GDCPC2018 Template

By Jsphlim

2018.5.4

DataStrure

1.Segment Tree

```
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
const int maxn = 55555;
int sum[maxn<<2];
void PushUP(int rt) {
	sum[rt] = sum[rt<<1] + sum[rt<<1|1];   //求和操作 可更改(极值)
}
void build(int l,int r,int rt) {
	if (l == r) {
		scanf("%d",&sum[rt]);
		return ;
	}
	int m = (l + r) >> 1;
	build(lson);
	build(rson);
	PushUP(rt);
}
void update(int p,int add,int l,int r,int rt) {
	if (l == r) {
		sum[rt] += add;
		return ;
	}
	int m = (l + r) >> 1;
	if (p <= m) update(p , add , lson);
	else update(p , add , rson);
	PushUP(rt);
}
int query(int L,int R,int l,int r,int rt) {
	if (L <= l && r <= R) {
		return sum[rt];
	}
	int m = (l + r) >> 1;
	int ret = 0;
	if (L <= m) ret += query(L , R , lson);
	if (R > m) ret += query(L , R , rson);
	return ret;
}
int main() {
	int T , n;
	scanf("%d",&T);
	for (int cas = 1 ; cas <= T ; cas ++) {
		printf("Case %d:\n",cas);
		scanf("%d",&n);
		build(1 , n , 1);
		char op[10];
		while (scanf("%s",op)) {
			if (op[0] == 'E') break;
			int a , b;
			scanf("%d%d",&a,&b);
			if (op[0] == 'Q') printf("%d\n",query(a , b , 1 , n , 1));
			else if (op[0] == 'S') update(a , -b , 1 , n , 1);
			else update(a , b , 1 , n , 1);
		}
	}
	return 0;
}
```

线段树扫描线

```
const int maxn=1e5+50;
struct Type
{
    double l,r,h;  //以横坐标建立线段树，纵坐标为高度
    int cur;   //cur=-1说明是出边，cur=1说明是入边

    void sets( double x1, double x2, double h, int cur )
    {
        this->l=x1; this->r=x2; this->h=h; this->cur=cur;
    }
    bool operator < ( const Type& a )const
    {
        return h<a.h;
    }
};
Type seg[maxn];
double sumv[maxn*4]; //用来维护线段并
int cntv[maxn*4]; //用来维护当前节点代表的区间被覆盖了几次
double point[maxn*2];//用来维护离散化后的端点
int n;
int uniq(int k)
```

```
{
    int m=1;
    sort( point+1,point+1+k );
    for( int i=1;i<k;i++ )
    {
        if( point[i]!=point[i+1] )point[++m]=point[i+1];
    }
    return m;
}

void build( int O,int L,int R )
{
    if(L==R){ sumv[O]=0; cntv[O]=0; }
    else
    {
        int mid=(L+R)/2;
        build( O*2,L,mid );
        build( O*2+1,mid+1,R );
        sumv[O]=0; cntv[O]=0;
    }
}


void maintain( int O, int L, int R )
{
    if( cntv[O] )
    {
        sumv[O]=point[R+1]-point[L];
    }
    else if( L<R )
    {
        sumv[O]=sumv[O*2]+sumv[O*2+1];
       // cntv[O]=min( cntv[O*2],cntv[O*2+1] );
    }
    else { sumv[O]=0; cntv[O]=0; }
}

void pushdown( int O )
{
    if( cntv[O] )
    {
        cntv[O*2]=cntv[O*2+1]=cntv[O];
        cntv[O]=0;
        sumv[O]=0;
    }
}

void update( int O, int L, int R, int qL, int qR,int op )
{
    if( qL<=L && R<=qR )
    {
        cntv[O]+=op;
    }
    else
    {
       // pushdown(O);    //pushdown其实是不需要的，因为我们遇到一个cnt就可以返回整段信息，也就是说，更深的cnt信息不需要考虑，所以cnt没必要下传
        int mid=(L+R)/2;
        if( qL<=mid )update( O*2,L,mid,qL,qR,op );
        if( qR>mid )update( O*2+1,mid+1,R,qL,qR,op );
    }
    maintain( O,L,R );//重新计算sumv[O];
}

int main()
{
    int kase=0;
    while( ~scanf("%d",&n) &&n )
    {
        int k=0;
        for( int i=1; i<=n; i++ )
        {
            double x1,x2,y1,y2;
            scanf( "%lf%lf%lf%lf",&x1,&y1,&x2,&y2 );
            seg[++k].sets( x1,x2,y1,-1 );
            point[k]=x1;
            seg[++k].sets( x1,x2,y2,1 );
            point[k]=x2;
        }
        int m=uniq(k);  //对端点进行了离散化
        sort( seg+1,seg+1+k ); //所有k条线段从低到高排序
        build( 1,1,m );
        double ans=0;
        for( int i=1; i<k; i++ )
        {
            int L=lower_bound( point+1,point+1+m,seg[i].l )-point;
            int R=lower_bound( point+1,point+1+m,seg[i].r )-point-1;
            update( 1,1,m,L,R,seg[i].cur );
            ans+=sumv[1]*( seg[i+1].h-seg[i].h );
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n",++kase,ans);
    }

    return 0;
}
```

数对交换问题

交换相邻两数 如果只是交换相邻两数，那么最少交换次数为该序列的逆序数。

交换任意两数

```
int getMinSwaps(vector<int> &A)
{
    //  排序
    vector<int> B(A);
    sort(B.begin(), B.end());
    map<int, int> m;
    int len = (int)A.size();
    for (int i = 0; i < len; i++)
    {
        m[B[i]] = i;    //  建立每个元素与其应放位置的映射关系
    }

    int loops = 0;      //  循环节个数
    vector<bool> flag(len, false);
    //  找出循环节的个数
    for (int i = 0; i < len; i++)
    {
        if (!flag[i])
        {
            int j = i;
            while (!flag[j])
            {
                flag[j] = true;
                j = m[A[j]];    //  原序列中j位置的元素在有序序列中的位置
            }
            loops++;
        }
    }
    return len - loops;
}

vector<int> nums;

int main()
{
    nums.push_back(1);
    nums.push_back(2);
    nums.push_back(4);
    nums.push_back(3);
    nums.push_back(5);

    int res = getMinSwaps(nums);

    cout << res << '\n';

    return 0;
}
```

交换任意区间

```
/*
 *  默认目标映射关系是 key 1 => val 1 …… key n => val n
 *  如果序列不是 1~n 可以通过 map 建立新的目标映射关系
 *  交换任意区间的本质是改变了元素的后继，故建立元素与其初始状态后继的映射关系
 */
const int MAXN = 30;

int n;
int vis[MAXN];
int A[MAXN], B[MAXN];

int getMinSwaps()
{
    memset(vis, 0, sizeof(vis));

    for (int i = 1; i <= n; i++)
    {
        B[A[i]] = A[i % n + 1];
    }
    for (int i = 1; i <= n; i++)
    {
        B[i] = (B[i] - 2 + n) % n + 1;
    }

    int cnt = n;
    for (int i = 1; i <= n; i++)
    {
        if (vis[i])
        {
            continue;
        }
        vis[i] = 1;
        cnt--;
        for (int j = B[i]; j != i; j = B[j])
        {
            vis[j] = 1;
        }
    }

    return cnt;
```

```
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> A[i];
    }

    int res = getMinSwaps();

    cout << res << '\n';

    return 0;
}
```

带权并查集

```
const int N = 1010;
struct lset
{
    int p[N], rank[N], sz;
    void link(int x, int y)
    {
        if (x == y)
        {
            return ;
        }
        if (rank[x] > rank[y])
        {
            p[y] = x;
        }
        else
        {
            p[x] = y;
        }
        if (rank[x] == rank[y])
        {
            rank[y]++;
        }
        return ;
    }
    void makeset(int n)
    {
        sz = n;
        for (int i = 0; i < sz; i++)
        {
            p[i] = i;
            rank[i] = 0;
        }
        return ;
    }
    int findset(int x)
    {
        if (x != p[x])
        {
            p[x] = findset(p[x]);
        }
        return p[x];
    }
    void unin(int x, int y)
    {
        link(findset(x), findset(y));
        return ;
    }
    void compress()
    {
        for (int i = 0; i < sz; i++)
        {
            findset(i);
        }
        return ;
    }
};
```

主席树系列

查询区间有多少个不同的数

```
/*
 *  给出一个序列,查询区间内有多少个不相同的数
 */
const int MAXN = 30010;
const int M = MAXN * 100;
int n, q, tot;
int a[MAXN];
int T[MAXN], lson[M], rson[M], c[M];

int build(int l, int r)
{
    int root = tot++;
    c[root] = 0;
    if (l != r)
```

```
    {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}

int update(int root, int pos, int val)
{
    int newroot = tot++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = n;
    while (l < r)
    {
        int mid = (l + r) >> 1;
        if (pos <= mid)
        {
            lson[newroot] = tot++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        }
        else
        {
            rson[newroot] = tot++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}

int query(int root, int pos)
{
    int ret = 0;
    int l = 1, r = n;
    while (pos < r)
    {
        int mid = (l + r) >> 1;
        if (pos <= mid)
        {
            r = mid;
            root = lson[root];
        }
        else
        {
            ret += c[lson[root]];
            root = rson[root];
            l = mid + 1;
        }
    }
    return ret + c[root];
}

int main()
{
    //  freopen("in.txt", "r", stdin);
    //  freopen("out.txt", "w", stdout);
    while (scanf("%d", &n) == 1)
    {
        tot = 0;
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
        }
        T[n + 1] = build(1, n);
        map<int,int> mp;
        for (int i = n; i >= 1; i--)
        {
            if (mp.find(a[i]) == mp.end())
            {
                T[i] = update(T[i + 1], i, 1);
            }
            else
            {
                int tmp = update(T[i + 1], mp[a[i]], -1);
                T[i] = update(tmp, i, 1);
            }
            mp[a[i]] = i;
        }
        scanf("%d", &q);
        while (q--)
        {
            int l, r;
            scanf("%d%d", &l, &r);
            printf("%d\n", query(T[l], r));
        }
    }
    return 0;
}
```

静态区间第k小

```
const int MAXN = 100010;
const int M = MAXN * 30;
int n, q, m, tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];

void Init_hash()
{
    for (int i = 1; i <= n; i++)
    {
        t[i] = a[i];
    }
    sort(t + 1, t + 1 + n);
    m = (int)(unique(t + 1, t + 1 + n) - t - 1);
}

int build(int l, int r)
{
    int root = tot++; c[root] = 0;
    if (l != r)
    {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}

int hash_(int x)
{
    return (int)(lower_bound(t + 1, t + 1 + m, x) - t);
}

int update(int root, int pos, int val)
{
    int newroot = tot++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = m;
    while (l < r)
    {
        int mid = (l + r) >> 1;
        if (pos <= mid)
        {
            lson[newroot] = tot++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        }
        else
        {
            rson[newroot] = tot++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}

int query(int left_root, int right_root, int k)
{
    int l = 1, r = m;
    while ( l < r)
    {
        int mid = (l + r) >> 1;
        if (c[lson[left_root]] - c[lson[right_root]] >= k )
        {
            r = mid;
            left_root = lson[left_root];
            right_root = lson[right_root];
        }
        else
        {
            l = mid + 1;
            k -= c[lson[left_root]] - c[lson[right_root]];
            left_root = rson[left_root];
            right_root = rson[right_root];
        }
    }
    return l;
}

int main()
{
    //  freopen("in.txt","r",stdin);
    //  freopen("out.txt","w",stdout);
    while (scanf("%d%d", &n, &q) == 2)
    {
        tot = 0;
        for (int i = 1; i <= n; i++)
        {
```

```
            scanf("%d", &a[i]);
        }
        Init_hash();
        T[n + 1] = build(1, m);
        for (int i = n; i; i--)
        {
            int pos = hash_(a[i]);
            T[i] = update(T[i + 1], pos, 1);
        }
        while (q--)
        {
            int l, r, k;
            scanf("%d%d%d", &l, &r, &k);
            printf("%d\n", t[query(T[l], T[r + 1], k)]);
        }
    }
    return 0;
}
```

