算法模版

- $Author$： 缪语博

本文档基于 $GPL-3.0License$

本文档 $GitHub$ 储存库：model

目录 $Contents$

- 基本格式

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    freopen(".in", "r", stdin);
    freopen(".out", "w", stdout);

    return 0;
}
```

- 很好用的宏定义和函数

```
#define ll long long
#define N 1000010
```

```
// 二分实现
#define MID ((l + r) >> 1)
// 线段树实现
#define PII pair<int, int>
#define MAX(x, y, z) max((x), max((y), (z)))
#define PRI priority_queue
//此处注意到底是几个零
#define MOD 1000000007
#define FI first
#define SE second
#define IL inline
#define RE register
#define MINN -0x7fffffff
#define MAXX 0x7fffffff
#define HMINN -0x3f3f3f3f
#define HMAXX 0x3f3f3f3f
#define ENDL putchar('\n')

int nxt[N], head[N], to[N], to[N];
inline void add_edge(int u, int v, ll c) {
    nxt[++cnt] = head[u];
    head[u] = cnt;
    to[cnt] = v;
    w[cnt] = c;
}
// 遍历:
for(int i = head[p]; i; i = nxt[i]) {
    int v = to[i];
    // Write what you want to write.
}


inline void madd(ll &a, ll b) {
    a += b;
    a %= MOD;
}
```

快读快写

```
inline int read() {
    int x = 0, f = 1;
    char ch = getchar();
    while(ch < '0' || ch > '9') {
        if(ch == '-') {
            f = -1;
        }
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + (ch ^ 48);
        ch = getchar();
    }
    return x * f;
```

```
}
inline void write(int x) {
    if(x < 0) {
        putchar('-');
        x = -x;
    }
    if(x > 9)
        write(x / 10);
    putchar(x % 10 + '0');
}
inline string stringread() {
    string str = "";
    char ch = getchar();
    while(ch == ' ' || ch == '\n' || ch == '\r') {
        ch = getchar();
    }
    while(ch != ' ' && ch != '\n' && ch != '\r') {
        str += ch;
        ch = getchar();
    }
    return str;
}
inline void stringwrite(string str) {
    for(int i = 0; str[i] != '\0'; ++i)
        putchar(str[i]);
}
```

线段树

```
#include <bits/stdc++.h>

#define N 100010
#define ll long long
#define ls (p << 1)
#define rs (p << 1 | 1)
#define mid ((l + r) >> 1)

using namespace std;

int n, m;
int a[N];
ll tree[N << 2];
int siz[N << 2];
int lazy[N << 2];

void upd(int p) {
    tree[p] = tree[ls] + tree[rs];
}

void upds(int p) {
    siz[p] = siz[ls] + siz[rs];
}
```

```
void pushd(int p) {
    tree[ls] += lazy[p] * siz[ls];
    tree[rs] += lazy[p] * siz[rs];

    lazy[ls] += lazy[p];
    lazy[rs] += lazy[p];

    lazy[p] = 0;
}

void build(int p, int l, int r) {

    lazy[p] = 0;

    if(l == r)  {
        siz[p] = 1;
        tree[p] = a[l];
        return ;
    }

    build(ls, l, mid);
    build(rs, mid + 1, r);

    upd(p);
    upds(p);
}

void mdf(int p, int l, int r, int ql, int qr, int k) {
    if(ql <= l && r <= qr) {
        tree[p] += 1ll * siz[p] * k;
        lazy[p] += k;
        return;
    }
    pushd(p);
    if(ql <= mid) {
        mdf(ls, l, mid, ql, qr, k);
    }
    if(qr > mid) {
        mdf(rs, mid + 1, r, ql, qr, k);
    }
    upd(p);
}

ll qry(int p, int l, int r, int ql, int qr) {
    if(ql <= l && r <= qr) {
        return tree[p];
    }
    pushd(p);

    ll sum = 0;

    if(ql <= mid) {
        sum += qry(ls, l, mid, ql, qr);
    }
    if(qr > mid) {
        sum += qry(rs, mid + 1, r, ql, qr);
```

