Node{
double v;
double t;
double time;
};
bool cmp(Node a,Node b)
{
    return a.time<b.time;
};
Node no[max_size];
int main()
{
    int N;//总人数
    while(cin>>N)
    {
        for(int i=0;i<N;i++)
        {
            cin>>no[i].v>>no[i].t;
            no[i].v/=3.6;
            if(no[i].t<0)
                no[i].v=0.1;
                no[i].time=no[i].t+(3000.0/(no[i].v));
        }
        sort(no,no+N,cmp);
        int ans=ceil(no[0].time);
        cout<<ans<<endl;
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5037520>

万恶的双“12”

2016年5月7日　　22:34

题目来源：https://biancheng.love/contest-ng/index.html#/34/problems

题目描述

想必大家双“11”剁掉的小爪子们已经长好了，所以双“12”又快到了。然而Nova君对剁手之痛记忆犹新，不想再次尝试，所以想进行穿越，企图避开双“12”。Nova君能力有限，只可以穿越到2015年随机的某一天，请大家算算，Nova君是否能逃过双“12”的浩劫？

输入

多组测试数据（组数不超过10组），对于每组数据，输入一行，包含两个正整数，分别代表2015年的某天日期（包含月和日，格式详见样例）

输出

对于每组数据，输出一行：

如果刚巧这天是双“12”，则输出 “Oh my god! It's today!!” ；

如果在双“12”之前，则输出 “It will come in x day(s) ORZ”，x代表距离双“12”的天数；

如果在双“12”之后，则输出“Lucky , it has passed!” ；

输入样例

```
12 18
12 12
1 1
```

输出样例

```
Lucky , it has passed!
Oh my god! It's today!!
It will come in 345 day(s) ORZ
```

解题分析：

双十一过去了，双十二就要来了。该怎么办呢？我们穿越吧，争取穿越到双十二之后，这样就不用剁手了呀。

签到题代码：

```
#include <bits/stdc++.h>

using namespace std;

int a[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};

int main()
{
    int m,d,ans;
    while(~scanf("%d%d",&m,&d))
    {
        ans=0;
        if(m==12&&d==12){
            printf("Oh my god! It's today!!\n");
        }
        else if(m==12&&d>12)
        {
            printf("Lucky , it has passed!\n");
        }
```

```
20          else {
21              if(m==12)
22                  ans=12-d;
23              else if(m<12){
24                  for(int i=m+1;i<=11;i++)
25                      ans+=a[i];
26                  ans+=a[m]-d+12;
27              }
28              printf("It will come in %d day(s) ORZ\n",ans);
29          }
30      }
31  }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5037393>

吃豆子过桥问题（转）

2016年5月7日　　22:34

转载博客：

http://www.cnblogs.com/webary/p/4827688.html

吃豆子过桥问题

本题来自于百度校招面试题，通过一个简单的智力问题理解递归问题的解法。

一：问题描述

一个人要过一座80米的桥，每走一米需要吃一颗豆子，他最多可以装60颗豆子，问最少需要吃多少颗豆子才能走完桥？

二：初步分析

1.一趟（不折回）最多只能走60米豆子就会被吃完；

2.如果有折回，必须保证能够返回到有豆子的地点，且在折回点放下的豆子尽量多；

3.尽可能少的折回（次数和折回的距离都要少，毕竟一趟折回就要多消耗一个来回的豆子）；

4.折回点的豆子数量要求至少能够走完剩余的部分；

三：具体分析

1.由条件1得一趟走不完，由条件3我们可以考虑折回尽量少的次数能否走完。

先考虑只折回一次，那么此时就要求得折回点的位置。

2.由条件3得折回的距离尽量短，那么可以考虑最短的折回距离，很明显应该是20米处；设折回点距离起始位置x米，则此时x=20;那么在x处，需要至少有60颗豆子，一次性拿60颗豆子走完最后一程。

那么问题就来了，怎么在x处囤积60颗豆子呢，假使他第一次信心满满的拿着60颗豆子，走到x处就只剩下40颗了，现在不够走完剩余的全程啊，只能放下一部分折回去再取了，那放多少呢，很明显由条件2得我们需要放下20颗，拿着20颗刚好可以折回到起点，继续拿着60颗豆子第二次来到x处，此时手上还有40颗，再捡起第一次放的20颗，共60颗豆子刚好可以走完全程。

在这种情况下，我们易知，他共走了（20）*3+（60）= 120米，故吃掉120颗豆子。

四：问题拓展

如果桥长81米呢？

此时我们如果还是只折回一次的话，在21米处还是得有60颗豆子，也就是需要搬运60颗豆子到21米处；分析得不可能只折回一次就搬运60颗豆子到21米处，因此需要再设置一个折返点y，此时x=21，0<y<x；

刚才考虑了80米的情况，那么我们可以假设现在在80米的基础上加了1米，因此可以设y=1，即先折回两趟到1米处，这样y处就有60*3-5*(1)=175颗豆子，再从y到x折返1趟，x处就有60*

2-3*(20)=60颗豆子，最后从x出发拿着60颗豆子就可以愉快的走过桥了。

因此他共走过5*(1)+3*(20)+(60) = 125米，故吃掉125颗豆子。

五：再次拓展

如果桥长n米，最多装m颗豆子，最少消耗f(n,m)与n和m的关系是什么呢？

此时问题突然就变得很复杂了，别急，我们分析一下刚才的思路，桥长从80米到81米，就是要在1米处放足够多豆子（当然只要不小于桥长80米时的消耗就行），那么这些豆子又需要从起点处运到1米处，那么在这1米内又需要消耗多少豆子呢？

我们可以考虑先把可装的最大豆子数m固定（假设还是60），当桥长0<n<=60米时，
f(n,m) = n；

当n=61时，需要在1米处囤积f(n-1,6)即f(60,60)=60颗豆子，囤积过程中需要往返一次，第一次到达1米处留下58颗豆子赶紧回去再取60颗到达1米处还剩59颗，现在有117颗足够走完最后60米。消耗豆子1*2+(1)+60 = 63颗豆子；

当n=62时，需要在1米处囤积f(n-1,m)即f(61,60)=63颗豆子，囤积过程中需要往返一次，第一次到达1米处留下58颗豆子赶紧回去再取60颗到达1米处还剩59颗，现在有117颗足够走完最后61米。消耗豆子1*2+(1)+63 = 66颗豆子；

以此类推...

那么问题来了，细心的你肯定发现了上面那种情况都是往返一次的，但是不是每次都只往返一次，比如当n=81时，该怎么确定往返的次数呢？

分析上面的递推关系我们可以知道，每次往返的趟数与f(n-1,m)的大小有关。分析往返过程易知：最后一次到达1米处剩59颗豆子，之前每次到达1米处可以放下58颗豆子，假设之前到达了1米处T次，则有：

59 + 58 * T >= f(n-1,60) ==> T >= [f(n-1,60)-59]/58

由于T必须是整数，故T = ceil((f(n-1,60)-59) / 58)，ceil(x)表示对x向上取整，即不小于x的最小整数

另一种表示方式为T = floor((f(n-1,60)-2) / 58)，这种写法是为了方便编程实现，整型数的除法会自动向下取整

之前往返T次消耗掉2T颗豆子，最后一次到达1米处消耗1颗豆子，故总消耗：
f(n-1,60) + 2T + 1颗豆子

这时候我们考虑将固定为60的m扩展为任意m，则有：f(n,m) = f(n-1,m) + ceil(f(n-1,m)-(m-1)) / (m-2)) * 2+ 1 = f(n-1,m) + (f(n-1,m)-2) / (m-2) * 2+ 1 ;

好了到这里，这个问题也彻底解决完了，现在就让我们用简短的代码来实现这个过程吧！

六：编程实现

```
#include<cmath>
#include<iostream>
using namespace std;
typedef unsigned long long int64;

//参数说明:length为桥的长度,maxNum为最大可带的豆子数
//递归实现
int64 getMinConsume(int length, int maxNum) {
    if(length <= maxNum)
```

```
10          return length;
11      int64 get_n_1 = getMinConsume(length-1,maxNum); //上一次的豆子消耗
12      return get_n_1 + (get_n_1-2)/(maxNum-2) * 2 + 1;
13  }
14  //第二种公式实现——递归
15  int64 getMinConsume2(int length, int maxNum) {
16      if(length <= maxNum)
17          return length;
18      int64 get_n_1 = getMinConsume2(length-1,maxNum);  //上一次的豆子消耗
19      return get_n_1 + ceil((double)(get_n_1-maxNum+1)/(maxNum-2)) * 2 + 1;
20  }
21  //非递归的实现方式
22  int64 getMinConsume_NoRecursive(int length, int maxNum) {
23      if(length <= maxNum)
24          return length;
25      int64 result = maxNum, i;
26      for(i=maxNum; i<length; i++)
27          result += (result-2)/(maxNum-2) * 2 + 1;
28      return result;
29  }
30
31  int main()
32  {
33      int maxNum=60, length;
34      for(length=50; length<201; length++)
35          cout<<length<<"米至少消耗"<<getMinConsume2(length,maxNum)<<"颗豆子!"<<endl;
36      return 0;
37  }
38
39  int main2()
40  {
41      while(1){
42          cout<<"请输入最大可带的豆子数量和桥的长度: ";
43          int maxNum, length;
44          if(!(cin>>maxNum>>length))
45              break;
46          cout<<"至少消耗"<<getMinConsume_NoRecursive(length,maxNum)<<"颗豆
子!"<<endl;
47      }
48      return 0;
49  }
```

来自 <http://i.cnblogs.com/EditArticles.aspx?postid=5023363>

最大流（代码以及博客云集版）

2016年5月7日　　22:35

codes：

解决最大流问题参考代码1：

