

# OI 模板

panda\_2134

2018 年 4 月 2 日

# 目录

<b>1 图的 DFS 树</b>	<b>4</b>
1.1 强连通分量	4
1.2 桥和割点	6
1.3 点双连通分量	6
1.4 边双连通分量	6
<b>2 最短路</b>	<b>7</b>
2.1 Dijkstra	7
2.1.1 Using <code>std::priority_queue</code>	7
2.1.2 Using <code>__gnu_pbds::priority_queue</code>	8
<b>3 网络流</b>	<b>9</b>
3.1 最大流	9
3.2 Dinic	9
3.3 最小费用最大流	11
3.3.1 zkw 费用流	11
3.3.2 Primal Dual	12
<b>4 树</b>	<b>16</b>
4.1 倍增 LCA	16
4.2 欧拉序列求 LCA	17
4.3 树链剖分	18
4.4 点分治	19
<b>5 单调数据结构</b>	<b>20</b>
5.1 单调队列 (滑动窗口)	20
5.2 单调栈	21
<b>6 线段树</b>	<b>22</b>
6.1 Lazy-Tag	22
6.2 动态开点线段树	25
6.3 可持久化线段树	25
<b>7 离线二维数点</b>	<b>25</b>
7.1 带修改	25
7.1.1 静态: 线段树 + 扫描线	25
7.1.2 动态: CDQ 分治	25
<b>8 在线二维数点</b>	<b>25</b>
8.0.1 动态: 二维线段树	25
8.0.2 动态: 树状数组套动态开点线段树	26
8.0.3 动态: 树状数组套平衡树	26

<b>9 平衡树</b>	<b>27</b>
9.1 Treap . . . . .	27
9.2 Splay . . . . .	27
<b>10 动态树</b>	<b>28</b>
10.1 Link-cut Tree . . . . .	28
<b>11 字符串</b>	<b>29</b>
11.1 KMP 字符串匹配 . . . . .	29
11.2 AC 自动机 . . . . .	29
11.3 后缀数组 . . . . .	31
<b>12 Miscellaneous</b>	<b>33</b>
12.1 ST 表 . . . . .	33
12.2 Fenwick Tree . . . . .	33
12.3 左偏树 . . . . .	33
<b>13 悬线法</b>	<b>34</b>
13.1 Algorithm 1 . . . . .	34
13.2 Algorithm 2 . . . . .	34
<b>14 莫队</b>	<b>35</b>
14.1 普通莫队 . . . . .	35
14.2 带修改莫队 . . . . .	35
<b>15 分块相关</b>	<b>36</b>
15.1 分块 . . . . .	36
15.2 区间众数 . . . . .	36
<b>16 数论</b>	<b>37</b>
16.1 线性求逆元 . . . . .	37
16.2 线性筛 . . . . .	37
16.2.1 求 $\varphi(n)$ . . . . .	37
16.2.2 求 $\mu(n)$ . . . . .	37
16.2.3 求 $d(n)$ . . . . .	37
16.3 扩展欧几里得定理 . . . . .	37
16.4 中国剩余定理 . . . . .	37
16.5 扩展欧拉定理 . . . . .	37
16.6 Lucas 定理 . . . . .	37

# 1 图的 DFS 树

## 1.1 强连通分量

一有向图上每个点有非负权值，求一条路径，使得路径上点权值和最大。点和边都可以多次经过，但是权值只计入答案一次。

Solution: 缩点后直接在 DAG 上 DP.

GraphTheory/TarjanSCC.cpp

```
1 #include <bits/stdc++.h>
2 #define fst first
3 #define snd second
4 using namespace std;
5
6 typedef pair<int, int> pii;
7 const int MAXN = 1e5, INF = 0x3f3f3f3f;
8
9 struct Graph {
10     struct Edge {
11         int v, next;
12     };
13
14     int n, m, e_ptr = 1, head[MAXN+10]; Edge E[(MAXN+10)<<1];
15
16     void add_edge(int u, int v) {
17         E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr;
18     }
19 } G1, G2;
20
21 int dfs_clock, scc_cnt, sccno[MAXN+10], dfn[MAXN+10], low[MAXN+10];
22 int ans, topo_cnt, topo_seq[MAXN+10], w[MAXN+10],
23     tot[MAXN+10], vis[MAXN+10], dp[MAXN+10];
24
25 stack<int> S;
26 void dfs(int u) {
27     dfn[u] = low[u] = ++dfs_clock;
28     S.push(u);
29     for(int j=G1.head[u]; j; j=G1.E[j].next) {
30         int v = G1.E[j].v;
31         if(!dfn[v]) {
32             dfs(v);
33             low[u] = min(low[u], low[v]);
34         } else if(!sccno[v])
35             low[u] = min(low[u], dfn[v]);
36     }
37     if(low[u] == dfn[u]) {
38         int v; ++scc_cnt;
39         do {
40             v = S.top(); S.pop();
41             sccno[v] = scc_cnt;
42             tot[scc_cnt] += w[v];
43         } while(u != v);
44     }
45 }
46
47 void Tarjan() {
```

```

48     for(int u = 1; u <= G1.n; u++)
49         if(!dfn[u]) dfs(u);
50 }
51
52 void scc_graph() {
53     set<pii> evis;
54     for(int u = 1; u <= G1.n; u++)
55         for(int j=G1.head[u]; j; j=G1.E[j].next) {
56             int v = G1.E[j].v;
57             if(sccno[u] == sccno[v] || evis.count(make_pair(sccno[u], sccno[v])))
58                 continue;
59             else {
60                 evis.insert(make_pair(sccno[u], sccno[v]));
61                 G2.add_edge(sccno[u], sccno[v]);
62             }
63         }
64     G2.n = scc_cnt;
65 }
66
67 bool topo_dfs(int u) {
68     vis[u] = -1;
69     for(int j=G2.head[u]; j; j=G2.E[j].next) {
70         int v = G2.E[j].v;
71         if(vis[v] == -1 || (vis[v] == 0 && !topo_dfs(v)))
72             return false;
73     }
74     vis[u] = 1;
75     topo_seq[topo_cnt--] = u;
76     return true;
77 }
78
79 bool toposort() {
80     topo_cnt = G2.n;
81     for(int u = G2.n; u >= 1; u--)
82         if(vis[u] == 0 && !topo_dfs(u)) return false;
83     return true;
84 }
85
86 inline int readint() {
87     int f=1, r=0; char c=getchar();
88     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
89     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
90     return f*r;
91 }
92
93 void init() {
94     int u, v;
95     G1.n = readint(); G1.m = readint();
96     for(int i = 1; i <= G1.n; i++)
97         w[i] = readint();
98     for(int i = 1; i <= G1.m; i++) {
99         u = readint(); v = readint();
100         G1.add_edge(u, v);
101     }
102     Tarjan(); scc_graph();
103     assert(toposort());
104 }