```
    }

    return sum;
}

int main() {

    cin >> n >> m;

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

    build(1, 1, n);

    while(m--) {
        int op, x, y, k;

        cin >> op >> x >> y;

        if(op == 1) {
            cin >> k;
            mdf(1, 1, n, x, y, k);
        }
        else {
            cout << qry(1, 1, n, x, y) << endl;
        }
    }

    return 0;
}
```

中国剩余定理（拓展）

```
#include <bits/stdc++.h>

#define N 100010
#define ll long long

using namespace std;

ll n;
ll a[N], b[N];

ll gcd(ll a, ll b) {
    if(!b)
        return a;
    else return gcd(b, a % b);
}

inline ll read() {
    ll x = 0, f = 1;
    char ch = getchar();
```

```
    while(ch < '0' || ch > '9') {
        if(ch == '-') {
            f = -1;
        }
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + (ch ^ 48);
        ch = getchar();
    }
    return x * f;
}

void merge(ll p1, ll a1, ll p2, ll a2, ll &p, ll &a) {
    p = p1 / gcd(p1, p2) * p2;
    if(p1 < p2) {
        swap(p1, p2);
        swap(a1, a2);
    }
    a = a1;
    while(a % p2 != a2)
        a += p1;
}
int main() {

    n = read();

    for(int i = 1; i <= n; ++i) {
        a[i] = read();
        b[i] = read();
        b[i] %= a[i];
    }

    for(int i = 2; i <= n; ++i) {
        merge(a[i], b[i], a[i - 1], b[i - 1], a[i], b[i]);
    }

    cout << b[n] << endl;

    return 0;
}
```

最短路算法

Floyed 算法

```
#include <bits/stdc++.h>
using namespace std;

int n, m;
int f[101][101];
int u, v, w;
```

```
int main() {

    cin >> n >> m;

    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= n; ++j) {
            f[i][j] = 0x3f3f3f3f;
            if(i == j) {
                f[i][j] = 0;
            }
        }
    }

    for(int i = 1; i <= m; ++i) {
        cin >> u >> v >> w;
        f[u][v] = min(f[u][v], w);
        f[v][u] = f[u][v];
    }

    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][j], f[i][k] + f[k][j]);

    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= n; ++j) {
            cout << f[i][j] << ' ';
        }
        cout << endl;
    }
    return 0;
}
```

Dijkstra 算法

```
#include <bits/stdc++.h>
using namespace std;

int n, m, s;
struct Edge {
    int v, c;
}tmp;
vector<Edge> edges[100005];
int dis[100005];
bool vis[100005];
priority_queue<pair<int, int> , vector<pair<int, int> >, greater<pair<int, int> > > pq;

void dijkstra() {
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, false, sizeof(vis));
    dis[s] = 0;
    pq.push(make_pair(0, s));
```

```
    while(!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        if(vis[u])
            continue;
        vis[u] = true;

        for(Edge e : edges[u]) {
            if(dis[e.v] > dis[u] + e.c) {
                dis[e.v] = dis[u] + e.c;
                pq.push(make_pair(dis[e.v], e.v));
            }
        }
    }
}

int main() {

    cin >> n >> m >> s;

    for(int i = 1; i <= m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        tmp.c = w;
        tmp.v = v;
        edges[u].push_back(tmp);
    }

    dijkstra();

    for(int i = 1; i <= n; ++i)
        cout << (dis[i] == 0x3f3f3f3f ? 2147483647 : dis[i]) << ' ';
    return 0;
}
```

$Bellman-Ford$ 算法

