References For ACM/ICPC World Finals 2016

University of California at Berkeley

Berkeley Blue
Alexander Dai, Weiqiao Han, and Pasin Manurangsi

May 15, 2016

Contents

Practical Tools

Common Math Formulas

1. Summations

$$\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1) \quad , \quad \sum_{i=1}^{n} i^3 = \frac{1}{4}n^2(n+1)^2$$

$$\sum_{i=1}^{n} i^4 = \frac{1}{30}n(n+1)(2n+1)(3n^2+3n-1)$$

$$\sum_{i=1}^{n} i^5 = \frac{1}{12}n^2(n+1)^2(2n^2+2n-1)$$

$$\sum_{i=1}^{n} \frac{1}{i(i+1)(i+2)} = \frac{1}{4} - \frac{1}{2(n+1)(n+2)}$$

$$\sum_{i=1}^{n} \frac{1}{i(i+1)(i+2)(i+3)} = \frac{1}{18} - \frac{1}{3(n+1)(n+2)(n+3)}$$

2. Simpson's rule

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

3. Circum-Circle in 3D space given A, B, C

$$\begin{aligned} O &= \Big(\|C-A\|^2[(B-A)\times(C-A)]\times(B-A) \\ &\quad +\|B-A\|^2(C-A)\times[(B-A)\times(C-A)]\Big) \\ &\quad /\left(2\|(B-A)\times(C-A)\|^2\right) + A \end{aligned}$$

4. Circum-Ball in 3D space given A, B, C, D

$$\begin{aligned} V &= \begin{vmatrix} x_b - x_a & y_b - y_a & z_b - z_a \\ x_c - x_a & y_c - y_a & z_c - z_a \\ x_d - x_a & y_d - y_a & z_d - z_a \end{vmatrix} \\ O &= \Big(\|D-A\|^2[(B-A)\times(C-A)] + \|C-A\|^2[(D-A)\times(B-A)] \\ &\quad +\|B-A\|^2[(C-A)\times(D-A)]\Big)/V + A \end{aligned}$$

Magic Square

```
void build(int n,int a[][maxn]) //No solutions when n=2
{
  int i,j,k,n2=n*n,m=n/2,m2=m*m;
  for(i=0;i<n;i++) for(j=0;j<n;j++) a[i][j]=0;
  if (n==2) return;        // No solutions
  if (n%2==1)
    for(i=0,j=n/2,k=1;k<=n2;k++) {
      a[i][j] = k;
      if(!a[(i+n-1)%n][(j+1)%n]) { i=(i+n-1)%n;  j=(j+1)%n; }
      else i=(i+1)%n;
    }
  else if(n%4==0)
    for(k=0,i=0;i<n;i++) for(j=0;j<n;j++) {
      a[i][j] = ++k;
      if (i%4==j%4||i%4+j%4==3) a[i][j]=n2+1-a[i][j];
    }
  else if(n%4==2)
    for(i=0,j=m/2,k=0;k<m2;k++) {
      if((i<=m/2 &&
        !(i==m/2&&j==m/2))||(i==m/2+1&&j==m/2)){ // L
        a[i*2  ][j*2+1]=k*4+1;  a[i*2+1][j*2]=k*4+2;
        a[i*2+1][j*2+1]=k*4+3;  a[i*2  ][j*2]=k*4+4;
      } else if (i>m/2+1) { // X
        a[i*2  ][j*2]=k*4+1;  a[i*2+1][j*2+1]=k*4+2;
        a[i*2+1][j*2]=k*4+3;  a[i*2  ][j*2+1]=k*4+4;
      } else { // U
        a[i*2  ][j*2  ]=k*4+1; a[i*2+1][j*2]=k*4+2;
        a[i*2+1][j*2+1]=k*4+3; a[i*2][j*2+1]=k*4+4;
      }
      if (!a[(i+m-1)%m*2][(j+1)%m*2]) i=(i+m-1)%m,j=(j+1)%m;
      else i=(i+1)%m;
    }
}
```

High Precision

```
const int maxlen = 10000, base = 10;

class HP { public:
  int len, s[maxlen];  HP() { (*this)=0; };
  HP(int inte) {(*this)=inte; };
  HP(const char*str) { (*this)=str; };
  friend ostream& operator<<(ostream &cout,const HP &x);
  HP operator=(int inte);   HP operator=(const char*str);
  HP operator*(const HP &b);  HP operator+(const HP &b);
  HP operator-(const HP &b);  HP operator/(const HP &b);
  HP operator%(const HP &b);  int Compare(const HP &b);
};

ostream& operator<<(ostream &cout,const HP &x)
{
  cout<<x.s[x.len];
  for(int i=x.len-1;i>=1;i--) {
    for (int j=base/10;j>1;j/=10)
    if (x.s[i]<j) cout<<'0';
    cout<<x.s[i];
  }
  return cout;
}

HP HP::operator=(const char *str)
{
  len=strlen(str);
  for(int i=1,j=1,k=1;i<=len;i++) {
    s[k]+=(str[len-i]-'0')*j;
    if ((j*=10)==base) {j=1;++k;}
  }
  return *this;
}

HP HP::operator=(int inte)
{
  if(inte==0) { len=1; s[1]=0; return (*this);};
  for(len=0;inte>0;){ s[++len]=inte%base; inte/=base;};
  return (*this);
}

HP HP::operator*(const HP &b)
{
  static long long buf[maxlen];
  int i,j; HP c; c.len=len+b.len;
  for(i=1;i<=c.len;i++) buf[i]=0;
  for(i=1;i<=len;i++)
    for(j=1;j<=b.len;j++) buf[i+j-1]+=s[i]*b.s[j];
  for(i=1;i<c.len;i++) {
    buf[i+1]+=buf[i]/base; buf[i]%=base;
  }
  while(buf[i]) {buf[i+1]=buf[i]/base;buf[i]%=base;i++;}
  while(i>1 && !buf[i]) i--; c.len=i;
  for (i=1;i<=c.len;++i) c.s[i]=buf[i];
  return c;
}

HP HP::operator+(const HP &b)
{
  int i;  HP c;  c.s[1]=0;
  for(i=1; i<=len || i<=b.len || c.s[i] ;i++) {
    if(i<=len) c.s[i]+=s[i];
    if(i<=b.len) c.s[i]+=b.s[i];
    c.s[i+1]=c.s[i]/base;  c.s[i]%=base;
  }
  c.len=i-1;  if( c.len==0 ) c.len=1;
  return c;
}

HP HP::operator-(const HP &b)
{
  int i,j; HP c;
  for(i=1,j=0; i<=len ;i++) {
    c.s[i]=s[i]-j; if(i<=b.len) c.s[i]-=b.s[i];
    if(c.s[i]<0){ j=1 ; c.s[i]+=base; } else j=0;
  }
  c.len=len; while(c.len>1 && !c.s[c.len]) c.len--;
  return c;
}

int HP::Compare(const HP &y)
{
  if(len>y.len) return 1;
  if(len<y.len) return -1;
  int i=len;
  while((i>1)&&(s[i]==y.s[i])) i--;
  return s[i]-y.s[i];
}

HP HP::operator/(const HP &b)
{
  int i,j; HP d(0),c;
  for(i=len;i>0;i--) {
    if(!(d.len==1 && d.s[1]==0))
      { for(j=d.len;j>0;j--) d.s[j+1]=d.s[j]; ++d.len; }
    d.s[1]=s[i];  c.s[i]=0;
    while( (j=d.Compare(b))>=0 )
      { d=d-b;  c.s[i]++;  if(j==0) break; }
  }
  c.len=len;  while((c.len>1)&&(c.s[c.len]==0)) c.len--;
  return c;
}

HP HP::operator%(const HP &b)
{
  int i,j; HP d(0);
  for(i=len;i>0;i--) {
    if(!(d.len==1 && d.s[1]==0))
      { for(j=d.len;j>0;j--) d.s[j+1]=d.s[j]; ++d.len; }
    d.s[1]=s[i];
    while( (j=d.Compare(b))>=0 ){ d=d-b; if(j==0) break; }
  }
  return d;
}
```

HP SQRT

```
int x[maxlen],y[maxlen],z[maxlen],bck[maxlen],lx,ly,lz;

int IsSmaller()  // is z<=y ?
{int i=ly;while(i>1&&z[i]==y[i])i--;return(z[i]<=y[i]);}

void Solve() // y^2=x
{
  int i,j,k;
  lx=(ly+1)/2;  ly=lx*2;
  memset(x,0,sizeof(x)); memset(z,0,sizeof(z));
  for(i=lx;i>0;i--){
    for(j=1; j<10; x[i]=j++){
      memcpy(bck,z,sizeof(z));
      z[2*i-1]++; for(k=i;k<=lx;k++)
      {z[i-1+k]+=2*x[k];z[i+k]+=z[i-1+k]/10;z[i-1+k]%=10;}
      for(k=lx+i;k<=ly;k++){ z[k+1]+=z[k]/10; z[k]%=10;}
      if(!IsSmaller()) break;
    };
    if(j<10) memcpy(z,bck,sizeof(bck));
  };
  for(i=lx;i>0;i--) cout<<x[i]; cout<<endl;
}

int main()
{
  char ch,s[maxlen]; int i,j;
  memset(y,0,sizeof(y));
  cin>>s; ly=strlen(s);
  for(i=0;i<ly;i++) y[i+1]=s[ly-1-i]-'0';
  Solve();
  return 0;
}
```

Graph Theory

Minimum Cost Flow

```

// Const INF,MAXE,MAXN;
// Const Struct Edge
// Edge e:MAXE
// int head,tot,vst,flag,dist,mark:MAXN
class MinCostFlow{ private:
  int st, en, N;
  int ret,val;// ret denotes flow, val denotes cost
  int dfs(int v, int cap) {
    if(v==en) {
      val+=cap*dist[st];
      return cap;
    }
    vst[v]=true;
    int flow=0;
    for(int i=head[v];i!=-1;i=e[i].nxt)
      if(e[i].cap && !vst[e[i].v]
        && dist[e[i].v]+e[i].cost==dist[v]) {
        int det=dfs(e[i].v,min(cap,e[i].cap));
        if(det) {
          e[i].cap-=det; e[i^1].cap+=det; flow+=det;
          if(!(cap-=det)) break;
        }
      }
    return flow;
  }
  int relabel() {
    int det=0x7fffffff;
    for(int i=0;i<N;++i)
      if(vst[i])
        for(int j=head[i];j!=-1;j=e[j].nxt)
          if(e[j].cap && !vst[e[j].v])
            det=min(det,dist[e[j].v]+e[j].cost-dist[i]);
    if(det==0x7fffffff) return false;
    for(int i=0;i<N;++i)
      if(vst[i]) dist[i]+=det;
    return true;
  }
  int spfa() {
    deque<int>que;
    fill(vst,vst+N,-1); vst[en]=-2;
    fill(dist,dist+N,INF); dist[en]=0;
    que.push_back(en);
    fill(mark,mark+N,0);
    while(que.size()) {
      int u=que.front(); que.pop_front();
      mark[u]=0;
      for(int i=head[u];i!=-1;i=e[i].nxt)
        if(e[i^1].cap&&dist[u]+e[i^1].cost<dist[e[i].v]) {
          dist[e[i].v]=dist[u]+e[i^1].cost;vst[e[i].v]=i^1;
          if(!mark[e[i].v]) {
            mark[e[i].v]=1;
            if(dist[e[i].v]<dist[e[i].u])
              que.push_front(e[i].v);
            else que.push_back(e[i].v);
          }
        }
    }
  }
public:
  inline int setit(int S=0,int T=0,int _=0)
  {st=S;en=T;N=_;ret=0;fill(head,head+N,-1);tot=0;val=0;}
  inline int insert(int u,int v,int cap,int cost){
    e[tot]=edge(u,v,head[u],cap,cost);head[u]=tot++;
    e[tot]=edge(v,u,head[v],0,-cost);head[v]=tot++;
  }
  inline int work(){
    spfa();
    do
      do fill(vst,vst+N,0);
      while (dfs(st, 0x7fffffff));
    while(relabel());
  }
};
```

Canceling Negative Circles

```
// Const MAXE,MAXN;
// Const Struct Edge
// Edge e:MAXE
// int dist,sign,pre,mark:MAXN
int n,m,base;//V,E,Answer

int Find_Circle() {
  bool ret = false;
  fill(mark,mark+n+1,false);
  fill(sign, sign + n + 1, -1);
  for(int i=0;i<=n;++i){
    if(mark[i]) continue;
    int u = i, len = 0;
    if(pre[i] < 0) {
      mark[i] = true; continue;
    }
    while(!mark[u] && pre[u] > -1) {
      mark[u] = true;
      sign[u] = i;
      u = e[pre[u]].u;
    }
    if(sign[u] == i) {
      int v=u;
      do {
        if(abs(e[pre[v]].w)<inf)base+=e[pre[v]].w;
        e[pre[v]].cap--;e[pre[v]^1].cap++;
        int t = pre[v];
        v=e[t].u;
      } while(v != u);
      ret = true;
    }
  }
  return ret;
}

int Cancel_Nagtive_Circles() {
  fill(dist,dist+n+1,0);
  fill(pre,pre+n+1,-1);
  while(1) {
    bool found=false;
    for(int tm=0;tm<=n+1;++tm){
      bool flag = true;
      for(int i=0;i<tot;++i)  {
        if(e[i].cap && dist[e[i].u]+e[i].w<dist[e[i].v]){
          pre[e[i].v]=i;
          dist[e[i].v]=dist[e[i].u]+e[i].w;
          flag=false;
        }
      }
      if(Find_Circle()){
        found = true;
        break;
      }
      if(flag) break;
    }
    if(!found) return 0;
    fill(dist,dist+n+1,0);
    fill(pre,pre+n+1,-1);
  }
}
```

Bridge

```
int n,g[maxn][maxn],mk[maxn],d[maxn],low[maxn];
int color,ti,bridgenum,bridgeu[maxn],bridgev[maxn];

void dfsvisit(int u,int p)
{
  int v,s=0,bBridge=0; low[u]=d[u]=++ti; mk[u]=-color;
  for(v=1; v<=n; v++) if(g[u][v] && v!=p)
    if(mk[v]==0){ dfsvisit(v,u); s++;
      if(low[v]<low[u]) low[u]=low[v];
      if(low[v]==d[v]) {
        bridgeu[bridgenum  ]=u;
        bridgev[bridgenum++]=v;
      }
    } else if(d[v]<low[u]) low[u]=d[v];
  mk[u]=color;
}

void dfs()
{
  int i,j,k; memset(mk,0,sizeof(mk));
  color=ti=bridgenum=0;
  for(i=1; i<=n; i++)
    if(!mk[i]){ ++color; dfsvisit(i,0); }
  cout<<bridgenum<<endl;
}
```