树上路径点权第k大

```
/*
 *  LCA + 主席树
 */
//  主席树部分
const int MAXN = 200010;
const int M = MAXN * 40;
int n, q, m, TOT;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
void Init_hash()
{
    for (int i = 1; i <= n; i++)
    {
        t[i] = a[i];
    }
    sort(t + 1, t + 1 + n);
    m = (int)(unique(t + 1, t + n + 1) - t - 1);
    return ;
}

int build(int l, int r)
{
    int root = TOT++;
    c[root] = 0;
    if (l != r)
    {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}

int hash_(int x)
{
    return (int)(lower_bound(t + 1, t + 1 + m, x) - t);
}

int update(int root, int pos, int val)
{
    int newroot = TOT++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = m;
    while (l < r)
    {
        int mid = (l + r) >> 1;
        if (pos <= mid)
        {
            lson[newroot] = TOT++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        }
        else
        {
            rson[newroot] = TOT++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}

int query(int left_root, int right_root, int LCA, int k)
{
    int lca_root = T[LCA];
    int pos = hash_(a[LCA]);
    int l = 1, r = m;
    while (l < r)
    {
        int mid = (l + r) >> 1;
```

```
        int tmp = c[lson[left_root]] + c[lson[right_root]] - 2 * c[lson[lca_root]] + (pos >= l && pos <= mid);
        if (tmp >= k)
        {
            left_root = lson[left_root];
            right_root = lson[right_root];
            lca_root = lson[lca_root];
            r = mid;
        }
        else
        {
            k -= tmp;
            left_root = rson[left_root];
            right_root = rson[right_root];
            lca_root = rson[lca_root];
            l = mid + 1;
        }
    }
    return l;
}

//  LCA部分
int rmq[2 * MAXN];  //  rmq数组,就是欧拉序列对应的深度序列

struct ST
{
    int mm[2 * MAXN];
    int dp[2 * MAXN][20];   //  最小值对应的下标
    void init(int n)
    {
        mm[0] = -1;
        for (int i = 1; i <= n; i++)
        {
            mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
            dp[i][0] = i;
        }
        for (int j = 1; j <= mm[n]; j++)
        {
            for (int i = 1; i + (1 << j) - 1 <= n; i++)
            {
                dp[i][j] = rmq[dp[i][j - 1]] < rmq[dp[i + (1 << (j - 1))][j - 1]] ? dp[i][j - 1] : dp[i + (1 << (j - 1))][j - 1];
            }
        }
        return ;
    }
    int query(int a, int b) //  查询[a,b]之间最小值的下标
    {
        if (a > b)
        {
            swap(a, b);
        }
        int k = mm[b - a + 1];
        return rmq[dp[a][k]] <= rmq[dp[b - (1 << k) + 1][k]] ? dp[a][k] : dp[b - (1 << k) + 1][k];
    }
};

//  边的结构体定义
struct Edge
{
    int to, next;
};

Edge edge[MAXN * 2];
int tot, head[MAXN];
int F[MAXN * 2];    //  欧拉序列,就是dfs遍历的顺序,长度为2*n-1,下标从1开始
int P[MAXN];        //  P[i]表示点i在F中第一次出现的位置
int cnt;
ST st;

void init()
{
    tot = 0;
    memset(head, -1, sizeof(head));
    return ;
}

void addedge(int u, int v)  //  加边,无向边需要加两次
{
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
    return ;
}

void dfs(int u, int pre, int dep)
{
    F[++cnt] = u;
    rmq[cnt] = dep;
    P[u] = cnt;
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].to;
        if (v == pre)
        {
            continue;
        }
        dfs(v, u, dep + 1);
        F[++cnt] = u;
```

```
        rmq[cnt] = dep;
    }
    return ;
}

void LCA_init(int root, int node_num)   //  查询LCA前的初始化
{
    cnt = 0;
    dfs(root, root, 0);
    st.init(2 * node_num - 1);
    return ;
}

int query_lca(int u, int v)             //  查询u,v的lca编号
{
    return F[st.query(P[u], P[v])];
}

void dfs_build(int u, int pre)
{
    int pos = hash_(a[u]);
    T[u] = update(T[pre], pos, 1);
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].to;
        if (v == pre)
        {
            continue;
        }
        dfs_build(v, u);
    }
    return ;
}


int main()
{
    //  freopen("in.txt", "r", stdin);
    //  freopen("out.txt", "w", stdout);
    while (scanf("%d%d", &n, &q) == 2)
    {
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
        }
        Init_hash();
        init();
        TOT = 0;
        int u, v;
        for (int i = 1; i < n; i++)
        {
            scanf("%d%d", &u, &v);
            addedge(u, v);
            addedge(v, u);
        }
        LCA_init(1, n);
        T[n + 1] = build(1, m);
        dfs_build(1, n + 1);
        int k;
        while (q--)
        {
            scanf("%d%d%d", &u, &v, &k);
            printf("%d\n", t[query(T[u], T[v], query_lca(u, v), k)]);
        }
    }
    return 0;
}
```

动态第区间第k大

```
/*
 *  树状数组套主席树
 */
const int MAXN = 60010;
const int M = 2500010;
int n, q, m, tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M],c[M];
int S[MAXN];

struct Query
{
    int kind;
    int l, r, k;
} query[10010];

void Init_hash(int k)
{
    sort(t, t + k);
    m = (int)(unique(t, t + k) - t);
    return ;
}

int hash_(int x)
{
    return (int)(lower_bound(t, t + m, x) - t);
```

```
}

int build(int l, int r)
{
    int root = tot++;
    c[root] = 0;
    if (l != r)
    {
        int mid = (l + r) / 2;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}

int Insert(int root, int pos, int val)
{
    int newroot = tot++, tmp = newroot;
    int l = 0, r = m - 1;
    c[newroot] = c[root] + val;
    while (l < r)
    {
        int mid = (l + r) >> 1;
        if (pos <= mid)
        {
            lson[newroot] = tot++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        }
        else
        {
            rson[newroot] = tot++;
            lson[newroot] = lson[root];
            newroot = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}

int lowbit(int x)
{
    return x & (-x);
}

int use[MAXN];

void add(int x, int pos, int val)
{
    while (x <= n)
    {
        S[x] = Insert(S[x], pos, val);
        x += lowbit(x);
    }
    return ;
}
int sum(int x)
{
    int ret = 0;
    while (x > 0)
    {
        ret += c[lson[use[x]]];
        x -= lowbit(x);
    }
    return ret;
}

int Query(int left, int right, int k)
{
    int left_root = T[left - 1];
    int right_root = T[right];
    int l = 0, r = m - 1;
    for (int i = left - 1; i; i -= lowbit(i))
    {
        use[i] = S[i];
    }
    for (int i = right; i; i -= lowbit(i))
    {
        use[i] = S[i];
    }
    while (l < r)
    {
        int mid = (l + r) / 2;
        int tmp = sum(right) - sum(left - 1) + c[lson[right_root]] - c[lson[left_root]];
        if (tmp >= k)
        {
            r = mid;
            for (int i = left - 1; i; i -= lowbit(i))
            {
                use[i] = lson[use[i]];
            }
            for (int i = right; i; i -= lowbit(i))
            {
```

```
                use[i] = lson[use[i]];
            }
            left_root = lson[left_root];
            right_root = lson[right_root];
        }
        else
        {
            l = mid + 1;
            k -= tmp;
            for (int i = left - 1; i; i -= lowbit(i))
            {
                use[i] = rson[use[i]];
            }
            for (int i = right; i; i -= lowbit(i))
            {
                use[i] = rson[use[i]];
            }
            left_root = rson[left_root];
            right_root = rson[right_root];
        }
    }
    return l;
}

void Modify(int x, int p, int d)
{
    while (x <= n)
    {
        S[x] = Insert(S[x], p, d);
        x += lowbit(x);
    }
    return ;
}

int main()
{
    //  freopen("in.txt", "r", stdin);
    //  freopen("out.txt", "w", stdout);
    int Tcase;
    scanf("%d", &Tcase);
    while (Tcase--)
    {
        scanf("%d%d", &n, &q);
        tot = 0;
        m = 0;
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
            t[m++] = a[i];
        }
        char op[10];
        for (int i = 0; i < q; i++)
        {
            scanf("%s", op);
            if (op[0] == 'Q')
            {
                query[i].kind = 0;
                scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
            }
            else
            {
                query[i].kind = 1;
                scanf("%d%d", &query[i].l, &query[i].r);
                t[m++] = query[i].r;
            }
        }
        Init_hash(m);
        T[0] = build(0, m - 1);
        for (int i = 1; i <= n; i++)
        {
            T[i] = Insert(T[i - 1], hash_(a[i]), 1);
        }
        for (int i = 1; i <= n; i++)
        {
            S[i] = T[0];
        }
        for (int i = 0; i < q; i++)
        {
            if (query[i].kind == 0)
            {
                printf("%d\n", t[Query(query[i].l, query[i].r, query[i].k)]);
            }
            else
            {
                Modify(query[i].l, hash_(a[query[i].l]), -1);
                Modify(query[i].l, hash_(query[i].r), 1);
                a[query[i].l] = query[i].r;
            }
        }
    }
    return 0;
}
```

动态规划

最长有序子序列O(nlogn)

```
/*
 *  递增（默认）
 *  递减
 *  非递增
 *  非递减 (1)>= && <  (2)<  (3)>=
 */
const int MAXN = 1001;

int a[MAXN], f[MAXN], d[MAXN];   //  d[i] 用于记录 a[0...i] 以 a[i] 结尾的最大长度

int bsearch(const int *f, int size, const int &a)
{
    int l = 0, r = size - 1;
    while (l <= r)
    {
        int mid = (l + r) / 2;
        if (a > f[mid - 1] && a <= f[mid])  //  (1)
        {
            return mid;
        }
        else if (a < f[mid])
        {
            r = mid - 1;
        }
        else
        {
            l = mid + 1;
        }
    }
    return -1;
}

int LIS(const int *a, const int &n)
{
    int i, j, size = 1;
    f[0] = a[0];
    d[0] = 1;
    for (i = 1; i < n; ++i)
    {
        if (a[i] <= f[0])               //  (2)
        {
            j = 0;
        }
        else if (a[i] > f[size - 1])    //  (3)
        {
            j = size++;
        }
        else
        {
            j = bsearch(f, size, a[i]);
        }
        f[j] = a[i];
        d[i] = j + 1;
    }
    return size;
}

int main()
{
    int i, n;
    while (scanf("%d", &n) != EOF)
    {
        for (i = 0; i < n; ++i)
        {
            scanf("%d", &a[i]);
        }
        printf("%d\n", LIS(a, n));      // 求最大递增 / 上升子序列(如果为最大非降子序列,只需把上面的注释部分给与替换)
    }
    return 0;
}
```

