My Template

Billy Inn

August 7, 2014

Contents

Chapter 1

Data Structures

1.1 Fenwick Tree

Test: CF 383C

1.1.1 unidimension

```
int C[maxn];
int lowbit(int x){return x&-x;}
void add(int x,int v)
{
    while(x<=n)
    {
        C[x]+=v;x+=lowbit(x);
    }
}
int getsum(int x)
{
    int ans=0;
    while(x>0)
    {
        ans+=C[x];x-=lowbit(x);
    }
    return ans;
}
```

1.1.2 two-dimension

```
int C[maxn][maxn];
int n,m;
int lowbit(int x){return x&-x;}
void add(int x,int y,int v)
```

```
{
    for(int i=x;i<=n;i+=lowbit(i))
        for(int j=y;j<=m;j+=lowbit(j))
        {
            C[i][j]+=v;
        }
}
int getsum(int x,int y)
{
    int ans=0;
    for(int i=x;i>0;i-=lowbit(i))
        for(int j=y;j>0;j-=lowbit(j))
        {
            ans+=C[i][j];
        }
    return ans;
}
```

1.2 RMQ

Test: CF 6E

```
int d[maxn][power];
void RMQ_init(int* A,int n)
{
        for(int i=0;i<n;i++) d[i][0]=A[i];
        for(int j=1;(1<<j)<=n;j++)
                for(int i=0;i+(1<<j)-1<n;i++)
                        d[i][j]=min(d[i][j-1],d[i+(1<<(j-1))][j-1]);

}
int RMQ(int L,int R)
{
        int k=0;
        while((1<<(k+1))<=R-L+1) k++;
        return min(d[L][k],d[R-(1<<k)+1][k]);
}
```

1.3 Splay Tree

1.3.1 Implemented by List

Test: hdu1754,1890,3436,3487,poj3468,UVa11922

```
const int maxn=200010;
struct Node
```

```
{
    Node *ch[2];    //children
    Node *pre;      //ancestor
    int sz;         //the number of nodes that are rooted in this node
    int val;        //the value to be maintained
    inline void push_up()       //like the function of segment tree
    {
        sz=1+ch[0]->sz+ch[1]->sz;
    }
    inline void push_down()     //like the function of segment tree
};
Node *null=new Node();          //virtual node
struct SplaySequence
{
    Node seq[maxn];
    Node *root;
    int a[maxn];
    Node* build(int l,int r,Node *f)
    {
        if(l>r) return null;
        int m=(l+r)>>1;
        Node *o=&seq[m];
        o->pre=f;
        o->val=a[m];
        o->ch[0]=build(l,m-1,o);
        o->ch[1]=build(m+1,r,o);
        o->push_up();
        return o;
    }
    void init(int sz)
    {
        null->sz=0;
        root=build(1,sz,null);
    }
    //d=0 rotate toward the   leftd  =1 rotate toward the right
    inline void rotate(Node *x,int d)
    {
        Node *y=x->pre;
        y->push_down();
        x->push_down();
        y->ch[d^1]=x->ch[d];
        if(x->ch[d]!=null) x->ch[d]->pre=y;
        x->pre=y->pre;
        if(y->pre!=null) y->pre->ch[y->pre->ch[1]==y]=x;
        x->ch[d]=y;
        y->pre=x;
```

```
        y->push_up();
        if(y==root) root=x;
    }
    //rotate x to the position below f and return x
    inline Node* splay(Node *x,Node *f)
    {
        while(x->pre!=f)
        {
            if(x->pre->pre==f) rotate(x,x->pre->ch[0]==x);
            else
            {
                Node *y=x->pre,*z=y->pre;
                int d=(z->ch[0]==y);
                if(y->ch[d]==x) rotate(x,d^1),rotate(x,d);
                else rotate(y,d),rotate(x,d);
            }
        }
        x->push_up();
        return x;
    }
    //rotate the node that is rooted in r
    //and ranks k to the position below f, and return it
    inline Node* select(int k,Node *f,Node *r)
    {
        Node *t=r;
        while(1)
        {
            int s=t->r-t->l+1;
            int tmp=t->ch[0]->sz;
            if(k<=tmp+s && k>tmp) break;
            if(k<=tmp) t=t->ch[0];
            else k-=tmp+s,t=t->ch[1];
        }
        return splay(t,f);
    }
    //split o to two trees
    //the first k nodes are rooted in l, the other are rooted in r
    inline void split(Node *o,int k,Node* &l,Node* &r)
    {
        l=select(k,null,o);
        r=l->ch[1];
        l->ch[1]=null;
        r->pre=null;
        l->push_up();
    }
    //merge l and r, and return the root
```

```
    inline Node* merge(Node *l,Node *r)
    {
        Node *t=select(l->sz,null,l);
        t->ch[1]=r;
        r->pre=t;
        t->push_up();
        return t;
    }
} ss;
void debug(Node *o)      //debug function
{
    if(o!=null)
    {
        debug(o->ch[0]);
        printf("%d\n",o->val);
        debug(o->ch[1]);
    }
}
```

