ACM 常用模板

V2.0

MXY

2014/10/7

目录

LCARMQ、树状数组

```
#define MAXN 200003
#define LOGMN 30
#define lowbit(x) (x&(-x))
int n, q, s;
int pfirst[MAXN], plast[MAXN], dis[MAXN];   //初次和最后的位置
int ehead[MAXN], edge[MAXN<<1], enext[MAXN<<1], wedge[MAXN<<1]; //边集（以下全部 1 起始）
bool eout[MAXN<<1];    //对应边是不是出边
int euler[MAXN<<1], depth[MAXN<<1], anc[MAXN<<1], cnt, sig;//欧拉序列，伪深序列，对应节点
int trearr[MAXN<<1], rmqarr[MAXN][LOGMN];//树状数组，RMQ 数组
double lg2 = log(2.00);
//得到欧拉序列
void DFS(int st, int dist, int par){
    int i, now;
    euler[++cnt] = st;
    pfirst[st] = cnt;
    dis[st] = dist;
    now = ++sig;
    anc[now] = st;
    depth[cnt] = now;
    for(i=ehead[st];i>0;i=enext[i]){
        if(edge[i] == par)     continue;
        eout[i] = true;
        DFS(edge[i], dist + wedge[i], st);
        euler[++cnt] = st;
        depth[cnt] = now;
    }
    plast[st] = cnt;
}
//RMQ 初始化
void rmqst_init(){
    int i, j, m = (int)(floor(log((double)cnt)/lg2));
    for(i=1;i<=cnt;i++)
        rmqarr[i][0] = depth[i];
    for(j=1;j<=m;j++)
        for(i=1;i<=cnt-(1<<(j-1));i++)
            rmqarr[i][j]=min(rmqarr[i][j-1],rmqarr[i + (1<<(j-1))][j-1]);
}
//RMQ 计算
int rmq_get(int a, int b){
    if(b < a) swap(a, b);
```

```
	int m = (int)(floor(log((double)(b-a+1))/lg2));
	return min(rmqarr[a][m], rmqarr[b-(1<<m)+1][m]);
}
//树状数组求和
int getsum(int i){
	int sum = 0;
	for(;i>0;i-=lowbit(i))
		sum += trearr[i];
	return sum;
}
//树状数组更新
void tarr_upd(int i, int val){
	for(;i<=cnt;i+=lowbit(i))
		trearr[i] += val;
}
//计算结果
int calc(int a, int b){
	int lca, dc;
	int aa = pfirst[a], bb = pfirst[b];
	if(a == b)  return 0;
	if(aa > bb){
		swap(a, b);
		swap(aa, bb);
	}
	lca = anc[rmq_get(aa, bb)];
	dc = dis[lca] + getsum(pfirst[lca]);
	return ((dis[a] + getsum(aa)) + (dis[b] + getsum(bb)) - (dc << 1));
}
```

Tarjan 查割点、桥

```
int n, m;
int DFN[MAXN], low[MAXN], vis[MAXN];
int oplst[MAXN], pos, root, rtcnt;
bool cut;
int ehead[MAXN], edge[MAXM], enext[MAXM];
//Tarjan 算法查割点
void Tarjan(int cur, int dep, int par){
	int i, j;
	DFN[cur] = low[cur] = dep;
	vis[cur] = 1;
	for(i=ehead[cur];i>=0;i=enext[i]){
		j = edge[i];
		if(vis[j] == 2)    //废除顶点
			continue;
```

```
        if(vis[j] == 0)   {
            Tarjan(j, dep+1, cur);
            if(cur == root)
                rtcnt++;
            else{
                low[cur] = min(low[cur], low[j]);
                if(low[j] >= DFN[cur])
                    cut = true;
            }
        }else if(j != par)
            low[cur] = min(low[cur], DFN[j]);
    }
}
```

一个顶点 u 是割点，当且仅当满足(1)或(2)

(1) u 为树根，且 u 有多于一个子树。

(2) u 不为树根，且满足存在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。

一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)。

一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。

把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1。

统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加(leaf+1)/2 条边，就能使树达到边二连通，所以至少添加的边数就是(leaf+1)/2。

