ACM 模板

武铎

目录

AC 自动机结构体版

```
#include<cstdio>
#include<string.h>
#include<cstring>
#include<queue>

using namespace std;

struct node
{
	int next[26];
	int val, fail;
}AC_node[1100];
int AC_total;
char s[100];
int dp[30][100][2048];

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

void AC_init()
{
	AC_total = 1;
	AC_node[0].val = 0;
	for(int i =0; i < 26; i ++)
	{
		AC_node[0].next[i] = 0;
	}
}

int AC_new()
{
	for(int i = 0; i < 26; i ++)
	{
		AC_node[AC_total].next[i] = 0;
	}
	AC_node[AC_total].val = 0;
	AC_node[AC_total].fail = 0;
	return AC_total ++;
}
```

```
void AC_insert(char *c, int num)
{
	int len = strlen(c);
	int pos = 0;
	for(int i = 0; i < len; i ++)
	{
		int index = c[i] - 'a';
		if(AC_node[pos].next[index] == 0)
		{
			AC_node[pos].next[index] = AC_new();
		}
		pos = AC_node[pos].next[index];
	}
	AC_node[pos].val |= 1<<num;
}

void AC_build()
{
	queue<int> q;
	while(!q.empty())
		q.pop();

	for(int i = 0; i < 26; i ++)
	{
		int temp = AC_node[0].next[i];
		if(temp)
		{
			AC_node[temp].fail = 0;
			q.push(temp);
		}
	}

	while(!q.empty())
	{
		int id = q.front();
		q.pop();
		AC_node[id].val |= AC_node[AC_node[id].fail].val;
		for(int i = 0; i < 26; i ++)
		{
			if(AC_node[id].next[i])
			{
				AC_node[AC_node[id].next[i]].fail = AC_node[AC_node[id].fail].next[i];
				q.push(AC_node[id].next[i]);
```

```
                        }
                        else
                                AC_node[id].next[i] = AC_node[AC_node[id].fail].next[i];
                }
        }
//      for(int i = 0; i < AC_total; i ++)
//      {
//              printf("%d %d\n", i, AC_node[i].val);
//      }

}

int solve(int n, int m, int k)
{
        for(int i = 0; i <= n; i ++)
                for(int j = 0;j <= AC_total; j ++)
                        for(int l = 0; l < 1 << m; l ++)
                        dp[i][j][l] = 0;
//因为数组的范围并不精确，因此使用 memset 反而会比较慢
//      memset(dp, 0, sizeof(dp));
        dp[0][0][0] = 1;
        for(int i = 0; i < n; i ++)
        {
                for(int j = 0; j < AC_total; j ++)
                {
                        for(int l = 0; l < 1<<(m); l ++)
                        {
                                if(dp[i][j][l])
                                {
                                        for(int h = 0; h < 26; h ++)
                                        {

                                                dp[i+1][AC_node[j].next[h]][AC_node[AC_node[j].next[h]].val|l]
+= dp[i][j][l];

dp[i+1][AC_node[j].next[h]][AC_node[AC_node[j].next[h]].val|l] %= 20090717;
                                        }
                                }
                        }
                }
        }
        int ans = 0, total, temp;
        for(int j = 0; j < 1<<(m); j ++)
        {
            total = 0;
            temp = j;
            while(temp)
            {
                total += temp&1;
                temp>>= 1;
            }
            if(total < k)
                continue;
            for(int i = 0; i < AC_total; i ++)
            {
                if(dp[n][i][j])
                {
                    ans+= dp[n][i][j];
                    ans %= 20090717;
                }
            }
        }

        return ans % 20090717;
}

int main ()
{
        int n, m, k;
        while(~scanf("%d %d %d", &n, &m, &k) && (n+m+k))
        {
            AC_init();
            for(int i = 0; i < m; i ++)
            {
                scanf("%s", s);
                AC_insert(s, i);
            }
            AC_build();
            int ans = solve(n, m, k);
            printf("%d\n", ans);
        }
}
```

AC 自动机基本查找

```
#include <cstdio>
```

```
#include <string.h>
#include <queue>
using namespace std;

int ch[500005][26], fail[500005], val[500005], total[500005];
int AC_total;
char s[1000006];

void AC_init()
{
        memset(ch, 0, sizeof(ch));
        memset(fail, 0, sizeof(fail));
        memset(val, 0, sizeof(val));
        memset(total, 0, sizeof(total));
        AC_total = 1;
}

void AC_insert()
{
        int len = strlen(s), id;
        int u = 0;
        for(int i = 0; i < len; i ++)
        {
                id = s[i] - 'a';
                if(ch[u][id] == 0)
                        ch[u][id] = AC_total ++;
                u = ch[u][id];
        }

        val[u] ++;
}

void AC_build()
{
        queue<int> q;
        while(!q.empty())
                q.pop();

        for(int i = 0; i < 26; i ++)
                if(ch[0][i])
                        q.push(ch[0][i]);

        while(!q.empty())
        {
```

```
        int u = q.front();
        q.pop();
        for(int i = 0; i < 26; i ++)
        {
            int temp = ch[u][i];
            if(temp != 0)
            {
                int v = fail[u];
                while(v && !ch[v][i])
                    v = fail[v];
                fail[temp] = ch[v][i];
                q.push(temp);
            }

        }
    }
}

int AC_find()
{
    int n = strlen(s);
    int j = 0, ans = 0;
    for(int i = 0; i < n; i ++)
    {
        int c = s[i] - 'a';
        while(j && !ch[j][c])
            j = fail[j];
        j = ch[j][c];
        int temp = j;
        while(temp && val[temp] != -1)
        {
            ans += val[temp];
            val[temp] = -1;
            temp = fail[temp];
        }
    }
    return ans;
}

int main ()
{
    int T, n;
    scanf("%d", &T);
    while(T --)
```

```
	{
		AC_init();
		scanf("%d", &n);
		while(n --)
		{
			scanf("%s", s);
			AC_insert();
		}

		scanf("%s", s);
		AC_build();
		int ans = AC_find();

//		int ans = 0;
//		for(int i = 0; i < AC_total; i ++)
//		{
//			if(total[i])
//				ans += val[i];
//		}
		printf("%d\n", ans);

	}
	return 0;
}
```

Floyd

```
#include<cstdio>

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

int w[1000][1000];

int main ()
{
	int T;
	scanf("%d", &T);
	int N, M, W, a, b, c;
	while(T --)
```

```
        {
                scanf("%d%d%d", &N, &M, &W);
                for(int i = 0; i <= N; i ++)
                        for(int j = 0; j <= N; j ++)
                                if(i == j)
                                        w[i][j] = 0;
                                else
                                        w[i][j] = 0x3fffffff;
                for(int i = 0; i < M; i ++)
                {
                        scanf("%d%d%d", &a, &b, &c);
                        if(w[a][b] > c)
                                w[a][b] = w[b][a] = c;
                }
                for(int i = 0; i < W; i ++)
                {
                        scanf("%d%d%d", &a, &b, &c);
                        if(w[a][b] > -c)
                                w[a][b] = -c;
                }
                int flag = 0;
                for(int k = 1; k <= N && !flag; k ++)
                {
                        for(int i = 1; i <= N && !flag; i ++)
                        {
                                for(int j = 1; j <= N && !flag; j ++)
                                {
                                        int t=w[i][k]+w[k][j];
                                        if(w[i][j]>t)w[i][j]=t;
                                        //用下面的方式就会 t。。。。
//                                         w[i][j] = min(w[i][j], w[i][k] + w[k][j]);
                                }
                                if(w[i][i] < 0)
                                        flag = 1;
                        }
                }

                if(flag)
                        printf("YES\n");
                else
                        printf("NO\n");
        }
        return 0;
}
```

