ACM-ICPC Template

GuessEver

Last Modify at November 9, 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 RMQ - st

```
int _rmq[N][30], Log[你想开多大就开多大];
for(int i = 2; i < 随便; i++) Log2[i] = Log2[i>>1] + 1;
void init_RMQ(int *_orig) // [1, n]
{
    for(int i = 1; i <= n; i++) _rmq[i][0] = _orig[i];
    for(int j = 1; j <= Log2[n]; j++)
        for(int i = 1; i <= n + 1 - (1 << j); i++)
            _rmq[i][j] = std::max(_rmq[i][j-1], _rmq[i+(1<<(j-1))][j-1]);
}
int query_RMQ(int l, int r) // max{x E [l, r]}
{
    int k = Log2[r - l + 1];
    return std::max(_rmq[l][k], _rmq[r-(1<<k)+1][k]);
}
```

1.5 数位 dp

```
#include <cstdio>
#include <cstring>
#include <algorithm>
// calculate the number of numbers in [l, r] which not contain '4' or
    '62'
long long l, r;
int k;
int L[100], R[100];
long long f[100][2][2][10];

int predo(long long a, int *num)
{
    int len = 0;
    do num[++len] = a % 10; while(a /= 10);
    return len;
}

long long calc(int pos, bool d, bool u, int pre)
{
    if(pos == 0) return 1;
    long long &res = f[pos][d][u][pre];
    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(i == 4 || (pre == 6 && i == 2)) continue;
        res += calc(pos-1, d && i == L[pos], u && i == R[pos], i);
    }
    return res;
}

int main()
{
    while(scanf("%lld%lld", &l, &r) == 2 && (l || r))
    {
        memset(f, -1, sizeof(f));
        memset(L, 0, sizeof(L));
        memset(R, 0, sizeof(R));
        int len = std::max(predo(l, L), predo(r, R));
        printf("%lld\n", calc(len, 1, 1, 0));
    }
    return 0;
}
```

1.6 状压 dp

1.6.1 枚举子集

```
for(int st = S; st; st = (st - 1) & S) ;
```

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

```
namespace prime{
    const int N = 10000000 + 10;
    int pri[N], h[N], Cnt;
    void make(int maxp) // O(2*N)
    {
        for(int i = 2; i <= maxp; i++)
        {
            if(!h[i]) pri[Cnt++] = i;
            for(int j = 0; j < Cnt && 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);
}
```

2.3 Fast Power

```
//x^y  % mod
int mul(int x, LL y, int mod) // 递归
{
    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) // 非递归
{
    int s = 1;
    int ss = x;
    while(y)
    {
        if(y & 1) s = s * ss;
        y /= 2;
        ss *= ss;
    }
    return s;
}
```

2.4 约瑟夫环、丢手绢问题

```
#include <cstdio>
//UVALive 4727
int n, m;

int Joseph(int totalPeople, int nextNumber, int startIndex, int lastIdx)
{ // All based on 0_Index , the Answer is the last `lastIdx` to leave
    int now = (nextNumber - 1) % lastIdx + (startIndex - nextNumber);
    for(int i = lastIdx + 1; i <= totalPeople; i++)
        now = (now + nextNumber) % i;
    return now;
}

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        scanf("%d%d", &n, &m);
        printf("%d␣%d␣%d\n", Joseph(n, m, m, 3)+1, Joseph(n, m, m, 2)+1, Joseph(n, m, m, 1)+1);
    }
    return 0;
}
```

2.5 康拓展开 Cantor

```
#include <cstdio>
#include <cstring>

int fac[10], a[10];

bool Read(int *p)
{
    for(int i = 0; i < 9; i++)
    {
        char chtmp;
        if(scanf("␣%c", &chtmp) != 1) return 0;
        p[i] = chtmp == 'x' ? 0 : chtmp - '0';
    }
    return 1;
}

int Cantor(int *p) // Eight puzzle status -> Integer
{
    int res = 0;
    for(int i = 0; i < 9; i++)
    {
        int cnt = 0;
        for(int j = i + 1; j < 9; j++)
```

```
            if(p[j] < p[i]) cnt++;
        res += cnt * fac[9 - i - 1];
    }
    return res;
}

bool used[10] = {0};
int getRank(int r)
{
    for(int i = 0, j = 0; i < 9; i++)
    {
        if(!used[i] && j == r) return i;
        if(!used[i]) j++;
    }
}
void getStatus(int cantor, int *p) // Integer -> Eight puzzle status
{
    memset(used, 0, sizeof(used));
    for(int i = 0; i < 9; i++)
    {
        p[i] = getRank(cantor / fac[9 - i - 1]);
        used[p[i]] = 1;
        cantor %= fac[9 - i - 1];
    }
}

void PRINT(int *p)
{
    int hash = Cantor(p);
    printf("Cantor␣value␣=␣%d\n", hash);
    getStatus(hash, p);
    printf("Cantor␣Status␣=␣");
    for(int i = 0; i < 9; i++) printf("%d", p[i]); puts("");
}

int main()
{
    fac[0] = 1; for(int i = 1; i < 10; i++) fac[i] = fac[i-1] * i;
    while(Read(a)) PRINT(a);
    return 0;
}
```

3 Datastructure

3.1 带权并查集

```
#include <cstdio>
#include <cstdlib>

const int N = 100000 + 10;

```

```
int n, f[N], g[N];

int getroot(int x)
{
    if(f[x] == x) return x;
    int tmp = getroot(f[x]);
    g[x] += g[f[x]]; // update the value
    return f[x] = tmp;
}

void merge(int x, int y) // merge x's set and y's set
{ // Guarantee that the x must be the root of its set, which means x == getroot(x) is true, but it may not be same for y
    int fy = getroot(y);
    g[x] += g[y] + abs(x - y) % 1000; // update the value
    f[x] = fy;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) f[i] = i;
    char op; int x, y;
    while(scanf("␣%c", &op) == 1 && op != 'O')
    {
        if(op == 'I')
        {
            scanf("%d%d", &x, &y);
            if(getroot(x) == getroot(y)) continue;
            merge(x, y);
        }
        else{
            scanf("%d", &x);
            getroot(x); // !!! update the value of x before output
            printf("%d\n", g[x]);
        }
        //for(int i = 1; i <= n; i++) printf("%d ", f[i]); puts("");
    }
    return 0;
}
```