```

```

105
106 void work() {
107     for(int i = G2.n; i >= 1; i--) {
108         int u = topo_seq[i], maxv = 0;
109         for(int j=G2.head[u]; j; j=G2.E[j].next) {
110             int v = G2.E[j].v;
111             if(dp[v] > maxv) maxv = dp[v];
112         }
113         dp[u] = tot[u] + maxv;
114         ans = max(ans, dp[u]);
115     }
116     printf("%d", ans);
117 }
118
119 int main() {
120     init(); work();
121     return 0;
122 }

```

## 1.2 桥和割点

## 1.3 点双连通分量

## 1.4 边双连通分量

## 2 最短路

### 2.1 Dijkstra

#### 2.1.1 Using std::priority\_queue

GraphTheory/Dijkstra-STL.cpp

```
1 #include <bits/stdc++.h>
2 #define fst first
3 #define snd second
4 using namespace std;
5
6 typedef pair<int, int> HeapNode;
7
8 struct Edge {
9     int v, len, next;
10 };
11
12 const int MAXN = 1e4, MAXM = 5e5, INF = 0x3f3f3f3f;
13 int n, m, s, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
14 int dist[MAXN+10], done[MAXN+10];
15
16 void add_edge(int u, int v, int len) {
17     E[++e_ptr] = (Edge) { v, len, head[u] }; head[u] = e_ptr;
18 }
19
20 void add_pair(int u, int v, int len) {
21     add_edge(u, v, len); add_edge(v, u, len);
22 }
23
24 void Dijkstra() {
25     priority_queue<HeapNode, vector<HeapNode>, greater<HeapNode> > pq;
26     memset(done, 0, sizeof(done));
27     memset(dist, 0x3f, sizeof(dist));
28     dist[s] = 0; pq.push(make_pair(dist[s], s));
29     while(!pq.empty()) {
30         HeapNode p = pq.top(); pq.pop();
31         int u = p.snd;
32         if(done[u]) continue;
33         done[u] = true;
34         for(int j=head[u]; j; j=E[j].next) {
35             int v = E[j].v, len = E[j].len;
36             if(dist[v] > dist[u] + len) {
37                 dist[v] = dist[u] + len;
38                 pq.push(make_pair(dist[v], v));
39             }
40         }
41     }
42 }
43
44 inline int readint() {
45     int f=1, r=0; char c=getchar();
46     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
47     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
48     return f*r;
49 }
```

```

50
51 int main() {
52     int u, v, w;
53     n = readint(); m = readint(); s = readint();
54     for(int i = 1; i <= m; i++) {
55         u = readint(); v = readint(); w = readint();
56         add_edge(u, v, w);
57     }
58     Dijkstra();
59     for(int i = 1; i <= n; i++) {
60         if(dist[i] < INF)
61             printf("%d ", dist[i]);
62         else
63             printf("%d ", INT_MAX);
64     }
65     return 0;
66 }

```

### 2.1.2 Using \_\_gnu\_pbds::priority\_queue

使用了扩展库 `pb_ds` 中的配对堆，自带修改堆内元素操作，速度更快。仅在允许使用 STL 扩展时才使用。

#### GraphTheory/Dijkstra-pb\_ds.cpp

```

1 #include <bits/stdc++.h>
2 #include <bits/extc++.h>
3 #define fst first
4 #define snd second
5 using namespace std;
6
7 typedef pair<int, int> HeapNode;
8 typedef __gnu_pbds::priority_queue<HeapNode, greater<HeapNode>,
9     __gnu_pbds::pairing_heap_tag> PairingHeap;
10
11 struct Edge {
12     int v, len, next;
13 };
14
15 const int MAXN = 1e4, MAXM = 5e5, INF = 0x3f3f3f3f;
16 int n, m, s, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
17 int dist[MAXN+10]; PairingHeap pq; PairingHeap::point_iterator it[MAXN+10];
18
19 void add_edge(int u, int v, int len) {
20     E[++e_ptr] = (Edge) { v, len, head[u] }; head[u] = e_ptr;
21 }
22
23 void add_pair(int u, int v, int len) {
24     add_edge(u, v, len); add_edge(v, u, len);
25 }
26
27 void Dijkstra() {
28     memset(it, 0, sizeof(it));
29     memset(dist, 0x3f, sizeof(dist));
30     dist[s] = 0; it[s] = pq.push(make_pair(dist[s], s));
31     while(!pq.empty()) {

```



```

32     HeapNode p = pq.top(); pq.pop();
33     int u = p.snd;
34     for(int j=head[u]; j; j=E[j].next) {
35         int v = E[j].v, len = E[j].len;
36         if(dist[v] > dist[u] + len) {
37             dist[v] = dist[u] + len;
38             if(it[v] == NULL)
39                 it[v] = pq.push(make_pair(dist[v], v));
40             else
41                 pq.modify(it[v], make_pair(dist[v], v));
42         }
43     }
44 }
45 }
46
47 inline int readint() {
48     int f=1, r=0; char c=getchar();
49     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
50     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
51     return f*r;
52 }
53
54 int main() {
55     int u, v, w;
56     n = readint(); m = readint(); s = readint();
57     for(int i = 1; i <= m; i++) {
58         u = readint(); v = readint(); w = readint();
59         add_edge(u, v, w);
60     }
61     Dijkstra();
62     for(int i = 1; i <= n; i++) {
63         if(dist[i] < INF)
64             printf("%d ", dist[i]);
65         else
66             printf("%d ", INT_MAX);
67     }
68     return 0;
69 }