具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一定是双连通的。

然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2 次，把所有点收缩到了一起。

线段树和圆扫描线

线段树常用标记：区间加和，区间置位、复位、取反状态。

常用维护内容：和，左中右连续最长长度，被覆盖次数。

圆的嵌套时，把圆的上下弧用扫描线加入到 set 中（保序性）

圆的相交时，用双扫描线加入和删除圆

圆的面积并，把圆的未覆盖弧按逆时针用向量链接，计算未覆盖弓形的面积和多边形的顺序面积。

线段围住点，把线段端点和点连线角度作为边的权，判负环。

最小割经典构图

最大点权值闭合子图

原图中所有边化为容量为 INF 的边，增加源 S 向正权值点连容量为 wi 的边，增加汇 T 接受负权值点的容量为-wi 的边，取最小割后，源点 S 所在的子图除去 S 即为最大权闭合图。

平均边权最小割
二分结果 p，取 wi=wi-p，对所有大于 0 的 wi 求最小割后加上所有小于 0 的边权，最后结果 f 为 0 的 p 即是解。f>0 则 left=p，否则 right=p。

最大密度子图
二分结果 p,原有无向边替换为容量为 1 的有向边,增设 S 到所有点容量为大 M(可以取 m),增设 T 接受所有点 v 容量为大 M+2g-degv，最小割 f 满足 n*m>f 则 left=g，否则 right=g，eps=1/n^2。

最短路

原始问题：

$$\min \sum_{(i,j)} w_{ij} x_{ij}$$
$$\text{s.t.} \sum_{j} x_{ij} - \sum_{j} x_{ji} = \begin{cases} 1, i = s \\ -1, i = t \\ 0, etc. \end{cases}$$
$$x_{ij} \geq 0$$

对偶问题：

$$\max \ (u_t - u_s)$$
$$\text{s.t.} \ u_j - u_i \leq w_{ij} \forall (i,j)$$

Bellman 距离标号方程（只有正权圈）：

$$\begin{cases} u_s = 0 \\ u_j = \min\limits_{i \neq j} (u_i + w_{ij}) \end{cases}$$

Warshall 迭代方程：

$$\begin{cases} u_{ii}^1 = 0 \\ u_{ij}^1 = w_{ij}, i \neq j \\ u_{ij}^{k+1} = \min\{u_{ij}^k, u_{ik}^k + u_{kj}^k\}, i,j,k = 1,2,\dots,n \end{cases}$$

最小树形图（朱刘算法）

```
typedef int COST;
#define MAXP 555
#define MAXQ 10005
#define INF    0x3fffffff
struct EDGE
{
	int u, v;
	COST w;
}edge[MAXQ];
int pcnt, ecnt;
int pre[MAXP], id[MAXP], vis[MAXP];
COST minin[MAXP];
```

```
bool DMST(int root, COST& ret){
    int i, u, v, cnt;
    ret = 0;
    while(true){
        //找每个顶点的最小入弧
        for(i=0;i<pcnt;i++)
            minin[i] = INF;
        for(i=0;i<ecnt;i++){
            u = edge[i].u;
            v = edge[i].v;
            if(edge[i].w < minin[v] && u != v){
                pre[v] = u;
                minin[v] = edge[i].w;
            }
        }
        for(i=0;i<pcnt;i++)
            if(i != root && minin[i] == INF)
                return false;
        //找环缩点
        cnt = 0;
        for(i=0;i<pcnt;i++){
            id[i] = -1;
            vis[i] = -1;
        }
        minin[root] = 0;
        for(i=0;i<pcnt;i++){
            ret += minin[i];
            v = i;
            while(vis[v] != i && id[v] == -1 && v != root){
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && id[v] == -1){
                for(u=pre[v];u!=v;u=pre[u])
                    id[u] = cnt;
                id[v] = cnt++;
            }
        }
        if(cnt == 0)
            break;
        //重构图
        for(i=0;i<pcnt;i++)
            if(id[i] == -1)
                id[i] = cnt++;
```

```
		for(i=0;i<ecnt;i++){
			u = edge[i].u;
			v = edge[i].v;
			edge[i].u = id[u];
			edge[i].v = id[v];
			if(id[u] != id[v])
				edge[i].w -= minin[v];
		}
		pcnt = cnt;
		root = id[root];
	}
	return true;
}
```

欧拉回路

```
procedure Euler-circuit(st)
	for  st 的每个邻接顶点 v do
		if 边(st,v)无标记 then
			标记(st,v)
			标记(v,st)?（无向）
			Euler-circuit(v)
			(st,v)入栈
最后全部出栈即可
```