1.3.2 Implemented by array

Test: POJ 3468

```
#include <cstdio>
#define keyTree (ch[ ch[root][1] ][0])
const int maxn = 222222;
struct SplayTree{
        int sz[maxn];
        int ch[maxn][2];
        int pre[maxn];
        int root , top1 , top2;
        int ss[maxn] , que[maxn];

        inline void Rotate(int x,int f) {
                int y = pre[x];
                push_down(y);
                push_down(x);
                ch[y][!f] = ch[x][f];
                pre[ ch[x][f] ] = y;
                pre[x] = pre[y];
                if(pre[x]) ch[ pre[y] ][ ch[pre[y]][1] == y ] = x;
                ch[x][f] = y;
                pre[y] = x;
                push_up(y);
        }
        inline void Splay(int x,int goal) {
```

```
        push_down(x);
        while(pre[x] != goal) {
            if(pre[pre[x]] == goal) {
                Rotate(x , ch[pre[x]][0] == x);
            } else {
                int y = pre[x] , z = pre[y];
                int f = (ch[z][0] == y);
                if(ch[y][f] == x) {
                    Rotate(x , !f) , Rotate(x , f);
                } else {
                    Rotate(y , f) , Rotate(x , f);
                }
            }
        }
        push_up(x);
        if(goal == 0) root = x;
    }
    inline void RotateTo(int k,int goal) {
        int x = root;
        push_down(x);
        while(sz[ ch[x][0] ] != k) {
            if(k < sz[ ch[x][0] ]) {
                x = ch[x][0];
            } else {
                k -= (sz[ ch[x][0] ] + 1);
                x = ch[x][1];
            }
            push_down(x);
        }
        Splay(x,goal);
    }
    inline void erase(int x) {
        int father = pre[x];
        int head = 0 , tail = 0;
        for (que[tail++] = x ; head < tail ; head++) {
            ss[top2++] = que[head];
            if(ch[ que[head] ][0]) que[tail++] = ch[ que[head] ][0];
            if(ch[ que[head] ][1]) que[tail++] = ch[ que[head] ][1];
        }
        ch[ father ][ ch[father][1] == x ] = 0;
        pushup(father);
    }
//above usually unchanged//////////////////////////////////////////////////

    inline void NewNode(int &x,int c) {
        if (top2) x = ss[--top2];
```

```
        else x = ++top1;
        ch[x][0] = ch[x][1] = pre[x] = 0;
        sz[x] = 1;

        val[x] = sum[x] = c;/*specialized function*/
        add[x] = 0;
    }

    inline void push_down(int x) {/*specialized function*/
        if(add[x]) {
            val[x] += add[x];
            add[ ch[x][0] ] += add[x];
            add[ ch[x][1] ] += add[x];
            sum[ ch[x][0] ] += (long long)sz[ ch[x][0] ] * add[x];
            sum[ ch[x][1] ] += (long long)sz[ ch[x][1] ] * add[x];
            add[x] = 0;
        }
    }
    inline void push_up(int x) {
        sz[x] = 1 + sz[ ch[x][0] ] + sz[ ch[x][1] ];
        /*specialized function*/
        sum[x] = add[x] + val[x] + sum[ ch[x][0] ] + sum[ ch[x][1] ];
    }

    /* initialize */
    inline void makeTree(int &x,int l,int r,int f) {
        if(l > r) return ;
        int m = (l + r)>>1;
        NewNode(x , num[m]);        /*varies according to the problem*/
        makeTree(ch[x][0] , l , m - 1 , x);
        makeTree(ch[x][1] , m + 1 , r , x);
        pre[x] = f;
        push_up(x);
    }
    inline void init(int n) {/*specialized funciton*/
        ch[0][0] = ch[0][1] = pre[0] = sz[0] = 0;
        add[0] = sum[0] = 0;

        root = top1 = 0;
        //for convience to add two nodes
        NewNode(root , -1);
        NewNode(ch[root][1] , -1);
        pre[top1] = root;
        sz[root] = 2;


```

```
            for (int i = 0 ; i < n ; i ++) scanf("%d",&num[i]);
            makeTree(keyTree , 0 , n-1 , ch[root ][1]);
            push_up(ch[root ][1]);
            push_up(root);
        }
        inline void update( ) {/* specialized funciton*/
            int l , r , c;
            scanf("%d%d%d",&l,&r,&c);
            RotateTo(l-1,0);
            RotateTo(r+1,root);
            add[ keyTree ] += c;
            sum[ keyTree ] += (long long)c * sz[ keyTree ];
        }
        inline void query() {/* specialized function*/
            int l , r;
            scanf("%d%d",&l,&r);
            RotateTo(l-1 , 0);
            RotateTo(r+1 , root);
            printf ("%lld\n",sum[keyTree]);
        }

        int num[maxn];
        int val [maxn];
        int add[maxn];
        long long sum[maxn];
}spt;


int main() {
        int n , m;
        scanf("%d%d",&n,&m);
        spt. init (n);
        while(m --) {
            char op [2];
            scanf("%s",op);
            if (op[0] == 'Q') {
                spt.query ();
            } else {
                spt.update();
            }
        }
        return 0;
}
```

Chapter 2

Math

2.1 High Precision Class

```
const int maxn = 200;
struct bign{
  int len, s[maxn];
  bign() {
    memset(s, 0, sizeof(s));
    len = 1;
  }
  bign(int num) {
    *this = num;
  }
  bign(const char* num) {
    *this = num;
  }
  bign operator = (int num) {
    char s[maxn];
    sprintf(s, "%d", num);
    *this = s;
    return *this;
  }
  bign operator = (const char* num) {
    len = strlen(num);
    for(int i = 0; i < len; i++) s[i] = num[len-i-1] - '0';
    return *this;
  }
  string str() const {
    string res = "";
    for(int i = 0; i < len; i++) res = (char)(s[i] + '0') + res;
    if(res == "") res = "0";
```

```
    return res;
  }
  bign operator + (const bign& b) const{
    bign c;
    c.len = 0;
    for(int i = 0, g = 0; g || i < max(len, b.len); i++) {
      int x = g;
      if(i < len) x += s[i];
      if(i < b.len) x += b.s[i];
      c.s[c.len++] = x % 10;
      g = x / 10;
    }
    return c;
  }
  void clean() {
    while(len > 1 && !s[len-1]) len--;
  }
  bign operator * (const bign& b) {
    bign c; c.len = len + b.len;
    for(int i = 0; i < len; i++)
      for(int j = 0; j < b.len; j++)
        c.s[i+j] += s[i] * b.s[j];
    for(int i = 0; i < c.len-1; i++){
      c.s[i+1] += c.s[i] / 10;
      c.s[i] %= 10;
    }
    c.clean();
    return c;
  }
  bign operator - (const bign& b) {
    bign c; c.len = 0;
    for(int i = 0, g = 0; i < len; i++) {
      int x = s[i] - g;
      if(i < b.len) x -= b.s[i];
      if(x >= 0) g = 0;
      else {
        g = 1;
        x += 10;
      }
      c.s[c.len++] = x;
    }
    c.clean();
    return c;
  }
  bool operator < (const bign& b) const{
    if(len != b.len) return len < b.len;
```

```
    for(int i = len-1; i >= 0; i--)
      if(s[i] != b.s[i]) return s[i] < b.s[i];
    return false;
  }
  bool operator > (const bign& b) const{
    return b < *this;
  }
  bool operator <= (const bign& b) {
    return !(b > *this);
  }
  bool operator == (const bign& b) {
    return !(b < *this) && !(*this < b);
  }
  bign operator += (const bign& b) {
    *this = *this + b;
    return *this;
  }
};
istream& operator >> (istream &in, bign& x) {
  string s;
  in >> s;
  x = s.c_str();
  return in;
}
ostream& operator << (ostream &out, const bign& x) {
  out << x.str();
  return out;
}
```

2.2 Number Theory

2.2.1 Sieve of Eratosthenes

```
const int maxn=10000010;
const int maxp=700000;
int vis[maxn],prime[maxp];
void sieve(int n)        //sieve the primes no larger than n
{
        int m=(int)sqrt(n+0..5);
        memset(vis,0,sizeof(vis));
        for(int i=2;i<=m;i++) if(!vis[i])
                for(int j=i*i;j<=n;j+=i) vis[j]=1;
        int c=0;
        for(int i=2;i<=n;i++) if(!vis[i])
                prime[c++]=i;
}
```

