

BELJING NORMAL UNIVERSITY  
SCHOOL OF MATHEMATICS

---

# Template

---

appleDog

2024 年 10 月 30 日

# 目录

|          |                                    |           |
|----------|------------------------------------|-----------|
| <b>1</b> | <b>hpp</b>                         | <b>3</b>  |
| 1.1      | heading . . . . .                  | 3         |
| 1.2      | debug.h . . . . .                  | 4         |
| 1.3      | mod int . . . . .                  | 4         |
| <b>2</b> | <b>Shell Scripts</b>               | <b>6</b>  |
| 2.1      | checker.sh . . . . .               | 6         |
| <b>3</b> | <b>data structure</b>              | <b>6</b>  |
| 3.1      | cartesian tree . . . . .           | 6         |
| 3.2      | segment tree . . . . .             | 6         |
| 3.3      | hjt segment tree . . . . .         | 10        |
| 3.4      | LiChao tree . . . . .              | 12        |
| 3.5      | treap . . . . .                    | 12        |
| 3.6      | splay . . . . .                    | 14        |
| 3.7      | Link Cut Tree . . . . .            | 17        |
| 3.8      | ODT . . . . .                      | 17        |
| 3.9      | tree in tree . . . . .             | 17        |
| <b>4</b> | <b>string</b>                      | <b>17</b> |
| 4.1      | KMP . . . . .                      | 17        |
| 4.2      | z-function . . . . .               | 17        |
| 4.3      | Manacher . . . . .                 | 18        |
| 4.4      | AC automaton . . . . .             | 18        |
| 4.5      | PAM . . . . .                      | 18        |
| 4.6      | suffix array . . . . .             | 18        |
| <b>5</b> | <b>graph</b>                       | <b>20</b> |
| 5.1      | shortest path . . . . .            | 20        |
| 5.2      | minimum spanning tree . . . . .    | 20        |
| 5.3      | SCC . . . . .                      | 20        |
| 5.4      | DCC . . . . .                      | 20        |
| 5.5      | 2-SAT . . . . .                    | 20        |
| 5.6      | minimum ring . . . . .             | 20        |
| 5.7      | tree - center of gravity . . . . . | 20        |

|   |           |
|---|-----------|
| 目录  | 3         |
| 5.8 tree - DSU on tree . . . . .                      | 20        |
| 5.9 tree - AHU . . . . .                              | 20        |
| 5.10 tree - LCA . . . . .                             | 20        |
| 5.11 tree - LLD . . . . .                             | 20        |
| 5.12 tree - HLD . . . . .                             | 20        |
| 5.13 tree - virtual tree . . . . .                    | 20        |
| 5.14 tree - pseudo tree . . . . .                     | 20        |
| 5.15 tree - prufer sequence . . . . .                 | 20        |
| 5.16 tree - divide and conquer on tree . . . . .      | 20        |
| 5.17 network flow - maximal flow . . . . .            | 20        |
| 5.18 network flow - minimal cost flow . . . . .       | 20        |
| 5.19 network flow - minimal cut . . . . .             | 20        |
| 5.20 matching - matching on bipartite graph . . . . . | 20        |
| 5.21 matching - general on general graph . . . . .    | 20        |
| <b>6 math - number theory</b>                         | <b>20</b> |
| <b>7 math - polynomial</b>                            | <b>20</b> |
| <b>8 math - numerical analysis</b>                    | <b>20</b> |
| <b>9 math - game theory</b>                           | <b>20</b> |
| <b>10 math - linear algebra</b>                       | <b>20</b> |
| <b>11 math - complex number</b>                       | <b>20</b> |
| <b>12 geometry</b>                                    | <b>20</b> |
| <b>13 sweep line</b>                                  | <b>20</b> |
| <b>14 offline algorithm</b>                           | <b>20</b> |