```

## 3 网络流

### 3.1 最大流

### 3.2 Dinic

NetworkFlow/Dinic.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Edge {
5     int v, flow, cap, next;
6 };
7
8 const int MAXN = 1e4, MAXM = 1e5, INF = 0x3f3f3f3f;

```

```

9  int n, m, s, t, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
10 int d[MAXN+10], cur[MAXN+10];
11
12 void AddEdge(int u, int v, int cap) {
13     E[++e_ptr] = (Edge) { v, 0, cap, head[u] }; head[u] = e_ptr;
14     E[++e_ptr] = (Edge) { u, 0, 0, head[v] }; head[v] = e_ptr;
15 }
16
17 bool BFS() {
18     queue<int> Q;
19     memset(d, 0xff, sizeof(d));
20     Q.push(s); d[s] = 0;
21     while(!Q.empty()) {
22         int u = Q.front(); Q.pop();
23         for(int j=head[u]; j; j=E[j].next) {
24             int v = E[j].v, f = E[j].flow, c = E[j].cap;
25             if(f < c && d[v] == -1) {
26                 d[v] = d[u] + 1;
27                 if(v == t) return true;
28                 else Q.push(v);
29             }
30         }
31     }
32     return false;
33 }
34
35 int DFS(int u, int flow) {
36     if(u == t || flow == 0) return flow; // !!!!!
37     int res = flow;
38     for(int &j=cur[u]; j; j=E[j].next) { // !!!!!
39         int v = E[j].v, f = E[j].flow, c = E[j].cap;
40         if(f < c && d[v] == d[u] + 1) {
41             int aug = DFS(v, min(res, c-f));
42             E[j].flow += aug; E[j^1].flow -= aug;
43             res -= aug;
44             if(res == 0) break; // !!!!!
45         }
46     }
47     return flow - res;
48 }
49
50 int Dinic() {
51     int MaxFlow = 0, CurFlow = 0;
52     while(BFS()) {
53         memcpy(cur, head, sizeof(head));
54         while((CurFlow = DFS(s, INF)))
55             MaxFlow += CurFlow;
56     }
57     return MaxFlow;
58 }
59
60 int main() {
61     int u, v, c;
62     scanf("%d%d%d", &n, &m, &s, &t);
63     for(int i = 1; i <= m; i++) {
64         scanf("%d%d", &u, &v, &c);
65         AddEdge(u, v, c);

```

```

66     }
67     printf("%d", Dinic());
68     return 0;
69 }

```

### 3.3 最小费用最大流

#### 3.3.1 zkw 费用流

NetworkFlow/zkw.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long int64;
5  struct Edge {
6      int u, v;
7      int64 flow, cap, cost;
8      int next;
9  };
10
11 const int MAXN = 5e3, MAXM = 5e4;
12 const int64 LL_INF = 0x3f3f3f3f3f3f3fLL;
13 int n, m, s, t, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1]; // ** E[(MAXM+10)<<1] **
14 int64 MaxFlow, MinCost, dist[MAXN+10], inq[MAXN+10], vis[MAXN+10];
15
16 void add_edge(int u, int v, int64 cap, int64 cost) {
17     E[++e_ptr] = (Edge) { u, v, 0, cap, cost, head[u] }; head[u] = e_ptr;
18     E[++e_ptr] = (Edge) { v, u, 0, 0, -cost, head[v] }; head[v] = e_ptr;
19 }
20
21 bool spfa() {
22     queue<int> Q;
23     memset(dist, 0x3f, sizeof(dist));
24     Q.push(t); dist[t] = 0; inq[t] = true;
25     while(!Q.empty()) {
26         int u = Q.front(); Q.pop(); inq[u] = false;
27         for(int j=head[u]; j; j=E[j].next) {
28             int v = E[j].v; int64 f = E[j^1].flow, c = E[j^1].cap, len = E[j^1].cost;
29             if(f < c && dist[v] > dist[u] + len) {
30                 dist[v] = dist[u] + len;
31                 if(!inq[v]) {
32                     inq[v] = true; Q.push(v);
33                 }
34             }
35         }
36     }
37     return dist[s] != LL_INF;
38 }
39
40 int64 dfs(int u, int64 flow) {
41     if(u == t || flow == 0) return flow;
42     vis[u] = true;
43     int64 res = flow;
44     for(int j=head[u]; j; j=E[j].next) {
45         int v = E[j].v; int64 f = E[j].flow, c = E[j].cap, len = E[j].cost;

```

```

46     if(f < c && !vis[v] && dist[v] == dist[u] - len) {
47         int64 aug = dfs(v, min(res, c-f));
48         E[j].flow += aug; E[j^1].flow -= aug;
49         res -= aug;
50         if(res == 0LL) break;
51     }
52 }
53 return flow - res;
54 }
55
56 void zkw() {
57     int64 CurFlow = 0LL;
58     while(spfa()) {
59         while(memset(vis, 0, sizeof(vis)),
60             CurFlow = dfs(s, LL_INF)) {
61             MaxFlow += CurFlow;
62             MinCost += dist[s] * CurFlow;
63         }
64     }
65 }
66
67 template<typename T>
68 inline void readint(T &x) {
69     T f=1, r=0; char c=getchar();
70     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
71     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
72     x = f*r;
73 }
74
75 int main() {
76     int u, v; int64 w, c;
77     readint(n); readint(m); readint(s); readint(t);
78     for(int i = 1; i <= m; i++) {
79         readint(u); readint(v); readint(w); readint(c);
80         add_edge(u, v, w, c);
81     }
82     zkw();
83     printf("%lld %lld", MaxFlow, MinCost);
84     return 0;
85 }

```

### 3.3.2 Primal Dual

NetworkFlow/PrimalDual.cpp

```

1 #include <bits/stdc++.h>
2 #include <bits/extc++.h>
3 #define fst first
4 #define snd second
5
6 using namespace std;
7
8 typedef long long int64;
9 typedef pair<int64, int> HeapNode;
10 typedef __gnu_pbds::priority_queue<HeapNode, greater<HeapNode>,
11     __gnu_pbds::pairing_heap_tag> PairingHeap;