```
#include <bits/stdc++.h>

#define MAXN 100005

using namespace std;

int n, m, s;

struct Edge {
    int v;
    int c;
};

vector<Edge> edges[MAXN];
int dis[MAXN];

inline int read() {
```

```
    int x = 0, f = 1;
    char ch = 0;
    while(ch < '0' || ch > '9') {
        if(ch == '-') {
            f = -1;
        }
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + (ch ^ 48);
        ch = getchar();
    }
    return x * f;
}

inline void write(int x) {
    if(x < 0) {
        putchar('-');
        x = -x;
    }
    if(x > 9)
        write(x / 10);
    putchar(x % 10 + '0');
}

void bellman_ford() {

    memset(dis, 0x3f, sizeof(dis));

    dis[s] = 0;

    bool flag;

    for(int i = 1; i <= n - 1; ++i) {
        flag = false;
        for(int u = 1; u <= n; ++u) {
            for(Edge e : edges[u]) {
                if(dis[e.v] > dis[u] + e.c) {
                    dis[e.v] = dis[u] + e.c;
                    flag = true;
                }
            }
        }
        if(!flag)
            break;
    }
}

int main() {

    n = read();
    m = read();
    s = read();

    for(int i = 1; i <= m; ++i) {
        int u, v, c;
```

```
        u = read();
        v = read();
        c = read();
        Edge tmp;
        tmp.c = c;
        tmp.v = v;
        edges[u].push_back(tmp);
    }

    bellman_ford();

    for(int i = 1; i <= n; ++i) {
        write((dis[i] == 0x3f3f3f3f ? 2147483647 : dis[i]));
        putchar(' ');
    }
    putchar('\n');
    return 0;
}
```

LCA 最近公共祖先

倍增写法：

```
#include <bits/stdc++.h>

#define N 500005
#define ll long long
#define M 31

using namespace std;

int n, m, s;
int lg[N];
int xx, yy;
vector<int> edge[N];
int dep[N];
int fa[N][M];

void add_edge(int u, int v) {
    edge[u].push_back(v);
    edge[v].push_back(u);
}

void dfs(int p, int pre) {
    fa[p][0] = pre;
    dep[p] = dep[pre] + 1;
    for(int i = 1; i <= lg[dep[p]]; ++i) {
        fa[p][i] = fa[fa[p][i - 1]][i - 1];
    }
    for(int nxt : edge[p]) {
        if(nxt != pre) {
            dfs(nxt, p);
```

```
        }
    }
}

int lca(int u, int v) {
    if(dep[u] < dep[v]) {
        swap(u, v);
    }
    while(dep[u] > dep[v]) {
        u = fa[u][lg[dep[u] - dep[v]] - 1];
    }
    if(u == v) {
        return u;
    }
    for(int llg = lg[dep[u]] - 1; llg >= 0; --llg) {
        if(fa[u][llg] != fa[v][llg]) {
            u = fa[u][llg];
            v = fa[v][llg];
        }
    }

    return fa[u][0];
}

int main() {

    cin >> n >> m >> s;

    for(int i = 1; i <= n - 1; ++i) {
        cin >> xx >> yy;
        add_edge(xx, yy);
    }

    int cntt = 0;

    for(int i = 1; i <= n; ++i) {
        if(i >= (1 << cntt)) {
            ++cntt;
            lg[i] = cntt;
        }
        else {
            lg[i] = cntt;
        }
    }

    dfs(s, 0);

    while(m--) {
        cin >> xx >> yy;
        cout << lca(xx, yy) << endl;
    }

    return 0;
}
```

高精度

```
int compare(string str1,string str2)
{
    if(str1.length()>str2.length()) return 1;
    else if(str1.length()<str2.length())  return -1;
    else return str1.compare(str2);
}

//高精度加法
//只能是两个正数相加
string add(string str1,string str2)
{
    string str;
    int len1=str1.length();
    int len2=str2.length();
    //前面补0，弄成长度相同
    if(len1<len2)
    {
        for(int i=1;i<=len2-len1;i++)
           str1="0"+str1;
    }
    else
    {
        for(int i=1;i<=len1-len2;i++)
           str2="0"+str2;
    }
    len1=str1.length();
    int cf=0;
    int temp;
    for(int i=len1-1;i>=0;i--)
    {
        temp=str1[i]-'0'+str2[i]-'0'+cf;
        cf=temp/10;
        temp%=10;
        str=char(temp+'0')+str;
    }
    if(cf!=0)  str=char(cf+'0')+str;
    return str;
}

