ICPC Template

BJTU_SpadeAce 冯大蕗

Last build at October 23, 2015

Contents

1 基础

1.1 头文件

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <set>
#include <map>
#include <vector>
#include <queue>
#include <algorithm>
#include <cmath>
#include <sstream>
#include <fstream>
using namespace std;
#define LL long long
#define u32  unsigned int
#define AA first
#define BB second
typedef pair<int, int> PII;
#define cmin(x, y) x = min(x, y)
#define cmax(x, y) x = max(x, y)
#define PB(x) push_back(x)
#define SZ size()
#define MP(a, b) make_pair(a, b)
#define OP begin()
#define ED end()
#define CLR clear()
#define INS(x) insert(x)
#define FOR(i, n) for(int i = 0; i < (n); i++)
#define FORR(i, n) for(int i = 1; i <= (n); i++)
#define REP(i, L, R) for(int i = (L); i <= (R); i++)
#define MEM(a) memset(a, 0, sizeof a)
#define ECH(x) for(__typeof x.OP it = x.OP; it != x.ED; it++)
#define ONES(x) __builtin_popcount(x)
/*==========================================================*/
```

1.2 二进制

__builtin_clz 前导 0
__builtin_ctz 后缀 0
__builtin_popcount 1 个数

1.3 稀疏矩阵乘法

```
int n; //Matrix Size
void matrixMul(LL a[][],LL b[][],LL c[][])
{
  /*
  * CLEAR c[][]
  */
    for(int i = 0;i < n; i++)
```

```
    {
        for(int j = 0;j < n; j++)
        {
            if(0 == a[i][j])continue;
            for(int k = 0; k < n; k++)
            c[i][k] += a[i][j] * b[j][k];
        }
    }
}
```

2 图论

2.1 2-SAT

```
#define bf(x) ((x)<<1)
#define bt(x) ((x)<<1|1)
int ind[N], id[N];
int t[M], nt[M];
int dfn[N], low[N];
int s[N];
bool vis[N];
int l;
int n;
int cnt, num, idn;
int a[N], b[N], c[N];

void add(int i, int j)
{
    cnt++;
    t[cnt] = j;
    nt[cnt] = ind[i];
    ind[i] = cnt;
}

void tarjan(int x)
{
    num++;
    dfn[x] = low[x] = num;
    s[++l] = x;
    vis[x] = true;
    for(int k = ind[x]; k != -1; k = nt[k])
    {
        if(dfn[t[k]] == 0) tarjan(t[k]);
        if(vis[t[k]]) low[x] = min(low[x], low[t[k]]);
    }
    if(dfn[x] == low[x])
    {
        idn++;
        while(true)
        {
            int tmp = s[l--];
            id[tmp] = idn;
            vis[tmp] = false;
            if(tmp == x) break;
```

```
        }
    }
}

bool chk(int M)
{
    for(int i = 0; i < 2 * n; i++) ind[i] = -1;
    cnt = 0;
  /*
  * ADD Edge
  */
    memset(vis, 0, sizeof vis);
    num = 0;
    idn = 0;
    l = 0;
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(id, 0, sizeof id);
    for(int i = 0; i < 2 * n; i++)
    {
        if(dfn[i] == 0) tarjan(i);
    }
    for(int i = 0; i < 2 * n; i++)
    {
        if(id[i] == id[i ^ 1]) return false;
    }
    return true;
}
```

2.2 哈密尔顿回路

设 dp[i][j] 表示站在 i 点, 盘面状态为 j 时的最短路, 最后答案需要遍历所有 j == (1 « n) - 1 注意有的题需要用 Floyd 预处理最短路

```
int f[N][N];
int dp[N][(1 << 18)];
int n, m;

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        scanf("%d%d", &n, &m);
        for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) f[i][j] = INF;
        for(int i = 1; i <= n; i++) f[i][i] = 0;
        for(int i = 0; i < m; i++)
        {
            int x, y, z; scanf("%d%d%d", &x, &y, &z);
            f[x][y] = min(f[x][y], z);
            f[y][x] = min(f[y][x], z);
        }
        for(int k = 1; k <= n; k++) for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++)
            f[i][j] = min(f[i][k] + f[k][j], f[i][j]);
```

