Profile Picture

y4612s's Blog

- Blog
- About
- Archives
- About Me

Google+ Twitter GitHub RSS
April 16, 2014
树套树 分块

[Solution]BZOJ3295: [Cqoi2011]动态逆序对

题目：

戳这里

题解：
线段树套平衡树 or 分块法
看到这题的第一反应应该是线段树套平衡树。然后看了下题解发现大家这么水掉了，就开心的编起来了……
打了一个小时代码最后发现超时啊！常数太大啊！模板太渣啊！
为了缩小常数写的zkw线段树还是超时了……坑爹……
不过，这道题是可以用分块过掉的。
树套树思路：
用线段树套平衡树维护一段区间之内比某数小/大的数字的个数。删除一个数就查找一下在这个数前面且比这个数大的数字有多少、在这个数后面且比这个数小的数字有多少。答案减掉这个，然后在logN棵平衡树中删除。思路很直观，但是时间和空间消耗都比较大(神犇们纷纷用这方法水掉了，本人实在写的太渣而TLE了……)
时间复杂度O(Mlog^2N)
分块法思路：
可以把原来的数列分为sqrt(N)块，对每一个块按照数字大小排好序。也就是说，块与块之间元素大小无序而元素位置有序，块内元素大小有序而元素位置无序。
首先需要用归并排序求出最开始的逆序对数目。
询问一个元素的时候，对于这个元素之前/之后的块，因为块内元素大小有序，可以二分法找出有多少大于/小于这个数的数。
而对于元素所在块本身，暴力查找即可(块的大小只有sqrt(N)，最大约为300)
删除一个元素，直接在所在块暴力删除(块的大小仍然只有300左右……)
然后就可以过掉了……
时间复杂度O(M*sqrt(N)*log(sqrt(N)))
吐槽一：听说考试的时候时限是三秒……fuck!为毛时间给的这么短啊！

吐槽二：继deque之后另一个效率奇低的stl被发现了：lower_bound!前后时间整整差了一倍多有木有啊！必须手写啊！话说为什么我之前用的那么开心……大视野你卡常数有意思吗……

贴代码：
第一个：树套树代码，Time Limit Exceed，随机数据关02约耗时1.6s

bzoj3295_TLE.cpp

```
//这不是本篇的重点，故不写注释了……反正思路很裸
#include<cstdio>
#include<cstdlib>
int n,m,num[100050],pos[100050];
long long ans;
struct Node
{
    int val,pri,siz;
    Node *ch[2];
    Node(){}
    Node(int v);
    void maintain(){siz=1+ch[0]->siz+ch[1]->siz;}
    void* operator new(size_t);
}mem[2000005],Tnull,*null=&Tnull;
Node::Node(int v)
{
    val=v,pri=rand(),siz=1;
    ch[0]=ch[1]=null;
}
void* Node::operator new(size_t)
{
    static Node *P=mem;
    return P++;
}
struct Treap
{
    Node *root;
    Treap():root(null){}
    void rotate(Node *&p,int d)
    {
        Node *X=p->ch[!d];
        p->ch[!d]=X->ch[d];
        X->ch[d]=p;
        p->maintain();
        X->maintain();
        p=X;
    }
    void insert(int x,Node *&p)
    {
        if(p==null){p=new Node(x);return;}
        if(x < p->val)
        {
            insert(x,p->ch[0]);
            if(p->ch[0]->pri > p->pri)
                rotate(p,1);
        }
        else
        {
            insert(x,p->ch[1]);
            if(p->ch[1]->pri > p->pri)
                rotate(p,0);
        }
        p->maintain();
    }
    void erase(int x,Node *&p)
    {
```

```
            if(p->val==x)
            {
                if(p->ch[0]==null || p->ch[1]==null)
                    p=(p->ch[0]==null)?p->ch[1]:p->ch[0];
                else
                {
                    if(p->ch[0]->pri > p->ch[1]->pri)
                        {rotate(p,1);erase(x,p->ch[1]);}
                    else {rotate(p,0);erase(x,p->ch[0]);}
                }
            }
            else if(x < p->val)erase(x,p->ch[0]);
            else erase(x,p->ch[1]);
            if(p!=null)p->maintain();
        }
        int query1(int x)
        {
            int ret=0;Node *p=root;
            while(p!=null)
            {
                if(x >= p->val)p=p->ch[1];
                else ret+=1+p->ch[1]->siz,p=p->ch[0];
            }
            return ret;
        }
        int query2(int x)
        {
            int ret=0;Node *p=root;
            while(p!=null)
            {
                if(x <= p->val)p=p->ch[0];
                else ret+=1+p->ch[0]->siz,p=p->ch[1];
            }
            return ret;
        }
    };
    struct Segment_Tree
    {
        Treap dat[262144];int M;
        void build()
        {
            for(M=1;M<n+2;M<<=1);
            for(int i=1;i<=n;++i)
                for(int j=i+M;j;j>>=1)
                    dat[j].insert(num[i],dat[j].root);
        }
        int query1(int l,int r,int x)
        {
            if(l>r)return 0;int ret=0;
            for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
            {
                if(~l&1)ret+=dat[l^1].query1(x);
                if(r&1)ret+=dat[r^1].query1(x);
            }
            return ret;
        }
        int query2(int l,int r,int x)
        {
            if(l>r)return 0;int ret=0;
            for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
            {
                if(~l&1)ret+=dat[l^1].query2(x);
                if(r&1)ret+=dat[r^1].query2(x);
            }
            return ret;
```

```
    }
    void erase(int x)
    {
        for(int i=pos[x]+M;i;i>>=1)
            dat[i].erase(x,dat[i].root);
    }
}st;
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;++i)
    {
        scanf("%d",num+i);
        pos[num[i]]=i;
    }
    st.build();
    for(int i=1;i<=n;++i)
        ans+=st.query2(i+1,n,num[i]);
    int x;
    while(m--)
    {
        printf("%lld\n",ans);
        scanf("%d",&x);
        ans-=st.query1(1,pos[x]-1,x);
        ans-=st.query2(pos[x]+1,n,x);
        st.erase(x);
    }
}
```