2.2.2 Sieve of Euler

The Sieve can be used to solve problem of multiplicative function.
Test: CF 10C

```
int vis[maxn],prime[maxp];
void sieve(int n)
{
        int tot=0;
        for(int i=2;i<=n;i++)
        {
                if(!vis[i]) prime[tot++]=i;
                for(int j=0;j<tot;j++)
                {
                        if(i*prime[j]>n) break;
                        vis[i*prime[j]]=1;
                        if(i%prime[j]==0) break;
                }
}
```

2.2.3 Euclid Algorithm

```
typedef long long LL;
LL gcd(LL a,LL b)
{
        return b==0?a:gcd(b,a%b);
}
void exgcd(LL a,LL b,LL& d,LL& x,LL& y)
{
        if(!b){d=a;x=1;y=0;}
        else
        {
                gcd(b,a%b,d,y,x);
                y-=x*(a/b);
        }
}
```

2.2.4 Euler Function

```
int euler_phi(int n)
{
        int m=(int)sqrt(n+0.5);
        int ans=n;
        for(int i=2;i<=m;i++) if(n%i==0)
        {
                ans=ans/i*(i-1);
```

```
            while(n%i==0) n/=i;
        }
        if(n>1) ans=ans/n*(n-1);
}
int phi[maxn];
void phi_table(int n)
{
        for(int i=2;i<=n;i++) phi[i]=0;
        phi[1]=1;
        for(int i=2;i<=n;i++) if(!phi[i])
        {
            for(int j=i;j<=n;j+=i)
            {
                if(!phi[j]) phi[j]=j;
                phi[j]=phi[j]/i*(i-1);
            }
        }
}
```

2.2.5 Calculate the Inversion

```
LL inv(LL a, LL n)
{
        LL d,x,y;
        exgcd(a,n,d,x,y);
        return d==1?(x+n)%n:-1;
}
```

2.2.6 Chinese Remainder Theorem

```
//n equations:x=a[i](mod m[i]) (0<=i<n)
LL china(int n,int *a,int *m)
{
        LL M=1,d,y,x=0;
        for(int i=0;i<n;i++) M*=m[i];
        for(int i=0;i<n;i++)
        {
            LL w=M/m[i];
            exgcd(m[i],w,d,d,y);
            x=(x+y*w*a[i])%M;
        }
        return (x+M)%M;
}
```

2.2.7 Bezout Theorem

For any two integers a,b, assume that d is their gcd. The equation $ax + by = m$ has integer solutions (x, y) if and only if m is the multiple of d. When the equation has solutions, the set of solutions are $\{(\frac{mx_0+kb}{d}, \frac{my_0-ka}{d})|k \in \mathbb{Z}\}$, where (x_0, y_0) is one solution of this equation.

2.3 Combinatorics

2.3.1 Calculate Combinations by Recurrence

```
void init()
{
    memset(C,0,sizeof(C));
    C[0][0]=1;
    for(int i=1;i<n;i++)
    {
        C[i][0]=C[i][i]=0;
        for(int j=1;j<i;j++)
            C[i][j]=C[i-1][j]+C[i-1][j-1];
    }
}
```

2.4 Matrix and System of Linear Equations

2.4.1 Matrix Fast Power

```
typedef matrix int[maxn][maxn];
void mat_mul(matrix A,matrix B,matrix res)
{
    matrix C;
    memset(C,0,sizeof(C));
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            for(int k=0;k<n;k++)
                C[i][j]+=A[i][k]*B[k][j];
    memcpy(res,C,sizeof(C));
}
void mat_pow(matrix A,int m,matrix res)
{
    matrix a,r;
    memcpy(a,A,sizeof(a));
    memset(r,0,sizeof(r));
    for(int i=0;i<n;i++) r[i][i]=1;
    while(m)
    {
        if(m&1) mat_mul(r,a,r);
        m>>=1;
        mat_mul(a,a,a);
```

```
    }
    memcpy(res,r,sizeof(r));
}
```

2.5 Numerical Method

2.5.1 Adaptive Simpson's Rule

```
//F is a global function
double simpson(double a,double b)
{
        double c=a+(b-a)/2;
        return (F(a)+4*F(c)+F(b))*(b-a)/6;
}
//recursive part
double asr(double a,double b,double eps,double A)
{
        double c=a+(b-a)/2;
        double L=simpson(a,c),R=simpson(c,b);
        if(fabs(L+R-A)<=15*eps) return L+R+(L+R-A)/15.0;
        return asr(a,c,eps/2,L)+asr(c,b,eps/2,R);
}
//main part
double asr(double a,double b,double eps)
{
        return asr(a,b,eps,simpson(a,b));
}
```

Chapter 3

Graph Theory

3.1 Fundamentals

3.1.1 Topo Sort

Test:UVaLive 4255

```
const int maxn=20;
int g[maxn][maxn]; //adjacency matrix
int in[maxn];       //indegree
int topo[maxn];     //result
int toposort(int n)
{
    int top=-1;
    for(int i=0;i<n;i++)
        if(in[i]==0)
        {
            in[i]=top;top=i;
        }
    for(int i=0;i<n;i++)
        if(top==-1) return 0; //exist cycle
        else
        {
            int j=top;
            top=in[j];
            topo[i]=j;
            for(int k=0;k<n;k++)
                if(g[j][k] && (--in[k])==0)
                {
                    in[k]==top;
                    top=k;
                }
```

```
        }
    return 1;       //succeed
}
```

3.1.2 Eulerian trail and circuit

Test:UVa 10054,10441

- undirected graph
    1. whether there exists an Eulerian circuit
        (a) the graph is connected
        (b) every vertex has even degree
    2. whether there exists an Eulerian trail
        (a) the graph is connected
        (b) there are only two vertex with odd degree which are the start point and the end point
- directed graph
    1. whether there exists an Eulerian circuit
        (a) the graph is connected
        (b) every vertex has equal indegree and outdegree
    2. whether there exists an Eulerian trail
        (a) the graph is connected
        (b) there is only one vertex with outdegree greater than indegree by one as the start point, and there is only one vertex with indegree greater than outdegree by one as end point

```
int g[maxn][maxn];      //adjacency matrix
int cnt[maxn];          //degree
vector<pair<int,int> > ans;     //edge list, stored in reverse order
int n,m;                        //n is the number of vertices, m is the number of edges
void euler(int u)
{
    for(int i=0;i<n;i++) if(g[u][i])
    {
        g[u][i]--;g[i][u]--;
        euler(i);
        ans.push_back(make_pair(u,i));
    }
}
int solve(int start)
{
```

```
    int flag=1;
    for(int i=0;i<n;i++)
        if(cnt[i]%2)
        {
            flag=0;break;
        }
    if(flag)
    {
        ans.clear();
        euler(start);
        if(ans.size()!=m || ans[0].second!=ans[ans.size()-1].first) flag=0;
    }
    return flag;        //whether exists an Eulerian circuit
}
```

3.2 Shortest Paths

3.3 Minimal Spanning Tree

3.3.1 Fundamentals

Theorem 1
For an n-vertex graph G(with $n \geq 1$),the following are equivalent(and character the trees with n vertices).

