省选基础算法

雷宇辰

2017 年 2 月 27 日

目录

1 day1 图论 **1**
1.1 有向图强连通分量的 Tarjan 算法 1
1.2 图的割点、桥与双连通分量 5
1.3 2-SAT 8
1.4 欧拉回路 14

2 day2 字符串（一） **16**
2.1 KMP 16
2.2 Trie 19
2.3 Aho-Corasick Automaton 21
2.4 Manacher 23

3 day3 简单数学 **24**
3.1 整除及剩余 24
3.2 素数 25
3.3 欧几里得算法 29
3.4 线性同余方程 30
3.5 逆元 31
3.6 离散对数问题 32
3.7 原根 34

4 day4 数据结构 **35**
4.1 树状数组 35
4.2 Sparse Table 38
4.3 左偏树 41
4.4 线段树 43

1 day1 图论

1.1 有向图强连通分量的 Tarjan 算法

定义 在**有向图** G 中，如果两个顶点 u, v 间存在一条路径 u 到 v 的路径且也存在一条 v 到 u 的路径，则称这两个顶点 u, v 是**强连通的 (strongly connected)**。如果有向图 G 的每两个顶点都强连通，称 G 是一个**强连通图**。有向非强连通图的极大强连通子图，称为**强连通分量 (strongly connected components)**。若将有向图中的强连通分量都缩为一个点，则原图会形成一个 DAG（有向无环图）。

极大强连通子图 G 是一个极大强连通子图当且仅当 G 是一个强连通子图且不存在另一个强连通子图 G' 使得 G 是 G' 的真子集。

Tarjan 算法 定义 $dfn(u)$ 为结点 u 搜索的次序编号，给出函数 $low(u)$ 使得

$low(u) = min$

{

$dfn(u)$,

$low(v)$, (u, v) 为树枝边，u 为 v 的父结点

$dfn(v)$ (u, v) 为后向边或指向栈中结点的横叉边

}

当结点 u 的搜索过程结束后，若 $dfn(u) = low(u)$，则以 u 为根的搜索子树上所有还在栈中的结点是一个强连通分量。

代码

tarjan - SCC

```
void tarjan(int u)
{
    dfn[u] = low[u] = ++idx;
    st[top++] = u;
    for (Edge cur : G[u])
        if (!dfn[cur.to])
            tarjan(cur.to),
            low[u] = min(low[u], low[cur.to]);
        else if (!scc[cur.to])
            low[u] = min(low[u], dfn[cur.to]);
    if (dfn[u] == low[u] && ++cnt)
        do scc[st[--top]] = cnt;
        while (st[top] != u);
}
```

练习题

POJ2186/BZOJ1051 - Popular Cows 双倍的快乐

Popular Cows

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
```

```
int head[10010], next[50010], to[50010], ecnt;
int dfn[10010], low[10010], stk[10010], scc[10010], top, idx, scccnt;
bool instk[10010];
int deg[10010];
inline void addEdge(int f, int t)
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
}
void tarjan(int x)
{
    dfn[x] = low[x] = ++idx;
    instk[stk[top++] = x] = true;
    for (int cur = head[x]; cur; cur = next[cur])
        if (!dfn[to[cur]])
            tarjan(to[cur]), low[x] = min(low[x], low[to[cur]]);
        else if (instk[to[cur]])
            low[x] = min(low[x], dfn[to[cur]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
        do
        {
            top--;
            scc[stk[top]] = scccnt;
            instk[stk[top]] = false;
        } while (stk[top] != x);
    }
}
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 0, x, y; i < m; i++)
    {
        scanf("%d%d", &x, &y);
        addEdge(x, y);
    }
    for (int i = 1; i <= n; i++)
        if (!dfn[i])
            tarjan(i);
    for (int i = 1; i <= n; i++)
        for (int cur = head[i]; cur; cur = next[cur])
            if (scc[i] != scc[to[cur]])
                deg[scc[i]]++;
    int zcnt = 0, id = 0;
    for (int i = 1; i <= scccnt; i++)
        if (deg[i] == 0)
            zcnt++, id = i;
    if (zcnt != 1)
```

```
        putchar('0');
    else
    {
        int ans = 0;
        for (int i = 1; i <= n; i++)
            if (scc[i] == id)
                ans++;
        printf("%d", ans);
    }
    return 0;
}
```

POJ3180 - The Cow Prom The $N(2 <= N <= 10,000)$ cows are so excited.

The Cow Prom

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
const int maxn = 100010;
int head[maxn], next[maxn << 1], to[maxn << 1], ecnt, n, m;
int dfn[maxn], scc[maxn], cnt[maxn], scccnt, stk[maxn], low[maxn], idx, top;
inline void addEdge(int f, int t)
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
}
void tarjan(int x)
{
    dfn[x] = low[x] = ++idx;
    stk[top++] = x;
    for (int i = head[x]; i; i = next[i])
        if (!dfn[to[i]])
            tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
        else if (!scc[to[i]])
            low[x] = min(low[x], dfn[to[i]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
        do
            scc[stk[--top]] = scccnt;
        while (stk[top] != x);
    }
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0, x, y; i < m; i++)
    {
        scanf("%d%d", &x, &y);
        addEdge(x, y);
```

```
    }
    for (int i = 1; i <= n; i++)
        if (!dfn[i]) tarjan(i);
    int ans = 0;
    for (int i = 1; i <= n; i++) cnt[scc[i]]++;
    for (int i = 1; i <= scccnt; i++)
        if (cnt[i] > 1) ans++;
    printf("%d", ans);
    return 0;
}
```

POJ1236 - Network of Schools 强连通分量缩点求出度为 0 的和入度为 0 的分量个数

Network of Schools

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
const int maxn = 110, maxm = 10100;
int head[maxn], next[maxm], to[maxm], ecnt, f[maxn], g[maxn];
inline void addEdge(int f, int t)
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
}
int dfn[maxn], low[maxn], stk[maxn], scc[maxn], scccnt, top, idx;
void tarjan(int x)
{
    dfn[x] = low[x] = ++idx;
    stk[top++] = x;
    for (int i = head[x]; i; i = next[i])
        if (!dfn[to[i]])
            tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
        else if (!scc[to[i]])
            low[x] = min(low[x], dfn[to[i]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
        do
            scc[stk[--top]] = scccnt;
        while (stk[top] != x);
    }
}
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1, x; i <= n; i++)
        for (scanf("%d", &x); x; scanf("%d", &x))
            addEdge(i, x);
    for (int i = 1; i <= n; i++)
```

```
        if (!dfn[i]) tarjan(i);
    for (int i = 1; i <= n; i++)
        for (int j = head[i]; j; j = next[j])
            if (scc[i] != scc[to[j]])
                f[scc[i]]++, g[scc[to[j]]]++;
    int ans1 = 0, ans2 = 0;
    if (scccnt == 1)
        printf("1\n0");
    else
    {
        for (int i = 1; i <= scccnt; i++)
            ans1 += f[i] == 0, ans2 += g[i] == 0;
        printf("%d\n%d", ans2, ans1 > ans2 ? ans1 : ans2);
    }
    return 0;
}
```

1.2 图的割点、桥与双连通分量

定义

点连通度与边连通度 在一个**无向连通图**中，如果有一个顶点集合 V，删除顶点集合 V，以及与 V 中顶点相连（至少有一端在 V 中）的所有边后，原图**不连通**，就称这个点集 V 为**割点集合**。
一个图的**点连通度**的定义为：最小割点集合中的顶点数。
类似的，如果有一个边集合，删除这个边集合以后，原图不连通，就称这个点集为**割边集合**。

双连通图、割点与桥 如果一个无向连通图的**点连通度大于** 1，则称该图是**点双连通的 (point biconnected)**，简称双连通或重连通。一个图有**割点**，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为**割点 (cut point)**，又叫关节点 (articulation point)。一个图可能有多个割点。
如果一个无向连通图的**边连通度大于** 1，则称该图是**边双连通的 (edge biconnected)**，简称双连通或重连通。一个图有**桥**，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥 (bridge)**，又叫关节边 (articulation edge)。一个图可能有多个桥。
可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。（但这并不意味着它们等价）
双连通分量（分支）：在图 G 的所有子图 G' 中，如果 G' 是双连通的，则称 G' 为双连通子图。如果一个双连通子图 G' 它不是任何一个双连通子图的真子集，则 G' 为极大双连通子图。双连通分量 (biconnected component)，或重连通分量，就是图的极大双连通子图。特殊的，点双连通分量又叫做块。

Tarjan 算法 给出函数 $low(u)$ 使得
$low(u) = min$
$\{$
$dfn(u),$
$low(v),$ (u, v) 为树枝边 (父子边)
$dfn(v)$ (u, v) 为后向边 (返祖边) 等价于 $dfn(v) < dfn(u)$ 且 v 不为 u 的父亲结点
$\}$

代码

tarjan - BCC

```
void tarjan(int u, int p)
{
    dfn[u] = low[u] = ++idx;
    for (int e = head[u]; e; e = next[e])
        if (!dfn[to[e]])
            tarjan(to[e], u), low[u] = min(low[u], low[to[e]]);
        else if (to[e] != p)
            low[u] = min(low[u], dfn[to[e]]);
}
```

练习题

POJ3177 - Redundant Paths 将一张有桥图通过加边变成边双连通图，至少要加 $\frac{leaf+1}{2}$ 条边。

Redundant Paths

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
int head[5010], to[20010], next[20010], ecnt, map[5010][5010];
int dfn[5010], low[5010], idx, cnt[5010];
void addEdge(int f, int t)
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
}
void tarjan(int u, int p)
{
    dfn[u] = low[u] = ++idx;
    for (int e = head[u]; e; e = next[e])
        if (!dfn[to[e]])
            tarjan(to[e], u), low[u] = min(low[u], low[to[e]]);
        else if (to[e] != p)
            low[u] = min(low[u], dfn[to[e]]);
}
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1, x, y; i <= m; i++)
    {
        scanf("%d%d", &x, &y);
        if (!map[x][y])
        {
            addEdge(x, y);
            addEdge(y, x);
            map[x][y] = map[y][x] = true;
```

```
        }
    }
    tarjan(1, 0);
    for (int i = 1; i <= n; i++)
        for (int e = head[i]; e; e = next[e])
            if (low[to[e]] != low[i])
                cnt[low[i]]++;
    int ans = 0;
    for (int i = 1; i <= n; i++)
        ans += cnt[i] == 1;
    printf("%d", (ans + 1) >> 1);
    return 0;
}
```