Hash

```
#include<cstdio>

struct Node
{
	int num[6];
	int next;
}node[100005];
int hashtable[1000007], cur;

void init_hash();
int get_hash(int num[]);
int search_hash(int num[]);
void insert_hash(int num[], int h);
int cmp(int num1[], int num2[]);

int main ()
{
	int n, num0[6];
	while(~scanf("%d", &n))
	{
		cur = 0;
		init_hash();
		int temp, twins = 0;
		for(int i = 0; i < n;   i ++)
		{
			for(int j = 0; j < 6; j ++)
			{
				scanf("%d", &temp);
				num0[j] = temp;
			}
			if(twins)
				continue;
			twins = search_hash(num0);
		}
		if(twins)
			printf("Twin snowflakes found.\n");
		else
			printf("No two snowflakes are alike.\n");
	}
```

```
    return 0;
}

void init_hash()
{
    for(int i = 0; i < 1000007; i ++)
        hashtable[i] = -1;
    for(int i = 0; i < 100005; i ++)
        node[i].next = -1;
}

int get_hash(int num[6])
{
    int total = 0;
    for(int i = 0; i < 6; i ++)
        total = (total + num[i]) % 1000007;
    return total;
}

int search_hash(int num0[6])
{
    int h = get_hash(num0);
    if(hashtable[h] != -1)
    {
        int t = hashtable[h];
        while(t != -1)
        {
            if(cmp(node[t].num, num0))
                return 1;
            t = node[t].next;
        }
    }
    insert_hash(num0, h);
    return 0;
}

void insert_hash(int num0[6], int h)
{
    node[cur].next = hashtable[h];
    hashtable[h] = cur;
    for(int i = 0; i < 6; i ++)
        node[cur].num[i] = num0[i];
    cur ++;
}
```

```
int cmp(int num1[6], int num2[6])
{
        int ans = 1;
        for(int i = 0; i < 6; i ++)
        {
                ans = 1;
                for(int j = 0; j < 6; j ++)
                        if(num1[j] != num2[(j + i) % 6])
                        ans = 0;
                if(ans)
                        return 1;
        }
        for(int i = 0; i < 6; i ++)
        {
                ans = 1;
                for(int j = 0; j < 6; j ++)
                        if(num1[j] != num2[(6 - j + i) % 6])
                        ans = 0;
                if(ans)
                        return 1;
        }
        return 0;
}
```

IASP

```
#include <cstdio>
#include <string.h>
#include <queue>

using namespace std;

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

const int inf = 0x3fffffff;
const int ver = 500, edg = 30000;
struct Iasp
{
```

```
int top;
int head[ver], d[ver], gap[edg], pre[edg];
struct Edge
{
	int v, next;
	int c, f;
}edges[edg];

void init()
{
	memset(d, -1, sizeof(d));
	memset(gap, 0, sizeof(gap));
	memset(head, -1, sizeof(head));
	top = 0;
}

void add_edge(int u, int v, int c)
{
	edges[top].v = v;
	edges[top].c = c;
	edges[top].f = 0;
	edges[top].next = head[u];
	head[u] = top ++;
}

//每次加边的时候都要加原边和回边两条边。
void add(int u, int v, int c)
{
	add_edge(u, v, c);
	add_edge(v, u, 0);
}

//为 d 数组赋值，求出每个点所在的层次。
//汇点处于 0 层
void set_d(int t)
{
	queue<int> q;
	d[t] = 0;
	q.push(t);
	while(!q.empty())
	{
		int v = q.front();
		q.pop();
		gap[d[v]] ++;
```

```
            for(int i = head[v]; i != -1; i = edges[i].next)
            {
                int u = edges[i].v;
                if(d[u] == -1)
                {
                    d[u] = d[v] + 1;
                    q.push(u);
                }
            }
        }
    }

    //求此图的最大流
    int sap(int s, int t)
    {
        set_d(t);
        int ans = 0, u = s;
        int flow = inf;

        while(d[s] <= t)
        {
            int i;
            for(i = head[u]; i != -1; i = edges[i].next)
            {
                int v = edges[i].v;
                if(edges[i].c > edges[i].f && d[u] == d[v] + 1)
                {
                    u = v;
                    pre[v] = i;
                    flow = min(flow, edges[i].c - edges[i].f);
                    if(u == t)
                    {
                        while(u != s)
                        {
                            int j = pre[u];
                            edges[j].f += flow;
                            edges[j^1].f -= flow;
                            u = edges[j ^ 1].v;
                        }
                        ans += flow;
//                        printf("%d\n", flow);
                        flow = inf;
                    }
                    break;
                }
            }
            if(i == -1)
            {
                if(--gap[d[u]] == 0)
                    break;

                int dmin = t;
                for(int j = head[u];j != -1; j = edges[j].next)
                {
                    if(edges[j].c > edges[j].f)
                        dmin = min(dmin, d[edges[j].v]);
                }
                d[u] = dmin + 1;
                gap[d[u]] ++;
                if(u != s)
                    u = edges[pre[u] ^ 1].v;
            }
        }
        return ans;
    }

}Sap;

int main ()
{
    int n, f, d;
    while(~scanf("%d %d %d", &n, &f, &d))
    {
        int s = 0;
        int t = n * 2 + f + d + 1;
        int num = t + 1;
        Sap.init();

        //先建立原点与食物的边，容量为 1
        for(int i = 1; i <= f; i ++)
        {
            Sap.add(s, i, 1);
        }
        int temp, food, drink;
        for(int i = 1; i <= n; i ++)
        {
            scanf("%d %d", &food, &drink);
            //建立食物与左牛的边，容量为 1；
            for(int j = 1; j <= food; j ++)
            {
                scanf("%d", &temp);
                Sap.add(temp, f + i * 2 - 1, 1);
            }

            //建立左牛与优牛的边，容量为 1；
            Sap.add(f + i * 2 - 1, f + i * 2, 1);

            //建立右牛与饮料的边，容量为 1；
            for(int j = 1; j <= drink; j ++)
            {
                scanf("%d", &temp);
                Sap.add(f+i*2, temp + f+ n*2, 1);
            }
        }

        //建立饮料与汇点的边，容量为 1；
        for(int i = 1; i <= d; i ++)
        {
            Sap.add(2*n+f+i, t, 1);
        }

        int ans = Sap.sap(s, t);
        printf("%d\n", ans);
    }
    return 0;
}
```

Manachar

```
#include<cstdio>
#include<string.h>
#include<algorithm>

using namespace std;

const int maxn = 110005;
char str[maxn], str1[maxn * 2];
int dp[maxn * 2], n, maxx = 0;

void Manacher()
```

```
{
        memset(dp, 0, sizeof(dp));
        int mx = 0, id;
        for(int i = 1; i < n; i ++)
        {
                if(mx > i)
                        dp[i] = min(dp[2 * id - i], mx - i);
                else
                        dp[i] = 1;
                for(; str1[i - dp[i]] == str1[i + dp[i]]; dp[i] ++);
                maxx = max(maxx, dp[i]);
                if(i + dp[i] > mx)
                {
                        mx = i + dp[i];
                        id = i;
                }
        }
}

void pre()                                                                    //处理 str1；
{
        int i = 0, k = 1, t = 0;
        str1[0] = '$';
        while(str[i] != '\0')
        {
                str1[k ++] = t? str[i ++] : '#';
                t ^= 1;
        }
        str1[k ++] = '#';
        str1[k] = '\0';
        n = k;                                  //n 是记录 str1  的长度，但是要记住 strlen 的意思
                                                  //也可以每次用 n 的时候都赋值为  n = 2 * strlen(str);
}

int main ()
{
        while(~scanf("%s", str))
        {
                maxx = 0;
                pre();
                Manacher();
                printf("%d\n", maxx - 1);
        }
}
```