Build Block Tree

```
struct edge e[MaxM];
int head[MaxN], Index[MaxM], tot;
int dep[MaxN], low[MaxN], iscut[MaxN];
vector<vector<int> > block;
int seq[MaxN], top;
int clear() {
  block.clear(); fill(head,head + n, -1); top = tot = 0;
}
int Dfs(int u, int d, int pre) {
  low[u] = dep[u] = d;
  seq[top ++] = u;
  iscut[u] = false;
  int degree = 0;
  for(int i=head[u]; i!=-1; i=e[i].nxt) {
    if((i ^ 1) == pre) continue;
    int v = e[i].v;
    if(dep[v] < 0) {
      ++ degree;
      int rec = top;
      Dfs(v, d + 1, i);
      if(low[v] >= d) { // find a block
        if(d || degree > 1)
          iscut[u] = true;
        vector<int> tmp;
        tmp.push_back(u);
        while(top > rec)
          tmp.push_back(seq[--top]);
        block.push_back(tmp);
      } else
        low[u] = min(low[u], low[v]);
    } else
      low[u] = min(low[u], dep[v]);
  }
}

int mark[MaxN], vst[MaxN];

int Bfs(int st, int id) {
  queue<int> Q;
  Q.push(st);
  vst[st] = true;
  while(!Q.empty()) {
    int u = Q.front(); Q.pop();
    for(int i=head[u]; i!=-1;i=e[i].nxt) {
      int v = e[i].v;
      if(mark[v] != id) continue;
      Index[i] = id;
      if(!vst[v]) {
        vst[v] = true;
        Q.push(v);
      }
    }
  }
}

int cutid[MaxN],s;// s is the number blocks
vector<vector<int> > adj;//adj is the tree

int Build_Tree() {
  for(int i=0;i<block.size();++i) {
    vector<int>&vec = block[i];
    for(int j=0;j<vec.size();++j) {
      mark[vec[j]] = i;
      vst[vec[j]] = false;
    }
    Bfs(vec[0], i);
  }
  for(int i=0;i<n;++i) {
    cutid[i] = -1;
    if(iscut[i]) {
      cutid[i] = block.size();
      vector<int> tmp;
      tmp.push_back(i);
      block.push_back(tmp);
    }
  }
  adj.clear();
  s = block.size();
  adj.resize(block.size());
  for(int i=0;i<block.size();++i)
    if(block[i].size() > 1) {
      vector<int>&vec = block[i];
      for(int j=0;j<vec.size();++j)
        if(iscut[vec[j]]) {
          adj[cutid[vec[j]]].push_back(i);
          adj[i].push_back(cutid[vec[j]]);
        }
    }
}
```

Minimum Directed Spanning Tree

```
int n,g[maxn][maxn],used[maxn],pass[maxn];
int eg[maxn],more,queue[maxn];
void combine(int id, int& sum) {
  int tot = 0, from, i, j, k;
  for(;id!=0&&!pass[id]; id=eg[id])
    {queue[tot++]=id; pass[id]=1;}
  for(from=0;from<tot&&queue[from]!=id;from++);
  if(from==tot) return;  more = 1;
  for(i=from; i<tot; i++) {
    sum+=g[eg[queue[i]]][queue[i]];
    if(i!=from){  used[queue[i]]=1;
      for(j = 1; j <= n; j++) if(!used[j])
        if(g[queue[i]][j]<g[id][j])
          g[id][j]=g[queue[i]][j];
    }
  }
  for(i=1; i<=n; i++) if(!used[i]&&i!=id) {
    for(j=from; j<tot; j++){
      k=queue[j];
      if(g[i][id]>g[i][k]-g[eg[k]][k])
        g[i][id]=g[i][k]-g[eg[k]][k];
    }
  }
}

int msdt(int root) {
  // return the total length of MDST
  int i, j, k, sum = 0;
  memset(used, 0, sizeof(used));
  for(more=1; more;){ more = 0;
    memset(eg, 0, sizeof(eg));
    for(i = 1; i <= n; i++)
      if(!used[i] && i != root) {
        for(j = 1, k = 0; j <= n; j++)
          if(!used[j] && i != j)
            if(k==0 || g[j][i]<g[k][i]) k=j;
        eg[i] = k;
    } memset(pass, 0, sizeof(pass));
    for(i=1;i<=n;i++)
      if(!used[i]&&!pass[i]&&i!=root)
        combine(i,sum);
  }
  for(i=1; i<=n; i++)
    if(!used[i] && i!=root)
      sum+=g[eg[i]][i];
  return sum;
}
```

KM $O(N^3)$

```
int N,M,val[][],lx[],ly[],vx[],vy[],match[];
int slack[],slackx[],conn[];
int find(int s) {
  for(int i=0;i<M;++i)
    slack[i] = lx[s] + ly[i] - val[s][i],
    slackx[i] = s;
  int flag, det;
  ME(vx); ME(vy); MM(conn, -1); vx[s] = 1;
  while (1) {
    flag = false; det = 0x7fffffff;
    for(int i=0;i<M;++i) {
      if(!vy[i]) {
        det <?= slack[i];
        if(slack[i] == 0) {
```

```
     flag = true; vy[i] = 1;
     if(match[i] < 0) {
      int u = i;
      for(; 1; ) {
       match[u] = slackx[u];
       if(match[u] == s) break;
       u = conn[match[u]];
      }
      return 1;
     } else {
      int j = match[i];
      if(!vx[j]) {
       vx[j] = 1;
       conn[j] = i;
       for(int k=0;k<M;++k)
        if(!vy[k]&&slack[k]>lx[j]+ly[k]-val[j][k])
         slack[k] = lx[j]+ly[k]-val[j][k],
         slackx[k] = j;
      }}}}
  }
  if(!flag) {
   for(int i=0;i<N;++i)
    if(vx[i]) lx[i] -= det;
   for(int j=0;j<M;++j)
    if(vy[j]) ly[j] += det;
    else slack[j] -= det;
  }}
}
int run() { // KM algo
 MM(match, -1); ME(ly);
 for(int i=0;i<N;++i) {
  lx[i] = -0x7fffffff;
  for(int j=0;j<M;++j)
   lx[i] >?= val[i][j];
 }
 for(int i=0;i<N;++i) find(i);
 res = 0;
 for(int i=0;i<M;++i)
  res += val[match[i]][i];
}
```

Matching on General Graph

```
// total is the maximum cardinality
// p[1..n] means a match: i <-> p[i]
int g[maxn][maxn],p[maxn],l[maxn][3];
int n,total,status[maxn],visited[maxn];

void solve()
{
 int i,j,k,pass;
 memset(p,0,sizeof(p));
 do{ i=0;
  do{ if(p[++i]) pass=0; else {
    memset(l,0,sizeof(l));
    l[i][2]=0xff; pass=path(i);
    for(j=1;j<=n;j++) for(k=1;k<=n;k++)
     if(g[j][k]<0) g[j][k]=-g[j][k];
   };
  }while( i!=n && !pass);
  if(pass) total+=2;
 }while(i!=n && total!=n);
}

void upgrade(int r)
{
 int j=r, i=l[r][1];
 for(p[i]=j; l[i][2]<0xff;){
  p[j]=i; j=l[i][2]; i=l[j][1]; p[i]=j;
 }  p[j]=i;
}
```

```
int path(int r)
{
 int i,j,k,v,t,quit;
 memset(status,0,sizeof(status)); status[r]=2;
 do{ quit=1;
  for(i=1;i<=n;i++) if(status[i]>1)
   for(j=1;j<=n;j++) if(g[i][j]>0 && p[j]!=i)
    if(status[j]==0) {
     if(p[j]==0){l[j][1]=i; upgrade(j); return 1;}
     else
     if(p[j]>0) {
      g[i][j]=g[j][i]=-1; status[j]=1;
      l[j][1]=i;  g[j][p[j]]=g[p[j]][j]=-1;
      l[p[j]][2]=j;   status[p[j]]=2;
      quit=0;
     }
    } else
    if(status[j]>1 && (status[i]+status[j]<6)){
     quit=0; g[i][j]=g[j][i]=-1;
     memset(visited,0,sizeof(visited));
     visited[i]=1;   k=i; v=2;
     while(l[k][v]!=0xff)
      {k=l[k][v]; v=3-v; visited[k]=1;}
     k=j; v=2;
     while(!visited[k]) { k=l[k][v]; v=3-v; }
     if(status[i]!=3) l[i][1]=j;
     if(status[j]!=3) l[j][1]=i;
     status[i]=status[j]=3;  t=i; v=2;
     while(t!=k) {
      if(status[l[t][v]]!=3) l[l[t][v]][v]=t;
      t=l[t][v]; status[t]=3; v=3-v;
     }
     t=j; v=2;
     while(t!=k) {
      if(status[l[t][v]]!=3) l[l[t][v]][v]=t;
      t=l[t][v]; status[t]=3; v=3-v;
     }
    }
 }while(!quit);
 return 0;
}
```

Check Chordal Graph

```
int n,m,mk[maxn],degree[maxn],PEO[maxn],g[maxn][maxn];

int Chordal()
{
 memset(mk,0,sizeof(mk));
 memset(degree,0,sizeof(degree));
 for(int j,k,u,v,i=0;i<n;i++) {
    j=-1;   u=-1;
    for(k=0;k<n;k++)
      if(!mk[k]&&(j<0||degree[k]>degree[j])) j=k;
    mk[j]=1; PEO[i]=j;
    for(k=i-1;k>=0;k--) if( g[j][PEO[k]] )
      if( u<0 ) u=PEO[k]; else if( !g[u][PEO[k]]) return 0;
    for(k=0;k<n;k++) if(!mk[k] && g[j][k]) degree[k]++;
  }
  return 1;
}
```

Degree Restriction MST

```
#include <stdio.h>
#include <string.h>
#define MAX (100+5)
int n, graph[MAX][MAX], K;
int tree[MAX][MAX], sol;
void in(); void out();
void min_deg_mst();
int inc_deg(); void k_deg_mst();
int main() {
  in(); k_deg_mst(); out();
  return 0;
}
void in() {
  int i, e, x, y, l;
  memset(graph, 0x3F, sizeof(graph));
  scanf("%d %d", &n, &K);
  scanf("%d", &e);
  for (i = 0; i < e; i++) {
    scanf("%d %d %d", &x, &y, &l);
    if (l<graph[x][y]) graph[x][y]=graph[y][x]=l;
  }
}
void out() {
  int i, j, sum = 0;
  if (!sol) { printf("No solution.\n"); return; }
  for (i = 1; i <= n; i++)
    for (j = 1; j <= tree[i][0]; j++) if(i<tree[i][j]){
      printf("%d %d\n", i, tree[i][j]);
      sum += graph[i][tree[i][j]];
    }
  printf("%d\n", sum);
}

int value[MAX], used[MAX], from[MAX], que[MAX];
void min_deg_mst() { // Prim: O(n^2)
  int now, root, minv, i, j;

  memset(value, -1, sizeof(value));
  memset(used, 0, sizeof(used));
  memset(tree, 0, sizeof(tree));
  memset(from, 0, sizeof(from));
  for (now = 1, root = 2; now < n;) {
    for (; root <= n; root++)
      if (!used[root]) break;
    value[root] = 0;
    for (que[0] = 0;; now++) {
      for (i = 2, minv = 2147483647; i <= n; i++)
        if (!used[i] && value[i] < minv)
           minv = value[i], j = i;
      if (minv > 1000000000) break;
      used[j] = 1, que[++que[0]] = j;
      for (i = 2; i <= n; i++)
        if (!used[i] && graph[j][i] < value[i])
          value[i] = graph[j][i], from[i] = j;
    }
    for (i = 1, minv = INF; i <= que[0]; i++)
      if (minv > graph[1][que[i]])
        minv = graph[1][que[i]], j = que[i];
    tree[1][++tree[1][0]] = j;
    tree[j][++tree[j][0]] = 1;
  }
  for (i = 2; i <= n; i++) if (from[i]) {
    tree[i][++tree[i][0]] = from[i];
    tree[from[i]][++tree[from[i]][0]] = i;
  }
}

int ledge[MAX][3];
//ledge[i][0] is longest edge e' on the path between v1 ans vi
//ledge[i][1] and ledge[i][2] are two connections of e'

int inc_deg() { // O(n)
  int i, j, k, now, next, minv = 2147483647, id;

  memset(used, 0, sizeof(used)); used[1] = 1;
  for (i = 1, que[0] = 0; i <= tree[1][0]; i++) {
    que[++que[0]] = tree[1][i];
    used[tree[1][i]] = 1, ledge[tree[1][i]][0] = -INF;
  }
  for (i = 1; i <= que[0]; i++)
    for (now = que[i], j = 1; j <= tree[now][0]; j++)
      if (!used[next = tree[now][j]]) {
        que[++que[0]] = next; used[next] = 1;
        memcpy(ledge[next], ledge[now], sizeof(ledge[now]));
        if (graph[now][next] > ledge[next][0]) {
          ledge[next][0] = graph[now][next];
          ledge[next][1] = now;
          ledge[next][2] = next;
        }
      if (graph[1][next] - ledge[next][0] < minv)
        minv = graph[1][next] - ledge[next][0], id = next;
      }

  if (minv >= 1000000000) return 1;
  tree[1][++tree[1][0]] = id;
  tree[id][++tree[id][0]] = 1;
  j = ledge[id][1], k = ledge[id][2];
  for (i = 1; i <= tree[j][0]; i++)
    if (tree[j][i] == k)
      break;
  tree[j][i] = tree[j][tree[j][0]--];
  for (i = 1; i <= tree[k][0]; i++)
    if (tree[k][i] == j)
      break;
  tree[k][i] = tree[k][tree[k][0]--];
  return 0;
}

void k_deg_mst() {
  min_deg_mst();
  while (tree[1][0] < K)
```

```
    if (inc_deg()) break;
  sol = tree[1][0] == K;
}
```