3.2 手写 Heap

```
#include <cstdio>
#include <algorithm>

const int N = 250000;

int n, a[N], x, size = 0;

void update(int i)
{
    while(i > 1 && a[i] > a[i/2])
    {
        std::swap(a[i], a[i/2]);
        i /= 2;
    }
}

void pop()
{
    int i = 1; a[i] = 0;
    while(i * 2 <= size && (a[i] < a[i*2] || a[i] < a[i*2+1]))
    {
        if(i * 2 == size || (i * 2 < size && a[i*2] >= a[i*2+1]))
        {
            a[i] = a[i*2];
            a[i*2] = 0;
            i = i * 2;
        }
        else {
            a[i] = a[i*2+1];
            a[i*2+1] = 0;
            i = i * 2 + 1;
        }
    }
    a[i] = a[size]; size--;
    update(i);
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)
    {
        scanf("%d", &x);
        a[++size] = x;
        update(size);
    }
    for(int i = 1; i <= n; i++)
    {
        printf("%d␣", a[1]);
        pop();
    }
    return 0;
}
```

3.3 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;
}
```

3.4 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);
}
```

3.5 Treap

3.5.1 @ Array

```
struct treap {
    const int N = 100000 + 10;
    int L[N*20], R[N*20], S[N*20], fix[N*20], A[N*20];
    int root, total;
    void rotate_left(int &p)
    {
        int tmp = R[p];
        R[p] = L[tmp];
        int zsize = S[L[tmp]];
        S[p] = S[p] - S[tmp] + zsize;
        L[tmp] = p;
        S[tmp] = S[tmp] - zsize + S[p];
        p = tmp;
    }
    void rotate_right(int &p)
    {
        int tmp = L[p];
        L[p] = R[tmp];
        int zsize = S[R[tmp]];
        S[p] = S[p] - S[tmp] + zsize;
        R[tmp] = p;
        S[tmp] = S[tmp] - zsize + S[p];
        p = tmp;
    }
    void Insert(int &p, int x)
    {
        if(!p)
        {
            p = ++total;
            L[p] = R[p] = 0;
            S[p] = 1;
            fix[p] = rand();
            A[p] = x;
            return;
        }
        S[p]++;
        if(x < A[p])
        {
            Insert(L[p], x);
            if(fix[L[p]] > fix[p]) rotate_right(p);
        }
        else {
            Insert(R[p], x);
            if(fix[R[p]] > fix[p]) rotate_left(p);
        }
    }
    int Delete_min(int &p)
    {
        S[p]--;
        if(!L[p])
        {
            int value = A[p];
            p = R[p];
            return value;
        }
        else return Delete_min(L[p]);
    }
    void Delete(int &p, int x)
    {
        if(!p) return;
        S[p]--;
        if(x < A[p]) Delete(L[p], x);
        else if(x > A[p]) Delete(R[p], x);
        else {
            if(!L[p] && !R[p]) p = 0;
            else if(!L[p] || !R[p])
            {
                if(!L[p]) p = R[p];
                else p = L[p];
            }
            else A[p] = Delete_min(R[p]);
        }
    }
    int Count_leq(int &p, int x)
    {
        if(!p) return 0;
        if(A[p] <= x) return S[L[p]] + 1 + Count_leq(R[p], x);
        else return Count_leq(L[p], x);
    }
    int Count_geq(int &p, int x)
    {
        if(!p) return 0;
        if(A[p] >= x) return S[R[p]] + 1 + Count_geq(L[p], x);
        else return Count_geq(R[p], x);
    }
    int Find_kth(int &p, int k)
    {
        if(k == S[L[p]] + 1) return A[p];
        if(k <= S[L[p]]) return Find_kth(L[p], k);
        else return Find_kth(R[p], k - S[L[p]] - 1);
    }
};
```

3.5.2 @ Pointer

```
struct treap {
    struct Treap{
        int fix, key, size;
        Treap *left, *right;
        Treap() { fix = key = size = left = right = 0; }
    }*root, *null;

    void init()
    {
        null = new Treap;
        root = null;
    }
    void rotate_left(Treap *&p)
    {
        Treap *tmp = p -> right;
        p -> right = tmp -> left;
        int zsize = tmp -> left -> size;
        p -> size = p -> size - tmp -> size + zsize;
        tmp -> left = p;
        tmp -> size = tmp -> size - zsize + p -> size;
        p = tmp;
    }
    void rotate_right(Treap *&p)
    {
        Treap *tmp = p -> left;
        p -> left = tmp -> right;
        int zsize = tmp -> right -> size;
        p -> size = p -> size - tmp -> size + zsize;
        tmp -> right = p;
        tmp -> size = tmp -> size - zsize + p -> size;
        p = tmp;
    }

    void Insert(Treap *&p, int x)
    {
        if(p == null)
        {
            p = new Treap;
            p -> fix = rand();
            p -> key = x;
            p -> size = 1;
            p -> left = null;
            p -> right = null;
            return;
        }
        if(x < p -> key)
        {
            Insert(p -> left, x);
            p -> size++;
            if(p -> left -> fix > p -> fix) rotate_right(p);
        }
        else {
            Insert(p -> right, x);
            p -> size++;
            if(p -> right -> fix > p -> fix) rotate_left(p);
        }
    }
    int Delete_min(Treap *&p)
    {
        p -> size--;
        if(p -> left == null)
        {
            int value = p -> key;
            p = p -> right;
            return value;
        }
        else return Delete_min(p -> left);
    }
    void Delete(Treap *&p, int x) // Make sure that `x` is existed
    {
        if(p == null) return;
        p -> size--;
        if(x < p -> key) Delete(p -> left, x);
        else if(x > p -> key) Delete(p -> right, x);
        else { // delete *p
            if(p -> left == null && p -> right == null)
            {
                p = null;
            }
            else if(p -> left == null || p -> right == null)
            {
                if(p -> left == null)
                {
                    p = p -> right;
                }
                else { // p -> right == null
                    p = p -> left;
                }
            }
            else { // p -> left != null && p -> right != null
                p -> key = Delete_min(p -> right);
            }
        }
    }
    int Count_leq(Treap *&p, int x)
    {
        if(p == null) return 0;
        if(p -> key <= x) return p -> left -> size + 1 + Count_leq(p -> right, x);
        else return Count_leq(p -> left, x);
    }
    int Count_geq(Treap *&p, int x)
    {
        if(p == null) return 0;
        if(p -> key >= x) return p -> right -> size + 1 + Count_geq(p -> left, x);
        else return Count_geq(p -> right, x);
    }
    int Find_kth(Treap *&p, int x)
    {
        if(k == p -> left -> size + 1) return p -> key;
        if(k <= p -> left -> size) return Find_kth(p -> left, k);
        else return Find_kth(p -> right, k - p -> left -> size - 1);
    }
};
```