```
#include <bits/stdc++.h>
#define MAX 1100
#define INF 0x3f3f3f3f
using namespace std;
struct Node{
    int to;//终点
    int cap;//容量
    int rev;//反向边
};

vector<Node> v[MAX];
bool visited[MAX];

void Add_Node(int from, int to, int cap)
{
    v[from].push_back((Node){to,cap,v[to].size()});
    v[to].push_back((Node){from,0,v[from].size()-1});
}

int DFS(int s, int t,int f)
{
    if(s==t)
        return f;
    visited[s]=true;
    for(int i=0;i<v[s].size();i++)
    {
        Node &temp=v[s][i];
        if(visited[temp.to]==false &&temp.cap>0)
        {
            int d=DFS(temp.to,t,min(f,temp.cap));
            if(d>0)
            {
                temp.cap-=d;
                v[temp.to][temp.rev].cap+=d;
                return d;
            }
        }
    }
    return 0;
}

int Max_Flow(int s,int t)
{
    int flow=0;
    for(;;)
    {
        memset(visited,false,sizeof(visited));
        int f=DFS(s,t,INF);
        if(f==0)
            return flow;
        flow+=f;
    }
}
int main()
{
    int n,s,t;
    int start,over,capacity;
    while(~scanf("%d%d%d",&n,&s,&t))
    {
        memset(v,0,sizeof(v));
```

```
61          for(int i=0;i<n;i++)
62          {
63              scanf("%d%d%d",&start,&over,&capacity);
64              Add_Node(start,over,capacity);
65          }
66          printf("%d\n",Max_Flow(s,t));
67      }
68  }
```

解决最大流问题参考代码2：

```
 1  #include <iostream>
 2  #include <queue>
 3  #include<string.h>
 4  using namespace std;
 5  #define arraysize 201
 6  int maxData = 0x7fffffff;
 7  int capacity[arraysize][arraysize]; //记录残留网络的容量
 8  int flow[arraysize];                //标记从源点到当前节点实际还剩多少流量可用
 9  int pre[arraysize];                 //标记在这条路径上当前节点的前驱,同时标记该节点是否
在队列中
10  int n,m;
11  queue<int> myqueue;
12  int BFS(int src,int des)
13  {
14      int i,j;
15      while(!myqueue.empty())        //队列清空
16          myqueue.pop();
17      for(i=1;i<=m;++i)
18      {
19          pre[i]=-1;
20      }
21      pre[src]=0;
22      flow[src]= maxData;//初始化为最大值
23      myqueue.push(src);
24      while(!myqueue.empty())
25      {
26          int index = myqueue.front();
27          myqueue.pop();
28          if(index == des)            //找到了增广路径
29              break;
30          for(i=1;i<=m;++i)
31          {
32              if(i!=src && capacity[index][i]>0 && pre[i]==-1)
33              {
34                  pre[i] = index; //记录前驱!!!对应的是后继
35                  flow[i] = min(capacity[index][i],flow[index]);    //关键：迭代的找
到增量(change)
36                  myqueue.push(i);//push进队列
37              }
38          }
39      }
40      if(pre[des]==-1)       //残留图中不再存在增广路径
41          return -1;
42      else
43          return flow[des];
44  }
45  int maxFlow(int src,int des)
46  {
47      int increasement= 0;
48      int sumflow = 0;
49      while((increasement=BFS(src,des))!=-1)
50      {
51           int k = des;           //利用前驱寻找路径
```