```

```

12 const int MAXN = 5e3, MAXM = 5e4;
13 const int64 LL_INF = 0x3f3f3f3f3f3f3fLL;
14
15 struct Edge {
16     int u, v;
17     int64 flow, cap, cost;
18     int next;
19 };
20
21 int n, m, s, t, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
22 int64 MaxFlow, MinCost, delta, dist[MAXN+10], vis[MAXN+10], inq[MAXN+10];
23 PairingHeap::point_iterator it[MAXN+10];
24
25 void add_edge(int u, int v, int64 cap, int64 cost) {
26     E[++e_ptr] = (Edge) { u, v, 0, cap, cost, head[u] }; head[u] = e_ptr;
27     E[++e_ptr] = (Edge) { v, u, 0, 0, -cost, head[v] }; head[v] = e_ptr;
28 }
29
30 void Reduce() {
31     for(int i = 2; i <= e_ptr; i++)
32         E[i].cost -= (dist[E[i].u] - dist[E[i].v]);
33     delta += dist[s];
34 }
35
36 bool BellmanFord() {
37     queue<int> Q;
38     memset(dist, 0x3f, sizeof(dist));
39     Q.push(t); dist[t] = 0; inq[t] = true;
40     while(!Q.empty()) {
41         int u = Q.front(); Q.pop(); inq[u] = false;
42         for(int j=head[u]; j; j=E[j].next) {
43             int v = E[j].v; int64 f = E[j^1].flow, c = E[j^1].cap, len = E[j^1].cost;
44             if(f < c && dist[v] > dist[u] + len) {
45                 dist[v] = dist[u] + len;
46                 if(!inq[v]) {
47                     inq[v] = true; Q.push(v);
48                 }
49             }
50         }
51     }
52     return dist[s] != LL_INF;
53 }
54
55 bool Dijkstra() {
56     PairingHeap pq;
57     memset(dist, 0x3f, sizeof(dist));
58     memset(it, 0, sizeof(it));
59     dist[t] = 0; it[t] = pq.push(make_pair(dist[t], t));
60     while(!pq.empty()) {
61         HeapNode t = pq.top(); pq.pop();
62         int u = t.snd;
63         for(int j=head[u]; j; j=E[j].next) {
64             int v = E[j].v; int64 f = E[j^1].flow, c = E[j^1].cap, len = E[j^1].cost;
65             if(f < c && dist[v] > dist[u] + len) {
66                 dist[v] = dist[u] + len;
67                 if(it[v] == NULL)
68                     it[v] = pq.push(make_pair(dist[v], v));

```

```

69         else
70             pq.modify(it[v], make_pair(dist[v], v));
71     }
72 }
73 }
74 return dist[s] != LL_INF;
75 }
76
77 int64 dfs(int u, int64 flow) {
78     if(u == t || flow == 0) return flow;
79     vis[u] = true;
80     int64 res = flow;
81     for(int j=head[u]; j; j=E[j].next) {
82         int v = E[j].v; int64 f = E[j].flow, c = E[j].cap, len = E[j].cost;
83         if(f < c && !vis[v] && len == 0) {
84             int64 aug = dfs(v, min(res, c-f));
85             E[j].flow += aug; E[j^1].flow -= aug;
86             res -= aug;
87             if(res == 0) break;
88         }
89     }
90     return flow - res;
91 }
92
93 void Augment() {
94     int64 CurFlow = 0;
95     while( memset(vis, 0, sizeof(vis)),
96           (CurFlow = dfs(s, LL_INF)) ) {
97         MaxFlow += CurFlow;
98         MinCost += delta * CurFlow;
99     }
100 }
101
102 void PrimalDual() {
103     if(!BellmanFord()) return;
104     Reduce(); Augment();
105     while(Dijkstra()) {
106         Reduce(); Augment();
107     }
108 }
109
110 template<typename T>
111 inline void readint(T &x) {
112     T f=1, r=0; char c=getchar();
113     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
114     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
115     x = f*r;
116 }
117
118 int main() {
119     int u, v; int64 w, c;
120     readint(n); readint(m); readint(s); readint(t);
121     for(int i = 1; i <= m; i++) {
122         readint(u); readint(v); readint(w); readint(c);
123         add_edge(u, v, w, c);
124     }
125     PrimalDual();

```

```
126     printf("%lld %lld", MaxFlow, MinCost);
127     return 0;
128 }
```

## 4 树

### 4.1 倍增 LCA

Tree/DoublingLCA.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Edge { int v, next; };
5
6 const int MAXN = 1e6, LOG = 20;
7 int n, q, s, e_ptr = 1, head[MAXN+10]; Edge E[(MAXN+10)<<1];
8 int dep[MAXN+10], anc[MAXN+10][LOG+1];
9
10 void add_edge(int u, int v) { E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr; }
11 void add_pair(int u, int v) { add_edge(u, v); add_edge(v, u); }
12
13 void dfs(int u) {
14     for(int i = 1; i <= LOG; i++)
15         anc[u][i] = anc[anc[u][i-1]][i-1];
16     for(int j=head[u]; j; j=E[j].next) {
17         int v = E[j].v;
18         if(v == anc[u][0]) continue;
19         anc[v][0] = u; dep[v] = dep[u] + 1;
20         dfs(v);
21     }
22 }
23
24 int lca(int u, int v) {
25     if(dep[u] < dep[v]) swap(u, v);
26     for(int i = LOG; i >= 0; i--)
27         if(dep[anc[u][i]] >= dep[v])
28             u = anc[u][i];
29     if(u == v) return u;
30     for(int i = LOG; i >= 0; i--)
31         if(anc[u][i] != anc[v][i])
32             u = anc[u][i], v = anc[v][i];
33     u = anc[u][0], v = anc[v][0];
34     return u;
35 }
36
37 inline int readint() {
38     int f=1, r=0; char c=getchar();
39     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
40     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
41     return f*r;
42 }
43
44 int main() {
45     int u, v;
46     n = readint(); q = readint(); s = readint();
47     for(int i = 1; i <= n-1; i++) {
48         u = readint(); v = readint();
49         add_pair(u, v);
50     }
51     dep[s] = 1; dfs(s);
```