RMQ（）

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

int max(int x, int y)
{
	return x>y? x: y;
}
int min(int x, int y)
{
	return x<y? x: y;
}

int a[50005];
int st_min[50005][20], st_max[50006][20];

void initst(int n)
{
	for(int i = 0; i < n; i ++)
	{
		st_min[i][0] = st_max[i][0] = a[i];
	}

	for(int j = 1; (1<<j) < n; j ++)
	{
		for(int i = 0; i + (1<<j) <= n; i ++)
		{
			st_min[i][j] = min(st_min[i][j - 1], st_min[i + (1<<(j-1))][j-1]);
			st_max[i][j] = max(st_max[i][j - 1], st_max[i+(1<<(j-1))][j-1]);
		}
	}
}

int queryst_max(int l, int r)
{
	int k = (int)(log(r-l+1.0)/log(2.0));
	return max(st_max[l][k], st_max[r-(1<<k)+1][k]);
}

int queryst_min(int l, int r)
```

```
{
	int k = (int)(log(r-l+1.0)/log(2.0));
	return min(st_min[l][k], st_min[r-(1<<k)+1][k]);
}

int main ()
{
	int n, m, l, r;
	scanf("%d %d", &n, &m);
	for(int i = 0; i < n; i ++)
	{
		scanf("%d", a+i);
	}
	initst(n);
	while(m --)
	{
		scanf("%d %d", &l, &r);
		l --;
		r --;
		int x=queryst_max(l, r);
		int y=queryst_min(l, r);
		printf("%d\n", x-y);
	}
	return 0;
}
```

Sap

```
#include<cstdio>
#include<string.h>
#include<queue>
using namespace std;

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

const int inf = 0x3fffffff;
const int ver = 1000005, edg = 1000005;
int top;
int head[ver], d[ver], gap[edg], pre[edg];
```

```
struct Edge
{
        int v, next;
        int c, f;
}edges[edg];

struct Iasp
{


        void init()
        {
                memset(d, -1, sizeof(d));
                memset(gap, 0, sizeof(gap));
                memset(head, -1, sizeof(head));
                top = 0;
        }

        void add_edge(int u, int v, int c)
        {
                edges[top].v = v;
                edges[top].c = c;
                edges[top].f = 0;
                edges[top].next = head[u];
                head[u] = top ++;
        }

        void add(int u, int v, int c)
        {
                add_edge(u, v, c);
                add_edge(v, u, 0);
        }

        void set_d(int t)
        {
                queue<int> q;
                d[t] = 0;
                q.push(t);
                while(!q.empty())
                {
                        int v = q.front();
                        q.pop();
                        gap[d[v]] ++;
                        for(int i = head[v]; i != -1; i = edges[i].next)
        {
            int u = edges[i].v;
            if(d[u] == -1)
            {
                d[u] = d[v] + 1;
                q.push(u);
            }
        }
    }
}

int sap(int s, int t)
{
    set_d(t);
    int ans = 0, u = s;
    int flow = inf;
    while(d[s] <= top)
    {
        int i;
        for(i = head[u]; i != -1; i = edges[i].next)
        {
            int v = edges[i].v;
            if(edges[i].c > edges[i].f && d[u] == d[v] + 1)
            {
                u = v;
                pre[v] = i;
                flow = min(flow, edges[i].c - edges[i].f);
                if(u == t)
                {
                    while(u != s)
                    {
                        int j = pre[u];
                        edges[j].f += flow;
                        edges[j ^ 1].f -= flow;
                        u = edges[j ^ 1].v;
                    }
                    ans += flow;
//                        printf("%d ", flow);
                    flow = inf;
                }
                break;
            }
        }
        if(i == -1)
```

```
                {
                    if(--gap[d[u]] == 0)
                        break;

                    int dmin = t;
                    for(int j = head[u]; j != -1; j = edges[j].next)
                    {
                        if(edges[j].c > edges[j].f)
                            dmin = min(dmin, d[edges[j].v]);
                    }
                    d[u] = dmin + 1;
                    gap[d[u]] ++;
                    if(u != s)
                        u = edges[pre[u] ^ 1].v;
                }
            }
            return ans;
        }
}Sap;

int main ()
{
        int T, n, m, x, y, a, c, b, s, t;
        scanf("%d", &T);
        while(T --)
        {
            Sap.init();
            scanf("%d %d", &n, &m);
            int sx = 1000005, tx = -1000005;
            for(int i = 0; i < n; i ++)
            {
                scanf("%d %d", &x, &y);
                if(sx > x)
                {
                    s = i + 1;
                    sx = x;
                }
                if(tx < x)
                {
                    t = i + 1;
                    tx = x;
                }
            }
//              printf("%d %d\n", s, t);
```

```
        for(int i = 0; i < m; i ++)
        {
            scanf("%d %d %d", &a, &b, &c);
            Sap.add(a, b, c);
            Sap.add(b, a, c);
        }
        int ans = Sap.sap(s, t);
        printf("%d\n", ans);
    }
    return 0;
}
```

Spfa

```
int n;   //表示 n 个点，从 1 到 n 标号

int s,t;   //s 为源点，t 为终点

int d[N];   //d[i]表示源点 s 到点 i 的最短路

int p[N];   //记录路径（或者说记录前驱）

queue <int> q;   //一个队列，用 STL 实现，当然可有手打队列，无所谓

bool vis[N];     //vis[i]=1 表示点 i 在队列中  vis[i]=0 表示不在队列中

//bfs 求最短路
int spfa_bfs(int s)
{
    queue <int> q;
    memset(d,0x3f,sizeof(d));
    d[s]=0;
    memset(c,0,sizeof(c));
    memset(vis,0,sizeof(vis));

    q.push(s);   vis[s]=1; c[s]=1;
    //顶点入队 vis 要做标记，另外要统计顶点的入队次数
    int OK=1;
    while(!q.empty())
    {
        int x;
        x=q.front(); q.pop();    vis[x]=0;
        //队头元素出队，并且消除标记
        for(int k=f[x]; k!=0; k=nnext[k]) //遍历顶点 x 的邻接表
        {
            int y=v[k];
            if( d[x]+w[k] < d[y])
            {
                d[y]=d[x]+w[k];    //松弛
                if(!vis[y])    //顶点 y 不在队内
                {
                    vis[y]=1;        //标记
                    c[y]++;            //统计次数
                    q.push(y);      //入队
                    if(c[y]>NN)    //超过入队次数上限，说明有负环
                        return OK=0;
                }
            }
        }
    }

    return OK;

}

//dfs  判断负环
int spfa_dfs(int u)
{
    vis[u]=1;
    for(int k=f[u]; k!=0; k=e[k].next)
    {
        int v=e[k].v,w=e[k].w;
        if( d[u]+w < d[v] )
        {
            d[v]=d[u]+w;
            if(!vis[v])
            {
                if(spfa_dfs(v))
                    return 1;
            }
            else
                return 1;
        }
    }
```

```
	vis[u]=0;
	return 0;
}
```

Spfa 检测是否有正环

```
#include<cstdio>
#include<string.h>
#include<queue>

using namespace std;

int e;
int head[10500], vis[10005], cnt[10050];
double dis[10050];
//dis 可能是小数

struct node
{
	int v, next;
	double r, c;
}edge[1500];

//采用邻接表的方式存储图
void add(int a, int b, double r, double c)
{
	edge[e].v = b;
	edge[e].r = r;
	edge[e].c = c;
	edge[e].next = head[a];
	head[a] = e ++;
}

void SPFA_init()
{
	e = 0;
	memset(vis, 0, sizeof(vis));
	memset(dis, 0, sizeof(dis));
	memset(cnt, 0, sizeof(cnt));
	memset(head, -1, sizeof(head));
}
```

```
int SPFA(int source, double much, int N)
{
	queue<int> q;
	q.push(source);
	vis[source] = 1;
	dis[source] = much;
	cnt[source] ++;
	while(!q.empty())
	{
		int first = q.front();
		q.pop();
		vis[first] = 0;
		for(int i = head[first]; i != -1; i = edge[i].next)
		{
			int v = edge[i].v;
			double tempdis = (dis[first] - edge[i].c) * edge[i].r;
			if(dis[v] < tempdis)
			{
				dis[v] = tempdis;
				if(!vis[v])
				{
					q.push(v);
					vis[v] = 1;
				}
				cnt[v] ++;
				if(cnt[v] > N + 1)
					return -1;
			}
		}
	}
	return 1;
}

int main ()
{
	int N, M, a, b, source;
	double much, rab, rba, cba, cab;
	scanf("%d%d%d%lf", &N, &M, &source, &much);
	SPFA_init();
	for(int i = 0; i < M; i ++)
	{
		scanf("%d%d%lf%lf%lf%lf", &a, &b, &rab, &cab, &rba, &cba);
		add(a, b, rab, cab);
		add(b, a, rba, cba);
	}
	int ans = SPFA(source, much, N);
	if(ans == -1)
		printf("YES\n");
	else
		printf("NO\n");
	return 0;
}
```

Splay