3.6 Size Balanced Tree

```
struct SBT {
    const int N = 100000 + 10;
    int A[N*20], S[N*20], L[N*20], R[N*20];
    int 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)
        {
            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 x)
    {
        if(!p)
        {
            p = ++total;
            L[p] = R[p] = 0;
            A[p] = x; S[p] = 1;
            return;
        }
        S[p]++;
        if(x < A[p]) Insert(L[p], x);
        else Insert(R[p], x);
        maintain(p, x >= A[p]);
    }
    int Delete_min(int &p)
    {
        S[p]--;
        if(!L[p])
        {
            int value = A[p];
            p = R[p];
            return value;
        }
        else return Delete_min(L[p]);
    }
    void Delete(int &p, int x)
    {
        if(!p) return;
        S[p]--;
        if(x < A[p]) Delete(L[p], x);
        else if(x > A[p]) Delete(R[p], x);
        else {
            if(!L[p] && !R[p]) p = 0;
            else if(!L[p] || !R[p])
            {
                if(!L[p]) p = R[p];
                else p = L[p];
            }
            else A[p] = Delete_min(R[p]);
        }
    }
    int Count_leq(int &p, int x)
    {
        if(!p) return 0;
        if(A[p] <= x) return S[L[p]] + 1 + Count_leq(R[p], x);
        else return Count_leq(L[p], x);
    }
    int Count_geq(int &p, int x)
    {
        if(!p) return 0;
        if(A[p] >= x) return S[R[p]] + 1 + Count_geq(L[p], x);
        else return Count_geq(R[p], x);
    }
    int Find_kth(int &p, int k)
    {
        if(k == S[L[p]] + 1) return A[p];
        if(k <= S[L[p]]) return Find_kth(L[p], k);
        else return Find_kth(R[p], k - S[L[p]] - 1);
    }
};
```

3.7 树链剖分 Heavy-Light Decomposition

```
// Solution: www.guessbug.com/problem/HDU/3966
#pragma comment(linker, "/STACK:1024000000,1024000000")
#include <cstdio>
#include <cstring>
#include <vector>
using std::vector;
// HDU 3966 : Increase or decrease a value on path [x - y] on a tree.
//            Query a value of a certain point
const int N = 50000 + 10;

int n, m, q, a[N];
vector<int> path[N];

// Heavy-Light Decomposition
int size[N], father[N], deep[N], heavy_son[N];
int top[N], segid[N], time_stamp;
void dfs1(int x, int fa, int deepth)
{
    size[x] = 1; father[x] = fa; deep[x] = deepth;
    for(vector<int>::iterator it = path[x].begin(); it != path[x].end(); it++)
    {
        if(*it == father[x]) continue;
        dfs1(*it, x, deepth + 1);
        size[x] += size[*it];
        if(size[*it] > size[heavy_son[x]]) heavy_son[x] = *it;
    }
}
void dfs2(int x, int topx)
{
    top[x] = topx;
    segid[x] = ++time_stamp;
    if(heavy_son[x]) dfs2(heavy_son[x], topx); // not leaf
    for(vector<int>::iterator it = path[x].begin(); it != path[x].end(); it++)
        if(*it != father[x] && *it != heavy_son[x])
            dfs2(*it, *it);
}
// Heavy-Light Decomposition ---- END

int add[N*4];
void pushDown(int p)
{
    add[p*2] += add[p];
    add[p*2+1] += add[p];
    add[p] = 0;
}
void modify(int p, int l, int r, int a, int b, int c)
{
    if(a <= l && b >= r)
    {
        add[p] += c;
        return;
    }
    int mid = l + (r - l) / 2;
    pushDown(p);
    if(a <= mid) modify(p*2, l, mid, a, b, c);
    if(b > mid) modify(p*2+1, mid+1, r, a, b, c);
}
int query(int p, int l, int r, int a)
{
    if(l == r && l == a) return add[p];
    int mid = l + (r - l) / 2;
    pushDown(p);
    if(a <= mid) return query(p*2, l, mid, a);
    else return query(p*2+1, mid+1, r, a);
}

void change(int a, int b, int c)
{
    while(top[a] != top[b])
    {
        if(deep[top[a]] < deep[top[b]]) std::swap(a, b);
        modify(1, 1, n, segid[top[a]], segid[a], c);
        a = father[top[a]];
    }
    if(deep[a] > deep[b]) std::swap(a, b);
    modify(1, 1, n, segid[a], segid[b], c);
}

int main()
{
    while(scanf("%d%d%d", &n, &m, &q) == 3)
    {
        time_stamp = 0;
        for(int i = 1; i <= n; i++)
        {
            size[i] = father[i] = heavy_son[i] = 0;
            deep[i] = top[i] = segid[i] = 0;
            path[i].clear();
        }
        for(int i = 1; i <= n; i++) scanf("%d", &a[i]);
        for(int i = 1; i <= m; i++)
        {
            int x, y; scanf("%d%d", &x, &y);
            path[x].push_back(y);
            path[y].push_back(x);
        }
        dfs1(1, 0, 1);
        dfs2(1, 1);
        memset(add, 0, sizeof(add));
        for(int i = 1; i <= n; i++) change(i, i, a[i]);
        while(q--)
        {
            char op; scanf("␣%c", &op);
            if(op == 'I' || op == 'D')
            {
                int a, b, c; scanf("%d%d%d", &a, &b, &c);
                if(op == 'I') change(a, b, c);
                else change(a, b, -c);
            }
            else {
                int x; scanf("%d", &x);
                printf("%d\n", query(1, 1, n, segid[x]));
            }
        }
    }
    return 0;
}
```

3.8 三维偏序 - CDQ 分治

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define lowbit(_X) ((_X)&(-(_X)))
// SPOJ LIS2
const int N = 100000 + 10;

int n, f[N], idx[N], hash[N];
struct Node{
    int x, y, z;
    void Read(int i)
    {
        scanf("%d%d", &y, &z);
        x = i; f[i] = 1; idx[i] = i;
    }
}a[N];
int maxp;
int c[N]; // tree Array

bool cmpx(int i, int j) { return a[i].x < a[j].x; }
bool cmpy(int i, int j) { return a[i].y < a[j].y; }
bool cmpz(int i, int j) { return a[i].z < a[j].z; }

void discrete()
{
    std::sort(idx+1, idx+1+n, cmpy); maxp = 0;
    for(int i = 1; i <= n; i++)
    {
        if(i == 1 || a[idx[i]].y != a[idx[i-1]].y) hash[idx[i]] = ++maxp;
        else hash[idx[i]] = maxp;
    }
    for(int i = 1; i <= n; i++) a[idx[i]].y = hash[idx[i]];
    std::sort(idx+1, idx+1+n, cmpz); maxp = 0;
    for(int i = 1; i <= n; i++)
    {
        if(i == 1 || a[idx[i]].z != a[idx[i-1]].z) hash[idx[i]] = ++maxp;
        else hash[idx[i]] = maxp;
    }
    for(int i = 1; i <= n; i++) a[idx[i]].z = hash[idx[i]];
}

void insert(int a, int x)
{
    for( ; a <= maxp; a += lowbit(a)) c[a] = std::max(c[a], x);
}
int query(int a) // [1, a]
{
    int res = 0;
    for( ; a > 0; a -= lowbit(a)) res = std::max(res, c[a]);
    return res;
}

void solve(int l, int mid, int r)
{
    std::sort(&idx[l], &idx[mid]+1, cmpy);
    std::sort(&idx[mid+1], &idx[r]+1, cmpy);
    // [l, mid] .. calculated ok
    // now calculating [mid+1, r]
    // f[i] = max{f[j]} + 1;
    int j = l;
    for(int i = mid + 1; i <= r; i++)
    {
        for( ; j <= mid && a[idx[j]].y < a[idx[i]].y; j++)
            insert(a[idx[j]].z, f[a[idx[j]].x]);
        int tmp = query(a[idx[i]].z - 1);
```