1. G is connected and has no cycles
2. G is connected and has $n-1$ edges
3. G has $n-1$ edges and no cycles
4. G has no loops and has, for each $u, v \in V(G)$, exactly one u,v-path

Corollary 1

1. Every edge of a tree is a cut-edge
2. Adding one edge to a tree forms exactly one cycle
3. Every connected graph contains a spanning tree

3.3.2 Kruskal algorithm

```
int cmp(const int i,const int j){return w[i]<w[j];} //indirect sorting function
int find(int x){return p[x]==x?x:p[x]=find(p[x]);}  //disjoint set
int Kruskal()
{
        int ans=0;
```

```
	for(int i=0;i<n;i++) p[i]=i; //initial disjoint set
	for(int i=0;i<m;i++) r[i]=i; //initial edge index
	sort(r,r+m,cmp); //sort edges
	for(int i=0;i<m;i++)
	{
		int e=r[i]; int x=find(u[e]); int y=find(v[e]);
		if(x!=y) ans+=w[e],p[x]=y;
	}
	return ans;
}
```

3.3.3 Second-best Minimum Spanning Tree

Test:UVa 10600

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn=110;
const int maxm=maxn*maxn/2;
struct edges
{
	int u,v,c;
	bool operator<(const edges& tmp) const
	{
		return c<tmp.c;
	}
} edge[maxm];
int n,m,e;
int head[maxn],pnt[maxn*2],nxt[maxn*2],cost[maxn*2];
int maxcost[maxn][maxn];
int f[maxn];
int vis[maxn],vis2[maxm];
void addedge(int u,int v,int c)
{
	pnt[e]=v;cost[e]=c;nxt[e]=head[u];head[u]=e++;
}
int find(int u){return f[u]==u?u:f[u]=find(f[u]);}
void dfs(int u,int fa,int c)
{
	maxcost[u][fa]=maxcost[fa][u]=c;
	for(int i=1;i<=n;i++) if(vis[i]) maxcost[u][i]=maxcost[i][u]=max(maxcost[i][fa],c);
	vis[u]=1;
	for(int i=head[u];i!=-1;i=nxt[i])
	{
```

```
                int v=pnt[i];
                if(v!=fa) dfs(v,u,cost[i]);
        }
}
int main()
{
        //freopen("in.txt","r",stdin);
        int T;
        scanf("%d",&T);
        while(T--)
        {
                scanf("%d%d",&n,&m);
                for(int i=0;i<m;i++) scanf("%d%d%d",&edge[i].u,&edge[i].v,&edge[i].c);
                for(int i=1;i<=n;i++) f[i]=i;
                memset(head,-1,sizeof(head));
                e=0;
                sort(edge,edge+m);
                int ans1=0;
                memset(vis2,0,sizeof(vis2));
                for(int i=0;i<m;i++)
                {
                        int u=find(edge[i].u),v=find(edge[i].v);
                        if(u!=v)
                        {
                                f[u]=v;
                                addedge(u,v,edge[i].c);
                                addedge(v,u,edge[i].c);
                                ans1+=edge[i].c;
                                vis2[i]=1;
                        }
                }
                memset(vis,0,sizeof(vis));
                dfs(1,0,0);
                int ans2=0x3fffffff;
                for(int i=0;i<m;i++) if(!vis2[i])
                {
                        int u=edge[i].u,v=edge[i].v,c=edge[i].c;
                        ans2=min(ans2,ans1+c-maxcost[u][v]);
                }
                printf("%d %d\n",ans1,ans2);
        }
        return 0;
}
```

3.3.4 Minimal Bottleneck Spanning Tree

Actually, the MBST is the MST. And we can get it by Kruskal Algorithm. If we want to query minimal bottleneck path between any two vertices, we can use dfs to calculate maxcost array just as we do in Second-best Minimal Spanning Tree. Then we can answer each query in $O(1)$, and the total complexity is $O(n^2)$.

3.3.5 Minimal Directed Spanning Tree

```
const int INF = 1000000000;
const int maxn = 100 + 10;

struct MDST {
  int n;
  int w[maxn][maxn];
  int vis[maxn];
  int ans;
  int removed[maxn];
  int cid[maxn];
  int pre[maxn];
  int iw[maxn];
  int max_cid;

  void init(int n) {
    this->n = n;
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++) w[i][j] = INF;
  }

  void AddEdge(int u, int v, int cost) {
    w[u][v] = min(w[u][v], cost);
  }

  int dfs(int s) {
    vis[s] = 1;
    int ans = 1;
    for(int i = 0; i < n; i++)
      if(!vis[i] && w[s][i] < INF) ans += dfs(i);
    return ans;
  }

  bool cycle(int u) {
    max_cid++;
    int v = u;
    while(cid[v] != max_cid) { cid[v] = max_cid; v = pre[v]; }
    return v == u;
```

```
  }

  void update(int u) {
    iw[u] = INF;
    for(int i = 0; i < n; i++)
      if (!removed[i] && w[i][u] < iw[u]) {
        iw[u] = w[i][u];
        pre[u] = i;
      }
  }

  bool solve(int s) {
    memset(vis, 0, sizeof(vis));
    if (dfs(s) != n) return false;

    memset(removed, 0, sizeof(removed));
    memset(cid, 0, sizeof(cid));
    for(int u = 0; u < n; u++) update(u);
    pre[s] = s; iw[s] = 0;
    ans = max_cid = 0;
    for (;;) {
      bool have_cycle = false;
      for(int u = 0; u < n; u++) if(u != s && !removed[u] && cycle(u)){
        have_cycle = true;
        int v = u;
        do {
          if(v != u) removed[v] = 1;
          ans += iw[v];
          for(int i = 0; i < n; i++) if(cid[i] != cid[u] && !removed[i]) {
            if(w[i][v] < INF) w[i][u] = min(w[i][u], w[i][v]-iw[v]);
            w[u][i] = min(w[u][i], w[v][i]);
            if(pre[i] == v) pre[i] = u;
          }
          v = pre[v];
        } while(v != u);
        update(u);
        break;
      }
      if (!have_cycle) break;
    }
    for(int i = 0; i < n; i++)
      if (!removed[i]) ans += iw[i];
    return true;
  }
};
```

3.4 Bipartite Graph Matching

3.4.1 Augmenting Path Algorithm

Test:POJ 3020