第二个：分块法代码，Accepted，随机数据关02约耗时1.1s

bzoj3295_AC.cpp

```
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
int num[100050],pos[100050],n,m,bel[100050],cnt,cntb=1,begin[100050];
//num和begin为原数组(begin归排后顺序破坏)，pos[x]为数字x的位置，bel[x]为x位置上的数字属于哪个块
long long ans;
int lower_bound(int *num,int siz,int x)//手写二分，stl效率太低
{
    int l=1,r=siz,m,ans=0;
    while(l<=r)
    {
        m=(l+r)>>1;
        if(num[m]<=x)
            ans=m,l=m+1;
        else r=m-1;
    }
    return ans;
}
struct Block//块
{
    int o[320],siz;
    int lower(int x)//查找比x小的数字有多少个
     {return lower_bound(o,siz,x);}
    int upper(int x)//查找比x大的数字有多少个
     {return siz-lower(x);}
    void erase(int x)//暴力删除
 {
        int t=lower_bound(o,siz,x);
        for(;t<siz;++t)
            o[t]=o[t+1];
```

```
            --siz;
        }
    }block[320];
    int calc(int x)//计算数字x参与了多少逆序对
    {
        int ans=0,k=bel[pos[x]];
        for(int i=1;i<k;++i)ans+=block[i].upper(x);//之前的块
     for(int i=k+1;i<=cntb;++i)ans+=block[i].lower(x);//之后的块
     for(int i=1;i<=block[k].siz;++i)//当前块
     {
            if(block[k].o[i]<x && pos[block[k].o[i]]>pos[x])++ans;
            if(block[k].o[i]>x && pos[block[k].o[i]]<pos[x])++ans;
        }
        return ans;
    }
    void merge(int *data,int l,int m,int r)//归并排序求初始逆序对数
    {
        static int res[100002];
        int b1=l,b2=m+1,p=l;
        while(b1<=m&&b2<=r)
            if(data[b1]>data[b2])res[p++]=data[b2++],ans+=m-b1+1;
            else res[p++]=data[b1++];
        while(b1<=m)res[p++]=data[b1++];
        while(b2<=r)res[p++]=data[b2++];
        memcpy(data+l,res+l,sizeof(int)*(r-l+1));
    }
    void merge_sort(int *data,int l,int r)//归并排序
    {
        if(l>=r)return;
        int mid=(l+r)>>1;
        merge_sort(data,l,mid);
        merge_sort(data,mid+1,r);
        merge(data,l,mid,r);
    }
    int main()
    {
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;++i)
        {
            scanf("%d",num+i);
            pos[num[i]]=i;
            begin[i]=num[i];
        }
        int cnt=sqrt(n)+0.5,d;//cnt为块的大小
     for(int i=1;i<=n;++i)
        {
            block[cntb].o[++block[cntb].siz]=num[i];
            bel[i]=cntb;
            if(block[cntb].siz>=cnt)++cntb;
        }
        for(int i=1;i<=cntb;++i)std::sort(block[i].o+1,block[i].o+1+block[i].siz);
        merge_sort(begin,1,n);
        while(m--)
        {
            printf("%lld\n",ans);
            scanf("%d",&d);
            ans-=calc(d);
            block[bel[pos[d]]].erase(d);
        }
    }
```

April 15, 2014

[动态规划](#) [单调队列](#)

[Solution]BZOJ1855: [Scoi2010]股票交易

题目：

戳这里

题解：
单调队列DP
补上数据范围+吐槽BZOJ的坑爹bugs
对于30%的数据，0<=W<T<=50,1<=MaxP<=50
对于50%的数据，0<=W<T<=2000,1<=MaxP<=50
对于100%的数据，0<=W<T<=2000,1<=MaxP<=2000
对于所有的数据，1<=BPi<=APi<=1000,1<=ASi,BSi<=MaxP
看到题第一眼想到了三维的状态……不过显然不行，还要优化
可以定义一个二维的状态f(i,j)代表到第i天结束，手中持有的股票数为j的最大收益
所以可以得出以下的状态转移方程:
$f(i,j)=f(i-1,j)$(没有行动)
$f(i,j)=f(i-w-1,k)-AP_i*(j-k)(j-AS_i\geq k\geq j)$(买入股票)
$f(i,j)=f(i-w-1,k)+BP_i*(k-j)(j\geq k\geq j+BS_i)$(卖出股票)
这是一个2D1D动态规划，算法的瓶颈在于枚举k。所以可以使用单调队列优化转移的时间
将下面的两个方程化开，得到
$f(i,j)=f(i-w-1,k)-AP_i*(j-k)=f(i-w-1,k)+AP_i*k-AP_i*j$
$f(i,j)=f(i-w-1,k)+BP_i*(k-j)=f(i-w-1,k)+BP_i*k-BP_i*j$
我们的状态都是一行一行转移的，而f(i-w-1,k)+APi*k和f(i-w-1,k)+BPi*k只和k相关。所以可以用单调队列维护这两个东西，转移的时候就不需要枚举k了。
时间复杂度O(T*maxP)，可以过掉本题
十个注意事项：
1.千万不要用STL的deque！慢成翔啊！个人认为queue还有存在的价值，而deque这慢死人的常数实在让人接受不能……TM劳资的瑰丽华尔兹用了deque不开O2只有六十分啊！这题TM暴力都有70分啊！悲伤的故事
2.除了这个应该没有其他的注意事项了……
3~10.没有注意事项
贴代码：