# 1 hpp

## 1.1 heading

```

1  #include <bits/stdc++.h>
2
3  // using namespace std;
4
5  #define typet typename T
6  #define typeu typename U
7  #define types typename... Ts
8  #define tempt template <typet>
9  #define tempu template <typeu>
10 #define temps template <types>
11 #define tandu template <typet, typeu>
12
13 using LL = long long;
14 using i128 = __int128;
15 using PII = std::pair<int, int>;
16 /*
17 using UI = unsigned int;
18 using ULL = unsigned long long;
19 using ULL = unsigned long long;
20 using PIL = std::pair<int, LL>;
21 using PLI = std::pair<LL, int>;
22 using PLL = std::pair<LL, LL>;
23 */
24 using vi = std::vector<int>;
25 using vvi = std::vector<vi>;
26 using vl = std::vector<LL>;
27 using vvl = std::vector<vl>;
28 using vpi = std::vector<PII>;
29
30 #define ff first
31 #define ss second
32 #define all(v) v.begin(), v.end()
33 #define rall(v) v.rbegin(), v.rend()
34
35 #ifdef LOCAL
36 #include "../debug.h"
37 #else
38 #define debug(...) \
39     do { \
40     } while (false)
41 #endif
42
43 constexpr int mod = 998244353;
44 constexpr int inv2 = (mod + 1) / 2;
45 constexpr int inf = 0x3f3f3f3f;
46 constexpr LL INF = 1e18;
47 constexpr double pi = 3.141592653589793;
48 constexpr double eps = 1e-6;
49
50 constexpr int lowbit(int x) { return x & -x; }
51 /*
52 constexpr int add(int x, int y) { return x + y < mod ? x + y : x - mod + y; }
53 constexpr int sub(int x, int y) { return x < y ? mod + x - y : x - y; }
54 constexpr int mul(LL x, int y) { return x * y % mod; }
55 constexpr void Add(int& x, int y) { x = add(x, y); }
56 constexpr void Sub(int& x, int y) { x = sub(x, y); }
57 constexpr void Mul(int& x, int y) { x = mul(x, y); }
58 constexpr int pow(int x, int y, int z = 1) {
59     for (; y; y /= 2) {
60         if (y & 1) Mul(z, x);
61         Mul(x, x);
62     }
63     return z;
64 }
65 temps constexpr int add(Ts... x) {
66     int y = 0;
67     (... , Add(y, x));
68     return y;
69 }
70 temps constexpr int mul(Ts... x) {
71     int y = 1;
72     (... , Mul(y, x));
73     return y;
74 }
75 */
76
77 tandu bool Max(T& x, const U& y) { return x < y ? x = y, true : false; }
78 tandu bool Min(T& x, const U& y) { return x > y ? x = y, true : false; }
79

```

```

80 void solut() {
81
82 }
83
84 int main() {
85     std::ios::sync_with_stdio(false);
86     std::cin.tie(0);
87     std::cout.tie(0);
88
89     int t = 1;
90     std::cin >> t;
91     while (t-- > 0) {
92         solut();
93     }
94     return 0;
95 }

```

## 1.2 debug.h

```

1  template <typename T, typename U>
2  std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p) {
3      return os << "<" << p.first << ", " << p.second << ">";
4  }
5
6  template <
7      typename T, typename = decltype(std::begin(std::declval<T>())),
8      typename = std::enable_if_t<!std::is_same_v<T, std::string>>>
9  std::ostream& operator<<(std::ostream& os, const T& c) {
10     auto it = std::begin(c);
11     if (it == std::end(c)) return os << "{}";
12     for (os << "{" << *it; ++it != std::end(c); os << ", " << *it);
13         return os << "}";
14 }
15
16 #define debug(arg...) \
17     do { \
18         std::cerr << "[" #arg "]" :"; \
19         dbg(arg); \
20     } while (false)
21
22 template <typename... Ts>
23 void dbg(Ts... args) {
24     (... , (std::cerr << " " << args));
25     std::cerr << std::endl;
26 }

```

## 1.3 mod int

```

1  template <int P>
2  struct Mint {
3      int v = 0;
4
5      // reflection
6      template <typet = int>
7      constexpr operator T() const {
8          return v;
9      }
10
11     // constructor //
12     constexpr Mint() = default;
13     template <typet>
14     constexpr Mint(T x) : v(x % P) {}
15     constexpr int val() const { return v; }
16     constexpr int mod() { return P; }
17
18     // io //
19     friend std::istream& operator>>(std::istream& is, Mint& x) {
20         LL y;
21         is >> y;
22         x = y;
23         return is;
24     }
25     friend std::ostream& operator<<(std::ostream& os, Mint x) { return os << x.v; }
26
27     // comparision //
28     friend constexpr bool operator==(const Mint& lhs, const Mint& rhs) { return lhs.v == rhs.v; }
29     friend constexpr bool operator!=(const Mint& lhs, const Mint& rhs) { return lhs.v != rhs.v; }
30     friend constexpr bool operator<(const Mint& lhs, const Mint& rhs) { return lhs.v < rhs.v; }