```

        for(int i = 1; i <= n; i++) for(int j = 0; j < 1 << (n + 1); j++) dp[i][j] = INF;
        dp[1][2] = 0;

        for(int j = 0; j < (1 << (n + 1)); j++)
        {
            for(int i = 1; i <= n; i++)
            if(((1 << i) & j) != 0)
            {
                for(int k = 1; k <= n; k++)
                    dp[k][(1 << k) | j] = min(dp[i][j] + f[i][k], dp[k][(1 << k) | j]);
            }
        }
        int ans = INF;
        for(int i = 1; i <= n; i++) { ans = min(ans, dp[i][(1 << (n + 1)) - 2] + f[i][1]); }
        printf("%d\n", ans);
    }
    return 0;
}
```

2.3 欧拉回路

存在条件为奇数度的点 0 个 (无向图), 所有点入度等于出度 (有向图), 求欧拉回路需要注意访问过的边需要删除

2.4 DINIC

```
int f[N];
int q[N];
bool vis[N];
int h[N];
int ind[N];
int t[M], c[M], nt[M], opp[M];
int cnt;
int n, m;
int ST, ED;

int add(int a, int b, int C)
{
    t[cnt] = b;
    nt[cnt] = ind[a];
    ind[a] = cnt;
    c[cnt] = C;
    return cnt++;
}

void add1(int a, int b, int c)
{
    static int x, y;
    x = add(a, b, c);
```

```
    y = add(b, a, 0);
    opp[x] = y; opp[y] = x;
}

bool bfs()
{
    int l = 0, r = 0;
    q[l] = ST;
    memset(h, 0, sizeof h);
    h[ST] = 1;
    while(l <= r)
    {
        int x = q[l++];
        for(int k = ind[x]; k != -1; k = nt[k])
        if(!h[t[k]] && c[k] > 0)
        {
            h[t[k]] = h[x] + 1;
            q[++r] = t[k];
        }
    }
    return h[ED] != 0;
}

int dfs(int x, int p)
{
    if(x == ED) return p;
    bool flg = false;
    int tot = 0;
    for(int k = ind[x]; k != -1; k = nt[k])
    if(h[t[k]] == h[x] + 1)
    {
        int d = dfs(t[k], min(p, c[k]));
        if(d)
        {
            p -= d;
            tot += d;
            c[k] -= d;
            flg = true;
            c[opp[k]] += d;
        }
    }
    return tot;
}

int dinic()
{
    int ans = 0, tmp;
    while(bfs())
    {
        while(true)
        {
            int fw = dfs(ST, INF);
            if(!fw) break;
            else ans += fw;
```

```
        }
    }
    return ans;
}
```

2.5 费用流

增广路版费用流，复杂度 $O(n^2m)$

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
#define N 3000
#define M 100000
#define INF 0x7fffffff
#define LL long long

int ind[N], pre[N], f[N];
bool vis[N];
int bg[M], t[M], nt[M], c[M], op[M], v[M];
int cnt;
int S, T;
int n, m;

int add(int a, int b, int C, int V)
{
    bg[cnt] = a;
    t[cnt] = b;
    v[cnt] = V;
    c[cnt] = C;
    nt[cnt] = ind[a];
    ind[a] = cnt;
    return cnt++;
}

int ADD(int a, int b, int c, int v)
{
    int x = add(a, b, c, v);
    int y = add(b, a, 0, -v);
    op[x] = y; op[y] = x;
    return x;
}

int h[N], q[N];

bool spfa()
{
    memset(vis, 0, sizeof vis);
    for(int i = S; i <= T; i++) f[i] = INF;
    for(int i = S; i <= T; i++) pre[i] = -1;
    pre[S] = -1;
    f[S] = 0;
    int l = 0, r = 0; q[l] = S; vis[S] = true;
    while(l <= r)
```

```
    {
        int x = q[l % N];
        l++; vis[x] = false;
        for(int k = ind[x]; k != -1; k = nt[k])
        if(c[k] > 0 && f[t[k]] > f[x] + v[k])
        {
            f[t[k]] = f[x] + v[k];
            pre[t[k]] = k;
            if(!vis[t[k]])
            {
                r++;
                q[r % N] = t[k];
                vis[t[k]] = true;
            }
        }
    }
    return pre[T] != -1;
}

int dinic()
{
    LL ans = 0, tmp;
    while(spfa())
    {
        ans += (LL)f[T] * dfs();
    }
    return ans;
}
```