```
52          while(k!=src)
53          {
54               int last = pre[k];
55               capacity[last][k] -= increasement; //改变正向边的容量
56               capacity[k][last] += increasement; //改变反向边的容量
57               k = last;
58          }
59          sumflow += increasement;
60     }
61     return sumflow;
62 }
63 int main()
64 {
65     int i,j;
66     int start,end,ci;
67     while(cin>>n>>m)
68     {
69         memset(capacity,0,sizeof(capacity));
70         memset(flow,0,sizeof(flow));
71         for(i=0;i<n;++i)
72         {
73             cin>>start>>end>>ci;
74             if(start == end)                //考虑起点终点相同的情况
75                continue;
76             capacity[start][end] +=ci;      //此处注意可能出现多条同一起点终点的情况
77         }
78         cout<<maxFlow(0,m)<<endl;
79     }
80     return 0;
81 }
```

Blogs:

基础知识博客：
http://www.cnblogs.com/luweiseu/archive/2012/07/14/2591573.html

模板博客提供：
http://blog.csdn.net/y990041769/article/details/21026445

详解博客（推荐）：
http://www.cnblogs.com/zsboy/archive/2013/01/27/2878810.html

详解博客（推荐）：
http://www.cnblogs.com/kuangbin/archive/2011/07/26/2117636.html

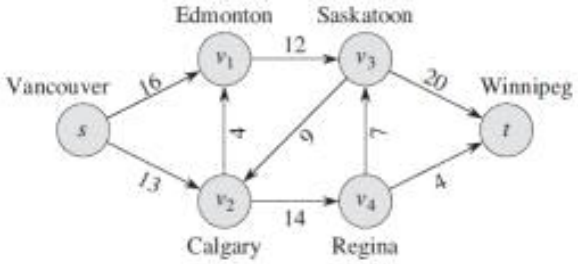
各种算法汇集博客：
http://www.cnblogs.com/longdouhzt/archive/2012/05/20/2510753.html

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5011491>

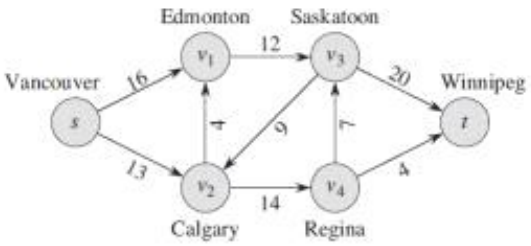
最大流（图片版详解） ------ 前方高能预警

2016年5月7日　　22:35

- Flow networks
  - Liquids flowing through pipes
  - Parts through assembly lines
  - Current through electrical networks
  - Information through communication networks
  - Cars through highway traffic networks

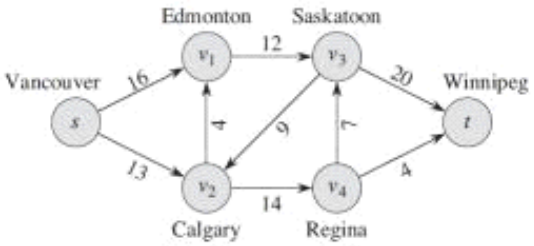


- **Capacity**: a maximum rate at which the material can flow through the conduit.
- **Flow conservation**: the rate at which material enters a vertex must equal the rate at which it leaves the vertex.
- **Maximum-flow problem**: we wish to compute the greatest rate at which we can ship material from the source to the sink without violating any capacity constraints.



Flow networks and flows

- A flow network $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. If $(u, v) \notin E$, $c(u, v) = 0$.
- Each vertex lies on some path from the source to the sink.
- ***source s***
- ***sink t***
- A ***flow*** in G is a real-valued function $f: V \times V \rightarrow R$ that satisfies The following three properties:



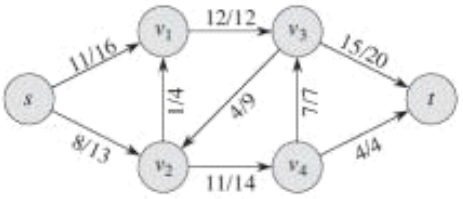
Flow networks and flows

- A ***flow*** in G is a real-valued function $f: V \times V \rightarrow R$ that satisfies The following three properties:
  - **Capacity constraint**: For all $u, v \in V$, we require $0 \leq f(u, v) \leq c(u, v)$.
  - **Skew symmetry**: For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.
  - **Flow conservation**: For all $u \in V - \{s, t\}$, we require

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v).$$

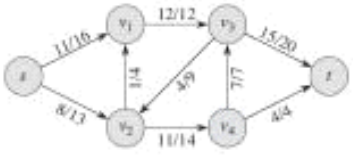
" flow in equals flow out. "

When $(u, v) \notin E$, there can be no flow from u to v, and $f(u, v) = 0$.



Flow networks and flows

- $f(u, v)$: the flow from vertex u to v.



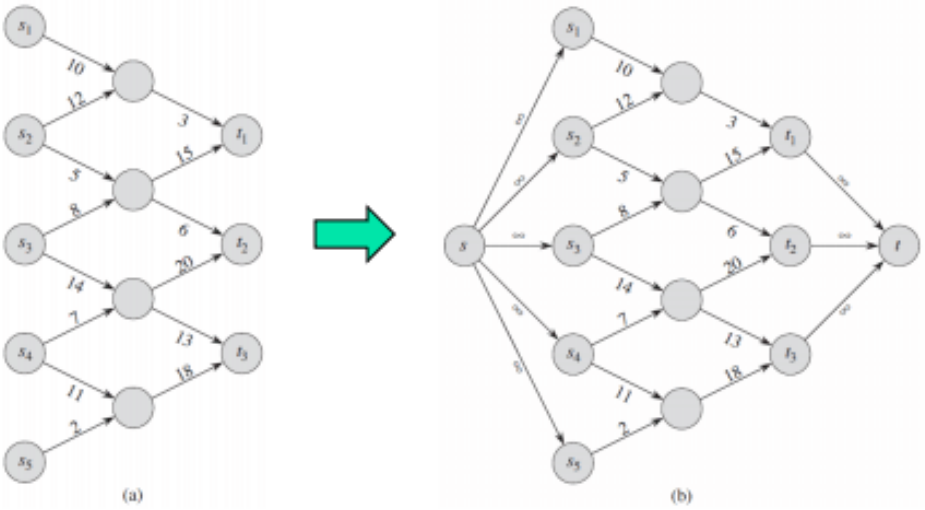
- The value $|f|$ of a flow f is defined as

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s).$$

that is, the total flow out of the source minus the flow into the source. (Here, the $|\cdot|$ notation denotes flow value, not absolute value.)

- **Maximum-flow problem**: we are given a flow network G with source s and sink t, and we wish to find a flow of maximum value.

Networks with multiple sources and sinks

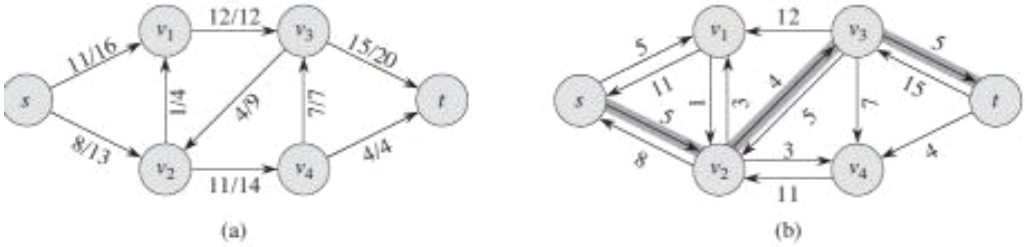


Residual networks

- **residual capacity**

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E, \\ f(v,u) & \text{if } (v,u) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

- Example: if $c(u, v) = 16$ and $f(u, v) = 11$, then we can increase $f(u, v)$ by up to $c_f(u, v) = 5$ units before we exceed the capacity constraint on edge (u, v). We also wish to allow an algorithm to return up to 11 units of flow from v to u, and hence $c_f(v, u) = 11$.



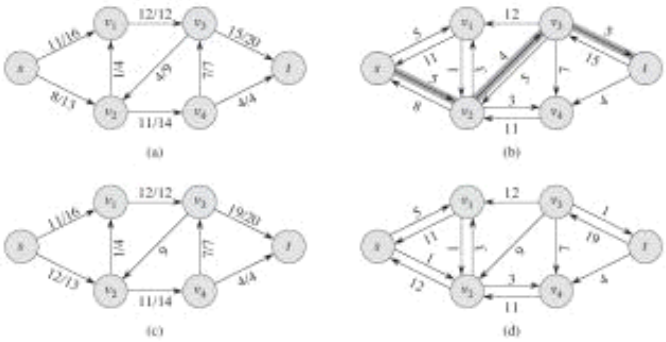
Residual networks

- **residual capacity**

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E, \\ f(v,u) & \text{if } (v,u) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

- **residual network:** $G_f = (V, E_f)$, where

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

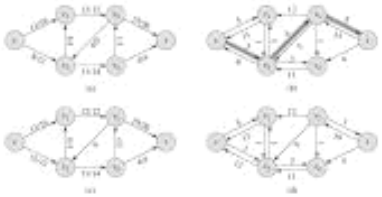


Residual networks

- **Lemma 26.2** Let $G = (V, E)$ be a flow network with source s and sink t, and let f be a flow in G. Let G_f be the residual network of G induced by f, and let f' be a flow in G_f. Then the flow sum $f + f'$ defined by equation (26.4) is a flow in G with value

  $$|f + f'| = |f| + |f'|.$$

  < (26.4) $(f_1 + f_2)(u, v) = f_1(u, v) + f_2(u, v)$ >



- ***Proof*** We must verify that **skew symmetry**, the **capacity constraints**, and **flow conservation** are obeyed.

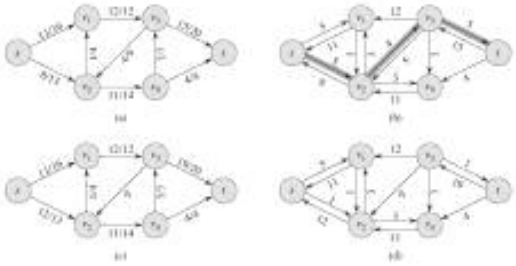
Augmenting paths

- An **augmenting path** p is a simple path from s to t in the residual network G_f.
- ***residual capacity*** of p : the maximum amount by which we can increase the flow on each edge in the augmenting path p .

  $c_f(p) = \min \{c_f(u, v) : (u, v) \text{ is on } p\}$.
- **Lemma 26.3** Let $G = (V, E)$ be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Define a function $f_p : V \times V \to \mathbf{R}$ by

$$f_p(u,v) = \begin{cases} c_f(p), & \text{if } (u,v) \text{ is on } p, \\ -c_f(p), & \text{if } (v,u) \text{ is on } p, \\ 0, & \text{otherwise.} \end{cases}$$

Then, f_p is a flow in G_f with value $|f_p| = c_f(p) > 0$.

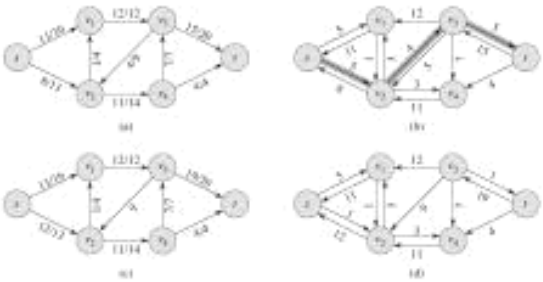


Augmenting paths

$$f_p(u,v) = \begin{cases} c_f(p), & \text{if } (u,v) \text{ is on } p, \\ -c_f(p), & \text{if } (v,u) \text{ is on } p, \\ 0, & \text{otherwise.} \end{cases}$$

- **Corollary 26.4** Let $G = (V, E)$ be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Define a function $f' : V \times V \to \mathbf{R}$ by $f' = f + f_p$. Then f' is a flow in G with value $|f'| = |f| + |f_p| > |f|$.

  Proof Immediate from Lemmas 26.2 and 26.3.



13

Cuts of flow networks

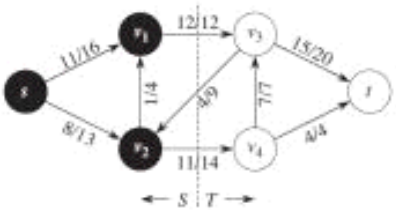
- A ***cut*** (S, T) of flow network $G = (V, E)$ is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$. (source is in S, sink is in T.)
- If f is a flow, then the ***net flow*** across the cut (S, T) is defined to be $f(S, T)$.

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u)$$

- The ***capacity*** of the cut (S, T) is $c(S, T)$.

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

- A ***minimum cut*** of a network is a cut whose capacity is minimum over all cuts of the network.（一个网络的最小割是网络中具有最小容量的割）



$f(S, T) = ?$

$c(S, T) = ?$

14

Cuts of flow networks

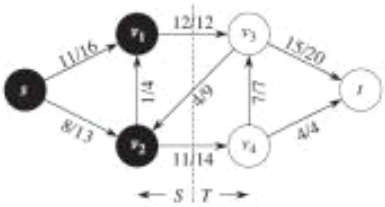
- **Lemma 26.5** Let f be a flow in a flow network G with source s and sink t, and let (S, T) be a cut of G. Then the net flow across (S, T) is $f(S, T) = |f|$.

  Proof …

- **Corollary 26.6** The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.

（任意割的容量都是流的上界）

Proof …



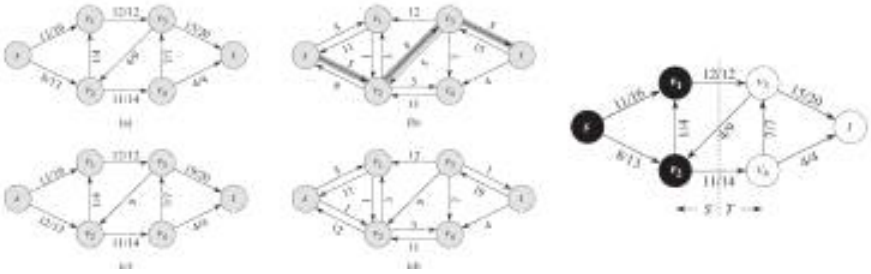
Cuts of flow networks

- **Theorem 26.7: (Max-flow min-cut theorem)**

  If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

  1. f is a maximum flow in G.
  2. The residual network G_f contains no augmenting paths.
  3. $|f| = c(S, T)$ for some cut (S, T) of G.

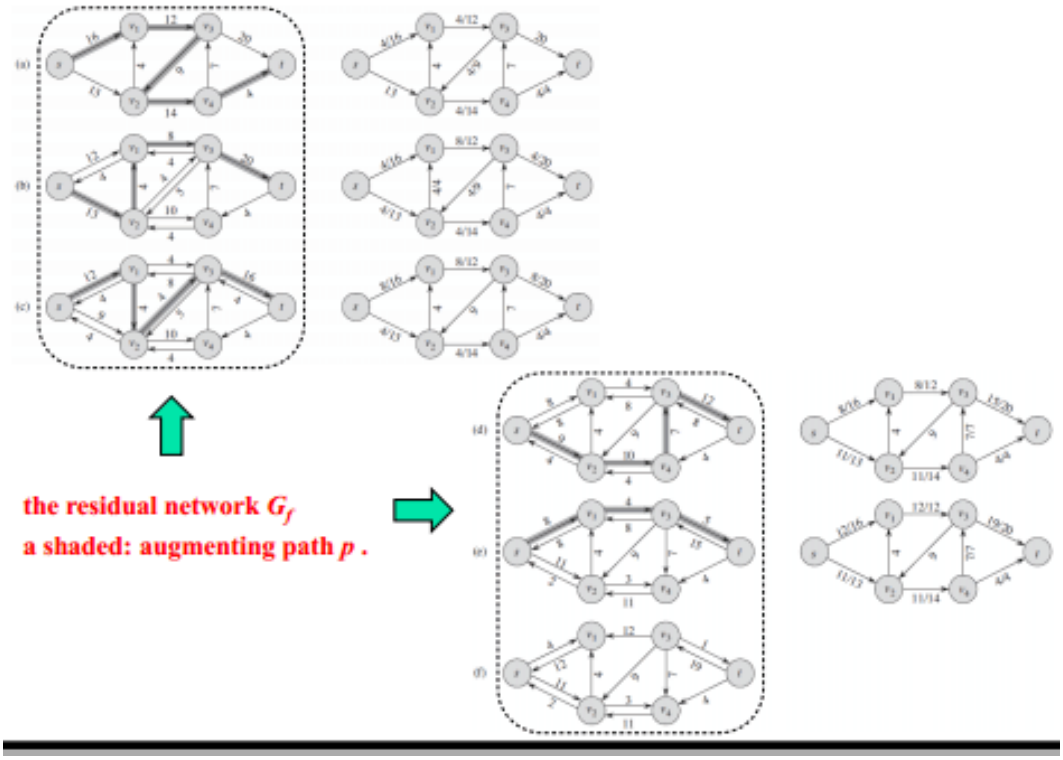
  Proof …



The basic Ford-Fulkerson algorithm