```
            if(tmp + 1 > f[a[idx[i]].x]) f[a[idx[i]].x] = tmp + 1;
    }
    //memset(c, 0, sizeof(c));
    for(int i = l; i <= mid; i++)
    {
        int b = a[idx[i]].z;
        for( ; b <= maxp; b += lowbit(b)) c[b] = 0;
    }
    std::sort(&idx[mid+1], &idx[r]+1, cmpx);
    // CDQ(mid+1, r) next, so sort back it
}

void CDQ(int l, int r)
{
    if(l == r) return;
    int mid = l + (r - l) / 2;
    CDQ(l, mid);
    solve(l, mid, r);
    CDQ(mid + 1, r);
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) a[i].Read(i);
    discrete();
    std::sort(idx+1, idx+1+n, cmpx);
    CDQ(1, n);
    int res = 1;
    //for(int i = 1; i <= n; i++) printf("%d ", f[i]); puts("");
    for(int i = 1; i <= n; i++) if(f[i] > res) res = f[i];
    printf("%d\n", res);
    return 0;
}
```

4 Graph

4.1 Shortest path

4.1.1 Dijkstra

```
void dijkstra()
{
    memset(dist, 0x3f, sizeof(dist));
    dist[1] = 0; Q.push(make_pair(0, 1));
    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(inQ, 0, sizeof(inQ));
    memset(dist, 0x3f, sizeof(dist));
    dist[S] = 0; Q.push(S); inQ[S] = 1; //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

```
int prim()
{
    memset(dist, 0x3f, sizeof(dist));
    dist[1] = 0; Q.push(make_pair(0, 1));
    int res = 0;
    while(!Q.empty())
    {
        int x = Q.top().second; Q.pop();
        if(done[x]) continue;
        res += dist[x]; done[x] = 1;
        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));
            }
    }
    return res;
}
```

4.2.2 Kruskal

```
int kruskal()
{
    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;
        f[getroot(edge[i].x)] = getroot(edge[i].y);
        res += edge[i].z;
    }
    return res;
}
```

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

```
#include <cstdio>
#include <cstring>
#include <algorithm>
// POJ 1330   LCA_Doubling Algorithm
const int N = 10000 + 10;

const int UPDeepth = 14;
int n;
struct Link{
    int y;
    Link *next;
}*head[N];
bool in[N];
int ancient[N][UPDeepth+1];
int deep[N];

```

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

void dfs(int x, int deepth, int father)
{
    deep[x] = deepth;
    ancient[x][0] = father;
    for(Link *p = head[x]; p; p = p -> next)
        dfs(p -> y, deepth + 1, x);
}

void getLCA()
{
    for(int i = 1; i <= n; i++)
        if(!in[i]) dfs(i, 1, 0);
    for(int j = 1; j <= UPDeepth; j++)
        for(int i = 1; i <= n; i++)
            ancient[i][j] = ancient[ancient[i][j-1]][j-1];
}

int LCA(int x, int y)
{
    if(deep[x] > deep[y]) std::swap(x, y); // deep[x] <= deep[y]
    for(int j = UPDeepth; j >= 0; j--)
        if(deep[x] <= deep[ancient[y][j]]) y = ancient[y][j];
    if(x == y) return x;
    for(int j = UPDeepth; j >= 0; j--)
        if(ancient[x][j] != ancient[y][j])
        {
            x = ancient[x][j];
            y = ancient[y][j];
        }
    return ancient[y][0];
}

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        memset(head, 0, sizeof(head));
        memset(ancient, 0, sizeof(ancient));
        memset(in, 0, sizeof(in));
        memset(deep, 0, sizeof(deep));
        scanf("%d", &n);
        for(int i = 1; i < n; i++)
        {
            int x, y; scanf("%d%d", &x, &y);
            inLink(x, y); in[y] = 1;
        }
        getLCA();
        int x, y; scanf("%d%d", &x, &y);
        printf("%d\n", LCA(x, y));
    }
    return 0;
}
```

4.5 Bipartite Graph

4.5.1 Maximal Matching - The Hungarian algorithm

```
int timeStamp = 0;
int n, m, g[N][N];
int vis[N], pre[N];

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

int maxMatch()
{
    int res = 0;
    memset(pre, -1, sizeof(pre));
    for(int i = 1; i <= n; i++)
    {
        ++timeStamp;
        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

```
#include <cstdio>
#include <algorithm>

const int N = 200 + 10;

int n, m, g[N][N];
int v[N], h[N];
int S, T;

int sap(int x, int flow)
{
    if(x == T) 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, std::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()
{
    scanf("%d%d", &m, &n); // m = number of edges, n = number of points
    for(int i = 1; i <= m; i++)
    {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        g[x][y] += z;
    }
    v[0] = T; S = 1; T = n; // all idx started from `1`
    int maxflow = 0;
    while(h[S] < T) maxflow += sap(1, 0x3f3f3f3f);
    printf("%d\n", maxflow);
    return 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 BASICS

```
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <vector>
using std::vector;

const double EPS = 1e-10, INF = 1e20;
const double PI = acos(-1.0);
const int sign(const double &x) {
    if(fabs(x) < EPS) return 0;
    return x < 0 ? -1 : 1;
}
const int dcmp(const double &x, const double &y) {
    return sign(x - y);
}
const double toDegree(const double &alpha) {
    return alpha * 180.0 / PI;
}
const double toRad(const double &alpha) {
    return alpha * PI / 180.0;
}

struct Point {
    double x, y;
    Point() {}
    Point(const double &_x, const double &_y) { x = _x; y = _y; }
    void Read() { scanf("%lf%lf", &x, &y); }
    bool operator < (const Point &b) const {
        if(dcmp(x, b.x) == 0) return dcmp(y, b.y) < 0;
        return dcmp(x, b.x) < 0;
    }
    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 &k) const {
        return Point(x * k, y * k);
    }
    Point operator / (const double &k) const {
        return Point(x / k, y / k);
    }
    double operator * (const Point &b) const {
        return x * b.x + y * b.y;
    }
    double operator ^ (const Point &b) const {
        return x * b.y - y * b.x;
    }
```

