ACM-ICPC Template

GuessEver

Last Modify at January 30, 2015

Contents

1 Dynamic Programming

1.1 LCS - Longest Common Subsequence

```
int LCS() // O(N*N)
{//字符串纠正到以 1 为下标
    int f[N][N];
    int res = 0;
    for(int i = 1; i < lena; i++)
        for(int j = 1; j < lenb; j++)
        {
            if(a[i] == a[j]) f[i][j] = f[i-1][j-1] + 1;
            else f[i][j] = max(f[i-1][j], f[i][j-1]);
            res = max(res, f[i][j]);
        }
    return res;
}

int LCS() // O(NlogN)
{//把 LCM 转化为 LIS 来做
// 1 2 5 9 3 --> 1 2 3 4 5
// 1 5 3 9 2 --> 1 3 5 4 2 --> 对这个序列跑LIS()
    //------change------
        //这里就要针对数据自己想尽办法转化了
        for(int i = 1; i <= n; i++) h[a[i]] = i;
        for(int i = 1; i <= n; i++) b[i] = h[b[i]];
    //------end------
    return LIS();
}
```

1.2 LIS - Longest Increasing Subsequence

```
int f[N];
int LIS()//O(N*N)
{
    for(int i = 1; i <= n; i++)
        for(int j = i-1; j >= 1; j--)
            if(a[i] > a[j]) f[i] = max(f[i], f[j] + 1);
    int res = 0;
    for(int i = 1; i <= n; i++) res = max(res, f[i]);
    return res;
}

int c[N], len = 0;
int LIS()//(NlogN)
{
    for(int i = 1; i <= n; i++)
    {
        //------find------
            int l = 1, r = len, mid;
            while(l <= r)
            {
                mid = (l + r) / 2;
                if(a[i] > c[mid]) l = mid + 1;
                else r = mid - 1;
            }
        //------end------
        c[l] = a[i];
        len = max(len, l);
    }
    return len;
}
```

1.3 Maximum Continuous Subsequence Sum

```
int MaxSubSum()
{
    int f[N], res;
    for(int i = 1; i <= n; i++)
    {
        f[i] = max(a[i], f[i-1] + a[i]);
        res = max(res, f[i]);
    }
    return res;
}

int MaxSubSum()
{
    int res = 0, now = 0;
    for(int i = 1; i <= n; i++)
    {
        now += a[i];
        res = max(res, now);
        if(now < 0) now = 0;
    }
    return res;
}
```

1.4 数位 dp

```
int predoing(LL a, int *num)
{
    int le = 0;
    while(a)
    {
        num[++le] = a % 10;
        a /= 10;
    }
    return le;
}
int calc(int pos, int d, int u, int last)
{
    if(pos == 0) return 1;
    int &res = f[pos][d][u][last];
    if(res != -1) return res;
    res = 0;
    int st = d ? L[pos] : 0;
    int ed = u ? R[pos] : 9;
    for(int i = st; i <= ed; i++)
        if(合法) res += calc(pos - 1, d && i == L[pos], u && i == R[pos], i);
    return res;
}
```

2 Math

2.1 GCD && LCM

2.1.1 GCD - Greatest Common Divisor

```
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
```

2.1.2 LCM - Least Common Multiple

```
inline int lcm(int a, int b) { return a / gcd(a, b) * b; }
```

2.1.3 E_GCD - Extended Greatest Common Divisor

```
ax + by = 1
bx1 + (a%b)y1 = 1    ==>   bx + (a-a/b*b)y = 1
  ==> ay1 + b(x1-a/b*y1) = 1
对应  ax  + by         = 1

int egcd(int a, int b, int &x, int &y)
{
    if(b == 0)
    {
        x = 1; y = 0;
        return a;
    }
    int x1, y1;
    int e = egcd(b, a%b, x1, y1);
    x = y1;
    y = x1 - a / b * y1;
    return e;
}
```

2.2 Prime

2.2.1 Make Prime List

```
void make_prime_list(int maxp) // O(2*N)
{
    for(int i = 2; i <= maxp; i++)
    {
        if(!h[i]) pri[l++] = i;
        for(int j = 0; j < l && pri[j] <= maxp / i; j++)
        {
            h[i * pri[j]] = true;
            if(i % pri[j] == 0) break;
        }
    }
}
```

2.2.2 Prime Factor

```
void factor()
{
    make_prime_list();
    for(int j = 0; j < Cnt && pri[j]*pri[j] <= n; j++)
    {
        if(n % pri[j] == 0)
        {
            printf("%d␣", pri[j]);
            while(n % pri[j] == 0) n /= pri[j];
        }
    }
    if(n!=1) printf("%d",n);
}
```