```

52     while(q--){
53         u = readint(); v = readint();
54         printf("%d\n", lca(u, v));
55     }
56     return 0;
57 }

```

## 4.2 欧拉序列求 LCA

Tree/EulerTourLCA.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int MAXN = 1e6;
5
6  struct Edge {
7      int v, next;
8  };
9
10 int n, q, s, e_ptr = 1, dfs_clock, head[MAXN+10]; Edge E[(MAXN+10)<<1];
11 int dfn[MAXN+10], dfs_seq[MAXN+10], idx[MAXN+10], euler_seq[(MAXN+10)<<1], st[(MAXN+10)<<1][22];
12 /*
13     dfn: dfs-clock of vertex u
14     idx: the index of vertex u in euler-tour sequence
15     dfs_seq: the dfs sequence
16 */
17
18 void add_edge(int u, int v) {
19     E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr;
20 }
21
22 void add_pair(int u, int v) {
23     add_edge(u, v); add_edge(v, u);
24 }
25
26 inline int readint() {
27     int f=1, r=0; char c=getchar();
28     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
29     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
30     return f*r;
31 }
32
33 void dfs(int u, int fa) {
34     euler_seq[++euler_seq[0]] = dfn[u] = ++dfs_clock;
35     idx[u] = euler_seq[0]; dfs_seq[dfs_clock] = u;
36     for(int j=head[u]; j; j=E[j].next) {
37         int v = E[j].v;
38         if(v == fa) continue;
39         dfs(v, u);
40         euler_seq[++euler_seq[0]] = dfn[u];
41     }
42 }
43
44 void init_lca() {
45     memset(st, 0x3f, sizeof(st));

```

```

46     for(int i = 1; i <= euler_seq[0]; i++)
47         st[i][0] = euler_seq[i];
48     for(int j = 1; j <= 21; j++)
49         for(int i = 1; i <= euler_seq[0] - (1<<j) + 1; i++) // bounds of sparse-table!
50             st[i][j] = min(st[i][j-1], st[i + (1 << (j-1))][j-1]);
51 }
52
53 int query(int l, int r) {
54     if(l > r) swap(l, r);
55     int j;
56     for(j = 0; (1 << (j+1)) <= (r-l+1); j++);
57     return min(st[l][j], st[r - (1<<j) + 1][j]);
58 }
59
60 int lca(int u, int v) {
61     return dfs_seq[query(idx[u], idx[v])];
62 }
63
64 int main() {
65     int u, v;
66     n = readint(); q = readint(); s = readint();
67     for(int i = 1; i <= n-1; i++) {
68         u = readint(); v = readint();
69         add_pair(u, v);
70     }
71     dfs(s, -1); init_lca();
72     while(q--) {
73         u = readint(); v = readint();
74         printf("%d\n", lca(u, v));
75     }
76     return 0;
77 }

```

### 4.3 树链剖分

Tree/HLD.cpp

```

1 // call Dfs1(1) and Dfs2(1, 1)
2 const int MAXN = 1e5;
3 int dfs_clock, Fa[MAXN+10], Son[MAXN+10], Sz[MAXN+10],
4     Dep[MAXN+10], Top[MAXN+10], Dfn[MAXN+10];
5
6 void Dfs1(int u) { // Fa Son Sz Dep
7     int maxsz = 0; Sz[u] = 1;
8     for(int j=head[u]; j; j=E[j].next) {
9         int v = E[j].v;
10        if(v == Fa[u]) continue;
11        Fa[v] = u; Dep[v] = Dep[u] + 1; // !
12        Dfs1(v); Sz[u] += Sz[v];
13        if(Sz[v] > maxsz) {
14            maxsz = Sz[v];
15            Son[u] = v;
16        }
17    }
18 }
19

```

```

20 void Dfs2(int u, int anc) { // Top Dfn
21     Dfn[u] = ++dfs_clock; Top[u] = anc;
22     if(Son[u]) Dfs2(Son[u], anc);
23     for(int j=head[u]; j; j=E[j].next) {
24         int v = E[j].v;
25         if(v == Fa[u] || v == Son[u]) continue;
26         Dfs2(v, v);
27     }
28 }
29
30 int LCA(int u, int v) {
31     while(Top[u] != Top[v]) {
32         if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
33         u = Fa[Top[u]];
34     }
35     if(Dep[u] > Dep[v]) swap(u, v);
36     return u;
37 }
38
39 int HLDQuery(int u, int v) {
40     int ret = -INF;
41     while(Top[u] != Top[v]) {
42         if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
43         ret = max(ret, st_query(1, 1, n, Dfn[Top[u]], Dfn[u]));
44         u = Fa[Top[u]];
45     }
46     if(Dep[u] > Dep[v]) swap(u, v);
47     ret = max(ret, st_query(1, 1, n, Dfn[u], Dfn[v]));
48     return ret;
49 }