POJ1523 - SPF 求割点与删除这个点之后有多少个连通分量

Redundant Paths

```
#include <cstdio>
#include <cctype>
#include <cstring>
#define clz(X) memset(X, 0, sizeof(X))
inline int max(int a, int b) { return a > b ? a : b; }
inline int min(int a, int b) { return a < b ? a : b; }
inline void read(int &x)
{
    int ch = x = 0;
    while (!isdigit(ch = getchar()));
    for (; isdigit(ch); ch = getchar())
        x = x * 10 + ch - '0';
}
int map[1010][1010], range;
int dfn[1010], low[1010], idx;
int son, subnet[1010];
void tarjan(int u)
{
    dfn[u] = low[u] = ++idx;
    for (int v = 1; v <= range; v++)
        if (map[u][v])
            if (!dfn[v])
            {
                tarjan(v);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u])
                    (u == 1 ? son : subnet[u])++;
            }
            else
                low[u] = min(low[u], dfn[v]);
}
int main()
{
    int x, y, T = 0;
```

```
    while (read(x), x)
    {
        clz(map), clz(dfn), clz(low), clz(subnet), son = idx = 0;
        read(y);
        map[x][y] = map[y][x] = 1;
        range = max(x, y);
        while (read(x), x)
        {
            read(y);
            map[x][y] = map[y][x] = 1;
            range = max(range, max(x, y));
        }
        printf("Network #%d\n", ++T);
        tarjan(1);
        bool flag = false;
        if (son > 1) subnet[1] = son - 1;
        for (int i = 1; i <= range; i++)
            if (subnet[i])
                printf("  SPF node %d leaves %d subnets\n", i, subnet[i] + 1),
                flag = true;
        if (!flag)
            puts("  No SPF nodes");
        putchar('\n');
    }
    return 0;
}
```

POJ2942 - Knights of the Round Table 这题过于复杂，我来先给个别人的题解。然后是我自己的实现（仿佛还是没看懂。
//实现被狗吃了

1.3 2-SAT

定义 给定一个布尔方程，判断是否存在一组布尔变量的取值方案，使得整个方程值为真的问题，被称为布尔方程的可满足性问题 (SAT)。SAT 问题是 NP 完全的，但对于一些特殊形式的 SAT 问题我们可以有效求解。
我们将下面这种布尔方程称为合取范式：

$$(a \vee b \vee c \vee \cdots) \wedge (d \vee e \vee f \vee \cdots) \wedge \cdots$$

其中 $a, b, c, \cdots$ 称为文字，它是一个布尔变量或其否定。像 $(a \vee b \vee c \vee \cdots)$ 这样用 $\vee$ 连接的部分称为子句。如果合取范式的每个子句中的文字个数都不超过两个，那么对应的 SAT 问题又称为 **2-SAT** 问题。

解法 对于给定的 **2-SAT** 问题，首先利用 $\Rightarrow$ 将每个子句 $(a \vee b)$ 改写成等价形式 $(\neg a \Rightarrow b \wedge a \Rightarrow \neg b)$. 这样原布尔公式就变成了把 $a \Rightarrow b$ 形式的布尔公式用 $\wedge$ 连接起来的形式。
对每个布尔变量 x 构造两个顶点分别代表 x 与 $\neg x$。以 $\Rightarrow$ 关系为边建立有向图。若在此图中 a 点能到达 b 点，就表示 a 为真时 b 也一定为真。因此该图中同一个强连通分量中所含的所有变量的布尔值均相同。
若存在某个变量 x, 代表 x 与 $\neg x$ 的两个顶点在同一个强连通分量中，则原布尔表达式的值无法为真。
反之若不存在这样的变量，那么我们先将原图中所有的强连通分量缩为一个点，构出一个新图，新图显然

是一个拓扑图，我们求出它的一个拓扑序。那么对于每个变量 x，

$$x\text{所在的强连通分量（新图中的点）的拓扑序在}\neg x\text{所在的强连通分量之后} \Leftrightarrow x\text{为真}$$

就是一组合适布尔变量赋值。注意到 Tarjan 算法所求的强连通分量就是按拓扑序的逆序得出的，因此不需要真的缩点建新图求拓扑序，直接利用强连通分量的编号来当做顺序即可。

练习题

POJ3648 - Wedding Additionally, there are several pairs of people conducting adulterous relationships (both different-sex and same-sex relationships are possible)

adulterous relationships

```
#include <cstdio>
#include <cstring>
inline int min(int a, int b) { return a < b ? a : b; }
const int maxn = 2010, maxm = 500010;
int head[maxn], next[maxm], to[maxm], ecnt;
inline void addEdge(int f, int t)
{
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
    ecnt++;
}
int dfn[maxn], low[maxn], stk[maxn], scc[maxn], top, idx, scccnt;
void tarjan(int x)
{
    dfn[x] = low[x] = ++idx;
    stk[top++] = x;
    for (int i = head[x]; ~i; i = next[i])
        if (!dfn[to[i]])
            tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
        else if (!scc[to[i]])
            low[x] = min(low[x], dfn[to[i]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
        do
            scc[stk[--top]] = scccnt;
        while (stk[top] != x);
    }
}
int main()
{
    int m, n;
    while (scanf("%d%d", &n, &m) && (m + n))
    {
        memset(head, -1, sizeof(head));
        memset(dfn, 0, sizeof(dfn));
        memset(low, 0, sizeof(low));
        memset(scc, 0, sizeof(scc));
```

```
        idx = top = ecnt = scccnt = 0;
        int a1, a2;
        char c1, c2;
        for (int i = 0; i < m; i++)
        {
            scanf("%d%c %d%c", &a1, &c1, &a2, &c2);
            a1 = a1 << 1 | (c1 == 'h'), a2 = a2 << 1 | (c2 == 'h');
            addEdge(a1, a2 ^ 1), addEdge(a2, a1 ^ 1);
        }
        addEdge(0, 1);
        for (int i = 0; i < (n << 1); i++)
            if (!dfn[i]) tarjan(i);
        bool flag = true;
        for (int i = 0; i < n && flag; i++)
            if (scc[i << 1] == scc[i << 1 | 1])
                flag = false;
        if (!flag)
            puts("bad luck");
        else if (n < 1)
            putchar('\n');
        else
            for (int i = 1; i < n; i++)
                printf("%d%c%c", i, (scc[i << 1] > scc[i << 1 | 1]) ? 'w' : 'h', " \n"[i == n - 1]);
    }
    return 0;
}
```

POJ3678 - Katu Puzzle 我什么时候做过这个题？

Katu Puzzle

```
#include <cstdio>
#include <cstring>
inline int min(int a, int b) { return a < b ? a : b; }
const int maxn = 10010, maxm = 4000010;
int head[maxn], next[maxm], to[maxm], ecnt;
inline void addEdge(int f, int t)
{
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
    ecnt++;
}
int dfn[maxn], low[maxn], stk[maxn], scc[maxn], top, idx, scccnt;
void tarjan(int x)
{
    dfn[x] = low[x] = ++idx;
    stk[top++] = x;
    for (int i = head[x]; ~i; i = next[i])
        if (!dfn[to[i]])
            tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
```

```
        else if (!scc[to[i]])
            low[x] = min(low[x], dfn[to[i]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
        do
            scc[stk[--top]] = scccnt;
        while (stk[top] != x);
    }
}
int main()
{
    int n, m;
    while (~scanf("%d%d", &n, &m))
    {
        memset(dfn, 0, sizeof(dfn));
        memset(low, 0, sizeof(low));
        memset(scc, 0, sizeof(scc));
        memset(head, -1, sizeof(head));
        ecnt = top = idx = scccnt = 0;
        for (int i = 0, u, v, w; i < m; ++i)
        {
            char op[5];
            scanf("%d%d%d%s", &u, &v, &w, op);
            if (op[0] == 'A')
                if (w)
                {
                    addEdge(u << 1, v << 1 | 1), addEdge(v << 1, u << 1 | 1);
                    addEdge(u << 1, v << 1), addEdge(v << 1 | 1, u << 1 | 1);
                    addEdge(u << 1 | 1, v << 1 | 1), addEdge(v << 1, u << 1);
                }
                else
                {
                    addEdge(u << 1 | 1, v << 1), addEdge(v << 1 | 1, u << 1);
                }
            if (op[0] == 'O')
                if (w)
                {
                    addEdge(u << 1, v << 1 | 1), addEdge(v << 1, u << 1 | 1);
                }
                else
                {
                    addEdge(u << 1, v << 1), addEdge(v << 1 | 1, u << 1 | 1);
                    addEdge(u << 1 | 1, v << 1 | 1), addEdge(v << 1, u << 1);
                    addEdge(u << 1 | 1, v << 1), addEdge(v << 1 | 1, u << 1);
                }
            if (op[0] == 'X')
                if (w)
                {
                    addEdge(u << 1, v << 1 | 1), addEdge(v << 1, u << 1 | 1);
                    addEdge(u << 1 | 1, v << 1), addEdge(v << 1 | 1, u << 1);
                }
```

```
                else
                {
                    addEdge(u << 1, v << 1), addEdge(v << 1 | 1, u << 1 | 1);
                    addEdge(u << 1 | 1, v << 1 | 1), addEdge(v << 1, u << 1);
                }
        }
        for (int i = 0; i < (n << 1); i++)
            if (!dfn[i]) tarjan(i);
        bool flag = true;
        for (int i = 0; i < n && flag; i++)
            if (scc[i << 1] == scc[i << 1 | 1])
                flag = false;
        puts(flag ? "YES" : "NO");
    }
    return 0;
}
```