bzoj1855.cpp

```
#include<cstdio>
#include<cstring>
inline void go(int &a,int b){if(a<b)a=b;}
int T,maxP,W,AP[2005],BP[2005],AS[2005],BS[2005],f[2005][2005];
struct ele//单调队列元素
{
    int val,k;
    ele(){}
    ele(int v,int kk):val(v),k(kk){}
};
struct Deque//手写双端队列
{
    ele dat[2005];int s,t;
    ele front(){return dat[s];}
    ele back(){return dat[t-1];}
    bool empty(){return s==t;}
    void push(ele x){dat[t++]=x;}
    void pop_front(){++s;}
    void pop_back(){--t;}
```

```
    void clear(){s=t=0;}
}Q;
int DP()
{
    memset(f,192,sizeof(f));
    for(int i=1;i<=T;++i)
    {
        for(int j=0;j<=maxP;++j)//不买
         f[i][j]=f[i-1][j];
        for(int j=0;j<=AS[i];++j)//如果当天是第一次买股票，则不需考虑上次购买时限的限制
         go(f[i][j],-AP[i]*j);
        if(i-W-1<0)continue;
        Q.clear();
        for(int j=0;j<=maxP;++j)//买进股票
     {
            while(!Q.empty() && Q.front().k<j-AS[i])Q.pop_front();
            while(!Q.empty() && Q.back().val<=f[i-W-1][j]+AP[i]*j)Q.pop_back();
            Q.push(ele(f[i-W-1][j]+AP[i]*j,j));
            go(f[i][j],Q.front().val-AP[i]*j);
        }
        Q.clear();
        for(int j=maxP;j>=0;--j)//卖出股票
     {
            while(!Q.empty() && Q.front().k>j+BS[i])Q.pop_front();
            while(!Q.empty() && Q.back().val<=f[i-W-1][j]+BP[i]*j)Q.pop_back();
            Q.push(ele(f[i-W-1][j]+BP[i]*j,j));
            go(f[i][j],Q.front().val-BP[i]*j);
        }
    }
    return f[T][0];
}
int main()
{
    scanf("%d%d%d",&T,&maxP,&W);
    for(int i=1;i<=T;++i)
        scanf("%d%d%d%d",AP+i,BP+i,AS+i,BS+i);
    printf("%d\n",DP());
}
```

April 14, 2014
最小生成树

[Solution]BZOJ2753: [SCOI2012]滑雪与时间胶囊

题目：

戳这里

题解：
Kruskal算法变形
这题狗眼一看应该是最小树形图裸题……第一问求树形图有多大，第二问求最小树形图边权和
不过看一眼数据范围立刻就跪了……朱刘算法O(VE)有木有……
想了半天没有思路，最后还是去看了题解……省选肿么办啊……
首先，第一问是个SB题，略过不计……
这个题中只能从高的地方滑到低的地方，所以是一个有向图。高度相同的地方连的是一个无向边
膜拜题解之后得知做法应该是把边排序，其中第一关键字是有向边终点的高度，从高到低排序，第二关键字是权值，从小到大排序
为什么是这样呢？补上本人的一个简单证明

最小树形图为什么不能用Prim或Kruskal解呢？都是因为有向所以不能保证形成生成树或者生成树最小
Kruskal算法不能形成最小树形图应该很明显，随便画一个图就能发现，有可能排完序并查之后出来的根本不是树形图
Prim算法可以生成树形图，但不能保证最小。比如下面的这个图：
1->2:7
2->3:4
1->3:8
3->2:1
因为坑爹的循环顺序，Prim的结果是11。可明显答案是9。因为循环顺序的原因导致可能更优的决策没有执行
但是这个题的层次性非常明显，从低的地方不可能通过边回到高的地方。也就是说，假设我们考虑完了所有比点x高的点，现在正在考虑所有与x高度相同的点的入边，那么已经比x高的点的状况不会改变了，因为现在考虑的所有边都不能回到比x高的点
所以我们可以按照终点高度排序。考虑到一个点的入边时，不会发生Prim算法下面的点连回上面使最小树形图变小的状况。对于高度相同的点来说应该是一个强连通分量，但是这个无所谓，因为按照终点高度排序，高度相同还有第二关键字权值，仍然可以保证树形图是最小的
如果还是不能理解，可以这样想：之前考虑的那些高的点的状态已经不会改变了，我们当前的事务是给当前高度的所有点选好入边，把它们加入树形图。至于那些高的点就“假设”他们已经在树形图里吧……所以这些边按照权值排序之后K算法一下就好了
所以可以放心地使用Kruskal求出解，复杂度O(MlogM)。第一问复杂度O(N)
补了超电磁炮看着Tree-Diagram还是挺带感的……
贴代码：