Geometry

Basic Operations

```
const double eps = 1e-10;
const double pi = acos(-1);

inline int dcmp(const double&a)
{return fabs(a)<=eps?0:(a<0?-1:1);}

inline double operator ^(CPX a, CPX b)
{return a.real() * b.imag() - a.imag() * b.real();}
inline double operator &(CPX a, CPX b)
{return a.real() * b.real() + a.imag() * b.imag();}
inline bool operator < (CPX a, CPX b)
{return dcmp(a.real() - b.real()) ?
  a.real() < b.real() : dcmp(a.imag() - b.imag()) < 0;}

// Crossing Angle of P0P1 -> P0P2, range in (-pi,pi]
double angle(CPoint p0,CPoint p1,CPoint p2)
{
  double cr = cross(p0,p1,p2);
  double dt =  dot (p0,p1,p2);
  if(dcmp(cr)==0) cr=0.0;
  if(dcmp(dt)==0) dt=0.0;
  return atan2(cr,dt);
}

int PointOnLine(CPoint p0,CPoint p1,CPoint p2)
{
  return dcmp(cross(p0,p1,p2))==0;
}

int PointOnSegment(CPoint p0,CPoint p1,CPoint p2)
{
  return dcmp(cross(p0,p1,p2))==0 && dcmp(dot(p0,p1,p2))<=0;
}

// 1 = cross;   0 = parallel;   -1 = overlap
int LineIntersection
  (CPoint p1,CPoint p2,CPoint p3,CPoint p4,CPoint &cp)
{
  double u=cross(p1,p2,p3), v=cross(p2,p1,p4);
  if( dcmp(u+v) )
  {
    cp.x=(p3.x*v + p4.x*u) / (v+u);
    cp.y=(p3.y*v + p4.y*u) / (v+u);
    return 1;
  }
  if( dcmp(u) ) return 0;   // else u=v=0;
  if( dcmp(cross(p3,p4,p1)) ) return 0;
  return -1;
}

int SegmentIntersection
  (CPoint p1,CPoint p2,CPoint p3,CPoint p4,CPoint &cp)
{
  int ret=LineIntersection(p1,p2,p3,p4,cp);
  if(ret==1) return PointOnSegment(cp,p1,p2)
                  && PointOnSegment(cp,p3,p4);
  if(ret==-1 &&
    ( PointOnSegment(p1,p3,p4) || PointOnSegment(p2,p3,p4)
      || PointOnSegment(p3,p1,p2) || PointOnSegment(p4,p1,p2) ))
    return -1;
  return 0;
}

int SegmentIntersecTest
  (CPoint p1,CPoint p2,CPoint p3,CPoint p4)
{
  if(  max(p1.x, p2.x) + eps < min(p3.x, p4.x) ||
     max(p3.x, p4.x) + eps < min(p1.x, p2.x) ||
     max(p1.y, p2.y) + eps < min(p3.y, p4.y) ||
     max(p3.y, p4.y) + eps < min(p1.y, p2.y) ) return 0;
  int d1=dcmp(cross(p3,p4,p2));
  int d2=dcmp(cross(p3,p4,p1));
  int d3=dcmp(cross(p1,p2,p4));
  int d4=dcmp(cross(p1,p2,p3));
  if( d1*d2==1 || d3*d4 ==1 ) return 0;
  if( d1==0 && d2==0 && d3==0 && d4==0 ) return -1;
  return 1;
}
```

```

// 0 = outside;   1 = inside;   2 = boundary
int PointInPolygon(CPoint cp,CPoint p[],int n)
{
  int i,k,d1,d2,wn=0;
  double sum=0;
  p[n]=p[0];
  for(i=0;i<n;i++)
  {
    if( PointOnSegment(cp,p[i],p[i+1]) ) return 2;
    k  = dcmp( cross(p[i],p[i+1],cp) );
    d1 = dcmp( p[i+0].y - cp.y );
    d2 = dcmp( p[i+1].y - cp.y );
    if(k>0 && d1<=0 && d2>0) wn++;
    if(k<0 && d2<=0 && d1>0) wn--;
  }
  return wn!=0;
}

double PointToLine(CPoint p0,CPoint p1,CPoint p2,CPoint &cp)
{
  double d=dis(p1,p2);
  double s = cross(p1,p2,p0)/d;
  cp.x = p0.x + s*(p2.y-p1.y)/d;
  cp.y = p0.y - s*(p2.x-p1.x)/d;
  return s; // ************* Signed Magnitude *************
}

void PointProjLine(CPoint p0,CPoint p1,CPoint p2,CPoint &cp)
{
  double t = dot(p1,p2,p0)/dot(p1,p2,p2);
  cp.x = p1.x + t*(p2.x-p1.x);
  cp.y = p1.y + t*(p2.y-p1.y);
}
```

Circles

Crossing of $|P - P_0| = r$ and $ax + by + c = 0$

```
int CircleCrossLine_1( CPoint p0, double r,
  double a, double b, double c, CPoint &cp1, CPoint &cp2)
{
  double aa = a * a, bb = b * b, s = aa + bb;
  double d = r*r*s - sqr(a*p0.x+b*p0.y+c);
  if( d+eps<0 ) return 0;
  if( d<eps ) d = 0; else d = sqrt( d );
  double ab = a * b, bd = b * d, ad = a * d;
  double xx = bb * p0.x - ab * p0.y - a * c;
  double yy = aa * p0.y - ab * p0.x - b * c;
  cp2.x = ( xx + bd ) / s;  cp2.y = ( yy - ad ) / s;
  cp1.x = ( xx - bd ) / s;  cp1.y = ( yy + ad ) / s;
  if( d>eps ) return 2; else return 1;
}
```

Crossing of $|P - P_0| = r$ and $\overrightarrow{P_1P_2}$

```
int CircleCrossLine_2( CPoint p0, double r,
        CPoint p1, CPoint p2, CPoint &cp1, CPoint &cp2)
{
  double d, d12, dx, dy;
  d = fabs(PointToLine( p0, p1, p2, cp1 ));
  if( dcmp(d-r) >0 ) return 0;
  if( dcmp(d-r)==0 ) { cp2 = cp1; return 1; }
  d = sqrt( r*r - d*d ) / dis( p1, p2 );
  dx = ( p2.x - p1.x ) * d;
  dy = ( p2.y - p1.y ) * d;
  cp2.x = cp1.x + dx;  cp2.y = cp1.y + dy;
  cp1.x = cp1.x - dx;  cp1.y = cp1.y - dy;
  return 2;
}
```

Crossing of $|P - P_1| = r_1$ and $|P - P_2| = r_2$

```
int CircleCrossCircle_1
  ( CPoint p1, double r1, CPoint p2, double r2,
    CPoint &cp1, CPoint &cp2 )
{
  double mx = p2.x-p1.x, sx = p2.x+p1.x, mx2 = mx*mx;
  double my = p2.y-p1.y, sy = p2.y+p1.y, my2 = my*my;
  double sq = mx2+my2, d = -(sq-sqr(r1-r2))*(sq-sqr(r1+r2));
  if( d+eps<0 )return 0; if( d<eps )d = 0; else d = sqrt(d);
  double x = mx*( (r1+r2)*(r1-r2) + mx*sx ) + sx*my2;
  double y = my*( (r1+r2)*(r1-r2) + my*sy ) + sy*mx2;
  double dx = mx*d, dy = my*d;  sq *= 2;
  cp1.x = ( x - dy ) / sq;  cp1.y = ( y + dx ) / sq;
  cp2.x = ( x + dy ) / sq;  cp2.y = ( y - dx ) / sq;
  if( d>eps ) return 2; else return 1;
}
```

Crossing of $|P - P_1| = r_1$ and $|P - P_2| = r_2$

```
int CircleCrossCircle_2
  ( CPoint p1, double r1, CPoint p2, double r2,
    CPoint &cp1, CPoint &cp2 )
{
  double a, b, c; CommonAxis( p1, r1, p2, r2, a, b, c);
  return CircleCrossLine_1( p1, r1, a, b, c, cp1, cp2);
}
```

Common Axis of $|P - P_1| = r_1$ and $|P - P_2| = r_2$ of the $ax + by + c = 0$ form

```
void CommonAxis
  ( CPoint p1, double r1, CPoint p2, double r2,
    double &a, double &b, double &c )
{
  double sx = p2.x + p1.x, mx = p2.x - p1.x;
  double sy = p2.y + p1.y, my = p2.y - p1.y;
  a = 2*mx; b = 2*my;
  c = - sx*mx - sy*my - (r1+r2)*(r1-r2);
}
```

Convex Poly Intersect Line

```
struct point {
  CPX p, q;
  double d;
  inline point (CPX p=0,CPX q=0)
    :p(p),q(q){d = arg(q - p);};
  CPX cross(CPX a, CPX b) {
    double sa = (q - p) ^ (a - p);
    double sb = (b - p) ^ (q - p);
    return (a * sb + b * sa) / (sa + sb);
  }
};
bool operator < (const point&a, const point &b)
{return a.d < b.d;}

struct polygon {
  vector<CPX > a, b, t;
  vector<point > deg;
  int calc_degree() {
    deg.clear();
    for(int i=0;i<b.size();++i) {
      CPX p = b[i], q = b[i + 1 == b.size() ? 0 : i + 1];
      deg.push_back(point(p, q));
    }
    sort(deg.begin(), deg.end());
  }
  int read() {
    a.clear();
    int Q; scanf("%d", &Q);
    double x, y;
    while(Q--){
      scanf("%lf %lf", &x, &y);
      a.push_back(CPX(x,y));
    }
    t = a;
    calc_convex(b, t);
    calc_degree();
  }
  int find_polar(double d) {
    if(dcmp(d - deg.back().d) > 0) return 0;
    if(!dcmp(d - deg.back().d)) return deg.size()-1;
    if(!dcmp(d - deg[0].d)) return 0;
    if(dcmp(d - deg[0].d) < 0) return 0;
    int lo = 0, hi = deg.size() - 1, mid;
    while(lo +1 < hi) {
      mid = lo + hi >> 1;
      int t = dcmp(d - deg[mid].d);
      if(!t) return mid;
      if(t < 0) hi = mid;
      else lo = mid;
    }
    return hi;
  }
  int find_polar(CPX p, CPX q, int &l , int & r) {
    l = find_polar(arg(q - p));
    r = find_polar(arg(p - q));
    if(l > r) swap(l, r);
  }
  int intersect(CPX p, CPX q, CPX &o) {
  // line [p, q]  o is the intersection point closest to p
    int l, r;
    find_polar(p, q, l, r);// assert(l < r);
    CPX v = p - q;
    double argv = arg(v);
    int sl = dcmp(v ^ (deg[l].p - q));
    int sr = dcmp(v ^ (deg[r].p - q));
    if(sl * sr == 1) return false;
    if(sl * sr == 0) {
      if(sl == 0) {
        if(dcmp(argv - deg[l].d) == 0) {
          if(norm(deg[l].p - p) < norm(deg[l].q - p))
            o = deg[l].p;
          else
            o = deg[l].q;
        } else
          o = deg[l].p;
      } else {
        if(dcmp(argv - deg[r].d) == 0) {
          if(norm(deg[r].p - p) < norm(deg[r].q - p))
            o = deg[r].p;
          else
            o = deg[r].q;
        } else
          o = deg[r].p;
      }
      return true;
    }
    int half = deg.size() - (r - l), size = deg.size();
    int lo, hi, mid, sig;
    bool found;
    lo = l, hi = r; found = false;
    while(lo + 1 < hi) {
      mid = lo + hi >> 1;
      sig = dcmp(v ^ (deg[mid].p - q));
      if(sig == 0) {
        o = deg[mid].p;
        found = true;
        break;
      }
      if(sig == sl) lo = mid;
      else hi = mid;
    }
    if(!found) o = deg[lo].cross(p, q);
    CPX tmp_o; int id;
    lo = 0, hi = half; found = false;
    while(lo + 1 < hi) {
      mid = lo + hi >> 1;
      id = (mid + r) % size;
      sig = dcmp(v ^ (deg[id].p - q));
      if(sig == 0) {
        tmp_o = deg[id].p;
        found = true;
        break;
      }
      if(sig == sr) lo = mid;
      else hi = mid;
    }
    if(!found) tmp_o = deg[(r + lo) % size].cross(p, q);
    if(norm(tmp_o - p) < norm(o - p)) o = tmp_o;
    return true;
  }
} t_poly;
```

Smallest Ball

```
const double eps = 1e-10;
struct point_type { double x, y, z; };

int npoint, nouter;
point_type point[10000], outer[4],res;
double radius,tmp;

inline double dist(point_type p1, point_type p2) {
  double dx=p1.x-p2.x, dy=p1.y-p2.y, dz=p1.z-p2.z;
  return ( dx*dx + dy*dy + dz*dz );
}

inline double dot(point_type p1, point_type p2)
{ return p1.x*p2.x + p1.y*p2.y + p1.z*p2.z; }

void minball(int n) {
  ball();
  if( nouter<4 )
    for(int i=0; i<n; ++i)
      if( dist(res, point[i])-radius>eps ) {
        outer[nouter]=point[i];
        ++nouter;
        minball(i);
        --nouter;
        if( i>0 ) {
          point_type Tt = point[i];
          memmove(&point[1], &point[0],
                  sizeof(point_type)*i);
          point[0]=Tt;
        }
      }
}
```

```
void ball() {
  point_type q[3];
  double m[3][3], sol[3], L[3], det; int i,j;
  res.x = res.y = res.z = radius = 0;
  switch ( nouter ) {
    case 1: res=outer[0]; break;
    case 2:
      res.x=(outer[0].x+outer[1].x)/2;
      res.y=(outer[0].y+outer[1].y)/2;
      res.z=(outer[0].z+outer[1].z)/2;
      radius=dist(res, outer[0]);
      break;
    case 3:
      for(i=0; i<2; ++i ) {
        q[i].x=outer[i+1].x-outer[0].x;
        q[i].y=outer[i+1].y-outer[0].y;
        q[i].z=outer[i+1].z-outer[0].z;
      }
      for(i=0; i<2; ++i ) for(j=0; j<2; ++j )
        m[i][j]=dot(q[i], q[j])*2;
      for(i=0; i<2; ++i ) sol[i]=dot(q[i], q[i]);
      if( fabs( det=m[0][0]*m[1][1]
                   -m[0][1]*m[1][0])<eps ) return;

      L[0]=(sol[0]*m[1][1]-sol[1]*m[0][1])/det;
      L[1]=(sol[1]*m[0][0]-sol[0]*m[1][0])/det;

      res.x=outer[0].x+q[0].x*L[0]+q[1].x*L[1];
      res.y=outer[0].y+q[0].y*L[0]+q[1].y*L[1];
      res.z=outer[0].z+q[0].z*L[0]+q[1].z*L[1];
      radius=dist(res, outer[0]);
      break;
    case 4:
      for(i=0; i<3; ++i){
        q[i].x=outer[i+1].x-outer[0].x;
        q[i].y=outer[i+1].y-outer[0].y;
        q[i].z=outer[i+1].z-outer[0].z;
        sol[i]=dot(q[i], q[i]);
      }
      for(i=0;i<3;++i)
        for(j=0;j<3;++j) m[i][j]=dot(q[i],q[j])*2;
      det=m[0][0]*m[1][1]*m[2][2] + m[0][1]*m[1][2]*m[2][0]
        + m[0][2]*m[2][1]*m[1][0] - m[0][2]*m[1][1]*m[2][0]
        - m[0][1]*m[1][0]*m[2][2] - m[0][0]*m[1][2]*m[2][1];

      if( fabs(det)<eps ) return;

      for(j=0; j<3; ++j){
        for(i=0; i<3; ++i) m[i][j]=sol[i];
        L[j]=(m[0][0]*m[1][1]*m[2][2] + m[0][1]*m[1][2]*m[2][0]
            + m[0][2]*m[2][1]*m[1][0] - m[0][2]*m[1][1]*m[2][0]
            - m[0][1]*m[1][0]*m[2][2] - m[0][0]*m[1][2]*m[2][1]
           )/ det;
        for(i=0; i<3; ++i) m[i][j]=dot(q[i], q[j])*2;
      }
      res=outer[0];
      for(i=0; i<3; ++i ) {
        res.x += q[i].x * L[i];
        res.y += q[i].y * L[i];
        res.z += q[i].z * L[i];
      }
      radius=dist(res, outer[0]);
  }
}
```