2.6 最小路径覆盖最小点覆盖最大独立集

2.6.1 定义

对于图 G=(V,E) 中的一个点覆盖是一个集合 S V 使得每一条边至少有一个端点在 S 中

2.6.2 算法

对于任意边 (i, j)，建立 $(i, j'), (j, i')$ 两条边求最大匹配，最后答案为 ans / 2 + ans mod 2

2.6.3 其他问题

对于二分图

最小路径覆盖 = | P | - 最大匹配数

最小点覆盖数 = 最大匹配数

最大独立顶点集 = 总顶点数 - 最大匹配数

2.7 匈牙利算法

kuangbin 模板，边方向为 u 到 v, uN 为 u 类节点个数, vN 为 v 类节点个数

```cpp

const int MAXN=1000;
int uN, vN;
int g[MAXN][MAXN];
int linker[MAXN];
bool used[MAXN];
bool dfs(int u)
{
    int v;
    for(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;
    int u;
    memset(linker, -1, sizeof(linker));
    for(u = 0; u < uN; u++)
    {
        memset(used, 0, sizeof(used));
        if(dfs(u)) res++;
    }
    return res;
}
```

3 数据结构

3.1 LCA

```cpp
const int POW = 18;
void dfs(int u,int fa){
    d[u]=d[fa]+1;
    p[u][0]=fa;
    for(int i=1;i<POW;i++) p[u][i]=p[p[u][i-1]][i-1];
    int sz=edge[u].size();
    for(int i=0;i<sz;i++){
        int v=edge[u][i];
        if(v==fa) continue;
        dfs(v,u);
    }
}
int lca( int a, int b ){
    if( d[a] > d[b] ) a ^= b, b ^= a, a ^= b;
    if( d[a] < d[b] ){
```

```
        int del = d[b] - d[a];
        for( int i = 0; i < POW; i++ ) if(del&(1<<i)) b=p[b][i];
    }
    if( a != b ){
        for( int i = POW-1; i >= 0; i-- )
            if( p[a][i] != p[b][i] )
                a = p[a][i] , b = p[b][i];
        a = p[a][0], b = p[b][0];
    }
    return a;
}
```

3.2 RMQ

```
int f[N][POW];
for(int j = 0; j < POW; j++)
    for(int i = 0; i < n; i++)
    if(i + (1 << (j + 1)) <= n)
    {
        f[i][j + 1] = max(f[i][j], f[i + (1 << j)][j]);
    }
int k = log2(r - l + 1);
ans = max(ans, max(f[l][k], f[r - qpow(2, k) + 1][k]));
```