```

## 4.4 点分治

## 5 单调数据结构

### 5.1 单调队列 (滑动窗口)

Monotonic/SlidingWindow.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAXN = 1e6;
5 int n, k, Hd, Tl, A[MAXN+10], Q[MAXN+10];
6
7 void SlideMin() {
8     Hd = 1, Tl = 0;
9     for(int i = 1; i <= k; i++) {
10         while(Hd <= Tl && A[Q[Tl]] >= A[i]) Tl--;
11         Q[++Tl] = i;
12     }
13     printf("%d ", A[Q[Hd]]);
14     for(int i = k+1; i <= n; i++) {
15         while(Hd <= Tl && Q[Hd] < i-k+1) Hd++;
16         while(Hd <= Tl && A[Q[Tl]] >= A[i]) Tl--;
17         Q[++Tl] = i;
18         printf("%d ", A[Q[Hd]]);
19     }
20 }
21
22 void SlideMax() {
23     Hd = 1, Tl = 0;
24     for(int i = 1; i <= k; i++) {
25         while(Hd <= Tl && A[Q[Tl]] <= A[i]) Tl--;
26         Q[++Tl] = i;
27     }
28     printf("%d ", A[Q[Hd]]);
29     for(int i = k+1; i <= n; i++) {
30         while(Hd <= Tl && Q[Hd] < i-k+1) Hd++;
31         while(Hd <= Tl && A[Q[Tl]] <= A[i]) Tl--;
32         Q[++Tl] = i;
33         printf("%d ", A[Q[Hd]]);
34     }
35 }
36
37 inline int readint() {
38     int f=1, r=0; char c=getchar();
39     while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
40     while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
41     return f*r;
42 }
43
44 int main() {
45     n = readint(); k = readint();
46     for(int i = 1; i <= n; i++) A[i] = readint();
47     SlideMin(); putchar(10); SlideMax();
48     return 0;
49 }
```

## 5.2 单调栈

## 6 线段树

### 6.1 Lazy-Tag

**Solution:** 暴力拆开式子后 (或者根据《重难点手册》的结论), 发现要维护区间的  $\sum x_i, \sum y_i, \sum x_i y_i, \sum x_i^2$ , 同时要支持区间加和区间设置为  $S + i$  和  $T_j$ . 在线段树上维护  $add_s, add_t, set_s, set_t$ , 然后推一推式子找出 Lazy-tag 更新主 Tag 的公式即可。几个坑点:

1.  $add_s, add_t$  标记在下推的时候, 不能赋值, 要累加!!! 累加!!! 累加!!!
2. 只有  $set_s, set_t$  用  $-1$  来标记不存在,  $add_s, add_t$  必须用  $0$  标记不存在! 不然是给自己找麻烦, 多出来各种特判!!!

SegTree/CorrelationAnalyse.cpp

```
1  /*
2     [SDOI2017] 相关分析
3     Coded by panda_2134
4  */
5  #include <bits/stdc++.h>
6  #define LC(o) ((o)*2)
7  #define RC(o) ((o)*2+1)
8  #define Mid(x, y) (((x) + (y)) / 2)
9  using namespace std;
10
11 const double eps = 1e-6, NONE = -1e6;
12 const int MAXN = 1e5;
13
14 int dcmp(double x) {
15     return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1);
16 }
17
18 struct Info {
19     double x, y, xy, x2;
20     Info() { x = y = xy = x2 = .0; }
21     Info(double a, double b, double c, double d):
22         x(a), y(b), xy(c), x2(d) {}
23     Info operator+(const Info &rhs) const {
24         return Info(x + rhs.x, y + rhs.y, xy + rhs.xy, x2 + rhs.x2);
25     }
26     Info operator+=(const Info &rhs) { return *this = *this + rhs; }
27 };
28
29 struct Node {
30     double x, y, xy, x2, add_s, add_t, set_s, set_t;
31     Node() {
32         x = y = xy = x2 = .0;
33         add_s = add_t = .0;
34         set_s = set_t = NONE;
35     }
36     void clear() { x = y = xy = x2 = .0; }
37 } nd[(MAXN+10)<<2];
38
39 int n, q;
40 double X[MAXN+10], Y[MAXN+10];
41
```

```

42 void Maintain(int o, double L, double R) {
43     if(dcmp(nd[o].set_s - NONE) == 0) {
44         // no set (if set_s exists, then set_t exists, and vice versa)
45         assert(dcmp(nd[o].set_t - NONE) == 0);
46         nd[o].clear();
47         if(L != R) {
48             nd[o].x = nd[LC(o)].x + nd[RC(o)].x;
49             nd[o].y = nd[LC(o)].y + nd[RC(o)].y;
50             nd[o].xy = nd[LC(o)].xy + nd[RC(o)].xy;
51             nd[o].x2 = nd[LC(o)].x2 + nd[RC(o)].x2;
52         }
53     } else {
54         nd[o].x2 = (R-L+1) * nd[o].set_s * nd[o].set_s
55             + R * (R+1) * (2*R+1) / 6 - L * (L-1) * (2*L-1) / 6
56             + nd[o].set_s * (L+R) * (R-L+1);
57         nd[o].xy = (R-L+1) * nd[o].set_s * nd[o].set_t
58             + (nd[o].set_s + nd[o].set_t) * (L+R) * (R-L+1) / 2
59             + R * (R+1) * (2*R+1) / 6 - L * (L-1) * (2*L-1) / 6;
60         nd[o].x = (R-L+1) * nd[o].set_s + (L+R) * (R-L+1) / 2;
61         nd[o].y = (R-L+1) * nd[o].set_t + (L+R) * (R-L+1) / 2;
62     }
63     nd[o].x2 += 2 * nd[o].add_s * nd[o].x + (R-L+1) * nd[o].add_s * nd[o].add_s;
64     nd[o].xy += nd[o].add_t * nd[o].x
65         + nd[o].add_s * nd[o].y + (R-L+1) * nd[o].add_s * nd[o].add_t;
66     nd[o].x += (R-L+1) * nd[o].add_s;
67     nd[o].y += (R-L+1) * nd[o].add_t; // update last
68 }
69
70 void Pushdown(int o) {
71     if(dcmp(nd[o].set_s - NONE) != 0) { // mark exist
72         assert(dcmp(nd[o].set_t - NONE) != 0);
73         nd[LC(o)].set_s = nd[RC(o)].set_s = nd[o].set_s;
74         nd[LC(o)].set_t = nd[RC(o)].set_t = nd[o].set_t;
75         nd[LC(o)].add_s = nd[RC(o)].add_s = .0;
76         nd[LC(o)].add_t = nd[RC(o)].add_t = .0;
77         nd[o].set_s = NONE;
78         nd[o].set_t = NONE;
79     }
80     if(dcmp(nd[o].add_s) != 0) {
81         nd[LC(o)].add_s += nd[o].add_s; //add 标记要累加!!!!!!!!!!!!
82         nd[RC(o)].add_s += nd[o].add_s;
83         nd[o].add_s = .0;
84     }
85     if(dcmp(nd[o].add_t) != 0) {
86         nd[LC(o)].add_t += nd[o].add_t;
87         nd[RC(o)].add_t += nd[o].add_t;
88         nd[o].add_t = .0;
89     }
90 }
91
92 Info Query(int o, int L, int R, int qL, int qR) {
93     Maintain(o, L, R);
94     if(qL <= L && R <= qR)
95         return Info(nd[o].x, nd[o].y, nd[o].xy, nd[o].x2);
96     else {
97         Info ret;
98         Pushdown(o);

```