全局最小割

```
#define MAXN 505
#define MAXI 0x3fffffff
int mat[MAXN][MAXN], wage[MAXN];
bool vis[MAXN], bool used[MAXN];
int n, m, s, t, mcut;
//p - 次数
int Prim(int p){
	int i, j, m, v, max;
	for(i=1;i<=n;i++){ //构建临时树
		wage[i] = mat[1][i];
		vis[i] = false;
	}
	t = 1;
	wage[1] = 0;
	vis[1] = true;
	for(i=1;i<=n-p;i++){
		max = 0;
		for(j=1;j<=n;j++){		//寻找最大联通数
```

```
            if(wage[j] >= max && !vis[j] && !used[j]){
                m = j;
                max = wage[j];
            }
        }
        if(t==m)
            return wage[t];
        s = t;
        t = m;
        vis[m] = true;
        for(j=1;j<=n;j++)            //更新 wage
            if(!vis[j] && !used[j] && j!=m)
                wage[j] += mat[m][j];
    }
    return wage[t];
}
int Stoer_Wagner()
{
    int i, j;
    mcut = MAXI;
    memset(used, 0, sizeof(used));
    for(i=1;i<=n-1;i++)
    {
        mcut = min(mcut, Prim(i));
        if(mcut == 0)
            return 0;
        used[t] = true;
        for(j=1;j<=n;j++)
            if(!used[j] && j!=s)
            {
                mat[j][s] += mat[j][t];
                mat[s][j] += mat[t][j];
            }
    }
    return mcut;
}
```

其他图论

混合图欧拉回路（最大流）

A.某节点 i 入度大于出度，就从源点 S 连一条流量为(in[i] - out[i]) / 2 的边到 i 。

B.某节点 i 出度大于入度，就从 i 连一条流量为(out[i] - in[i]) / 2 的边到汇点 T。

C.遍历 <i, j> 若假定 i, j 之间连接了 mp[i][j] 条无向边，那么连接一条从 j 到 i 流量为 mp[i][j]的边。为什么反过来，因为这是我们假设的，意味着我们有多少反悔的资本。

第 K 最短路
先求所有点到 T 的距离，然后 A-star

DAG 图多 DP
流量题注意流和割的对偶性

Burnside Lemma 和简单数论

```
typedef signed long int SINT;
#define MODN 9973
#define MAXN 10005
#define PRIMENUM 3410
long primes[PRIMENUM] = {};
bool notprime[32000];
//n 各个素因子，幂次，积，n 快速取幂
long nfac, faci[100], fcnt[100], fval[100], nmod[32];
SINT ans;                    //最终答案
int n;
//筛法预处理素数
void init_prime(){
        int i, j = 0, k = 0;
        for(i=2;k<PRIMENUM;i++){
                if(notprime[i])
                        continue;
                for(j=i+i; j < 32000; j+=i)
                        notprime[j] = true;
                primes[k] = i;
                k++;
        }
}
//因子分解
void fact(int x){
        int i;
        nfac = 0;
        for(i=0;(i<PRIMENUM)&&(x>1);i++){
                if(x % primes[i] == 0){
                        faci[nfac] = primes[i];
                        fcnt[nfac] = 0;
                        fval[nfac] = 1;
                        do{
                                x /= primes[i];
                                fval[nfac] *= primes[i];
                                fcnt[nfac]++;
                        }while(x % primes[i] == 0);
                        nfac++;
```

```
		}
	}
	if(x > 1){
		faci[nfac] = x;
		fcnt[nfac] = 1;
		fval[nfac] = x;
		nfac++;
	}
}
//扩展欧几里得算法
SINT ExtendGcd(SINT a,SINT b,SINT &x,SINT &y){
	SINT d,tmp;
	if(b==0){
		x = 1;
		y = 0;
		return a;
	}
	d = ExtendGcd(b,a%b,x,y);
	tmp = x;
	x = y;
	y = tmp - (a/b)*y;
	return d;
}
//x 对 y 乘法逆元
SINT Divisor(SINT x, SINT y){
	SINT d, ret, tmp;
	d = ExtendGcd(x,y,ret,tmp);
	//不互质则无解
	if(d != 1)
		return -1;
	//变成最小正
	ret = (ret % y + y) % y;
	return ret;
}
SINT getd(int cnt)
{
	/////////这里填写具体计算内容
}
void DFS(int pfi, int fc, int phi){	//素因子序号，当前因子，n/i 的欧拉函数值
	int i, p;
	if(pfi < nfac)
		for(i=0,p=fval[pfi];i<=fcnt[pfi];i++,p/=faci[pfi])
			DFS(pfi + 1, fc * (fval[pfi] / p), phi * (p - p / faci[pfi]));
	else{
```

```
            ans += (getd(fc) * (SINT)phi) % MODN;
            ans %= MODN;
        }
}
int main()
{
        init_prime();
        while(true)
        {
            ////////这里处理输入
            ans = 0;
            fact(n);
            DFS(0, 1, 1);
            ans *= Divisor(n, MODN);
            ans %= MODN;
            printf("%lld\n",ans);
        }
        return 0;
}
```