3.3 树链剖分

siz[v] 表示以 v 为根的子树的节点数 dep[v] 表示 v 的深度 top[v] 表示 v 所在的重链的顶端节点 fa[v] 表示 v 的父亲 son[v] 表示与 v 在同一重链上的 v 的儿子节点 w[v] 表示 v 与其父亲节点的连边在线段树中的位置初始需要调用 cnt1 = cnt2 = cnt3 = 0; dfs1(ROOT, 0); dfs2(ROOT, 1); bt(1, cnt2); 模板为边带权值，点带权值需要修改 query(x, y); update(x, p, c) 的 p 为线段树中的编号，更新 x 需要调用 w[x]

```
#define MID(x, y) (((x) + (y)) >> 1)

int fa[N], top[N], w[N], son[N], dep[N], sz[N], r[N];
int a[N], b[N];
LL c[N];
int ind[N];
int t[M], nt[M];
int cnt1, cnt2, cnt3;
int n, m;

struct node
{
    int l, r;
    int a, b;
    LL sum;
}f[M];
int rt;

void dfs1(int x, int d)
{
    dep[x] = d;
    son[x] = 0;
    sz[x] = 1;
    for(int k = ind[x]; k != -1; k = nt[k])
```

```
    if(t[k] != fa[x])
    {
        fa[t[k]] = x;
        dfs1(t[k], d + 1);
        sz[x] += sz[t[k]];
        if(sz[t[k]] > sz[son[x]]) son[x] = t[k];
    }
}

void dfs2(int x, int tt)
{
    w[x] = ++cnt2;
    top[x] = tt;
    if(son[x]) dfs2(son[x], tt);
    for(int k = ind[x]; k != -1; k = nt[k]) if(t[k] != fa[x] && t[k] !=
       son[x])
        dfs2(t[k], t[k]);
}

LL add(int a, int b)
{
    t[cnt1] = b;
    nt[cnt1] = ind[a];
    ind[a] = cnt1++;
}

void update(int x)
{
    f[x].sum = f[f[x].l].sum + f[f[x].r].sum;
}

int bt(int a, int b)
{
    int x = cnt3++;
    f[x].a = a; f[x].b = b;
    if(a < b)
    {
        int mid = MID(a, b);
        f[x].l = bt(a, mid);
        f[x].r = bt(mid + 1, b);
        f[x].sum = 0;
    }
    else
    {
        f[x].sum = 0;
    }
    return x;
}

// Query On ST, Do not Call Directly
LL query(int x, int a, int b)
{
    if(a <= f[x].a && f[x].b <= b) return f[x].sum;
    int mid = MID(f[x].a, f[x].b);
```

```
    LL ans = 0;
    if(a <= mid) ans += query(f[x].l, a, b);
    if(b > mid) ans += query(f[x].r, a, b);
    return ans;
}

//Modify Point
void update(int x, int p, int cc)
{
    if(f[x].a == f[x].b) { f[x].sum = cc; return; }
    int mid = MID(f[x].a, f[x].b);
    if(p <= mid) update(f[x].l, p, cc);
    else update(f[x].r, p, cc);
    update(x);
}

//Query Segment
LL query(int x, int y)
{
    int fx = top[x], fy = top[y];
    LL sum = 0;
    while(fx != fy)
    {
        if(dep[fx] < dep[fy])
        {
            swap(x, y);
            swap(fx, fy);
        }
        sum += query(rt, w[fx], w[x]);
        x = fa[top[x]];
        fx = top[x];
    }
    if(dep[x] > dep[y]) swap(x, y);
    if(x == y) return sum;
    return sum + query(rt, w[son[x]], w[y]);
}
```

3.4 区间第 k 大

静态区间第 k 大模板，每次查询 [l, r] 区间

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
#define N 100010
#define NN (30 * N)
namespace ST
{
    int T[N];
    struct node
    {
```

```
        int l, r, a, b;
        int c;
        node() {}
    }f[NN];
    int cnt;
    int lb, ub;

    int add(int a, int b, int l, int r, int c)
    {
        f[cnt].a = a; f[cnt].b = b; f[cnt].l = l; f[cnt].r = r; f[cnt].c
           = c;
        return cnt++;
    }

    int build(int a, int b)
    {
        int rt = add(a, b, -1, -1, 0);
        if(a < b)
        {
            int mid = (a + b) >> 1;
            f[rt].l = build(a, mid);
            f[rt].r = build(mid + 1, b);
        }
        return rt;
    }

    void init(int a, int b) { cnt = 0; T[b + 1] = build(a, b); lb = a; ub
        = b; }

    int update(int x, int p)
    {
        int rt = add(lb, ub, -1, -1, f[x].c + 1), tmp = rt;
        int l = lb, r = ub;
        while(l < r)
        {
            int mid = (l + r) >> 1;
            if(p <= mid)
            {
                f[rt].l = add(l, mid, -1, -1, 0);
                f[rt].r = f[x].r;
                rt = f[rt].l; x = f[x].l;
                r = mid;
            }
            else
            {
                f[rt].r = add(mid + 1, r, -1, -1, 0);
                f[rt].l = f[x].l;
                rt = f[rt].r; x = f[x].r;
                l = mid + 1;
            }
            f[rt].c = f[x].c + 1;
        }
        return tmp;
    }
```