```cpp
    bool operator == (const Point &b) const {
        return dcmp(x, b.x) == 0 && dcmp(y, b.y) == 0;
    }
    double Abs() const {
        return sqrt(x * x + y * y);
    }
    Point Rotate(const Point &o, const double &alpha) const {
        Point z = *this - o;
        double nx = z.x * cos(alpha) - z.y * sin(alpha);
        double ny = z.x * sin(alpha) + z.y * cos(alpha);
        return Point(nx, ny) + o;
    }
    double Angle() const {
        return atan2(y, x);
    }
};
typedef Point Vector;
// 单位法向量
const Vector getNormalVector(const Vector &P) {
    double L = P.Abs(); // `L` CANNOT BE `0` !!!!
    return Vector(-P.y / L, P.x / L);
}
// 两向量夹角
const double getAngle(const Vector &a, const Vector &b) {
    return acos(a * b / a.Abs() / b.Abs());
}

struct Line {
    Point s, e;
    Line() {}
    Line(const Point &_s, const Point &_e) { s = _s; e = _e; }
};
typedef Line Segment;
// 判断两直线的关系
const int getRelationBetweenLines(const Line &L1, const Line &L2) {
    if(sign((L1.e - L1.s) ^ (L2.e - L2.s)) == 0) {
        if(sign((L2.s - L1.s) ^ (L2.e - L1.s)) == 0) return 0; // coincidence
        else return 1; // parallel
    }
    return 2; // intersection
}
// 直线交点
const Point getLineIntersection(const Line &L1, const Line &L2) {
    Vector L1v = L1.e - L1.s, L2v = L2.e - L2.s;
    Vector u = L1.s - L2.s;
    double t = (L2v ^ u) / (L1v ^ L2v);
    return L1.s + L1v * t;
}
// 点到直线的距离
const double getDistanceFromPointToLine(const Point &P, const Line &L) {
    Vector v1 = L.e - L.s, v2 = P - L.s;
    return fabs(v1 ^ v2) / v1.Abs();
}
// 点到直线最近的点
const Point nearestPointToLine(const Point &P, const Line &L) {
    Point P2 = P + getNormalVector(L.e - L.s);
    return getLineIntersection(Line(P, P2), L);
}
// 点到线段的距离
const double getDistanceFromPointToSegment(const Point &P, const Segment &L) {
    if(L.s == L.e) return (P - L.s).Abs();
    Vector v1 = L.e - L.s, v2 = P - L.s, v3 = P - L.e;
    if(sign(v1 * v2) < 0) return v2.Abs();
    if(sign(v1 * v3) > 0) return v3.Abs();
    return fabs(v1 ^ v2) / v1.Abs();
}
// 点到线段最近的点
const Point nearestPointToSegment(const Point &P, const Segment &L) {
    if(L.s == L.e) return L.s;
    Vector v1 = L.e - L.s, v2 = P - L.s, v3 = P - L.e;
    if(sign(v1 * v2) < 0) return L.s;
    if(sign(v1 * v3) > 0) return L.e;
    return nearestPointToLine(P, L);
}
// 判断点在线端上
const bool isPointOnSegment(const Point &P, const Segment &L) {
    Vector v1 = P - L.s, v2 = P - L.e;
    return sign(v1 ^ v2) == 0 && sign(v1 * v2) < 0;
}
// 判断线段相交
const bool isSegmentIntersection(const Segment &L1, const Segment &L2) {
    return std::max(L1.s.x, L1.e.x) >= std::min(L2.s.x, L2.e.x)
        && std::max(L1.s.y, L1.e.y) >= std::min(L2.s.y, L2.e.y)
        && std::max(L2.s.x, L2.e.x) >= std::min(L1.s.x, L1.e.x)
        && std::max(L2.s.y, L2.e.y) >= std::min(L1.s.y, L1.e.y)
        && sign((L1.e - L1.s) ^ (L2.s - L1.s)) * sign((L1.e - L1.s) ^ (L2.e - L1.s)) <= 0
        && sign((L2.e - L2.s) ^ (L1.s - L2.s)) * sign((L2.e - L2.s) ^ (L1.e - L2.s)) <= 0;
}
// 判断直线和线段相交
const bool isLineSegmentIntersection(const Line &L1, const Segment &L2) {
    return sign((L1.s - L2.s) ^ (L1.e - L2.s)) * sign((L1.s - L2.e) ^ (L1.e - L2.e)) <= 0;
}

struct Circle {
    Point o; double r;
    Circle() {}
    Circle(const Point &_o, const double &_r) { o = _o; r = _r; }
    Point getPoint(const double &alpha) const {
        return Point(o.x + r * cos(alpha), o.y + r * sin(alpha));
    }
};
// 直线和圆的交（切）点，返回交点个数，p1和p2为两个交点
const int getLineCircleIntersection(const Line &L, const Circle &C, Point &p1, Point &p2) {
    double d = getDistanceFromPointToLine(C.o, L);
    if(dcmp(d, C.r) > 0) return 0;
    Point P = nearestPointToLine(C.o, L);
    if(dcmp(d, C.r) == 0) {
        p1 = p2 = P;
        return 1;
    }
    Vector v = L.e - L.s; v = v / v.Abs();
    double length = sqrt(C.r * C.r - d * d);
    p1 = P + v * length; p2 = P - v * length;
    return 2;
}
// 两个圆的交点，返回交点个数，p1和p2为两个交点
const int getCircleIntersection(const Circle &C1, const Circle &C2, Point &p1, Point &p2) {
    double d = (C1.o - C2.o).Abs();
    if(sign(d) == 0) {
        if(dcmp(C1.r, C2.r) == 0) return -1; // 重合
        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 = (C2.o - C1.o).Angle();
    double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
    p1 = C1.getPoint(a - da); p2 = C1.getPoint(a + da);
    if(p1 == p2) return 1; else return 2;
}
const int getRelationBetweenCircles(const Circle &C1, const Circle &C2) {
    double d = (C1.o - C2.o).Abs(), r1 = C1.r, r2 = C2.r;
    if(sign(d) == 0) {
        // 0 重合 - d == 0 && r1 == r2
        if(dcmp(r1, r2) == 0) return 0;
        // 1 同心圆 - d == 0 && r1 != r2
        else return 1;
    }
    // 2 内含 - |r1 - r2| > d
    if(dcmp(fabs(r1 - r2), d) > 0) return 2;
    // 3 内切 - |r1 - r2| == d
    if(dcmp(fabs(r1 - r2), d) == 0) return 3;
    // 4 相交 - r1 + r2 > d && |r1 - r2| < d
    if(dcmp(r1 + r2, d) > 0 && dcmp(fabs(r1 - r2), d) < 0) return 4;
    // 5 外切 - r1 + r2 == d
    if(dcmp(r1 + r2, d) == 0) return 5;
    // 6 相离 - r1 + r2 < d
    if(dcmp(r1 + r2, d) < 0) return 6;
```