bzoj2753.cpp

```
#include<cstdio>
#include<vector>
#include<algorithm>
using namespace std;
vector<int> G[100005];//邻接表存图，因为偷懒用了vector
int n,m,hei[100005],par[100005],dcnt;//点的高度，并查集的父亲，树形图点数
bool used[100005];
int find(int p){return par[p]==p?p:par[p]=find(par[p]);}
struct Edge
{
    int s,t,h,w;//起点，终点，终点高度，权值
 Edge(){}
    Edge(int a,int b,int c,int d):s(a),t(b),h(c),w(d){}
    bool operator < (const Edge &s)const
    {
        if(h!=s.h)return h>s.h;
        return w<s.w;
    }
}e[2000005];int ecnt;
void dfs(int v)//第一问
{
    used[v]=true;++dcnt;
    int siz=G[v].size();
    for(int i=0;i<siz;++i)
        if(!used[G[v][i]])
            dfs(G[v][i]);
}
long long Tree_Diagram()//第二问
{
    long long ans=0;
    sort(e,e+ecnt);
    for(int i=1;i<=n;++i)//初始化并查集父亲
     par[i]=i;
    for(int i=0;i<ecnt;++i)
```

```
        {
            if(!used[e[i].s] || !used[e[i].t])continue;//不在树形图中的边略过不计
         int u=find(e[i].s),v=find(e[i].t);//Kruskal
         if(u!=v)par[v]=u,ans+=e[i].w;
        }
        return ans;
    }
    int main()
    {
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;++i)
            scanf("%d",hei+i);
        int a,b,c;
        for(int i=1;i<=m;++i)
        {
            scanf("%d%d%d",&a,&b,&c);
            if(hei[a]<=hei[b])
                e[ecnt++]=Edge(b,a,hei[a],c),G[b].push_back(a);
            if(hei[a]>=hei[b])
                e[ecnt++]=Edge(a,b,hei[b],c),G[a].push_back(b);
        }
        dfs(1);printf("%d ",dcnt);
        printf("%lld\n",Tree_Diagram());
    }
```

April 10, 2014
费用流 网络流 二分图

[Solution]BZOJ2261: [BeiJing wc2012]连连看

题目：

戳这里

题解：
二分图最佳匹配 or 最小费用最大流
1000的数据范围还是比较可爱的。可以先n^2暴力一下哪些点可以连（本人竟然在数论的方向想了半天……沙茶……分明暴力就好）
然后我们可以把每一个数字拆成两个点，如果a和b可以连，从a到b连一条容量1费用(a+b)的边，从b到a也连一条一样的
然后就是裸的费用流问题了……（其实只是二分图KM算法不会写有木有！沙茶啊！）
二分图匹配问题，KM应该会快一些，但是本人实在是太沙茶了，只会SPFA费用流……
第一问输出最大流/2，第二问输出最小费用/2即可
时间复杂度：
SPFA增广费用流复杂度为n*n*m，本题n=1000，好像不能过……
其实这都是唬人的
先打一个暴力看看有多少对点符合条件，发现只有316对数字符合条件而已……
在输出一下这316对数关联了多少点，发现只有367个点是实际需要的……
开一个数组映射一下再建边就可以秒杀了……
本人用了100ms，用KM的话应该会更快……不过内存还是很Q的
贴代码：

bzoj2661.cpp

```
#include<cstdio>
#include<cmath>
#include<cstring>
#include<queue>
```

```
const int inf=0x3f3f3f3f;
int gcd(int a,int b){return b?gcd(b,a%b):a;}
bool used[1005],isinQ[740];
int a,b,map[1005],ds,dt,dis[740],way[740],maxflow,mincost;
struct Edge;
struct Vertex//点
{
    Edge *next;
}V[740];
struct Edge//边
{
    Vertex *to;
    Edge *next;
    int flow,cost;
    Edge(){}
    Edge(Vertex *t,Edge *n,int f,int c):to(t),next(n),flow(f),cost(c){}
    void* operator new(size_t);
}E[3000];
void* Edge::operator new(size_t)
{
    static Edge *P=E;
    return P++;
}
inline void addEdge(Vertex *A,Vertex *B,int f,int c)
{
    A->next=new Edge(B,A->next,f,-c);//费用取相反数，最后再取回来，求最大费用
 B->next=new Edge(A,B->next,0,c);
}
bool spfa()//费用流SPFA部分
{
    std::queue<Vertex*> Q;
    memset(dis,0x3f,sizeof(dis));
    dis[ds]=0,Q.push(V+ds);
    while(!Q.empty())
    {
        Vertex *v=Q.front();Q.pop();
        isinQ[v-V]=false;
        for(Edge *e=v->next;e;e=e->next)
        {
            if(!e->flow || dis[e->to-V]<=dis[v-V]+e->cost)continue;
            dis[e->to-V]=dis[v-V]+e->cost;
            way[e->to-V]=e-E;
            if(!isinQ[e->to-V])
                isinQ[e->to-V]=true,Q.push(e->to);
        }
    }
    return dis[dt]<inf;
}
void MCMF()//最小费用最大流
{
    while(spfa())
    {
        int cur=dt,mini=inf;
        for(;cur!=ds;cur=E[way[cur]^1].to-V)
        {
            Edge *e=E+way[cur];
            if(mini > e->flow)
                mini=e->flow;
        }
        for(cur=dt,maxflow+=mini;cur!=ds;cur=E[way[cur]^1].to-V)
        {
            Edge *e=E+way[cur];
            mincost+=mini*e->cost;
            e->flow-=mini;
            E[way[cur]^1].flow+=mini;
```

```
            }
        }
        maxflow>>=1,mincost=(-mincost)>>1;//最大流和最小费用均取反输出
    }
    int main()
    {
        scanf("%d%d",&a,&b);
        int flag=0;ds=0,dt=739;
        for(int i=a;i<b;++i)
        {
            for(int j=i+1;j<=b;++j)
            {
                int t=sqrt(j*j-i*i);
                if(t*t==j*j-i*i && gcd(i,t)==1)//暴力判断是否符合条件
                {
                    if(!used[i])used[i]=true,map[i]=++flag;//数字i映射为map[i]
                 if(!used[j])used[j]=true,map[j]=++flag;
                    addEdge(V+map[i],V+map[j]+370,1,i+j);
                    addEdge(V+map[j],V+map[i]+370,1,i+j);
                }
            }
        }
        for(int i=1;i<=flag;++i)
        {
            addEdge(V+ds,V+i,1,0);
            addEdge(V+i+370,V+dt,1,0);
        }
        MCMF();
        printf("%d %d\n",maxflow,mincost);
    }
```