```
#define N 500000
#define lc (tr[id].c[0])
#define rc (tr[id].c[1])
#define KEY (tr[tr[root].c[1]].c[0])//根的右孩子的左孩子

struct Tr {
	int fa, sum, val, c[2], lz;
}tr[N];

int newtr(int k, int f) {//新建立一个节点
	tr[tot].sum = 1, tr[tot].val =   k;
	tr[tot].c[0] = tr[tot].c[1] = -1;
	tr[tot].lz = 0;
	tr[tot].fa = f;
	return tot++;
}

void Push(int id) {
	int lsum, rsum;
	lsum = (lc == -1)?0:tr[lc].sum;
	rsum = (rc == -1)?0:tr[rc].sum;
	tr[id].sum = lsum+rsum+1;
}


void lazy(int id) {//flip 专属懒操作
	if (tr[id].lz) {
		swap(lc, rc);
		tr[lc].lz ^= 1, tr[rc].lz ^= 1;
		tr[id].lz = 0;
	}
```

```
}

int build(int l, int r, int f) {//建树
        if (r < l) return-1;
        int mid = l+r>>1;
        int ro = newtr(data[mid], f);
        tr[ro].c[0] = build(l, mid-1, ro);
        tr[ro].c[1] = build(mid+1, r, ro);
        Push(ro);
        return ro;
}

void Rotate(int x, int k) {//k=1 右旋,k=0 左旋
        if (tr[x].fa == -1) return;
        int fa = tr[x].fa, w;
        lazy(fa), lazy(x);
        tr[fa].c[!k] = tr[x].c[k];
        if (tr[x].c[k] != -1) tr[tr[x].c[k]].fa = fa;
        tr[x].fa = tr[fa].fa, tr[x].c[k] = fa;
        if (tr[fa].fa != -1) {
                w = tr[tr[fa].fa].c[1]==fa;
                tr[tr[fa].fa].c[w] = x;
        }
        tr[fa].fa = x;
        Push(fa);
        Push(x);
}

void Splay(int x, int goal) {//将 x 节点转到 goal 的儿子上
        if (x == -1) return;
        lazy(x);
        while (tr[x].fa != goal) {
                int y = tr[x].fa;
                lazy(tr[y].fa), lazy(y), lazy(x);
                bool w = x==tr[y].c[1];
                if (tr[y].fa != goal && w == (y==tr[tr[y].fa].c[1]))
                        Rotate(y, !w);
                Rotate(x, !w);
        }
        if (goal == -1) root = x;
        Push(x);
}

int find(int k) {//找到第 k 个节点的 ID
```

```
    int id = root;
    while (id != -1) {
        lazy(id);
        int lsum = (lc==-1)?0:tr[lc].sum;
        if (lsum >= k) {
            id = lc;
        }
        else if (lsum+1 == k) break;
        else {
            k = k-lsum-1;
            id = rc;
        }
    }
    return id;
}

int Index(int l, int r) {//将区间(l+1, r-1)化成一颗子树
    Splay(find(l), -1);
    Splay(find(r),root);
}

int Getnext(int id) {//寻找后继节点
    lazy(id);
    int p = tr[id].c[1];
    if (p == -1) return id;
    lazy(p);
    while (tr[p].c[0] != -1) {
        p = tr[p].c[0];
        lazy(p);
    }
    return p;
}

int del(int l, int r) {//将【l,r】切掉,返回切掉子树的根节点
    Index(l-1, r+1);
    int ro = KEY;
    tr[KEY].fa = -1;
    KEY = -1;
    Push(tr[root].c[1]);
    Push(root);
    return ro;
}

void cut(int k, int ro) {//将子树 ro 接到第 k 个树之后
```

```
	Index(k, k+1);
	KEY = ro;
	tr[ro].fa = tr[root].c[1];
	Push(tr[root].c[1]);
	Push(root);
}

void filp(int l, int r) {//对区间【l,r】反转
	Index(l-1, r+1);
	lazy(root), lazy(tr[root].c[1]);
	tr[KEY].lz ^= 1;
}

void Add(int l, int r, int d) {//区间【l,r】的数加上 d
	Index(l-1, r+1);
	tr[KEY].add += d;
	tr[KEY].mi += d;
	tr[KEY].val += d;
	Push(tr[root].c[1]);
	Push(root);
}

void Delete(int x) {//删除第 x 个数
	Index(x-1, x+1);
	tr[KEY].fa = -1;
	tr[tr[root].c[1]].c[0] = -1;
	Push(tr[root].c[1]);
	Push(root);
}

void Insert(int l, int x) {//在 l 之后插入 x
	Index(l, l+1);
	int ro;
	ro = newtr(x, tr[root].c[1]);
	KEY = ro;
	Push(tr[root].c[1]);
	Push(root);
}

void Revolve(int l, int r, int d) {//【l, r】整体右移 d 位
	int ro = del(r+1-d, r);
	cut(l-1, ro);
}
```

Treep

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

struct treap
{
	treap *left, *right;
	int val, pri;
	int size;
	treap (int vv)
	{
		left = right = NULL;
		pri = rand();
		val = vv;
	}
}*root;

void print(treap *p)
{
	if(!p)
		return;
	print(p->left);
	printf("%d ", p->val);
	print(p->right);
}

int lsize(treap *p)
{
	return p->left ? p->left->size : 0;
}

int rsize(treap *p)
{
	return p->right ? p->right->size : 0;
}

//传参数的时候一定记得&
//左旋。
void l_rotate(treap *&p)
{
	treap *temp = p->right;
	p->right = temp->left;
```

```
        temp->left = p;
        temp->size = p->size;
        p->size = lsize(p) + rsize(p) + 1;
        p = temp;
}

void r_rotate(treap *&p)
{
        treap *temp = p->left;
        p->left = temp->right;
        temp->right = p;
        temp->size = p->size;
        p->size = lsize(p) + rsize(p) + 1;
        p = temp;
}

void insert(treap *&p, int val)
{
        if(!p)
        {
                p = new treap(val);
                p->size = 1;
        }
        else if(val <= p->val)
        {
                p->size ++;
                insert(p->left, val);
                if(p->left->pri < p->pri)
                        r_rotate(p);
        }
        else
        {
                p->size ++;
                insert(p->right, val);
                if(p->right->pri < p->pri)
                        l_rotate(p);
        }
}

int find(int k, treap *p)
{
        int temp = lsize(p);
        if(k == temp + 1)
                return p->val;
```

```
	else if(k <= temp)
		return find(k, p->left);
	else return find(k - temp - 1, p->right);
}



int main ()
{
	int m, n, num[30005];
	scanf("%d%d", &m, &n);
	for(int i = 1; i <= m; i ++)
		scanf("%d", &num[i]);
	int temp = 1, len, ans;
	root = NULL;

	for(int i = 1; i <= n; i ++)
	{
		scanf("%d", &len);
		for(; temp <= len; temp ++)
		{
			insert(root, num[temp]);
		}
		ans = find(i, root);
		printf("%d\n", ans);
	}
	return 0;
}
```