```
}

// 三角形外接圆
const Circle getCircumscribedCircle(const Point &A, const Point &B, const Point &C) {
    Point AB = (A + B) / 2, AC = (A + C) / 2;
    Vector NAB = getNormalVector(B - A);
    Vector NAC = getNormalVector(C - A);
    Point O = getLineIntersection(Line(AB, AB + NAB), Line(AC, AC + NAC));
    return Circle(O, (O - A).Abs());
}
// 角BAC的角平分线
const Line getAngleDividingLine(const Point &B, const Point &A, const Point &C) {
    Vector AB = B - A, AC = C - A;
    return Line(A, C.Rotate(A, ((B - A).Angle() - (C - A).Angle()) / 2));
}
// 三角形内接圆
const Circle getInscribedCircle(const Point &A, const Point &B, const Point &C) {
    Line BAC = getAngleDividingLine(B, A, C);
    Line ABC = getAngleDividingLine(A, B, C);
    Point O = getLineIntersection(BAC, ABC);
    return Circle(O, getDistanceFromPointToLine(O, Line(B, C)));
}

// 多边形的有向面积
const double getPolygonArea(const Point *poly, const int &n) {
    double area = 0;
    for(int i = 1; i < n - 1; i++)
        area += (poly[i] - poly[0]) ^ (poly[i+1] - poly[0]);
    return area / 2.0;
}
// 判断点在多边形内
const int isPointInPolygon(const Point &p, const Point *poly, const int &n) {
    int wn = 0;
    for(int i = 0; i < n; i++) {
        if(isPointOnSegment(p, Segment(poly[i], poly[(i+1)%n]))) return 2; // on border
        int k = sign((poly[(i+1)%n] - poly[i]) ^ (p - poly[i]));
        int d1 = sign(poly[i].y - p.y);
        int d2 = sign(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
}

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

5.2 Convex Hull - Andrew

```
#include <cstdio>
#include <cmath>
#include <algorithm>

const double EPS = 1e-10;
int sign(double x) {
    if(fabs(x) < EPS) return 0;
    return x > 0 ? 1 : -1;
}
int dcmp(double x, double y) {
    return sign(x - y);
}
struct Point {
    double x, y;
    Point() {}
    Point(double _x, double _y) { x = _x; y = _y; }
    bool operator < (const Point &b) const {
        if(dcmp(x, b.x) == 0) return dcmp(y, b.y) < 0;
        return dcmp(x, b.x) < 0;
    }
    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);
    }
    double operator ^ (const Point &b) const {
        return y * b.x - x * b.y;
    }
};
// 凸包Andrew算法，输入点p[]，n个点，输出点ch，返回个数
// p[]中不能有重复点，执行完成后顺序被破坏
// 两个 < 改成 <= 可以让凸包边上含有点
int Andrew(Point *p, int n, Point *ch) {
    std::sort(p, p + n);
    // n = std::unique(p, p + n) - p;
    int m = 0;
    for(int i = 0; i < n; i++) {
        while(m > 1 && sign((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 && sign((ch[m-1]-ch[m-2]) ^ (p[i]-ch[m-2])) < 0) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
    return m;
}

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

5.3 Halfplane Intersection

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
#include <algorithm>
using std::vector;

const double EPS = 1e-10, INF = 1e20;
const double PI = acos(-1.0);
const int sign(const double &x) {
    if(fabs(x) < EPS) return 0;
    return x < 0 ? -1 : 1;
}
const int dcmp(const double &x, const double &y) {
    return sign(x - y);
}
struct Point {
    double x, y;
    Point() {}
    Point(const double &_x, const double &_y) { x = _x; y = _y; }
    void Read() { scanf("%lf%lf", &x, &y); }
    bool operator < (const Point &b) const {
        if(dcmp(x, b.x) == 0) return dcmp(y, b.y) < 0;
        return dcmp(x, b.x) < 0;
    }
    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 &k) const {
        return Point(x * k, y * k);
    }
    Point operator / (const double &k) const {
        return Point(x / k, y / k);
    }
    double operator * (const Point &b) const {
        return x * b.x + y * b.y;
    }
    double operator ^ (const Point &b) const {
        return x * b.y - y * b.x;
    }
    bool operator == (const Point &b) const {
        return dcmp(x, b.x) == 0 && dcmp(y, b.y) == 0;
    }
    double Abs() const {
        return sqrt(x * x + y * y);
    }
};
typedef Point Vector;
// 单位法向量
const Vector getNormalVector(const Vector &P) {
    double L = P.Abs(); // `L` CANNOT BE `0` !!!!
    return Vector(-P.y / L, P.x / L);
}
struct Line {
    Point P; // 直线上任意一点
    Vector v; // 直线向量，左边为对应半平面
    double angle;
    Line() {}
    Line(const Point &_P, const Vector &_v) {
        P = _P; v = _v;
        angle = atan2(v.y, v.x);
    }
    bool operator < (const Line &L) const {
        return angle < L.angle;
    }
};

// 判断点在直线左边（线上不算）
const bool isPointOnLineLeft(const Point &P, const Line &L) {
    return sign(L.v ^ (P - L.P)) > 0;
}
// 两直线交点（假设交点唯一存在）
const Point getLineIntersection(const Line &L1, const Line &L2) {
    Vector u = L1.P - L2.P;
    double t = (L2.v ^ u) / (L1.v ^ L2.v);
    return L1.P + L1.v * t;
}
// 半平面交（结果在poly，返回顶点数）
int HalfplaneIntersection(Line *L, int n, Point *poly) {
    std::sort(L, L + n);
    int first, last;
    Point *p = new Point[n];
    Line *q = new Line[n];
    q[first=last=0] = L[0];
    for(int i = 1; i < n; i++) {
        while(first < last && !isPointOnLineLeft(p[last-1], L[i])) last--;
        while(first < last && !isPointOnLineLeft(p[first], L[i])) first++;
        q[++last] = L[i];
        if(sign(q[last].v ^ q[last-1].v) == 0) {
            last--;
            if(isPointOnLineLeft(L[i].P, q[last])) q[last] = L[i];
        }
        if(first < last) p[last-1] = getLineIntersection(q[last-1], q[last]);
    }
    while(first < last && !isPointOnLineLeft(p[last-1], q[first])) last--;
    if(last - first <= 1) return 0; // 空集
    p[last] = getLineIntersection(q[last], q[first]); // 计算首位两个半平面交点
    // 保存答案
    for(int i = first; i <= last; i++) poly[i-first] = p[i];
    return last - first + 1;
}

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

6 String

6.1 HASH

P = 102929; mod1 = 1000000000 + 7; mod2 = 1000000000 + 9;

6.2 Minimum Representation - 最小表示法