//高精度减法
//只能是两个正数相减，而且要大减小
string sub(string str1,string str2)
{
    string str;
    int tmp=str1.length()-str2.length();
    int cf=0;
    for(int i=str2.length()-1;i>=0;i--)
    {
        if(str1[tmp+i]<str2[i]+cf)
        {
            str=char(str1[tmp+i]-str2[i]-cf+'0'+10)+str;
            cf=1;
```

```
        }
        else
        {
            str=char(str1[tmp+i]-str2[i]-cf+'0')+str;
            cf=0;
        }
    }
    for(int i=tmp-1;i>=0;i--)
    {
        if(str1[i]-cf>='0')
        {
            str=char(str1[i]-cf)+str;
            cf=0;
        }
        else
        {
            str=char(str1[i]-cf+10)+str;
            cf=1;
        }
    }
    str.erase(0,str.find_first_not_of('0'));//去除结果中多余的前导0
    return str;
}

//高精度乘法
//只能是两个正数相乘
string mul(string str1,string str2)
{
    string str;
    int len1=str1.length();
    int len2=str2.length();
    string tempstr;
    for(int i=len2-1;i>=0;i--)
    {
        tempstr="";
        int temp=str2[i]-'0';
        int t=0;
        int cf=0;
        if(temp!=0)
        {
            for(int j=1;j<=len2-1-i;j++)
              tempstr+="0";
            for(int j=len1-1;j>=0;j--)
            {
                t=(temp*(str1[j]-'0')+cf)%10;
                cf=(temp*(str1[j]-'0')+cf)/10;
                tempstr=char(t+'0')+tempstr;
            }
            if(cf!=0) tempstr=char(cf+'0')+tempstr;
        }
        str=add(str,tempstr);
    }
    str.erase(0,str.find_first_not_of('0'));
    return str;
}
```

```
//高精度除法
//两个正数相除，商为quotient,余数为residue
//需要高精度减法和乘法
void div(string str1,string str2,string &quotient,string &residue)
{
    quotient=residue="";//清空
    if(str2=="0")//判断除数是否为0
    {
        quotient=residue="ERROR";
        return;
    }
    if(str1=="0")//判断被除数是否为0
    {
        quotient=residue="0";
        return;
    }
    int res=compare(str1,str2);
    if(res<0)
    {
        quotient="0";
        residue=str1;
        return;
    }
    else if(res==0)
    {
        quotient="1";
        residue="0";
        return;
    }
    else
    {
        int len1=str1.length();
        int len2=str2.length();
        string tempstr;
        tempstr.append(str1,0,len2-1);
        for(int i=len2-1;i<len1;i++)
        {
            tempstr=tempstr+str1[i];
            tempstr.erase(0,tempstr.find_first_not_of('0'));
            if(tempstr.empty())
              tempstr="0";
            for(char ch='9';ch>='0';ch--)//试商
            {
                string str,tmp;
                str=str+ch;
                tmp=mul(str2,str);
                if(compare(tmp,tempstr)<=0)//试商成功
                {
                    quotient=quotient+ch;
                    tempstr=sub(tempstr,tmp);
                    break;
                }
            }
        }
        residue=tempstr;
    }
```

```
    quotient.erase(0,quotient.find_first_not_of('0'));
    if(quotient.empty()) quotient="0";
}
```

树链剖分

```
#include <bits/stdc++.h>

#define N 200010
#define ENDL putchar('\n')

#define mid ((l + r) >> 1)
#define ls (p << 1)
#define rs (p << 1 | 1)

using namespace std;

int n, m, r, MOD;
int nxt[N], head[N], to[N], w[N];
int top[N];
int si[N], fa[N], id[N], son[N], wt[N], dep[N];
int cnt = 0, e = 0;
int tree[N << 2], siz[N << 2], lazy[N << 2];

inline int read() {
    int x = 0, f = 1;
    char ch = getchar();
    while(ch < '0' || ch > '9') {
        if(ch == '-') {
            f = -1;
        }
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + (ch ^ 48);
        ch = getchar();
    }
    return x * f;
}