```

```

31 friend constexpr bool operator<=(const Mint& lhs, const Mint& rhs) { return lhs.v <= rhs.v; }
32 friend constexpr bool operator>(const Mint& lhs, const Mint& rhs) { return lhs.v > rhs.v; }
33 friend constexpr bool operator>=(const Mint& lhs, const Mint& rhs) { return lhs.v >= rhs.v; }
34
35 // arithmetic //
36 template <typet>
37 friend constexpr Mint power(Mint a, T n) {
38     Mint ans = 1;
39     while (n) {
40         if (n & 1) ans *= a;
41         a *= a;
42         n >>= 1;
43     }
44     return ans;
45 }
46 friend constexpr Mint inv(const Mint& rhs) { return power(rhs, P - 2); }
47 friend constexpr Mint operator+(const Mint& lhs, const Mint& rhs) {
48     return lhs.val() + rhs.val() < P ? lhs.val() + rhs.val() : lhs.val() - P + rhs.val();
49 }
50 friend constexpr Mint operator-(const Mint& lhs, const Mint& rhs) {
51     return lhs.val() < rhs.val() ? lhs.val() + P - rhs.val() : lhs.val() - rhs.val();
52 }
53 friend constexpr Mint operator*(const Mint& lhs, const Mint& rhs) {
54     return static_cast<LL>(lhs.val()) * rhs.val() % P;
55 }
56 friend constexpr Mint operator/(const Mint& lhs, const Mint& rhs) { return lhs * inv(rhs); }
57 Mint operator+() const { return *this; }
58 Mint operator-() const { return Mint() - *this; }
59 constexpr Mint& operator++() {
60     v++;
61     if (v == P) v = 0;
62     return *this;
63 }
64 constexpr Mint& operator--() {
65     if (v == 0) v = P;
66     v--;
67     return *this;
68 }
69 constexpr Mint& operator++(int) {
70     Mint ans = *this;
71     ++*this;
72     return ans;
73 }
74 constexpr Mint& operator--(int) {
75     Mint ans = *this;
76     --*this;
77     return ans;
78 }
79 constexpr Mint& operator+=(const Mint& rhs) {
80     v = v + rhs;
81     return *this;
82 }
83 constexpr Mint& operator-=(const Mint& rhs) {
84     v = v - rhs;
85     return *this;
86 }
87 constexpr Mint& operator*=(const Mint& rhs) {
88     v = v * rhs;
89     return *this;
90 }
91 constexpr Mint& operator/=(const Mint& rhs) {
92     v = v / rhs;
93     return *this;
94 }
95 };
96 using Z = Mint<998244353>;

```

## 2 Shell Scripts

### 2.1 checker.sh

Linux version.

```

1  #!/bin/bash
2
3  cd "$1"
4
5  g++ -o main -O2 -std=c++17 -DLOCAL main.cpp -ftrapv -fsanitize=address,undefined
6
7  for input in *.in; do
8      output=${input%.*}.out
9      answer=${input%.*}.ans
10
11      ./main < $input > $output
12
13      echo "case ${input%.*}: "
14      echo "My: "
15      cat $output
16      echo "Answer: "
17      cat $answer
18
19  done

```

Windows version.

```

1  @echo off
2
3  cd %1
4
5  del .\main.exe
6
7  g++ -o main.exe main.cpp -DLOCAL -std=c++17 -ftrapv
8
9  for %%i in (*.in) do (
10     main.exe < %%i > %%~ni.out
11     echo case %%~ni:
12     echo My:
13     type %%~ni.out
14     echo Answer:
15     type %%~ni.ans
16 )
17
18 cd ../shell

```

## 3 data structure

### 3.1 cartesian tree

### 3.2 segment tree

```

1  /* segment tree */
2  struct Info {
3      /* 重载 operator+ */
4  };
5
6  struct Tag {
7      /* 重载 operator== */
8  };
9
10 void infoApply(Info& a, int l, int r, const Tag& tag) {}
11
12 void tagApply(Tag& a, int l, int r, const Tag& tag) {}
13
14 template <class Info, class Tag>
15 class segTree {
16 #define ls i << 1
17 #define rs i << 1 | 1
18 #define mid ((l + r) >> 1)
19 #define lson ls, l, mid

```

```

20 #define rson rs, mid + 1, r
21
22 int n;
23 std::vector<Info> info;
24 std::vector<Tag> tag;
25
26 public:
27 segTree(const std::vector<Info>& init) : n(init.size() - 1) {
28     assert(n > 0);
29     info.resize(4 << std::lg(n));
30     tag.resize(4 << std::lg(n));
31     auto build = [&](auto dfs, int i, int l, int r) {
32         if (l == r) {
33             info[i] = init[l];
34             return;
35         }
36         dfs(dfs, lson);
37         dfs(dfs, rson);
38         push_up(i);
39     };
40     build(build, 1, 1, n);
41 }
42
43 private:
44 void push_up(int i) { info[i] = info[ls] + info[rs]; }
45
46
47 template <class... T>
48 void apply(int i, int l, int r, const T&... val) {
49     ::infoApply(info[i], l, r, val...);
50     ::tagApply(tag[i], l, r, val...);
51 }
52
53 void push_down(int i, int l, int r) {
54     if (tag[i] == Tag{}) return;
55     apply(lson, tag[i]);
56     apply(rson, tag[i]);
57     tag[i] = {};
58 }
59
60 public:
61 template <class... T>
62 void rangeApply(int ql, int qr, const T&... val) {
63     auto dfs = [&](auto dfs, int i, int l, int r) {
64         if (qr < l or r < ql) return;
65         if (ql <= l and r <= qr) {
66             apply(i, l, r, val...);
67             return;
68         }
69         push_down(i, l, r);
70         dfs(dfs, lson);
71         dfs(dfs, rson);
72         push_up(i);
73     };
74     dfs(dfs, 1, 1, n);
75 }
76
77 Info rangeAsk(int ql, int qr) {
78     Info res{};
79     auto dfs = [&](auto dfs, int i, int l, int r) {
80         if (qr < l or r < ql) return;
81         if (ql <= l and r <= qr) {
82             res = res + info[i];
83             return;
84         }
85         push_down(i, l, r);
86         dfs(dfs, lson);
87         dfs(dfs, rson);
88     };
89     dfs(dfs, 1, 1, n);
90     return res;
91 }
92
93 #undef rson
94 #undef lson
95 #undef mid
96 #undef rs
97 #undef ls
98 };
99