Half Plane Intersection

```
const double inf = 1e9, eps = 1e-9, pi = acos(-1.0);

int sg(double x) {return abs(x)<eps?0:(x>0?1:-1);}

struct Point {
  Point(): x(0), y(0) {}
  Point(double _x, double _y):
    x(_x), y(_y) {}
  double x, y;
};

double cs(Point a, Point b, Point c)
{ return (b.x-a.x)*(c.y-a.y)-(c.x-a.x)*(b.y-a.y); }

Point cs(Point a, Point b, Point c, Point d)
{
  double c1 = cs(a, b, c);
  double c2 = cs(a, b, d);
  double x = (d.x * c1 - c.x * c2) / (c1 - c2);
  double y = (d.y * c1 - c.y * c2) / (c1 - c2);
  return Point(x, y);
}

struct Line {
  //ax + by + c >= 0
  Line(double a, double b, double c) {
    if (sg(b) == 0) {
      p1 = Point(-c / a, inf);
      p2 = Point(-c / a, -inf);
      if (a < 0) swap(p1, p2);
    }else {
      p1 = Point(-inf, (-c + inf * a) / b);
      p2 = Point(inf, (-c - inf * a) / b);
      if (b < 0) swap(p1, p2);
    }
    k = atan2(p2.y - p1.y, p2.x - p1.x);
  }
  Point p1, p2;
  double k;
};

bool operator<(Line a, Line b) {return a.k < b.k;}

int main()
{
  int n; cin >> n;
  vector<Line> l, ll;
  for (int i = 1; i <= n; ++i) {
    double a, b, c;
    cin >> a >> b >> c;
    l.push_back(Line(a, b, c));
  }
  sort(l.begin(), l.end());
  for (int i = 0; i < l.size(); ++i)
    if (i == 0 || sg(l[i].k - l[i - 1].k) > 0)
      ll.push_back(l[i]);
    else
      if (sg(cs(l[i].p1,l[i].p2,ll.back().p2))>=0)
        ll.back() = l[i];
  l.swap(ll);
  vector<Point> z;
  z.push_back(l[0].p1);
  z.push_back(l[0].p2);
  for (int i = 1; i < l.size(); ++i) {
    Point last;
    while (z.size()&&sg(cs(l[i].p1,l[i].p2,z.back()))<=0)
    { last = z.back(); z.pop_back(); }
    z.push_back(cs(l[i].p1, l[i].p2, z.back(), last));
    z.push_back(l[i].p2);
  }
  int i = 1, j = z.size() - 2, ii, jj;
  do {
    ii = i; jj = j;
    while (sg(cs(z[jj], z[jj + 1], z[i])) < 0) ++i;
    while (sg(cs(z[ii - 1], z[ii], z[j])) < 0) --j;
  }while (i != ii || j != jj);
  z[++j] = cs(z[i - 1], z[i], z[j - 1], z[j]);
  double ret = -inf, P, Q;
  cin >> P >> Q;
  for (int k = i; k <= j; ++k)
    if (P * z[k].x + Q * z[k].y > ret)
      ret = P * z[k].x + Q * z[k].y;
  cout << ret << endl;
}
```

3D Convex Hull

```
const double eps = 1e-8;
int dcmp(double x) { return fabs(x)<=eps?0:(x<0?-1:1); }

struct Point3 { double x, y, z; };
typedef Point3 Vector3;

// Define Operator +,-,*,/,==

double Dot(const Vector3& A, const Vector3& B)
{ return A.x*B.x + A.y*B.y + A.z*B.z; }
double Length(const Vector3& A)
{ return sqrt(Dot(A, A)); }
double Angle(const Vector3& A, const Vector3& B)
{ return acos(Dot(A, B) / Length(A) / Length(B)); }
Vector3 Cross(const Vector3& A, const Vector3& B) { return
  Vector3(A.y*B.z-A.z*B.y,A.z*B.x-A.x*B.z,A.x*B.y-A.y*B.x); }
double Area2(Point3 A, Point3 B, Point3 C)
{ return Length(Cross(B-A, C-A)); }
double Volume6(Point3 A, Point3 B, Point3 C, Point3 D)
{ return Dot(D-A, Cross(B-A, C-A)); }

double rand01() { return rand() / (double)RAND_MAX; }
double randeps() { return (rand01() - 0.5) * eps; }
Point3 add_noise(const Point3& p)
 { return Point3(p.x+randeps(),
                 p.y+randeps(), p.z+randeps()); }

struct Face {
  int v[3];
  Face(int a, int b, int c)
  { v[0] = a; v[1] = b; v[2] = c; }
  Vector3 Normal(const vector<Point3>& P) const
  { return Cross(P[v[1]]-P[v[0]], P[v[2]]-P[v[0]]); }
  int CanSee(const vector<Point3>& P, int i) const {
    return Dot(P[i]-P[v[0]], Normal(P)) > 0;
  } // whether f can see P[i]
};
// Incremental Algorithm for Convex Hull
// Assume no 4 points coplanar. Need add_noise otherwise
vector<Face> CH3D(const vector<Point3>& P) {
  int n = P.size();
  vector<vector<int> > vis(n);
  for(int i = 0; i < n; i++) vis[i].resize(n);
  vector<Face> cur;  //assume not coplanar
  cur.push_back(Face(0, 1, 2));
  cur.push_back(Face(2, 1, 0));
  for(int i = 3; i < n; i++) {
    vector<Face> next;
    for(int j = 0; j < cur.size(); j++) {
      Face& f = cur[j]; int res = f.CanSee(P, i);
      if(!res) next.push_back(f);
      for(int k = 0; k < 3; k++)
        vis[f.v[k]][f.v[(k+1)%3]] = res;
    } // compute whether it can be seen on the left side
    for(int j = 0; j < cur.size(); j++)
      for(int k = 0; k < 3; k++) {
        int a = cur[j].v[k], b = cur[j].v[(k+1)%3];
        if(vis[a][b] != vis[b][a] && vis[a][b])
          next.push_back(Face(a, b, i));
      } //(a,b) is the separating line
    cur = next;
  } return cur;
}
```

Mathematics

Chinese Remaining

$extended\_euclid(a, b) = ax + by$

```
int extended_euclid(int a,int b,int &x,int &y) {
  if(b==0){ x=1,y=0; return a; } else {
    int res=extended_euclid(b,a%b,x,y);
    int t=x; x=y; y=t-(a/b)*y;
    return res;
  }
}
```

$ax \equiv b \pmod{n}$, $n > 0$

```
void modular_linear_equation_solver(int a,int b,int n) {
  int d,x,y,e,i;
  d=extended_euclid(a,n,x,y);
  if(b%d!=0) cout<<"No answer!"; else {
    e=x*(b/d)%n;  // x=e is a basic solution
    for(i=0;i<d;i++) cout<<(e+i*(n/d))%n<<endl;
  }
}
```

Given b_i , w_i , $i = 0 \cdots len-1$ which $w_i > 0$, $i = 0 \cdots len-1$ and $(w_i, w_j) = 1$, $i \neq j$
Find an x which satisfies: $x \equiv b_i \pmod{w_i}$, $i = 0 \cdots len-1$

```
int china(int b[],int w[],int len) {
  int i,d,x,y,x,m,n;
  x=0; n=1; for(i=0;i<len;i++) n*=w[i];
  for(i=0;i<len;i++){
    m=n/w[i];
    d=extended_euclid(w[i],m,x,y);
    x=(x+y*m*b[i])%n;
  }
  return (n+x%n)%n;
}
```

FFT

```
const double PI = acos(-1);
void FFT (cp* a, int n, int t) {
    if(n == 1) return;
    cp *x = new cp[n>>1], *y = new cp[n>>1];
    for(int i = 0; i < n ; ++i) if (i&1) y[i>>1] = a[i]; else x[i>>1] = a[i];
    FFT(x, n>>1, t); FFT(y, n>>1, t);
    double arc = 2 * PI * t / n; cp wn(cos(arc),sin(arc)),w = 1;
    for(int i = 0; i < (n>>1) ; ++i) {
        a[i] = x[i] + w * y[i]; a[i + (n>>1)] = x[i] - w * y[i]; w = w * wn;
    }
    delete[] x; delete[] y;
}

void polymult(cp *X, cp *Y, cp *C, int n) { // n must be power of 2
    FFT(X, n, 1); FFT(Y, n, 1);
    for (int i = 0 ; i < n ; i ++) C[i] = X[i] * Y[i];
    FFT(C, n, -1);
}
```

Romberg

```
double F(double h) {
  // the function to integrate
}

double romberg(double a, double b, double eps) {
  vector <double> R;
  int k = -1;
  double r = 0.5 * (b - a) * (F(a) + F(b));
  R.push_back(r);
  do {
    k += 1;
    r = 0.0;
    for(int i = 0; i < pow(2., k); i ++)
      r += F(a + (b - a) * (i + 0.5) / pow(2., k));
    r *= (b - a) / pow(2., k + 1);
    r += 0.5 * R[k];
    R.push_back(r);
    for(int m = 0; m <= k; m ++)
      R[k - m] =
       (pow(4.,m+1)*R[k+1-m]-R[k-m])/(pow(4.,m+1)-1);
  } while (fabs(R[0] - R[1]) > eps);
  return R[0];
}
```

Linear Programming – Simplex

Primal Simplex Method for solving Linear Programming problem in Standard Form

maximize

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n = \texttt{ans}$$

subject to

$$\begin{aligned} a_{1,1}\,x_1 + a_{1,2}\,x_2 + \cdots + a_{1,n}\,x_n &\leq rhs_1 \\ a_{2,1}\,x_1 + a_{2,2}\,x_2 + \cdots + a_{2,n}\,x_n &\leq rhs_2 \\ &\vdots \\ a_{m,1}\,x_1 + a_{m,2}\,x_2 + \cdots + a_{m,n}\,x_n &\leq rhs_m \end{aligned}$$

```
const double eps = 1e-8;
const double inf = 1e15;

#define OPTIMAL -1
#define UNBOUNDED -2
#define FEASIBLE -3
#define INFEASIBLE -4
#define PIVOT_OK 1

int basic[maxn],row[maxm],col[maxn];
double c0[maxn];

double dcmp(double x){return (x<-eps?-1:(x>eps?1:0));}
int Pivot(int n, int m, double *c,double a[maxn][maxn],
    double *rhs,int &i,int &j)
{
  double min = inf; int k = -1;
  for(j=0; j<=n; j++) if( !basic[j] && dcmp(c[j])>0 )
    if( k<0 || dcmp(c[j]-c[k])>0 ) k=j;
  j=k; if( k < 0 ) return OPTIMAL;
  for(k=-1,i=1; i<=m; i++) if( dcmp(a[i][j])>0 )
    if( dcmp(rhs[i]/a[i][j]-min) < 0 )
    { min = rhs[i]/a[i][j]; k=i; }
  i=k; if( k < 0 ) return UNBOUNDED; else return PIVOT_OK;
}

int PhaseII(int n, int m, double *c, double a[maxn][maxn],
    double *rhs, double &ans, int PivotIndex)
{
  int i,j,k,l; double tmp;
  while(k=Pivot(n,m,c,a,rhs,i,j),k==PIVOT_OK || PivotIndex)
  {
    if( PivotIndex ) { j=0; i=PivotIndex; PivotIndex=0; }
    basic[row[i]]=col[row[i]]=0;basic[j]=1;
    col[j]=i;row[i]=j; tmp=a[i][j];
    for(k=0;k<=n;k++) a[i][k]/=tmp; rhs[i]/=tmp;
    for(k=1;k<=m;k++) if(k!=i && dcmp(a[k][j])) {
      tmp = -a[k][j]; for(l=0;l<=n;l++) a[k][l]+=tmp*a[i][l];
      rhs[k] += tmp*rhs[i];
    }
    tmp=-c[j]; for(l=0;l<=n;l++) c[l]+=a[i][l]*tmp;
    ans-=tmp*rhs[i];
  } return k;
}

int PhaseI(int n,int m,double *c,double a[maxn][maxn],
    double *rhs,double &ans)
{
  int i,j,k = -1; double tmp, min = 0, ans0 = 0;
  for(i=1; i<=m; i++) if(dcmp(rhs[i]-min)<0){min=rhs[i];k=i;}
  if( k<0 ) return FEASIBLE;
  for(i=1; i<=m; i++) a[i][0] = -1;
  for(j=1; j<=n; j++) c0[j]=0;  c0[0] = -1;
  PhaseII(n, m, c0, a, rhs, ans0, k);
  if( dcmp(ans0)<0 ) return INFEASIBLE;
  for(i=1; i<=m; i++) a[i][0] = 0;
  for(j=1; j<=n; j++) if( dcmp(c[j]) &&  basic[j] ) {
    tmp = c[j]; ans += rhs[col[j]]*tmp;
    for(i=0; i<=n; i++) c[i] -= tmp*a[col[j]][i];
  }
  return FEASIBLE;
}

int simplex(int n, int m, double *c, double a[maxn][maxn],
    double *rhs, double &ans, double *x) // standard form
{
  int i,j,k;
  for(i=1; i<=m; i++) {
    for(j=n+1; j<=n+m; j++) a[i][j]=0;
    a[i][n+i] = 1; a[i][0] = 0;
    row[i] = n+i; col[n+i] = i;
  }
  k = PhaseI (n+m, m, c, a, rhs, ans);
  if( k == INFEASIBLE ) return k;
  k = PhaseII(n+m, m, c, a, rhs, ans, 0);
  for(j=0;j<=n+m;j++) x[j]=0;
  for(i=1;i<=m;i++) x[row[i]]=rhs[i];
  return k;
}
int n, m;
double c[maxn],ans, a[maxm][maxn], rhs[maxm], x[maxn];
int main()
{
  int i,j;
  while( cin>>n>>m && !cin.fail() )
  {
    for(j=1; j<=n; j++) cin>>c[j]; cin>>ans; c[0]=0;
    for(i=1; i<=m; i++){
      for(j=1; j<=n; j++) cin>>a[i][j]; cin>>rhs[i]; }
    switch( simplex(n, m, c, a, rhs, ans, x) ) {
      case OPTIMAL :
        printf("OPTIMAL\n%10lf\n",ans);
        for(j=1;j<=n;j++)printf("x[%2d]=%lf\n",j,x[j]);
        break;
      case UNBOUNDED :
        printf("UNBOUNDED\n");  break;
      case INFEASIBLE :
        printf("INFEASIBLE\n"); break;
    }   printf("\n");
  } return 0;
}
```