```
int n,m; //size of bipartite sets
int G[maxn][maxn]; //adjacency matrix
int linker[maxn],vis[maxn];
int dfs(int u)
{
        for(int v=0;v<m;v++)
                if(G[u][v] && !vis[v])
                {
                        vis[v]=1;
                        if(linker[v]==-1 || dfs(linker[v]))
                        {
                                linker[v]=u;
                                return 1;
                        }
                }
        return 0;
}
int hungary()
{
        int ans=0;
        memset(linker,-1,sizeof(linker));
        for(int u=0;u<n;u++)
        {
                memset(vis,0,sizeof(vis));
                if(dfs(u)) ans++;
        }
        return ans;
}
```

3.4.2 Kuhn-Munkers Algorithm

Test:UVaLive 4043

```
int W[maxn][maxn],n;
int Lx[maxn],Ly[maxn];  //label on X and Y
int linker[maxn],slack[maxn];
int S[maxn],T[maxn];    //mark whether visited
int dfs(int u)
{
        S[u]=1;
        for(int v=0;v<n;v++)
```

```
	{
		if(T[v]) continue;
		int tmp=Lx[u]+Ly[v]-W[u][v];
		if(tmp==0)
		{
			T[v]=1;
			if(linker[v]==-1 || dfs(linker[v]))
			{
				linker[v]=u;
				return 1;
			}
		}
		else if(slack[v]>tmp)
			slack[v]=tmp;
	}
	return 0;
}
void update()
{
	int a=inf;
	for(int i=0;i<n;i++)
		if(!T[i] && a-slack[i]>0)
			a=slack[i];
	for(int i=0;i<n;i++)
	{
		if(S[i]) Lx[i]-=a;
		if(T[i]) Ly[i]+=a;
		else slack[i]-=a;
	}
}
void KM()
{
	for(int i=0;i<n;i++)
	{
		linker[i]=-1;
		Lx[i]=Ly[i]=0;
		for(int j=0;j<n;j++)
			Lx[i]=max(Lx[i],W[i][j]);
	}
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++) slack[j]=inf;
		while(1)
		{
			memset(S,0,sizeof(S));
			memset(T,0,sizeof(T));
```

```
                if(dfs(i)) break;
                else update();
            }
        }
}
```

3.5 Network Flows

3.6 Depth First Search

3.6.1 Calculate DFS Sequence

Test:CF 383C

```
//tree is stored in adjacency lists
//dfs_clock is initialed as 0 vis  array is initialed as 0
//dfn is the dfs sequence, ldfs is the time the vertex enters and rdfs the time leaves
int ldfs[maxn],rdfs[maxn],dfn[maxn],vis[maxn];
int dfs_clock;
void dfs(int u)
{
    vis[u]=1;
    ldfs[u]=++dfs_clock;
    dfn[dfs_clock]=u;
    for(int i=head[u];i!=-1;i=nxt[i])
    {
        int v=pnt[i];
        if(!vis[u]) dfs(v);
    }
    rdfs[u]=++dfs_clock;
    dfn[dfs_clock]=u;
}
```

3.6.2 Multiplication LCA Algorithm(online)

Test:POJ 1330,3728

```
const int pow=20;
int d[maxn],p[maxn][pow];
int head[maxn],nxt[maxn],pnt[maxn];

void dfs(int u,int fa)                    //get d and p array
{
        d[u]=d[fa]+1;
        p[u][0]=fa;
```

```
        for(int i=1;i<pow;i++) p[u][i]=p[p[u][i-1]][i-1];
        for(int i=head[u];i!=-1;i=nxt[i])
        {
                int v=pnt[i];
                dfs(v,u);
        }
}
int lca(int a,int b)                    //get the lca of a and b in O(logn)
{
        if(d[a]>d[b]) a^=b,b^=a,a^=b;
        if(d[a]<d[b])
        {
                int del=d[b]-d[a];
                for(int i=0;i<pow;i++) if(del&(1<<i)) b=p[b][i];
        }
        if(a!=b)
        {
                for(int i=pow-1;i>=0;i--)
                        if(p[a][i]!=p[b][i])
                                a=p[a][i],b=p[b][i];
                a=p[a][0],b=p[b][0];
        }
        return a;
}
```

3.6.3 Targan's Off-line LCA Algorithm

Test:POJ 1330,1470

```
vector<int> g[maxn],q[maxn];//adjacency lists and query lists
int f[maxn];
int a[maxn];//record the current lca of the vertex
int vis[maxn];
//disjoint set
int find(int x) {return x==f[x]?x:f[x]=find(f[x]);}
void uni(int x,int y)
{
        int a=find(x);
        int b=find(y);
        if(x!=y) f[a]=b;
}
//use:lca(root);
//usually insert (u,v) and (v,u) into q
void lca(int u)
{
        a[u]=u;
```

```
        for(int i=0;i<g[u].size(); i++)
        {
                lca(g[u][i]);
                uni(u,g[u][i]);
                a[find(u)]=u;
        }
        vis[u]=1;
        for(int i=0;i<q[u].size(); i++)
        {
                if(vis[q[u][i]])
                {
                    int LCA=a[find(q[u][i])];     //answer the query
                    operation();                  //operate corresponding operations
                }
        }
}
```

Chapter 4

String

4.1 KMP

```
//fail function
int f[maxn];
void getfail(char* P)
{
    int m=strlen(P);
    f[0]=0;f[1]=0;
    for(int i=1;i<m;i++)
    {
        int j=f[i];
        while(j && P[i]!=P[j]) j=f[j];
        f[i+1]=P[i]==P[j]?j+1:0;
    }
}

//find function
void find(char* T,char* P)
{
    int n=strlen(T),m=strlen(P);
    int j=0;
    for(int i=0;i<n;i++)
    {
        while(j && P[j]!=T[i]) j=f[j];
        if(P[j]==T[i]) j++;
        if(j==m)
        {
            operate();
            j=f[j];     //note when find a match, remember to get backward along fail edge once
        }
```