最优三角剖分

```
#define N 6        //顶点数/边数
int weight[][N] = {{0,2,2,3,1,4},{2,0,1,5,2,3},{2,1,0,2,1,4},{3,5,2,0,6,2},
{1,2,1,6,0,1},{4,3,4,2,1,0}};
int t[N][N];    //t[i][j]表示多边形{Vi-1VkVj}的最优权值
int s[N][N];    //s[i][j]记录Vi-1到Vj最优三角剖分的中间点K
int get_weight(const int a, const int b, const int c)
{
    return weight[a][b] + weight[b][c] + weight[c][a];
}
void minest_weight_val()
{
    int i,r,k,j;
    int min;

    for (i = 1; i < N; i++)
    {
        t[i][i] = 0;
    }

    for (r = 2; r < N; r++)
```

```
    {
        for (i = 1; i < N-r+1; i++)
        {
            j = i + r -1;
            min = 9999;        //假设最小值
            for (k = i; k < j; k++)
            {
                t[i][j] = t[i][k] + t[k+1][j] + get_weight(i-1,k,j);
                if (t[i][j] < min)        //判断是否是最小值
                {
                    min = t[i][j];
                    s[i][j] = k;
                }
            }
            t[i][j] = min;        //取得多边形{Vi-1，Vj}的划分三角最小权值
        }
    }
}
void back_track(int a, int b)
{
    if (a == b) return;
    back_track(a,s[a][b]);
    back_track(s[a][b]+1,b);    //记得这是要加一
    printf("最优三角: V%d V%d V%d.\n",a-1,s[a][b],b);
}
int main()
{
    minest_weight_val();
    printf("result:%d\n",t[1][N-1]);
    back_track(1,5);
    return 0;
}
```

0-1背包问题

```
int main()
{
    cin>>m>>n;
    for(i=1;i<=n;i++)
        cin>>w[i]>>c[i];
    for(i=1;i<=n;i++)
        for(j=m;j>0;j--)
        {
            if(w[i]<=j)
            f[i][j]=maxn(f[i-1][j],f[i-1][j-w[i]]+c[i]);//状态转移方程式
            else f[i][j]=f[i-1][j];
        }
        cout<<f[n][m];
        return 0;
}
```

完全背包问题

设有n种物品，每种物品有一个价值，但每种物品的数量是无限的

```
int main()
{
    scanf("%d%d",&m,&n);
    for(int i=1;i<=n;i++)
     scanf("%d%d",&w[i],&c[i]);
    for(int i=1;i<=n;i++){
      for(int v=1;v<=m;v++){
      if(v<w[i]) f[i][v]=f[i-1][v];
      else {
        if(f[i-1][v]>f[i][v-w[i]]+c[i]) f[i][v]=f[i-1][v];
        else f[i][v]=f[i][v-w[i]]+c[i];
      }
    }
  }
    printf("%d",f[n][m]); return 0;
}
```

多重背包

设有n种物品，每种物品有一个价值，但每种物品的数量是有限的
二进制优化，减少运算次数

```
int main()
{
    scanf("%d%d",&n,&m); //物体种数 背包容量
    for(int i=1;i<=n;i++)
    {
        int x,y,s,t=1;
        scanf("%d%d%d",&x,&y,&s); //重量 价值 数量
        while(s>=t)
        {
            v[++n1]=x*t;
            w[n1]=y*t;
            s-=t;
            t*=2;
        }
        v[++n1]=x*s;
        w[n1]=y*s;
```

```
    }
    for(int i=1;i<=n1;i++)
     for(int j=m;j>=v[i];j--)
      f[j]=max(f[j],f[j-v[i]]+w[i]);
    printf("%d\n",f[m]); return 0;

}
```

二维费用背包

```
int main()
{
    memset(f,127,sizeof(f));
    f[0][0]=0;
    scanf("%d%d%d",&v,&u,&k); //所需1总量 所需2总量 背包容量
    for(int i=1;i<=k;i++)
     scanf("%d%d%d",&a[i],&b[i],&c[i]);//价值1 价值2 重费
    for(int i=1;i<=k;i++)
     for(int j=v;j>=0;j--)
      for(int l=u;l>=0;l--)
      {
          int t1=j+a[i],t2=l+b[i];;
          if(t1>v) t1=v;
          if(t2>u) t2=u;
          if(f[t1][t2]>f[j][l]+c[i]) f[t1][t2]=f[j][l]+c[i];
      }
      printf("%d",f[v][u]);
      return 0;
}
```

混合三种背包

```
```

计算几何

海伦公式

```
s = sqrt(p * (p - a) * (p - b) * (p - c));
p = (a + b + c) / 2;
```

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);
}
```

关键点与A, B, C三顶点距离之和 费马点 该点到三角形三个顶点的距离之和最小。 有个有趣的结论: 若三角形的三个内角均小于120度,那么该点连接三个顶点形成的三个角均为120度;若三角形存在一个内角大于120度,则该顶点就是费马点。 计算公式如下: 若有一个内角大于120度(这里假设为角C),则距离为a + b;若三个内角均小于120度,则距离为sqrt((a * a + b * b + c * c + 4 * sqrt(3.0) * s) / 2)。

内心 角平分线的交点。 令x = (a + b - c) / 2, y = (a - b + c) / 2, z = (-a + b + c) / 2, h = s / p. 计算公式为sqrt(x * x + h * h) + sqrt(y * y + h * h) + sqrt(z * z + h * h)。

重心 中线的交点。 计算公式如下: 2.0 / 3 * (sqrt((2 * (a * a + b * b) - c * c) / 4)

- sqrt((2 * (a * a + c * c) - b * b) / 4) + sqrt((2 * (b * b + c * c) - a * a) / 4))。

垂心 垂线的交点。 计算公式如下: 3 * (c / 2 / sqrt(1 - cosC * cosC))。

Graham扫描线

```
struct node
{
    int x,y;
} a[105],p[105];
int top,n;
double cross(node p0,node p1,node p2)//计算叉乘，注意p0,p1,p2的位置，这个决定了方向
{
    return (p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x);
}
double dis(node a,node b)//计算距离，这个用在了当两个点在一条直线上
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
bool cmp(node p1,node p2)//极角排序
{
    double z=cross(a[0],p1,p2);
    if(z>0||(z==0&&dis(a[0],p1)<dis(a[0],p2)))
        return 1;
    return 0;
}
void Graham()
{
    int k=0;
    for(int i=0; i<n; i++)
        if(a[i].y<a[k].y||(a[i].y==a[k].y&&a[i].x<a[k].x))
```

```
            k=i;
        swap(a[0],a[k]);
        //找p[0]
        sort(a+1,a+n,cmp);
        top=1;
        p[0]=a[0];
        p[1]=a[1];
        for(int i=2; i<n; i++)//控制进栈出栈
        {
            while(cross(p[top-1],p[top],a[i])<0&&top)
                top--;
            top++;
            p[top]=a[i];
        }
}
int main()
{
    int m;
    scanf("%d",&m);
    while(m--)
    {
        scanf("%d",&n);
            for(int i=0; i<n; i++)
            {
                scanf("%d%d",&a[i].x,&a[i].y);
            }
            Graham();
            for(int i=0; i<=top; i++)
            {
                printf("%d %d\n",p[i].x,p[i].y);
            }
    }
    return 0;
}
```

四点共面判断

```
struct  point
{
    double x, y, z;
    point  operator - (point &o)
    {
        point  ans;
        ans.x = this->x - o.x;
        ans.y = this->y - o.y;
        ans.z = this->z - o.z;
        return ans;
    }
};

double  dot_product(const point &a, const point &b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

point cross_product(const point &a, const point &b)
{
    point  ans;
    ans.x = a.y * b.z - a.z * b.y;
    ans.y = a.z * b.x - a.x * b.z;
    ans.z = a.x * b.y - a.y * b.x;
    return ans;
}

int main()
{
    point p[4];
    int T;
    for (scanf("%d", &T); T--;)
    {
        for (int i = 0; i < 4; ++i)
        {
            scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
        }
        puts(dot_product(p[3] - p[0], cross_product(p[2] - p[0], p[1] - p[0])) == 0.0 ? "Yes\n" : "No\n");
    }
    return 0;
}
```

多边形重心

```
/*
 *  求多边形重心
 *  INIT: pnt[]已按顺时针(或逆时针)排好序; | CALL: res = bcenter(pnt, n);
 */
struct point
{
    double x, y;
};

point bcenter(point pnt[], int n)
{
    point p, s;
    double tp, area = 0, tpx = 0, tpy = 0;
```

```
    p.x = pnt[0].x;
    p.y = pnt[0].y;
    for (int i = 1; i <= n; ++i)
    {   //  point:0 ~ n - 1
        s.x = pnt[(i == n) ? 0 : i].x;
        s.y = pnt[(i == n) ? 0 : i].y;
        tp = (p.x * s.y - s.x * p.y);
        area += tp / 2;
        tpx += (p.x + s.x) * tp;
        tpy += (p.y + s.y) * tp;
        p.x = s.x;
        p.y = s.y;
    }
    s.x = tpx / (6 * area);
    s.y = tpy / (6 * area);
    return s;
}
```

最接近点对问题

```
struct Point{
    double x, y;
}point[maxn];
int n, mpt[maxn];
//以x为基准排序
bool cmpxy(const Point& a, const Point& b){
    if (a.x != b.x)
        return a.x < b.x;
    return a.y < b.y;
}

bool cmpy(const int& a, const int& b){
    return point[a].y < point[b].y;
}

double min(double a, double b){
    return a < b ? a : b;
}

double dis(int i, int j){
    return sqrt((point[i].x - point[j].x)*(point[i].x - point[j].x) + (point[i].y - point[j].y)*(point[i].y - point[j].y));
}

double Closest_Pair(int left, int right){
    double d = inf;
    if (left == right)
        return d;
    if (left + 1 == right)
        return dis(left, right);
    int mid = (left + right) >> 1;
    double d1 = Closest_Pair(left, mid);
    double d2 = Closest_Pair(mid + 1, right);
    d = min(d1, d2);
    int i, j, k = 0;
    //分离出宽度为d的区间
    for (i = left; i <= right; i++){
        if (fabs(point[mid].x - point[i].x) <= d)
            mpt[k++] = i;
    }
    sort(mpt, mpt + k, cmpy);
    //线性扫描
    for (i = 0; i < k; i++){
        for (j = i + 1; j < k && point[mpt[j]].y - point[mpt[i]].y<d; j++){
            double d3 = dis(mpt[i], mpt[j]);
            if (d > d3)    d = d3;
        }
    }
    return d;
}

int main(){
    while (~scanf("%d", &n) && n){
        for (int i = 0; i < n; i++)
            scanf("%lf %lf", &point[i].x, &point[i].y);
        sort(point, point + n, cmpxy);
        printf("%.2lf\n", Closest_Pair(0, n - 1) / 2);
    }
    return 0;
}
```

线段相交判断