Roots of Cubic and Quartic

$$c_0 + c_1 * x + c_2 * x^2 + c_3 * x^3 + c_4 * x^4 = 0$$

The functions return the number of distinct non-complex roots and put the values into the s array.

```
const double pi = acos(-1.0);

double cbrt(double x) {
  if( x> eps ) return  pow(  x, 1/3.0);
  if( x<-eps ) return -pow( -x, 1/3.0);
  return 0;
}

int SolveQuadric(double c[3], double s[2]) {
  double p, q, d; // normal form: x^2 + px + q = 0
  p = c[1]/(2*c[2]); q = c[0]/c[2]; d = p*p-q;
  if( dcmp(d)==0 ) { s[0] = - p; return 1; }
  if( dcmp(d) < 0 ) return 0;
  d = sqrt( d );
  s[0] = - p + d; s[1] = - p - d;
  return 2;
}

int SolveCubic(double c[4], double s[3])
{
  int i, num; // normal form: x^3 + Ax^2 + Bx + C = 0
  double sub, A, B, C, sqa, p, q, cbp, d;
  A = c[2]/c[3]; B = c[1]/c[3]; C = c[0]/c[3];
  sqa = A * A; //  x = y - A/3  =>  x^3 +px + q = 0
  p = 1.0/3 * (- 1.0/3 * sqa + B);
  q = 1.0/2 * (2.0/27 * A * sqa - 1.0/3 * A * B + C);
  cbp = p * p * p;  // use Cardano's formula
  d = q * q + cbp;
  if( dcmp(d)==0 ) {
    if( dcmp(q)==0 ) { s[0]=0; num=1; } // one triple
    else { // one single and one double solution
      double u = cbrt( -q );
      s[0] = 2 * u;  s[1] = - u;  num = 2;
    }
  } else if( dcmp(d)<0 ) {
      // Casus irreducibilis: three real solutions
      double phi = 1.0/3 * acos(-q / sqrt(-cbp));
      double t = 2 * sqrt(-p);
      s[ 0 ] =   t * cos(phi);
      s[ 1 ] = - t * cos(phi + pi / 3);
      s[ 2 ] = - t * cos(phi - pi / 3);
      num = 3;
  } else { /* one real solution */
      d=sqrt(d); double u=cbrt(d-q), v=-cbrt(d+q);
      s[ 0 ] = u + v; num = 1;
  }
  /* resubstitute */
  sub = 1.0/3 * A;  for( i=0; i<num; ++i) s[i] -= sub;
  return num;
}

int SolveQuartic(double c[5], double s[4])
{
  double e[4], z, u, v, sub, A, B, C, d, sqa, p, q, r;
  int i, num; // x^4 + Ax^3 + Bx^2 + Cx + D = 0
  A=c[3]/c[4]; B=c[2]/c[4]; C=c[1]/c[4]; d=c[0]/c[4];
  sqa = A * A; // x=y-A/4 => x^4+px^2+qx+r=0
  p = - 3.0/8 * sqa + B;
  q = 1.0/8 * sqa * A - 1.0/2 * A * B + C;
  r = - 3.0/256*sqa*sqa + 1.0/16*sqa*B - 1.0/4*A*C + d;
  if( dcmp(r)==0 ) { //no absolute term: y(y^3+py+q)=0
      e[0] = q;  e[1] = p;  e[2] = 0;  e[3] = 1;
      num = SolveCubic(e, s);  s[ num++ ] = 0;
  } else { // solve the resolvent cubic ...
      e[0] = 1.0/2 * r * p - 1.0/8 * q * q;  e[1] = - r;
      e[2] = - 1.0/2 * p; e[3] = 1;
      SolveCubic(e, s);
      z = s[ 0 ]; // ... and take the one real solution
      u = z*z-r;  v = 2*z-p; // .. to build two quadric eqs
      if(dcmp(u)==0) u=0; else
        if(dcmp(u)>0) u=sqrt(u); else return 0;
      if(dcmp(v)==0) v=0; else
        if(dcmp(v)>0) v=sqrt(v); else return 0;
      e[0] = z-u;  e[1] = dcmp(q)<0 ? -v : v;  e[2] = 1;
      num = SolveQuadric(e, s);
      e[0] = z+u;  e[1] = dcmp(q)<0 ? v : -v;  e[2] = 1;
      num += SolveQuadric(e, s + num);
  }// resubstitute
  sub = 1.0/4*A;  for( i=0; i<num; ++i) s[i] -= sub;
  return num;
}
```

Data Structure

Sudoku DancingLinks

```
const int Size = 1000000;

int up[Size],dw[Size],lt[Size],rt[Size];
int col[Size],repx[Size],repy[Size],repnum[Size],cnt[Size];
int N = 9, n = 3;

int board[10][10];
int wall[10][10];
int idx[10][10], idx_no;
int tot, ans;

int build() {
  //////////////////////////////////////////
  Clear lt,rt,up,dw,col,repx,repy,repnum,cnt;
  //////////////////////////////////////////
  Build: Every column denotes a position,
       every row denotes an approach
  //////////////////////////////////////////
}

int Cover(int c) {
  rt[lt[c]] = rt[c];
  lt[rt[c]] = lt[c];
  for(int i=dw[c];i!=c;i=dw[i]) {
    for(int j=rt[i];j!=i;j=rt[j]) {
      --cnt[col[j]];
      dw[up[j]] = dw[j];
      up[dw[j]] = up[j];
    }
  }
}

int Recover(int c) {
  for(int i=up[c];i!=c;i=up[i])
    for(int j=lt[i];j!=i;j=lt[j]){
      up[dw[j]] = j;
      dw[up[j]] = j;
      ++ cnt[col[j]];
    }
  lt[rt[c]] = c; rt[lt[c]] = c;
}

bool found;
int rem[10], ptrem[20];
int out[10][10], record[10][10];

int dfs(int dep) {
  if(lt[0] == 0) {
    // Found Answer recorded in out
    found = true;
    return 0;
  }
  int c = -1;
  for(int i=rt[0];i!=0;i=rt[i])
    if(c < 0 || cnt[i] < cnt[c]) c=i;

  Cover(c);
  for(int i=dw[c];i!=c;i=dw[i]) {
    int x = repx[i],y = repy[i],dig=repnum[i];

    out[x][y]=dig;

    for(int j=rt[i];j!=i;j=rt[j]) Cover(col[j]);
    if(dfs(dep+1) < 0) return -1;
    for(int j=lt[i];j!=i;j=lt[j]) Recover(col[j]);
  }
  Recover(c);
  return 0;
}
```

Extended KMP

```
struct extKMP {
  string S; int n,A[MaxN],nxt[MaxN];
  int set(string _t){S="#"+_t; n=_t.size();}
  int buildNxt() {
    fill(nxt,nxt+1+n,0);
    for (int i=2,k=0;i<=n;i++) {
      for (; k>0 && S[k+1] != S[i]; k = nxt[k]);
      nxt[i] = (S[k+1] == S[i]? ++k : k);
    }
  }
  vector<int> patMatch(string P) {
    int m=P.size();P="#"+P;vector<int> pos;
    for (int i=1,j=0;i<=m;i++) {
      for (; j>0 && P[i]!=S[j+1]; j=nxt[j]);
      if (P[i] == S[j+1]) j++;
      if (j == n)
       { pos.push_back(i-n+1); j=nxt[j];}
    }return pos;
  }
  int buildA() {
    fill(A,A+n+1,0);int j=0;
    for(;2+j<=n&&S[j+1]==S[j+2];++j);
    A[1]=n;A[2]=j;
    for(int i=3,k=2;i<=n;++i){
      int len=k+A[k]-1,L=A[i-k+1];
      if(L<len-i+1)A[i]=L;
      else { k=i; for(j=max(0,len-i+1);i+j<=n
             &&S[1+j]==S[i+j];++j); A[i]=j; }
    }
  }
  vector<int>patCount(string P) {
    int m=P.size();P="#"+P;
    vector<int>res(m+1,0); int j=0;
    for(;j<n&&j<m&&S[1+j]==P[1+j];++j); res[1]=j;
    for(int k=1,i=2;i<=m;++i) {
      int len=k+res[k]-1,L=A[i-k+1];
      if(L<len-i+1)res[i]=L;
      else { k=i; for(j=max(0,len-i+1);
             S[1+j]==P[i+j];++j);res[i]=j; }
    } return res;
  }
};
```

Suffix Array

```
// MaxLen is TWICE longer than actual length
// string Stored in S[MaxLen]
const int MaxLog = 21;
int lg[MaxLen], tmp[2000];
struct SuffixArray
{
int rank[MaxLen], SA[MaxLen], h[MaxLen], D[MaxLen];
int n, dep, count_rank[MaxLen],f[MaxLog][MaxLen];
void Build()
{
  for(int len = 1; len < n; len <<= 1)
  {
    fill(count_rank, count_rank + 1 + n, 0);
    for(int i=1;i<=n;++i)
      ++ count_rank[rank[SA[i]+len]];
    for(int i=1;i<=n;++i)
      count_rank[i]+=count_rank[i-1];
    for(int i=n;i>0;--i)
      D[count_rank[rank[SA[i]+len]]--] = SA[i];
    fill(count_rank, count_rank + 1 + n, 0);
    for(int i=1;i<=n;++i)
      ++ count_rank[rank[SA[i]]];
    for(int i=1;i<=n;++i)
      count_rank[i]+=count_rank[i-1];
    for(int i=n;i>0;--i)
      SA[count_rank[rank[D[i]]]--] = D[i];
    copy(rank, rank + 1 + n, D);
    rank[SA[1]]=1;
    for(int i=2;i<=n;++i)
      if(D[SA[i]] != D[SA[i-1]] ||
        D[SA[i]+len] != D[SA[i-1] + len])
        rank[SA[i]]=rank[SA[i-1]]+1;
      else
        rank[SA[i]]=rank[SA[i-1]];
    if(rank[SA[n]] == n) break;
  }
}

int strsuf(int *p, int *q)
{
  int ret=0;
  for(; *p == *q; ++p, ++q, ++ ret);
  return ret;
}

void CalcHeight()
{
  for(int i=1;i<=n;++i)
  {
    if(rank[i] == 1)
      h[i] = 0;
    else
    if(i == 1 || h[i-1] <= 1)
      h[i]=strsuf(S+i, S+SA[rank[i]-1]);
    else
      h[i]=strsuf(S+i+h[i-1]-1,
          S+SA[rank[i]-1]+h[i-1]-1)+h[i-1]-1;
    f[0][rank[i]]=h[i];
  }
  dep=1;
  for(int len=1;len*2<=n;len<<=1,dep++)
    for(int i=1;i+len*2-1<=n;++i)
      f[dep][i]=min(f[dep-1][i],f[dep-1][i+len]);
}

void init(int _n) // String Stored in (S+1)
{
  n = _n;
  fill(rank,rank+2*n+2,0);
  memset(tmp,0,sizeof(tmp));
  for(int i=1;i<=n;++i)
    ++ tmp[S[i]];
  for(int i=1;i<2000;++i)tmp[i]+=tmp[i-1];
  for(int i=n;i>0;--i)
    SA[tmp[S[i]]--]=i;
  rank[SA[1]]=1;
  for(int i=2;i<=n;++i)
    if(S[SA[i]] != S[SA[i-1]])
      rank[SA[i]] = rank[SA[i-1]]+1;
    else
      rank[SA[i]] = rank[SA[i-1]];

  Build();
  CalcHeight();
}

inline int lcp(int a, int b)
{ // lcp of S[a] and S[b]
  if(a == b) return n - a + 1;
  a = rank[a], b = rank[b];
  if(a > b) swap(a, b);
  int d = lg[b - a];
  if((1 << d) == (b - a)) return f[d][a+1];
  else return min(f[d][a+1], f[d][b-(1<<d)+1]);
}};
```

Suffix Automata & Suffix Tree

```
// string = str[1....n], str[i] in [0, maxchar-1]
const int maxn=200100, maxchar=9;
int str[maxn];
struct State {
  State *trans[maxchar]; int mask;
  State *par; int dep, start, idx;
  State(){memset(trans,0,sizeof(trans)); mask=0;}
  void clear_trans() {
    for(int t = mask;t > 0; t&=t-1)
      trans[__builtin_ctz(t)] = 0;
    mask=0; }
  void clear() { par=0; start=dep=idx=0; clear_trans(); }
  void copy(State*s) {
    start=s->start; par=s->par;
    clear_trans(); mask=s->mask;
    for(int t=mask; t >0; t&=t-1) {
      int ch = __builtin_ctz(t);
      trans[ch] = s->trans[ch];
    }
  }
};

class SuffixTree { public:
int n;
State states[maxn*2], *new_state, *root, *whole;
int arr[maxn], m, hei[maxn]; // suffix array
void extend(int ch) {
  State *nwhole = new_state++;

  nwhole->clear();
  nwhole->dep = whole->dep+1;
  nwhole->idx = n - nwhole->dep + 1;

  State *cur = whole;
  while(cur && cur->trans[ch]==0) {
    cur->trans[ch] = nwhole; cur->mask|=1<<ch,
    cur = cur->par;
  }
  if(cur==0)
    nwhole->par = root;
  else {
    State *fork = cur->trans[ch];
    if(cur->dep+1 == fork->dep)
      nwhole->par = fork;
    else {
      State *nfork = new_state++;
      nfork->copy(fork);
      nfork->dep = cur->dep+1;
      nfork->start += fork->dep - (cur->dep+1);
      if(nfork->start) nfork->idx = -1;
      else nfork->idx = n - nfork->dep + 1;

      nwhole->par = fork->par = nfork;
      while(cur && cur->trans[ch] == fork) {
        cur->trans[ch] = nfork, cur->mask|=1<<ch,
        cur = cur->par; }
    } }
  whole = nwhole;
}
void sa() {
  new_state = states;  root = new_state++;
  // build automata
  root->clear(); whole = root;
  for(int i=1; i<=n; i++) extend(str[i]);
  // build suffix tree
  for(State *s=states; s<new_state; s++)
    s->clear_trans();
  for(State *s=states; s<new_state; s++)
    if(s->par) {
      int ch = str[s->start + s->dep - s->par->dep];
      State* p = s->par;
      p->trans[ch] = s; p->mask|=1<<ch;
    }
}
int go(State* s) {
  int ret = -1;
  if(s->idx > 0) { arr[++m] = s->idx; ret = m; }
  for(int t = s->mask; t >0; t&=t-1){
    int ch = __builtin_ctz(t);
    int u = go(s->trans[ch]);
    if(ret<0) ret=u; else hei[u]=s->dep;
  }
  return ret;
}
void build_suffix_array() {
  m = 0; hei[go(root)] = 0;// hei array
}
void init(int _n) {
  n = _n;  reverse(str+1,str+1+n);
  sa(); // suffix tree
  reverse(str+1,str+1+n);
  build_suffix_array(); // suffix array in arr[]
}};
```

DP for Monotonous Option

```
#include<deque>
int n, m, a[maxn], s[maxn], *gg;
int _f[maxn], _g[maxn];
struct T {int l, r, x; };

//cost of option l for dp[r]
int cost(int l, int r){};

int main() {
  cin >> n >> m;
  for (int i = 1; i <= n; ++i) cin >> a[i];
  for (int i = 1; i <= n; ++i) s[i]=s[i-1]+a[i];
  int *f = _f, *g = _g; gg = g; g[0] = 0;
  for (int i = 1; i <= n; ++i) g[i] = cost(0, i);
  for (int j = 2; j <= m; ++j) {
    memset(f + 1, 0, j * sizeof(int));
    int up = n - (m - j);
    gg = g; deque<T> q;
    q.push_back(T(j, up, j - 1));
    for (int i = j; i <= up; ++i) {
      while (q[0].r < i) q.pop_front();
      f[i] = cost(q[0].x, i);
      while (!q.empty()) {
        T &t = q.back();
        if (cost(t.x, t.l) <= cost(i, t.l)) {
          int lef = t.l, rig = t.r;
          while (lef < rig) {
            int mid = (lef + rig + 1) / 2;
            if (cost(t.x, mid) <= cost(i, mid))
              lef = mid;
            else
              rig = mid - 1;
          }
          t.r = lef; break;
        } else q.pop_back();
      }
      if (q.empty()) q.push_back(T(j, up, i));
      else if (q.back().r < up)
          q.push_back(T(q.back().r + 1, up, i));
    }
    swap(f, g);
  } cout << g[n] << endl;
}
```