2.2.3 Fast Power

```
//x^y  % mod
int mul(int x, LL y, int mod) // μ□�
{
    if(y == 1) return x;
    if(y & 1) return (mul((x * (LL)x) % mod, y / 2, mod) * (LL)x) % mod;
    else return mul((x * (LL)x) % mod, y / 2, mod) % mod;
}
int mul(int x, int y, int mod) // ·ǵ□�
{
    int s = 1;
    int ss = x;
    while(y)
    {
        if(y & 1) s = s * ss;
        y /= 2;
        ss *= ss;
    }
    return s;
}
```

3 Datastructure

3.1 Leftist Tree

```
//很多时候需要配合并查集一起使用
int getroot(int x){return f[x]==x ? x : f[x]=getroot(f[x]);}

//把x和y合并在一起，其实就是把y插入x
int merge(int x,int y)//返回合并后子树的根
{
    if(!x || !y) return x|y;
    if(A[x] < A[y]) swap(x,y);//大根堆，如果y比x大，与其让y插入x，不如让x插入y
    R[x]=merge(R[x],y);//始终往右子树合并
    f[R[x]] = x;//更新并查集
    if(D[R[x]] > D[L[x]]) swap(L[x],R[x]);//保持左偏树性质
    D[x] = D[R[x]] + 1;
    若还有其他维护信息也需要更新;
    return x;//返回根
}

int del(int x)
{
    int t = merge(L[x],R[x]);
    f[L[x]] = L[x]; f[R[x]] = R[x];//更新并查集
    L[x] = R[x] = D[x] = 0;
    return t;
}
```

3.2 Partition Tree

```
struct Parti{int val, left;} val[30][N];
void build_tree(int d, int l, int r)
{
    if(l == r) return;
    int m = (l + r) >> 1, same = m - l + 1;
    int lcnt = l, rcnt = m + 1;
    for(int i = l; i <= r; i++)
        if(val[d][i].val < sorted[m]) same--;
    for(int i = l; i <= r; i++)
    {
        int flag = 0;
        if((val[d][i].val < soted[m]) || (val[d][i].val == sorted[m] && same))
        {
            flag = 1;
            val[d + 1][lcnt++] = val[d][i];
            if(val[d][i].val == sorted[m]) same--;
        }
        else val[d][rcnt++] = val[d][i];
        val[d][i].left = val[d][i - 1].left + flag;
    }
    build_tree(d + 1, l, m);
    build_tree(d + 1, m + 1, r);
}
int query(int d, int l, int r, int x, int y, int k)
{
    if(l == r) return val[d][l].val;
    int m = (l + r) >> 1;
    int lx = val[d][x - 1].left - val[d][l - 1].left; //[l,x-1] to left
    int ly = val[d][y].left - val[d][x - 1].left; //[x,y] to left
    int rx = (x - 1 - l + 1) - lx; //[l,x-1] to right
    int ry = (y - x + 1) - ly; //[x,y] to right
    if(ly >= k) return query(d+1, l, m, l-1+lx+1, l-1+lx+ly, k);
    else return query(d+1, m+1, r, m+1-1+rx+1, m+1-1+rx+ry, k-ly);
}
```

3.3 Size Balanced Tree

```
int A[N], S[N], L[N], R[N], root, total;
void rotate_left(int &x)
{
    int y = R[x];
    R[x] = L[y];
    L[y] = x;
    S[y] = S[x];
    S[x] = S[L[x]] + S[R[x]] + 1;
    x = y;
}
void rotate_right(int &x)
{
    int y = L[x];
    L[x] = R[y];
    R[y] = x;
    S[y] = S[x];
    S[x] = S[L[x]] + S[R[x]] + 1;
    x = y;
}

void maintain(int &p, bool flag)
{
    if(flag)//调整右边
    {
        if(S[R[R[p]]] > S[L[p]] rotate_left(p);
                else if(S[R[L[p]]] > S[L[p]])
        {
            rotate_right(R[p]);
                rotate_left(p);
            }
            else return;
            }
    else
    {
        if(S[L[L[p]]] > S[R[p]]) rotate_right(p);
        else if(S[L[R[p]]] > S[R[p]])
        {
            rotate_left(L[p]);
            rotate_right(p);
        }
        else return;
    }
    maintain(L[p], 0);
    maintain(R[p], 1);
    maintain(p, 0);
    maintain(p, 1);
}

void insert(int &p, int e)
{
    if(!p)
    {
        p = ++total;
        L[p] = R[p] = 0;
        A[p] = e; S[p] = 1;
        return;
    }
    S[p]++;
    if(e < A[p]) insert(L[p], e);
    else insert(R[p], e);
    maintain(p, k >= A[p]);
}

int getmin()
{
    for(int x = root; L[x]; x = L[x]);
    return A[x];
}
int getmax()
{
    for(int x = root; R[x]; x = R[x]);
    return A[x];
}
int kth(int &p, int k)
{
    int tmp = S[L[p]] + 1;
    if(k == tmp) return A[p];
    else if(k < tmp) return kth(L[p], k);
    else return kth(R[p], k – tmp);
}
```