三进制插头 DP

```
#define HASH    30007
#define STATE 1000010
struct MAPDP{    //开放定址 hash
        int head[HASH],next[STATE],size;
        QWORD state[STATE];
        QWORD f[STATE];
};
MAPDP dp[2]; //方案数[当前之前]
int cur;          //当前取数据的 dp
QWORD encode(BYTE code[]){ //编码状态(0=#,1=(,2=)，从左往右依次算)
        QWORD ret = 0;
        for(int i=0;i<m+1;i++){
            ret <<= 2;
            ret += code[i];
        }
        return ret;
}
void decode(QWORD st, BYTE code[]){ //解码状态
        for(int i=m;i>=0;i--)
        {
            code[i] = (st & 3);
            st >>= 2;
        }
```

```
}
void shrcode(BYTE code[]){        //换行右移位 ()##()# => #()##()
	for(int i=m;i>=1;i--)
	{
		code[i] = code[i-1];
	}
	code[0] = 0; //最左边不会被插
}
void hash_init(int obj){
	dp[obj].size = 0;
	memset(dp[obj].head , -1, sizeof(dp[obj].head));
}
int hash_find(QWORD state, int obj){
	for(int i=dp[obj].head[state % HASH];i!=-1;i=dp[obj].next[i])
	{
		if(dp[obj].state[i]==state) return i;
	}
	return -1;
}
void hash_in(QWORD state, QWORD count, int obj)
{
	int i = hash_find(state, obj);
	if(i != -1)
		dp[obj].f[i]+=count;
	else{
		dp[obj].state[dp[obj].size]=state;
		dp[obj].f[dp[obj].size]=count;
		dp[obj].next[dp[obj].size]=dp[obj].head[state % HASH];
		dp[obj].head[state % HASH]=dp[obj].size++;
	}
}
```

斜率 DP

```
LL dpcalc()
{
	int i, x, y;
	back = front = que[0] = dp[0] = 0;
	for(i=1;i<=n;i++){
		while(front < back && (gety(que[front]) - gety(que[front+1])) >= i * (getx(que[front]) - getx(que[front+1])))
			++front;
		dp[i] = ...;
		z = i - k + 1;
		if(z < k) continue;
```

```
        while(front < back){
            x = que[back-1];
            y = que[back];
            if((gety(x) - gety(y)) * (getx(y) - getx(z)) >= (gety(y) - gety(z)) * (getx(x) - getx(y)))
                --back;
            else
                break;
        }
        que[++back] = i;
    }
    return dp[n];
}
```

分治优化 DP

$$dp[i,j] = \min\{dp[i-1,k] + C[k,j]\}$$
$$A[i,j] = \min\{k|(dp[i,j] = dp[i-1,k] + C[k,j])\}$$
$$C[a,c]+C[b,d]<=C[a,d]+C[b,c]$$

compute(i,l,r,ol,or)

1. 令 m=(l+r)>>1
2. 寻找 k=ol..or,使得 dp[i,m]=dp[i-1,k]+C[k,m]最小
3. 如果 l==r，返回。否则执行 compute(i,l,m-1,ol,k);compute(i,m+1,r,k,or);