POJ2749 - Building roads 杀光奶牛问题就会得到解决

Building roads

```
#include <cstdio>
#include <cstring>
inline int abs(int x) { return x >= 0 ? x : -x; }
inline int min(int a, int b) { return a < b ? a : b; }
const int inf = 0x3f3f3f3f, maxn = 10010, maxm = 1200010;
int head[maxn], next[maxm], to[maxm], ecnt, n, A, B;
int dfn[maxn], low[maxn], stk[maxn], scc[maxn], top, idx, scccnt;
int sx1, sy1, sx2, sy2, sLen, X[maxn], Y[maxn], hate[maxn][2], like[maxn][2],
d[maxn];
inline void addEdge(int f, int t)
{
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
    ecnt++;
}
void tarjan(int x)
{
    dfn[x] = low[x] = ++idx;
    stk[top++] = x;
    for (int i = head[x]; ~i; i = next[i])
        if (!dfn[to[i]])
            tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
        else if (!scc[to[i]])
            low[x] = min(low[x], dfn[to[i]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
        do
            scc[stk[--top]] = scccnt;
        while (stk[top] != x);
```

```
    }
}
bool check(int x)
{
    memset(dfn, 0, sizeof(dfn));
    memset(low, 0, sizeof(low));
    memset(scc, 0, sizeof(scc));
    memset(head, -1, sizeof(head));
    ecnt = top = idx = scccnt = 0;
    for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
        {
            int l1 = d[i << 1], l2 = d[i << 1 | 1];
            int r1 = d[j << 1], r2 = d[j << 1 | 1];
            if (l1 + r1 > x)
                addEdge(i << 1, j << 1 | 1), addEdge(j << 1, i << 1 | 1);
            if (l1 + r2 + sLen > x)
                addEdge(i << 1, j << 1), addEdge(j << 1 | 1, i << 1 | 1);
            if (l2 + r1 + sLen > x)
                addEdge(i << 1 | 1, j << 1 | 1), addEdge(j << 1, i << 1);
            if (l2 + r2 > x)
                addEdge(i << 1 | 1, j << 1), addEdge(j << 1 | 1, i << 1);
        }
    for (int i = 1, a, b; i <= A; i++)
    {
        a = hate[i][0], b = hate[i][1];
        addEdge(a << 1, b << 1 | 1);
        addEdge(a << 1 | 1, b << 1);
        addEdge(b << 1, a << 1 | 1);
        addEdge(b << 1 | 1, a << 1);
    }
    for (int i = 1, a, b; i <= B; i++)
    {
        a = like[i][0], b = like[i][1];
        addEdge(a << 1, b << 1);
        addEdge(a << 1 | 1, b << 1 | 1);
        addEdge(b << 1, a << 1);
        addEdge(b << 1 | 1, a << 1 | 1);
    }
    for (int i = 1; i <= (n << 1); i++)
        if (!dfn[i])
            tarjan(i);
    for (int i = 1; i <= n; i++)
        if (scc[i << 1] == scc[i << 1 | 1])
            return false;
    return true;
}
int main()
{
    scanf("%d%d%d%d%d%d%d", &n, &A, &B, &sx1, &sy1, &sx2, &sy2);
    sLen = abs(sx1 - sx2) + abs(sy1 - sy2);
    for (int i = 1; i <= n; i++)
```

```
        scanf("%d%d", X + i, Y + i);
    for (int i = 1; i <= n; i++)
        d[i << 1] = abs(X[i] - sx1) + abs(Y[i] - sy1),
        d[i << 1 | 1] = abs(X[i] - sx2) + abs(Y[i] - sy2);
    for (int i = 1; i <= A; i++)
        scanf("%d%d", &hate[i][0], &hate[i][1]);
    for (int i = 1; i <= B; i++)
        scanf("%d%d", &like[i][0], &like[i][1]);
    int l = 0, r = 8000000, m, ans = -1;
    while (l <= r)
        check(m = (l + r) >> 1) ? r = (ans = m) - 1 : l = m + 1;
    printf("%d\n", ans);
    return 0;
}
```

1.4 欧拉回路

定义 设 $G = (V, E)$ 是一个图。

欧拉回路 图 G 中经过**每条边一次**并且**仅一次**的回路称作欧拉回路。

欧拉路径 图 G 中经过**每条边一次**并且**仅一次**的路径称作欧拉路径。

欧拉图 存在**欧拉回路**的图称为欧拉图。

半欧拉图 存在欧拉路径但不存在欧拉回路的图称为半欧拉图。

性质与定理 以下不加证明的给出一些定理 ~~（因为我懒得抄讲义子~~

定理 1 无向图 G 为欧拉图，当且仅当 G 为连通图且所有顶点的度为偶数。

推论 1 无向图 G 为半欧拉图，当且仅当 G 为连通图且除了两个顶点的度为奇数之外，其它所有顶点的度为偶数。

定理 2 有向图 G 为欧拉图，当且仅当 G 的基图[1]连通，且所有顶点的入度等于出度。

推论 2 有向图 G 为半欧拉图，当且仅当 G 的基图连通，且存在顶点 u 的入度比出度大 1、v 的入度比出度小 1，其它所有顶点的入度等于出度。

解法 由此可以得到以下求欧拉图 G 的欧拉回路的算法：

1. 在图 G 中任意找一个回路 C。
2. 将图 G 中属于回路 C 的边删除
3. 在残留图的各极大连通子图中分别寻找欧拉回路。
4. 将各极大连通子图的欧拉回路合并到 C 中得到图 G 的欧拉回路。

[1] 忽略有向图所有边的方向，得到的无向图称为该有向图的基图。

该算法的伪代码如下：

```
void dfs(u)
{
    for (edge e : edges[u])
        if (!flag[e])
        {
            flag[e] = true;
            flag[rev(e)] = true; //如果图 G 是有向图则删去本行
            dfs(e.to);
            S.push(v);
        }
}
```

最后依次取出栈 S 每一条边而得到图 G 的欧拉回路 (也就是边出栈序的逆序)。由于该算法执行过程中每条边最多访问两次，因此该算法的时间复杂度为 $O(|E|)$。

练习题

UOJ117 - 欧拉回路　混合两个子任务使代码风格变得鬼畜起来。

Building roads

```
#include <cstdio>
#include <cctype>
inline void read(int &x)
{
    int ch = x = 0;
    while (!isdigit(ch = getchar()));
    for (; isdigit(ch); ch = getchar()) x = x * 10 + ch - '0';
}
const int N = 100000 + 10, E = N << 2;
int adj[N], to[E], nxt[E], ecnt = 1;
int out[N], in[N], t, n, m;
bool flag[E];
int ans[E], tail;
inline void addEdge(int u, int v)
{
    ecnt++;
    nxt[ecnt] = adj[u];
    adj[u] = ecnt;
    to[ecnt] = v;
}
void dfs(int u)
{
    for (int &i = adj[u]; i; i = nxt[i])
    {
        int c = t == 1 ? i >> 1 : i - 1;
        bool sig = i & 1;
        if (!flag[c])
        {
```

```
            flag[c] = true;
            dfs(to[i]);
            ans[tail++] = (t == 1 && sig) ? -c : c;
        }
    }
}
int main()
{
    read(t), read(n), read(m);
    for (int i = 0, u, v; i < m; i++)
    {
        read(u), read(v);
        addEdge(u, v);
        out[u]++, in[v]++;
        if (t == 1) addEdge(v, u);
    }
    for (int i = 1; i <= n; i++)
        if (t == 1 ? ((in[i] + out[i]) & 1) : (in[i] != out[i]))
            return puts("NO"), 0;
    for (int i = 1; i <= n; i++)
        if (adj[i])
        {
            dfs(i);
            break;
        }
    if (tail != m) return puts("NO"), 0;
    puts("YES");
    for (int i = m - 1; i >= 0; i--) printf("%d ", ans[i]);
    return 0;
}
```

2 day2 字符串（一）

2.1 KMP

算法介绍 用来在线性时间内匹配字符串

算法流程 我觉得镰镰镰在 WC 上讲的比较清楚，于是我开始抄讲义。
字符串： $s[1\dots n], |s| = n$。
子串： $s[i\dots j] = s[i]s[i+1]\cdots[j]$。
前缀： $pre(s,x) = s[1\dots x]$, 后缀：$suf(s,x) = s[n{-}x+1\dots n]$
若 $0 \le r \le |s|, pre(s,r) = suf(s,r)$, 就称 $pre(s,r)$ 是 s 的 **border**。
KMP 算法的第一步主要做这么一件事：在 $O(n)$ 时间求出数组 $next[1\dots n]$, 其中 $next[i]$ 表示前缀 $s[1\dots i]$ 的最大 border 长度。于是可以知道 s 的所有 border 长度为 $next[n], next[next[n]], \cdots$，我想这是显然的，于是不加证明的在这里给出。
第二步就是匹配，如果失配了就把模式串的当前位置指针 i 跳到 $next[i]$ 处然后继续匹配，然后就好了。

算法实现

genNext

```
for (int i = 1, j = -1; i < len; i++)
{
    while (~j && str[j + 1] != str[i]) j = next[j];
    if (str[j + 1] == str[i]) j++;
    next[i] = j;
}
```

Find

```
for (int i = 0, j = -1; i < len; i++)
{
    while (~j && t[j + 1] != s[i]) j = next[j];
    if (t[j + 1] == s[i]) j++;
    if (j == len - 1) ans++, j = next[j];
}
```