April 10, 2014
网络流 最小割

[Solution]BZOJ2561: 最小生成树

题目：

戳这里

题解：
最大流
刚看到这道题表示不明觉厉……但是联系一下Kruskal算法的流程还是可以搞一搞的
假设最后新加的那条边为(u,v,w)
Kruskal算法中我们对边按照权值排序，在计算最小生成树的时候，我们优先考虑了权值小于w的边，也就是说，如果排在前面的那些边成功将u,v连接在了一起，那么这条边就再也没有机会了
所以，我们需要删去一些边，使删除后u,v不连通。显然这是一个最小割模型，以u为源点，v为汇点，如果一条边的权值小于w则连上这条边，流量是1。那么最后求出最小割，就是构成最小生成树所需删除的边的数量。
然后清空图，以相同的方法求最大生成树需要删除的边数。两者相加即为答案。
会不会有冲突，比如说在最小生成树的图中删去边后影响最大生成树的结果呢？不会的，因为这两个图没有任何重复的边，在一个图中的删除操作不会影响另一个图，我们在求最小/大生成树的时候不是也没有考虑大于/小于w的边的影响么。在构建最小生成树的时候，为了加入(u,v,w)而删去了一条权值比w小的边，在构建最大生成树的时候根本就不会考虑到删的这条边是什么。所以两个图互不冲突，可以直接答案加和
贴代码：(仍然只贴主函数)

bzoj2561_main.cpp

```
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m+1;++i)
        list[i].read();//读边
 int w=list[m+1].weight,ans=0;
    for(int i=1;i<=m;++i)//第一次加边，最小生成树
     if(list[i].weight<w)
        addEdge(V+list[i].from,V+list[i].to,1);
    ds=list[m+1].from;//源点
 dt=list[m+1].to;//汇点
 while(bfs())//第一次dinic
     ans+=dinic(V+ds,inf);
    P=E;memset(V,0,sizeof(V));//清空原图
 for(int i=1;i<=m;++i)//第二次加边，最大生成树
     if(list[i].weight>w)
        addEdge(V+list[i].from,V+list[i].to,1);
    while(bfs())//第二次dinic
     ans+=dinic(V+ds,inf);
    printf("%d\n",ans);
}
```

April 10, 2014
费用流 网络流

[Solution]BZOJ1449: [JSOI2009]球队收益

题目：

戳这里

题解：
最小费用最大流
这道题求最小收益，可以想到使用费用流求解
因为平方费用的存在，直接构图不是很容易
可以这么想，我们假设后面我们需要规划的m场比赛双方都输了，我们要求一个分配方法，把m次胜利分配到每个球队上使总收益最小
设一个球队目前赢了x场，输了y场。我们现在想让这个队多赢一场，即(x,y)->(x+1,y-1)，能够获得多少收益？
数学推导一下就有了：
C(x+1)^2+D(y-1)^2-Cx^2-Dy^2
=Cx^2+2Cx+C+Dy^2-2Dy+D-Cx^2-Dy^2
=2Cx-2Dy+C+D
其中C和D是题目中给定的Ci和Di
所以我们可以得出这样的构图方法：
从源点向每场比赛连一条边，流量1费用0
从每场比赛向他关联的两支球队各连一条边，流量1费用0
从每个球队向汇点t条边（t为该球队的剩余比赛次数），流量1，按照上面的式子计算出每多赢一场能有多少收益，作为费用
这样构图相当于我们把m次胜利分配给参与比赛的队伍，使他们获得一些收益，最小费用流使收益最小化
最后的答案就是开始的总收益（赢了win[i]场，输了lose[i]+t场）加上最小费用
代码效率很低，排到倒数……只把主函数发上来吧，想看的可以看一看建图的部分
贴代码：

bzoj1449_main.cpp

```
int main()
{
    scanf("%d%d",&n,&m);
    ds=0,dt=n+m+1;//源点，汇点
 for(int i=1;i<=n;++i)
        scanf("%d%d%d%d",win+i,lose+i,c+i,d+i);
    for(int i=1;i<=m;++i)
    {
        addEdge(V+ds,V+i,1,0);//从源点向每场比赛连一条边，流量1费用0
     scanf("%d%d",a+i,b+i);
        addEdge(V+i,V+a[i]+m,1,0);//从每场比赛向他关联的两支球队各连一条边，流量1费用0
     addEdge(V+i,V+b[i]+m,1,0);
        ++rest[a[i]],++rest[b[i]];//每个球队剩余出场次数
 }
    for(int i=1;i<=n;++i)
    {
        int x=win[i],y=lose[i]+rest[i];//假设后面全输，胜利次数和失败次数
     ans+=c[i]*x*x+d[i]*y*y;
        while(y>lose[i])//计算每多赢一场可以获得多少收入
     {
            addEdge(V+i+m,V+dt,1,2*c[i]*x-2*d[i]*y+c[i]+d[i]);
            ++x,--y;
        }
    }
    printf("%d\n",ans+MCMF());//总收益+最小费用
}
```

April 10, 2014
网络流 最小割

最小割系列(bzoj2039,2127,2132,1976)