其他优化 DP

可以用单调栈，线段树或者四边形不等式。

矩阵算法

```
#define MAXN 105
#define TINY 1e-20
typedef double matrix2[MAXN][MAXN];
typedef double vector2[MAXN];
//三角分解 A
//输入：阶数 N，矩阵 A
//输出：A 为三角分解结果（下 L 上半 U）；INDX：行交换结果
bool LUDCMP(matrix2 A, const int n, int indx[]){
    double VV[MAXN], aamax, sum, dum;
    int i, j, k, imax;
    //选取行最大元为主元
    for(i=1;i<=n;i++){
        aamax = 0.0;
        for(j=1;j<=n;j++)
            if(abs(A[i][j]) > aamax)
                aamax = abs(A[i][j]);
```

```
        if(aamax == 0.0)
            return false;   //奇异矩阵不可分解
        VV[i] = 1.0 / aamax;
    }
    for(j=1;j<=n;j++){
        for(i=1;i<j;i++){
            sum = A[i][j];
            for(k=1;k<i;k++)
                sum -= A[i][k] * A[k][j];
            A[i][j] = sum;
        }
        aamax = 0;
        for(i=j;i<=n;i++){
            sum = A[i][j];
            for(k=1;k<j;k++)
                sum -= A[i][k] * A[k][j];
            A[i][j] = sum;
            dum = VV[i] * abs(sum);
            if(dum >= aamax){
                imax = i;
                aamax = dum;
            }
        }
        //行交换
        if(j != imax){
            for(k=1;k<=n;k++){
                dum = A[imax][k];
                A[imax][k] = A[j][k];
                A[j][k] = dum;
            }
            VV[imax] = VV[j];
        }
        indx[j] = imax;
        if(A[j][j] == 0)
            A[j][j] = TINY;
        if(j < n){
            dum = 1.0 / A[j][j];
            for(i=j+1;i<=n;i++)
                A[i][j] *= dum;
        }
    }
    return true;
}
//LU 求解方程
```

```
void LUBKSB(matrix2 A, const int n, int indx[], vector2 B){
	int i, j, ll, ii = 0;
	double sum;
	for(i=1;i<=n;i++){
		ll = indx[i];
		sum = B[ll];
		B[ll] = B[i];
		if(ii != 0){
			for(j=ii;j<i;j++)
				sum -= A[i][j] * B[j];
		}else if(sum != 0)
			ii = i;
		B[i] = sum;
	}
	for(i=n;i>0;i--){
		sum = B[i];
		if(i < n)
			for(j=i+1;j<=n;j++)
				sum -= A[i][j] * B[j];
		B[i] = sum / A[i][i];
	}
}
//奇异值分解
//A=>AWV',M 行 N 列，M>=N
bool SVDCMP(matrix2 A, const int M, const int N, vector2 W, matrix2 V){
	double RV1[MAXN];
	int i, j, l, its, nm, k, tmp;
	double g = 0, s, scale1 = 0, c, f, h, x, y, z, sgn, anorm = 0;
	if(M < N) return false;
	//构建双对角矩阵?
	for(i=1;i<=N;i++){
		l = i + 1;
		RV1[i] = scale1 * g;
		g = s = scale1 = 0;
		if(i <= M){
			for(k=i;k<=M;k++)
				scale1 += abs(A[k][i]);
			if(scale1 != 0){
				for(k=i;k<=M;k++){
					A[k][i] /= scale1;
					s += A[k][i] * A[k][i];
				}
				f = A[i][i];
				sgn = (f > 0) ? 1 : -1;
```

```
            g = -sqrt(s) * sgn;
            h = f * g - s;
            A[i][i] = f - g;
            if(i != N)
                for(j=l;j<=N;j++){
                    s = 0;
                    for(k=i;k<=M;k++)
                        s += A[k][i] * A[k][j];
                    f = s / h;
                    for(k=i;k<=M;k++)
                        A[k][j] += f * A[k][i];
                }
            for(k=i;k<=M;k++)
                A[k][i] *= scale1;
        }
    }
    W[i] = scale1 * g;
    g = s = scale1 = 0;
    if(i <= M && i != N){
        for(k=l;k<=N;k++)
            scale1 += abs(A[i][k]);
        if(scale1 != 0){
            for(k=l;k<=N;k++){
                A[i][k] /= scale1;
                s += A[i][k] * A[i][k];
            }
            f = A[i][l];
            sgn = (f > 0) ? 1 : -1;
            g = -sqrt(s) * sgn;
            h = f * g - s;
            A[i][l] = f - g;
            for(k=l;k<=N;k++)
                RV1[k] = A[i][k] / h;
            if(i != M)
                for(j=l;j<=M;j++){
                    s = 0;
                    for(k=l;k<=N;k++)
                        s += A[j][k] * A[i][k];
                    for(k=l;k<=N;k++)
                        A[j][k] += s * RV1[k];
                }
            for(k=l;k<=N;k++)
                A[i][k] *= scale1;
        }
```

```
        }
        anorm = max(anorm, abs(W[i]) + abs(RV1[i]));
    }
    for(i=N;i>=1;i--){
        if(i < N){
            if(g >= 0){
                for(j=l;j<=N;j++)
                    V[j][i] = (A[i][j] / A[i][l]) / g;
                for(j=l;j<=N;j++){
                    s = 0;
                    for(k=l;k<=N;k++)
                        s += A[i][k] * V[k][j];
                    for(k=l;k<=N;k++)
                        V[k][j] += s * V[k][i];
                }
            }
            for(j=l;j<=N;j++)
                V[i][j] = V[j][i] = 0;
        }
        V[i][i] = 1;
        g = RV1[i];
        l = i;
    }