```

    int qry(int lx, int rx, int k)
    {
        int l = lb, r = ub;
        while(l < r)
        {
            int mid = (l + r) >> 1;
            if(f[f[lx].l].c - f[f[rx].l].c >= k)
            {
                rx = f[rx].l; lx = f[lx].l;
                r = mid;
            }
            else
            {
                k -= (f[f[lx].l].c - f[f[rx].l].c);
                rx = f[rx].r; lx = f[lx].r;
                l = mid + 1;
            }
        }
        return l;
    }

    int q(int l, int r, int k)
    {
        return qry(T[l], T[r + 1], k);
    }
}

int n, m;
int h[N], a[N];

int main()
{
  scanf("%d%d", &n, &m);
    for(int i = 0; i < n; i++) scanf("%d", &a[i]);
    memcpy(h, a, sizeof a);
    sort(h, h + n);
    int cnt = unique(h, h + n) - h;
    for(int i = 0; i < n; i++) a[i] = lower_bound(h, h + n, a[i]) - h;

    ST::init(0, cnt - 1);
    for(int i = n - 1; i > -1; i--) ST::T[i] = ST::update(ST::T[i + 1], a[i]);
    while(m--)
    {
        int i, j, k; scanf("%d%d%d", &i, &j, &k);
        printf("%d\n", h[ST::q(i - 1, j - 1, k)]);
    }
    return 0;
}
```

3.5 二维线段树

3.5.1 点修改区间查询

```
int c[N][N];
int lx[N], ly[N];
int n, Q;
struct Node
{
    struct node
    {
        int a, b;
        int Min, Max;
    }f[N];
    int a, b;

    void update(int x)
    {
        f[x].Min = min(f[x << 1].Min, f[x << 1 | 1].Min);
        f[x].Max = max(f[x << 1].Max, f[x << 1 | 1].Max);
    }

    void build(int x, int a, int b)
    {
        f[x].a = a; f[x].b = b;
        if(a < b)
        {
            int mid = (a + b) / 2;
            build(x << 1, a, mid);
            build(x << 1 | 1, mid + 1, b);
            update(x);
        } else { f[x].Min = f[x].Max = 0; ly[a] = x; }
    }

    int queryMin(int x, int a, int b)
    {
        if(a <= f[x].a && f[x].b <= b) return f[x].Min;
        else
        {
            int mid = (f[x].a + f[x].b) / 2;
            if(b <= mid) return queryMin(x << 1, a, b);
            else if(a > mid) return queryMin(x << 1 | 1, a, b);
            else return min(queryMin(x << 1, a, b), queryMin(x << 1 | 1,
               a, b));
        }
    }

    int queryMax(int x, int a, int b)
    {
        if(a <= f[x].a && f[x].b <= b) return f[x].Max;
        else
        {
            int mid = (f[x].a + f[x].b) / 2;
            if(b <= mid) return queryMax(x << 1, a, b);
            else if(a > mid) return queryMax(x << 1 | 1, a, b);
            else return max(queryMax(x << 1, a, b), queryMax(x << 1 | 1,
               a, b));
```

```
        }
    }
}f[N];

void build(int x, int a, int b, int p, int q)
{
    f[x].a = a; f[x].b = b;
    f[x].build(1, p, q);
    if(a < b)
    {
        int mid = (a + b) / 2;
        build(x << 1, a, mid, p, q);
        build(x << 1 | 1, mid + 1, b, p, q);
    }
    else lx[a] = x;
}

void update(int x, int y, int c)
{
    x = lx[x]; y = ly[y];
    for(int X = x; X != 0; X >>= 1)
        for(int Y = y; Y != 0; Y >>= 1)
        {
            if(f[X].f[Y].a == f[X].f[Y].b)
            {
                if(f[X].a == f[X].b)
                {
                    f[X].f[Y].Min = f[X].f[Y].Max = c;
                }
                else
                {
                    f[X].f[Y].Min = min(f[X << 1].f[Y].Min, f[X << 1 |
                       1].f[Y].Min);
                    f[X].f[Y].Max = max(f[X << 1].f[Y].Max, f[X << 1 |
                       1].f[Y].Max);
                }
            }
            else { f[X].update(Y); }
        }
}

int queryMin(int x, int a, int b, int p, int q)
{
    if(a <= f[x].a && f[x].b <= b) return f[x].queryMin(1, p, q);
    else
    {
        int mid = (f[x].a + f[x].b) / 2;
        if(b <= mid) return queryMin(x << 1, a, b, p, q);
        else if(a > mid) return queryMin(x << 1 | 1, a, b, p, q);
        else return min(queryMin(x << 1, a, b, p, q), queryMin(x << 1 |
           1, a, b, p, q));
    }
}

```