```
FORD-FULKERSON(G, s, t)
1  for each edge (u, v) ∈ E[G]
2      f[u, v] ← 0
3      f[v, u] ← 0
4  while there exists a path p from s to t in the residual network G_f
5      c_f(p) ← min {c_f(u, v) : (u, v) is in p}
6      for each edge (u, v) in p
7          f[u, v] ← f[u, v] + c_f(p)
8          f[v, u] ← -f[u, v]
```

26.2 The Ford-Fulkerson method - F-F Algo



the residual network G_f

a shaded: augmenting path p .

Analysis of Ford-Fulkerson

```
FORD-FULKERSON(G, s, t)
1  for each edge (u, v) ∈ E[G]
2      f[u, v] ← 0
3      f[v, u] ← 0
4  while there exists a path p from s to t in the residual network G_f
5      c_f(p) ← min {c_f(u, v) : (u, v) is in p}
6      for each edge (u, v) in p
7          f[u, v] ← f[u, v] + c_f(p)
8          f[v, u] ← -f[u, v]
```

$O(E|f^*|)$

- **When the capacities are integral and the optimal flow value $|f^*|$ is small, the running time of the Ford-Fulkerson algorithm is good.**

The Edmonds-Karp algorithm

```
FORD-FULKERSON(G, s, t)
1  for each edge (u, v) ∈ E[G]
2      f[u, v] ← 0
3      f[v, u] ← 0
4  while there exists a path p from s to t in the residual network G_f
5      c_f(p) ← min {c_f(u, v) : (u, v) is in p}
6      for each edge (u, v) in p
7          f[u, v] ← f[u, v] + c_f(p)
8          f[v, u] ← -f[u, v]
```

$O(E|f^*|)$

- **We can improve the bound on F-F by finding the augmenting path p in line 4 with a breadth-first search. That is, we choose p as a shortest path from s to t in the residual network, where each edge has unit distance (weight). We call the F-F method so implemented the Edmonds-Karp algorithm. The E-K algorithm runs in $O(VE^2)$ time.** ***Proof*** **…?**

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5011481>

Android技巧分享——Android开发超好用工具吐血推荐(转)

2016年5月7日　　22:35

转载：http://www.cnblogs.com/JohnTsai

Github上好用的资源太多太多，要想成为一名优秀的开发者，学会寻找并使用别人已经造好的轮子是一项必不可少的技能。当然，技术成熟后，自己造轮子更重要。

今天分享三个Android开发必备的工具应用：

1.Libraries for developers

这是一款由国外开发者整理的一款应用，收集了**Android**各种各样的开源库，能直接在手机上查看开源库运行效果的**demo**。

各种各样常用的开源库都按种类划分好了，并且有**Github**地址供我们**Star**学习。这个应用不时会更新一下，加入新的开源库，如**Android5.0**的**MaterialDesign**。

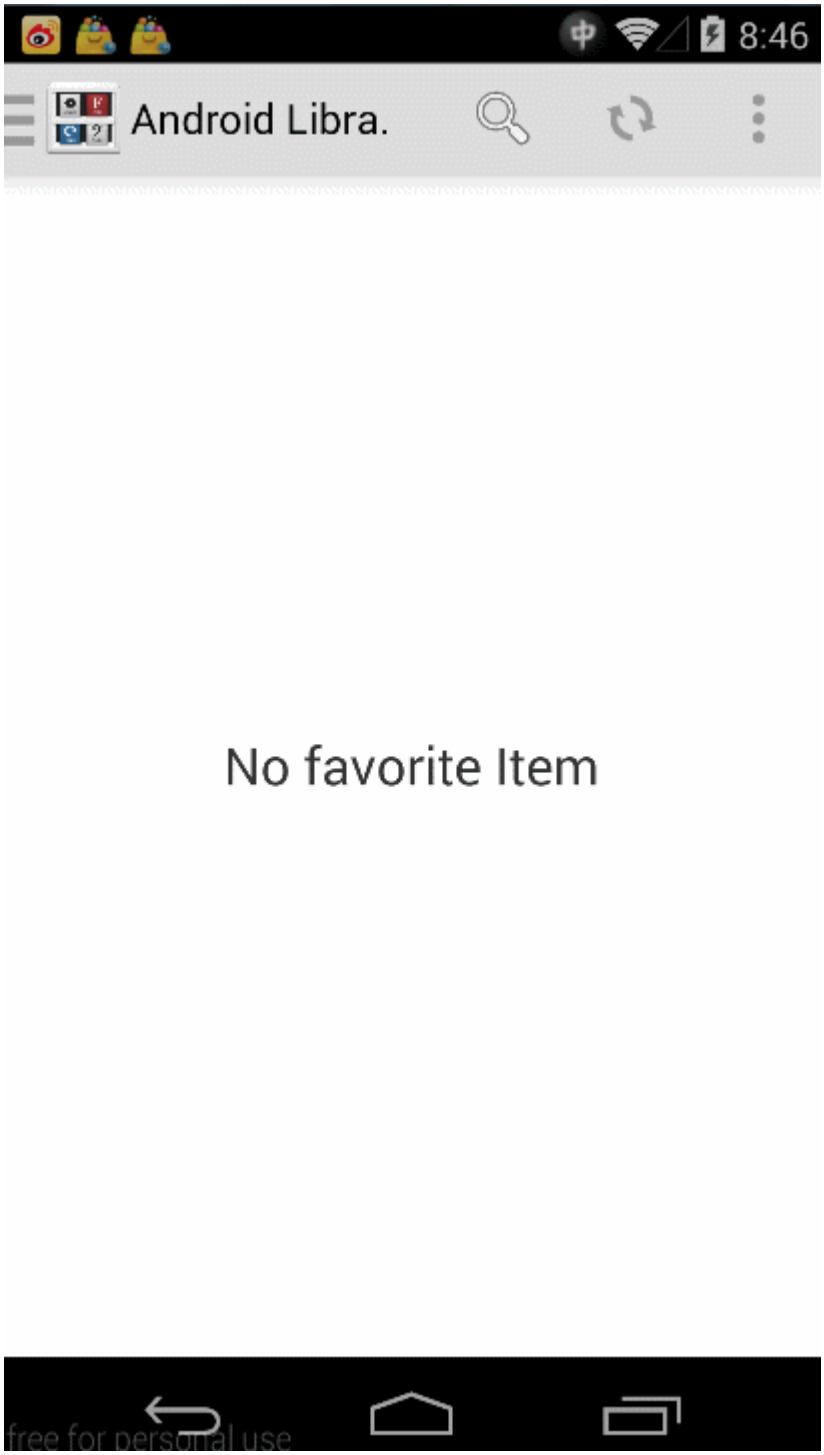
我们可以把这个应用安装在自己的手机上，平时没事的时候可以看看各种各样的**Demo**，做做笔记。更赞的是，这个应用中还有**Android**开发中常用的代码段。

Google Play地址：**https://play.google.com/store/apps/details?id=com.desarrollodroide.repos**

官网地址：**http://desarrollodroide.com/apps.html**

2.Android Libraries

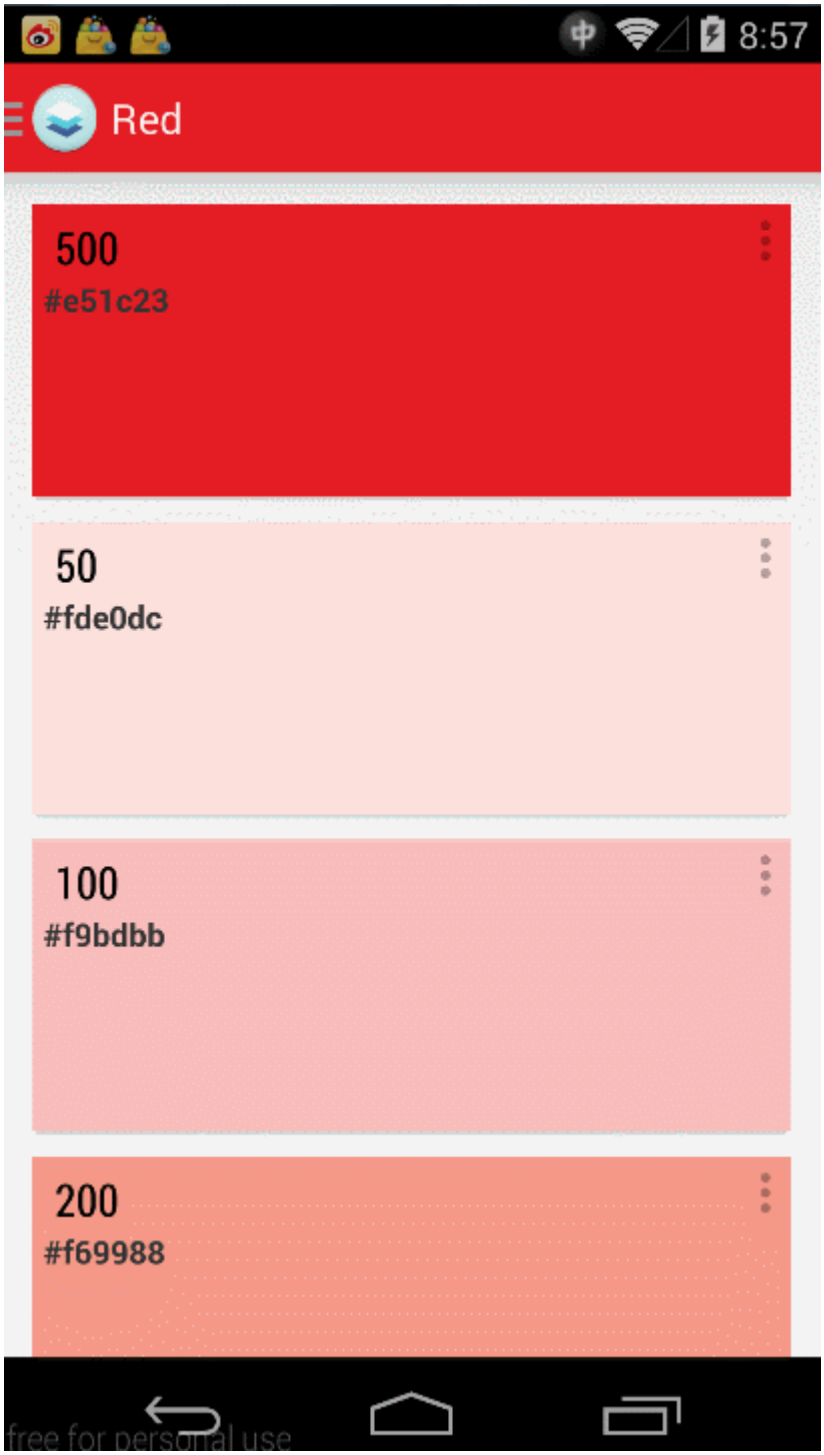
功能与第一款应用非常类似。



GooglePlay地址：https://play.google.com/store/apps/details?id=com.rise.livdev

3.Material Design Color

这款应用对美工和个人开发者有用，主要提供各种颜色，都是**MaterialDesign**常用的颜色。



GooglePlay地址： https://play.google.com/store/apps/details?id=fr.hozakan.materialdesigncolorpalette

由于GooglePlay不好访问在此将3个应用分享到百度网盘供大家下载

链接：http://pan.baidu.com/s/1ntqQULF 密码：h36q

[Android技巧分享系列]

1.Android技巧分享——让官方模拟器和genymotion虚拟机飞起来

2.Android技巧分享——如何用电脑下载在Google play中应用的apk文件

3.Android技巧分享——Android开发超好用工具吐血推荐

想及时获取最新最有用的Android开发干货，请**关注我**

转载请注明网址：http://www.cnblogs.com/JohnTsai

如果觉得本文对你的学习工作有所帮助，不妨在右下方点**推荐一下**，谢谢。

联系我：JohnTsai.Work@gmail.com

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5007669>

忙碌的Nova君 （活动安排问题、贪心算法）

2016年5月7日　　22:38

题目描述

理论上，Nova君是个大闲人，但每天还是有一大堆事要干，大作业啦，创新杯啦，游戏啦，出题坑人啦，balabala.....然而精力有限，Nova君同一时间只能做一件事，并不能一心二用。假设现在有N项工作等待Nova君完成，分别在 Si 时刻开始，在 Ti 时刻结束，对于每项工作可以选择做或者不做，但不可以同时选择时间重叠的工作（即使是开始的瞬间和结束的瞬间重叠也是不允许的）。Nova君自然希望尽量多做一些事情，那么最多能做几件事呢？

输入

多组测试数据（数据组数不超过10），对于每组数据，第一行输入一个正整数N，代表可选的工作数量，接下来输入N行，每行两个正整数，代表第 i 个工作的开始时间 si 和结束时间 ti 。

1<=N<=100000 | 1<=si<=ti<=10^9

输出

对于每组数据，输出一行，为可完成的工作的最大数量。

输入样例

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

输出样例