练习题

POJ3461 - Oulipo 求出所有匹配位置

Oulipo

```
#include <cstdio>
#include <cstring>
char a[1 << 20 | 1], b[1 << 14 | 1];
int la, lb;
int next[1 << 14 | 1];
int main()
{
    next[0] = -1;
    int n;
    scanf("%d", &n);
    while (n--)
    {
        scanf("%s%s", b, a);
        la = strlen(a), lb = strlen(b);
        for (int i = 1, j = -1; i < lb; i++)
        {
            while (~j && b[j + 1] != b[i]) j = next[j];
            if (b[j + 1] == b[i]) j++;
            next[i] = j;
        }
        int ans = 0;
        for (int i = 0, j = -1; i < la; i++)
        {
            while (~j && b[j + 1] != a[i]) j = next[j];
            if (b[j + 1] == a[i]) j++;
            if (j == lb - 1) ans++, j = next[j];
        }
        printf("%d\n", ans);
    }
```

```
    return 0;
}
```

POJ2406 - Power Strings $next$ 数组的奇妙性质

Power Strings

```
#include <cstdio>
#include <cstring>
char str[1 << 20 | 1];
int next[1 << 20 | 1];
int len;
int main()
{
    next[0] = -1;
    while (scanf("%s", str))
    {
        if (str[0] == '.') break;
        len = strlen(str);
        for (int i = 0, j = -1; i < len;)
            (~j && str[i] != str[j]) ? j = next[j] : next[++i] = ++j;
        printf("%d\n", len % (len - next[len]) == 0 ? len / (len - next[len]) : 1);
    }
    return 0;
}
```

CF526D - Om Nom and Necklace 啥？

Om Nom and Necklace

```
#include <cstdio>
#include <cstring>
char s[1000010];
int next[1000010], n, k, len;
int main()
{
    scanf("%d%d%s", &n, &k, s);
    next[0] = -1;
    len = strlen(s);
    for (int i = 1; i < len; ++i)
    {
        int j;
        for (j = next[i - 1]; j != -1 && s[j + 1] != s[i]; j = next[j]);
        if (s[j + 1] == s[i]) j++;
        next[i] = j;
    }
    for (int i = 0; i < len; ++i)
    {
        int p = i + 1, q = p / (i - next[i]);
        putchar(((p % (i - next[i]) == 0) ? (q / k >= q % k ? '1' : '0') : (q / k > q % k
            ? '1' : '0')) );
    }
    return 0;
```

```
}
```

讲道理我 KMP 真的学的不是很明白，望各位 dalao 给予指导。

2.2 Trie

简介 字典树，也称 Trie、字母树，指的是某个字符串集合对应的形如下图的有根树。树的每条边上对应有恰好一个字符，每个顶点代表从根到该节点的路径所对应的字符串（将所有经过的边上的字符按顺序连接起来）。

实现 水

Trie - impl

```
struct node
{
    node *trans[26];
    int cnt;
};
void insert(node *n, char *str)
{
    for (; *str; n = n->trans[*str - '0'])
        if (n->trans[*str - '0'] == 0)
            n->trans[*str - '0'] = new_node();
    n->cnt++;
}
```

练习题

POJ3630 - Phone List 若插入过程中，有某个经过的节点带有串结尾标记，则之前插入的某个串是当前串的前缀。

Oulipo

```
#include <cstdio>
#include <cstring>
struct node
{
    node *trans[10];
    bool is_end;
} nodes[100010];
node *root;
int cnt;
node *new_node() { return &nodes[cnt++]; }
char buf[11];
bool try_insert(node *n, char *str)
{
    if (n->is_end) return false;
    if (*str == '\0')
    {
        for (int i = 0; i < 10; i++)
            if (n->trans[i])
                return false;
```

```
        n->is_end = true;
        return true;
    }
    if (n->trans[*str - '0'] == 0) n->trans[*str - '0'] = new_node();
    return try_insert(n->trans[*str - '0'], str + 1);
}
int main()
{
    int t, n;
    scanf("%d", &t);
    while (t--)
    {
        memset(nodes, 0, sizeof(nodes));
        cnt = 0;
        root = new_node();
        scanf("%d", &n);
        bool flag = true;
        while (n--)
        {
            scanf("%s", buf);
            if (flag) flag = try_insert(root, buf);
        }
        puts(flag ? "YES" : "NO");
    }
    return 0;
}
```

POJ2945 - Find the Clones n 个基因片段，每个长度为 m，输出 n 行表示重复出现 i 次 $(1 \leq i \leq n)$ 的基因片段的个数

Find the Clones

```
#include <cstdio>
#include <cstring>
struct node
{
    node *trans[4];
    int cnt;
} nodes[400010];
int tot;
node *root;
inline node *new_node() { return &nodes[tot++]; }
void try_insert(node *n, char *str)
{
    if (*str == '\0')
        n->cnt++;
    else
    {
        if (n->trans[*str - '0'] == 0) n->trans[*str - '0'] = new_node();
        try_insert(n->trans[*str - '0'], str + 1);
    }
}
```

```
char f[1 << 8 | 1];
int ans[20010];
int main()
{
    f['A'] = '0', f['C'] = '1', f['G'] = '2', f['T'] = '3';
    int n, m;
    char buf[22];
    while (~scanf("%d%d", &n, &m) && (n + m))
    {
        memset(ans, 0, sizeof(ans));
        memset(nodes, 0, sizeof(nodes));
        tot = 0;
        root = new_node();
        for (int i = 0; i < n; i++)
        {
            scanf("%s", buf);
            for (int j = 0; j < m; j++)
                buf[j] = f[buf[j]];
            try_insert(root, buf);
        }
        for (int i = 0; i < tot; i++) ans[nodes[i].cnt]++;
        for (int i = 1; i <= n; i++)
            printf("%d\n", ans[i]);
    }
    return 0;
}
```

2.3 Aho-Corasick Automaton

简介 多模式串字符串匹配，Trie 上的 KMP，其中 $next$ 数组变成了 $fail$ 指针，功能相同。

实现 Trie 的实现上文已经出现，所以此处不再重复。

buildFail

```
void buildFail()
{
    int h = 0, t = 0;
    root->fail = &virt;
    que[t++] = root;
    while (h < t)
    {
        node *cur = que[h++];
        for (int i = 0; i < 26; i++)
        {
            node *f = cur->fail;
            while (f->trans[i] == 0) f = f->fail;
            f = f->trans[i];
            if (cur->trans[i])
                (que[t++] = cur->trans[i])->fail = f;
            else
                cur->trans[i] = f;
```

```
        }
    }
}
```

练习题

HDU2222 - Keywords Search AC 自动机模板题，注意统计答案时，每个节点只能统计一次不要重复统计。

Keywords Search

```
#include <cstdio>
#include <cstring>
struct node
{
    node *trans[26], *fail;
    int cnt;
} nodes[500010], virt;
node *que[500010];
int tot;
node *root;
node *new_node() { return &nodes[tot++]; }
void insert(char *str)
{
    node *cur = root;
    for (; *str; str++)
    {
        if (cur->trans[*str - 'a'] == 0)
            cur->trans[*str - 'a'] = new_node();
        cur = cur->trans[*str - 'a'];
    }
    cur->cnt++;
}
void buildFail()
{
    int h = 0, t = 0;
    root->fail = &virt;
    que[t++] = root;
    while (h < t)
    {
        node *cur = que[h++];
        for (int i = 0; i < 26; i++)
        {
            node *f = cur->fail;
            while (f->trans[i] == 0) f = f->fail;
            f = f->trans[i];
            if (cur->trans[i])
                (que[t++] = cur->trans[i])->fail = f;
            else
                cur->trans[i] = f;
        }
    }
```

```
}
char buf[1000010];
int vis[500010];
int main()
{
    memset(vis, -1, sizeof(vis));
    int T, n;
    scanf("%d", &T);
    while (T--)
    {
        memset(nodes, 0, sizeof(nodes));
        tot = 0, root = new_node();
        for (int i = 0; i < 26; i++) virt.trans[i] = root;
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
        {
            scanf("%s", buf);
            insert(buf);
        }
        buildFail();
        scanf("%s", buf);
        node *cur = root, *tmp;
        int ans = 0;
        for (char *ch = buf; *ch; ch++)
        {
            tmp = cur = cur->trans[*ch - 'a'];
            while (tmp != &virt && vis[tmp - nodes] != T)
            {
                vis[tmp - nodes] = T;
                ans += tmp->cnt;
                tmp = tmp->fail;
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

2.4 Manacher

求出字符串每一位的回文半径，算法流程奥妙重重，不易让常人理解，然后我就抄了份板子改了改，然后就比讲义上的标程快了 20%。

练习题

POJ3974 - Palindrome Manacher 模板题

Palindrome

```
#include <cstdio>
#include <cstring>
inline int min(int a, int b) { return a < b ? a : b; }
```