```
namespace MinimumRepresentation{
    int get(int *s, int l)
    {
        int i = 0, j = 1, k = 0, t;
        while(i < l && j < l && k < l) {
            t = s[(i + k) >= l ? i + k - l : i + k] - s[(j + k) >= l ? j + k - l : j + k];
            if(!t) k++;
            else{
                if(t > 0) i = i + k + 1;
                else j = j + k + 1;
                if(i == j) ++ j;
                k = 0;
            }
        }
        return (i < j ? i : j);
    }
}
```

6.3 Manacher

```
#include <cstdio>
#include <algorithm>
// HDU 3068
const int N = 110000 + 10;

char t[N], s[2*N];
int n, p[2*N];

void pre(char *origin, char *str, int &_len)
{
    _len = 0;
    str[_len++] = '$';
    for(int i = 0; origin[i]; i++)
    {
        str[_len++] = '#';
        str[_len++] = origin[i];
    }
    str[_len++] = '#';
    str[_len] = 0;
    //puts(str);
}

void getPi(char *str, int _len, int *_P)
{
    int mx = 0, id;
    for(int i = 1; i < _len; i++)
    {
        if(mx > i) _P[i] = std::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;
        }
    }
}

int main()
{
    while(scanf("%s", t) == 1)
    {
        pre(t, s, n);
        getPi(s, n, p);
        int res = 1;
        for(int i = 1; i < n; i++)
            res = std::max(res, p[i]-1);
        printf("%d\n", res);
    }
    return 0;
}
```

6.4 KMP

```
#include <cstdio>
#include <cstring>
// POJ 3461 : Count the number of t occurrences in s
char s[1000000+10], t[1000000+10];
int next[1000000+10];

void getNext(char *t, int len, int *Next)
{
    memset(Next, 0, sizeof(Next)); Next[0] = -1;
    for(int j = 0, k = -1; j < len; )
    {
        if(k == -1 || t[j] == t[k]) Next[++j] = ++k;
        else k = Next[k];
    }
}
int kmp(char *s, int lens, char *t, int lent)
{
    int res = 0;
    getNext(t, lent, next);
    for(int i = 0, j = 0; i < lens; )
    {
        if(j == -1 || s[i] == t[j]) { i++; j++; }
        else j = next[j];
        if(j == lent) res++; // Bingo! [pos = j - lent]
    }
    return res;
}

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        scanf("%s%s", t, s);
        printf("%d\n", kmp(s, strlen(s), t, strlen(t)));
    }
    return 0;
}
```

6.5 Suffix Array

```
#include <cstdio>
#include <algorithm>
#include <map>
using std::map;
// POJ 3261 找重复了K次的最长子串
const int N = 20000 + 10;
/*
    sa[rank[i]] = i
    sa[i] = j       : rank i is s[j, n)
    rank[j] = i     : s[j, n) is rank i
    height[i] = j   : the longest common prefix of string rank _i and _i-1
*/

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 *s, int n, int m) // s[0...n-1] E [1, m)
{
    int i, j, p, *x = rank, *y = tmp;
    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(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;
        for(i = 0; i < m; i++) c[i] = 0;
        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];
        for(std::swap(x, y), p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i], sa[i-1], j) ? p - 1 : p++;
    }
    for(i = 0; i < n; i++) rank[sa[i]] = i;

    int k = 0; height[0] = 0;
    for(i = 0; i < n; height[rank[i++]] = k) if(rank[i])
        for(k ? k-- : 0, j = sa[rank[i]-1]; s[j+k] == s[i+k]; k++);
}

int n, K, a[N];
map<int, int> hash;

bool check(int len)
{
    int cnt = 0;
    for(int i = 1; i < n; i++)
    {
        if(height[i] >= len) cnt++;
        else cnt = 0;
        if(cnt >= K - 1) return 1;
    }
    return 0;
}
```

```

int Solve()
{
    int low = 0, high = n, ans = 0;
    while(low <= high)
    {
        int mid = low + (high - low) / 2;
        if(check(mid)) { low = mid + 1; ans = mid; }
        else high = mid - 1;
    }
    return ans;
}

int main()
{
    //-------Read--------
    scanf("%d%d", &n, &K);
    for(int i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
        tmp[i] = a[i];
    }
    std::sort(tmp, tmp+n);
    int cnt = 0;
    for(int i = 0; i < n; i++)
        if(i == 0 || tmp[i] != tmp[i-1]) hash[tmp[i]] = ++cnt;
    for(int i = 0; i < n; i++) a[i] = hash[a[i]];
    a[n++] = 0; ////////////
    DA(a, n, cnt+1);
/*  for(int i = 0; i < n; i++)
    {
        printf("rank = %d -> [%d, %d) [%d] :", i, sa[i], n, height[i]);
        for(int j = sa[i]; j < n; j++) printf(" %d", a[j]);
        puts("");
    }   */
    printf("%d\n", Solve());
    return 0;
}
```

6.6 Aho-Corasick Automaton

```
#include <cstdio>
#include <cstring>
#include <queue>
using std::queue;
// HDU 2222 查询 n 个模式串中有几个在原串 str 中出现了
struct ACG{
    int count;
    ACG *fail, *next[26];
    ACG()
    {
        fail = 0;
        count = 0;
        for(int i = 0; i < 26; i++) next[i] = 0;
    }
}*root;
queue<ACG*> Q;

void insert(char *str, ACG *p)
{
    int len = strlen(str);
    for(int i = 0; i < len; i++)
    {
        int x = str[i] - 'a';
        if(!p -> next[x]) p -> next[x] = new ACG;
        p = p -> next[x];
    }
    p -> count ++;
}

void build_acg()
{
    while(!Q.empty()) Q.pop();
    Q.push(root);
    while(!Q.empty())
    {
        ACG *p = Q.front(); Q.pop();
        for(int i = 0; i < 26; i++)
        {
            if(p -> next[i])
            {
                if(p == root) p -> next[i] -> fail = root;
                else{
                    ACG *temp = p -> fail;
                    while(temp)
                    {
                        if(temp -> next[i])
                        {
                            p -> next[i] -> fail = temp -> next[i];
                            break;
                        }
                        temp = temp -> fail;
                    }
                    if(!temp) p -> next[i] -> fail = root;
                }
                Q.push(p -> next[i]);
            }
        }
    }
}

int query(char *str, ACG *p)
{
    int len = strlen(str), res = 0;
    for(int i = 0; i < len; i++)
    {
        int x = str[i] - 'a';
        while(!p -> next[x] && p != root) p = p -> fail;
        p = p -> next[x];
        if(!p) p = root;
        ACG *temp = p;
        while(temp != root && temp -> count != -1)
        {
            res += temp -> count;
            temp -> count = -1;
            temp = temp -> fail;
        }
    }
    return res;
}

int n;
char tmp[1000000+10];