Trie

```
#include<cstdio>
#include<string.h>

struct node{
	node * next[3];
	int val;
	void clean()
	{
		val = 0;
		memset(next, 0, sizeof(next));
	}
```

```
}*root;

void release(node *p)
{
        for(int i = 0; i < 3; i ++)
        {
                if(p->next[i] != NULL)
                        release(p->next[i]);
        }
        delete p;
}

void insert(char *s)
{
        node *p = root;
        int len = strlen(s);
        for(int i =0 ;i < len ; i ++)
        {
                int t = s[i] - 'a';
                if(p->next[t] == 0)
                {
                         p->next[t] = new node;
                         p->next[t]->clean();
                }
                p = p->next[t];

        }
        p->val = 1;
}

//flag = 0: no change
bool search(char *s, int flag, node *p)
{
        if(strlen(s) == 0)
                return flag;
        int t= s[0] - 'a';
        if(flag)
        {
                if(p->next[t] == NULL)
                        return 0;
                return search(s + 1, 1, p->next[t]);
        }
        else
        {
            if(p->next[t] != NULL)
                if(search(s + 1, 0, p->next[t]))
                return 1;
            t = (t + 1) % 3;
            if(p->next[t] != NULL)
                if(search(s + 1, 1, p->next[t]))
                return 1;
            t = (t + 1) %3;
            if(p->next[t] != NULL)
                if(search(s + 1, 1, p->next[t]))
                return 1;
            return 0;
        }
}

char c[1000000];

int main ()
{
    int n ,m;

    while(~scanf("%d%d", &n, &m))
    {
        root = new node;
        root ->clean();
        for(int i = 0; i < n; i ++)
        {
            scanf("%s", c);
            insert(c);
        }

        for(int i = 0; i < m; i ++)
        {
            scanf("%s", c);
            if(search(c, 0, root))
                printf("YES\n");
            else
                printf("NO\n");
        }
        release(root);
    }
}
```

```
/*
#include<cstdio>
#include<string>
#include<iostream>
#include<map>

using namespace std;

map<string, int> ma;

int main ()
{
	int n, m;
	string s;
	while(~scanf("%d%d", &n, &m))
	{
		for(int i = 0; i < n; i ++)
		{
			cin>> s;
			ma[s] = 1;
		}
		char temp;
		for(int i = 0; i < m; i ++)
		{
			cin>> s;
			int flag = 0;
			for(int j = 0; j < s.length();j ++)
			{
				temp = s[j];
				if(temp != 'a')
				{
					s[j] = 'a';
					if(ma[s] == 1)
					{
						flag = 1;
						break;
					}
				}
				if(temp != 'b')
				{
					s[j] = 'b';
					if(ma[s] == 1)
					{
						flag = 1;
```

```
                        break;
                    }
                }
                if(temp != 'c')
                {
                    s[j] = 'c';
                    if(ma[s] == 1)
                    {
                        flag = 1;
                        break;
                    }
                }
                s[j] = temp;
            }
            if(flag)
                printf("YES\n");
            else
                printf("NO\n");
        }
    }
}
*/
```

递归实现排列 n 各元素

```
#include<cstdio>

using namespace std;

void combine_increase(const int *numbers, int *result, const int arrysize,const int elements,
                       int current = 0, int start = 0)
{
    for(int i = start; i <= arrysize - elements + current; i ++)
    {
        result[current] = i;
        if(elements - current - 1)
        {
            combine_increase(numbers, result, arrysize, elements, current + 1, i + 1);
        }
        else
        {
            for(int j = current; j >= 0; j --)
				{
					printf("%d\t", numbers[result[current - j]]);
				}
				printf("\n");
			}
		}
}

int main()
{
	int numbers[] = {0, 1, 2, 3, 4, 5};
	int elements = 3;
	int *result = new int[3];
	combine_increase(numbers, result, 6, elements);
	return 0;
}
```

堆

```
#include<cstdio>
#include<algorithm>
using namespace std;

int a[10000], b[2000], c[2000];
int main ()
{
	int T, n, m;
	scanf("%d", &T);
	while(T --)
	{
		scanf("%d%d", &n, &m);
		int temp = m;
		for(int i = 0; i < m; i ++)
			scanf("%d", &c[i]);
		for(int i = 1; i < n; i ++)
		{
			sort(c, c + m);
			for(int i = 0; i < m; i ++)
				scanf("%d", &b[i]);
			for(int i = 0; i < m; i ++)
				a[i] = b[0] + c[i];
			make_heap(a, a + m);
```

```
            for(int i = 1; i < m; i ++)
            {
                for(int j = 0; j < m; j ++)
                {
                    if(b[i] + c[j] > a[0])
                        break;
                    pop_heap(a, a + m);
                    a[m - 1] = b[i] + c[j];
                    push_heap(a, a + m);
                }
            }
            for(int i = 0; i < m; i ++)
                c[i] = a[i];
        }
        sort(c, c + m);
        for(int i = 0; i < m; i ++)
        {
            if(i)
                printf(" %d", c[i]);
            else
                printf("%d", c[i]);
        }
        printf("\n");
    }
    return 0;
}
```

二层魔方

```
#include<cstdio>

int B[6][24]={ {6,1,12,3,5,11,16,7,8,9,4,10,18,13,14,15,20,17,22,19,0,21,2,23}, //ok
               {20,1,22,3,10,4,0,7,8,9,11,5,2,13,14,15,6,17,12,19,16,21,18,23}, //ok
               {1,3,0,2,23,22,4,5,6,7,10,11,12,13,14,15,16,17,18,19,20,21,9,8}, //ok
               {2,0,3,1,6,7,8,9,23,22,10,11,12,13,14,15,16,17,18,19,20,21,5,4}, //ok
               {0,1,8,14,4,3,7,13,17,9,10,2,6,12,16,15,5,11,18,19,20,21,22,23}, //ok
               {0,1,11,5,4,16,12,6,2,9,10,17,13,7,3,15,14,8,18,19,20,21,22,23}   //ok
             };

int ans;

int one(int *x, int a, int b, int c, int d)
```

```
{
        if(x[a] == x[b] && x[b] == x[c] && x[c] == x[d])        return 1;
        return 0;
}
int now(int *x)
{
        int ret = 0;
        if(one(x, 0,1,2,3))          ret ++;
        if(one(x, 4,5,10,11))      ret ++;
        if(one(x, 6,7,12,13))      ret ++;
        if(one(x, 8,9,14,15))      ret ++;
        if(one(x, 16,17,18,19)) ret ++;
        if(one(x, 20,21,22,23)) ret ++;

        return ret;
}

int max(int x, int y)
{
        return x> y? x:y;
}

void dfs(int *x, int n)
{
        ans = max(ans, now(x));
        if(n == 0)
                return ;
        int temp[24];
        for(int i = 0; i < 6; i ++)
        {
                for(int j = 0; j < 24; j ++)
                {
                        temp[j] = x[B[i][j]];
                }
                dfs(temp, n - 1);
        }
}

int main ()
{
        int n;
        while(~scanf("%d", &n))
        {
                int a[30];
```

```
		for(int i = 0; i < 24; i ++)
		{
			scanf("%d", &a[i]);
		}

		ans = 0;
		dfs(a, n);
		printf("%d\n", ans);
	}
	return 0;
}
```

二分图最大匹配

```
#include<cstdio>
#include<string.h>

int g[107][107], msp[1007][1007], msw[1007][1007], mpw[1005][1005], used[1005], linker[1004];
int uN, vN;

bool dfs(int u)
{
	for(int v=0; v<vN; v++){
		if(g[u][v]&&!used[v]){
			used[v]=true;
			if(linker[v]==-1||dfs(linker[v])){
				linker[v]=u;
				return true;
			}
		}
	}
	return false;
}

int hungary()
{
	int res=0, u;
	memset(linker, -1, sizeof(linker));
	for(u=0; u<uN; u++){
		memset(used, 0, sizeof(used));
		if(dfs(u))res++;
	}
	return res;
}

int main ()
{
	int T, ns, np, nw;
	scanf("%d", &T);
	while(T --)
	{
		int to;
		scanf("%d %d %d", &ns, &np, &nw);
		memset(msp, 0, sizeof(msp));
		memset(msw, 0, sizeof(msw));
		memset(mpw, 0, sizeof(mpw));
		for(int i = 1; i <= ns; i ++)
		{
			int temp;
			scanf("%d", &temp);
			while(temp --)
			{
				scanf("%d", &to);
				msp[i][to] = 1;
			}
		}
		for(int i = 1; i <= ns; i ++)
		{
			int temp;
			scanf("%d", &temp);
			while(temp --)
			{
				scanf("%d", &to);
				msw[i][to] = 1;
			}
		}

		for(int i = 1; i <= np; i ++)
		{
			int temp;
			scanf("%d", &temp);
			while(temp --)
			{
				scanf("%d", &to);
				mpw[i][to] = 1;
```

```
                }
        }


        memset(g, 0, sizeof(g));
        for(int i = 1; i <= np; i ++)
        {
                for(int j = 1; j <= ns; j ++)
                {
                        if(msp[j][i])
                        {
                                for(int k = 1; k <= nw; k ++)
                                {
                                        if(msw[j][k] && mpw[i][k])
                                        {
                                                g[i - 1][j - 1] = 1;
                                        }
                                }
                        }
                }
        }
        uN = np, vN = ns;
        printf("%d\n", hungary());
    }
    return 0;

}
```