    for(i=N;i>=1;i--){
        l = i + 1;
        g = W[i];
        for(j=l;j<=N;j++)
            A[i][j] = 0;
        if(g != 0){
            g = 1.0 / g;
            for(j=l;j<=N;j++){
                s = 0;
                for(k=l;k<=M;k++)
                    s += A[k][i] * A[k][j];
                f = (s / A[i][i]) * g;
                for(k=i;k<=M;k++)
                    A[k][j] += f * A[k][i];
            }
            for(j=i;j<=M;j++)
                A[j][i] *= g;
        }else{
            for(j=i;j<=M;j++)
                A[j][i] = 0;
        }
```

```
            A[i][i] += 1;
        }
        for(k=N;k>=1;k--){
            //迭代 30 次
            for(its=1;its<=30;its++){
                for(l=k;l>=1;l--){
                    nm = l - 1;
                    if(abs(RV1[l]) + anorm == anorm)
                        goto l2;
                    if(abs(W[nm]) + anorm == anorm)
                        goto l1;
                }
l1:             c = 0;
                s = 1;
                for(i=l;i<=k;i++){
                    f = s * RV1[i];
                    if(abs(f) + anorm != anorm){
                        g = W[i];
                        h = sqrt(f * f + g * g);
                        W[i] = h;
                        h = 1.0 / h;
                        c = g * h;
                        s = -f * h;
                        for(j=1;j<=M;j++){
                            y = A[j][nm];
                            z = A[j][i];
                            A[j][nm] = y * c + z * s;
                            A[j][i] = - y * s + z * c;
                        }
                    }
                }
l2:             z = W[k];
                if(l == k){
                    if(z < 0){
                        W[k] = -z;
                        for(j=1;j<=N;j++)
                            V[j][k] = -V[j][k];
                    }
                    goto l3;
                }
                if(its == 30)
                    return false;
                x = W[l];
                nm = k - 1;
```

```
y = W[nm];
g = RV1[nm];
h = RV1[k];
f = ((y-z)*(y+z) + (g-h)*(g+h)) / (2 * h * y);
g = sqrt(f * f + 1);
sgn = (f > 0) ? 1 : -1;
f = ((x-z)*(x+z) + h * ((y / (f+abs(g)*sgn)) - h)) / x;
c = s = 1;
tmp = nm;
for(j=l;j<=tmp;j++){
	i = j + 1;
	g = RV1[i];
	y = W[i];
	h = s * g;
	g = g * c;
	z = sqrt(f * f + h * h);
	RV1[j] = z;
	c = f / z;
	s = h / z;
	f = (x*c) + (g*s);
	g = -(x*s) + (g*c);
	h = y * s;
	y = y * c;
	for(nm=1;nm<=N;nm++){
		x = V[nm][j];
		z = V[nm][i];
		V[nm][j] = (x * c) + (z * s);
		V[nm][i] = -(x * s) + (z * c);
	}
	z = sqrt(f * f + h * h);
	W[j] = z;
	if(z != 0){
		z = 1.0 / z;
		c = f * z;
		s = h * z;
	}
	f = (c * g) + (s * y);
	x = -(s * g) + (c * y);
	for(nm=1;nm<=M;nm++){
		y = A[nm][j];
		z = A[nm][i];
		A[nm][j] = (y*c) + (z*s);
		A[nm][i] = -(y*s) + (z*c);
	}
```

```
                }
                RV1[l] = 0;
                RV1[k] = f;
                W[k] = x;
            }
l3:         continue;
        }
        return true;
}
//QR 分解，输入矩阵 A 和阶数 N，输出 Q 和 R
void QRDCMP(matrix2 A, matrix2 Q, matrix2 R, int N){
        int i, j, k;
        double f, a;
        //单位化 i,j
        for(i=1;i<=N;i++)
            for(j=1;j<=N;j++)
                Q[i][j] = (i==j);
        //k 轮分解
        for(k=1;k<=N;k++){
            //把 A[k][k]..A[N][k]变成变换基准向量 v
            a = 0;
            for(i=k;i<=N;i++)
                a += A[i][k] * A[i][k];
            a = sqrt(a);
            //f = sqrt(2 * a * (a - A[k][k]));
            //上面那种取巧的写法会导致精度的严重损失
            f = (A[k][k] - a) * (A[k][k] - a);
            for(i=k+1;i<=N;i++)
                f += A[i][k] * A[i][k];
            f = sqrt(f);
            if(fabs(f) < TINY)
                continue;
            A[k][k] -= a;
            for(i=k;i<=N;i++)
                A[i][k] /= f;
            //逐列变换 A
            for(j=k+1;j<=N;j++){
                f = 0;
                for(i=k;i<=N;i++)
                    f += 2 * A[i][j] * A[i][k];
                for(i=k;i<=N;i++)
                    A[i][j] -= f * A[i][k];
            }
            //逐列变换 Q
```

```
		for(j=0;j<=N;j++){
			f = 0;
			for(i=k;i<=N;i++)
				f += 2 * Q[i][j] * A[i][k];
			for(i=k;i<=N;i++)
				Q[i][j] -= f * A[i][k];
		}
		//确定 R 的第 k 列
		for(i=k+1;i<=N;i++)
			A[i][k] = 0;
		A[k][k] = a;
	}
	//转置 Q
	for(i=1;i<=N;i++)
		for(j=i+1;j<=N;j++)
			std::swap(Q[i][j], Q[j][i]);
	for(i=1;i<=N;i++)
		for(j=1;j<=N;j++)
			R[i][j] = A[i][j];
}
```