4 Graph

4.1 Shortest path

4.1.1 Dijkstra

```
void dijkstra()
{
    memset(dist, 0, sizeof(dist));
    while(!Q.empty())
    {
        int x = Q.top().second; Q.pop();
        if(done[x]) continue;
        done[x] = 1;
        for(Link p = head[x]; p; p = p->next)
            if(dist[p->y] > dist[x] + p->z)
            {
                dist[p->y] = dist[x] + p->z;
                Q.push(make_pair(dist[p->y], p->y));
            }
    }
}
```

4.1.2 Spfa

```
void spfa()
{
    memset(dist, 0x3f, sizeof(dist));
    Q.push(S);//S为源点
    while(!Q.empty())
    {
        int x = Q.front();
        Q.pop(); inQ[x] = 0;
        for(Link p = head[x]; p; p = p->next)
            if(dist[p->y] > dist[x] + p->z)
            {
                dist[p->y] = dist[x] + p->z;
                if(!inQ[p->y])
                {
                    Q.push(p->y);
                    inQ[p->y] = 1;
                }
            }
    }
}
```

4.1.3 Floyd

```
void floyd()
{
    for(int k = 1; k <= n; k++) // 这里可以看作是一个加边的过程
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                map[i][j] = min(map[i][j], map[i][k] + map[k][j]);
}

// 最小环
```

```
void MinCircle()
{
    cap[] = map[];
    int circle = 0x3f3f3f3f;
    for(int k = 1; k <= n; k++)
    {
        for(int i = 1; i < k; i++)
            for(int j = i+1; j < k; j++)
                circle = min(circle, map[i][j] + cap[j][k]+cap[k][i]);
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                map[i][j] = min(map[i][j], map[i][k] + map[k][j]);
    }
    return circle == 0x3f3f3f3f ? -1 : circle;
}

// floyd判圈法 (大白书 p44)
void Circle()
{
    int ans = k;
    int k1 = k, k2 = k;
    do{
        k1 = next(k1);
        k2 = next(k2); ans = max(ans, k2);
        k2 = next(k2); ans = max(ans, k2);
    }while(k1 != k2);
    return ans;
}
```

4.2 Minimum Spanning Tree

4.2.1 Prim

```
void prime()
{
    memset(dist, 0, sizeof(dist));
    int res = 0;
    while(!Q.empty())
    {
        int x = Q.top().second;
        if(done[x]) {Q.pop(); continue;}
        res += Q.top().first;
        Q.pop();
        for(Link p = head[x]; p; p = p->next)
            if(dist[p->y] > p->z)
            {
                dist[p->y] = p->z;
                Q.push(make_pair(dist[p->y], p->y));
            }
    }
}
```

4.2.2 Kruskal

```
void prime()
{
    sort(edge, edge+Cnt, cmp);
    int res = 0;
    for(int i = 0; i < Cnt; i++)
    {
        if(getroot(edge[i].x) == getroot(edge[i].y)) continue;
        merge(edge[i].x, edge[i].y);
        res += edge[i].z;
    }
}
```

4.3 Tarjan - Strong Union

```
void dfs(int x)
{
    now[x] = low[x] = ++dfstime;
    hash[x] = 1;
    st.push(x); inst[x] = 1;
    for(int i = 1; i <= n; i++)
        if(map[x][i])
        {
            if(!hash[i])
            {
                dfs(i);
                low[x] = min(low[x], low[i]);
            }
            else if(inst[i]) low[x] = min(low[x], now[i]);
        }
    if(low[x] == now[x])
    {
        while(!st.empty())
        {
            int u = st.top();
            st.pop(); inst[u] = 0;
            belong[u] = number;
            if(u == x) break;
        }
        numer++;
    }
}
void tarjan()
{
    for(int i = 1; i <= n; i++)
        if(!hash[i]) dfs(i);
    if(!st.empty()) // 这是一个未知bug  栈中还会剩下一个强连通分量
    {
        while!st.empty())
        {
            int u = st.top();
            st.pop();
            belong[u] = number;
        }
        number++;
    }
}
```