最近做到几个最小割划分集合的问题，记录一下
常见思想：通过边的权值体现分到相同集合或不同集合的收益，转化为最小割，进而通过最大流算法求解
首先膜拜strongoier大神……如果本文实在太渣就去膜拜大神百度空间吧

bzoj2039 [2009国家集训队]employ人员雇佣

题目：戳这里
如果我们想求最大的收益，可以把所有的收益相加，然后减去构成合法方案所必须的收益损失
假设有两个人i和j
如果我们同时雇用了两个人，损失收益Ai+Aj（正常雇用费用）
如果我们雇用了i而没有雇用j，损失收益为2*E[i][j]+Ai（不但损失了两人合作的收益，而且承受了j干扰i的损失）
如果两人都不雇用，损失收益为2*E[i][j]（两人不能合作）
所以得出以下的连边方法：
1.从源点连向每个点i，流量为Ai
2.从每个点i连向点j，流量为E[i][j]
3.从每个点i连向汇点，流量为sigma[i]（sigma[i]为所有E[i][j]的总和）
这样的话，求出最小割将原点集划分为两个集合S集和T集，S集相当于没有雇用这个人（没有割Ai），T集相当于雇用了这个人。两个人同在T集合（同时雇用）产生的负收益就是Ai+Aj，不雇用一个人i，产生的负收益为sigma[i]，之后如果有点j与i处于不同的集合，那么收益继续损失E[i][j]
建好图后直接跑最大流即可

bzoj2127 happiness

题目：戳这里
吐槽这个题无比神奇的input……好好排一下版会死吗……好歹加个断句符号也好啊……
仍然是求最大收益的问题，划分成两个集合，仍然是最小割求解，仍然是所有收益-合法方案的收益损失
如果一个人去了文科班，我们假设他被分到了集合S，所以从起点连向每个点i，权值为这个人去理科班的收益
如果一个人去了理科班，我们假设他被分到了集合T，所以从每个点i连向汇点，权值为这个人去文科班的收益
现在人员的去向确定了，两个小伙伴如果没有分到一个班，怎么减去这个收益损失呢？
如果两个人同时选择了文科，我们需要扣除两人同选理科的收益
如果两个人一文一理，我们需要扣除两个人同选文+同选理的收益
两个人如果被分到一个集合，扣除的收益就不能在两个人之间的边体现出来。那么我们可以对于每对点i,j，从源点向点i,j分别连两者同选文科班收益的一半的边，从i,j分别向汇点连同选理的收益的一半的边
如果两人不选同一科，扣除的收益要在两者之间的边体现一部分。在i,j之间连上两人（同选文+同选理）收益的一半
为了方便处理一半的关系而不出现小数，可以在最开始将所有收益*2，得出答案后除以2即可
仍然是跑最大流出解

bzoj2132 圈地计划

题目：戳这里
这题跟上一题的不同之处是划分到两个不同集合有收益，那么割边就相当于划分不同集合的收益，最小割就成了最小收益了……
不过这个题是一个二分图，二分图的一侧点集里一定没有边相连，可以先对原图进行染色，像棋盘一样分为白色和黑色
对于白点，从源点出发向点i连流量为Ai的边，点i向汇点连流量为Bi的边
对于黑点，从源点出发向点i连流量为Bi的边，点i向汇点连流量为Ai的边
相邻的点i,j连边，流量为Ci+Cj
这回不是S集合和T集合一定是一种颜色了，画一画就能知道，白色点和黑色点的含义是相反的，一个分到S集合代表建商业区，一个分到T集合代表建商业区。这样的话割边就相当于需要扣除的建立同种地域设施的收益了
仍然总收益-最大流

bzoj1976 [BeiJing2010组队]能量魔方 Cube

题目：戳这里
这题和圈地计划很像，只是三维看起来比较恶心。做法和上一题基本相同
和上一题基本是一样的染色，一样的建图。不过这题开始已经给定了一些P和N了怎么办？
可以按照黑白色在每个P or N和源 or 汇连边，流量为inf，代表这个边不能被割

April 8, 2014
二分图 最大流

[Solution]BZOJ3158: 千钧一发

题目：

戳这里

题解：
二分图最大点权独立集
一道比较水的题做的人不是很多啊……
首先有一个直观的想法，如果两个装置不能同时取的话，在两个点之间连上一条边，然后取出一些点使取到的点之间没有边相连，看起来比较像一个最大独立集的问题
但是，这个图是二分图吗……不是的话怎么求最大独立集呢……
可以这样考虑这个图，按照ai的奇偶把点分成两个集合，那么这个图是一个二分图
为什么可以按照奇偶性来划分呢？因为我们可以证明，任意两个奇数一定满足条件1成立，任意两个偶数一定满足条件2成立
证明1：对于两个奇数a和b，a^2+b^2不是完全平方数
证明：假设一个奇数为x=2k+1，那么x^2=4k^2+4k+1，所以x^2===1(mod 4)
假设一个偶数为x=2k，那么x^2=4k^2，所以x^2===0(mod 4)
所以任意一个自然数x，x^2 mod 4的值为0或1
而两个奇数a和b，a^2+b^2===2(mod 4)，显然a^2+b^2不是完全平方数，满足条件1
证明2：对于两个偶数a和b，gcd(a,b)>1
证明：今天你学数论了吗？一目了然法可证
所以，奇数集合中一定没有边连接其中的两个点，偶数集合中也一定没有边连接其中的两个点
所以这个图是一个二分图，可以转化为二分图最大点权独立集
二分图最大点权独立集=二分图总点权-二分图最大点权覆盖
二分图最大点权覆盖转化为最小割=最大流
这个应该不用细说了吧……基础理论，详见胡伯涛论文或度娘一下
本人图论模板一向效率奇低无比且写法奇葩，该题竟然1A+rank3，发博文庆祝一下……