```
const double eps = 1e-10;

struct point
{
    double x, y;
};

double min(double a, double b)
{
    return a < b ? a : b;
}
```

```
double max(double a, double b)
{
    return a > b ? a : b;
}

bool inter(point a, point b, point c, point d)
{
    if (min(a.x, b.x) > max(c.x, d.x) || min(a.y, b.y) > max(c.y, d.y) || min(c.x, d.x) > max(a.x, b.x) || min(c.y, d.y) > max(a.y, b.y))
    {
        return 0;
    }
    double h, i, j, k;
    h = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    i = (b.x - a.x) * (d.y - a.y) - (b.y - a.y) * (d.x - a.x);
    j = (d.x - c.x) * (a.y - c.y) - (d.y - c.y) * (a.x - c.x);
    k = (d.x - c.x) * (b.y - c.y) - (d.y - c.y) * (b.x - c.x);
    return h * i <= eps && j * k <= eps;
}
```

图论

二分图匹配匈牙利算法

```
const int N=500;
int mp[N][N];
int match[N],vis[N];
int k,m,n;  //k是连接数 m是左边个数 n是右边个数
bool dfs(int x)
{
    for(int i=1; i<=n; i++)
    {
        if(mp[x][i]&&!vis[i]) //可以连接并且未连接过
        {
            vis[i]=1;
            if(match[i]==0||dfs(match[i]))
            {
                match[i]=x;
                return 1;
            }
        }
    }
    return 0;
}
int main()
{
    int x,y;
    while(scanf("%d",&k)&&k)
    {
        scanf("%d%d",&m,&n);
        memset(mp,0,sizeof(mp));
        memset(match,0,sizeof(vis));
        for(int i=0; i<k; i++)
        {
            scanf("%d%d",&x,&y);
            mp[x][y]=1;
        }
        int sum=0;
        for(int i=1; i<=m; i++)
        {
            memset(vis,0,sizeof(vis));
            if(dfs(i)) sum++;
        }
        printf("%d\n",sum);
    }
    return 0;
}
```

KM二分图最有匹配

```
const int qwq=0x7fffffff;
int w[1000][1000];  //w数组记录边权值
int line[1000],usex[1000],usey[1000],cx[1000],cy[1000];  //line数组记录右边端点所连的左端点， usex，usey数组记录是否曾访问过，也是判断是否在增广路上，cx，cy数组就是记录点的顶标
int n,ans,m;  //n左m右
bool find(int x){
    usex[x]=1;
    for (int i=1;i<=m;i++){
        if ((usey[i]==0)&&(cx[x]+cy[i]==w[x][i])){   //如果这个点未访问过并且它是子图里面的边
            usey[i]=1;
            if ((line[i]==0)||find(line[i])){   //如果这个点未匹配或者匹配点能更改
                line[i]=x;
                return true;
            }
        }
    }
    return false;
}
int km(){
    for (int i=1;i<=n;i++){  //分别对左边点依次匹配
        while (true){
        int d=qwq;
        memset(usex,0,sizeof(usex));
        memset(usey,0,sizeof(usey));
```

```
        if (find(i)) break;  //直到成功匹配才换下一个点匹配
        for (int j=1;j<=n;j++)
            if (usex[j])
             for (int k=1;k<=m;k++)
             if (!usey[k]) d=min(d,cx[j]+cy[k]-w[j][k]);  //计算d值
        if (d==qwq) return -1;
        for (int j=1;j<=n;j++)
         if (usex[j]) cx[j]-=d;
        for (int j=1;j<=m;j++)
         if (usey[j]) cy[j]+=d;     //添加新边
       }
    }
    ans=0;
    for (int i=1;i<=m;i++)
     ans+=w[line[i]][i];
    return ans;
}
int main(){
    while (~scanf("%d%d",&n,&m)){
    memset(cy,0,sizeof(cy));
    memset(w,0,sizeof(w));
    memset(cx,0,sizeof(cx));
    for (int i=1;i<=n;i++){
     int d=0;
     for (int j=1;j<=n;j++){
      scanf("%d",&w[i][j]);
      d=max(d,w[i][j]);   //此处顺便初始化左边点的顶标
       }
    cx[i]=d;
    }
    memset(line,0,sizeof(line));
    printf("%d\n",km());
}
    return 0;
}
```

一般图匹配带花树算法

```
const int maxn = 300;

int N;
bool G[maxn][maxn];
int match[maxn];
bool InQueue[maxn], InPath[maxn], InBlossom[maxn];
int head, tail;
int Queue[maxn];
int start, finish;
int NewBase;
int father[maxn], Base[maxn];
int Count;

void CreateGraph()
{
    int u, v;
    memset(G, 0, sizeof(G));
    scanf("%d", &N);
    while (scanf("%d%d",&u,&v) != EOF)
    {
        G[u][v] = G[v][u] = 1;
    }
}

void Push(int u)
{
    Queue[tail++] = u;
    InQueue[u] = 1;
}

int Pop()
{
    int res = Queue[head++];
    return res;
}

int FindCommonAncestor (int u, int v)
{
    memset(InPath, 0, sizeof(InPath));
    while (true)
    {
        u = Base[u];
        InPath[u] = 1;
        if (u == start)
        {
            break;
        }
        u = father[match[u]];
    }
    while (true)
    {
        v = Base[v];
        if (InPath[v])
        {
            break;
        }
        v = father[match[v]];
```

```
    }
    return v;
}

void ResetTrace(int u)
{
    int v;
    while (Base[u] != NewBase)
    {
        v = match[u];
        InBlossom[Base[u]] = InBlossom[Base[v]] = 1;
        u = father[v];
        if (Base[u] != NewBase)
        {
            father[u] = v;
        }
    }
}

void BlossomContract(int u, int v)
{
    NewBase = FindCommonAncestor(u, v);
    memset(InBlossom, 0, sizeof(InBlossom));
    ResetTrace(u);
    ResetTrace(v);
    if (Base[u] != NewBase)
    {
        father[u]=v;
    }
    if (Base[v] != NewBase)
    {
        father[v]=u;
    }
    for (int tu=1; tu <= N; tu++)
    {
        if (InBlossom[Base[tu]])
        {
            Base[tu] = NewBase;
            if (!InQueue[tu])
            {
                Push(tu);
            }
        }
    }
}

void FindAugmentingPath()
{
    memset(InQueue, 0, sizeof(InQueue));
    memset(father, 0, sizeof(father));
    for (int i = 1; i <= N; i++)
    {
        Base[i] = i;
    }
    head = tail = 1;
    Push(start);
    finish = 0;
    while (head < tail)
    {
        int u = Pop();
        for (int v = 1; v <= N; v++)
        {
            if (G[u][v] && (Base[u] != Base[v]) && match[u] != v)
            {
                if ((v == start) || ((match[v] > 0) && father[match[v]] > 0))
                {
                    BlossomContract(u, v);
                }
                else if (father[v] == 0)
                {
                    father[v] = u;
                    if (match[v] > 0)
                    {
                        Push(match[v]);
                    }
                    else
                    {
                        finish = v;
                        return ;
                    }
                }
            }
        }
    }
}

void AugmentPath()
{
    int u, v, w;
    u = finish;
    while (u > 0)
    {
        v = father[u];
        w = match[v];
        match[v] = u;
        match[u] = v;
        u = w;
```

```
    }
}

void Edmonds()
{
    memset(match, 0, sizeof(match));
    for (int u = 1; u <= N; u++)
    {
        if (match[u] == 0)
        {
            start = u;
            FindAugmentingPath();
            if (finish > 0)
            {
                AugmentPath();
            }
        }
    }
}

void PrintMatch()
{
    Count = 0;
    for (int u = 1; u <= N; u++)
    {
        if (match[u] > 0)
        {
            Count++;
        }
    }
    printf("%d\n", Count);
    for (int u = 1; u <= N; u++)
    {
        if (u < match[u])
        {
            printf("%d %d\n", u, match[u]);
        }
    }
}

int main()
{
    CreateGraph();
    Edmonds();      //  进行匹配
    PrintMatch();   //  输出匹配
    return 0;
}
```

LCA问题

```
/*
 * LCA在线算法(倍增法)
 */
const int MAXN = 10010;
const int DEG = 20;
struct Edge
{
    int to, next;
} edge[MAXN * 2];

int head[MAXN], tot;
void addedge(int u, int v)
{
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}

void init()
{
    tot = 0;
    memset(head, -1, sizeof(head));
}

int fa[MAXN][DEG];      //  fa[i][j]表示结点i的第2^j个祖先
int deg[MAXN];          //  深度数组

void BFS(int root)
{
    queue<int>que;
    deg[root] = 0;
    fa[root][0] = root;
    que.push(root);
    while (!que.empty())
    {
        int tmp = que.front();
        que.pop();
        for (int i = 1; i < DEG; i++)
        {
            fa[tmp][i] = fa[fa[tmp][i - 1]][i - 1];
        }
        for (int i = head[tmp]; i != -1; i = edge[i].next)
        {
            int v = edge[i].to;
            if (v == fa[tmp][0])
```

```
            {
                continue;
            }
            deg[v] = deg[tmp] + 1;
            fa[v][0] = tmp;
            que.push(v);
        }
    }
}

int LCA(int u, int v)
{
    if (deg[u] > deg[v])
    {
        swap(u, v);
    }
    int hu = deg[u], hv = deg[v];
    int tu = u, tv = v;
    for (int det = hv-hu, i = 0; det ; det >>= 1, i++)
    {
        if (det & 1)
        {
            tv = fa[tv][i];
        }
    }
    if (tu == tv)
    {
        return tu;
    }
    for (int i = DEG - 1; i >= 0; i--)
    {
        if (fa[tu][i] == fa[tv][i])
        {
            continue;
        }
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];
}

bool flag[MAXN];

int main()
{
    int T;
    int n;
    int u, v;
    scanf("%d", &T);

    while(T--)
    {
        scanf("%d", &n);
        init();
        memset(flag, false, sizeof(flag));
        for (int i = 1; i < n; i++)
        {
            scanf("%d%d", &u, &v);
            addedge(u, v);
            addedge(v, u);
            flag[v] = true;
        }
        int root;
        for (int i = 1; i <= n; i++)
        {
            if (!flag[i])
            {
                root = i;
                break;
            }
        }
        BFS(root);
        scanf("%d%d", &u, &v);
        printf("%d\n", LCA(u, v));
    }
    return 0;
}
```

拓扑排序

```
/*
 *  拓扑排序
 *  INIT:edge[][]置为图的邻接矩阵;cnt[0...i...n-1]:顶点i的入度。
 */
const int MAXV = 1010;

int edge[MAXV][MAXV];
int cnt[MAXV];

void TopoOrder(int n)
{
    int i, top = -1;
    for (i = 0; i < n; ++i)
    {
        if (cnt[i] == 0)
```

```
        {   //  下标模拟堆栈
            cnt[i] = top;
            top = i;
        }
    }
    for (i = 0; i < n; i++)
    {
        if (top == -1)
        {
            printf("存在回路\n");
            return ;
        }
        else
        {
            int j = top;
            top = cnt[top];
            printf("%d", j);
            for (int k = 0; k < n; k++)
            {
                if (edge[j][k] && (--cnt[k]) == 0)
                {
                    cnt[k] = top;
                    top = k;
                }
            }
        }
    }
}
```

2-sat问题