4.4 LCA

4.4.1 Tarjan

```
// poj 1330 (changed something)
// LCA tarjan
#include <cstdio>
#include <cstring>

const int N = 10000 + 10;

int n;
struct Link{int y, idx; Link *next;}*head[N], *ask[N];
int tx, ty;
bool in[N], vis[N];
int f[N];
int ans[N]; // Query Answer

void inLink(int x, int y)
{
    Link *p = new Link;
    p -> y = y;
    p -> next = head[x];
    head[x] = p;
}
void inAsk(int x, int y, int idx)
{
    Link *p = new Link;
    p -> y = y;
    p -> idx = idx;
    p -> next = ask[x];
    ask[x] = p;
}

int getroot(int x)
{
    return f[x] == x ? x : f[x] = getroot(f[x]);
}

void LCA(int x)
{
    vis[x] = 1;
    f[x] = x;
    for(Link *p = ask[x]; p; p = p -> next)
        if(vis[p->y]) ans[p->idx] = getroot(p->y);
    for(Link *p = head[x]; p; p = p -> next)
        if(!vis[p->y])
        {
            LCA(p->y);
            f[p->y] = x;
        }
}

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        memset(head, 0, sizeof(head));
        memset(ask, 0, sizeof(ask));
        memset(in, 0, sizeof(in));
        memset(vis, 0, sizeof(vis));
        scanf("%d", &n);
        for(int i = 1; i <= n; i++) f[i] = i;
        for(int i = 1; i < n; i++)
        {
            int x, y;
            scanf("%d%d", &x, &y);
            inLink(x, y);
            in[y] = 1;
        }
        int q = 1;// the number of query
        for(int i = 1; i <= q; i++)
        {
            int x, y; scanf("%d%d", &x, &y);
            inAsk(x, y, i); inAsk(y, x, i);
        }
        int root = -1;
        for(int i = 1; i <= n; i++)
            if(!in[i]) {root = i; break;}
        LCA(root);
        for(int i = 1; i <= q; i++)
            printf("%d\n", ans[i]);
    }
    return 0;
}
```

4.4.2 Doubling Algorithm

还不会...

4.5 Bipartite Graph

4.5.1 Maximal Matching - The Hungarian algorithm

```
int ttt = 0; // 全局时间戳变量

```

```
bool search(int x)
{
    for(int i = 1; i <= m; i++)
        if(map[x][i] && vis[i] != ttt)
        {
            vis[i] = ttt;
            if(pre[i] == -1 || search(pre[i]))
            {
                pre[i] = x;
                return 1;
            }
        }
    return 0;
}

int match()
{
    int res = 0;
    for(int i = 1; i <= n; i++)
    {
        ++ttt; // 这里不用memset节省时间
        res += search(i);
    }
    return res;
}
```

4.5.2 Optimal Matching - KM

不会... 用费用流解决

4.6 Network flow

4.6.1 Maximum Flow - isap

```
//   h[x] :   点 x 在第 h[x] 层
//   v[k] :   第 k 层有 v[k] 个点
int sap(int x, int flow)
{
    if(x == n) return flow;
    int res = 0;
    for(int i = S; i <= T; i++)
        if(g[x][i] && h[x] == h[i] + 1)
        {
            int t = sap(i, min(g[x][i], flow - res));
            res += t; g[x][i] -= t; g[i][x] += t;
            if(res == flow) return res;
            if(h[S] >= T) return res;
        }
    //if(h[S] >= T) return res;
    if((--v[h[x]]) == 0) h[S] = T;
    ++v[++h[x]];
    return res;
}
int main()
{
    v[0] = T;
    int maxflow = 0;
    while(h[S] < T) maxflow += sap(1, inf);
    reutrn 0;
}
```

4.6.2 Minimum Cost Maximum Flow - spfa

```
struct EG{int from,to,flow,cost,next;}edge[M];

void add_edge(int a,int b,int c,int d)
{
    edge[L]=(EG){a,b,c,+d,head[a]};
    head[a]=L++;
    edge[L]=(EG){b,a,0,-d,head[b]};
    head[b]=L++;
}

bool spfa()
{
    memset(inQ, 0, sizeof(inQ));
    memset(dist, 0x3f, sizeof(dist));
    dist[S] = 0;
    q.push(S);
    while(!q.empty())
    {
        int x = q.front();
        q.pop();
        inQ[x] = 0;
        for(int i = head[x]; i != -1; i = edge[i].next)
            if(edge[i].flow && dist[edge[i].to] > dist[x] + edge[i].cost)
            {
                pre[edge[i].to] = i;
                dist[edge[i].to] = dist[x] + edge[i].cost;
                if(!inQ[edge[i].to])
                {
                    inQ[edge[i].to] = 1;
                    q.push(edge[i].to);
                }
            }
    }
    return dist[T] != inf;
}
void MFMC()
{
    memset(head, -1, sizeof(head));
    建图调用 add_edge();

    int mincost = 0, maxflow = 0;
    while(spfa())
    {
        int res = inf;
        for(int i = T; i != S; i = edge[pre[i]].from)
        {
            res = min(res, edge[pre[i]].flow);
        }
        for(int i = T; i != S; i = edge[pre[i]].from)
        {
            edge[pre[i]].flow -= res;
            edge[pre[i] ^ 1].flow += res;
        }
        maxflow += res;
        mincost += res * dist[T];
    }
}
```