```
inline int max(int a, int b) { return a > b ? a : b; }
char buf[1000100], str[2000200];
int R[2000200], T;
int main()
{
    while (scanf("%s", buf), buf[0] != 'E')
    {
        int m = int(strlen(buf)), n = 0;
        str[n++] = '!';
        str[n++] = '#';
        for (int i = 0; i < m; i++)
            str[n++] = buf[i], str[n++] = '#';
        str[n++] = '#';
        str[n++] = '?';
        int p = 0, mx = 0, ans = 0;
        for (int i = 1; i < n; i++)
        {
            R[i] = mx > i ? min(R[2 * p - i], mx - i) : 1;
            while (str[i + R[i]] == str[i - R[i]]) R[i]++;
            if (R[i] + i > mx)
                mx = i + R[p = i];
            ans = max(ans, R[i]);
        }
        printf("Case %d: %d\n", ++T, ans - 1);
    }
    return 0;
}
```

3 day3 简单数学

说是简单数学其实我后半部分也没看懂

3.1 整除及剩余

整除定义 设 a, b 是两个整数，且 $b \neq 0$. 如果存在整数 c，使得 $a = bc$，则称 a 被 b 整除，或 b 整除 a，记作 $b|a$。此时，又称 a 是 b 的倍数，b 是 a 的因子。

整除的基本性质

1. $a|b \wedge a|c \Rightarrow a|(b+c)$
2. $a|b \Rightarrow \forall c \in \mathbb{Z},\ a|bc$
3. $a|b \wedge b|c \Rightarrow a|c$

同余基本定义和定理

定义 1：带余除法

$$\forall a \in \mathbb{Z}, b \in \mathbb{Z}^* \rightarrow \exists q, r \in \mathbb{Z}, a = qb + r, r \in [0, |b|)$$

定义 2：同余

$$a \bmod m = b \bmod m \iff a \equiv b \pmod m$$

定义 3：剩余类

$$\mathbf{A_i} = \{x|x \in \mathbb{Z} \land x \bmod m = i\} \rightarrow \forall a, b \in \mathbf{A_i}, a \equiv b \pmod m$$

定义 4：完系

$$\{a_1 \bmod m, a_2 \bmod m, \ldots, a_n \bmod m\} = \{0, 1, 2, \ldots, m-1\}$$

定理 1

$$a \equiv b \pmod m \iff \exists k \in \mathbb{Z}, a = b + km \iff m|(a-b)$$

定理 2 同余关系是等价关系

1. $a \equiv a \pmod m$
2. $a \equiv b \pmod m \Rightarrow b \equiv a \pmod m$
3. $a \equiv b \pmod m \land b \equiv c \pmod m \Rightarrow a \equiv c \pmod m$

定理 3 同余的三则运算
$a, b, c \in \mathbb{R}, m \in \mathbb{N}^*, a \equiv b \pmod m \Rightarrow$

1. $a + c \equiv b + c \pmod m$
2. $a - c \equiv b - c \pmod m$
3. $ac \equiv bc \pmod m$

定理 4 同余式的三则运算
$a, b, c \in \mathbb{R}, m \in \mathbb{N}^*, a \equiv b \pmod m \land c \equiv d \pmod m \Rightarrow$

1. $ax + cy \equiv bx + dy \pmod m \qquad x, y \in \mathbb{Z}$
2. $ac \equiv bd \pmod m$
3. $a^n \equiv b^n \pmod m \qquad n \in \mathbb{N}^*$
4. $f(a) \equiv f(b) \pmod m \qquad f(x)$ 为任一整系数多项式

定理 5

1. $a \equiv b \pmod m \land d|m \Rightarrow a \equiv b \pmod d$
2. $a \equiv b \pmod m \Rightarrow gcd(a, m) = gcd(b, m)$
3. $\forall i \in [1, n]$, $a \equiv b \pmod{m_i} \iff a \equiv b \pmod{lcm(m_1, m_2, \ldots, m_n)}$

3.2 素数

定义 素数（质数）是大于 1 的正整数，并且除了 1 和它本身不能被其他正整数整除。大于 1 的非素数的正整数称为合数。

分布 素数有无穷多个. 如果使用 $\pi(x)$ 表示小于一个正实数 x 的素数有多少个，那么有

$$\lim_{x\to\infty}\frac{\pi(x)}{x}=\ln x$$

算术基本定理/惟一分解定理

$$n=\prod_{i=1}^{\infty}p_i^{a_i}\qquad p_i\in\mathbb{P},\ a_i\in\mathbb{N}$$

判定

$$n\in\mathbb{P}\iff\forall i\in[2,\sqrt{n}],\ i\nmid n$$

Eratosthenes 筛法

Eratosthenes Sieve

```
fill(isprime, true);
for (int i = 2; i < n; i++)
    if (isprime[i])
        for (int j = i * i; j < n; j += i)
            isprime[i] = false;
```

欧拉函数 欧拉函数 $\varphi(n)$ 指不超过 n 且与 n 互素的正整数的个数，其中 n 是一个正整数。

欧拉函数的性质 $n=\prod\limits_{i=1}^{m}p_i^{a_i}\to\varphi(n)=\prod\limits_{i=1}^{m}\varphi(p_i^{a_i})$

定理 1 $p\in\mathbb{P}\iff\varphi(p)=p-1$

定理 2 $p\in\mathbb{P},\ a\in\mathbb{N}_+\Rightarrow\varphi(p^a)=p^a-p^{a-1}$

定理 3 $m,n\in\mathbb{N}_+\wedge gcd(m,n)=1\Rightarrow\varphi(mn)=\varphi(m)\varphi(n)$

定理 4 $n=\prod\limits_{i=1}^{m}p_i^{a_i}\to\varphi(n)=n\prod\limits_{i=1}^{m}(1-\frac{1}{p_i})$

推论 $n\equiv 1\pmod 2\to\varphi(2n)=\varphi(n)$

定理 5 $n\in(2,+\infty)\bigcap\mathbb{Z}\Rightarrow\varphi(n)\equiv 0\pmod 2$

定理 6 $n\in\mathbb{N}_+\Rightarrow\sum\limits_{d|n}\varphi(n)=n$

欧拉定理 $gcd(a,m)=1,a\in\mathbb{N}_+,m\in[2,+\infty)\bigcap\mathbb{Z}\Rightarrow a^{\varphi(m)}\equiv 1\pmod m$

费马小定理 $m\in\mathbb{P}\Rightarrow a^{m-1}\equiv 1\pmod m$

练习题

POJ2689 - Prime Distance 暴力筛掉合数

Prime Distance

```cpp
#include <cmath>
#include <cstdio>
#include <cstring>
int prime_count;
int prime[5140];
bool f[1000010];
bool notprime[50010];
int main()
{
    for (int i = 2; i < 50010; i++)
        if (!notprime[i])
            for (long long j = 1ll * i * i; j < 50010; j += i)
                notprime[j] = true;
    for (int i = 2; i < 50010; i++)
        if (!notprime[i])
            prime[prime_count++] = i;
    int l, r;
    while (~scanf("%d%d", &l, &r))
    {
        l = l == 1 ? 2 : l;
        memset(f, 0, sizeof(f));
        for (int i = 0, a, b; i < prime_count; i++)
        {
            a = (l - 1) / prime[i] + 1;
            b = r / prime[i];
            for (int j = a; j <= b; j++)
                if (j > 1)
                    f[j * prime[i] - l] = true;
        }
        int mx = -1, mn = 0x3f3f3f3f, x1 = 0, x2 = 0, y1 = 0, y2 = 0;
        for (int i = 0, p = -1; i <= r - l; i++)
            if (!f[i])
            {
                if (~p)
                {
                    if (mx < i - p)
                        mx = i - p, x1 = p + l, y1 = i + l;
                    if (mn > i - p)
                        mn = i - p, x2 = p + l, y2 = i + l;
                }
                p = i;
            }
        if (mx == -1)
            puts("There are no adjacent primes.");
        else
            printf("%d,%d are closest, %d,%d are most distant.\n", x2, y2, x1, y1);
    }
    return 0;
}
```

POJ3421 - X-factor Chains 质因子的排列组合

X-factor Chains

```
#include <cstdio>
int f[1 << 12 | 1];
long long _f(int x)
{
    long long ans = 1;
    for (int i = x; i; i--) ans *= i;
    return ans;
}
int main()
{
    int x;
    while (~scanf("%d", &x))
    {
        int _x = x, t = 0;
        for (int i = 2; i * i <= _x; i++)
        {
            f[t] = 0;
            while (_x % i == 0)
                _x /= i, f[t]++;
            t++;
        }
        if (_x != 1) f[t++] = 1;
        int sum = 0;
        for (int i = 0; i < t; i++) sum += f[i];
        long long fac = _f(sum);
        for (int i = 0; i < t; i++) fac /= _f(f[i]);
        printf("%d %lld\n", sum, fac);
    }
    return 0;
}
```

POJ3090 - Visible Lattice Points 欧拉函数

Visible Lattice Points

```
#include <cstdio>
const int maxn = 1010;
int phi[maxn], sum[maxn];
int main()
{
    phi[1] = 1;
    for (int i = 2; i <= 1005; i++) if (!phi[i])
        for (int j = i; j <= 1005; j += i)
        {
            if (!phi[j]) phi[j] = j;
            phi[j] = phi[j] / i * (i - 1);
        }
    for (int i = 1; i <= 1005; i++) sum[i] = sum[i - 1] + phi[i];
    int T, x;
```

```
    scanf("%d", &T);
    for (int i = 1; i <= T; i++)
    {
        scanf("%d", &x);
        printf("%d %d %d\n", i, x, sum[x] << 1 | 1);
    }
    return 0;
}
```

3.3 欧几里得算法

最大公约数与最小公倍数

定义 1 设 a 和 b 是两个整数，如果 $d|a$ 且 $d|b$，则称 d 是 a 与 b 的公因子

定义 2 设 a 和 b 是两个不全为 0 的整数，称 a 与 b 的公因子中最大的为 a 与 b 的最大公因子，或最大公约数，记作 $gcd(a,b)$

定义 3 设 a 和 b 是两个非零整数，称 a 与 b 最小的正公倍数为 a 与 b 的最小公倍数，记作 $lcm(a,b)$

最大公约数与最小公倍数的性质

1. $a|m \wedge b|m \Rightarrow lcm(a,b)|m$
2. $d|a \wedge d|b \Rightarrow d|gcd(a,b)$
3. $lcm(a,b) = \frac{ab}{gcd(a,b)}$
4. $m,a,b \in \mathbb{N}_+ \rightarrow lcm(ma,mb) = m \times lcm(a,b)$, $gcd(ma,mb) = m \times gcd(a,b)$

计算方法

素因子分解法 令

$$a = \prod_{i=1}^{m} p_i^{r_i} \ , \quad b = \prod_{i=1}^{m} p_i^{s_i}$$

于是

$$gcd(a,b) = \prod_{i=1}^{m} p_i^{min(r_i,s_i)} \ , \quad lcm(a,b) = \prod_{i=1}^{m} p_i^{max(r_i,s_i)}$$

欧几里得算法 1

$$gcd(a,b) = gcd(b, a \bmod b)$$

欧几里得算法 2

$$gcd(a,b) = gcd(a, a-b)$$

拓展欧几里得算法 不理解就记下来

exgcd