inline void write(int x) {
    if(x < 0) {
        putchar('-');
        x = -x;
    }
    if(x > 9)
        write(x / 10);
    putchar(x % 10 + '0');
}


inline void add(int &a, int b) {
    a = (a + b) % MOD;
```

```
}


inline void upd(int p) {
    tree[p] = (tree[ls] + tree[rs]) % MOD;
}

inline void upds(int p) {
    siz[p] = siz[ls] + siz[rs];
}

inline void pushd(int p) {
    if(!lazy[p])
        return;

    add(tree[ls], siz[ls] * lazy[p] % MOD);
    add(tree[rs], siz[rs] * lazy[p] % MOD);

    add(lazy[ls], lazy[p]);
    add(lazy[rs], lazy[p]);

    lazy[p] = 0;
}

inline void build(int p, int l, int r) {
    if(l == r) {
        tree[p] = wt[l];
        siz[p] = 1;
        lazy[p] = 0;
        return;
    }
    build(ls, l, mid);
    build(rs, mid + 1, r);
    upd(p);
    upds(p);
}

inline void mdf(int p, int l, int r, int ql, int qr, int x) {
    if(ql <= l && r <= qr) {
        add(tree[p], siz[p] * x % MOD);
        add(lazy[p], x);
        return;
    }
    pushd(p);
    if(ql <= mid) {
        mdf(ls, l, mid, ql, qr, x);
    }
    if(qr > mid) {
        mdf(rs, mid + 1, r, ql, qr, x);
    }
    upd(p);
}

inline int qry(int p, int l, int r, int ql, int qr) {
    if(ql <= l && r <= qr) {
        return tree[p];
```

```
    }
    pushd(p);
    int sum = 0;
    if(ql <= mid) {
        add(sum, qry(ls, l, mid, ql, qr));
    }
    if(qr > mid) {
        add(sum, qry(rs, mid + 1, r, ql, qr));
    }
    return sum;
}


inline void add_edge(int u, int v) {
    nxt[++e] = head[u];
    head[u] = e;
    to[e] = v;
}

inline void dfs1(int p, int pre, int depth) {
    dep[p] = depth;
    fa[p] = pre;
    si[p] = 1;
    int maxx = -1;

    for(int i = head[p]; i; i = nxt[i]) {
        int v = to[i];
        if(v != pre) {
            dfs1(v, p, depth + 1);
            si[p] +=  si[v];
            if(si[v] > maxx) {
                maxx = si[v];
                son[p] = v;
            }
        }
    }
}

inline void dfs2(int p, int topp) {
    id[p] = ++cnt;
    wt[cnt] = w[p];
    top[p] = topp;
    if(!son[p])
        return;
    dfs2(son[p], topp);
    for(int i = head[p]; i; i = nxt[i]) {
        int v = to[i];
        if(v != son[p] && v != fa[p]) {
            dfs2(v, v);
        }
    }
}


inline void modify1(int l, int r, int x) {
    x %= MOD;
```

```
    while(top[l] != top[r]) {
        if(dep[top[l]] < dep[top[r]]) {
            swap(l, r);
        }
        mdf(1, 1, n, id[top[l]], id[l], x);
        l = fa[top[l]];
    }
    if(dep[l] > dep[r])
        swap(l, r);
    mdf(1, 1, n, id[l], id[r], x);
}

inline int query1(int l, int r) {
    int sum = 0;
    while(top[l] != top[r]) {
        if(dep[top[l]] < dep[top[r]]) {
            swap(l, r);
        }
        add(sum, qry(1, 1, n, id[top[l]], id[l]));
        l = fa[top[l]];
    }
    if(dep[l] > dep[r])
        swap(l, r);
    add(sum, qry(1, 1, n, id[l], id[r]));
    return sum;
}

inline void modify2(int p, int x) {
    mdf(1, 1, n, id[p], id[p] + si[p] - 1, x);
}

inline int query2(int p) {
    return qry(1, 1, n, id[p], id[p] + si[p] - 1) % MOD;
}

int main() {

    n = read();
    m = read();
    r = read();
    MOD = read();