5 Geometry

5.1 Convex Hull

```
//��□°� µ�list[0~n-1]
//□°�½��stack[0~top-1]
Point list[Maxn];
int Stack[Maxn],top;
bool _cmp (Point p1,Point p2)
{
    double tmp=(p1-list[0])^(p2-list[0]);
    if (fuhao(tmp)>0) return true;
    else if (fuhao(tmp)==0&&fuhao(dist(p1,list[0])-dist(p2,list[0]))
       <=0)
        return true;
    else    return false;
}
void Graham(int n)
{
    Point p0;
    int k=0;
    p0=list[0];
    for (int i=1;i<n;++i)
    {
        if ((p0.y>list[i].y)||(p0.y==list[i].y&&p0.x>list[i].x))
        {
            p0=list[i];
            k=i;
        }
    }
    swap(list[k],list[0]);
    sort(list+1,list+n,_cmp);
    if (n==1)
    {
        top=1;
        stack[0]=0;
        return;
    }
    if (n==2)
    {
        top=2;
        stack[0]=0;
        stack[1]=1;
        return;
    }
    stack[0]=0;
    stack[1]=1;
    top=2;
    for (int i=2;i<n;++i)
    {
        while (top>1 && fuhao((list[stack[top-1]]-list[stack[top-2]])^(list[i]-list[stack[top-2]]))<=0)
            top--;
        stack[top++]=i;
    }
}
```

5.2 All

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <utility>
using std::max;
using std::min;
using std::sort;
using std::swap;
using std::pair;
using std::make_pair;
const double eps = 1e-8, inf = 1e20;
const double pi = 4.0 * atan(1.0);
#define Degree(_rad) (180.0 / pi * (_rad))

int fuhao(double x)
{
    if (fabs(x)<eps) return 0;
    if (x<0) return -1;
    else return 1;
}

///////////////////////////   Point && Vector ///////////////////////////
struct Point{
    double x, y;
    Point (){}
    Point (double _x,double _y):x(_x),y(_y){}
    void init(double a, double b) { x = a; y = b; }

    // basic calc
        bool operator == (const Point &b) const
        {
            return !fuhao(x - b.x) && !fuhao(y - b.y);
        }
        Point operator + (const Point &b) const
        {
            return Point(x + b.x, y + b.y);
        }
        Point operator - (const Point &b) const
        {
            return Point(x - b.x, y - b.y);
        }
        Point operator * (const double &b) const
        {
            return Point(x * b, y * b);
        }

        Point Rotate(Point p, double alpha) // alpha E [0, +oo) 逆时针
        {
            double x0 = p.x, y0 = p.y;
            double tx = x - x0, ty = y - y0;
            double nx = tx * cos(alpha) - ty * sin(alpha);
            double ny = tx * sin(alpha) + ty * cos(alpha);
            nx += x0; ny += y0;
            return Point(nx, ny);
        }

    // Vector
        double operator *(const Point &b)const
        {// Dot
            return x * b.x + y * b.y;
        }
        double operator ^ (const Point &b)const
        {// Cross
            return x * b.y - y * b.x;
        }
        double Abs() { return sqrt(x * x + y * y); }
};
double Dist(const Point &a, const Point &b) { return (a - b).Abs(); }
typedef Point Vector;

double Angle(Vector a, Vector b)
{
    return acos(a * b / a.Abs() / b.Abs());
}
Vector Get_H(Vector A)
{ // 求与向量垂直的单位向量    使用前确保不为0向量
```

```
    // A != Vector(0.0, 0.0);
    double L = A.Abs();
    return Vector(-A.y / L, A.x / L);
}

////////////////////////////      E - N - D      ////////////////////////////


////////////////////////////         Line        ////////////////////////////
struct Line{
    Point s,e;
    Line() {}
    Line(Point ss, Point ee)
    {
        s = ss; e = ee;
    }

    // 两直线的关系：重合0, 平行1, 相交2 并返回交点
    pair<int,Point> operator &(const Line &b) const
    {
        Point ans = s;
        if(fuhao((s-e)^(b.s-b.e))==0)
        {
            if (fuhao((s-b.e)^(b.s-b.e))==0)
                return make_pair(0,ans);//重合
            else return make_pair(1,ans);//平行
        }
        double t = ((s-b.s)^(b.s-b.e)) / ((s-e)^(b.s-b.e));
        ans.x += (e.x-s.x) * t;
        ans.y += (e.y-s.y) * t;
        return make_pair(2,ans);//相交
    }
};
////////////////////////////      E - N - D      ////////////////////////////