```

```

1  /* Problem: P3369 【模板】普通平衡树 */
2  struct node {
3      int id, l, r;

```



```

4     int ls, rs;
5     int sum;
6
7     node(int _id, int _l, int _r) : id(_id), l(_l), r(_r) {
8         ls = rs = 0;
9         sum = 0;
10    }
11 };
12
13 /* segment tree */
14 int idx = 1;
15 std::vector<node> tree = {node{0, 0, 0}};
16
17 auto new_node = [&](int l, int r) -> int {
18     tree.push_back(node(idx, l, r));
19     return idx++;
20 };
21
22 auto push_up = [&](int u) -> void {
23     tree[u].sum = 0;
24     if (tree[u].ls) tree[u].sum += tree[tree[u].ls].sum;
25     if (tree[u].rs) tree[u].sum += tree[tree[u].rs].sum;
26 };
27
28 auto build = [&]() { new_node(-inf, inf); };
29
30 std::function<void(int, int, int, int)> insert = [&](int u, int l, int r, int x) {
31     if (l == r) {
32         tree[u].sum++;
33         return;
34     }
35     int mid = (l + r - 1) / 2;
36     if (x <= mid) {
37         if (!tree[u].ls) tree[u].ls = new_node(l, mid);
38         insert(tree[u].ls, l, mid, x);
39     } else {
40         if (!tree[u].rs) tree[u].rs = new_node(mid + 1, r);
41         insert(tree[u].rs, mid + 1, r, x);
42     }
43     push_up(u);
44 };
45
46 std::function<void(int, int, int, int)> remove = [&](int u, int l, int r, int x) {
47     if (l == r) {
48         if (tree[u].sum) tree[u].sum--;
49         return;
50     }
51     int mid = (l + r - 1) / 2;
52     if (x <= mid) {
53         if (!tree[u].ls) return;
54         remove(tree[u].ls, l, mid, x);
55     } else {
56         if (!tree[u].rs) return;
57         remove(tree[u].rs, mid + 1, r, x);
58     }
59     push_up(u);
60 };
61
62 std::function<int(int, int, int, int)> get_rank_by_key = [&](int u, int l, int r, int x) -> int {
63     if (l == r) {
64         return 1;
65     }
66     int mid = (l + r - 1) / 2;
67     int ans = 0;
68     if (x <= mid) {
69         if (!tree[u].ls) return 1;
70         ans = get_rank_by_key(tree[u].ls, l, mid, x);
71     } else {
72         if (!tree[u].rs) return tree[tree[u].ls].sum + 1;
73         if (!tree[u].ls) {
74             ans = get_rank_by_key(tree[u].rs, mid + 1, r, x);
75         } else {
76             ans = get_rank_by_key(tree[u].rs, mid + 1, r, x) + tree[tree[u].ls].sum;
77         }
78     }
79     return ans;
80 };
81
82 std::function<int(int, int, int, int)> get_key_by_rank = [&](int u, int l, int r, int x) -> int {
83     if (l == r) {
84         return l;
85     }
86     int mid = (l + r - 1) / 2;
87     if (tree[u].ls) {
88         if (x <= tree[tree[u].ls].sum) {
89             return get_key_by_rank(tree[u].ls, l, mid, x);
90         } else {

```

```

91         return get_key_by_rank(tree[u].rs, mid + 1, r, x - tree[tree[u].ls].sum);
92     }
93 } else {
94     return get_key_by_rank(tree[u].rs, mid + 1, r, x);
95 }
96 };
97
98 std::function<int(int)> get_prev = [&](int x) -> int {
99     int rank = get_rank_by_key(1, -inf, inf, x) - 1;
100     debug(rank);
101     return get_key_by_rank(1, -inf, inf, rank);
102 };
103
104 std::function<int(int)> get_next = [&](int x) -> int {
105     debug(x + 1);
106     int rank = get_rank_by_key(1, -inf, inf, x + 1);
107     debug(rank);
108     return get_key_by_rank(1, -inf, inf, rank);
109 };