```
3
```

题目来源 ：http://biancheng.love/contest/23/problem/E/index

解题思路：按照活动结束时间进行排序，先结束的排在前列，后结束的排在后面。

每次放入先结束的活动，将其的结束时间和下一个活动的开始时间进行比较，如果满足下一个活动在上个活动结束之后开始那么将该活动放入进来，依次进行操作，直到最后一个活动为止。输出结果为活动最多数目。

本题代码：

```
#include <bits/stdc++.h>
#define max_size 10010
using namespace std;
int order[max_size];

struct node{
int start,end;
int id;
};

node act[max_size];

bool cmp(node a,node b)
{
    return a.end<b.end;
};

```

```
int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        for(int i=0;i<n;i++)
           {
               scanf("%d%d",&act[i].start,&act[i].end);
                act[i].id=i+1;
           }
        sort(act,act+n,cmp);
        order[0]=0;
        int number=1;
        for(int i=1;i<n;i++)
        {
            if(act[i].start>=act[order[number-1]].end)
            order[number++]=i;
        }
        cout<<number<<endl;
    }
}
```

在之前的博客文章讲解活动安排的区别点：

1、之前博客讲述了dp求活动安排问题，也讲述了使用贪心求解。但是之前的问题默认输入的活动已经按照活动结束时间进行了排序，现在这个题目并没有排序。

2、之前的活动可以是下一个活动的开始正好是上一个活动的结束，但是这道题目要求不能够有重合。

下面给出之前博客的链接：

http://www.cnblogs.com/zpfbuaa/p/4951105.html

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5002078>

出口胡萝卜 (贪心加暴力算法)

2016年5月7日　　22:38

题目描述

终于，Nova君饲养的小兔子们成精了，他们可以帮着赚钱啦。春天，兔子们播下胡萝卜的种子；夏天，兔子们耕耘；秋天，兔子们收获硕大无比的胡萝卜；冬天，兔子们把胡萝卜包装好，远销海内外。现在问题来了，Nova君希望削减包装的运输的成本，因而要尽可能少的用包装箱。兔子们种植的胡萝卜很奇葩，他们的长度都为H，但粗细度不一致，为了方便，事先把胡萝卜风干并且切成规则的长方体，规格为AAH，H为长度，A为横切面的边长。假设胡萝卜只有1x1xH，2x2xH，3x3xH， 4x4xH， 5x5xH， 6x6xH这6种规格，通常使用一个 6x6xH的长方体包裹包装然后邮寄给客户。

输入

多组测试数组（组数少于100），对于每组数据，输入一行，为6个正整数（int范围内），分别代表规格为 1x1xH 至6x6xH 这六种胡萝卜的数量。

输出

对于每组数据，输出一行，为所需最少的包裹数。

输入样例