割点割边

```
#include<cstdio>
#include<string.h>

int dfn[106], vis[105], low[105], head[106], flag[105];
int time, total, ans;

struct node
{
    int to, next;
}edge[10000006];

int min(int a, int b)
```

```
{
        return a>b? b: a;
}

void add(int a, int b)
{
        edge[total].to = b;
        edge[total].next = head[a];
        head[a] = total ++;
}

void dfs(int id)
{
        time ++;
        low[id] = dfn[id] = time;
        vis[id] = 1;
        int cnum = 0;

        for(int i = head[id]; i; i = edge[i].next)
        {
                int temp = edge[i].to;
                if(vis[temp])
                {
                        low[id] = min(low[id], dfn[temp]);
                }
                else
                {
                        cnum++;
                        dfs(temp);
                        low[id] = min(low[id], low[temp]);
                        if(id == 1 && cnum > 1)
                                flag[id] = 1;
                        if(id != 1 && low[temp] >= dfn[id])
                                flag[id] = 1;
                }
        }
}

int main ()
{
        int N, a, b;
        while(~scanf("%d", &N) && N)
        {
                ans = time = 0;
```

```
		total = 1;
		memset(vis, 0, sizeof(vis));
		memset(head, 0, sizeof(head));
		//head=0 表示没有该边。因此 edge 的下标必须从 1 开始
		memset(flag, 0, sizeof(flag));

		while(~scanf("%d", &a) && a)
		{
			while(~scanf("%d", &b))
			{
				add(a, b);
				add(b, a);
				if(getchar()=='\n')
					break;
			}
		}
		dfs(1);

		for(int i = 1; i <= N; i ++)
			ans += flag[i];
		printf("%d\n", ans);
	}
}
```

归并排序求逆序数

```
#include<cstdio>

int b[500005], a[500005];
long long ans;
void merge(int l, int r, int mid)
{
//	int mid = (l + r) >> 1;
	int last = mid + 1, temp = l;
	while(l <= mid && last <= r)
	{
		if(a[l] <= a[last])
			b[temp ++] = a[l ++];
		else
		{
			ans += mid - l + 1;
			b[temp ++] = a[last ++];
		}
	}
	while(l <= mid)
		b[temp ++] = a[l ++];
	while(last <= r)
		b[temp ++] = a[last ++];
}

void mergesort(int l, int r)
{
	if(l >= r)
		return ;
	int mid = (l + r) >> 1;
	mergesort(l, mid);
	mergesort(mid + 1, r);
	merge(l, r, mid);
	for(int i = l; i <= r; i ++)
		a[i] = b[i];
}

int main ()
{
	int n;
	while(~scanf("%d", &n) && n)
	{
		ans = 0;
		for(int i = 0; i < n; i ++)
			scanf("%d", &a[i]);
		mergesort(0,n - 1);
//		for(int i = 0; i < n; i ++)
//			printf("%d ", a[i]);
//		printf("\n");

		printf("%lld\n", ans);
	}
	return 0;
}
```

后缀数组

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

```
#include<string.h>
const int maxn = 100005;

int wa[maxn], wb[maxn], wsf[maxn], wv[maxn], sa[maxn];
int rank[maxn], height[maxn], s[maxn];
char str[maxn], str1[maxn];

int cmp(int *r, int a, int b, int k)
{
	return r[a] == r[b] && r[a + k] == r[b + k];
}

void get_sa(int *r, int *sa, int n, int m)
{
	int *x = wa, *y = wb, *t, i, j, p;
	for(i = 0; i < m; i ++) wsf[i] = 0;
	for(i = 0; i < n; i ++) wsf[x[i] = r[i]] ++;
	for(i = 1; i < m; i ++) wsf[i] += wsf[i - 1];
	for(i = n - 1; i >= 0; i --) sa[-- wsf[x[i]]] = i;

	p = 1, j = 1;
	for(; p < n; j *= 2, m = p)
	{
		for(p = 0, i = n - j; i < n; i ++) y[p ++] = sa[i] - j;
		for(i = 0; i < n; i ++) if(sa[i] >= j) y[p ++] = sa[i] - j;
		for(i = 0; i < n; i ++) wv[i] = x[y[i]];
		for(i = 0; i < m; i ++) wsf[i] = 0;
		for(i = 0; i < n; i ++) wsf[wv[i]] ++;
		for(i = 1; i < m; i ++) wsf[i] += wsf[i - 1];
		for(i = n - 1; i >= 0; i --) sa[--wsf[wv[i]]] = y[i];
		t = x;
		x = y;
		y = t;
		x[sa[0]] = 0;
		for(p = 1, i = 1; i < n; i ++)
			x[sa[i]] == cmp(y, sa[i - 1], sa[i], j)? p - 1: p ++;
	}
}

void getheight(int *r, int n)
{
	int i, j, k = 0;
	for(i = 1; i <= n; i++)
		rank[sa[i]] = i;
```

```
    for(i = 0; i < n; i ++)
    {
        if(k)
            k --;
        j = sa[rank[i] - 1];
        while(r[i + k] == r[j + k])
            k ++;
        height[rank[i]] = k;
    }
}


int main()
{
    int T, n;
    scanf("%d", &T);
    while(T --)
    {
        scanf("%d", &n);
        scanf("%s", str);
        strcpy(str1, str);
        strcat(str1, str1);
        for(int i = 0; i < n; i ++)
            str[i] = str1[n - 1 - i];
        strcat(str, str);
        n *= 2;
        for(int i = 0; i < n; i ++)
        {
            s[i] = str[i] - 'a';
        }
        s[n ++] = 28;

//          for(int i = 0; i < strlen(str1); i ++)
//              printf("%c", str1[i]);
//          printf("\n");
//          for(int i = 0; i < strlen(str); i ++)
//              printf("%c", str[i]);
//          printf("\n");

        get_sa(s, sa, n + 1, 30);
        getheight(s, n);
        for(int i = 0; i < n; i ++)
        {
            if(height[i] == n / 2)
				{
					ans = i;
					break;
				}
			}

		}
		return 0;
}
```

快速幂

```
#include<cstdio>

long long multi(long long a, long long b, long long mod)
{
	long long ret;
	ret = 1;
	while(b > 0)
	{
		if(b & 1)
			ret	= ret * a % mod;
		a = (a * a) % mod;
		b = b >> 1;
	}
	return ret;
}

int main ()
{
	long long a, b;
	while(~scanf("%lld%lld", &a, &b))
	{
		if(b == 1)
		{
			printf("1\n");
			continue;
		}
		long long ans = multi(2, b, 1000000007);
		ans --;
		ans = multi(ans , a, 1000000007);
		printf("%lld\n", ans);
	}
	return 0;
}
```