```

```

1  /* Problem: P4556 [Vani有约会]雨天的尾巴 / 【模板】线段树合并 */
2  struct node {
3      int l, r, id;
4      int ls, rs;
5      int cnt, ans;
6
7      node(int _id, int _l, int _r) : id(_id), l(_l), r(_r) {
8          ls = rs = 0;
9          cnt = ans = 0;
10     }
11 };
12
13 int main() {
14     std::ios::sync_with_stdio(false);
15     std::cin.tie(0);
16     std::cout.tie(0);
17
18     int n, m;
19     std::cin >> n >> m;
20     vvi e(n + 1);
21     vi ans(n + 1);
22     for (int i = 1; i < n; i++) {
23         int u, v;
24         std::cin >> u >> v;
25         e[u].push_back(v);
26         e[v].push_back(u);
27     }
28
29     // Segment tree //
30     int idx = 1;
31     vi rt(n + 1);
32     std::vector<node> tree = {node{0, 0, 0}};
33
34     auto new_node = [&](int l, int r) -> int {
35         tree.push_back(node(idx, l, r));
36         return idx++;
37     };
38
39     auto push_up = [&](int u) -> void {
40         if (!tree[u].ls) {
41             tree[u].cnt = tree[tree[u].rs].cnt;
42             tree[u].ans = tree[tree[u].rs].ans;
43         } else if (!tree[u].rs) {
44             tree[u].cnt = tree[tree[u].ls].cnt;
45             tree[u].ans = tree[tree[u].ls].ans;
46         } else {
47             if (tree[tree[u].rs].cnt > tree[tree[u].ls].cnt) {
48                 tree[u].cnt = tree[tree[u].rs].cnt;
49                 tree[u].ans = tree[tree[u].rs].ans;
50             } else {
51                 tree[u].cnt = tree[tree[u].ls].cnt;
52                 tree[u].ans = tree[tree[u].ls].ans;
53             }
54         }
55     };
56
57     std::function<void(int, int, int, int, int)> modify = [&](int u, int l, int r, int x, int k) {
58         if (l == r) {
59             tree[u].cnt += k;
60             tree[u].ans = 1;
61             return;
62         }
63         int mid = (l + r) >> 1;
64         if (x <= mid) {
65             if (!tree[u].ls) tree[u].ls = new_node(l, mid);
66             modify(tree[u].ls, l, mid, x, k);

```

```

67     } else {
68         if (!tree[u].rs) tree[u].rs = new_node(mid + 1, r);
69         modify(tree[u].rs, mid + 1, r, x, k);
70     }
71     push_up(u);
72 };
73
74 std::function<int(int, int, int, int)> merge = [&](int u, int v, int l, int r) -> int {
75     // v 的信息传递给 u //
76     if (!u) return v;
77     if (!v) return u;
78     if (l == r) {
79         tree[u].cnt += tree[v].cnt;
80         return u;
81     }
82     int mid = (l + r) >> 1;
83     tree[u].ls = merge(tree[u].ls, tree[v].ls, l, mid);
84     tree[u].rs = merge(tree[u].rs, tree[v].rs, mid + 1, r);
85     push_up(u);
86     return u;
87 };
88
89 // LCA //
90
91 for (int i = 1; i <= n; i++) {
92     rt[i] = idx;
93     new_node(1, 100000);
94 }
95
96 for (int i = 1; i <= m; i++) {
97     int u, v, w;
98     std::cin >> u >> v >> w;
99     int lca = LCA(u, v);
100     modify(rt[u], 1, 100000, w, 1);
101     modify(rt[v], 1, 100000, w, 1);
102     modify(rt[lca], 1, 100000, w, -1);
103     if (father[lca][0]) {
104         modify(rt[father[lca][0]], 1, 100000, w, -1);
105     }
106 }
107
108 // dfs //
109 std::function<void(int, int)> Dfs = [&](int u, int fa) {
110     for (auto v : e[u]) {
111         if (v == fa) continue;
112         Dfs(v, u);
113         merge(rt[u], rt[v], 1, 100000);
114     }
115     ans[u] = tree[rt[u]].ans;
116     if (tree[rt[u]].cnt == 0) ans[u] = 0;
117 };
118
119 Dfs(1, 0);
120
121 for (int i = 1; i <= n; i++) {
122     std::cout << ans[i] << endl;
123 }
124
125 return 0;
126 }

```

### 3.3 hjt segment tree

```

1  /* 洛谷 P3919 【模板】可持久化线段树 1（可持久化数组）*/
2  struct node {
3      int l, r, key;
4  };
5
6  int main() {
7      std::ios::sync_with_stdio(false);
8      std::cin.tie(0);
9      std::cout.tie(0);
10
11     int n, m;
12     std::cin >> n >> m;
13     vi a(n + 1);
14     for (int i = 1; i <= n; i++) {
15         std::cin >> a[i];
16     }
17
18     // hjt segment tree //
19     int idx = 0;