AC 自动机

```
const int MAXC = 401;	//最大状态数
const int MAXL = 11;	//最大长度
const int ALB  = 65;	//字母表
class ACA{
public:
	struct NODE{
		char ch;		//当前字符
		int fail, next[ALB];//失败、转移指针
		bool ed, vis;		//终结状态和探查
		inline int& operator[](int i){return next[i];};
	}node[MAXC];
	int cnt;
	void init(){
		cnt = 1;
		memset(&node[0], -1, sizeof(NODE));
		node[0].ed = false;
	}
	void insert(char *str, int len){
		int i, cur = 0;
		for(i=0;i<len;i++){
			if(node[cur][str[i]] == -1){
				node[cur][str[i]] = cnt;
```

```
                memset(&node[cnt], -1, sizeof(NODE));
                node[cnt].ch = str[i];
                node[cnt].ed = false;
                cnt++;
            }
            cur = node[cur][str[i]];
        }
        node[cur].ed = true;
}
void build(){
        queue<int> Q;
        int cur, j, p;
        for(j=0;j<ALB;j++){
            if(node[0][j] >= 0){
                Q.push(node[0][j]);
                node[node[0][j]].fail = 0;
            }else
                node[0][j] = 0;
        }
        while(!Q.empty()){
            cur = Q.front();
            Q.pop();
            p = node[cur].fail;
            for(j=0;j<ALB;j++){
                if(node[cur][j] >= 0){
                    node[node[cur][j]].fail = node[p][j];
                    node[node[cur][j]].ed |= node[node[p][j]].ed;//纯匹配时不加这句
                    Q.push(node[cur][j]);
                }else
                    node[cur][j] = node[p][j];
            }
        }
}
int match(char *str, int len){
        int i, p, cur = 0, ret = 0;
        for(i=0;i<cnt;i++)
            node[i].vis = false;
        for(i=0;i<len;i++){
            p = cur = node[cur][str[i]];
            while(p > 0 && !node[p].vis){
                ret += node[p].ed;
                node[p].vis = true;
                p = node[p].fail;
            }
```

```
        }
        return ret;
    }
}aca;
```