Splay Tree

```
int lch[], rch[], fa[], rev[], tot, root;
int sum[], K[], L[], R[];
void update(int p) {
  sum[p] = K[p];
  if(lch[p]>-1)sum[p]+=sum[lch[p]];
  if(rch[p]>-1)sum[p]+=sum[rch[p]];
}
int zig(int p) {
  int q=fa[p],f=fa[q]; fa[p]=f;
  if(f > -1)
    if(lch[f] == q) lch[f] = p;
    else rch[f] = p;
  lch[q] = rch[p];
  if(rch[p] > -1) fa[rch[p]] = q;
  rch[p] = q; fa[q] = p;
  update(q); update(p);
}
int zag(int p) {
  int q=fa[p],f=fa[q]; fa[p]=f;
  if(f > -1)
    if(lch[f] == q) lch[f] = p;
    else rch[f] = p;
  rch[q] = lch[p];
  if(lch[p] > -1) fa[lch[p]] = q;
  lch[p] = q; fa[q] = p;
  update(q); update(p);
}
int arr[MaxE], sz;
void Reverse(int t) {// assert(rev[t]>0)
  swap(L[t],R[t]); swap(lch[t],rch[t]);
  if(lch[t]>-1) rev[lch[t]]^=1;
  if(rch[t]>-1) rev[rch[t]]^=1;
  rev[t]=0;
}
void CheckReverse(int p) {
  sz = 0; int t;
  for(; p != -1; p = fa[p])
    arr[sz ++] = p;
  for(int i=sz-1;i>=0;--i)
    if(rev[t=arr[i]]) Reverse(t);
}
int splay(int p, int top = -1) {
  CheckReverse(p);
  while(fa[p]!=top && fa[p]!=-1) {
    int q = fa[p];
    if(fa[q]==top||fa[q]==-1) {
      if(lch[q] == p) zig(p);
      else zag(p);
      break;
    }
    int f = fa[q];
    if(lch[f] == q) {
      if(lch[q] == p) zig(q);
      else zag(p);
      zig(p);
    } else {
      if(rch[q] == p) zag(q);
      else zig(p);
      zag(p);
    }
  } return p;
}
```