```
void exgcd(int64 a, int64 b, int64 &x, int64 &y)
{ b == 0 ? (x = 1, y = 0) : (exgcd(b, a % b, y, x), y -= x * (a / b)); }
```

3.4 线性同余方程

二元一次不定方程

定义 1 $a,b,c\in\mathbb{Z}, a\neq 0, b\neq 0$，那形如 $ax+by=c$ 的方程称为二元一次不定方程。

定理 1 设 $a,b\in\mathbb{Z}$ 且 $d=gcd(a,b)$，如果 $d|c$，那么方程存在无穷多个整数解，否则方程不存在整数解。

定理 2 如果不定方程有解且特解为 $x=x_0, y=y_0$ 那么方程的解可以表示为

$$x=x_0+\frac{b}{d}t, y=y_0-\frac{a}{d}t, \text{其中} t\in\mathbb{Z}$$

同余方程与不定方程 在 $a>0$ 且 $b>0$ 的条件下，求二元一次方程 $ax+by=c$ 的整数解等价于求一元线性同余方程 $ax\equiv c \bmod b$ 的整数解

求一元线性同余方程 要求 $ax\equiv c \bmod b$，即为求 $ax+my=b$ 的解。记 $d=gcd(a,m)$，先使用拓展欧几里得求 $ax+my=b$，如果 $d\nmid b$ 则无解，否则 $\mod m$ 意义下的解有 d 个，可以通过对其中某个解不断地加 $\frac{m}{d}$ 得到。(这 d 个解的形式为 $x_0+\frac{m}{d}t,\ t\in\mathbb{Z}$, 其中 x_0 是已知的一个解)

中国剩余定理 若 $m_1, m_2, m_3, \ldots, m_r$ 是两两互素的正整数，则同余方程组

$$\begin{cases} x\equiv a_1 \pmod{m_1}\\ x\equiv a_2 \pmod{m_2}\\ x\equiv a_3 \pmod{m_3}\\ \ldots\\ x\equiv a_r \pmod{m_r}\end{cases} \tag{3.1}$$

有 $\mod M=\prod_1^r m$ 的唯一解，即为中国剩余定理。
若 $n=pq$ 且 $gcd(p,q)=1$，那么 $x \mod p, x \mod q$ 的值确定后，$x \mod n$ 的值也会随之确定。

算法说明

$$M_i=\prod_{j\neq i} m_j \to gcd(M_i,m_i)=1 \Rightarrow \exists p_i,q_i,\ M_ip_i+m_iq_i=1$$

$$(e_i=M_ip_i)\equiv int(j==i) \pmod{m_i} \to \sum_1^r e_ia_i \mod \sum_1^r m_i \text{是方程的最小非负整数解}$$

练习题

POJ1061 - 蛤蛤的约会 +1s

蛤蛤的约会

```
#include <cstdio>
typedef long long int64;
int64 gcd(int64 a, int64 b) { return b == 0 ? a : gcd(b, a % b); }
void exgcd(int64 a, int64 b, int64 &x, int64 &y) { b == 0 ? (x = 1, y = 0) : (exgcd(b, a
    % b, y, x), y -= x * (a / b)); }
```

```
int main()
{
    int64 s, t, p, q, L;
    scanf("%lld%lld%lld%lld%lld", &s, &t, &p, &q, &L);
    int64 a = (p - q + L) % L, b = L, c = (t - s + L) % L;
    int64 x = 0, y = 0, g = gcd(a, b);
    if (c % g)
        puts("Impossible");
    else
    {
        a /= g, b /= g, c /= g;
        exgcd(a, b, x, y);
        printf("%lld", (((x % b + b) % b) * c) % b);
    }
    return 0;
}
```

POJ2142 - The Balance 求 $ax+by=c$ 的一组解，使得 $|x|+|y|$ 尽量小，在前者尽量小时 $|ax|+|by|$ 尽量小

The Balance

```
#include <cstdio>
inline int abs(int x) { return x >= 0 ? x : -x; }
void exgcd(int a, int b, int &d, int &x, int &y) { !b ? (x = 1, y = 0, d = a) : (exgcd(b,
    a % b, d, y, x), y -= x * (a / b)); }
int main()
{
    int a, b, c, x, y, g, u1, v1, u2, v2;
    while (~scanf("%d%d%d", &a, &b, &c) && a + b + c)
    {
        exgcd(a, b, g, x, y);
        a /= g, b /= g, c /= g;
        u1 = (x % b * c % b + b) % b;
        v1 = abs((c - u1 * a) / b);
        v2 = (y % a * c % a + a) % a;
        u2 = abs((c - v2 * b) / a);
        if (u1 + v1 > u2 + v2 || (u1 + v1 == u2 + v2 && a * u1 + b * v1 > a * u2 + b * v2
            )) u1 = u2, v1 = v2;
        printf("%d %d\n", u1, v1);
    }
    return 0;
}
```

3.5 逆元

解一元线性同余方程

$$gcd(a,m)=1 \Rightarrow \exists x, ax \equiv 1 \pmod m$$

$$ax \equiv 1 \pmod m \iff \exists k \in \mathbb{Z}, ax-km=1$$

```
int inv(int a, int m)
{
    int x, y;
    exgcd(a, m, x, y);
    return (x % m + m) % m;
}
```

费马小定理

$$\forall p \in \mathbb{P} \to x^p \equiv x \pmod p$$

被称为费马小定理，若 $p \nmid x$，有

$$x^{p-1} \equiv 1 \pmod p$$

于是有

$$\forall p \in \mathbb{P}, x \in \mathbb{Z} \to x^{-1} \equiv x^{p-2} \pmod p$$

使用快速幂即可计算。

3.6 离散对数问题

解这鬼东西：

$$A^x \equiv B \pmod C$$

这玩意有个性质，$A^x \mod C$ 有周期性，最大周期不超过 C，我想这是显然的（除非你没上过小学）。

$C \in \mathbb{P}$ 普通的 BSGS

POJ2417 - Discrete Logging 模板题

Primitive Roots

```
#include <cmath>
#include <cstdio>
#include <cstring>
typedef long long i64;
void exgcd(i64 a, i64 b, i64 &x, i64 &y)
{
    b == 0 ? (x = 1, y = 0) : (exgcd(b, a % b, y, x), y -= (a / b) * x);
}
struct HashTable
{
    static const size_t sz = 500009;
    i64 idx[sz], val[sz];
    void init()
    {
        memset(idx, -1, sizeof(idx));
        memset(val, -1, sizeof(val));
    }
    void insert(i64 i, i64 v)
    {
        i64 j = i % sz;
```

```
        while (idx[j] != -1 && idx[j] != i)
        {
            j++;
            if (j == sz)
                j = 0;
        }
        if (val[j] == -1)
        {
            idx[j] = i;
            val[j] = v;
        }
    }
    i64 find(i64 i)
    {
        i64 j = i % sz;
        while (idx[j] != -1 && idx[j] != i)
        {
            j++;
            if (j == sz)
                j = 0;
        }
        return val[j];
    }
} H;
int main()
{
    for (i64 a, b, c; ~scanf("%lld%lld%lld", &c, &a, &b) && a | b | c;)
    {
        H.init();
        i64 m = i64(ceil(sqrt(c))), d = 1;
        for (int i = 0; i < m; i++, d = d * a % c)
            H.insert(d, i);
        i64 res = 1, x, y;
        bool flag = false;
        for (i64 i = 0; i < m && !flag; i++, res = res * d % c)
        {
            exgcd(res, c, x, y);
            x = (x * b % c + c) % c;
            i64 j = H.find(x);
            if (j != -1)
                printf("%lld\n", i * m + j), flag = true;
        }
        if (!flag)
            puts("no solution");
    }
    return 0;
}
```

$C \in \mathbb{N}_+$ 待续

3.7 原根

阶

$$n > 1, a \in \mathbb{Z}, gcd(a,n) = 1 \rightarrow \exists r \in [1,n], a^r \equiv 1 \pmod n$$

r 的最小整数值称为 a 模 n 的阶，记为 $Ord_n(a)$

阶的性质

$$gcd(a,n) = 1, r = Ord_n(a), \forall N \in \{x | a^N \equiv 1 \pmod n\} \Rightarrow r|N$$

$$gcd(a,n) = 1 \Rightarrow Ord_n(a)|\varphi(n)$$

$$\text{特别的，} p \in \mathbb{P}, gcd(a,p) = 1 \Rightarrow Ord_p(a)|p-1\text{，显然}\varphi(p) = p-1$$

原根 $n \in \mathbb{N}_+, a \in \mathbb{Z}, Ord_n(a) = \varphi(n)$，则称 a 为模 n 的一个原根，由阶的定义可知原根和 n 必然互质。

求质数 p 的原根算法 暴力从小到大枚举 $g \in \mathbb{N}_+$，$\forall a \in \{x \mid x|p-1, x \in \mathbb{P}\}, g^{\frac{p-1}{a}} \not\equiv 1 \pmod p$

Primitive Root

```
#include <cstdio>
typedef unsigned long long u64;
const size_t MAXN = 1 << 16 | 1;
bool notprime[MAXN];
int prime[MAXN], pcnt, x;
inline u64 fast_pow(u64 a, u64 b, u64 m)
{
    u64 ret = 1;
    for (; b; a = a * a % m, b >>= 1)
        if (b & 1)
            ret = ret * a % m;
    return ret;
}
int main()
{
    for (size_t i = 2; i < MAXN; i++)
        if (!notprime[i] && (prime[pcnt++] = int(i)))
            for (size_t j = i * i; j < MAXN; j += i)
                notprime[j] = true;
    while (~scanf("%d", &x))
        for (int a = 2, flag = 0; flag == 0; a++)
        {
            flag = 1;
            for (int i = 0, k = x - 1; i < pcnt && k > 1 && flag; i++)
                if (k % prime[i] == 0)
                {
                    flag = fast_pow(a, (x - 1) / prime[i], x) != 1;
                    while (k % prime[i] == 0)
                        k /= prime[i];
                }
            if (flag)
                printf("%d\n", a);
        }
```