树链剖分

```
typedef void(*CBFUNC)(int x, int y, bool last);
struct NODE{
    int head, siz, son, par, dep; //边临接表，子树尺寸,重儿子,父亲,深度
    int w, top;          //映射位置，链头
}node[MAXN];
int enext[MAXM], edge[MAXM];
int N, Q, ecnt, tz;
int que[MAXN], qt, qh;
//进行剖分
void DFS(int x){
    int i, j, u, top, nxt;
    qh = qt = 0;
    que[qt++] = x, node[x].par = -1, node[x].dep = 0;
    while (qh < qt){
        u = que[qh++];
        for(j=node[u].head;j>=0;j=enext[j])
            if (edge[j] != node[u].par){
                node[edge[j]].par = u;
                que[qt++] = edge[j];
                node[edge[j]].dep = node[u].dep + 1;
            }
    }
    for (i = N - 1; i >= 0; --i){
        u = que[i];
        node[u].siz = 1;
        for(j=node[u].head;j>=0;j=enext[j])
            if(node[edge[j]].par == u)
                node[u].siz += node[edge[j]].siz;
    }
    for (i = 0; i < N; ++i) {
        u = que[i];
        if(node[u].top != -1)
            continue;
        top = u;
        while(true){
            node[u].top = top;
            node[u].w = ++tz;
            nxt = -1;
```

```
            for(j=node[u].head;j>=0;j=enext[j])
                if (node[edge[j]].par == u)
                    if (nxt == -1 || node[edge[j]].siz > node[nxt].siz)
                        nxt = edge[j];
            if (nxt == -1)
                break;
            u = nxt;
        }
    }
}
//剖分路径
void breakdown(int va, int vb, CBFUNC cbf){
    int f1 = node[va].top, f2 = node[vb].top;
    while(f1 != f2){
        if(node[f1].dep < node[f2].dep){
            std::swap(f1, f2);
            std::swap(va, vb);
        }
        cbf(va, f1, false);
        va = node[f1].par;
        f1 = node[va].top;
    }
    if(node[va].dep > node[vb].dep)
        std::swap(va, vb);
    cbf(vb, va, true);
}
```

重心分治

```
    //重心划分
    void divide(int cur, int ct){
        int p, i, j, ed, sz;
        tail = 1;
        queue[0] = cur;
        node[cur].par = -1;
        node[cur].subsz = 1;    //和队列中的 i>0 一致
        node[cur].mson = 0;
        //划分重心
        for(head=0;head<tail;head++){
            p = queue[head];
            for(j=node[p].head;j>=0;j=edge[j].next){
                ed = edge[j].b;
                if(ed != node[p].par && !node[ed].vis){
                    node[ed].subsz = node[ed].mson = 0;
                    node[ed].par = p;
```

```
                    queue[tail] = ed;
                    tail++;
                }
            }
        }
        for(i=tail-1;i>0;i--){
            p = queue[i];
            ed = node[p].par;
            node[p].subsz++;
            node[ed].subsz += node[p].subsz;
            node[ed].mson = max(node[ed].mson, node[p].subsz);
        }
        sz = node[cur].subsz;
        for(i=0;i<tail;i++){
            p = queue[i];
            node[p].mson = max(node[p].mson, sz - node[p].subsz);
            if(node[p].mson < node[cur].mson)
                cur = p;
        }
        //建立结构
        node[cur].tarr = TARR(tail);
        if(ct > 0)
            node[cur].parr = TARR(tail << 1);
        node[cur].vis = true;
        node[cur].path[ct] = cur;
        node[cur].dis[ct] = 0;
        node[cur].par = -1;
        //更新重心数据
        tail = 1;
        queue[0] = cur;
        for(head=0;head<tail;head++){
            p = queue[head];
            for(j=node[p].head;j>=0;j=edge[j].next){
                ed = edge[j].b;
                if(ed != node[p].par && !node[ed].vis){
                    node[ed].dis[ct] = node[p].dis[ct] + 1;
                    node[ed].path[ct] = cur;
                    node[ed].par = p;
                    queue[tail] = ed;
                    tail++;
                }
            }
            node[cur].tarr.update(node[p].dis[ct], node[p].w);
            if(ct > 0)
```

```
                node[cur].parr.update(node[p].dis[ct-1], node[p].w);
        }
        //递归划分子树
        for(j=node[cur].head;j>=0;j=edge[j].next){
            ed = edge[j].b;
            if(!node[ed].vis)
                this->divide(ed, ct+1);
        }
};
int query(int cur, int dis){
        int i, sum = 0;
        for(i=0;i==0||node[cur].path[i-1]!=cur;i++){
            sum += node[node[cur].path[i]].tarr.getsum(dis - node[cur].dis[i]);
            if(i > 0)
                sum -= node[node[cur].path[i]].parr.getsum(dis - node[cur].dis[i-1]);
        }
        return sum;
};
```