```
/*
 *  2-sat 问题
 *  N个集团,每个集团2个人,现在要想选出尽量多的人,
 *  且每个集团只能选出一个人。如果两人有矛盾,他们不能同时被选中
 *  问最多能选出多少人
 */
const int MAXN = 3010;
int n, m;
int g[3010][3010], ct[3010], f[3010];
int x[3010], y[3010];
int prev[MAXN], low[MAXN], stk[MAXN], sc[MAXN];
int cnt[MAXN];
int cnt0, ptr, cnt1;
void dfs(int w)
{
    int min(0);
    prev[w] = cnt0++;
    low[w] = prev[w];
    min = low[w];
    stk[ptr++] = w;
    for (int i = 0; i < ct[w]; ++i)
    {
        int t = g[w][i];
        if (prev[t] == -1)
        {
            dfs(t);
        }
        if (low[t] < min)
        {
            min = low[t];
        }
    }
    if (min < low[w])
    {
        low[w] = min;
        return ;
    }
    do
    {
        int v = stk[--ptr];
        sc[v] = cnt1;
        low[v] = MAXN;
    } while(stk[ptr] != w);
    ++cnt1;
    return ;
}

void Tarjan(int N)
{   //  传入N为点数,结果保存在sc数组中,同一标号的点在同一个强连通分量内,
    //  强连通分量数为cnt1
    cnt0 = cnt1 = ptr = 0;
    int i;
    for (i = 0; i < N; ++i)
    {
        prev[i] = low[i] = -1;
    }
    for (i = 0; i < N; ++i)
    {
        if (prev[i] == -1)
        {
            dfs(i);
        }
    }
    return ;
```

```
}

int solve()
{
    Tarjan(n);
    for (int i = 0; i < n; i++)
    {
        if (sc[i] == sc[f[i]])
        {
            return 0;
        }
    }
    return 1;
}

int check(int Mid)
{
    for (int i = 0; i < n; i++)
    {
        ct[i] = 0;
    }
    for (int i = 0; i < Mid; i++)
    {
        g[f[x[i]]][ct[f[x[i]]]++] = y[i];
        g[f[y[i]]][ct[f[y[i]]]++] = x[i];
    }
    return solve();
}

int main()
{
    while (scanf("%d%d", &n, &m) != EOF && n + m)
    {
        for (int i = 0; i < n; i++)
        {
            int p, q;
            scanf("%d%d", &p, &q);
            f[p] = q, f[q] = p;
        }
        for (int i = 0; i < m; i++)
        {
            scanf("%d%d", &x[i], &y[i]);
        }
        n *= 2;
        int Min = 0, Max = m + 1;
        while (Min + 1 < Max)
        {
            int Mid = (Min + Max) / 2;
            if (check(Mid))
            {
                Min = Mid;
            }
            else
            {
                Max = Mid;
            }
        }
        printf("%d\n", Min);
    }
    return 0;
}
```

树的重心

```
struct CenterTree{
    int n;
    int ans;
    int siz;
    int son[maxn];
    void dfs(int u,int pa){
        son[u]=1;
        int res=0;
        for (int i=head[u];i!=-1;i=edges[i].next){
            int v=edges[i].to;
            if (v==pa) continue;
            if (vis[v]) continue;
            dfs(v,u);
            son[u]+=son[v];
            res=max(res,son[v]-1);
        }
        res=max(res,n-son[u]);
        if (res<siz){
            ans=u;
            siz=res;
        }
    }
    int getCenter(int x){
        ans=0;
        siz=INF;
        dfs(x,-1);
        return ans;
    }
}Cent;
```

莫队算法

```
const int MAXN = 50010;
const int MAXM = 50010;
struct Query
{
    int L, R, id;
} node[MAXM];

long long gcd(long long a, long long b)
{
    if (b == 0)
    {
        return a;
    }
    return gcd(b, a % b);
}
struct Ans
{
    long long a, b; //  分数a/b
    void reduce()   //  分数化简
    {
        long long d = gcd(a, b);
        a /= d;
        b /= d;
        return ;
    }
} ans[MAXM];

int a[MAXN];
int num[MAXN];
int n, m, unit;

bool cmp(Query a, Query b)
{
    if (a.L / unit != b.L / unit)
    {
        return a.L / unit < b.L / unit;
    }
    else
    {
        return a.R < b.R;
    }
}

void work()
{
    long long temp = 0;
    memset(num, 0, sizeof(num));
    int L = 1;
    int R = 0;
    for (int i = 0; i < m; i++)
    {
        while (R < node[i].R)
        {
            R++;
            temp -= (long long)num[a[R]] * num[a[R]];
            num[a[R]]++;
            temp += (long long)num[a[R]] * num[a[R]];
        }
        while (R > node[i].R)
        {
            temp -= (long long)num[a[R]] * num[a[R]];
            num[a[R]]--;
            temp += (long long)num[a[R]] * num[a[R]];
            R--;
        }
        while (L < node[i].L)
        {
            temp -= (long long)num[a[L]] * num[a[L]];
            num[a[L]]--;
            temp += (long long)num[a[L]] * num[a[L]];
            L++;
        }
        while (L > node[i].L)
        {
            L--;
            temp -= (long long)num[a[L]] * num[a[L]];
            num[a[L]]++;
            temp += (long long)num[a[L]] * num[a[L]];
        }
        ans[node[i].id].a = temp - (R - L + 1);
        ans[node[i].id].b = (long long)(R - L + 1) * (R - L);
        ans[node[i].id].reduce();
    }
    return ;
}

int main()
{
    while (scanf("%d%d", &n, &m) == 2)
    {
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
        }
```

```
        for (int i = 0; i < m; i++)
        {
            node[i].id = i;
            scanf("%d%d", &node[i].L, &node[i].R);
        }
        unit = (int)sqrt(n);
        sort(node,node+m,cmp);
        work();
        for (int i = 0; i < m; i++)
        {
            printf("%lld/%lld\n", ans[i].a, ans[i].b);
        }
    }
    return 0;
}
```

最小割

```
const int inf = 0x3f3f3f3f;   //  max of res
const int maxw = 1000;        //  maximum edge weight
const int V = 10010;
int g[V][V], w[V];
int a[V], v[V], na[V];

int minCut(int n)
{
    int i, j, pv, zj;
    typec best = maxw * n * n;
    for (i = 0; i < n; i++)
    {
        v[i] = i;   //  vertex: 0 ~ n-1
    }
    while (n > 1)
    {
        for (a[v[0]] = 1, i = 1; i < n; i++)
        {
            a[v[i]] = 0;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
        }
        for (pv = v[0], i = 1; i < n; i++)
        {
            for (zj = -1, j = 1; j < n; j++)
            {
                if (!a[v[j]] && (zj < 0 || w[j] > w[zj]))
                {
                    zj = j;
                }
            }
            a[v[zj]] = 1;
            if (i == n - 1)
            {
                if (best > w[zj])
                {
                    best = w[zj];
                }
                for (i = 0; i < n; i++)
                {
                    g[v[i]][pv] = g[pv][v[i]] += g[v[zj]][v[i]];
                }
                v[zj] = v[--n];
                break;
            }
            pv = v[zj];
            for (j = 1; j < n; j++)
            {
                if(!a[v[j]])
                {
                    w[j] += g[v[zj]][v[j]];
                }
            }
        }
    }
    return best;
}
```

最大1矩阵

```
const int N = 1000;
bool a[N][N];
int Run(const int &m, const int &n)     // a[1...m][1...n]
{                                       // O(m*n)
    int i, j, k, l, r, max=0;
    int col[N];
    for (j = 1; j <= n; j++)
    {
        if (a[1][j] == 0 )
        {
            col[j] = 0;
        }
        else
        {
            for (k = 2; k <= m && a[k][j] == 1; k++);
            col[j] = k - 1;
```

```
        }
    }
    for (i = 1; i <= m; i++)
    {
        if (i > 1)
        {
            for (j = 1; j <= n; j++)
            {
                if (a[i][j] == 0)
                {
                    col[j] = 0;
                }
                else
                {
                    if (a[i - 1][j] == 0)
                    {
                        for (k = i + 1; k <= m && a[k][j] == 1; k++);
                        col[j] = k-1;
                    }
                }
            }
        }
        for (j = 1; j <= n; j++)
        {
            if (col[j] >= i)
            {
                for (l = j - 1; l > 0 && col[l] >= col[j]; --l);
                l++;
                for (r = j + 1; r <= n && col[r] >= col[j]; ++r);
                r--;
                int res = (r - l + 1) * (col[j] - i + 1);
                if( res > max )
                {
                    max = res;
                }
            }
        }
    }
    return max;
}
```

Dijkstra 优先队列优化

```
const int maxn=200005;
#define INF 1e9
int n,m;
struct Edge
{
    int u,v,w;
    Edge(int u,int v,int w):u(u),v(v),w(w) {}
};

struct Node
{
    int d,u;
    Node(int d,int u):d(d),u(u) {}
    bool operator <(const Node &rhs)const
    {
        return d > rhs.d;
    }
};
vector<Edge> edges;
vector<int> G[maxn];
bool done[maxn];
int d[maxn];

void init()
{
    for(int i=0; i<n; i++)
        G[i].clear();
    edges.clear();
}

void AddEdge(int u,int v,int w)
{
    edges.push_back(Edge(u,v,w));
    int mm=edges.size();
    G[u].push_back(mm-1);
}

void dijkstra()
{
    priority_queue<Node> Q;
    for(int i=0; i<=n; i++) d[i]=INF;
    d[0]=0;
    memset(done,0,sizeof(done));
    Q.push(Node(d[0],0));

    while(!Q.empty())
    {
        Node x=Q.top();
        Q.pop();
        int u=x.u;
        if(done[u]) continue;
        done[u]=true;
```

```
        for(int i=0; i<G[u].size(); i++)
        {
            Edge e=edges[G[u][i]];
            if(d[e.v] > d[u]+e.w)
            {
                d[e.v] = d[u]+e.w;
                Q.push(Node(d[e.v],e.v));
            }
        }
    }
}
int main()
{
    while(scanf("%d%d",&n,&m)!=EOF)
    {

        init();
        for(int i=0; i<m; i++)
        {
            int u,v,d;
            scanf("%d%d%d",&u,&v,&d);
            u--,v--;

            AddEdge(u,v,d);
            AddEdge(v,u,d);
        }
        dijkstra();
                if(d[n-1]==INF) printf("qwb baka\n");
                else
        printf("%d\n",d[n-1]);
    }
    return 0;
}
```

树的点分治问题

求树上边长小于k的边的条数，树上统计问题

```
int head[inf],next[inf<<1],to[inf<<1],len[inf<<1],cnt;
int maxn[inf],siz[inf],G,subsiz;
bool vis[inf];
int dp[inf<<1],dep[inf<<1];//dp[]存储到根节点的距离；dep[]是用来sort的，dep[0]表示dep数组中元素的个数
int n,k,ans=0;

void init(void){
    memset(vis,false,sizeof vis);
    memset(head,0,sizeof head);
    cnt=0;ans=0;
}

void addedge(int u,int v,int w){
    to[++cnt]=v;len[cnt]=w;
    next[cnt]=head[u];head[u]=cnt;
}

void getG(int u,int f){//找重心
    siz[u]=1;maxn[u]=0;
    for (int i=head[u];i;i=next[i]){
        int v=to[i];if (v!=f && !vis[v]){
            getG(v,u);
            siz[u]+=siz[v];
            maxn[u]=max(maxn[u],siz[u]);
        }