```
    return 0;
}
```

练习题

POJ1284 - Primitive Roots 如果 $n \in \mathbb{N}_+$ 有一个原根，那么 n 一共有 $\varphi(\varphi(n))$ 个不同余的原根

Primitive Roots

```
#include <cstdio>
const int N = 1 << 16 | 1;
int phi[N];
int main()
{
    for (int i = 2; i < N; i++) if (!phi[i])
        for (int j = i; j < N; j += i)
        {
            if (!phi[j]) phi[j] = j;
            phi[j] = phi[j] / i * (i - 1);
        }
    for (int p; ~scanf("%d", &p);)
        printf("%d\n", phi[p - 1]);
    return 0;
}
```

4 day4 数据结构

4.1 树状数组

在 $\lg n$ 的时间内更新或查询 $\sum\limits_{i=1}^{n} a_i$

lowbit `lowbit(x) = x & -x`

两个操作 $\lg n$ 更新或查询

Fenwick tree

```
void add(int x, int v)
{
    for (; x <= n; x += lowbit(x))
        A[x] += v;
}

void sum(int x)
{
    int res = 0;
    for (; x; x -= lowbit(x))
        res += A[x];
    return res;
}
```

练习题

POJ2352 - Stars 。。。

Stars

```
#include <cstdio>
#include <cstring>
#define lowbit(x) ((x) & -(x))
const int maxn = 1 << 15 | 1;
int a[maxn], n, m, level[maxn];
void update(int pos, int x)
{
    for (; pos < maxn; pos += lowbit(pos)) a[pos] += x;
}
int query(int pos)
{
    int ans = 0;
    for (; pos; pos -= lowbit(pos)) ans += a[pos];
    return ans;
}
int main()
{
    scanf("%d", &n);
    memset(level, 0, sizeof(level));
    memset(a, 0, sizeof(a));
    for (int i = 0, x, y; i < n; ++i)
    {
        scanf("%d%d", &x, &y);
        level[query(++x)]++;
        update(x, 1);
    }
    for (int i = 0; i < n; ++i)
        printf("%d\n", level[i]);
    return 0;
}
```

POJ2299 - Ultra-QuickSort 求逆序对数量，不过配图是啥玩意？马桶橛子？

Ultra-QuickSort

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <stdint.h>
#define lowbit(x) ((x) & -(x))
const int N = 500005;
int sum[N];
int query(int x)
{
    int ans = 0;
    for (; x; x -= lowbit(x)) ans += sum[x];
    return ans;
```

```
}
void update(int x, int y)
{
    for (; x <= N; x += lowbit(x)) sum[x] += y;
}
struct abcd
{
    int val, pos;
    bool operator<(const abcd &rhs) const { return val < rhs.val; }
} nodes[N];
int map[N];
int main()
{
    int n;
    while (~scanf("%d", &n) && n)
    {
        memset(sum, 0, sizeof(sum));
        for (int i = 1; i <= n; i++) scanf("%d", &nodes[i].val), nodes[i].pos = i;
        std::sort(nodes + 1, nodes + n + 1);
        for (int i = 1; i <= n; i++) map[nodes[i].pos] = i;
        int64_t ans = 0;
        for (int i = 1; i <= n; i++)
        {
            update(map[i], 1);
            ans += i - query(map[i]);
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

POJ1990 - MooFest 杀光奶牛问题就会得到解决

MooFest

```
#include <algorithm>
#include <cstdio>
#define lowbit(x) ((x) & -(x))
const int N = 20005;
void add(int *arr, int pos, int val)
{
    for (; pos < N; pos += lowbit(pos))
        arr[pos] += val;
}
int query(int *arr, int pos)
{
    int ans = 0;
    for (; pos; pos -= lowbit(pos))
        ans += arr[pos];
    return ans;
}
int sum[2][N];
struct cow
{
    int pos, vol;
    bool operator<(const cow &rhs) const { return vol < rhs.vol; }
} cows[N];
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d%d", &cows[i].vol, &cows[i].pos);
    std::sort(cows + 1, cows + n + 1);
    long long ans = 0;
    for (int i = 1; i <= n; i++)
    {
        long long a = query(sum[0], cows[i].pos), b = query(sum[1], cows[i].pos);
        ans += (cows[i].pos * a - b + query(sum[1], 20000) - b -
                (i - 1 - a) * cows[i].pos) *
               cows[i].vol;
        add(sum[0], cows[i].pos, 1);
        add(sum[1], cows[i].pos, cows[i].pos);
    }
    printf("%lld", ans);
    return 0;
}
```

4.2 Sparse Table

处理区间最值，即 RMQ（Range Minimum Query）问题。

预处理 计算一个数组 f，使 $f[i][j] = min[i, i + 2^j)$，然后你就可以开始倍增了。

$$f[i][j] = \begin{cases} a[i] & j = 0 \\ min(f[i][j-1], f[i+2^{j-1}][j-1]) & j > 0 \end{cases} \tag{4.1}$$

询问 考虑一个询问区间 $[x, y)$ 的最小值的询问操作。
我们可以求出满足 $2^i \le y - x < 2^{i+1}$ 的 i，即 $\lfloor \log_2 y - x \rfloor$，这样我们可以用两个长度为 2^i 的小区间覆盖询问的大区间。
而长度为 2^i 的小区间的最小值在预处理时已经求出，于是区间 $[x, y)$ 的最小值为 $min\{f[x][i], f[y - 2^i][i]\}$，由于是求最值，区间有交集也没关系。

练习题

POJ3264 - Balanced Lineup 给你一个长度为 n 的序列 $a[N]$，询问 Q 次，每次输出 $[L, R]$ 区间最大值与最小值的差是多少，果真这种简化了的题面真是清晰易懂。

Balanced Lineup

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
inline int max(int a, int b) { return a > b ? a : b; }
const int N = 50010;
const int LogN = 16;
int minT[LogN][N], maxT[LogN][N], Log[N];
int main()
{
    Log[0] = -1;
    for (int i = 1; i < N; i++) Log[i] = Log[i >> 1] + 1;
    int m, n;
    scanf("%d%d", &n, &m);
    for (int i = 1, x; i <= n; i++)
    {
        scanf("%d", &x);
        minT[0][i] = maxT[0][i] = x;
    }
    for (int j = 1; j < LogN; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            minT[j][i] = min(minT[j - 1][i], minT[j - 1][i + (1 << (j - 1))]),
            maxT[j][i] = max(maxT[j - 1][i], maxT[j - 1][i + (1 << (j - 1))]);
    for (int i = 1, x, y, z; i <= m; i++)
    {
        scanf("%d%d", &x, &y);
        z = Log[y - x + 1];
        printf("%d\n", max(maxT[z][x], maxT[z][y - (1 << z) + 1]) - min(minT[z][x], minT[
            z][y - (1 << z) + 1]));
    }
    return 0;
}
```

CF359D - Pair of Numbers 首先要知道 gcd 有“最值的性质”，然后二分暴力。

Pair of Numbers

```
#include <cstdio>
#include <cctype>
inline int min(int a, int b) { return a < b ? a : b; }
inline int gcd(int a, int b) { return b == 0 ? a : gcd(b, a%b); }
inline void read(int &x)
{
    int ch = x = 0;
    while (!isdigit(ch = getchar()));
    for (; isdigit(ch); ch = getchar()) x = (x << 3) + (x << 1) + ch - '0';
}
int n, a[1 << 20 | 1], Log[1 << 20 | 1];
int M[1 << 20 | 1][20], G[1 << 20 | 1][20];
struct
{
    int c, a[1 << 20 | 1], r;
}ans;
inline bool can(int l, int r)
{
    int k = Log[r - l + 1];
    return min(M[l][k], M[r - (1 << k) + 1][k]) == gcd(G[l][k], G[r - (1 << k) + 1][k]);
}
bool check(int len)
{
    int cnt = 0;
    for (int i = 0; i + len < n; i++)
        if (can(i, i + len))
            ans.a[cnt++] = i;
    if (cnt) ans.r = len, ans.c = cnt;
    return cnt;
}
int main()
{
    read(n);
    Log[0] = -1;
    for (int i = 1; i <= n; i++) Log[i] = Log[i >> 1] + 1;
    for (int i = 0; i < n; i++) read(a[i]);
    for (int i = 0; i < n; i++) M[i][0] = G[i][0] = a[i];
    for (int j = 1; j <= Log[n]; j++)
        for (int i = 0; i + (1 << j) - 1 < n; i++)
            M[i][j] = min(M[i][j - 1], M[i + (1 << (j - 1))][j - 1]),
            G[i][j] = gcd(G[i][j - 1], G[i + (1 << (j - 1))][j - 1]);
    int L = 0, R = n, m;
    while (L < R)
        check(m = (L + R) >> 1) ? L = m + 1 : R = m;
    printf("%d %d\n", ans.c, ans.r);
    for (int i = 0; i < ans.c; i++)
        printf("%d ", ans.a[i] + 1);
    return 0;
}
```

4.3 左偏树

左偏树是一种可以合并的堆，所以它可以支持堆的几种操作，并且也都是在相同的时间复杂度下完成的，并且它能额外支持合并两棵左偏树的这种操作。

性质 我们先定义节点 i 的距离为节点 i 到它的后代中，最近的外节点所经过的边数，其中左子树或右子树为空的节点被称为外节点，这种节点的距离为 0。

性质 1：堆有序性质 节点的值不小于其子节点的值（假设这是一个大根左偏树）。

性质 2：左偏性质 节点的左子节点的距离不小于右子节点的距离。

性质 3 节点的距离等于它的右子节点的距离加一。

左偏树的操作

合并 现在我们有两棵左偏树 A 和 B，欲将其合并。我们假设 A 的根节点大于 B 的根节点，不然交换一下就行了。然后把 A 的右子节点与树 B 合并作为 A 的新右子节点，检查 A 现在的左子节点和右子节点的距离，如果右子节点的距离大于左子节点，那么交换这两棵子树，然后更新 A 的距离，如此这般递归下去，因为只有 $\log_2 n$ 的深度，所以铁定不会爆栈，除非你的脸黑到一定境界了。