```

99         if(qL <= Mid(L, R)) ret += Query(LC(o), L, Mid(L, R), qL, qR);
100        else Maintain(LC(o), L, Mid(L, R));
101        if(qR >= Mid(L, R)+1) ret += Query(RC(o), Mid(L, R)+1, R, qL, qR);
102        else Maintain(RC(o), Mid(L, R)+1, R);
103        return ret;
104    }
105 }
106
107 void BuildTree(int o, int L, int R) {
108     if(L == R) {
109         nd[o].add_s = X[L];
110         nd[o].add_t = Y[L];
111     } else {
112         BuildTree(LC(o), L, Mid(L, R));
113         BuildTree(RC(o), Mid(L, R)+1, R);
114     }
115     Maintain(o, L, R);
116 }
117
118 void Add(int o, int L, int R, int qL, int qR, double S, double T) {
119     if(qL <= L && R <= qR) {
120         nd[o].add_s += S;
121         nd[o].add_t += T;
122     } else {
123         Pushdown(o);
124         if(qL <= Mid(L, R)) Add(LC(o), L, Mid(L, R), qL, qR, S, T);
125         else Maintain(LC(o), L, Mid(L, R));
126         if(qR >= Mid(L, R)+1) Add(RC(o), Mid(L, R)+1, R, qL, qR, S, T);
127         else Maintain(RC(o), Mid(L, R)+1, R);
128     }
129     Maintain(o, L, R);
130 }
131
132 void Set(int o, int L, int R, int qL, int qR, double S, double T) {
133     if(qL <= L && R <= qR) {
134         nd[o].add_s = nd[o].add_t = .0; // override 'add' mark
135         nd[o].set_s = S;
136         nd[o].set_t = T;
137     } else {
138         Pushdown(o);
139         if(qL <= Mid(L, R)) Set(LC(o), L, Mid(L, R), qL, qR, S, T);
140         else Maintain(LC(o), L, Mid(L, R));
141         if(qR >= Mid(L, R)+1) Set(RC(o), Mid(L, R)+1, R, qL, qR, S, T);
142         else Maintain(RC(o), Mid(L, R)+1, R);
143     }
144     Maintain(o, L, R);
145 }
146
147 void init() {
148     scanf("%d%d", &n, &q);
149     for(int i = 1; i <= n; i++)
150         scanf("%lf", &X[i]);
151     for(int i = 1; i <= n; i++)
152         scanf("%lf", &Y[i]);
153     BuildTree(1, 1, n);
154 }
155

```



```

156 void work() {
157     int op, L, R; double S, T;
158     Info res;
159     while(q--){
160         scanf("%d", &op);
161         switch(op) {
162             case 1:
163                 scanf("%d%d", &L, &R);
164                 res = Query(1, 1, n, L, R);
165                 printf("%.12lf\n",
166                     (res.xy - res.x * res.y / (R-L+1)) / (res.x2 - res.x * res.x / (R-L+1)));
167                 break;
168             case 2:
169                 scanf("%d%d%lf%lf", &L, &R, &S, &T);
170                 Add(1, 1, n, L, R, S, T);
171                 break;
172             case 3:
173                 scanf("%d%d%lf%lf", &L, &R, &S, &T);
174                 Set(1, 1, n, L, R, S, T);
175                 break;
176         }
177     }
178 }
179
180 int main() {
181     init(); work();
182     return 0;
183 }
184 /*
185 5 2
186 1 -3 3 5 1
187 2 5 3 -4 -5
188
189 2 1 3 -3 -3
190 1 3 5
191 */

```

## 6.2 动态开点线段树

## 6.3 可持久化线段树

# 7 离线二维数点

## 7.1 带修改

### 7.1.1 静态：线段树 + 扫描线

### 7.1.2 动态：CDQ 分治

# 8 在线二维数点

## 8.0.1 动态：二维线段树

时间复杂度 插入 $O(\lg^2 n)$  – 查询 $O(\lg n)$  空间复杂度  $O(n^2)$

8.0.2 动态：树状数组套动态开点线段树

8.0.3 动态：树状数组套平衡树

## 9 平衡树

### 9.1 Treap

### 9.2 Splay

## 10 动态树

### 10.1 Link-cut Tree

## 11 字符串

### 11.1 KMP 字符串匹配

1-indexed

### 11.2 AC 自动机

0-indexed

String/ACAutomaton.cpp

```
1 #include <bits/stdc++.h>
2 #define CLEAR(x) memset((x), 0, sizeof(x))
3 using namespace std;
4
5 const int SIGMA = 26, MAX_TEMP_LEN = 70, MAXN = 150,
6 MAX_LEN = 1e6, MAX_NODE = MAXN * MAX_TEMP_LEN;
7
8 int N, sz, ch[MAX_NODE + 10][SIGMA + 2], f[MAX_NODE + 10], last[MAX_NODE+10],
9     val[MAX_NODE + 10], found_cnt[MAX_NODE+10];
10 char str[MAX_LEN+10], tpl[MAXN+10][MAX_TEMP_LEN+10];
11 unordered_map<string, int> ms;
12
13 inline int idx(char c) { return c - 'a' + 1; }
14
15 void insert(char *str) {
16     int u = 0, len = strlen(str);
17     for(int i = 0; i < len; i++) {
18         int c = idx(str[i]);
19         if(!ch[u][c]) ch[u][c] = ++sz;
20         u = ch[u][c];
21     }
22     ms[string(str)] = u;
23     ++val[u];
24 }
25
26 void get_fail() {
27     queue<int> Q;
28     f[0] = 0;
29     for(int c = 1; c <= SIGMA; c++) if(ch[0][c]) {
30         int v = ch[0][c];
31         f[v] = last[v] = 0;
32         Q.push(v);
33     }
34     while(!Q.empty()) {
35         int u = Q.front(); Q.pop();
36         for(int c = 1; c <= SIGMA; c++) {
37             int v = ch[u][c];
38             if(!v) {
39                 ch[u][c] = ch[f[u]][c];
40                 continue;
41             }
42             Q.push(v);
43
44             int u2 = f[u];
```