贴代码：

bzoj3158.cpp

```
#include<cstdio>
#include<queue>
#include<cmath>
#include<cstring>
typedef long long ll;
const int inf=0x3f3f3f3f;
inline int min(int a,int b){return a<b?a:b;}
int n,a[1005],b[1005],n1,n2,a1[1005],a2[1005],ds,dt,level[1005];
bool issqr(ll x)//判断一个数是否为完全平方数
{
    ll a=sqrt(x);
    return a*a==x;
}
int gcd(int a,int b){return b?gcd(b,a%b):a;}
struct Edge;
struct Vertex//点
{
    Edge *next;
}V[1005];
struct Edge//边
{
    Vertex *to;
    Edge *next;
    int flow;
    Edge(){}
    Edge(Vertex *t,Edge *n,int f):to(t),next(n),flow(f){}
    void* operator new(size_t);
}E[600000];
void* Edge::operator new(size_t)
{
    static Edge *P=E;
    return P++;
```

```
}
inline void addEdge(Vertex *A,Vertex *B,int f)//加边
{
    A->next=new Edge(B,A->next,f);
    B->next=new Edge(A,B->next,0);
}
int dinic(Vertex *v,int flo)//dinic算法
{
    if(v-V==dt)return flo;
    int maxf=flo;
    for(Edge *e=v->next;e;e=e->next)
    {
        if(!e->flow || level[e->to-V]!=level[v-V]+1)continue;
        int t=dinic(e->to,min(maxf,e->flow));
        e->flow-=t;
        E[(e-E)^1].flow+=t;
        maxf-=t;
        if(maxf<=0)break;
    }
    if(flo==maxf)level[v-V]=inf;
    return flo-maxf;
}
bool bfs()//bfs构造层次图
{
    std::queue<Vertex*> Q;
    memset(level,0x3f,sizeof(level));
    level[ds]=0;Q.push(V+ds);
    while(!Q.empty())
    {
        Vertex *v=Q.front();Q.pop();
        for(Edge *e=v->next;e;e=e->next)
        {
            if(!e->flow)continue;
            if(level[e->to-V] > level[v-V]+1)
            {
                level[e->to-V]=level[v-V]+1;
                Q.push(e->to);
            }
        }
    }
    return level[dt]<inf;
}
int main()
{
    scanf("%d",&n);
    ll ans=0;
    for(int i=1;i<=n;++i)
    {
        scanf("%d",a+i);
        if(a[i]&1)a1[++n1]=i;//a1,a2分别储存两个左右两个点集的点的编号
     else a2[++n2]=i;
    }
    for(int i=1;i<=n;++i)
        scanf("%d",b+i),ans+=b[i];
    ds=0,dt=n1+n2+1;//ds为网络源点，dt为网络汇点
 for(int i=1;i<=n1;++i)
        addEdge(V+ds,V+i,b[a1[i]]);
    for(int i=1;i<=n2;++i)
        addEdge(V+n1+i,V+dt,b[a2[i]]);
    for(int i=1;i<=n1;++i)
    {
        for(int j=1;j<=n2;++j)
        {
            int ai=a[a1[i]],aj=a[a2[j]];
            if(issqr((ll)ai*ai+(ll)aj*aj) && gcd(ai,aj)==1)//判断两个点集之间是否有边
```

```
                addEdge(V+i,V+j+n1,inf);
            }
        }
        while(bfs())//跑最大流
         ans-=dinic(V+ds,inf);
        printf("%lld\n",ans);
    }
```

April 6, 2014
概率论 高斯消元

[Solution]BZOJ3143:[Hnoi2013]游走

题目：

戳这里

高斯消元法
高斯消元+概率dp第三发……仍然没能1A……
如果这题边权已知，那么人民大众都会大呼水题一道，如果边权未知，可以给他定一个边权
我们可以进行概率DP解出每一条边的期望经过次数，按照这个顺序，显然给期望经过多的赋成小的边权就可以了
边有方向，加上他可以从四面八方转移过来，不是很方便。我们可以转而计算经过每个点的期望次数
设Ai为经过i号点的期望次数
那么每一个Ai就是所有能够到达i点的j点，他们的Aj/deg[j]（deg[j]是j点的出度）求和
这里有两个特殊情况：对于终点，An=0，对于起点，A1=sig(Aj/deg[j])+1
终点的期望次数等于0应该不难理解（都结束了还走什么），起点在开始的时候必然经过一次，所以要+1
那么只要扫一扫边，高斯消元一下就知道Ai了
对于一条边(u,v)，他的期望经过次数就是Au/deg[u]+Av/deg[v]
知道了每个边的期望走过次数，贪心得出了每个边的权值，乘一下求个和就是答案了
PS：本人竟然登山包地做了第二次高斯消元算了每个点到终点的期望总分……还是错的……简直沙茶……明明for一下就可以的说
贴代码：

bzoj3143.cpp

```
#include<cstdio>
#include<algorithm>
void swap(double &a,double &b){double t=a;a=b;b=t;}
inline double fabs(double x){return x<0?-x:x;}
int n,m,deg[505];
double A[505][505];
struct Edge
{
    int u,v;
    double w;
    void read()
    {
        scanf("%d%d",&u,&v);
        ++deg[u],++deg[v];
    }
    bool operator < (const Edge &s)const
        {return w<s.w;}
}e[130000];
void Gauss_Elimination()//高斯消元模板，还是那个样子，没有变
{
```

```
        int i,j,k,r;
        for(i=1;i<=n;++i)
        {
            r=i;
            for(j=i+1;j<=n;++j)
                if(fabs(A[j][i])>fabs(A[r][i]))r=j;
            if(r!=i)for(j=1;j<=n+1;++j)swap(A[r][j],A[i][j]);
            for(k=i+1;k<=n;++k)
            {
                double f=A[k][i]/A[i][i];
                for(j=i;j<=n+1;++j)A[k][j]-=f*A[i][j];
            }
        }
        for(i=n;i>=1;--i)
        {
            for(j=i+1;j<=n;++j)
                A[i][n+1]-=A[j][n+1]*A[i][j];
            A[i][n+1]/=A[i][i];
        }
    }
    int main()
    {
        scanf("%d%d",&n,&m);
        for(int i=1;i<=m;++i)
            e[i].read();
        for(int i=1;i<=m;++i)//构造系数矩阵
     {
            int u=e[i].u,v=e[i].v;
            if(u!=n)A[u][v]+=1.0/deg[v];
            if(v!=n)A[v][u]+=1.0/deg[u];
        }
        for(int i=1;i<n;++i)
            A[i][i]-=1;
        A[n][n]=1,A[n][n+1]=0;
        A[1][n+1]=-1;
        Gauss_Elimination();
        for(int i=1;i<=m;++i)//计算每个边的期望经过次数
     {
            e[i].w+=A[e[i].u][n+1]/deg[e[i].u];
            e[i].w+=A[e[i].v][n+1]/deg[e[i].v];
        }
        std::sort(e+1,e+1+m);
        double ans=0.0;
        for(int i=m;i>=1;--i)//排序定边权+求和
         ans+=e[i].w*(m+1-i);
        printf("%.3f\n",ans);
    }
```