```
int queryMax(int x, int a, int b, int p, int q)
{
    if(a <= f[x].a && f[x].b <= b) return f[x].queryMax(1, p, q);
    else
    {
        int mid = (f[x].a + f[x].b) / 2;
        if(b <= mid) return queryMax(x << 1, a, b, p, q);
        else if(a > mid) return queryMax(x << 1 | 1, a, b, p, q);
        else return max(queryMax(x << 1, a, b, p, q), queryMax(x << 1 | 1, a, b, p, q));
    }
}
```

4 字符串

4.1 KMP

```
void getNext()
{
  int j,k;
    j = 0;
    k = -1;
    next[0] = -1;
  while(j < m)
    {
    if(k == -1 || b[j] == b[k])
        next[++j] = ++k;
    else k = next[k];
    }
}
// get First Position
int KMP_Index()
{
  int i = 0, j = 0;
    getNext();
    while(i < n && j < m)
    {
    if(j == -1||a[i] == b[j])
        {
            i++;
            j++;
        }
        else j = next[j];
    }
    if(j == m) return i - m + 1;
  else return -1;
}
```

4.2 RKHash

把字符串变成 X 进制数，可以完成 O(1) 比较，有一定概率冲突，调用 init() 初始化幂，mh(0) 构造 hash 数组

```
char s[N];
ULL hash[N];
int len;
ULL pp[N];

void init()
{
    pp[0] = 1;
    for(int i = 1; i < N; i++) pp[i] = (pp[i - 1] * 31ULL) % MOD;
}

void mh(int p)
{
    if(!p) { hash[0] = (s[0] - 'a'); p++; }
    for(int i = p; i < len; i++) hash[i] = ((hash[i - 1] * 31ULL) % MOD + (s[i] - 'a')) % MOD;
}

int cp(int x, int y, int l)
{
    if(x + l > len || y + l > len) return false;
    x--; y--;
    if((hash[x + l] - hash[x] * pp[l] % MOD + MOD) % MOD
  == (hash[y + l] - hash[y] * pp[l] % MOD + MOD) % MOD) return true;
    return false;
}
```

4.3 Manacher

求以 i 为中心的最长回文串长度，结果保存在 $pk[i]$ 中，下标 $[0, n)$ 开头、末尾、字符之间插入 #，例如 ababa 变为 #a#b#a#b#a#

```
int pk[N];
void manacher()
{
    int mx = 0;
    int p;
    for(int i = 0; i < n; i++)
    {
        if(i < mx) pk[i] = min(pk[2 * p - i], mx - i);
        else pk[i] = 1;
        while(i + pk[i] < n && i - pk[i] > -1 && a[i + pk[i]] == a[i - pk[i]]) pk[i]++;
        if(i + pk[i] > mx) { p = i; mx = i + pk[i]; }
    }
}
```

4.4 SA

倍增算法，r 为待匹配数组，n 为总长度 +1，m 为字符范围，num 保存字符串使用时注意 num[] 有效位为 [0, n)，但是需要将 num[n] = 0，另外，对于模板的处理将空串也处理了，作为

rank 最小的串, 因此有效串为 [0, n] 共 n-1 个, 在调用 da() 函数时, 需要调用 da(num, n + 1, m) 对于 sa[], rank[], height[] 数组都将空串考虑在内, 作为 rank 最小的后缀注意 rank, height 范围从 [0, n]

```

namespace SA
{
    int len;
    int num[N];
    int sa[N], rank[N], height[N]; // sa[1~n]value(0~n-1); rank[0..n-1]
       value(1..n); height[2..n]
    int wa[N], wb[N], wv[N], wd[N];