Weiqiao's Stuff

Weiqiao's Geometry

```
const double eps = 1e-10;
int dcmp(double x) {
  if(fabs(x) < eps) return 0; else return x < 0 ? -1 : 1;}
const double PI = acos(-1);
const double TWO_PI = PI * 2;
double NormalizeAngle(double rad, double center = PI) {
  return rad - TWO_PI * floor((rad + PI - center) / TWO_PI);}
struct Point {double x, y;
  Point(double x=0, double y=0):x(x),y(y) { } };
typedef Point Vector;
Vector operator + (const Vector& A,const Vector& B) {return Vector(A.x+B.x,A.y+B.y);}
Vector operator - (const Point& A,const Point& B) {return Vector(A.x-B.x, A.y-B.y);}
Vector operator * (const Vector& A, double p) {return Vector(A.x*p, A.y*p); }
Vector operator / (const Vector& A, double p) {return Vector(A.x/p, A.y/p); }
bool operator < (const Point& a, const Point& b) {
  return a.x < b.x || (a.x == b.x && a.y < b.y);}
bool operator == (const Point& a, const Point &b) {
  return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0;}
double angle(Vector v){//the angle of the vector (x,y), in arc [0,2M_PI]
  return atan2(v.y,v.x);}
double Dot(const Vector& A, const Vector& B) {return A.x*B.x + A.y*B.y; }
double Dist(const Point& A, const Point& B) {return sqrt((A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y));}
double Dist2(const Point& A, const Point& B) {
  return (A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y);}
double Cross(const Vector& A, const Vector& B) { return A.x*B.y - A.y*B.x;}
double Length(Vector A) {return sqrt(Dot(A, A)); }
double Angle(Vector A,Vector B){return acos(Dot(A,B)/Length(A)/Length(B));}
double Area2(Point A,Point B, Point C){return Cross(B-A,C-A);}
Vector Rotate(Vector A, double rad){
  return Vector(A.x*cos(rad)-A.y*sin(rad),A.x*sin(rad)+A.y*cos(rad));}
Vector Normal(const Vector& A) {double L = Length(A);
  return Vector(-A.y/L, A.x/L);}
***************** point and line (segment) *****************
double DistanceToLine(Point P, Point A, Point B) {
  Vector v1 = B - A, v2 = P - A;
  return fabs(Cross(v1, v2)) / Length(v1);}
double DistanceToSegment(Point P, Point A, Point B){
  //find the distance from point P to the line segment AB (tested on UVa10263 - Railway)
  if (A==B) return Length(P-A);
  Vector v1 = B-A, v2 = P-A, v3 = P-B;
  if(dcmp(Dot(v1,v2))<0) return Length(v2);
  else if(dcmp(Dot(v1,v3)) > 0) return Length(v3);
  else return fabs(Cross(v1,v2)) / Length(v1);}
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
  Vector u = P-Q;
  double t = Cross(w, u) / Cross(v, w);
  return P+v*t;}
Point GetLineProjection(Point P, Point A, Point B){
  //return the projection of P on the straight line AB (tested)
  Vector v = B-A;
  return A+v*(Dot(v,P-A)/Dot(v,v));}
Point DistanceToSegment_return_point(Point P, Point A, Point B){
  //find the point on line segment AB that has the min distance
  to point P (tested)
  if (A==B) return A;
  Vector v1 = B-A, v2 = P-A, v3 = P-B;
  if(dcmp(Dot(v1,v2))<0) return A;
  else if(dcmp(Dot(v1,v3)) > 0) return B;
  else return GetLineProjection(P,A,B);}
bool SegmentProperIntersection(const Point& a1, const Point& a2,
  const Point& b1, const Point& b2) {
  double c1 = Cross(a2-a1,b1-a1), c2 = Cross(a2-a1,b2-a1),
  c3 = Cross(b2-b1,a1-b1), c4=Cross(b2-b1,a2-b1);
  return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;}
bool OnSegment(const Point& p, const Point& a1, const Point& a2) {
  return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0;//use <= for second ineq if you want to include endpts}
***************** polygon *****************
typedef vector<Point> Polygon;
// if don't want the input points on the edges of convex hull, change two <= into <
// note: the set of input points will be changed.
vector<Point> ConvexHull(vector<Point>& p) {
  //preprocessing, delete duplicated points
  sort(p.begin(), p.end());
  p.erase(unique(p.begin(), p.end()), p.end());
  int n = p.size();
  int m = 0;
  vector<Point> ch(n+1);
  for(int i = 0; i < n; i++) {
    while(m > 1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
    ch[m++] = p[i];}
  int k = m;
  for(int i = n-2; i >= 0; i--) {
    while(m > k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
    ch[m++] = p[i];}
  if(n > 1) m--;
  ch.resize(m);
  return ch;}
double PolygonArea(vector<Point> p) {
  int n = p.size();
  double area = 0;
  for(int i = 1; i < n-1; i++)
    area += Cross(p[i]-p[0], p[i+1]-p[0]);
  return area/2;}
// return the square of diameter of the set of points
double diameter2(vector<Point>& points) {
  vector<Point> p = ConvexHull(points);
  int n = p.size();
  if(n == 1) return 0;
  if(n == 2) return Dist2(p[0], p[1]);
  p.push_back(p[0]);
  double ans = 0;
  for(int u = 0, v = 1; u < n; u++) {
    // one straight line tangent to p[u]-p[u+1]
    for(;;) {
      //when Area(p[u],p[u+1],p[v+1])<=Area(p[u],p[u+1],p[v]), stop rotating
      //i.e.,Cross(p[u+1]-p[u], p[v+1]-p[u])<=Cross(p[u+1]-p[u], p[v]-p[u])
      // since Cross(A,B) - Cross(A,C) = Cross(A,B-C)
      // we have Cross(p[u+1]-p[u], p[v+1]-p[v]) <= 0
      double diff = Cross(p[u+1]-p[u], p[v+1]-p[v]);
      if(diff <= 0) {
        ans = max(ans, Dist2(p[u], p[v])); // u and v are bounding points
        //(can draw two parallel lines through u and v bounding all points)
        if(diff == 0) ans = max(ans, Dist2(p[u], p[v+1])); // when diff == 0,
        //u and v+1 are bounding points
        break;}
      v = (v + 1) % n;}}
  return ans;}
```

```
int isPointInPolygon(const Point& p,const Polygon& poly){
  //binary search, including the case when point is on edges or vertices
  int n = poly.size();int l = 1, r = n;int m = (l+r) >> 1;
  if(Cross(poly[1]-poly[0],p-poly[0])<0 || Cross(poly[n-1]-poly[0],p-poly[0])>0) return 0;
  if(Cross(poly[1]-poly[0],p-poly[0]) == 0 && !OnSegment(p,poly[0],poly[1])) return 0;//use <= in OnSegment function to include the endpoints
  if(Cross(poly[n-1]-poly[0],p-poly[0]) == 0 && !OnSegment(p,poly[0],poly[n-1])) return 0;
  while(r-l>1){
    m = (l+r) >> 1;
    if(Cross(poly[m]-poly[0],p-poly[0])>=0){//check is p is left of (0,m)
      l = m;}
    else{r = m;}}
  if(Cross(poly[l+1]-poly[l],p-poly[l])<0)return 0;
  return 1;}
bool isLineInPolygon(const Point A,const Point B,const vector<Point> &P) {
    //check if line AB is in polygon P
    if (isPointInPolygon(A,P) == 0 || isPointInPolygon(B,P) == 0) return 0;
    int n = P.size();
    vector<Point> v;
    for (int i = 0; i < n; ++ i) {
        if (SegmentProperIntersection(A, B, P[i], P[(i+1)%n])) return 0;
        if (OnSegment(P[i],A, B)) v.push_back(P[i]);}
    sort(v.begin(), v.end());
    for (size_t i = 1; i < v.size(); ++ i) {
        Point O = (v[i] + v[i - 1]) / 2;
        if (isPointInPolygon(P, O) == 0) return 0;}
    return 1;}
bool isDiagonal(const Polygon& poly, int a, int b) {
    // is the line segment (poly[a], poly[b]) a diagonal of poly?
  int n = poly.size();
  for(int i = 0; i < n; i++)
    if(i != a && i != b && OnSegment(poly[i], poly[a], poly[b]))
    return false; //can't have other points in between
  for(int i = 0; i < n; i++)
    if(SegmentProperIntersection(poly[i], poly[(i+1)%n],
      poly[a], poly[b])) return false; //can't properly intersect with sides.
  Point midp = (poly[a] + poly[b]) * 0.5;
  return (isPointInPolygon(midp, poly) == 1);//whole segment inside polygon.}
***************** half plane intersection *****************
struct Line {
  //directed line, its left half is the half-plane we want
  Point p;
  Vector v;
  double ang;
  Line() {}
  Line(Point p, Vector v):p(p),v(v){ ang = atan2(v.y, v.x); }
  Point point(double t){return p + v*t;}
  bool operator < (const Line& L) const {
    return ang < L.ang;}};
bool OnLeft(const Line& L, const Point& p) {
  return Cross(L.v, p-L.p) > 0;}
Point GetLineIntersection(const Line& a, const Line& b) {
  Vector u = a.p-b.p;
  double t = Cross(b.v, u) / Cross(a.v, b.v);
  return a.p+a.v*t;}
vector<Point> HalfplaneIntersection(vector<Line> L) {
  int n = L.size();
  sort(L.begin(), L.end());
  int first, last;
  vector<Point> p(n);
  vector<Line> q(n);
  vector<Point> ans;
  q[first=last=0] = L[0];
  for(int i = 1; i < n; i++) {
    while(first < last && !OnLeft(L[i], p[last-1])) last--;
    while(first < last && !OnLeft(L[i], p[first])) first++;
    q[++last] = L[i];
    if(fabs(Cross(q[last].v, q[last-1].v)) < eps) {
      last--;
      if(OnLeft(q[last], L[i].P)) q[last] = L[i];}
    if(first < last) p[last-1] = GetLineIntersection(q[last-1], q[last]);}
  while(first < last && !OnLeft(q[first], p[last-1])) last--;
  if(last - first <= 1) return ans;
  p[last] = GetLineIntersection(q[last], q[first]);
  for(int i = first; i <= last; i++) ans.push_back(p[i]);
  return ans;}
****************** polygon and circle ******************
// if you know the length (a,b,c) of the three sides of a triangle,
// let p = (a+b+c)/2;
// the area of triangle = sqrt(p*(p-a)*(p-b)*(p-c)) := S
// the radius of its circumcircle is given by a*b*c/sqrt((a+b+c)*
//(b+c-a)*(c+a-b)*(a+b-c)) = a*b*c/(4*S)
// the radius of its inscribed circle is 2*S/(a+b+c)
/* circle */
struct Circle {
  Point c;
  double r;
  Circle(Point c, double r):c(c),r(r) {}
  Point point(double a) {
    return Point(c.x + cos(a)*r, c.y + sin(a)*r);}};
int getLineCircleIntersection(Line L, Circle C, double& t1,
  double& t2, vector<Point>& sol){
  double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
  double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
  double delta = f*f - 4*e*g;
  if(dcmp(delta) < 0) return 0;
  if(dcmp(delta) == 0) {
    t1 = t2 = -f / (2 * e); sol.push_back(L.point(t1));
    return 1;}
  t1 = (-f - sqrt(delta)) / (2 * e); sol.push_back(L.point(t1));
  t2 = (-f + sqrt(delta)) / (2 * e); sol.push_back(L.point(t2));
  return 2;}
Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
  double Bx = p2.x-p1.x, By = p2.y-p1.y;
  double Cx = p3.x-p1.x, Cy = p3.y-p1.y;
  double D = 2*(Bx*Cy-By*Cx);
  double cx = (Cy*(Bx*Bx+By*By) - By*(Cx*Cx+Cy*Cy))/D + p1.x;
  double cy = (Bx*(Cx*Cx+Cy*Cy) - Cx*(Bx*Bx+By*By))/D + p1.y;
  Point p = Point(cx, cy);
  return Circle(p, Length(p1-p));}
Circle InscribedCircle(Point p1, Point p2, Point p3) {
  double a = Length(p2-p3);
  double b = Length(p3-p1);
  double c = Length(p1-p2);
  Point p = (p1*a+p2*b+p3*c)/(a+b+c);
  return Circle(p, DistanceToLine(p, p1, p2));}
// the tangent line through Point p to Circle C
// v[i] is the i-th tangent's vector. Return # of tangents
int getTangents(Point p, Circle C, Vector* v) {
  Vector u = C.c - p;
  double dist = Length(u);
  if(dist < C.r) return 0;
```

```
  else if(dcmp(dist - C.r) == 0) { //  p        Ł     ł
    v[0] = Rotate(u, PI/2);
    return 1;
  } else {
    double ang = asin(C.r / dist);
    v[0] = Rotate(u, -ang);
    v[1] = Rotate(u, +ang);
    return 2;}}
// Common tangent line to two circles.
// Return the number of tangents. -1 means infinitely many.
// a[i], b[i] are the ith tangent point on Circle A and Circle B
int getTangents(Circle A,circle B, Point* a, Point* b){
  int cnt=0;
  if(A.r<B.r){swap(A,B);swap(a,b);}
  int d2 = (A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y);
  int rdiff = A.r-B.r;
  int rsum  = A.r+B.r;
  if(d2 < rdiff*rdiff) return 0;//B inside A
  double base = atan2(B.y-A.y,B.x-A.x);
  if(d2 == 0 && A.r==B.r) return -1; //same circle, inf # of tangents
  if(d2 == rdiff*rdiff){//internally tangent (inscribe)
    a[cnt] = A.getPoint(base); b[cnt] = B.getPoint(base); cnt++;
    return 1;}
  //there is common outer tangents from now on
  double ang = acos((A.r-B.r)/sqrt(d2));
  a[cnt] = A.getPoint(base+ang); b[cnt] = B.getPoint(base+ang); cnt++;
  a[cnt] = A.getPoint(base-ang); b[cnt] = B.getPoint(base-ang); cnt++;
  if(d2==rsum*rsum){//externally tangent, 1 common internal tangent
    a[cnt] = A.getPoint(base); b[cnt] = B.getPoint(PI+base); cnt++;}
  else if(d2 > rsum*rsum){//separate 2 internal tangents
    double ang = acos((A.r+B.r)/sqrt(d2));
    a[cnt] = A.getPoint(base+ang);b[cnt]=B.getPoint(PI+base+ang);cnt++;
    a[cnt] = A.getPoint(base-ang);b[cnt]=B.getPoint(PI+base-ang);cnt++;}
  return cnt;}
//find the minimum circle covering the polygon.(tested on UVa10005)

void min_cover_circle(vector<Point> p,Point &c,double &r) {
  int n = p.size();
  random_shuffle(p.begin(),p.end());
  c = p[0]; r = 0;
  int cnt = 0;
  for(int i=1; i<n; i++)
    if( Dist(p[i],c)>r )  {
      c = p[i]; r = 0;
      for(int k=0; k<i; k++)
        if( Dist(p[k],c)>r )  {
          c.x = (p[i].x + p[k].x)/2;
          c.y = (p[i].y + p[k].y)/2;
          r = Dist(p[k],c);
          for(int j=0; j<k; j++)
            if( Dist(p[j],c)>r )  {//find the center of circumcircle,
              //three points must not be on the same line
              Circle C = CircumscribedCircle(p[i],p[k],p[j]);
              c = C.c;
              r = C.r;}}}}
void getCircleCircleIntersection(Point c1, double r1, Point c2, double r2,
  vector<double>& rad) {
  double d = Length(c1 - c2);
  if(dcmp(d) == 0) return;
  if(dcmp(r1 + r2 - d) < 0) return;
  if(dcmp(fabs(r1-r2) - d) > 0) return;
  double a = angle(c2 - c1);
  double da = acos((r1*r1 + d*d - r2*r2) / (2*r1*d));
  rad.push_back(NormalizeAngle(a-da));
  rad.push_back(NormalizeAngle(a+da));}
struct Point3 {
  double x, y, z;
  Point3(double x=0, double y=0, double z=0):x(x),y(y),z(z) { }};
typedef Point3 Vector3;
Vector3 operator + (const Vector3& A, const Vector3& B) {return Vector3(
  A.x+B.x,A.y+B.y, A.z+B.z); }
Vector3 operator - (const Point3& A, const Point3& B) {return Vector3(
  A.x-B.x, A.y-B.y, A.z-B.z); }
Vector3 operator * (const Vector3& A, double p) {return Vector3(A.x*p,
  A.y*p, A.z*p); }
Vector3 operator / (const Vector3& A, double p) { eturn Vector3(A.x/p,
  A.y/p, A.z/p); }
bool operator == (const Point3& a, const Point3& b) {
  return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0 && dcmp(a.z-b.z) == 0;}
Point3 read_point3() {
  Point3 p;
  scanf("%lf%lf%lf", &p.x, &p.y, &p.z);
  return p;}
double Dot(const Vector3& A, const Vector3& B) { return A.x*B.x +
  A.y*B.y + A.z*B.z; }
double Length(const Vector3& A) { return sqrt(Dot(A, A)); }
double Angle(const Vector3& A, const Vector3& B) { return acos(Dot(A, B)
  / Length(A) / Length(B)); }
Vector3 Cross(const Vector3& A, const Vector3& B) { return Vector3(A.y*B.z
  - A.z*B.y, A.z*B.x - A.x*B.z, A.x*B.y - A.y*B.x); }
double Area2(const Point3& A, const Point3& B, const Point3& C) { return
  Length(Cross(B-A, C-A)); }
double Volume6(const Point3& A, const Point3& B, const Point3& C, const
  Point3& D) { return Dot(D-A, Cross(B-A, C-A)); }
Point3 Centroid(const Point3& A, const Point3& B, const Point3& C, const
  Point3& D) { return (A + B + C + D)/4.0; }
double rand01() { return rand() / (double)RAND_MAX; }
double randeps() { return (rand01() - 0.5) * eps; }
Point3 add_noise(const Point3& p) {
  return Point3(p.x + randeps(), p.y + randeps(), p.z + randeps());}
double DistanceToPlane(const Point3& p, const Point3& p0, const Vector3& n){
  return fabs(Dot(p-p0,n));//distance between p and plane p0-n, n unit vec}
Point3 GetPlaneProjection(const Point3& p,const Point3& p0,const Vector3& n){
  return p-n*(Dot(p-p0,n));//the projection of p onto plane p0-n, n unit vec}
int LinePlaneIntersection(Point3 p1,Point3 p2,Point3 p0,Vector3 n,Point3& q){
    Vector3 v = p2-p1;//line: p = p1+v*t, plane: Dot(n,p-p0)=0
    if(Dot(n,p2-p1)==0) return 0;//parallel or inside plane
    double t = (Dot(n,p0-p1) / Dot(n,p2-p1));
    q = p1+v*t;return 1;}
int LinePlaneIntersection(Point3 p1,Point3 p2,Point3 p0,Vector3 n,Point3& q){
    Vector3 v = p2-p1;//line: p = p1+v*t, plane: Dot(n,p-p0)=0
    if(Dot(n,p2-p1)==0) return 0;//parallel or inside plane
    double t = (Dot(n,p0-p1) / Dot(n,p2-p1));
    q = p1+v*t;return 1;}
//check if P is in Triangle P0P1P2
bool PointInTri(const Point3& P, const Point3& P0, const Point3& P1, const Point3& P2) {
  double area1 = Area2(P, P0, P1);
  double area2 = Area2(P, P1, P2);
  double area3 = Area2(P, P2, P0);
  return dcmp(area1 + area2 + area3 - Area2(P0, P1, P2)) == 0;}
//check if lineseg AB intersects with Tri P0P1P2
//doesn't consider the case when AB and Tri P0P1P2 are in same plane
bool TriSegIntersection(const Point3& P0, const Point3& P1, const Point3& P2, const Point3& A, const Point3& B, Point3& P) {
  Vector3 n = Cross(P1-P0, P2-P0);
  if(dcmp(Dot(n, B-A)) == 0) return false; // parallel or in same plane
  else {
    double t = Dot(n, P0-A) / Dot(n, B-A);
    if(dcmp(t) < 0 || dcmp(t-1) > 0) return false; // not on seg AB
    P = A + (B-A)*t; // compute intersection point
    return PointInTri(P, P0, P1, P2);//check if point is in Tri}}
bool TriTriIntersection(Point3* T1, Point3* T2) {
  Point3 P;
  for(int i = 0; i < 3; i++) {
    if(TriSegIntersection(T1[0], T1[1], T1[2], T2[i], T2[(i+1)%3], P)) return true;
    if(TriSegIntersection(T2[0], T2[1], T2[2], T1[i], T1[(i+1)%3], P)) return true;  }
  return false;}
//distance from P to line AB
double DistanceToLine(Point3 P,Point3 A,Point3 B){
    Vector3 v1 = B-A, v2 = P-A;
    return Length(Cross(v1,v2))/Length(v1);}
//distance from P to line seg AB
double DistanceToSegment(Point3 P,Point3 A,Point3 B){
    if(A==B) return Length(P-A);
    Vector3 v1 = B-A, v2 = P-A, v3 = P-B;
    if(dcmp(Dot(v1,v2)<0)) return Length(v2);
    else if(dcmp(Dot(v1,v3))>0) return Length(v3);
    else return Length(Cross(v1,v2)) / Length(v1);}
struct Face {
  int v[3];
  Face(int a, int b, int c) { v[0] = a; v[1] = b; v[2] = c; }
  Vector3 Normal(const vector<Point3>& P) const {
    return Cross(P[v[1]]-P[v[0]], P[v[2]]-P[v[0]]);}
  int CanSee(const vector<Point3>& P, int i) const {
    return Dot(P[i]-P[v[0]], Normal(P)) > 0;}};
vector<Face> CH3D(const vector<Point3>& P) {
  int n = P.size();
  vector<vector<int> > vis(n);
  for(int i = 0; i < n; i++) vis[i].resize(n);
  vector<Face> cur;
  cur.push_back(Face(0, 1, 2));
  cur.push_back(Face(2, 1, 0));
  for(int i = 3; i < n; i++) {
    vector<Face> next;
    for(int j = 0; j < cur.size(); j++) {
      Face& f = cur[j];
      int res = f.CanSee(P, i);
      if(!res) next.push_back(f);
      for(int k = 0; k < 3; k++) vis[f.v[k]][f.v[(k+1)%3]] = res;}
    for(int j = 0; j < cur.size(); j++)
      for(int k = 0; k < 3; k++) {
        int a = cur[j].v[k], b = cur[j].v[(k+1)%3];
        if(vis[a][b] != vis[b][a] && vis[a][b])
          next.push_back(Face(a, b, i));}
    cur = next;}
  return cur;}
struct ConvexPolyhedron {
  int n;
  vector<Point3> P, P2;
  vector<Face> faces;
  bool read() {
    if(scanf("%d", &n) != 1) return false;
    P.resize(n);
    P2.resize(n);
    for(int i = 0; i < n; i++) { P[i] = read_point3();
      P2[i] = add_noise(P[i]); }
    faces = CH3D(P2);
    return true;}
  Point3 centroid() {
    Point3 C = P[0];
    double totv = 0;
    Point3 tot(0,0,0);
    for(int i = 0; i < faces.size(); i++) {
      Point3 p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]],
        p3 = P[faces[i].v[2]];
      double v = -Volume6(p1, p2, p3, C);
      totv += v;
      tot = tot + Centroid(p1, p2, p3, C)*v;}
    return tot / totv;}
  double mindist(Point3 C) {
    double ans = 1e30;
    for(int i = 0; i < faces.size(); i++) {
      Point3 p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]],
        p3 = P[faces[i].v[2]];
      ans = min(ans, fabs(-Volume6(p1, p2, p3, C) / Area2(p1, p2, p3)));}
    return ans;}};
***************Zimpha's Triangulation***************
typedef double flt;
const flt eps = 1e-12, INF = 1e18, PI = acos(-1.0);
flt sqr(flt x) {return x * x;}
int sgn(flt x) {return x<-eps?-1:(x>eps);}
flt fix(flt x) {return sgn(x)==0?0:x;}
struct Point {
  flt x, y;
  Point(flt a=0, flt b=0) : x(a), y(b) {}
  bool operator < (const Point &r) const {
    return sgn(x-r.x)<0||(sgn(x-r.x)==0&&sgn(y-r.y)<0);}
  bool operator == (const Point &r) const {
    return sgn(x-r.x)==0&&sgn(y-r.y)==0;}
  Point operator *(const flt &k) const {return Point(x*k,y*k);}
  Point operator /(const flt &k) const {return Point(x/k,y/k);}
  Point operator -(const Point &r) const {return Point(x-r.x,y-r.y);}
  Point operator +(const Point &r) const {return Point(x+r.x,y+r.y);}
  flt dot(const Point &r) {return x*r.x+y*r.y;}
  flt det(const Point &r) {return x*r.y-y*r.x;}
  flt sqr() {return x*x+y*y;}
  flt abs() {return hypot(x, y);}
  Point rot() {return Point(-y,x);}
  Point rot(flt A) {return Point(x*cos(A)-y*sin(A),x*sin(A)+y*cos(A));}
  Point trunc(flt a=1.0) {return (*this)*(a/this->abs());}};
struct Line {
  Point a, b, v, p; // a->b
  flt ang;
  Line() {}
  Line(const Point &a, const Point &b): a(a), b(b) {
    ang = atan2(b.y - a.y, b.x - a.x);
    v = b - a;p = a;}
  Point point(flt t){
      return a + v*t;}
  bool operator < (const Line &l) const {
    int res = sgn(ang - l.ang);
    return res == 0 ? l.side(a) >= 0: res < 0;}
  int side(const Point &p) const {// 1: left, 0: on, -1:right
    return sgn((b - a).det(p - a));}
  Point inter(const Line &l) const {
    flt k = (l.a - l.b).det(a - l.b);
    k = k / (k - (l.a - l.b).det(b - l.b));
    return a + (b - a) * k;}};
bool onSeg(const Point &A, const Point &B, const Point &O) {
  return sgn((A-O).det(B-O)==0)&&sgn((A-O).dot(B-O))<=0;}
bool intersect(const Point &A, const Point &B, const Point &C, const Point &D
    , Point &res) {
  Point AB(B-A), CD(D-C);
  if (sgn(AB.det(CD))==0) return false; //
  int d1=sgn(AB.det(C-A))*sgn(AB.det(D-A));
  int d2=sgn(CD.det(A-C))*sgn(CD.det(B-C));
  res=A+(B-A)*((D-C).det(C-A)/(D-C).det(B-A));
  return d1<0&&d2<0;}
int inPolygon(vector<Point> &P, Point O) {
  int cnt=0, n = P.size();
  for (int i=0;i<n;++i) {
    if (onSeg(P[i],P[(i+1)%n],O)) return 2;
    int k=sgn((P[(i+1)%n]-P[i]).det(O-P[i]));
    int d1=sgn(P[i].y-O.y),d2=sgn(P[(i+1)%n].y-O.y);
    cnt+=(k>0&&d1<=0&&d2>0)-(k<0&&d2<=0&&d1>0);}
  return cnt!=0;}
bool inPolygon(vector<Point> &P, Point A, Point B) {
  if (inPolygon(P, A) == 0 || inPolygon(P, B) == 0) return 0;
  int n = P.size();
  vector<Point> v;
  for (int i = 0; i < n; ++ i) {
    Point tmp;
    if (intersect(A, B, P[i], P[(i+1)%n], tmp)) return 0;
    if (onSeg(A, B, P[i])) v.push_back(P[i]);}
  sort(v.begin(), v.end());
  for (size_t i = 1; i < v.size(); ++ i) {
    Point O = (v[i] + v[i - 1]) / 2;
    if (inPolygon(P, O) == 0) return 0;}
  return 1;}
bool halfplane(vector<Line> v) {
  sort(v.begin(), v.end());
  deque<Line> q; q.push_back(v[0]);
  deque<Point> ans;
  for (size_t i = 1; i < v.size(); ++ i) {
    if (sgn(v[i].ang - v[i - 1].ang) == 0) continue;
    while (ans.size() && v[i].side(ans.back()) < 0) ans.pop_back(), q.
        pop_back();
    while (ans.size() && v[i].side(ans.front()) < 0) ans.pop_front(), q.
        pop_front();
    ans.push_back(q.back().inter(v[i])); q.push_back(v[i]);}
  while (ans.size() && q.front().side(ans.back()) < 0) ans.pop_back(), q.
      pop_back();
  while (ans.size() && q.back().side(ans.front()) < 0) ans.pop_front(), q.
      pop_front();
  if (q.size() <= 2) return false;
  vector<Point> pt(ans.begin(), ans.end());
  pt.push_back(q.front().inter(q.back()));
  sort(pt.begin(), pt.end());
  pt.erase(unique(pt.begin(), pt.end()), pt.end());
  return pt.size() > 2;}
struct Triangle {
  Point a, b, c;
  Triangle() {}
  Triangle(const Point &_a, const Point &_b, const Point &_c): a(_a), b(_b),
      c(_c) {
    if (sgn((c - a).det(b - a)) > 0) swap(b, c);}
  vector<Line> toHalfplane() const {
    vector<Line> r;
    r.push_back(Line(a, b));
    r.push_back(Line(b, c));
    r.push_back(Line(c, a));
    return r;}};
vector<Triangle> getTriangle(vector<Point> pt) {
  vector<Triangle> ret;
  while (pt.size() > 2) {
    int n = pt.size();
    for (int i = 0; i < n; ++ i) {
      Point A = pt[(i-1+n)%n], B = pt[(i+1)%n];
      if (inPolygon(pt, A, B)) {
        ret.push_back(Triangle(A, B, pt[i]));
        /*cerr << "(" << A.x << "," << A.y << ") ";
        cerr << "(" << B.x << "," << B.y << ") ";
        cerr << "(" << pt[i].x << "," << pt[i].y << ") " << endl;*/
        pt.erase(pt.begin() + i);
        break;}}}
  return ret;}
// common area
int main() {
  for (int cas(1); scanf("%d", &n) == 1; ++ cas) {
    A.clear(); B.clear();
    for (int i = 0; i < n; ++ i) {
      int x, y; scanf("%d%d", &x, &y);
      A.push_back(Point(x, y));}
    scanf("%d", &m);
    for (int i = 0; i < m; ++ i) {
      int x, y; scanf("%d%d", &x, &y);
      B.push_back(Point(x, y));}
    vector<Triangle> TA = getTriangle(A);
    //cerr << endl;
    vector<Triangle> TB = getTriangle(B);
    bool flag = true;
    for (size_t i = 0; i < TA.size() && flag; ++ i) {
      for (size_t j = 0; j < TB.size() && flag; ++ j) {
        vector<Line> la = TA[i].toHalfplane();
        vector<Line> lb = TB[j].toHalfplane();
        for (auto &x: lb) la.push_back(x);
        //cerr << halfplane(la) << endl;
        if (halfplane(la)) flag = false;}}
    printf("Case %d: %s\n", cas, flag ? "No" : "Yes");}
  return 0;}
```