```

```

20 vi root(m + 1);
21 std::vector<node> tr(n * 25);
22
23 std::function<int(int, int)> build = [&](int l, int r) -> int {
24     int p = ++idx;
25     if (l == r) {
26         tr[p].key = a[l];
27         return p;
28     }
29     int mid = (l + r) >> 1;
30     tr[p].l = build(l, mid);
31     tr[p].r = build(mid + 1, r);
32     return p;
33 };
34
35 std::function<int(int, int, int, int, int)> modify = [&](int p, int l, int r, int k,
36                                                         int x) -> int {
37     int q = ++idx;
38     tr[q].l = tr[p].l, tr[q].r = tr[p].r;
39     if (tr[q].l == tr[q].r) {
40         tr[q].key = x;
41         return q;
42     }
43     int mid = (l + r) >> 1;
44     if (k <= mid) {
45         tr[q].l = modify(tr[q].l, l, mid, k, x);
46     } else {
47         tr[q].r = modify(tr[q].r, mid + 1, r, k, x);
48     }
49     return q;
50 };
51
52 std::function<int(int, int, int, int)> query = [&](int p, int l, int r, int k) -> int {
53     if (tr[p].l == tr[p].r) {
54         return tr[p].key;
55     }
56     int mid = (l + r) >> 1;
57     if (k <= mid) {
58         return query(tr[p].l, l, mid, k);
59     } else {
60         return query(tr[p].r, mid + 1, r, k);
61     }
62 };
63 root[0] = build(1, n);
64
65 for (int i = 1; i <= m; i++) {
66     int op, ver, k, x;
67     std::cin >> ver >> op;
68     if (op == 1) {
69         std::cin >> k >> x;
70         root[i] = modify(root[ver], 1, n, k, x);
71     } else {
72         std::cin >> k;
73         root[i] = root[ver];
74         std::cout << query(root[ver], 1, n, k) << endl;
75     }
76 }
77 return 0;
78 }

```

```

1  /* 洛谷P3834 【模板】可持久化线段树 2 */
2
3  struct node {
4      int l, r, cnt;
5  };
6
7  int main() {
8      std::ios::sync_with_stdio(false);
9      std::cin.tie(0);
10     std::cout.tie(0);
11
12     int n, m;
13     std::cin >> n >> m;
14     vi a(n + 1), v;
15     for (int i = 1; i <= n; i++) {
16         std::cin >> a[i];
17         v.push_back(a[i]);
18     }
19     std::sort(all(v));
20     v.erase(unique(all(v)), v.end());
21     auto find = [&](int x) -> int { return std::lower_bound(all(v), x) - v.begin() + 1; };
22
23     // hjt segment tree //
24     std::vector<node> tr(n * 25);
25     vi root(n + 1);
26     int idx = 0;

```

```

27
28 std::function<int(int, int)> build = [&](int l, int r) -> int {
29     int p = ++idx;
30     if (l == r) return p;
31     int mid = (l + r) >> 1;
32     tr[p].l = build(l, mid), tr[p].r = build(mid + 1, r);
33     return p;
34 };
35
36 std::function<int(int, int, int, int)> modify = [&](int p, int l, int r, int x) -> int {
37     int q = ++idx;
38     tr[q] = tr[p];
39     if (tr[q].l == tr[q].r) {
40         tr[q].cnt++;
41         return q;
42     }
43     int mid = (l + r) >> 1;
44     if (x <= mid) {
45         tr[q].l = modify(tr[q].l, l, mid, x);
46     } else {
47         tr[q].r = modify(tr[q].r, mid + 1, r, x);
48     }
49     tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
50     return q;
51 };
52
53 std::function<int(int, int, int, int, int)> query = [&](int p, int q, int l, int r,
54                                                     int x) -> int {
55     if (l == r) return l;
56     int cnt = tr[tr[p].l].cnt - tr[tr[q].l].cnt;
57     int mid = (l + r) >> 1;
58     if (x <= cnt) {
59         return query(tr[p].l, tr[q].l, l, mid, x);
60     } else {
61         return query(tr[p].r, tr[q].r, mid + 1, r, x - cnt);
62     }
63 };
64 root[0] = build(1, v.size());
65
66 for (int i = 1; i <= n; i++) {
67     root[i] = modify(root[i - 1], 1, v.size(), find(a[i]));
68 }
69 for (int i = 1; i <= m; i++) {
70     int l, r, k;
71     std::cin >> l >> r >> k;
72     std::cout << v[query(root[r], root[l - 1], 1, v.size(), k) - 1] << endl;
73 }
74 return 0;
75 }