April 6, 2014
概率论 高斯消元

[Solution]BZOJ2337: [HNOI2011]XOR和路径

题目：

戳这里

题解：
高斯消元法解概率DP
第二道高斯消元求期望，这做的也太卡手了……做到现在都有两个半小时了……

首先，异或运算不能直接加和，但是我们可以把原来的边的权值拆成二进制，一位一位运算
比如说，我们算出从1走到n，第一位的期望异或和是0.2，第二位是0.7，那么可以合并最终答案：0.7*2+0.2*1
这样，我们可以拆成30次高斯消元，求出每一位的期望异或和，然后合并就可以了
至于为什么用高斯消元做概率DP，是因为转移关系错综复杂互相依赖，这种环形的数据范围不大的题目适合使用高斯消元求解
我们可以设从i号点到达n号点的异或和为1的概率是Ai（也可以理解成从i到n的期望异或和为Ai，仔细想想这其实是是一个意思）
那么状态转移方程就是：

$$A_i=\sum_{w(i,j)=0}\frac{A_j}{deg(i)}+\sum_{w(i,j)=1}\frac{1-A_j}{deg(i)},\text{其中}deg[i]\text{为}i\text{点的出度}$$

提示：1-Aj是j点异或和为0的概率
有了这个方程，我们可以依次考察每一条边，构建方程组
对于一条边，w(i,j)=0：A[i][j]+=1.0/deg[i]
对于一条边，w(i,j)=1：A[i][j]-=1.0/deg[i],A[i][n+1]-=1.0/deg[i]
对于每个点i，A[i][i]-=1
对于终点n，A[n][n]=1,A[n][n+1]=0（保证A[n]==0）
然后高斯消元模板即可AC……
注意事项：
1.题目说了有重边和自环，重边无所谓，自环需要注意一下(本人wa了一次)
2.这题应该不卡精度，直接double消元即可(本人第一次自环没处理好，调了半天以为是精度问题……SB啊)
3.注意0和0.0的区别（0.0默认是double，0默认是int），除法时容易导致奇葩错误
贴代码：

bzoj2337.cpp

```
#include<cstdio>
#include<cstring>
int deg[105],n,m;
double A[105][105];
void swap(double &a,double &b){double t=a;a=b;b=t;}
inline double abs(double x){return x<0?-x:x;}
struct Edge
{
    int u,v,w;
    void read()
    {
        scanf("%d%d%d",&u,&v,&w);
        ++deg[u];
        if(u!=v)++deg[v];//注意自环
 }
}e[10005];
void init_matrix(int t)//构建系数矩阵
{
    memset(A,0,sizeof(A));
    for(int i=1;i<=m;++i)
    {
        int u=e[i].u,v=e[i].v,w=e[i].w;
        if(u!=n)
        {
            if((w&(1<<t))==0)
                A[u][v]+=1.0/deg[u];
            else A[u][v]-=1.0/deg[u],
                A[u][n+1]-=1.0/deg[u];
        }
```

```
            if(v!=n && u!=v)//还是注意自环
         {
                if((w&(1<<t))==0)
                    A[v][u]+=1.0/deg[v];
                else A[v][u]-=1.0/deg[v],
                    A[v][n+1]-=1.0/deg[v];
            }
        }
        for(int i=1;i<n;++i)
            A[i][i]-=1;
        A[n][n]=1,A[n][n+1]=0;
    }
    void Gauss_Elimination()//高斯消元模板
    {
        int i,j,k,r;
        for(i=1;i<=n;++i)
        {
            r=i;
            for(j=i+1;j<=n;++j)
                if(abs(A[r][i]) < abs(A[j][i]))r=j;
            if(r!=i)for(j=1;j<=n+1;++j)swap(A[r][j],A[i][j]);
            for(k=i+1;k<=n;++k)
            {
                double f=A[k][i]/A[i][i];
                for(j=i;j<=n+1;++j)A[k][j]-=f*A[i][j];
            }
        }
        for(i=n;i>=1;--i)
        {
            for(j=i+1;j<=n;++j)
                A[i][n+1]-=A[j][n+1]*A[i][j];
            A[i][n+1]/=A[i][i];
        }
    }
    int main()
    {
        double ans=0.0;
        scanf("%d%d",&n,&m);
        for(int i=1;i<=m;++i)
            e[i].read();
        for(int i=0;i<30;++i)
        {
            init_matrix(i);
            Gauss_Elimination();
            ans+=A[1][n+1]*(1<<i);
        }
        printf("%.3f\n",ans);
    }
```

Next
Blog Archives