//判断线段相交
bool inter(Line l1,Line l2)
{
    return
    max(l1.s.x,l1.e.x) >= min(l2.s.x,l2.e.x) &&
    max(l1.s.y,l1.e.y) >= min(l2.s.y,l2.e.y) &&
    max(l2.s.x,l2.e.x) >= min(l1.s.x,l1.e.x) &&
    max(l2.s.y,l2.e.y) >= min(l1.s.y,l1.e.y) &&
    fuhao((l2.s-l1.e)^(l1.s-l1.e)) * fuhao((l2.e-l1.e)^(l1.s-l1.e))<=0 &&
    fuhao((l1.s-l2.e)^(l2.s-l2.e)) * fuhao((l1.e-l2.e)^(l2.s-l2.e))<=0;
}
//判断直线与线段相交
bool Seg_inter_line(Line l1,Line l2)//l1为直线 l2为线段
{
    return fuhao((l2.s-l1.e)^(l1.s-l1.e))*fuhao((l2.e-l1.e)^(l1.s-l1.e))<=0;
}
//点到直线距离
//返回点到直线最近的点
Point PointToLine(Point P,Line L)
{
    Point ans;
    double t=((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
    ans.x=L.s.x+(L.e.x-L.s.x)*t;
    ans.y=L.s.y+(L.e.y-L.s.y)*t;
    return ans;
}
//点到线段距离
//返回点到线段最近的点
Point NearestPointToLineSeg(Point P,Line L)
{
    Point ans;
    double t = ((P-L.s)*(L.e-L.s)) / ((L.e-L.s)*(L.e-L.s));
    if (t>=0&&t<=1)
    {
        ans.x = L.s.x + (L.e.x-L.s.x)*t;
        ans.y = L.s.y + (L.e.y-L.s.y)*t;
    }
    else {
        if (Dist(P,L.s)<Dist(P,L.e))
            ans = L.s;
        else    ans = L.e;
    }
    return ans;
}
//多边形面积
double CalcArea(Point p[],int n)
{
    double ans=0;
    for (int i=0;i<n;++i)
        ans+=(p[i]^p[(i+1)%n])/2;
    return fabs(ans);
}
//判断点在线段上
bool OnSeg(Point P,Line L)
{
    return
            fuhao((L.s-P)^(L.e-P))==0 &&
            fuhao((P.x-L.s.x)*(P.x-L.e.x))<=0 &&
            fuhao((P.y-L.s.y)*(P.y-L.e.y))<=0;
}
//三点求圆心坐标
Point waixin(Point a,Point b,Point c)
{
    double a1=b.x-a.x,b1=b.y-a.y,c1=(a1*a1+b1*b1)/2;
    double a2=c.x-a.x,b2=c.y-a.y,c2=(a2*a2+b2*b2)/2;
    double d=a1*b2-a2*b1;
    return Point(a.x+(c1*b2-c2*b1)/d,a.y+(a1*c2-a2*c1)/d);
}


///////////////////////////        Graham ///////////////////////////
//求凸包 点list[0~n-1]
//凸包结果Stack[0~top-1]
const int Maxn = 100;////////////////here!!
Point list[Maxn];           //////////?!?!?!?!补全Maxn !?!?!?!?!?!?!?!?!?!?!
int Stack[Maxn],top;
bool _cmp (Point p1,Point p2)
{
    double tmp=(p1-list[0])^(p2-list[0]);
    if (fuhao(tmp)>0) return true;
    else if (fuhao(tmp)==0&&fuhao(Dist(p1,list[0])-Dist(p2,list[0]))<=0)
        return true;
    else    return false;
}
void Graham(int n)
{
    Point p0;
    int k=0;
    p0=list[0];
    for (int i=1;i<n;++i)
    {
        if ((p0.y>list[i].y)||(p0.y==list[i].y&&p0.x>list[i].x))
        {
            p0=list[i];
            k=i;
        }
    }
    swap(list[k],list[0]);
    sort(list+1,list+n,_cmp);
    if (n==1)
    {
        top=1;
        Stack[0]=0;
        return;
    }
    if (n==2)
    {
        top=2;
        Stack[0]=0;
        Stack[1]=1;
        return;
    }
    Stack[0]=0;
    Stack[1]=1;
    top=2;
    for (int i=2;i<n;++i)
    {
        while (top>1 && fuhao((list[Stack[top-1]]-list[Stack[top-2]])^(list[i]-list[Stack[top-2]]))<=0)
            top--;
        Stack[top++]=i;
    }
}
////////////////////////////      E - N - D      ////////////////////////////


////////////////////////////         Area        ////////////////////////////
double PolygonArea(Point *pp, int nn) // pp[0, n-1]
{
    double ans_area = 0.0;
    for(int i = 1; i < nn-1; i++)
    {
        ans_area += (pp[i] - pp[0]) ^ (pp[i+1] - pp[0]);
    }
    return fabs(ans_area / 2);
}
////////////////////////////      E - N - D      ////////////////////////////