```

### 3.4 LiChao tree

### 3.5 treap

```

1  /* fhq treap */
2  struct node {
3      node *ch[2];
4      int key, val;
5      int cnt, size;
6
7      node(int _key) : key(_key), cnt(1), size(1) {
8          ch[0] = ch[1] = nullptr;
9          val = rand();
10     }
11
12     // node(node *_node) {
13     //     key = _node->key, val = _node->val, cnt = _node->cnt, size = _node->size;
14     // }
15
16     inline void push_up() {
17         size = cnt;
18         if (ch[0] != nullptr) size += ch[0]->size;
19         if (ch[1] != nullptr) size += ch[1]->size;
20     }
21 };
22
23 struct treap {
24     #define _2 second.first
25     #define _3 second.second
26
27     node *root;

```

```

28
29
30 pair<node *, node *> split(node *p, int key) {
31     if (p == nullptr) return {nullptr, nullptr};
32     if (p->key <= key) {
33         auto temp = split(p->ch[1], key);
34         p->ch[1] = temp.first;
35         p->push_up();
36         return {p, temp.second};
37     } else {
38         auto temp = split(p->ch[0], key);
39         p->ch[0] = temp.second;
40         p->push_up();
41         return {temp.first, p};
42     }
43 }
44
45 pair<node *, pair<node *, node *> > split_by_rank(node *p, int rank) {
46     if (p == nullptr) return {nullptr, {nullptr, nullptr}};
47     int ls_size = p->ch[0] == nullptr ? 0 : p->ch[0]->size;
48     if (rank <= ls_size) {
49         auto temp = split_by_rank(p->ch[0], rank);
50         p->ch[0] = temp._3;
51         p->push_up();
52         return {temp.first, {temp._2, p}};
53     } else if (rank <= ls_size + p->cnt) {
54         node *lt = p->ch[0];
55         node *rt = p->ch[1];
56         p->ch[0] = p->ch[1] = nullptr;
57         return {lt, {p, rt}};
58     } else {
59         auto temp = split_by_rank(p->ch[1], rank - ls_size - p->cnt);
60         p->ch[1] = temp.first;
61         p->push_up();
62         return {p, {temp._2, temp._3}};
63     }
64 }
65
66 node *merge(node *u, node *v) {
67     if (u == nullptr && v == nullptr) return nullptr;
68     if (u != nullptr && v == nullptr) return u;
69     if (v != nullptr && u == nullptr) return v;
70     if (u->val < v->val) {
71         u->ch[1] = merge(u->ch[1], v);
72         u->push_up();
73         return u;
74     } else {
75         v->ch[0] = merge(u, v->ch[0]);
76         v->push_up();
77         return v;
78     }
79 }
80
81 void insert(int key) {
82     auto temp = split(root, key);
83     auto l_tr = split(temp.first, key - 1);
84     node *new_node;
85     if (l_tr.second == nullptr) {
86         new_node = new node(key);
87     } else {
88         l_tr.second->cnt++;
89         l_tr.second->push_up();
90     }
91     node *l_tr_combined = merge(l_tr.first, l_tr.second == nullptr ? new_node : l_tr.second);
92     root = merge(l_tr_combined, temp.second);
93 }
94
95 void remove(int key) {
96     auto temp = split(root, key);
97     auto l_tr = split(temp.first, key - 1);
98     if (l_tr.second->cnt > 1) {
99         l_tr.second->cnt--;
100         l_tr.second->push_up();
101         l_tr.first = merge(l_tr.first, l_tr.second);
102     } else {
103         if (temp.first == l_tr.second) temp.first = nullptr;
104         delete l_tr.second;
105         l_tr.second = nullptr;
106     }
107     root = merge(l_tr.first, temp.second);
108 }
109
110 int get_rank_by_key(node *p, int key) {
111     auto temp = split(p, key - 1);
112     int ret = (temp.first == nullptr ? 0 : temp.first->size) + 1;
113     root = merge(temp.first, temp.second);
114     return ret;
115 }

```

```

115
116 int get_key_by_rank(node *p, int rank) {
117     auto temp = split_by_rank(p, rank);
118     int ret = temp._2->key;
119     root = merge(temp.first, merge(temp._2, temp._3));
120     return ret;
121 }
122
123 int get_prev(int key) {
124     auto temp = split(root, key - 1);
125     int ret = get_key_by_rank(temp.first, temp.first->size);
126     root = merge(temp.first, temp.second);
127     return ret;
128 }
129
130 int get_nex(int key) {
131     auto temp = split(root, key);
132     int ret = get_key_by_rank(temp.second, 1);
133     root = merge(temp.first, temp.second);
134     return ret;
135 }
136 } tr;