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

    void da(int *r, int n, int m) {
        int i, j, k, p, *x = wa, *y = wb, *t;
        for(i = 0; i < m; i++) wd[i] = 0;
     for(i = 0; i < n; i++) wd[x[i] = r[i]]++;
        for(i = 1; i < m; i++) wd[i] += wd[i - 1];
        for(i = n - 1; i >= 0; i--) sa[--wd[x[i]]] = i;
        for(j = 1, p = 1; p < n; j <<= 1, m = p) {
            for(p = 0, i = n - j; i < n; i++) y[p++] = i;
            for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;
            for(i = 0; i < n; i++) wv[i] = x[y[i]];
            for(i = 0; i < m; i++) wd[i] = 0;
            for(i = 0; i < n; i++) wd[wv[i]]++;
            for(i = 1; i < m; i++) wd[i] += wd[i - 1];
            for(i = n - 1; i >= 0; i--) sa[--wd[wv[i]]] = y[i];
            for(t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i
               ++) {
                x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
            }
        }

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

4.5 最小表示法

求一个字符串字典序最小的开头位置，注意要复制串两次

```
int smallest(char *s)
{
    int i = 0,j = 1,k = 0;
    int len = strlen(s);
    while(i<len&&j<len)
    {
```

```
        k=0;
        while(k<len&&(s[i+k]==s[j+k]))
            k++;
        if(k>=len)
            break;
        if(s[i+k]>s[j+k])
            i=max(i+k+1,j+1);
        else
            j=max(i+1,j+k+1);
    }
    return min(i,j);
}

int biggest(char *s)
{
    int i = 0,j = 1,k = 0;
    int len = strlen(s);
    while(i<len&&j<len)
    {
        k=0;
        while(k<len&&(s[i+k]==s[j+k]))
            k++;
        if(k>=len)
            break;
        if(s[i+k]<s[j+k])
            i=max(i+k+1,j+1);
        else
            j=max(i+1,j+k+1);
    }
    return min(i,j);
}
```

5 数论

5.1 欧拉函数

对正整数 n，欧拉函数是少于或等于 n 的数中与 n 互质的数的数目
P 是素数:
若 p 是 x 的约数，则 $E(x*p) = E(x)*p$
p 不是 x 的约数，则 $E(x*p) = E(x)*E(p) = E(x)*(p-1)$

```
int prime[N];
int phi[N];
bool is_prime[N];

void get_phi()
{
    int i, j, k;
    k = 0;
    for(i = 2; i < N; i++)
    {
        if(is_prime[i] == false)
        {
```

```
            prime[k++] = i;
            phi[i] = i - 1;
        }
        for(j = 0; j < k && i * prime[j] < N; j++)
        {
            is_prime[ 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 );
            }
        }
    }
}
```

5.2 线性逆元

inv[i] = (MOD - MOD / i) * inv[MOD % i] % MOD
适用于 MOD 是质数且 MOD > n 的情况，能够 O(n) 时间求出 $[1, n]$ 对模 MOD 的逆

5.3 高阶等差数列

$$\sum n = \frac{1}{2}n(n+1) \tag{1}$$

$$\sum n^2 = \frac{1}{6}n(n+1)(2n+1) \tag{2}$$

$$\sum n^3 = \left(\sum n\right)^2 \tag{3}$$

$$\sum n^4 = \left(\sum n^2\right)\frac{1}{5}(3n^2+3n-1) \tag{4}$$

$$\sum n^5 = \left(\sum n\right)^2\frac{1}{3}(2n^2+2n-1) \tag{5}$$

$$\sum n^6 = \left(\sum n^2\right)\frac{1}{7}(3n^4+6n^3-3n+1) \tag{6}$$

$$\sum n^7 = \left(\sum n\right)^2\frac{1}{6}(3n^4+6n^3-n^2-4n+2) \tag{7}$$

$$\sum n^8 = \left(\sum n^2\right)\frac{1}{15}(5n^6+15n^5+5n^4-15n^3-n^2+9n-3) \tag{8}$$

6 数表

6.1 数据范围

类型	字节	最小值	最大值
int	4	-2147483648	2147483647
unsigned	4	0	4294967295
long long	8	-9223372036854775808	9223372036854775807
unsigned long long	8	0	18446744073709551615
float	4	1.17549e-038	3.40282e+038
double	8	2.22507e-308	1.79769e+308
long double	12	3.3621e-4932	1.18973e+4932

Table 1: 数据范围

6.2 时间复杂度

x	log2(x)
10	3
1e2	6
1e3	9
1e4	13
1e5	16
1e6	19
1e7	23
1e8	26
1e9	29

Table 2: log2(x)

n	n!
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Table 3: 阶乘