合并的简单实现

```
struct node
{
    int val, dis;
    node *ch[2];
};
#define dis(x) ((x) ? (x)->dis : -1)
node *merge(node *a, node *b)
{
    if (a && b) return (node*)(uintptr_t(a) | uintptr_t(b));
    if (a->val < b->val) swap(a, b);
    a->ch[1] = merge(a->ch[1], b);
    if (dis(a->ch[0]) < dis(a->ch[1])) swap(a->ch[0], a->ch[1]);
    a->dis = dis(a->ch[1]) + 1;
    return a;
}
```

插入 把新节点当作一棵只有一个节点的树与原树合并。

删除极值 合并根节点下的两个子节点。

删除指定元素 将指定节点的两个子树合并作为这个节点。然后维护其父节点的数值，如果父节点被改变，就继续向上维护，直到节点数据不改变。

练习题

BZOJ2809 - [Apio2012]dispatching 考虑对每个点维护一个大根堆，堆内存储的是这个节点子树内所有点的 C_i。当堆内的和大于 M 时就要弹出堆顶元素；每个点的堆可以通过将所有儿子的堆合并起来再插入自己节点的 C_i 来得到

Apio2012 dispatching

```
#include <cctype>
#include <cstdio>
template <typename T> inline void swap(T &x, T &y) {
    T t = x;
    x = y;
    y = t;
}
template <typename T> inline T max(T x, T y) { return x > y ? x : y; }
inline void read(int &x)
{
    int ch = x = 0;
    while (!isdigit(ch = getchar()));
    for (; isdigit(ch); ch = getchar()) x = x * 10 + ch - '0';
}
const int maxn = 100010;
int n, m, fa[maxn], C[maxn], L[maxn], cnt;
typedef struct Node
{
    Node *lc, *rc;
    int val, dis, sz;
    long long sum;
} *lpNode;
lpNode merge(lpNode x, lpNode y)
{
    if (x == 0) return y;
    if (y == 0) return x;
    if (x->val < y->val) swap(x, y);
    x->rc = merge(x->rc, y);
    if (x->lc == 0 || x->lc->dis < x->rc->dis) swap(x->lc, x->rc);
    x->dis = x->rc ? x->rc->dis + 1 : 0;
    x->sz = (x->lc ? x->lc->sz : 0) + (x->rc ? x->rc->sz : 0) + 1;
    x->sum = (x->lc ? x->lc->sum : 0) + (x->rc ? x->rc->sum : 0) + x->val;
    return x;
}
Node mem[maxn];
lpNode nodes[maxn];
int head[maxn], to[maxn], next[maxn], ecnt, arr[maxn], aend;
inline void addEdge(int f, int t)
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
}
int main()
{
    read(n), read(m);
```

```
    for (int i = 1; i <= n; i++)
        read(fa[i]), read(C[i]), read(L[i]), addEdge(fa[i], i),
        (nodes[i] = mem + i)->sz = 1, nodes[i]->sum = nodes[i]->val = C[i];
    for (int i = 0; i <= n; i++)
        for (int cur = head[i]; cur; cur = next[cur])
            arr[++aend] = to[cur];
    long long ans = 0;
    for (int i = aend; i; i--)
    {
        int x = arr[i], y = fa[x];
        ans = max(ans, 1ll * nodes[x]->sz * L[x]);
        nodes[y] = merge(nodes[y], nodes[x]);
        while (y && nodes[y]->sum > m)
            nodes[y] = merge(nodes[y]->lc, nodes[y]->rc);
    }
    printf("%lld", ans);
    return 0;
}
```

4.4 线段树

既可以 $\log(n)$ 修改，也可以 $\log(n)$ 查询最值，也可以 $\log(n)$ 查询和，总之只要是能合并的数据都能在 $\log(n)$ 的时间内用其维护。

普通的线段树：单点修改，区间查询 直接搞

稍屌的线段树：区间修改，单点查询 普通的 Lazy-tag

通用的线段树：区间修改，区间查询 这东西包括以上两者，有两种实现方法，标记下传和标记永久化，其中标记永久化在可持久化数据结构中是必须的。现在给出一个大模板题，并且下面将会用两种方法加以实现。
这是题：POJ3468 - A Simple Problem with Integers

标记下传 这玩意儿是这么个搞法，还是照例的打标记，当我们要从某个节点递归下去时，将当前节点的 add 值下传，更新两个子节点的 add 与 sum 值，并将当前节点的 add 值清零。

标记下传例程：2.3s

```
#include <cstdio>
#include <cctype>
#define C(x) (x = getchar())
#define L(x) ((x) << 1)
#define R(x) ((x) << 1 | 1)
#define avg(x, y) (((x) + (y)) >> 1)
inline void read(int &x)
{
    int ch = x = 0, flag = 1;
    while (!isdigit(C(ch))) if (ch == '-') flag = -1;
    for (; isdigit(ch); C(ch))
        x = x * 10 + ch - '0';
    x *= flag;
```

```cpp
}
const int N = int(1e5 + 5);
typedef long long i64;
int a[N];
i64 A[N << 2], S[N << 2];
inline void add(int k, int l, int r, i64 v)
{
    A[k] += v;
    S[k] += (r - l) * v;
}
inline void pushdown(int k, int l, int r)
{
    if (A[k] == 0) return;
    int m = avg(l, r);
    add(L(k), l, m, A[k]);
    add(R(k), m, r, A[k]);
    A[k] = 0;
}
void build(int k, int l, int r)
{
    if (l == r - 1) return void(S[k] = a[l]);
    int m = avg(l, r);
    build(L(k), l, m);
    build(R(k), m, r);
    S[k] = S[L(k)] + S[R(k)];
}
i64 query(int k, int l, int r, int x, int y)
{
    if (x <= l && r <= y) return S[k];
    pushdown(k, l, r);
    int m = avg(l, r);
    i64 res = 0;
    if (x < m) res += query(L(k), l, m, x, y);
    if (y > m) res += query(R(k), m, r, x, y);
    return res;
}
void modify(int k, int l, int r, int x, int y, int v)
{
    if (x <= l && r <= y) return add(k, l, r, v);
    int m = avg(l, r);
    pushdown(k, l, r);
    if (x < m) modify(L(k), l, m, x, y, v);
    if (y > m) modify(R(k), m, r, x, y, v);
    S[k] = S[L(k)] + S[R(k)];
}
int main()
{
    int n, m;
    read(n), read(m);
    for (int i = 0; i < n; i++) read(a[i]);
    build(1, 0, n);
    while (m--)
```

```
    {
        int op = 0, x, y, z;
        while (op != 'Q' && op != 'C') C(op);
        if (op == 'Q')
        {
            read(x), read(y);
            printf("%lld\n", query(1, 0, n, x - 1, y));
        }
        else
        {
            read(x), read(y), read(z);
            modify(1, 0, n, x - 1, y, z);
        }
    }
    return 0;
}
```

标记永久化 这种情况下 add 标记将不会被下传，子节点的影响在修改时便已计算，递归时要累加祖先节点上的标记。

标记永久化例程：2.0s

```
#include <cctype>
#include <cstdio>
#define C(x) ((x) = getchar())
#define L(x) ((x) << 1)
#define R(x) ((x) << 1 | 1)
#define avg(x, y) (((x) + (y)) >> 1)
typedef long long i64;
inline int max(int a, int b) { return a > b ? a : b; }
inline int min(int a, int b) { return a < b ? a : b; }
inline void read(int &x)
{
    int ch = x = 0, sign = 1;
    while (!isdigit(C(ch))) if (ch == '-') sign = -1;
    for (; isdigit(ch); C(ch))
        x = x * 10 + ch - '0';
    x *= sign;
}
const int N = int(1e5 + 5);
i64 add[N << 2], sum[N << 2];
int a[N];
void build(int k, int l, int r)
{
    if (l == r - 1) return void(sum[k] = a[l]);
    int m = avg(l, r);
    build(L(k), l, m);
    build(R(k), m, r);
    sum[k] = sum[L(k)] + sum[R(k)];
}
void modify(int k, int l, int r, int x, int y, int v)
{
```

```
    if (x <= l && r <= y)
        return void(add[k] += v);
    sum[k] += 1ll * (min(r, y) - max(l, x)) * v;
    int m = avg(l, r);
    if (x < m) modify(L(k), l, m, x, y, v);
    if (y > m) modify(R(k), m, r, x, y, v);
}
i64 query(int k, int l, int r, int x, int y)
{
    if (x <= l && r <= y)
        return sum[k] + (r - l) * add[k];
    int m = avg(l, r);
    i64 res = add[k] * (min(r, y) - max(l, x));
    if (x < m) res += query(L(k), l, m, x, y);
    if (y > m) res += query(R(k), m, r, x, y);
    return res;
}
int main()
{
    int n, m;
    read(n), read(m);
    for (int i = 0; i < n; i++) read(a[i]);
    build(1, 0, n);
    while (m--)
    {
        int op = 0, x, y, z;
        while (op != 'C' && op != 'Q') C(op);
        if (op == 'Q')
        {
            read(x), read(y);
            printf("%lld\n", query(1, 0, n, x - 1, y));
        }
        else
        {
            read(x), read(y), read(z);
            modify(1, 0, n, x - 1, y, z);
        }
    }
    return  0;
}
```

经过测试，发现标记永久化的写法快一些，可能是因为没有那么多的 pushdown 导致的。