```
    }
}
```

4.2 AC Automation

```
//tire tree
const int maxn=100010;//maximal node number
const int sigma=26;
//the size of character size,the value of characters are 0~sigma-1
int ch[maxn][sigma];  //ch[i][j] means the node going from i along j to
int val[maxn];        //additive information
int sz;               //the total number of node
void init()           //initial
{
    sz=1;
    memset(ch[0],0,sizeof(ch[0]));
}
int idx(char c)       //index the character
{
    return c-'a';     //vary depending on specific condition
}
//insert string s,whose additive information is v
//note that v cannot be 0
void insert(char *s,int v)
{
    int u=0,n=strlen(s);
    for(int i=0;i<n;i++)
    {
        int c=idx(s[i]);
        if(!ch[u][c])
        {
            memset(ch[sz],0,sizeof(ch[sz]));
            val[sz]=0;
            ch[u][c]=sz++;
        }
        u=ch[u][c];
    }
    val[u]=v;
    //val[u]|=(1<<v);
    //operating like this when it comes to state compressing
}

//get fail function
int f[maxn];           //fail function
int last[maxn];        //suffix link
void getfail()
```

```
{
    queue<int> q;
    f[0]=0;
    for(int c=0;c<sigma;c++)
    {
        int u=ch[0][c];
        if(u) {f[u]=0;q.push(u); last[u]=0;}
    }
    while(!q.empty())
    {
        int r=q.front(); q.pop();
        for(int c=0;c<sigma;c++)
        {
            int u=ch[r][c];
            if(!u)
            {
                //ch[r][c]=ch[f[r]][c];
                //add all nonexist edges, applying on dp
                continue;
            }
            q.push(u);
            int v=f[r];
            while(v && !ch[v][c]) v=f[v]; //when all nonexist edges are added, code here can be deleted
            f[u]=ch[v][c];
            last[u]=val[f[u]]?f[u]: last[f[u]];
        }
    }
}

int find(char *T)
{
    int n=strlen(T);
    int j=0;
    for(int i=0;i<n;i++)
    {
        int c=idx(T[i]);
        while(j && !ch[j][c]) j=f[j];     //when all nonexist edges are added, code here can be deleted
        j=ch[j][c];
        if(val[j]) process(j);
        else if(last[j]) process(last[j]);
    }
}

void process(int j)
{
    if(j)
```

```
    {
        operate();      //vary depending on specific condition
        process(last[j]);
    }
}
```

4.3 Suffix Array

```
int s[maxn];                    //string to be constructed
int sa[maxn];                   //suffix array
int t[maxn],t2[maxn],c[maxn];
//every character's value is 0-m-1,n is usually the length of string plus 1
 void build_sa(int m,int n)            //construct suffix array
{
    int i,*x=t,*y=t2;
    for(i=0;i<m;i++) c[i]=0;
    for(i=0;i<n;i++) c[x[i]=s[i]]++;
    for(i=1;i<m;i++) c[i]+=c[i-1];
    for(i=n-1;i>=0;i--) sa[--c[x[i]]]=i;
    for(int k=1;k<=n;k<<=1)
    {
        int p=0;
        for(i=n-k;i<n;i++) y[p++]=i;
        for(i=0;i<n;i++) if(sa[i]>=k) y[p++]=sa[i]-k;
        for(i=0;i<m;i++) c[i]=0;
        for(i=0;i<n;i++) c[x[y[i]]]++;
        for(i=0;i<m;i++) c[i]+=c[i-1];
        for(i=n-1;i>=0;i--) sa[--c[x[y[i]]]]=y[i];
        swap(x,y);
        p=1;x[sa[0]]=0;
        for(i=1;i<n;i++)
            x[sa[i]]=y[sa[i-1]]==y[sa[i]]&&y[sa[i-1]+k]==y[sa[i]+k]?p-1:p++;
         if(p>=n) break;
        m=p;
    }
}

//calculate rank and height array
int rank[maxn];         //suffix i's index in sa
int height[maxn];       //the LCP of sa[i-1] and sa[i]
void getheight(int n) //n is the length of string
{
    int i,j,k=0;
    for(i=0;i<=n;i++) rank[sa[i]]=i;
    for(i=0;i<n;i++)
    {
```

```
        if(k) k--;
        int j=sa[rank[i]-1];
        while(s[i+k]==s[j+k]) k++;
        height[rank[i]]=k;
    }
}

//RMQ
int d[maxn][50];
void RMQ_init(int n)
{
    for(int i=0;i<n;i++) d[i][0]=height[i];
    for(int j=1;(1<<j)<=n;j++)
        for(int i=0;i+(1<<j)-1<n;i++)
            d[i][j]=min(d[i][j-1],d[i+(1<<(j-1))][j-1]);
}
int RMQ(int L,int R)
{
    if(L>R) swap(L,R);
    if(L==R) return L-sa[L];
    L++;
    int k=0;
    while((1<<(k+1))<=R-L+1) k++;
    return min(d[L][k],d[R-(1<<k)+1][k]);
}

//inital
build_sa(m,n+1);
getheight(n);
RMQ_init(n+1);
```

4.4 Suffix Automation

```
char s[maxn];               //string to be constructed
int sz;                     //the number of node
struct state
{
    state *pre;
    state *go[sigma];
    int val;                //the length of the longest path
    void clear()            // initialize node
    {
        pre=0;val=0;
        memset(go,0,sizeof(go));
    }
};
```

```
state *root,* last ;            //root node and last node added
state st[maxn];
void init()                     // initialize SAM
{
    sz=0;
    root=last=&st[sz++];
    root->clear();
}
void extend(int w)              //extend node
{
    state *p=last;              //last node added
    state *np=&st[sz++];        //create a new node
    np->clear();
    np->val=p->val+1;
    while(p && p->go[w]==0) //add edge to np when p->go[w]=0
    {
        p->go[w]=np;
        p=p->pre;
    }
    if(p==0) np->pre=root; //if w occurs first,add edge to the root
    else
    {
        state *q=p->go[w];
        if(q->val==p->val+1) //q is the proper state
            np->pre=q;
        else
        {
            state *nq=&st[sz++];//create a new node and copy the information
            nq->clear();
            nq->val=p->val+1;
            memcpy(nq->go,q->go,sizeof(q->go));
            nq->pre=q->pre;
            q->pre=nq;
            np->pre=nq;
            while(p && p->go[w]==q)
            {
                p->go[w]=nq;
                p=p->pre;
            }
        }
    }
    last=np;
}