    }maxn[u]=max(maxn[u],subsiz-siz[u]);
    G=(maxn[u]<maxn[G])?u:G;
}

void dfs(int u,int f){//dfs确定每个点到根节点的距离
    dep[++dep[0]]=dp[u];
    for (int i=head[u];i;i=next[i]){
        int v=to[i];if (v!=f && !vis[v]){
            dp[v]=dp[u]+len[i];
            dfs(v,u);
        }
    }
}

int calc(int u,int inidep){//inidep是这一点相对于根节点的初始距离
    dep[0]=0;
    dp[u]=inidep;
    dfs(u,0);
    sort(dep+1,dep+1+dep[0]);
    int sum=0;
    for (int l=1,r=dep[0];l<r;){//计算合法点对数目
        if (dep[l]+dep[r]<=k) {sum+=r-l;l++;}
        else r--;
    }
    return sum;
}

void divide(int g){ //递归，找到重心并以重心为根节点进行计算，再对子树递归处理
    ans+=calc(g,0);
    vis[g]=true;
```

```
    for (int i=head[g];i;i=next[i]){
        int v=to[i]; if (!vis[v]){
            ans-=calc(v,len[i]);
            maxn[0]=subsiz=siz[v];G=0;getG(v,0);
            divide(G);
        }
    }
}

int main(){
    while(scanf("%d%d",&n,&k)==2){
        if (!n && !k) break;
        init();
        for (int i=1,u,v,w;i<n;i++){
            scanf("%d%d%d",&u,&v,&w);
            addedge(u,v,w);addedge(v,u,w);
        }
        subsiz=maxn[0]=n;G=0;getG(1,0);
        divide(G);
        printf("%d\n",ans);
    }
    return 0;
}
```

数论

阶乘

```
/*
 *  阶乘最后非零位 复杂度O(nlongn)
 *  返回改为，n以字符串方式传入
 */
#define MAXN 10000

const int mod[20] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8, 4, 4, 8, 4, 6, 8, 8, 6, 8, 2};

int lastDigit(char *buf)
{
    int len = (int)strlen(buf);
    int a[MAXN], i, c, ret = 1;
    if (len == 1)
    {
        return mod[buf[0] - '0'];
    }
    for (i = 0; i < len; i++)
    {
        a[i] = buf[len - 1 - i] - '0';
    }
    for (; len; len -= !a[len - 1])
    {
        ret = ret * mod[a[1] % 2 * 10 + a[0]] % 5;
        for (c = 0, i = len - 1; i >= 0; i--)
        {
            c = c * 10 + a[i];
            a[i] = c / 5;
            c %= 5;
        }
    }
    return ret + ret % 2 * 5;
}
```

阶乘长度

```
#define PI 3.1415926

int main()
{
    int n, a;
    while (~scanf("%d", &n))
    {
        a = (int)((0.5 * log(2 * PI * n) + n * log(n) - n) / log(10));
        printf("%d\n", a + 1);
    }
    return 0;
}
```

基姆拉尔森公式

```
/*
 *  已知1752年9月3日是Sunday，并且日期控制在1700年2月28日后
 */
char name[][15] = { "monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"};

int main()
{
    int d, m, y, a;
    printf("Day: ");
    scanf("%d", &d);
    printf("Month: ");
    scanf("%d", &m);
    printf("Year: ");
    scanf("%d", &y);
    //  1月2月当作前一年的13,14月
```

```
    if (m == 1 || m == 2)
    {
        m += 12;
        y--;
    }
    //  判断是否在1752年9月3日之前,实际上合并在一起倒更加省事
    if ((y < 1752) || (y == 1752 && m < 9) || (y == 1752 && m == 9 && d < 3))
    {
        //  因为日期控制在1700年2月28日后，所以不用考虑整百年是否是闰年
        a = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 + 5) % 7;
    }
    else
    {
        //  这里需要考虑整百年是否是闰年的情况
        a = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7;  //  实际上这个可以当做公式背下来
    }
    printf("it's a %s\n", name[a]);
    return 0;
}
```

反素数

求最小的因子个数为n个正整数

```
typedef unsigned long long ULL;

const ULL INF = ~0ULL;
const int MAXP = 16;

int prime[MAXP] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53};

int n;
ULL ans;

void dfs(int dept, ULL tmp, int num, int pre)    //  深度/当前值/约数个数/上一个数
{
    if (num > n)
    {
        return;
    }
    if (num == n && ans > tmp)
    {
        ans = tmp;
    }
    for (int i = 1; i <= pre; i++)
    {
        if (ans / prime[dept] < tmp)
        {
            break;
        }
        dfs(dept + 1, tmp *= prime[dept], num * (i + 1), i);
    }
}

int main()
{
    while (cin >> n)
    {
        ans = INF;
        dfs(0, 1, 1, 15);
        cout << ans << endl;
    }
    return 0;
}
```

求n以内的因子最多的数（不止一个则取最小）

```
typedef long long ll;
const int MAXP = 16;
const int prime[MAXP] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53};
ll n, res, ans;
void dfs(ll cur, ll num, int key, ll pre)  //  当前值/当前约数数量/当前深度/上一个数
{
    if (key >= MAXP)
    {
        return ;
    }
    else
    {
        if (num > ans)
        {
            res = cur;
            ans = num;
        }
        else if (num == ans)    //  如果约数数量相同，则取较小的数
        {
            res = min(cur, res);
        }

        ll i;
        for ( i = 1; i <= pre; i++)
        {
            if (cur <= n / prime[key])  //  cur*prime[key]<=n
            {
                cur *= prime[key];
```

```
                dfs(cur, num * (i + 1), key + 1, i);
            }
            else
            {
                break;
            }
        }
    }
}

void solve()
{
    res = 1;
    ans = 1;

    dfs(1, 1, 0, 15);
    cout << res << ' ' << ans << endl;
}

int main(int argc, const char * argv[])
{
    int T;
    cin >> T;

    while (T--)
    {
        cin >> n;
        solve();
    }
    return 0;
}
```

容斥

```
const int MAXN = 1111;

int n;
double ans;
double p[MAXN];

void dfs(int x, int tot, double sum)    //  dfs(1, 0, ?)
{
    if (x == n + 1)
    {
        if (sum == 0.0)
        {
            return ;
        }

        if (tot & 1)
        {
            ans += 1 / sum; //  公式随意变
        }
        else
        {
            ans -= 1 / sum;
        }
        return ;
    }

    dfs(x + 1, tot, sum);
    dfs(x + 1, tot + 1, sum + p[x]);
}
```

整数划分

P(n) = ∑{P(n - k(3k - 1) / 2 + P(n - k(3k + 1) / 2 | k ≥ 1} n < 0时，P(n) = 0, n = 0时, P(n) = 1即可

```
//  划分元素可重复任意次
#define f(x) (((x) * (3 * (x) - 1)) >> 1)
#define g(x) (((x) * (3 * (x) + 1)) >> 1)
const int MAXN = 1e5 + 10;
const int MOD = 1e9 + 7;
int n, ans[MAXN];
int main()
{
    scanf("%d", &n);

    ans[0] = 1;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; f(j) <= i; ++j)
        {
            if (j & 1)
            {
                ans[i] = (ans[i] + ans[i - f(j)]) % MOD;
            }
            else
            {
                ans[i] = (ans[i] - ans[i - f(j)] + MOD) % MOD;
            }
        }
        for (int j = 1; g(j) <= i; ++j)
        {
```

```
            if (j & 1)
            {
                ans[i] = (ans[i] + ans[i - g(j)]) % MOD;
            }
            else
            {
                ans[i] = (ans[i] - ans[i - g(j)] + MOD) % MOD;
            }
        }
    }

    printf("%d\n", ans[n]);

    return 0;
}
```

生成树计数

```
//  求生成树计数部分代码,计数对10007取模
#define M 305
struct point{
    int x, y;
}p[M];

int C[M][M], G[M][M];
int mod = 10007;

int dis (point a, point b)
{
    return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
}
void Egcd (int a, int b, int &x, int &y)
{
    if (b == 0)
    {
        x = 1, y = 0;
        return ;
    }
    Egcd (b, a%b, x, y);
    int tp = x;
    x = y;
    y = tp - a/b*y;
}
int det (int n)    //计算n阶行列式
{
    int i, j, k, ans = 1, x, y, flg = 1;
    for (i = 0; i < n; i++)
    {
        if (C[i][i] == 0)
        {
            for (j = i+1; j < n; j++)
                if (C[j][i])
                    break;
            if (j == n) return -1;
            flg = !flg;
            for (k = i; k < n; k++)
                swap (C[i][k], C[j][k]);
        }
        ans = ans * C[i][i] % mod;
        Egcd (C[i][i], mod, x, y);
        x = (x%mod + mod) % mod;            //注意保证取余结果为最小非负数
        for (k = i+1; k < n; k++)
            C[i][k] = C[i][k] * x % mod;
        for (j = i+1; j < n; j++)
            if (C[j][i] != 0) for (k = i+1; k < n; k++)
                C[j][k] = ((C[j][k] - C[i][k]*C[j][i])%mod + mod) % mod;
                //注意保证取余结果为最小非负数
    }
    if (flg) return ans;
    return mod-ans;
}
int main ()
{
    int i, j, k, t, n, r;
    scanf ("%d", &t);
    while (t--)
    {
        scanf ("%d%d", &n, &r);
        for (i = 0; i < n; i++)
            scanf ("%d%d", &p[i].x, &p[i].y);
        memset (G, 0, sizeof(G));
        for (i = 0; i < n; i++)    //建图
        {
            for (j = i + 1; j < n; j++)
            {
                int tp = dis (p[i], p[j]);
                if (tp > r*r) continue;
                for (k = 0; k < n; k++)
                {
                    if (k == i || k == j) continue;
                    if ((p[i].x-p[k].x)*(p[j].y-p[k].y) ==
                        (p[j].x-p[k].x)*(p[i].y-p[k].y) &&
                        dis (p[i], p[k]) < tp && dis (p[j], p[k]) < tp)
                        break;
                }
```

```
                if (k == n) G[i][j] = G[j][i] = 1;
            }
        }
        memset (C, 0, sizeof(C));
        for (i = 0; i < n; i++)
            for (j = i + 1; j < n; j++)
                if (G[i][j])
                    ++C[i][i], ++C[j][j];
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
            {
                C[i][j] -= G[i][j];
                C[i][j] = (C[i][j]%mod + mod) % mod;
                //注意保证取余结果为最小非负数
            }
        printf ("%d\n", det(n-1));
    }
    return 0;
}
```

三分法

对于任意一个上凸函数，选取函数上任意两个点A,B（xA<xB），若满足yA<yB,那么该函数的极值点必然在[xA,+∞)中，若满足yA>yB，那么该函数极值点必然在(-∞,xB]中，若满足yA=yB，那么该函数的极值点必然在[xA,xB]中。

对于任意一个下凸函数，选取函数上任意两个点A,B（xA<xB），若满足yA<yB,那么该函数的极值点必然在(-∞,xB]中，若满足yA>yB，那么该函数极值点必然在[xA,+∞)中，若满足yA=yB，那么该函数的极值点必然在[xA,xB]中。