////////////////////////////      点在多边形内      ////////////////////////////
int isPointInPolygon(Point p, Point *poly, int nn)
{
    int w = 0;
    for(int i = 0; i < n; i++)
    {
        if(OnSeg(p, Line(poly[i], poly[(i+1)%n]))) return -1; // 边界上
        int k = fuhao((poly[(i+1)%n] - poly[i]) ^ (p - poly[i]));
        int d1 = fuhao(poly[i].y - p.y);
        int d2 = fuhao(poly[(i+1)%n].y - p.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn++;
        if(k < 0 && d1 > 0 && d2 <= 0) wn--;
    }
    if(wn != 0) return 1; //内部
    return 0; // 外部
}
////////////////////////////      E - N - D      ////////////////////////////


int main()
{
}
```

6 String Tools

6.1 KMP

```
void getnext(char *t)
{
    memset(next, 0, sizeof(next));
    next[0] = -1;
    int j = 0, k = -1;
    int len = strlen(t);
    while(j < len)
    {
        if(k == -1 || t[j] == t[k]) next[++j] = ++k;
        else k = next[k];
    }
}
void kmp()
{
    getnext();
    int lens = strlen(s), lent = strlen(t);
    int i = 0, j = 0;
    while(i < lens)
    {
        if(j == -1 || s[i] == t[j]) i++, j++;
        else j = next[j];
        if(j == lent) return j - lent;
    }
    return -1;
}
```

6.2 Suffix Array

```
int sa[N], rank[N];
int c[N], tmp[N];
int height[N];
bool cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}
void DA(int n, int m) //n为长度，m为字串的最大值。均为开区间
{
    int i, j, p, *x = rank, *y = tmp;
    memset(c, 0, sizeof(c));
    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(j = 1, p = 0; p < n; j *= 2, m = p)
    {
        for(p = 0, i = n - j; i < n; i++) y[p++] = i;
        for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;
        memset(c, 0, sizeof(c));
        for(i = 0; i < n; i++) c[ x[y[i]] ]++;
        for(i = 1; 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); x[sa[0]] = 0;
        for(i = 1, p = 1; i < n; i++)
            if(x[i]) x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
    for(i = 0; i < n; i++) rank[sa[i]] = i;
    int k = 0;
    for(i = 0; i < n; height[x[i++]] = k)
        for(k ? k-- : 0, j = sa[x[i] - 1]; s[i + k] == s[j + k]; k++);
}
int main()
{
    读入字串s，n=s的长度
    s[n++]=’$’;//最后加入一个不存在的值，排除超界情况
    DA(n,m);
}
```

7 Tools

7.1 BigInteger - C++

```
//程序中全部为正整数之间的操作
#include <cstdio>
#include <cstring>
#include <algorithm>
using std::max;