利用 kmp 的 next 数组求循环节

```
#include<cstdio>

char in[1000005];
int next[1000005];
int N;

void get(void)
{
	for(int i = 2; i <= N; i ++)
	{
		int j = next[i - 1];
		while(j && in[j] != in[i - 1])
			j = next[j];
		next[i] = in[i - 1] == in[j] ? j + 1: 0;
	}
}

void work(void)
{
	for(int i = 1; i <= N; i ++)
		if(i % (i - next[i]) == 0 && i / (i - next[i]) > 1)
			printf("%d %d\n", i, i / (i - next[i]));
}

int main ()
{
	int a = 1;
	while(scanf("%d", &N), N != 0)
	{
		scanf("%s", in);
		printf("Test case #%d\n", a ++);
		get();
		work();
		printf("\n");
	}
	return 0;
}
```

求素数个数

```
#include <cstdio>

long long f[340000], g[340000], n;

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

void init()
{
	long long i, j, m;
	for(m = 1; m * m <= n; m ++)
		f[m] = n / m - 1;
	for(i = 1; i <= m; i ++)
		g[i] = i - 1;
	for(i = 2; i <= m; i ++)
	{
		if(g[i] == g[i - 1])
			continue;
		for(j = 1; j <= min(m - 1, n / i / i); j ++)
		{
			if(i * j < m)
				f[j] -= f[i * j] - g[i - 1];
			else
				f[j] -= g[n / i / j] - g[i - 1];
		}
		for(j = m; j >= i * i; -- j)
			g[j] -= g[j / i] - g[i - 1];
	}
}

int main()
{
	while(~scanf("%lld", &n))
	{
		init();
		printf("%lld\n", f[1]);
	}
	return 0;
}
```

三维树状数组

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

using namespace std;

int N;
int cube[102][102][102];

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

int sum(int x, int y, int z)
{
	int ans = 0;
	for(int i = x; i > 0; i -= lowbit(i))
		for(int j = y; j > 0; j -= lowbit(j))
			for(int k = z; k > 0; k -= lowbit(k))
			ans += cube[i][j][k];

	return ans & 1;
}

void update(int x, int y, int z)
{
	for(int i = x; i <= N; i += lowbit(i))
		for(int j = y; j <= N; j += lowbit(j))
			for(int k = z; k <= N; k += lowbit(k))
			cube[i][j][k] ++;
}

int main ()
{
	int M;
	while(~scanf("%d%d", &N, &M))
```

```
    {
        memset(cube, 0, sizeof(cube));
        for(int i = 0; i < M; i ++)
        {
            int temp;
            scanf("%d", &temp);
            if(temp == 1)       //update
            {
                int x1, x2, y1, y2, z1, z2;
                scanf("%d %d %d %d %d %d", &x1, &y1, &z1, &x2, &y2, &z2);
                update(x1, y1, z1);
                update(x1, y1, z2+1);
                update(x1, y2+1, z1);
                update(x1, y2+1, z2+1);

                update(x2+1, y1, z1);
                update(x2+1, y1, z2+1);
                update(x2+1, y2+1, z1);
                update(x2+1, y2+1, z2+1);
            }

            else if(temp == 0)      //sum
            {
                int x, y, z;
                scanf("%d %d %d", &x, &y, &z);
                int ans = sum(x, y, z);
                printf("%d\n", ans);
            }
        }
    }
    return 0;
}
```

树链剖分

```
#include<cstdio>
#include<algorithm>
#include<string.h>
using namespace std;

const int N = 50015;
```

```
//树上节点的权值，以该节点为根的子树节点个数，节点所在重链的头，节点重链上的子节点
int num[N], siz[N], top[N], son[N];
//节点的深度，节点对应线段树上的位置下标，线段树上位置对应的节点下标，节点的父节点
int dep[N], tid[N], _rank[N], fa[N];
//建图所用
int head[N], to[N * 2], _next[N * 2], edge;
//线段树上每个节点所需维护的值，线段树上节点是否有更改操作
int sum[N * 4], col[N * 4];
//当前深度，树的总结点树（线段树的最右端点）
int tim, n;

void init()
{
    memset(head, -1, sizeof(head));
    memset(son, -1, sizeof(son));
    tim = 1;
    edge = 0;
}

void add_edge(int u, int v)
{
    to[edge] = v;
    _next[edge] = head[u];
    head[u] = edge ++;
    to[edge] = u;
    _next[edge] = head[v];
    head[v] = edge ++;
}

//当前结点，父节点，深度
void dfs1(int u, int f, int d)
{
    dep[u] = d;
    fa[u] = f;
    siz[u] = 1;
    for(int i = head[u]; i != -1; i = _next[i])
    {
        int v = to[i];
        if(v != f)
        {
            dfs1(v, u, d + 1);
            siz[u] += siz[v];
```

```
                if(son[u] == -1 || siz[v] > siz[son[u]])
                    son[u] = v;
            }
        }
}

//当前节点，所在重链
void dfs2(int u, int tp)
{
    top[u] = tp;
    tid[u] = tim;
    _rank[tim ++] = u;
    if(son[u] == -1)
        return ;
    dfs2(son[u], tp);
    for(int i = head[u]; i != -1; i = _next[i])
    {
        int v = to[i];
        if(v != son[u] && v != fa[u])
            dfs2(v, v);
    }
}

//由 r  t 节点的两个儿子节点更新 r  t
void push_up(int rt)
{
    sum[rt] = max(sum[rt << 1], sum[rt << 1 | 1]);
}

//rt  点的 lazy  操作
void push_down(int rt, int m)
{
    if(col[rt])
    {
        col[rt << 1] += col[rt];
        col[rt << 1 | 1] += col[rt];
        sum[rt << 1] += (m - (m >> 1)) * col[rt];
        sum[rt << 1 | 1] += (m >> 1) * col[rt];
        col[rt] = 0;
    }
}

//线段树建树
void build(int l, int r, int rt)
```

```
{
	col[rt] = 0;
	if(l == r)
	{
		sum[rt] = num[_rank[l]];
		return ;
	}
	int mid = (l + r) >> 1;
	build(l, mid, rt << 1);
	build(mid + 1, r, rt << 1 | 1);
	push_up(rt);
}

//线段树更新
void update(int l, int r, int v, int ll, int rr, int rt)
{
	if(l <= ll && r >= rr)
	{
		col[rt] += v;
		sum[rt] += v * (rr - ll + 1);
		return ;
	}
	push_down(rt, rr - ll + 1);
	int mid = (ll + rr ) >> 1;
	if(l <= mid)
		update(l, r, v, ll, mid, rt << 1);
	if(r > mid)
		update(l, r, v, mid + 1, rr, rt << 1 | 1);
	push_up(rt);
}

//线段树查询
int query(int l, int r, int rt, int val)
{
	if(l == r)
		return sum[rt];
	push_down(rt, r - l + 1);
	int mid = (l + r) >> 1;
	int ret = 0;
	if(val <= mid)
		ret = query(l, mid, rt << 1, val);
	if(val > mid)
		ret = query(mid + 1, r, rt << 1 | 1, val);
	push_up(rt);
	return ret;
}

//树链更新
void change(int x, int y, int val)
{
	while(top[x] != top[y])
	{
		if(dep[top[x]] < dep[top[y]])
			swap(x, y);
		update(tid[top[x]], tid[x], val, 1, n, 1);
		x = fa[top[x]];
	}
	if(dep[x] > dep[y])
		swap(x, y);
	update(tid[x], tid[y], val, 1, n, 1);
}

int main ()
{
	int a, b, c, m, q;
	while(~scanf("%d %d %d", &n, &m, &q))
	{
		init();
		memset(num, 0, sizeof(num));
		for(int i = 1; i<= n; i ++)
			scanf("%d", &num[i]);
		for(int i = 1; i <= m; i ++)
		{
			scanf("%d %d", &a, &b);
			add_edge(a, b);
		}
		dfs1(1, 0, 0);
		dfs2(1, 1);
		build(1, n, 1);

		char op[20];
		while(q --)
		{
			scanf("%s", op);
			if(op[0] == 'Q')
			{
				scanf("%d", &a);
				printf("%d\n", query(1, n, 1, tid[a]));
```