```
void Solve()
{
    double left, right, m1, m2, m1_value, m2_value;
    left = MIN;
    right = MAX;
    while (left + EPS < right)
    {
        m1 = left + (right - left)/3;
        m2 = right - (right - left)/3;
        m1_value = f(m1);
        m2_value = f(m2);
        //假设求解极大值
        if (m1_value >= m2_value)
            right = m2;
        else
            left = m1;
    }
}
```

快速乘

```
//快速乘法取模
int qmul_mod(int a,int b,int mod){
    int ans=0;
    while(b){
        if((b%=mod)&1)ans+=a%=mod;//这里需要b%=mod 以及a%=mod
        b>>=1;a<<=1;
    }
    return ans%mod;  //ans也需要对mod取模
}
```

快速幂

```
int qpow_mod(int a,int b,int mod){
    if(a==0)return 0;
    int ans=1;
    while(b){
        if(b&1)ans=(ans%mod)*(a%mod);//如果确定数据不会爆的话，可写成 ans*=a%=mod;
        b>>=1;a*=a%=mod;//等价于a=(a%mod)*(a%mod)，且将一个模运算通过赋值代替，提高了效率
    }
    return ans%mod;//数据不会爆的话，这里的%运算会等价于第5中不断重复的 ans%mod
}
```

快速幂求逆元

```
LL pow_mod(LL a, LL b, LL p){//a的b次方求余p
    LL ret = 1;
    while(b){
        if(b & 1) ret = (ret * a) % p;
        a = (a * a) % p;
        b >>= 1;
    }
    return ret;
}
LL Fermat(LL a, LL p){//费马求a关于b的逆元
        return pow_mod(a, p-2, p);
}
```

矩阵分治乘法

```
#define LEN 4
typedef struct
```

```
{
	int rowstart;
	int rowend;
	int colstart;
	int colend;
}Square;
int A[LEN][LEN]={{1,2,3,4},{3,4,5,6},{5,6,7,8},{7,8,9,10}};
int B[LEN][LEN]={{5,6,7,8},{7,8,9,10},{11,12,13,14},{15,16,17,18}};

void recurMult(int C[LEN][LEN],Square a,Square b)
{
	if(a.rowstart==a.rowend && a.colstart==a.colend && b.rowstart==b.rowend && b.colstart==b.colend)
	{
		C[a.rowstart][b.colstart]+=A[a.rowstart][a.colstart]*B[b.rowstart][b.colstart];
		return;
	}

  Square a11={a.rowstart,(a.rowstart+a.rowend)/2,a.colstart,(a.colstart+a.colend)/2};
	Square a12={a.rowstart,(a.rowstart+a.rowend)/2,(a.colstart+a.colend)/2+1,a.colend};
	Square a21={(a.rowstart+a.rowend)/2+1,a.rowend,a.colstart,(a.colstart+a.colend)/2};
	Square a22={(a.rowstart+a.rowend)/2+1,a.rowend,(a.colstart+a.colend)/2+1,a.colend};
	Square b11={b.rowstart,(b.rowstart+b.rowend)/2,b.colstart,(b.colstart+b.colend)/2};
	Square b12={b.rowstart,(b.rowstart+b.rowend)/2,(b.colstart+b.colend)/2+1,b.colend};
	Square b21={(b.rowstart+b.rowend)/2+1,b.rowend,b.colstart,(b.colstart+b.colend)/2};
	Square b22={(b.rowstart+b.rowend)/2+1,b.rowend,(b.colstart+b.colend)/2+1,b.colend};

	recurMult(C,a11,b11);
	recurMult(C,a12,b21);
	recurMult(C,a11,b12);
	recurMult(C,a12,b22);
	recurMult(C,a21,b11);
	recurMult(C,a22,b21);
	recurMult(C,a21,b12);
	recurMult(C,a22,b22);
}

void print(int a[LEN][LEN])
{
	for(int i=0;i<LEN;i++)
	{
		for(int j=0;j<LEN;j++)
		{
			printf("%4d ",a[i][j]);
		}
		printf("\n");
	}
}

int main()
{
	int C[LEN][LEN]={0};
	Square a={0,LEN-1,0,LEN-1};
	Square b={0,LEN-1,0,LEN-1};
	recurMult(C,a,b);
	print(C);
	getchar();
}
```

矩阵快速幂

```
const int MOD = 10000;
struct matrix {		//矩阵
	int m[2][2];
}ans;
matrix base = {1, 1, 1, 0};

matrix multi(matrix a, matrix b) {	//矩阵相乘，返回一个矩阵
	matrix tmp;
	for(int i = 0; i < 2; i++) {
		for(int j = 0; j < 2; j++) {
			tmp.m[i][j] = 0;
			for(int k = 0;  k < 2; k++)
				tmp.m[i][j] = (tmp.m[i][j] + a.m[i][k] * b.m[k][j]) % MOD;
		}
	}
	return tmp;
}

int matrix_pow(matrix a, int n) {	//矩阵快速幂，矩阵a的n次幂
	ans.m[0][0] = ans.m[1][1] = 1;  //初始化为单位矩阵
	ans.m[0][1] = ans.m[1][0] = 0;
	while(n) {
		if(n & 1) ans = multi(ans, a);
		a = multi(a, a);
		n >>= 1;
	}
	return ans.m[0][1];
}

int main() {
	int n;
	while(scanf("%d", &n), n != -1) {
		printf("%d\n", matrix_pow(base, n));
	}
	return 0;
```

```
}
```

GCD

```
int gcd(int x, int y) {
  if (!x || !y) {
    return x > y ? x : y;
  }
  for (int t; t = x % y, t; x = y, y = t) ;
  return y;
}
```

Extern_GCD

```
/* * 求x，y使得gcd(a, b) = a * x + b * y; */
int extgcd(int a, int b, int &x, int &y) {
  if (b == 0) { x = 1; y = 0; return a; }
  int d = extgcd(b, a % b, x, y);
  int t = x; x = y; y = t - a / b * y;
  return d;
}
```

欧拉函数

```
/* * 单独求解的本质是公式的应用 */
unsigned euler(unsigned x) {
  unsigned i, res = x;
  // unsigned == unsigned int
  for (i = 2; i < (int)sqrt(x * 1.0) + 1; i++) {
     if (!(x % i)) {
        res = res / i * (i - 1);
         while (!(x % i)) {
            x /= i;
            // 保证i一定是素数
        }
      }
    }
      if (x > 1) {
        res = res / x * (x - 1);
      }
       return res;
}
```

线性筛

```
/* * 同时得到欧拉函数和素数表 */
const int MAXN = 10000000;
bool check[MAXN + 10];
int phi[MAXN + 10];
int prime[MAXN + 10];
int tot; // 素数个数
void phi_and_prime_table(int N) {
  memset(check, false, sizeof(check));
  phi[1] = 1; tot = 0;
  for (int i = 2; i <= N; i++) {
     if (!check[i]) {
       prime[tot++] = i;
       phi[i] = i - 1;
      }
      for (int j = 0; j < tot; j++) {
        if (i * prime[j] > N) { break; }
        check[i * prime[j]] = true;
        if (i % prime[j] == 0) {
           phi[i * prime[j]] = phi[i] * prime[j]; break;
        } else {
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
      }
  }
  return ;
}
```

Ploya计数

```
/*
 *  c种颜色的珠子，组成长为s的项链，项链没有方向和起始位置
 */
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}

int main(int argc, const char * argv[])
{
    int c, s;
    while (cin >> c >> s)
    {
        int k;
        long long p[64];
```

```
        p[0] = 1;                   // power of c
        for (k = 0; k < s; k++)
        {
            p[k + 1] = p[k] * c;
        }
        // reflection part
        long long count = s & 1 ? s * p[s / 2 + 1] : (s / 2) * (p[s / 2] + p[s / 2 + 1]);
        // rotation part
        for (k = 1 ; k <= s ; k++)
        {
            count += p[gcd(k, s)];
            count /= 2 * s;
        }
        cout << count << '\n';
    }
    return 0;
}
```

problem 2

求斐波那契数列奇数项的和 推下规律 奇数项F[i]=4*F[i-3]+F[i-6] 打下表求和即可

problem3

求一个数的最大质因子是所有质数中的第几位
思路是打个表，把包含这个数作为质因子的数设置为这个数的位置，因为是从小到大，最后记录下的就是最大的质因子的位置

```
void init(){
    int i;
    int k = 1;
    for(i = 2 ; i < maxn ; ++i){//遍历数据范围内的所有数
        if(biao[i] == 0){//如果这一个数的最大质因子的位置还没有确定
            int j;
            for(j = 1 ; i*j < maxn ; ++j){//把含有这个质因子的所有数的位置都标记成这个质因子的位置
                biao[i*j] = k;
            }
            k++;//质因子的位置索引+1
        }
    }
}
```

输入一个n，最后只需输出biao[n]即可

字符串

后缀数组

```
/*
 *  suffix array
 *  倍增算法 O(n*logn)
 *  待排序数组长度为n,放在0~n-1中,在最后面补一个0
 *  da(str, sa, rank, height, n, m);
 *  例如:
 *  n = 8;
 *  num[] = { 1, 1, 2, 1, 1, 1, 1, 2, $ };      注意num最后一位为0,其他大于0
 *  rank[] = { 4, 6, 8, 1, 2, 3, 5, 7, 0 };     rank[0~n-1]为有效值,rank[n]必定为0无效值
 *  sa[] = { 8, 3, 4, 5, 0, 6, 1, 7, 2 };       sa[1~n]为有效值,sa[0]必定为n是无效值
 *  height[]= { 0, 0, 3, 2, 3, 1, 2, 0, 1 };    height[2~n]为有效值
 *  稍微改动可以求最长公共前缀，需要注意两串起始位置相同的情况
 *  另外需要注意的是部分数组需要开两倍空间大小
 */
const int MAXN = 20010;

int t1[MAXN];
int t2[MAXN];
int c[MAXN];    //  求SA数组需要的中间变量,不需要赋值

//  待排序的字符串放在s数组中,从s[0]到s[n-1],长度为n,且最大值小于m,
//  除s[n-1]外的所有s[i]都大于0,r[n-1]=0
//  函数结束以后结果放在sa数组中
bool cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a + l] == r[b + l];
}

void da(int str[], int sa[], int rank[], int height[], int n, int m)
{
    n++;
    int i, j, p, *x = t1, *y = t2;  //  第一轮基数排序,如果s的最大值很大,可改为快速排序
    for (i = 0; i < m; i++)
    {
        c[i] = 0;
    }
    for (i = 0; i < n; i++)
    {
        c[x[i] = str[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; j <= n; j <<= 1)
    {
        p = 0;
        //  直接利用sa数组排序第二关键字
        for (i = n - j; i < n; i++)
        {
            y[p++] = i;                 //  后面的j个数第二关键字为空的最小
        }
        for (i = 0; i < n; i++)
        {
            if (sa[i] >= j)
            {
                y[p++] = sa[i] - j;     //  这样数组y保存的就是按照第二关键字排序的结果
            }
        }
        //  基数排序第一关键字
        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];    //  根据sa和x数组计算新的x数组
        }
        swap(x, y);
        p = 1;
        x[sa[0]] = 0;
        for (i = 1; i < n; i++)
        {
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
        }
        if (p >= n)
        {
            break;
        }
        m = p;                          //  下次基数排序的最大值
    }
    int k = 0;
    n--;
    for (i = 0; i <= n; i++)
    {
        rank[sa[i]] = i;
    }
    for (i = 0; i < n; i++)
    {
        if (k)
        {
            k--;
        }
        j = sa[rank[i] - 1];
        while (str[i + k] == str[j + k])
        {
            k++;
        }
        height[rank[i]] = k;
    }
}

int _rank[MAXN], height[MAXN];
int RMQ[MAXN];
int mm[MAXN];

int best[20][MAXN];

void initRMQ(int n)
{
    mm[0] = -1;
    for (int i = 1; i <= n; i++)
    {
        mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
    }
    for (int i = 1; i <= n; i++)
    {
        best[0][i] = i;
    }
    for (int i = 1; i <= mm[n]; i++)
    {
        for (int j = 1; j + (1 << i) - 1 <= n; j++)
        {
            int a = best[i - 1][j];
            int b = best[i - 1][j + (1 << (i - 1))];
            if (RMQ[a] < RMQ[b])
            {
                best[i][j] = a;
            }
            else
```