```
0 0 4 0 0 1
7 5 1 0 0 0
```

输出样例

```
2
1
```

题目来源：

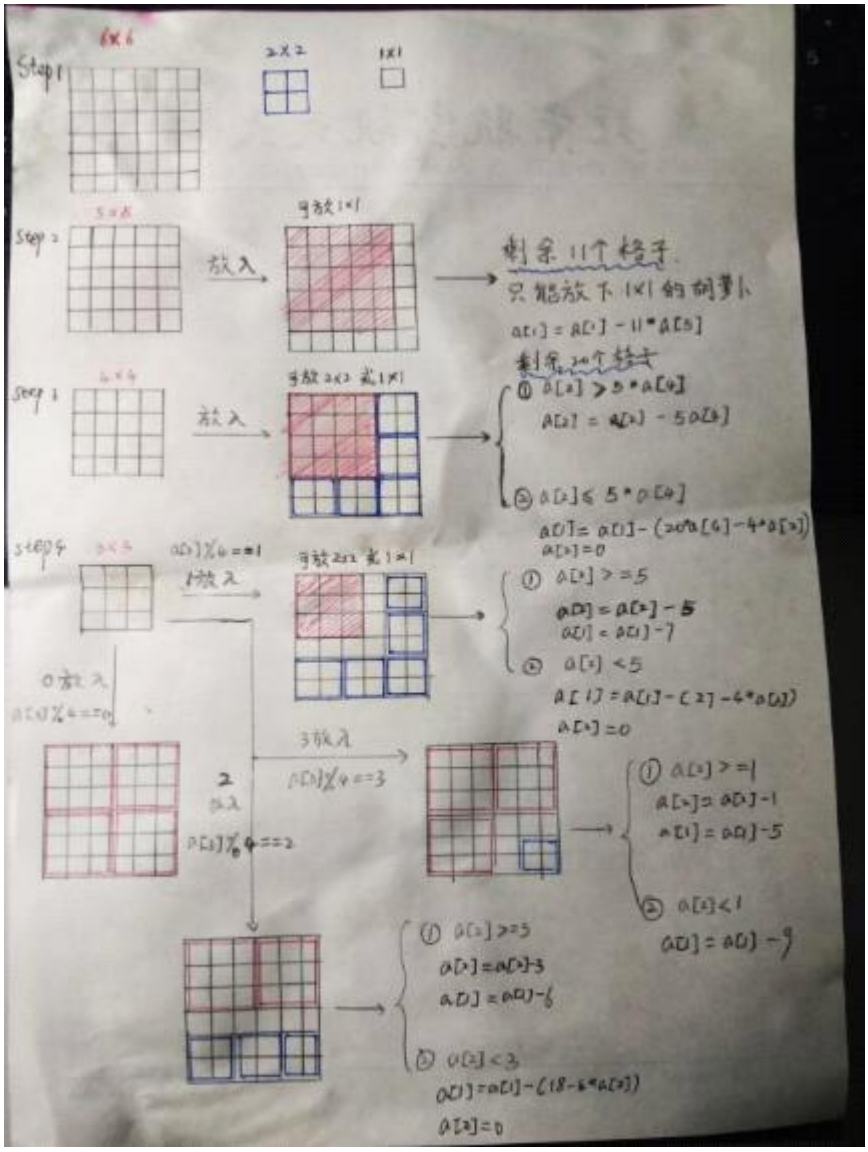
http://biancheng.love/contest/23/problem/C/index

解题思路：同样采用贪心算法，不过这次的贪心算法很麻烦，但是思路很清晰。（好比做高中数学题）

首先讨论一下我们应该先发哪一种萝卜。

电脑没电了，先给出代码。白天补上详解。

详解来的有点晚了。（像素不高，但是过程还是表示滴挺清楚的）



代码：

```
#include <bits/stdc++.h>

using namespace std;
int a[7];

int main()
{
    long long ans;
    while(~scanf("%d%d%d%d%d%d",&a[1],&a[2],&a[3],&a[4],&a[5],&a[6]))
    {
        ans=a[6];
        if(a[5]>0)
        {
            ans=ans+a[5];
            a[1]=a[1]-11*a[5];
        }
        if(a[4]>0)
        {
            ans=ans+a[4];
            if(a[2]>=5*a[4])
                a[2]=a[2]-5*a[4];
            else
            {
                a[1]=a[1]-(20*a[4]-4*a[2]);
                a[2]=0;
            }
        }
        if(a[3]>0)
        {
            if(a[3]%4==0)
```

```
                ans=ans+a[3]/4;
            else
            {
                ans=ans+a[3]/4+1;
                if(a[3]%4==1)
                {
                    if(a[2]>=5)
                    {
                        a[2]=a[2]-5;
                        a[1]=a[1]-7;
                    }
                    else
                    {
                        a[1]=a[1]-(27-4*a[2]);
                        a[2]=0;
                    }
                }
                if(a[3]%4==2)
                {
                    if(a[2]>=3)
                    {
                        a[2]=a[2]-3;
                        a[1]=a[1]-6;
                    }
                    else
                    {
                        a[1]=a[1]-(18-4*a[2]);
                        a[2]=0;
                    }
                }
                if(a[3]%4==3)
                {
                    if(a[2]>=1)
                    {
                        a[2]=a[2]-1;
                        a[1]=a[1]-5;
                    }
                    else
                    {
                        a[1]=a[1]-9;
                    }
                }
            }
        }
        if(a[1]<0) a[1]=0;
        if((a[2]*4+a[1])%36==0)
            ans=ans+(a[2]*4+a[1])/36;
        else
            ans=ans+(a[2]*4+a[1])/36+1;
        printf("%lld\n",ans);
    }
}
```

简化代码：（思路相同）

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int a,b,c,d,e,f,x,y;
    int sum;
    int u[4]= {0, 5, 3, 1}; //对应的是当c%4 = 0, 1, 2, 3 时，剩余空间可放2*2大小的个数

    while(~scanf("%d %d %d %d %d %d",&a,&b,&c,&d,&e,&f))
    {
        sum = 0;
        if (a == 0 && b == 0 && c == 0 && d == 0 && e == 0 && f == 0)
            break;
        sum = f; //每个包裹只能转一个6*6的
        sum += e;//每个包裹只能装一个5*5的
        sum += d;//每个包裹只能装一个4*4的
        sum += (c+3)/4;//每个包裹可以装4个 3*3的
        y = 5*d + u[c%4];
        if(b>y)
```

```
21              sum += (b-y+8)/9;
22          x = sum *36 - f*36 -e*25 - d*16 - c*9 - b*4;
23          if(a>x)
24              sum += (a-x+35)/36;
25
26          printf("%d\n",sum);
27      }
28  }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5002071>

机智零崎不会没梗III (摊还分析)

2016年5月7日　　22:38

题目描述

零崎总是说自己有一百种梗可玩，然而其实都是假的，是化学成分的，是特技。想要给摊还分析加个梗，实在是不好想，因为这个内容并不是什么算法，而是算法分析。

不过既然还得考，那么没有办法……

“势能法”是摊还分析中一种比较简单常用的方法，而且容易理解。现在零崎有K个硬币，规定每次“翻动”操作只能从最右侧开始翻转硬币，且如果把一个硬币从正面向上翻到背面向上，则需要对其左侧相邻的那个硬币也执行“翻动”操作(翻至最左则结束)。定义硬币组的势为硬币中正面向上的硬币的个数。现在要求你求出从某个给定的硬币组状态之后连续N个“翻动”操作的摊还代价。

硬币组的初始状态以一组数表示，0代表反面，1代表正面。

输入

多组测试数据。

每组测试数据共两行：

第一行K+1个正整数，分别为硬币组中硬币数K和初始状态(0<K<20)；

第二行一个正整数，为题目要求你求出的“翻动”操作的个数N（0<N<30）。

输出

每组测试数据输出一行，此行第i个数输出初始状态后第i个操作的摊还代价（1<=i<=N）。

输入样例

```
3 0 0 0
2
```

输出样例

```
2 2
```

题目来源：

http://biancheng.love/contest/23/problem/F/index

所谓摊还分析：求数据结构中的一个操作序列中所执行的所有操作的平均时间，来评价操作的代价。我们可以说明一个操作的平均代价很低，即使序列中某个单一操作的代价很高。瘫痪分析不同于平均情况分析，它不涉及到概率问题，它可以保证最坏的情况下每个操作的平均性能。

解题思路：在计算摊还代价时可以采用：聚合分析、核算法、势能算法。参考算法导论书中的例子：二进制计数器递增问题。通过题目要求可以得到，反转硬币其实也就是实现二进制计数器的递增问题，在摊还代价计算中得到如果反转到全为0，摊还代价为0，其他情况均为1，因此可以采用简单粗暴的方法。计算对应二进制的十进制数，由于反转硬币的摊还代价是有周期的（在给定二进制位数之后，周期为2^k）；因此只需要判断是输出0还是输出2就可以了。

本题代码：

```
#include <bits/stdc++.h>
#define max_size  21
int a[max_size];
using namespace std;
int main()
{
    int k,num,flag,N;
    while(~scanf("%d",&k))
    {
        num=0;
        flag=pow(2,k);
        for(int i=1;i<=k;i++)
            scanf("%d",&a[i]);
        for(int i=1;i<=k;i++)
            num+=a[i]*pow(2,k-i);
        scanf("%d",&N);
        int temp=flag-num;
        for(int i=1;i<=N;i++)
        {
            if(i<=temp-1)
            {
                if((i-1)==temp)
                    printf("0 ");
                else
                    printf("2 ");
            }
            else if(i>=temp)
            {
                if((i+num)%flag==0)
                    printf("0 ");
                else
                    printf("2 ");
            }
        }
        printf("\n");
    }
}
```

采用判断是否所有位数都为0，来进行输出的另外一种方法：

```
#include<iostream>
using namespace std;
int main()
{
    int N;
    while(cin>>N)
    {
        int A[N],k;
        for(int i=0; i<N; i++)
        {
            cin>>A[i];
        }
        cin>>k;
        int j=N-1;
        for(int i=1; i<=k; i++)
        {
            while(j>=0 && A[j]==1)
            {
                A[j]=0;
                j--;
            }
            if(j>=0)
            {
                A[j]=1;
                cout<<"2 ";
            }
            else
            {
                for(int k=0; k<N; k++)
                    A[k]=0;
                cout<<"0 ";
            }
            j=N-1;
```