//sort the nodes
int t[maxn];
```

```
state *r[maxn];
//l is the length of string,sz is the number of node
//note here contains root node,root node is indexed as 1 in r
for(int i=0;i<l;i++) t[i]=0;
for(int i=0;i<sz;i++) t[st[i].val]++;
for(int i=1;i<l;i++) t[i]+=t[i-1];
for(int i=0;i<sz;i++) r[t[st[i].val]--]=&st[i];
```

4.5 String Hash

```
typedef unsigned long long ULL;
const int maxn=10000;
const int x=123;          //hash's parameter
ULL H[maxn];              //the hash value of string
ULL xp[maxn];             //x^n
void hash_init(char *s) // initialize H(n),x^n
{
    int n=strlen(s);
    H[n]=0;
    for(int i=n-1;i>=0;i--) H[i]=H[i+1]*x+(s[i]-'a'); //calculate H(n)
    xp[0]=1;
    for(int i=1;i<=n;i++) xp[i]=xp[i-1]*x;               //calculate x^n
}
//hash function,return the hash value of string starts from i with length of L
ULL hash(int i,int L)
{
    return H[i]-H[i+L]*xp[L];
}
```

4.6 Longest Palindromic Substring

Test:LeetCode, Longest Palindromic Substring

```
int p[2010];
void get(string str)
{
        int mx=0,id;
        for(int i=1;i<str.size();i++)
        {
                if(mx>i) p[i]=min(p[2*id-i],mx-i);
                else p[i]=1;
                for(;str[i+p[i]]==str[i-p[i]];p[i]++)
                        ;
                if(p[i]+i>mx)
                {
                        mx=p[i]+i;
```

```
                id=i;
            }
        }
}
string longestPalindrome(string s)
{
        string t="$";
        for(int i=0;i<s.size(); i++)
            t+='#',t+=s[i];
        t+='#';
        get(t);
        int ans=0,pos;
        string str="";
        for(int i=1;i<t.size(); i++)
            if(p[i]>ans)
            {
                ans=p[i];
                pos=i;
            }
        for(int i=pos-ans+1;i<pos+ans;i++)
            if(t[i]!='#') str+=t[i];
        return str;
}
```

Chapter 5

Geometry

5.1 Basic

comparsion between real numbers

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

definition of point and vector

```
struct point
{
    double x,y;
    point(double x=0,double y=0):x(x),y(y){}
};
point operator+(point a,point b){return point(a.x+b.x,a.y+b.y);}
point operator-(point a,point b){return point(a.x-b.x,a.y-b.y);}
point operator*(point a,double p){return point(a.x*p,a.y*p);}
point operator/(point a,double p){return point(a.x/p,a.y/p);}
bool operator <(const point& a,const point& b)
{
    return dcmp(a.x-b.x)<0 || (dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)<0);
}
bool operator==(point a,point b)
{
    return dcmp(a.x-b.x)==0 && dcmp(a.y-b.y)==0;
}
```

$Dot(\vec{u},\vec{v}) = |\vec{u}| * |\vec{v}| * cos < \vec{u},\vec{v} >$

```
double dot(point a,point b){return a.x*b.x+a.y*b.y;}
double length(point a){return sqrt(dot(a,a));}
double angle(point a,point b){return acos(dot(a,b)/length(a)/length(b));}
```

$Cross(\vec{u},\vec{v})$ is the twice of the directed area of the triangle formed by $\vec{u}$ & $\vec{v}$

```
double cross(point a,point b){return a.x*b.y-a.y*b.x;}
double area2(point a,point b,point c){return cross(b-a,c-a);}
```

rotate the vector

```
point rotate(point a,double rad)
{
    return point(a.x*cos(rad)-a.y*sin(rad),a.x*sin(rad)+a.y*cos(rad));
}
```

calculate normal of the vector

```
point normal(point a)
{
    double L=length(a);
    return point(-a.y/L,a.x/L);
}
```

get intersection of two lines which are not parallel to each other

```
//p,q are points on two lines
//v,w are direction vector of two lines
point getlineinter(point p,point v,point q,point w)
{
    point u=p-q;
    double t=cross(w,u)/cross(v,w);
    return p+v*t;
}
```

distance from point to line

```
double distoline(point p,point a,point b)
{
    point v1=b-a,v2=p-a;
    return fabs(cross(v1,v2))/length(v1);
}
```

distance from point to segment

```
double distoseg(point p,point a,point b)
{
    if(a==b) return length(p-a);
    point 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);
}
```

projection of point on line

```
point getpro(point p,point a,point b)
{
    point v=b-a;
    return a+v*(dot(v,p-a)/dot(v,v));
}
```

judge whether two lines are properly intersected

```
bool segproperint(point a1,point a2,point b1,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;
}
```

judge whether point on segment, endpoints exclusive

```
bool onseg(point p,point a1,point a2)
{
    return dcmp(cross(a1-p,a2-p))==0 && dcmp(dot(a1-p,a2-p))<0;
}
```

directed area of polygon

```
double polyarea(point* p,int n)
{
    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;
}
```

5.2 Problems About Circle and Sphere

5.2.1 Definition of circle

```
struct circle
{
        point c;
        double r;
        circle (point c=point(0,0),double r=0):c(c),r(r){}
        point getpoint(double a)
        {
                return point(c.x+cos(a)*r,c.y+sin(a)*r);
```

```
        }
};
```

5.2.2 Calculate the intersection of line and circle

```
\\return the number of intersections
\\the intersections are stored in sol
int getlinecircleinter (point p,point v, circle C,double& t1,
                                        double& t2,vector<point>& sol)
{
        double a=v.x,b=p.x-C.c.x,c=v.y,d=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(p+v*t1);
                return 1;
        }
        t1=(-f-sqrt(delta))/(2*e);sol.push_back(p+v*t1);
        t2=(-f+sqrt(delta))/(2*e);sol.push_back(p+v*t2);
        return 2;
}
```

5.2.3 Calculate the intersection of circles

```
double angle(point v){return atan2(v.y,v.x);}   \\calculate the polar angle

\\return the number of intersections
\\the intersections are stored in sol
int getcircleinter ( circle C1, circle C2,vector<point>& sol)
{
        double d=length(C1.c-C2.c);
        if(dcmp(d)==0)
        {
                if(dcmp(C1.r-C2.r)==0) return -1;
                return 0;
        }
        if(dcmp(C1.r+C2.r-d)<0) return 0;
        if(dcmp(fabs(C1.r-C2.r)-d)>0) return 0;

        double a=angle(C2.c-C1.c);
        double da=acos((C1.r*C1.r+d*d-C2.r*C2.r)/(2*C1.r*d));
        point p1=C1.getpoint(a-da),p2=C1.getpoint(a+da);
        sol.push_back(p1);
```