```
            {
                best[i][j] = b;
            }
        }
    }
}

int askRMQ(int a, int b)
{
    int t;
    t = mm[b - a + 1];
    b -= (1 << t) - 1;
    a = best[t][a];
    b = best[t][b];
    return RMQ[a] < RMQ[b] ? a : b;
}

int lcp(int a, int b)
{
    a = _rank[a];
    b = _rank[b];
    if (a > b)
    {
        swap(a,b);
    }
    return height[askRMQ(a + 1, b)];
}

char str[MAXN];
int r[MAXN];
int sa[MAXN];

int main()
{
    while (scanf("%s", str) == 1)
    {
        int len = (int)strlen(str);
        int n = 2 * len + 1;
        for (int i = 0; i < len; i++)
        {
            r[i] = str[i];
        }
        for (int i = 0; i < len; i++)
        {
            r[len + 1 + i] = str[len - 1 - i];
        }
        r[len] = 1;
        r[n] = 0;
        da(r, sa, _rank, height, n, 128);
        for (int i = 1; i <= n; i++)
        {
            RMQ[i]=height[i];
        }
        initRMQ(n);
        int ans = 0, st = 0;
        int tmp;
        for (int i = 0; i < len; i++)
        {
            tmp = lcp(i, n - i);    //  偶对称
            if (2 * tmp > ans)
            {
                ans = 2 * tmp;
                st = i - tmp;
            }
            tmp=lcp(i, n - i - 1);  //  奇数对称
            if (2 * tmp - 1 > ans)
            {
                ans = 2 * tmp - 1;
                st = i - tmp + 1;
            }
        }
        str[st + ans] = 0;
        printf("%s\n", str + st);
    }
    return 0;
}
```

输入输出挂

```
template <class T>
bool scan_d(T &ret)
{
    char c;
    int sgn;
    T bit = 0.1;
    if (c=getchar(), c==EOF)
    {
        return 0;
    }
    while (c! = '-' && c != '.' && (c < '0' || c > '9'))
    {
        c = getchar();
    }
    sgn = (c == '-') ? -1 : 1;
    ret = (c == '-') ? 0 : (c - '0');
```

```
    while (c = getchar(), c >= '0' && c <= '9')
    {
        ret = ret * 10 + (c - '0');
    }
    if (c == ' ' || c == '\n')
    {
        ret *= sgn;
        return 1;
    }
    while (c = getchar(), c >= '0' && c <= '9')
    {
        ret += (c - '0') * bit, bit /= 10;
    }
    ret *= sgn;
    return 1;
}

template <class T>
inline void print_d(int x)
{
    if (x > 9)
    {
        print_d(x / 10);
    }
    putchar(x % 10 + '0');
}
```

AC自动机

```
/*
 *  求目标串中出现了几个模式串
 */
struct Trie{
    int next[500010][26], fail[500010], end[500010];
    int root, L;
    int newnode(){
        for (int i = 0; i < 26; i++){
            next[L][i] = -1;
        }
        end[L++] = 0;
        return L - 1;
    }
    void init(){
        L = 0;
        root = newnode();
    }

    void insert(char buf[]){
        int len = (int)strlen(buf);
        int now = root;
        for (int i = 0; i < len; i++){
            if (next[now][buf[i] - 'a'] == -1){
                next[now][buf[i] - 'a'] = newnode();
            }
            now = next[now][buf[i] - 'a'];
        }
        end[now]++;
    }
    void build(){
        queue<int>Q;
        fail[root] = root;
        for (int i = 0; i < 26; i++){
            if (next[root][i] == -1){
                next[root][i] = root;
            }
            else{
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }
        }
        while (!Q.empty()){
            int now = Q.front();
            Q.pop();
            for (int i = 0;i < 26;i++){
                if (next[now][i] == -1){
                    next[now][i] = next[fail[now]][i];
                }
                else{
                    fail[next[now][i]]=next[fail[now]][i];
                    Q.push(next[now][i]);
                }
            }
        }
    }

    int query(char buf[]){
        int len = (int)strlen(buf);
        int now = root;
        int res = 0;
        for (int i = 0; i < len; i++){
            now = next[now][buf[i] - 'a'];
            int temp = now;
            while (temp != root){
                res += end[temp];
                end[temp] = 0;
```

```
                temp = fail[temp];
            }
        }
        return res;
    }

    void debug(){
        for (int i = 0; i < L; i++){
            printf("id = %3d,fail = %3d,end = %3d,chi = [", i, fail[i], end[i]);
            for (int j = 0; j < 26; j++){
                printf("%2d", next[i][j]);
            }
            printf("]\n");
        }
    }
};

char buf[1000010];
Trie ac;
int main()
{
    int T;
    int n;
    scanf("%d", &T);
    while(T--){
        scanf("%d", &n);
        ac.init();
        for (int i = 0; i < n; i++){
            scanf("%s", buf);
            ac.insert(buf);
            printf("%s\n",buf );
        }
        ac.build();
        scanf("%s", buf);
        printf("%d\n", ac.query(buf));
    }
    return 0;
}
```

KMP算法

```
/*
字符串是从0开始的
Next数组是从1开始的
*/
const int N = 1000002;
int next[N];
char S[N], T[N];
int slen, tlen;

void getNext()
{
    int j, k;
    j = 0; k = -1; next[0] = -1;
    while(j < tlen)
        if(k == -1 || T[j] == T[k])
            next[++j] = ++k;
        else
            k = next[k];

}
int KMP_Index()
{
    int i = 0, j = 0;
    getNext();

    while(i < slen && j < tlen)
    {
        if(j == -1 || S[i] == T[j])
        {
            i++; j++;
        }
        else
            j = next[j];
    }
    if(j == tlen)
        return i - tlen;
    else
        return -1;
}
/*
返回模式串在主串S中出现的次数
*/
int KMP_Count()
{
    int ans = 0;
    int i, j = 0;

    if(slen == 1 && tlen == 1)
    {
        if(S[0] == T[0])
            return 1;
        else
            return 0;
    }
```

```
    getNext();
    for(i = 0; i < slen; i++)
    {
        while(j > 0 && S[i] != T[j])
            j = next[j];
        if(S[i] == T[j])
            j++;
        if(j == tlen)
        {
            ans++;
            j = next[j];
        }
    }
    return ans;
}
int main()
{
    int TT;
    int i, cc;
    cin>>TT;
    while(TT--)
    {
        cin>>S>>T;
        slen = strlen(S);
        tlen = strlen(T);
        cout<<"模式串T在主串S中首次出现的位置是: "<<KMP_Index()<<endl;
        cout<<"模式串T在主串S中出现的次数为: "<<KMP_Count()<<endl;
    }
    return 0;
}
```

附上strstr函数

```
/*
 *  strstr函数
 *  功能: 在串中查找指定字符串的第一次出现
 *  用法: char *strstr(char *strOne, char *strTwo);
 */
int main(int argc, const char * argv[])
{
    char strOne[] = "Borland International";
    char strTwo[] = "nation";
    char *ptr;
    ptr = strstr(strOne, strTwo);
    std::cout << ptr << '\n';
    return 0;
}
```

最长回文串问题

```
/*
 * 求最长回文子串
 */
const int MAXN = 110010;
char A[MAXN * 2];
int B[MAXN * 2];
void Manacher(char s[], int len)
{
    int l = 0;
    A[l++] = '$';   //0下标存储为其他字符
    A[l++] = '#';
    for (int i = 0; i < len; i++)
    {
        A[l++] = s[i];
        A[l++] = '#';
    }
    A[l] = 0;       //空字符
    int mx = 0;
    int id = 0;
    for (int i = 0; i < l; i++)
    {
        B[i] = mx > i ? std::min(B[2 * id - i], mx - i) : 1;
        while (A[i + B[i]] == A[i - B[i]])
        {
            B[i]++;
        }
        if (i + B[i] > mx)
        {
            mx = i + B[i];
            id = i;
        }
    }
    return ;
}
char s[MAXN];
int main(int argc, const char * argv[])
{
    while (std::cin >> s)
    {
        int len = (int)strlen(s);
        Manacher(s, len);
        int ans = 0;
        for (int i = 0; i < 2 * len + 2; i++)   //两倍长度并且首位插有字符，所以i < 2 * len + 2
        {
            ans = std::max(ans, B[i] - 1);
```

```
        }
        std::cout << ans << std::endl;
    }

    return 0;
}
```

```
void SUNDAY(char *text, char *patt)
{
    size_t temp[256];
    size_t *shift = temp;
    size_t i, patt_size = strlen(patt), text_size = strlen(text);
    cout << "size : " << patt_size << endl;
    for(i = 0; i < 256; i++)
    {
        *(shift+i) = patt_size + 1;
    }
    for(i = 0; i < patt_size; i++)
    {
        *(shift + (unsigned char)(*(patt+i))) = patt_size-i;    //  shift['s']=6步,shitf['e']=5以此类推
    }
    size_t limit = text_size - patt_size + 1;
    for(i = 0; i < limit; i += shift[text[i + patt_size]])
    {
        if(text[i] == *patt)
        {
            char *match_text = text + i + 1;
            size_t match_size = 1;
            do  //  输出所有匹配的位置
            {
                if(match_size == patt_size)
                {
                    cout << "the NO. is " << i << endl;
                }
            }
            while((*match_text++) == patt[match_size++]);
        }
    }
    cout << endl;
}

int main(void)
{
    char text[100] = "substring searching algorithm search";
    char patt[10] = "search";
    SUNDAY(text, patt);
    return 0;
}
```

编辑距离

编辑距离，又称Levenshtein距离（也叫做Edit Distance），是指两个字串之间，由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。

```
#include <iostream>
#include <cstring>
using namespace std;
typedef long long LL;
const int N = 1e3 + 5;
int T, cas = 0;
int n, m;
int dp[N][N];
char s[N], t[N];
int main() {
  while (scanf("%s%s", s, t) != EOF) {
    int n = (int)strlen(s), m = (int)strlen(t);
    for (int i = 0; i <= n; i++) {
       dp[i][0] = i;
    }
    for (int i = 0; i <= m; i++) {
      dp[0][i] = i;
    }
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= m; j++) {
        dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + 1;
        dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + (s[i - 1] != t[j - 1]));
      }
    }
    printf("%d\n", dp[n][m]);
  }
  return 0;
}
```