```
34          }
35          cout<<endl;
36      }
37  }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5002066>

机智零崎不会没梗II (哈夫曼编码、优先队列)

2016年5月7日 22:38

题目描述

你满心欢喜的召唤出了外星生物，以为可以变身超人拥有强大力量战胜一切怪兽，然而面对着身前高大的外星生物你一脸茫然，因为，你懂M78星云语吗？不过不用担心，因为零崎非常机智，他给出了关键性的提示：“讲道理，日语可是全宇宙通用语，所以为什么不试试和外星人讲日语呢？”

不过现在外星生物说的话都是“!@#$%^&%#%I&!.....”这样的东西，你要怎么转换成日语呢？

作位全宇宙通用的日语，自然有一套万能的转换算法，那就是Huffman编码转换！当然了这肯定不是普通的Huffman编码转换，而是根据不同的编码长度转换为不同的假名。

输入

第一行为一个整数t，接下来t行数据。1<=t<=100

每组输入数据为一个外星语字符串，为了表示方便，暂时使用大小写英文字母代替外星字母。字符串长度不超过2000

输出

对于每组数据，输出对应的二进制Huffman编码总长度

输入样例

```
2
abababac
abcdefg
```

输出样例

```
12
20
```

题目来源：

http://biancheng.love/contest/23/problem/D/index

解题思路:在前面的哈夫曼树构造以及哈夫曼编码和解码时，已经讲述了哈夫曼编码的方式。现在我们需要得到哈夫曼编码的长度。回忆一下在解码过程中的算法，如果是0向左搜索，如果是1向右搜索。因此一个字符在一个构造好的哈夫曼树中都是居于叶子结点。意思就是树的各个分叉的最低端。因此每个字符的长度也就是该字符在哈夫曼树中的深度。希望能够理解这一点。

在明白字符长度就是字符深度之后开始重新分析一下哈夫曼的树的构造过程。哈夫曼树的建立时通过对每个字符出现频率的统计来设计的，不同的出现频率会对应不同的深度也就是不同的长度。

建立过程如下：

1、取出频率最小的两个数，将这两个数按照最优二叉树的规则（左孩子<父节点<右孩子）得到父节点的值，将这个值放到上述频率中（相当于合并了两个最小的两个频率）

2、按照上述规则进行反复合并与放回。

3、得到哈夫曼树。

通过上述分析得到每次需要得到两个最小的频率。怎样才能实现每次得到最小的两个频率呢？可以通过数组，但是每次都要排序，很麻烦也很费时间；可以使用第i个顺序统计量的随机算法，但是其复杂度也是很高的。那我们可以选用什么结构呢？这时候就到了之前讲过的**优先队列**

使用优先队列的过程：

1、首先需要统计字符串中每个字母的出现频率，由于题目中已经简化问题，将字符限制在小写字符a到小写字符z，

2、初始化a到z的出现频率为0，所以在统计完频率频率之后需要借助频率不为0这一重要条件将出现的字符push进入队列。

3、每次取出最小的两个相加之后再放入优先队列。退出循环的条件为优先队列为空。

下面给出本题代码：

```
#include <bits/stdc++.h>
#define max_size 10010
using namespace std;
typedef long long LL;
char c[max_size];
long long f[max_size];
priority_queue<LL ,vector<LL>,greater<LL> >q;//建立小顶堆；
long long n,ans;
int main()
{
    int n;
    scanf("%d",&n);
    while(n--)
    {
        string s;
        cin>>s;
        getchar();
        memset(f,0,sizeof(f));
        int lens=s.size();
        for(int i=1; i<=lens; i++)
        {
            c[i]=s[i-1];
            f[c[i]]++;
        }
while(!q.empty())
            q.pop();
        for(int i=65; i<=122; i++)
        {
            if(f[char(i)]>0)
            {
                sum++;
                q.push(f[char(i)]);
            }
        }
        ans=0;
        while(q.size()>1)
        {
            LL a=q.top();
            q.pop();
            LL b=q.top();
            q.pop();
            ans+=(a+b); // 因为编码长度和其在树中的层数相关
            q.push(a+b);
        }
        printf("%lld\n",ans);
    }
    return 0;
```

```
}
```

可以发现在前面的统计频率不仅仅是为了统计频率，同时也是实现了将其push进优先队列。

下面给出按照哈夫曼树解决问题的代码，可以比较两种方法的优缺点：

```
View Code
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5002055>

机智零崎不会没梗Ⅰ (贪心算法)

2016年5月7日 22:38

题目描述

之前的上机中，背包问题已经基本都和大家混了个脸熟，不过还有一种不是背包却以背包为名的问题，零崎只能说“我从未见过如此厚颜无耻之包”。梗玩过了，就进入正题。

M87星云盛产矿物，有红色的绿色的黄色的蓝色的银色的白色的……不同颜色的矿物产量不同用途不同自然价值也不一样。隔壁M78星云的人虽然说主要是用银色的做头盔，不过其他颜色的还可以拿来卖给地球人啊23333

某外星生物一次可以携带重量为G的矿物，现在他们面前有K块重量W，价值V的矿石，不过他们很厉害所以可以把矿石搓成矿砂再把矿砂搓成矿石。

外星生物穿越星际路费可是很贵的，如果他们带来的矿石价值不足N，则请你补足路费，否则你只要大喊一声summon，他们就会出现了！

输入

多组输入数据

第一行为三个整数，分别为描述中的G K N

接下来K行，每行两个整数为W,V。

K小于10000，其他数据保证在int范围内，且W不为0

输出

对于每组数据，输出一行，为“summon！”或者需要补充的路费，结果保留3位小数。

输入样例

```
3 2 1
1 2
2 3
1 1 10
1 5
```

输出样例

```
summon!
5.000
```

题目来源：

http://biancheng.love/contest/23/problem/B/index

解题思路：贪心算法。

根据题目可知，既然外星人可以把矿石搓成矿砂再把矿砂搓成矿石，那么我们所要做的就是：求出外星人在最大载重为G时，携带矿石价值的最大值。如果该最大值比N小，需要输入路费也就是差了多少钱，如果大于N，输出summon！ 那么本体的关键在于求出最大价值。

下面分析最大值如何求解：

1、将矿石的性价比进行排序(所谓性价比也就是单位质量矿石的价格)，外星人每次将矿砂搓成矿石的顺序按照矿石的性价比依次进行。

2、将矿砂逐个搓成矿石，放入外星人的口袋中，同时这种矿石的数量减

一。（意思就是外星人只能还原矿石不能把原来性价比为1的矿石搓成性价比为100的矿石）。

3、在外星人将矿石放到自己的口袋的时候，需要帮助他判断是否能够装得下。同时我们还要记下一笔帐：这个外星人现在装的矿石价值总共是多少。

4、输出的判断。判断最大值和N之间的关系。输出格式printf("%.3lf \n",N-ans); 或者输出printf("summon!\n");

通过上述的讲解，我们在对矿石的性价比排序的时候可以采用结构体的形式。包括矿石的价值，数量，性价比。

给出本题的代码：

```
#include <bits/stdc++.h>
using namespace std;
struct node{
    double w;
    double v;
    double temp;
};

bool cmp(node a,node b)
{
    return a.temp>b.temp;
};

int main()
{
    int K;
    double G,N,ans;
    while(~scanf("%lf%d%lf",&G,&K,&N))
    {
        ans=0;
        node no[K];
        for(int i=0;i<K;i++)
        {
            scanf("%lf%lf",&no[i].w,&no[i].v);
            no[i].temp=no[i].v/no[i].w;
        }
        sort(no,no+K,cmp);
        for(int i=0;i<K;i++)
        {
            if(G>=no[i].w)
            {
                ans+=no[i].v;
                G-=no[i].w;
            }
            else
            {
                ans=ans+G*(no[i].temp);
                G=0;
            }
        }
        if(ans>=N)
            printf("summon!\n");
        else
            printf("%.3lf\n",N-ans);
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5002039>

大家一起数钢镚

2016年5月7日　　22:39

题目描述

Nova君在经历双十一风暴后，不得不靠在一家便利店打工来维持生计。作为一名合格的收银员，必须快速的计算价格并找钱个顾客。Nova君是个十足的硬币控，喜欢金属色闪闪的硬币，所以找钱是都希望用尽可能少的硬币。现在，假设收银台有面值为1元、5元、10元、50元、100元、500元的硬币各Ai、Bi、Ci、Di、Ei、Fi 个，需要找的钱的数额为A元，那最少需要多少个硬币呢？假定至少存在一种找钱方案。

被jhljx附体而不会数数Nova君求助中.....