```
        if(p1==p2) return 1;
        sol.push_back(p2);
        return 2;
}
```

5.3 2D Algorithm

5.3.1 Judge whether point in polygon

```
//p is the point,poly is the polygon
int isinpoly(point p,vector<point>& poly)
{
    int wn=0;
    int n=poly.size();
    for(int i=0;i<n;i++)
    {
        if(onseg(p,poly[i],poly[(i+1)%n])) return -1;  //on the boarder
        int k=dcmp(cross(poly[(i+1)%n]-poly[i],p-poly[i]));
        int d1=dcmp(poly[i].y-p.y);
        int d2=dcmp(poly[(i+1)%n].y-p.y);
        if(k>0 && d1<=0 && d2>0) wn++;
        if(k<0 && d2<=0 && d1>0) wn--;
    }
    if(wn!=0) return 1;     //inside
    return 0;               //outside
}
```

5.4 3D Algorithm

5.5 Stimulated Annealing Algorithm

5.5.1 Minimum Sphere Coverage

Test:UVa 10095

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
using namespace std;
const double eps=1e-8;
const double inf=1e20;
const int maxn=10010;
int dcmp(double x)
{
    if(fabs(x)<0) return 0;
```

```
    else return x<0?-1:1;
}
struct point
{
    double x,y,z;
} p[maxn];
double length(point a,point b)
{
    double dx=a.x-b.x;
    double dy=a.y-b.y;
    double dz=a.z-b.z;
    return sqrt(dx*dx+dy*dy+dz*dz);
}
int max_dis(int n,point q)
{
    int j;
    double res=0;
    for(int i=0;i<n;i++)
    {
        double tmp=length(q,p[i]);
        if(dcmp(tmp-res)>0)
        {
            res=tmp;
            j=i;
        }
    }
    return j;
}
int main()
{
    int n;
    while(scanf("%d",&n)&&n)
    {
        for(int i=0;i<n;i++) scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);
        if(n==1) printf("0.0000 %.4f %.4f %.4f\n",p[0].x,p[0].y,p[0].z);
        else
        {
            double r=inf;
            double step=100000;    //maximal coordinate range
            point q;
            q.x=q.y=q.z=0;         //select a node randomly
            while(step>eps)
            {
                int j=max_dis(n,q);
                double tmp=length(p[j],q);
                if(dcmp(r-tmp)>0) r=tmp;
```

```
                double dx=(p[j].x-q.x)/tmp;
                double dy=(p[j].y-q.y)/tmp;
                double dz=(p[j].z-q.z)/tmp;
                q.x+=dx*step;
                q.y+=dy*step;
                q.z+=dz*step;
                step*=0.993;        //decided by precision, better no less than 0.99
            }
            printf("%.4f %.4f %.4f %.4f\n",r,q.x,q.y,q.z);
        }
    }
    return 0;
}
```

Chapter 6

Dynamic Programming

6.1 Longest Common Increasing Sequence

Time Complexity: $O(n^2)$
Test: CF 10D

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int maxn = 510;
int n,m;
int a[maxn],b[maxn];
int dp[maxn][maxn];
int path[maxn][maxn];
void print(int i,int j,int pre)
{
	if(path[i][j]==0)
	{
		if(j!=pre) printf("%d ",b[j]);
		return;
	}
	print(i-1,path[i][j],j);
	if(j!=pre) printf("%d ",b[j]);
}
int main()
{
	scanf("%d",&n);
	for(int i=1;i<=n;i++) scanf("%d",&a[i]);
	scanf("%d",&m);
	for(int i=1;i<=m;i++) scanf("%d",&b[i]);
	memset(dp,0,sizeof(dp));
```

```
    memset(path,0,sizeof(path));
    for(int i=1;i<=n;i++)
    {
        int tmp=0,k=0;
        for(int j=1;j<=m;j++)
        {
            dp[i][j]=dp[i-1][j];
            path[i][j]=j;
            if(a[i]>b[j] && tmp<dp[i-1][j])
            {
                tmp=dp[i-1][j];
                k=j;
            }
            if(a[i]==b[j])
            {
                dp[i][j]=tmp+1;
                path[i][j]=k;
            }
        }
    }
    int ans=0,p=0;
    for(int i=1;i<=m;i++)
        if(ans<dp[n][i])
        {
            ans=dp[n][i];
            p=i;
        }
    printf("%d\n",ans);
    print(n,p,0);
    puts("");
    return 0;
}
```

Chapter 7

Other Topics

7.1 Dichotomy

7.1.1 condition 1

```
while(L<R)
{
        int M=L+(R-L+1)/2;
        if(judge(M)) L=M;
        else R=M-1;
}
```

7.1.2 condition 2

```
while(L<R)
{
        int M=L+(R-L)/2;
        if(judge(M)) R=M;
        else L=M+1;
}
```

7.1.3 condition 3

```
while(R-L>eps)
{
        double M=(L+R)/2;
        if(judge(M)) L=M;
        else R=M;
}
```

7.2 Hash List

Test:UVa 10085

```
const int maxn=1000000;
const int mod=1000003;
state st[maxn];
int head[mod],nxt[mod];
void init(){memset(head,0,sizeof(head));}     //initial, 0 represents nonexist
int try_to_insert(int s)                       //insert function
{
    int tmp=hash(s);                           //calculate the hash value
    int u=head[tmp];
    while(u)
    {
        if(memcmp(st[u],st[s],sizeof(st[s]))==0) return 0;
        u=nxt[u];
    }
    nxt[s]=head[tmp];
    head[tmp]=s;
    return 1;
}
```

7.3 Construct a proper answer for N Queens Problem

Test:UVa 10094

1. $n\%2 \neq 0 \bigwedge n\%3 \neq 0$

   n is even: 2,4,6,8, ...,n,1,3,5,7, ...,n-1

   n is odd: 2,4,6,8, ...,n-1,1,3,5,7, ...,n

2. $n\%2 = 0 \bigvee n\%3 = 0$

   when n is even, k is n/2;when n is odd k is (n-1)/2.

   n is even and k is even: k,k+2,...,n,2,4,6,...,k-2,k+3,k+5,...,n-1,1,3,5,...,k+1

   n is odd and k is even: k,k+2,...,n-1,2,4,6,...,k-2,k+3,k+5,...,n-2,1,3,5,...,k+1,n

   n is even and k is odd: k,k+2,...,n-1,1,3,5,...,k-2,k+3,k+5,...,n,2,4,6,...,k+1

   n is odd and k is odd: k,k+2,...,n-2,1,3,5,...,k-2,k+3,k+5,...,n-1,2,4,6,...,k+1,n