    for(int i = 1; i <= n; ++i) {
        w[i] = read();
    }

    for(int i = 1; i <= n - 1; ++i) {
        int u, v;
        u = read();
        v = read();
        add_edge(u, v);
        add_edge(v, u);
    }

    dfs1(r, 0, 1);
    dfs2(r, r);
```

```
    build(1, 1, n);

    while(m--) {
        int op;
        op = read();
        if(op == 1) {
            int x, y, z;
            x = read();
            y = read();
            z = read();
            modify1(x, y, z);
        }
        else if(op == 2) {
            int x, y;
            x = read();
            y = read();
            write(query1(x, y));
            ENDL;
        }
        else if(op == 3) {
            int x, z;
            x = read();
            z = read();
            modify2(x, z);
        }
        else {
            int x;
            x = read();
            write(query2(x));
            ENDL;
        }
    }

    return 0;
}
```

网络流

Dinic 最大流

```
#include <bits/stdc++.h>

#define N 500010

using namespace std;

long long nxt[N], to[N], head[N], w[N];
long long now[N];
int d[N];
long long n, m, s, t;
long long cnt = 1;
```

```cpp
inline void add(int u, int v, long long c) {
    nxt[++cnt] = head[u];
    head[u] = cnt;
    to[cnt] = v;
    w[cnt] = c;

    nxt[++cnt] = head[v];
    head[v] = cnt;
    to[cnt] = u;
    w[cnt] = 0;
}

inline bool bfs() {
    queue<int> q;
    q.push(s);
    memset(d, 0x3f, sizeof(d));
    d[s] = 0;
    now[s] = head[s];
    while(!q.empty()) {
        int x = q.front();
        q.pop();
        for(int i = head[x]; i; i = nxt[i]) {
            int v = to[i];
            if(d[v] == 0x3f3f3f3f && w[i] > 0) {
                d[v] = d[x] + 1;
                now[v] = head[v];
                q.push(v);
                if(v == t) {
                    return 1;
                }
            }
        }
    }
    return 0;
}

inline long long dfs(int x, long long flow) {
    if(x == t) {
        return flow;
    }
    long long k;
    for(int i = now[x]; i; i = nxt[i]) {
        now[x] = i;
        int v = to[i];
        if(d[v] != d[x] + 1 || w[i] <= 0) {
            continue;
        }
        k = dfs(v, min(flow, w[i]));
        if(k) {
            w[i] -= k;
            w[i ^ 1] += k;
            return k;
        }
        else {
            d[v] = 0x3f3f3f3f;
```

```
        }
    }
    return 0;
}

inline long long dinic() {
    long long sum = 0;
    while(bfs()) {
        sum += dfs(s, 0x7fffffff);
    }
    return sum;
}

int main() {

    cin >> n >> m >> s >> t;

    while(m--) {
        long long u, v, c;
        cin >> u >> v >> c;
        add(u, v, c);
    }

    cout << dinic() << endl;

    return 0;
}
```

KMP 算法

```
#include <bits/stdc++.h>

#define N 1000001

using namespace std;

char s1[N], s2[N];
int la, lb;
int j = 0;
int kmp[N];

int main() {
    cin >> s1 + 1;
    cin >> s2 + 1;

    la = strlen(s1 + 1);
    lb = strlen(s2 + 1);

    for(int i = 1; i <= lb; ++i) {
        while(j && s2[i + 1] != s2[j + 1]) {
            j = kmp[j];
        }
        if(s2[i + 1] == s2[j + 1]) {
```

```
            ++j;
            kmp[i + 1] = j;
        }
    }

    j = 0;

    for(int i = 1; i <= la; ++i) {
        while(j && s1[i] != s2[j + 1]) {
            j = kmp[j];
        }
        if(s1[i] == s2[j + 1]) {
            ++j;
        }
        if(j == lb) {
            cout << i - lb + 1 << endl;
            j = kmp[j];
        }
    }

    for(int i = 1; i <= lb; ++i) {
        cout << kmp[i] << ' ';
    }
    return 0;
}
```