输入

多组测试数据（组数不超过10），对于每组数据，输入两行，第一行为6个正整数，分别代表1元、5元、10元、50元、100元、500元的硬币个数，第二行为一个正整数A，代表需要支付的钱数。所有正整数都在INT范围内。

输出

对于每组数据，输出一行，为最少的硬币数量

输入样例

```
3 2 1 3 0 2
620
```

输出样例

```
6
```

Hint

放轻松，签到题~

题目来源:http://biancheng.love/contest/23/problem/A/index

解题方法：贪心算法

根据题目需要得到最少的硬币个数（不要问我问什么会有那么大面值的硬币）。考虑一种很实际的问题，比如说你去超市购物，营业员需要找零50元，那么你肯定不希望营业员找给你的钱都是1毛1毛钱。也不想都是1块1块。一般的营业员可能找零50面额的，也可能是10张5元的，或者25张2元的（好久没见过两元纸币了）等等方法。可以看出来找零最少的肯定是直接给一张50元。这就是我们很实际的贪心问题。因此营业员在找零的时候先去拿面值最大的并且小于找零。当满足条件之后，将找零数目减去适合的最大面值，再次进行上述操作，直到找零结束。这样的方法是很实际的贪心问题的应用。

贪心算法的博客推荐：
http://www.cnblogs.com/chinazhangjie/archive/2010/11/23/1885330.html

下面给出本题的代码：

```
1 #include<iostream>
2
3 using namespace std;
4 int main()
```

```
{
    int a[7];
    int b[7]={0,1,5,10,50,100,500};
    int num;
    while(cin>>a[1]){
    int ans=0;
    for(int i=2;i<=6;i++)
        cin>>a[i];
    cin>>num;
    for(int i=6;i>=1;i--)
    {
        while(num-b[i]>=0&&a[i]>0)
        {
            a[i]--;
            ans++;
            num-=b[i];
        }
    }
    cout<<ans<<endl;
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5002017>

DP大作战—状态压缩dp

2016年5月7日　　22:39

题目描述

阿姆斯特朗回旋加速式阿姆斯特朗炮是一种非常厉害的武器，这种武器可以毁灭自身同行同列两个单位范围内的所有其他单位（其实就是十字型），听起来比红警里面的法国巨炮可是厉害多了。现在，零崎要在地图上布置一片阿姆斯特朗回旋加速式阿姆斯特朗炮，那么在N行M列单位长度大小的地图上，求解阿姆斯特朗回旋加速式阿姆斯特朗炮最大的部署数量和对应部署方案总数。

输入

每组输入一行，为两个整数N,M （N <= 100；M <= 10）

输出

每组一行两个整数，

第一个为阿姆斯特朗回旋加速式阿姆斯特朗炮的个数，第二个为此数量下的摆放方式总数。

输入样例

```
3 3
```

输出样例

```
3 6
```

Hint

```
100    1为阿姆斯特朗回旋加速式阿姆斯特朗炮的位置，方案不唯一。
010
001
```

状态压缩是一种特殊的技巧，不止可以用在dp中。状态压缩其实是在利用数据结构，由于基础数据类型int有32位，一个int变量就可以表示2^32个状态。状态压缩其实是一种优化方法，有很大局限性。第一，单元状态通常只有两种(0、1，其实多了压缩的道理是一样的，但是没有位运算就没什么优势了)，第二，单元维度通常不超过50 (int才32，多了longlong都爆掉难道用高精度大整数压缩不成……)

状压dp本质上还是dp，问题大多数还是求解最大值或者解决方案总数之类的，但是状态数量巨大普通方法难以表示，所以利用整数可以将状态维度压缩到1维。

题目来源：http://biancheng.love/contest/10/problem/G/index

解题思路：

状态压缩dp问题。

关于状态压缩问题参见：http://www.cnblogs.com/avril/p/3282295.html

其他状态压缩DP问题：【POJ3254】【POJ1185】【POJ3311】【HDU3001】【POJ2288】【ZOJ4257】【POJ2411】【HDU3681】

给出本题代码：

```
#include <bits/stdc++.h>
#define INF 99999999
typedef long long LL;
using namespace std;

const int MAX=100+10;
int n,m,lastsize,lastlastsize,nowsize;
int last[MAX],lastlast[MAX],now[MAX];
int num[MAX],dp[MAX][MAX],temp[MAX][MAX];//dp[k][i][j]表示第k行选择i方案,第k-1行选择j方案的最大炮兵数
```

```
10
11 void dfs(int id,int k,int p,int sum)
12 {
13     if(k>=m)
14     {
15         now[++nowsize]=p;
16         num[nowsize]=sum;
17         return;
18     }
19     dfs(id,k+3,p|(1<<k),sum+1);
20     dfs(id,k+1,p,sum);
21 }
22
23 void DP()
24 {
25     for(int k=1; k<=n; ++k)
26     {
27         memset(now,0,sizeof now);
28         nowsize=0;
29         dfs(k,0,0,0);
30         for(int i=1; i<=nowsize; ++i)for(int j=1; j<=lastsize; ++j)dp[i][j]=0;
31         for(int i=1; i<=nowsize; ++i) //本行选择第几个方案
32         {
33             for(int j=1; j<=lastsize; ++j) //上一行选择第几个方案
34             {
35                 for(int t=1; t<=lastlastsize; ++t) //上上行选择第几个方案
36                 {
37                     if(now[i] & last[j])continue;//与上一行j方案不能共存
38                     if(now[i] & lastlast[t])continue;//与上上行t方案不能共存
39                     if(dp[i][j]<temp[j][t]+num[i])dp[i][j]=temp[j][t]+num[i];
40                 }
41             }
42         }
43         for(int i=1; i<=nowsize; ++i)for(int j=1; j<=lastsize; ++j)temp[i]
[j]=dp[i][j];
44         for(int i=1; i<=lastsize; ++i)lastlast[i]=last[i];
45         lastlastsize=lastsize;
46         for(int i=1; i<=nowsize; ++i)last[i]=now[i];
47         lastsize=nowsize;
48     }
49 }
50
51 int main()
52 {
53     while(~scanf("%d%d",&n,&m))
54     {
55         last[1]=lastlast[1]=temp[1][1]=0;
56         lastsize=lastlastsize=1;
57         DP();
58         int sum=0,cot;
59         for(int i=1; i<=lastsize; ++i)
60         {
61             for(int j=1; j<=lastlastsize; ++j)
62             {
63                 if(temp[i][j]>sum)
64                 {
65                     sum=temp[i][j];
66                     cot=1;
67                 }
68                 else if(temp[i][j]==sum)
69                     cot++;
70             }
71         }
72         printf("%d %d\n",sum,cot);
73     }
74     return 0;
75 }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=4991970>

DP大作战—组合背包

2016年5月7日　　22:39

题目描述

组合背包：有的物品只可以取一次（01背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。

```
DD大牛的伪代码
for i = 1 to N
    if 第i件物品属于01背包
        ZeroOnePack(F,Ci,Wi)
    else if 第i件物品属于完全背包
        CompletePack(F,Ci,Wi)
    else if 第i件物品属于多重背包
        MultiplePack(F,Ci,Wi,Ni)
```

输入

第一个数为数据组数n 1<=n<=10

接下来n组测试数据，每组测试数据由2部分组成。

第一行为背包容量V，物品种类数N。1<=V<=30000,1<=N<=200

接下来N行每行三个数为物品价值v，物品重量w，物品件数M。M=233表示物品无限。

1<=v,w<=200, 1<=M<=25

输出

对于每组数据，输出一行，背包能容纳的最大物品价值

输入样例

```
1
10 3
2 2 233
3 2 1
4 3 3
```

输出样例

```
13
```

题目来源：http://biancheng.love/contest/10/problem/F/index

解题思路：

组合背包：0-1背包、完全背包、多重背包。

因此需要结合上述三种背包问题的解决方法来solve组合背包

0-1背包代码：

```
1 void Zeronepack(int w,int v)
2 {
3     for(int i=V; i>=w; i--)
4         if(dp[i]<dp[i-w]+v)
5             dp[i]=dp[i-w]+v;
6 }
```

完全背包代码：

```
1 void Compack(int w,int v)
2 {
3     for(int i=w; i<=V; i++)
4         if(dp[i]<dp[i-w]+v)
5             dp[i]=dp[i-w]+v;
6 }
```

多重背包代码：

```
void Multipack(int w,int v,int num)
{
    int k;
    if(w*num>=V)
    {
        Compack(w,v);
        return;
    }
    for(k=1; k<num; k<<1)
    {
        Zeronepack(k*w,k*v);
        num-=k;
    }
    Zeronepack(num*w,num*v);
}
```

本题组合背包代码：

```
#include <bits/stdc++.h>
int dp[30005];
int V,N;

void Compack(int w,int v)
{
    for(int i=w; i<=V; i++)
        if(dp[i]<dp[i-w]+v)
            dp[i]=dp[i-w]+v;
}

void Zeronepack(int w,int v)
{
    for(int i=V; i>=w; i--)
        if(dp[i]<dp[i-w]+v)
            dp[i]=dp[i-w]+v;
}

void Multipack(int w,int v,int num)
{
    int k;
    if(w*num>=V)
    {
        Compack(w,v);
        return;
    }
    for(k=1; k<num; k<<1)
    {
        Zeronepack(k*w,k*v);
        num-=k;
    }
    Zeronepack(num*w,num*v);
}

int main()
{
    int kase,v,w,m;
    scanf("%d",&kase);
    while(kase--)
    {
        memset(dp,0,sizeof(dp));
        scanf("%d%d",&V,&N);
        for(int i=1;i<=N;i++)
        {
            scanf("%d%d%d",&v,&w,&m);
            if(m==1)
                Zeronepack(w,v);
            else if(m==233)
                Compack(w,v);
            else
                Multipack(w,v,m);
        }
```

```
53          printf("%d\n",dp[V]);
54      }
55      return 0;
56  }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=4991934>