```

46         while(u2 && !ch[u2][c]) u2 = f[u2];
47         f[v] = ch[u2][c];
48         last[v] = val[f[v]] ? f[v] : last[f[v]];
49     }
50 }
51 }
52
53 void found(int u) {
54     for(; u; u = last[u])
55         found_cnt[u] += val[u];
56 }
57
58 void search(char *str) {
59     int u = 0, len = strlen(str);
60     for(int i = 0; i < len; i++) {
61         int c = idx(str[i]);
62         u = ch[u][c];
63         if(val[u]) found(u);
64         else if(last[u]) found(last[u]);
65     }
66 }
67
68 inline void readstr(char *str) {
69     char c=getchar(); int p=0;
70     while(!isalnum(c) && !ispunct(c)) c = getchar();
71     while(isalnum(c) || ispunct(c)) {
72         str[p++] = c;
73         c = getchar();
74     }
75     str[p++] = '\0';
76 }
77
78 int main() {
79     while(true) {
80         int ans = 0;
81         sz = 0; CLEAR(ch); CLEAR(f); CLEAR(found_cnt);
82         CLEAR(last); CLEAR(tpl); CLEAR(val); CLEAR(str);
83
84         scanf("%d", &N); if(N == 0) break;
85         for(int i = 1; i <= N; i++) {
86             readstr(tpl[i]); insert(tpl[i]);
87         }
88         get_fail();
89
90         readstr(str); search(str);
91
92         for(int i = 0; i <= sz; i++)
93             ans = max(ans, found_cnt[i]);
94         printf("%d\n", ans);
95         for(int i = 1; i <= N; i++)
96             if(found_cnt[ms[string(tpl[i])]] == ans)
97                 printf("%s\n", tpl[i]);
98     }
99     return 0;
100 }

```

## 11.3 后缀数组

0-indexed

String/SuffixArray.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAXLEN = 1e6, SIGMA = 100;
5
6 inline int idx(char c) {
7     if(!c) return 0;
8     else if(isdigit(c)) return c - '0' + 1;
9     else if(isupper(c)) return c - 'A' + 1 + 10;
10    else if(islower(c)) return c - 'a' + 1 + 10 + 26;
11    else throw "Invalid Character";
12 }
13
14 struct SuffixArray {
15     int sa[MAXLEN+10], rk[MAXLEN+10], buf[3][MAXLEN+10], height[MAXLEN+10], c[MAXLEN+10];
16     void build_sa(char *s, int len) {
17         int m = SIGMA + 10, n = len + 1, *x = buf[0], *y = buf[1];
18         for(int i = 0; i < m; i++) c[i] = 0;
19         for(int i = 0; i < n; i++) ++c[x[i] = idx(s[i])];
20         for(int i = 1; i < m; i++) c[i] += c[i-1];
21         for(int i = n-1; i >= 0; i--) sa[--c[x[i]]] = i;
22         for(int k = 1; k <= n; k <= 1) {
23             int p = 0;
24             for(int i = n-k; i < n; i++) y[p++] = i;
25             for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
26             for(int i = 0; i < m; i++) c[i] = 0;
27             for(int i = 0; i < n; i++) ++c[x[y[i]]];
28             for(int i = 1; i < m; i++) c[i] += c[i-1];
29             for(int i = n-1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
30             swap(x, y);
31             p = 1, x[sa[0]] = 0;
32             for(int i = 1; i < n; i++)
33                 x[sa[i]] = (y[sa[i]] == y[sa[i-1]] && y[sa[i] + k] == y[sa[i-1] + k] ? p-1 : p++);
34             if(p >= n) break;
35             m = p;
36         }
37         memcpy(rk, x, sizeof(rk));
38         int k = 0;
39         for(int i = 0; i < n; i++) {
40             if(!rk[i]) continue;
41             if(k) k--;
42             int j = sa[rk[i]-1];
43             while(s[i+k] == s[j+k]) k++;
44             height[rk[i]] = k;
45         }
46     }
47 } SA;
48 inline void readstr(char* str) {
49     char c=getchar(); int p=0;
50     while(!isalnum(c) && !ispunct(c)) c=getchar();
51     while(isalnum(c) || ispunct(c)) {
52         str[p++] = c;
```

```
53     c = getchar();
54 }
55 str[p++] = '\0';
56 }
57
58 int len;
59 char str[MAXLEN+10];
60
61 int main() {
62     readstr(str); len = strlen(str);
63     SA.build_sa(str, len);
64     for(int i = 1; i <= len; i++)
65         printf("%d ", SA.sa[i]+1);
66     return 0;
67 }
```



## 12 Miscellaneous

### 12.1 ST 表

### 12.2 Fenwick Tree

### 12.3 左偏树

## 13 悬线法

### 13.1 Algorithm 1

### 13.2 Algorithm 2

## 14 莫队

### 14.1 普通莫队

### 14.2 带修改莫队

## 15 分块相关

### 15.1 分块

### 15.2 区间众数

## 16 数论

### 16.1 线性求逆元

推导 令  $p = ki + r (0 \leq r < i)$

则  $0 \equiv ki + r \pmod{p}$

$$\Rightarrow ki \cdot i^{-1}r^{-1} + r \cdot i^{-1}r^{-1} \equiv 0$$

$$\Rightarrow i^{-1} \equiv -k \cdot r^{-1} \equiv p - \left\lfloor \frac{p}{i} \right\rfloor + p \pmod{i} \pmod{p}$$

Math/inv.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 3e6;
4 int n, p, inv[MAXN+10];
5 int main() {
6     cin >> n >> p;
7     inv[1] = 1; printf("%d\n", inv[1]);
8     for(int i = 2; i <= n; i++)
9         printf("%d\n", inv[i] = (p - (long long)(p / i) * inv[p % i] % p) % p);
10    return 0;
11 }
```

### 16.2 线性筛

16.2.1 求  $\varphi(n)$

16.2.2 求  $\mu(n)$

16.2.3 求  $d(n)$

### 16.3 扩展欧几里得定理

### 16.4 中国剩余定理

### 16.5 扩展欧拉定理

### 16.6 Lucas 定理