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        root = new ACG;
        scanf("%d", &n);
        for(int i = 1; i <= n; i++)
        {
            scanf("%s", tmp);
            insert(tmp, root);
        }
        build_acg();
        scanf("%s", tmp);
        printf("%d\n", query(tmp, root));
    }
    return 0;
}
```

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++ 读入优化

```
inline int nextInt()
{
    char ch = getchar(); int res = 0; bool sign = 0;
    while(!isdigit(ch) && ch != '-') ch = getchar();
    if(ch == '-') { sign = 1; ch = getchar(); }
    do res = (res << 1) + (res << 3) + ch - '0';
    while(isdigit(ch = getchar()));
    return sign ? -res : res;
}
```

7.3 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
```

7.4 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 ）
```

7.5 Java

7.5.1 The overall framework

```java
import java.io.*;
import java.util.*;
import java.math.*;
public class Main{
    public static void main(String args[])
    {
    }
}
```

7.5.2 Input and Output

```java
Scanner cin = new Scanner(System.in);
Scanner cin = new Scanner(new BufferedInputStream(System.in));
Scanner cin = new Scanner(new File("data.in"));

PrintWriter cout = new PrintWriter(System.out);
PrintWriter cout = new PrintWriter(new BufferedOutputStream(System.out));
PrintWriter cout = new PrintWriter(new File("data.out"));

int n = cin.nextInt();
String s = cin.next();
double m = cin.nextDouble();
String line = cin.nextLine(); // 读一整行
BigInteger c = cin.nextBigInteger();
while(cin.hasNext()) {};
```

```

//PrintWriter 用 cout.println(...);
System.out.println(n + "-->" + s "-->" + m);

//使用format控制格式，与C/C++一样，double用%f,
System.out.format("%03d", c).println();
System.out.format("%.3f", c).println();

//变量声明
int a, b[] = new int[100];
double a, b[] = new double[100];
int a[][] = new int[100][100];
String ...
BigInteger/BigDecimal ...
```

7.5.3 BigInteger

```
BigInteger a = BigInteger.valueOf(100);
BigInteger b = BigInteger.valueOf(50);
BigInteger ONE = BigInteger.ONE;
BigInteger TWO = BigInteger.valueOf(2);
a = a.add(ONE).subtract(b);
a = a.multiply(TWO).divide(TWO);
a = a.mod(TWO);
a.compareTo(ONE); // 大于1, 小于-1, 等于0
//BigDecimal 为高精小数
```

7.5.4 String

```
String s = "abcdefg"; // 注意0下标!
char c = s.charAt(2);// 相当于`char c = s[2]`(C++)(c = 'c')
char ch[];
ch = s.toCharArray(); // 字符串转换为字符数组
for(int i = 0; i < ch.length; i++) ch[i] += 2;
System.out.println(ch); // 输出cdefghi
String tmp1 = s.substring(1); // bcdefg
String tmp2 = s.substring(2, 4); // cd
```

7.5.5 Hexadecimal Conversion

```
import java.io.*;
import java.util.*;
import java.math.*;
// Binary, Octal, Decimal(Integer/BigInteger), Hexadecimal
public class Main{
    public static void main(String args[])
    {
        //Decimal(123) to Others
        String a1 = Integer.toBinaryString(123);
        String a2 = Integer.toOctalString(123);
        String a3 = Integer.toHexString(123);
        //Others to Decimal(123)
        int b1 = Integer.valueOf("1111011", 2);
        int b2 = Integer.valueOf("173", 8);
        int b3 = Integer.valueOf("7b", 16);
        // Others to BigInteger(Decimal(123))
        BigInteger c1 = new BigInteger("1111011", 2);
        BigInteger c2 = new BigInteger("173", 8);
        BigInteger c3 = new BigInteger("7B", 16);
    }
}
```

7.5.6 function

```
Arrays.fill(a, x); // for(int i = 0; i < N; i++) a[i] = x;
Arrays.fill(a, l, r, x); // for(int i = l; i < r; i++) a[i] = x;
Arrays.sort(a); // 给a的所有元素排序 升序
Arrays.sort(a, l, r); // 给a的[l, r)元素排序 升序
Arrays.sort(a, l, r, new cmp());

import java.io.*;
import java.util.*;
import java.math.*;
class INT{
    int s;
    public INT(int x) { s = x; }// 构造函数 INT a = new INT(3);
}
class cmp implements Comparator<INT>{
    public int compare(INT a, INT b)
    {
        return a.s - b.s;
    }
}
public class Main{
    public static void main(String args[])
    {
        Scanner cin = new Scanner(System.in);
        int n;
        INT a[] = new INT[100];
        for(int i = 1; i <= 10; i++) a[i] = new INT(11 - i);
        Arrays.sort(a, 1, 11, new cmp());
    }
}
//a[i].s排序前10 9 8 7 6 5 4 3 2 1
//a[i].s排序后1 2 3 4 5 6 7 8 9 10

String s = Integer.toString(n, B); // 把十进制数n转换成B进制数
int b = Integer.parseInt(s, B); // 把B进制数s转换成10进制数
```

7.6 Batch test

7.6.1 @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
```

7.6.2 @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
```

7.7 Vimrc Config For Linux

```
filetype on
filetype indent on
set nobackup
set nu
set st=4
set ts=4
set sw=4

map <F7> <Esc>:w<CR>:!javac %:r.java<CR>:!java %:r<CR>
imap <F7> <Esc>:w<CR>:!javac %:r.java<CR>:!java %:r<CR>
map <F8> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!gdb %:r<CR>
imap <F8> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!gdb %:r<CR>
map <F9> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!./%:r<CR>
```

```
imap <F9> <Esc>:w<CR>:!g++ -g %:r.cpp -o %:r<CR>:!./%:r<CR>
map <c-a> <Esc>gg"+yG
imap␣<c-a>␣<Esc>gg"+yG
```

8 WHAT THE FUCK!!!

8.1 不用递归的 DFS - dfsWithoutDfs.cpp

```
void dfs()
{
    st.clear();
    DFS.push(std::make_pair(root, 0));
    int END = 3;
    while(!DFS.empty())
    {
        pair<int, int> &now = DFS.top();
        if(now.second != END)
        {
            ++now.second;
            DFS.push(std::make_pair(SON[now.first], 0));
        }
        else DFS.pop();
    }
}
```

8.2 时间结构体

```
const char mon[][5] = {"", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
const int days[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
struct Date{
    int year, month, day, hour;
    bool check(int y)
    {
        return (y % 4 == 0 && y % 100 != 0) || y % 400 == 0;
    }
    int getHours() // to 2000.01.01 0 o'clock
    {
        int hours = 1;
        for(int i = 2000; i < year; i++) hours += 24 * (365 + check(i));
        for(int i = 1; i < month; i++)
            hours += 24 * (days[i] + (i == 2 && check(year)));
        return hours += 24 * (day - 1) + hour;
    }
};
```