Weiqiao's Graph Theory

```
****************************Dinic********************************
#include <cstdio> <vector> <algorithm>,using namespace std;
// This Dinic code is copied from Stanford's ACM team notebook.
const long long INF = 2000000000;
struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}};
struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<Edge *> dad;
  vector<int> Q;
  // N = number of vertices
  Dinic(int N) : N(N), G(N), dad(N), Q(N) {}
  // Add an edge to initially empty network. from, to are 0-based
  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));}
  long long BlockingFlow(int s, int t) {
    fill(dad.begin(), dad.end(), (Edge *) NULL);
    dad[s] = &G[0][0] - 1;
    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size(); i++) {
        Edge &e = G[x][i];
        if (!dad[e.to] && e.cap - e.flow > 0) {
          dad[e.to] = &G[x][i];
          Q[tail++] = e.to;}}}
    if (!dad[t]) return 0;
    long long totflow = 0;
    for (int i = 0; i < G[t].size(); i++) {
      Edge *start = &G[G[t][i].to][G[t][i].index];
      int amt = INF;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
        if (!e) { amt = 0; break; }
        amt = min(amt, e->cap - e->flow);}
      if (amt == 0) continue;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
        e->flow += amt;
        G[e->to][e->index].flow -= amt;}
      totflow += amt;}
    return totflow;}
    // Call this to get the max flow. s, t are 0-based.
    // Note, you can only call this once.
    // To obtain the actual flow values, look at all edges with
    // capacity > 0 (zero capacity edges are residual edges).
  long long GetMaxFlow(int s, int t) {
    long long totflow = 0;
    while (long long flow = BlockingFlow(s, t))
      totflow += flow;
    return totflow;}};
int main() {
  int N, M; scanf("%d %d", &N, &M);
  Dinic D(N);
  while (M--) {
    int A, B, C;
    scanf("%d %d %d", &A, &B, &C);
    D.AddEdge(A-1, B-1, C);
    D.AddEdge(B-1, A-1, C);
    //to access flows, D.G[i][j].flow.
    //i is the node index, j is the j-th edge added to node i.
  }printf("%lld\n", D.GetMaxFlow(0, N-1));
  return 0;}
****************************mincost maxflow*************************
// UVa1658 Admiral, Rujia Liu
#include<cstdio>,<cstring>,<queue>,<vector>,<algorithm>,<cassert>
using namespace std;
const int maxn = 2000 + 10;
const int INF = 1000000000;
struct Edge {
  int from, to, cap, flow, cost;
  Edge(int u, int v, int c, int f, int w):from(u),to(v),cap(c),
    flow(f),cost(w) {} };
```

```
struct MCMF {
  int n, m;
  vector<Edge> edges;
  vector<int> G[maxn];
  int inq[maxn];          // in the queue or not
  int d[maxn];            // Bellman-Ford
  int p[maxn];            // the previous arc
  int a[maxn];            // amount to improve
  void init(int n) {
    this->n = n;
    for(int i = 0; i < n; i++) G[i].clear();
    edges.clear();}
  void AddEdge(int from, int to, int cap, int cost) {
    edges.push_back(Edge(from, to, cap, 0, cost));
    edges.push_back(Edge(to, from, 0, 0, -cost));
    m = edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);}
  bool BellmanFord(int s,int t,int flow_limit,int& flow,int& cost){
    for(int i = 0; i < n; i++) d[i] = INF;
    memset(inq, 0, sizeof(inq));
    d[s] = 0; inq[s] = 1; p[s] = 0; a[s] = INF;
    queue<int> Q;
    Q.push(s);
    while(!Q.empty()) {
      int u = Q.front(); Q.pop();
      inq[u] = 0;
      for(int i = 0; i < G[u].size(); i++) {
        Edge& e = edges[G[u][i]];
        if(e.cap > e.flow && d[e.to] > d[u] + e.cost) {
          d[e.to] = d[u] + e.cost;
          p[e.to] = G[u][i];
          a[e.to] = min(a[u], e.cap - e.flow);
          if(!inq[e.to]) { Q.push(e.to); inq[e.to] = 1; }}}}
    if(d[t] == INF) return false;
    if(flow + a[t] > flow_limit) a[t] = flow_limit - flow;
    flow += a[t];
    cost += d[t] * a[t];
    for(int u = t; u != s; u = edges[p[u]].from) {
      edges[p[u]].flow += a[t];
      edges[p[u]^1].flow -= a[t];}
    return true;}
  // need to make sure initial network doens't have negative cycles
  int MincostFlow(int s, int t, int flow_limit, int& cost) {
    int flow = 0; cost = 0;
    for (Edge &e : edges) e.flow = 0;//by Alex
    while(flow < flow_limit && BellmanFord(s, t, flow_limit, flow, cost));
    return flow;}};
MCMF g;
int main() {
  int n, m, a, b, c;
  while(scanf("%d%d", &n, &m) == 2 && n) {
    g.init(n*2-2);
    // spit Point 2~n-1 to arcs i->i', the former indexed 0~n-1,
    // latter indexed n~2n-3
    for(int i = 2; i <= n-1; i++)
      g.AddEdge(i-1, i+n-2, 1, 0);
    while(m--) {
      scanf("%d%d%d", &a, &b, &c);
      // connect a'-> b
      if(a != 1 && a != n) a += n-2; else a--;
      b--;
      g.AddEdge(a, b, 1, c);}
    int cost;
    g.MincostFlow(0, n-1, 2, cost);
    printf("%d\n", cost);}
  return 0;}
****************************two DFS to find SCC (white book version)
int V;//number of vertices
vector<int> G[MAX_V];   //adjacency representation of graph
vector<int> rG[MAX_V];  //graph after reversing the edges
vector<int> vs;         //post-order traverse of vertices
bool used[MAX_V];       //visiting masks
int cmp[MAX_V];         //topo order index of SCC
void add_edge(int from, int to){
    G[from].push_back(to);
    rG[to].push_back(from);}
void dfs(int v){
    used[v] = true;
    for(int i=0;i<G[v].size();i++){
        if(!used[G[v][i]]) dfs(G[v][i]);}
    vs.push_back(v);}
void rdfs(int v,int k){
    used[v] = true;
    cmp[v] = k;
    for(int i=0;i<rG[v].size();i++){
        if(!used[rG[v][i]])rdfs(rG[v][i],k);}}
int scc(){
    memset(used,0,sizeof(used));
    vs.clear();
    for(int v=0;v<V;v++){
        if(!used[v])dfs(v);}
    memset(used,0,sizeof(used));
    int k = 0;
    for(int i=vs.size()-1;i>=0;i--){
        if(!used[vs[i]]) rdfs(vs[i],k++);}
    return k;//number of SCC}
//example of using above alg (POJ 2186)
int N,M;
int A[MAX_M],B[MAX_M];
void solve(){
    V = N;
    for(int i=0;i<M;i++){
        add_edge(A[i]-1,B[i]-1);}
    int n = scc();
    //count the number of potential answers
    int u = 0, num = 0;
    for(int v = 0; v < V; v++){
        if (cmp[v] == n-1){
            u = v;
            num++; } }
    //check if reachable from all vertices
    memset(used, 0, sizeof(used));
    rdfs(u,0);//
    for(int v = 0; v < V; v++){
        if (!used[v]){
            //not reachable from this vertex
            num = 0;
            break;}}
    printf("%d\n",num);}
***conn comp of undirected graph (cuts) (can calculate articulation point,
    tested on UVa315)
***INIT: edge[][](adj matrix);vis[],pre[],anc[],deg[] set to 0;
***CALL: dfs(0, -1, 1, n);
```

```
***k=deg[0], deg[i]+1(i=1...n-1) num of conn comps after deleting the vertex
***Note: 0 as a root is special!
int edge[V][V], anc[V], pre[V], vis[V], deg[V];
void dfs(int cur, int father, int dep, int n){// vertex: 0 ~ n-1
    int cnt = 0;
    vis[cur] = 1; pre[cur] = anc[cur] = dep;
    for (int i=0; i<n; ++i) if (edge[cur][i]) {
        if (i != father && 1 == vis[i]) {
            if (pre[i] < anc[cur])
            anc[cur] = pre[i];}//back edge
        if (0 == vis[i]) { //tree edge
            dfs(i, cur, dep+1, n);
            ++cnt; // num of conn comps
            if (anc[i] < anc[cur]) anc[cur] = anc[i];
            if ((cur==0 && cnt>1) ||(cnt!=0 && anc[i]>=pre[cur]))
                ++deg[cur]; }}// link degree of a vertex
    vis[cur] = 2;}
void init(){mset(edge,0);mset(vis,0);mset(pre,0);mset(anc,0);mset(deg,0);}
***find bridge in undirected graph and print) (tested on UVa796 - Critical Links)
***INIT: edge[][](adj matrix);vis[],pre[],anc[],bridge set to 0;
***CALL: dfs(0, -1, 1, n);
const int V = 210;// max number of vertices
int bridge,edge[V][V], anc[V], pre[V], vis[V];
vector<ii> br;
void dfs(int cur, int father, int dep, int n){ // vertex: 0 ~ n-1
    //if (bridge) return;
    vis[cur] = 1; pre[cur] = anc[cur] = dep;
    for (int i=0; i<n; ++i) if (edge[cur][i]) {
        if (i != father && 1 == vis[i]) {
            if (pre[i] < anc[cur])
                anc[cur] = pre[i];}//back edge
        if (0 == vis[i]) { //tree edge
            dfs(i, cur, dep+1, n);
            //if (bridge) return;
            if (anc[i] < anc[cur]) anc[cur] = anc[i];
            if (anc[i] > pre[cur]) { bridge = 1; int a=min(i,cur);int b = max(i,cur); br.PB(MP(a,b));}}}
    vis[cur] = 2;}
void init(){mset(edge,0);mset(anc,0);mset(pre,0);mset(vis,0);bridge = 0;br.clear();}
int main(){
    --> init() and fill in edge[][]
        //for each connected component, do tree search
        fori(i,0,N){
            if (!vis[i])
                dfs(i, -1, 1, N);}
        if(!bridge) printf("0 critical links\n\n");
        else{
            sort(br.begin(),br.end(),cmp);
            printf("%lu critical links\n",br.size());
            fori(i,0,br.size()){
                printf("%d - %d\n",br[i].first,br[i].second);}
            printf("\n");}}
```

Weiqiao's biginteger

```
import java.util.Scanner;import java.math.BigInteger;
class Main {//UVa10925
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int caseNo = 1;
        while(true){
            int N = sc.nextInt(),F = sc.nextInt();
            if(N==0 && F==0) break;
            BigInteger sum = BigInteger.ZERO;
            for(int i = 0;i<N;i++){
                BigInteger V = sc.nextBigInteger();
                sum = sum.add(V);}
            System.out.println("Bill #" +  (caseNo++) + " costs " + sum +
                    ": each friend should pay " + sum.divide(BigInteger.valueOf(F)));
            System.out.println();//blank line}}}
//bigdecimal exponential
import java.math.BigDecimal;
import java.util.Scanner;
class Main{//UVa748 (bigdecimal exponential)
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int a; BigDecimal d;
        while(sc.hasNext()){
            d = sc.nextBigDecimal();
            a = sc.nextInt();
            String s = d.pow(a).toPlainString();
            //System.out.println(d.pow(a));
            //System.out.println(s);
            int l = 0, h = s.length() - 1;
            while(s.charAt(l)=='0'){l++;}
            while(s.charAt(h)=='0'){h--;}
            for(int b = l;b<=h;b++){
                System.out.print(s.charAt(b));
            }System.out.print("\n");}}}
addition -- add(BI), subtraction -- subtract(BI),
multiplication -- multiply(BI), power -- pow(int exponent)
division -- divide(BI), remainder -- remainder(BI)
modulo -- mod(BI), division and remainder -- divideAndRemainder(BI)
compareTo: b.compareTo(BigInteger.ZERO)==0
turn int to bigint: BigInteger.valueOf(int v)
```