const int base = 10000; // 压4位

struct BigInt{
    int c[1000], len, sign;
    BigInt() { memset(c, 0, sizeof(c)); len = 1; sign = 0; }
    void Zero()
    {
        while(len > 1 && c[len] == 0) len--;
        if(len == 1 && c[len] == 0) sign = 0;
    }
    void writein(char *s)
    {
        int k = 1, L = strlen(s);
        for(int i = L-1; i >= 0; i--)
        {
            c[len] += (s[i]-'0') * k;
            k *= 10;
            if(k == base)
            {
                k = 1;
                len++;
            }
        }
    }
    void Read()
    {
        char s[5000] = {0};
        scanf("%s", s);
        writein(s);
    }
    void Print()
    {
        if(sign) printf("-");
        printf("%d", c[len]);
        for(int i = len-1; i >= 1; i--) printf("%04d", c[i]);
        printf("\n");
    }
    BigInt operator = (int a)
    {
        char s[100] = {0};
        sprintf(s, "%d", a);
        writein(s);
        return *this;
    }
    bool operator > (const BigInt &b)
    {
        if(len != b.len) return len > b.len;
        for(int i = len; i >= 1; i--)
        {
            if(c[i] != b.c[i]) return c[i] > b.c[i];
        }
        return 0;
    }
    bool operator < (const BigInt &b)
    {
        if(len != b.len) return len < b.len;
        for(int i = len; i >= 1; i--)
        {
            if(c[i] != b.c[i]) return c[i] < b.c[i];
        }
        return 0;
    }
    bool operator == (const BigInt &b)
    {
        if(len != b.len) return 0;
        for(int i = 1; i <= len; i++)
            if(c[i] != b.c[i]) return 0;
        return 1;
    }
    bool operator == (const int &a)
    {
        BigInt b; b = a;
        return *this == b;
    }
    BigInt operator + (const BigInt &b)
    {
        BigInt r; r.len = max(len, b.len) + 1;
        for(int i = 1; i <= r.len; i++)
        {
            r.c[i] += c[i] + b.c[i];
            r.c[i+1] += r.c[i] / base;
            r.c[i] %= base;
        }
        r.Zero();
        return r;
    }
    BigInt operator + (const int &a)
    {
        BigInt b; b = a;
        return *this + b;
    }
    BigInt operator - (const BigInt &b)
    {
        BigInt a, c;// a - c
        a = *this; c = b;
        if(a < c)
        {
            std::swap(a, c);
            a.sign = 1;
        }
        for(int i = 1; i <= len; i++)
        {
            a.c[i] -= c.c[i];
            if(a.c[i] < 0)
            {
                a.c[i] += base;
                a.c[i+1]--;
            }
        }
        a.Zero();
        return a;
    }
    BigInt operator - (const int &a)
    {
        BigInt b; b = a;
        return *this - b;
    }
    BigInt operator * (const BigInt &b)
    {
        BigInt r; r.len = len + b.len + 2;
        for(int i = 1; i <= len; i++)
        {
            for(int j = 1; j <= b.len; j++)
            {
                r.c[j+i-1] += c[i] * b.c[j];
            }
        }
        for(int i = 1; i <= r.len; i++)
        {
            r.c[i+1] += r.c[i] / base;
            r.c[i] %= base;
        }
        r.Zero();
        return r;
    }
    BigInt operator * (const int &a)
    {
        BigInt b; b = a;
        return *this * b;
    }
    BigInt operator / (BigInt b)//整除
    {
        BigInt t, r;
        if(b == 0) return r;
        r.len = len;
        for(int i = len; i >= 1; i--)
        {
            t = t * base + c[i];
            int div;
            //--------try--------
                int up = 10000, down = 0;
                while(up >= down)
                {
                    int mid = (up + down) / 2;
                    BigInt ccc ; ccc = b * mid;
                    if(ccc > t) up = mid - 1;
                    else {
                        down = mid + 1;
                        div = mid;
                    }
                }
            //--------end--------
            r.c[i] = div;
            t = t - b * div;
        }
        //最后的t为余数，要用的自己想办法传出去
        r.Zero();
        return r;
    }
    BigInt operator / (const int &a)
    {
        BigInt b; b = a;
        return *this / b;
    }
    BigInt operator % (const BigInt &b)
    {//其实可以复制上面除法的，这里换一种写法
        return *this - *this / b * b;
    }
    BigInt operator % (const int &a)
    {
        BigInt b; b = a;
        return *this % b;
    }
};

int main()
{
    return 0;
}
```

7.2 C char*

```
头文件cstring
strlen(s);//获取长度  O(N)
strcpy(a+2,b+1)//从b+1开始全部赋值给a+2开始的字符串
strncpy(a+2,b+1,2)//从b+1开始赋值2个给a+2开始的字符串
strcmp(a,b)//比较a和b的大小，相等返回0，a>b返回正整数
strcat(a,b)//相当于string类的 a += b;
strstr(a,b)-a;//返回b在a中第一次出现的位置，不存在返回NULL(即0)，由于-a,所以最后应该是-a
```

7.3 C++ std::string

```
//====初始化====
头文件string并加上std::
string s(str);//相当于string s=str;
string s(cstr);//把char数组类型的字符串cstr作为s的初值
s.clear();//清空，相当于 s="";

//====长度====
s.length();//获取s的长度，O(1)
s.size();//一样

//====插入删除====
s.insert(2, "a"); //在s的位置2插入string类字符串"a"
s.erase(2, 3); //从s的位置2开始删除3个字符

//====查找====
s.find("abc");//查找字符串"abc"在s中第一次出现的位置（据说是KMP实现的）
//s="aabcc"; printf("%d %d\n",(int)s.find("abc"),(int)s.find("aabb"));
//上一行程序应输出 1 -1 （若没找到必须强行转换为int才为 -1 ）
```

7.4 Batch test

7.4.1 @Linux

```
mkdata=mk
filea=a
fileb=b

g++ $mkdata.cpp -o $mkdata
```

```
g++ $filea.cpp –o $filea
g++ $fileb.cpp –o $fileb
cas=0
while true; do
    ./$mkdata > $filea.in
    ./$filea < $filea.in > $filea.out
    ./$fileb < $filea.in > $fileb.out
    if ! diff $filea.out $fileb.out
    then
        echo "␣Wrong␣Answer"
        break
    fi
    echo $((cas=cas+1)) "␣Accepted"
done
```

7.4.2 @Windows

```
:loop
    mk > A.in
    A < A.in > A.out
    p < A.in > p.out
    fc A.out p.out
    if errorlevel 1 goto end
    goto loop
:end
    pause
```