```
                        }
                        else
                        {
                                scanf("%d %d %d", &a, &b, &c);
                                if(op[0] == 'D')
                                        c = -c;
                                change(a, b, c);
                        }
                }
        }
}
```

双联通分量

```
#include<cstdio>
#include<string.h>

//此题利用 tarjan 求加多少条边可以得到双连通分量

struct node
{
        int to, next;
}edge[3000];
int dfn[1005], vis[1005], low[1004], head[1005], in[1005];
int time, n, edge_total;

void addEdge(int a, int b)
{
        edge[edge_total].to = a;
        edge[edge_total].next = head[b];
        head[b] = edge_total ++;

        edge[edge_total].to = b;
        edge[edge_total].next = head[a];
        head[a] = edge_total ++;
}

void tarjan_init()
{
        memset(vis, 0, sizeof(vis));
        memset(dfn, 0, sizeof(dfn));
        memset(in, 0, sizeof(in));
```

```
	time = 1;
}

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

void dfs(int id, int fa)
{
//	printf("%d %d\n", id, fa);
	dfn[id] = low[id] = time ++;
	vis[id] = 1;
	for(int i = head[id]; i != -1; i = edge[i].next)
	{
		int t = edge[i].to;
		if(t == fa)
			continue;
		//因为建边的时候建的是双向边，因此必须检测这条边是否指向他的父亲
		if(!vis[t])
		{
			dfs(t, id);
			low[id] = min(low[id], low[t]);
		}
		else
		{
			low[id] = min(low[id], dfn[t]);
		}
	}
}

int tarjan()
{
	for(int i = 1; i <= n; i ++)
	{
		if(!vis[i])
			dfs(i, i);
	}

	for(int i = 1; i <= n; i ++)
	{
		for(int j = head[i]; j != -1; j = edge[j].next)
		{
			if(low[i] != low[edge[j].to])
```

```
                in[low[i]] ++;
            }
        }

        int ans = 0;
        for(int i = 1; i <= n; i ++)
        {
            if(in[i] == 1)
                ans ++;
        }
        return (ans + 1) / 2;
}

int main ()
{
        int r, a, b;
        while(~scanf("%d %d", &n, &r))
        {
            edge_total = 0;
            memset(head, -1, sizeof(head[0]) * (n+1));
            for(int i =   0; i < r; i ++)
            {
                scanf("%d %d", &a, &b);
                addEdge(a, b);
            }

            tarjan_init();
            printf("%d\n", tarjan());
        }
        return 0;
}
```

线性求中位数

```
#include<cstdio>

int find_mid(int arr[], int left, int right, int x)
{
        if(left >= right){
            return arr[left + x];
        }
        int mid = arr[left];
```

```
	int i = left;
	int j = right;
	while(i < j){
		while(i < j && arr[j] >= mid) j--;
		arr[i] = arr[j];
		while(i < j && arr[i] <= mid) i++;
		arr[j] = arr[i];
	}
	arr[j] = mid;
	if(i - left == x)
		return arr[i];
	if(i - left < x)
		return find_mid(arr, i + 1, right, x - (i - left + 1));
	else
		return find_mid(arr, left, i - 1, x);
}

int arr[10005];
int main(){
	int n;
	while(scanf("%d", &n) != EOF){
		for(int i = 0;i < n;i ++){
			scanf("%d", &arr[i]);
		}
		printf("%d\n", find_mid(arr, 0, n-1, n / 2));
	}
	return 0;
}
```

线性筛法求素数

```
#include<cstdio>
#include<string.h>

const int N = 25600000;
bool a[N];
int prime[N], num;


//a[i] = 0 表示 i 为素数
//prime[i]存储第 i 个素数
//num 存储一共多少个素数
```

```
void Prime(int n)              //n 表示最大界,但是不包括 n
{
        memset(a, 0, n * sizeof(a[0]));
        num = 0;
        a[0] = a[1] = 1;
        //不要冒昧的吧<改成<=
        //不然会错。亲测
        for(int i = 2; i < n; ++i)
        {
                if(!(a[i])) prime[num ++] = i;
                for(int j = 0; j < num && i * prime[j] < n; ++j)
                {
                        a[i * prime[j]] = 1;
                        if(!(i % prime[j])) break;
                }
        }
}

int main ()
{
        Prime(200005);
        printf("%d\n", num);
        for(int i =0; i < num; i ++)
        {
                printf("%d ", prime[i]);
        }
        printf("\n");
        for(int i = 0; i < 100; i ++)
                if(!a[i])
                printf("%d ", i);
        return 0;
}
```

最小费用最大流

```
#include<cstdio>
#include<string.h>
#include<queue>
#include<cmath>
using namespace std;

const int maxNode = 210;
const int INF = 0x3fffffff;

bool inq[maxNode];
char org[105][105];
int    pre[maxNode], res[maxNode][maxNode], cost[maxNode][maxNode], d[maxNode];

struct node
{
        int x, y;
}h[maxNode], m[maxNode];

bool SPFA(int s, int t)
{
        queue<int> q;
        memset(inq, 0, sizeof(inq));
        memset(pre, -1, sizeof(pre));
        inq[s] = 1;
        q.push(s);
        for(int i = s; i <= t; i ++)
                d[i] = INF;
        d[s] = 0;

        while(!q.empty())
        {
                int u = q.front();
                q.pop();
                inq[u] = 0;
//                 printf("%d\n", u);
                for(int i = s; i <= t; i ++)
                {
                        if(res[u][i] && d[u] + cost[u][i] < d[i])
                        {
                                d[i] = d[u] + cost[u][i];
                                pre[i] = u;
                                if(!inq[i])
                                {
                                        inq[i] = 1;
                                        q.push(i);
                                }
                        }
                }
        }

        if(pre[t] == -1)
```

```
                return false;
        return true;
}

int MCMF(int s, int t)
{
        int mincost = 0;
        while(SPFA(s, t))
        {
//              printf("%d %d\n", s, t);
                int v = t;
                while(v != -1)
                {
//                      printf("%d ", v);
                        res[pre[v]][v] -= 1;
                        res[v][pre[v]] += 1;
                        v = pre[v];

                }
//              printf("%d\n", d[t]);
                mincost += d[t];
        }
        return mincost;
}

int main ()
{
        int r, c;
        while(~scanf("%d %d", &r, &c) && r && c)
        {
                for(int i = 1; i <= r; i ++)
                {
                        scanf("%s", org[i]+1);
                }

                int house = 0, man = 0;
                for(int i = 1; i <= r; i ++)
                {
                        for(int j = 1; j <= c; j ++)
                        {
                                if(org[i][j] == 'H')
                                {
                                        h[house].x = i;
                                        h[house].y = j;
                        house ++;
                    }
                    if(org[i][j] == 'm')
                    {
                        m[man].x = i;
                        m[man].y = j;
                        man ++;
                    }
                }
            }
//          printf("house man: %d %d\n", house, man);
            memset(res, 0, sizeof(res));
            memset(cost, 0, sizeof(cost));
            int s = 0, t = house + man + 1;
            for(int i = 1; i <= house; i ++)
                res[s][i] = 1;

            for(int i = 0; i < house; i ++)
            {
                for(int j = 0; j < man; j ++)
                {
                    int dis = abs(h[i].x - m[j].x) + abs(h[i].y - m[j].y);
                    res[i + 1][j + house + 1] = 1;
                    cost[i + 1][j + house + 1] = dis;
                    cost[j + house + 1][i + 1] = -dis;
                }
            }

            for(int i = house + 1; i < t; i ++)
                res[i][t] = 1;

//          for(int i = s; i <= t; i ++)
//          {
//              for(int j = s; j <= t; j ++)
//              {
//                  printf("%d ", res[i][j]);
//              }
//              printf("\n");
//          }
//
//          for(int i = s; i <= t; i ++)
//          {
//              for(int j = s; j <= t; j ++)
//              {
```

```
//                                  printf("%d ", cost[i][j]);
//                          }
//                          printf("\n");
//                  }

                printf("%d\n", MCMF(s, t));
        }
        return 0;
}
```