```

### 3.6 splay

```

1  /* 洛谷 P3391 【模板】文艺平衡树 */
2  struct node {
3      int ch[2], fa, key;
4      int siz, flag;
5
6      void init(int _fa, int _key) { fa = _fa, key = _key, siz = 1; }
7  };
8
9  struct splay {
10     node tr[N];
11     int n, root, idx;
12
13     bool get(int u) { return u == tr[tr[u].fa].ch[1]; }
14
15     void pushup(int u) { tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + 1; }
16
17     void pushdown(int u) {
18         if (tr[u].flag) {
19             std::swap(tr[u].ch[0], tr[u].ch[1]);
20             tr[tr[u].ch[0]].flag ^= 1, tr[tr[u].ch[1]].flag ^= 1;
21             tr[u].flag = 0;
22         }
23     }
24
25     void rotate(int x) {
26         int y = tr[x].fa, z = tr[y].fa;
27         int op = get(x);
28         tr[y].ch[op] = tr[x].ch[op ^ 1];
29         if (tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
30         tr[x].ch[op ^ 1] = y;
31         tr[y].fa = x, tr[x].fa = z;
32         if (z) tr[z].ch[y == tr[z].ch[1]] = x;
33         pushup(y), pushup(x);
34     }
35
36     void opt(int u, int k) {
37         for (int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)) {
38             if (tr[f].fa != k) rotate(get(u) == get(f) ? f : u);
39         }
40         if (k == 0) root = u;
41     }
42
43     void output(int u) {
44         pushdown(u);
45         if (tr[u].ch[0]) output(tr[u].ch[0]);
46         if (tr[u].key >= 1 && tr[u].key <= n) {
47             std::cout << tr[u].key << ' ';
48         }
49         if (tr[u].ch[1]) output(tr[u].ch[1]);
50     }
51
52     void insert(int key) {
53         idx++;
54         tr[idx].ch[0] = root;
55         tr[idx].init(0, key);
56         tr[root].fa = idx;
57         root = idx;

```

```

58     pushup(idk);
59 }
60
61 int kth(int k) {
62     int u = root;
63     while (1) {
64         pushdown(u);
65         if (tr[u].ch[0] && k <= tr[tr[u].ch[0]].siz) {
66             u = tr[u].ch[0];
67         } else {
68             k -= tr[tr[u].ch[0]].siz + 1;
69             if (k <= 0) {
70                 opt(u, 0);
71                 return u;
72             } else {
73                 u = tr[u].ch[1];
74             }
75         }
76     }
77 }
78
79 } splay;
80
81 int n, m, l, r;
82 int main() {
83     std::ios::sync_with_stdio(false);
84     std::cin.tie(0);
85     std::cout.tie(0);
86
87     std::cin >> n >> m;
88     splay.n = n;
89     splay.insert(-inf);
90     for (int i = 1; i <= n; i++) splay.insert(i);
91     splay.insert(inf);
92     for (int i = 1; i <= m; i++) {
93         std::cin >> l >> r;
94         l = splay.kth(l), r = splay.kth(r + 2);
95         splay.opt(l, 0), splay.opt(r, 1);
96         splay.tr[splay.tr[r].ch[0]].flag ^= 1;
97     }
98     splay.output(splay.root);
99     return 0;
100 }

```

```

1  /* 洛谷 P3369 【模板】普通平衡树 */
2
3  struct node {
4      int ch[2], fa, key, siz, cnt;
5
6      void init(int _fa, int _key) { fa = _fa, key = _key, siz = cnt = 1; }
7
8      void clear() { ch[0] = ch[1] = fa = key = siz = cnt = 0; }
9  };
10
11  struct splay {
12      node tr[N];
13      int n, root, idk;
14
15      bool get(int u) { return u == tr[tr[u].fa].ch[1]; }
16
17      void pushup(int u) { tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + tr[u].cnt; }
18
19      void rotate(int x) {
20          int y = tr[x].fa, z = tr[y].fa;
21          int op = get(x);
22          tr[y].ch[op] = tr[x].ch[op ^ 1];
23          if (tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
24          tr[x].ch[op ^ 1] = y;
25          tr[y].fa = x, tr[x].fa = z;
26          if (z) tr[z].ch[y == tr[z].ch[1]] = x;
27          pushup(y), pushup(x);
28      }
29
30      void opt(int u, int k) {
31          for (int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)) {
32              if (tr[f].fa != k) {
33                  rotate(get(u) == get(f) ? f : u);
34              }
35          }
36          if (k == 0) root = u;
37      }
38
39      void insert(int key) {
40          if (!root) {
41              idk++;
42              tr[idk].init(0, key);

```



```

43         root = idx;
44         return;
45     }
46     int u = root, f = 0;
47     while (1) {
48         if (tr[u].key == key) {
49             tr[u].cnt++;
50             pushup(u), pushup(f);
51             opt(u, 0);
52             break;
53         }
54         f = u, u = tr[u].ch[tr[u].key < key];
55         if (!u) {
56             idx++;
57             tr[idx].init(f, key);
58             tr[f].ch[tr[f].key < key] = idx;
59             pushup(idx), pushup(f);
60             opt(idx, 0);
61             break;
62         }
63     }
64 }
65
66 // 返回节点编号 //
67 int kth(int rank) {
68     int u = root;
69     while (1) {
70         if (tr[u].ch[0] && rank <= tr[tr[u].ch[0]].siz) {
71             u = tr[u].ch[0];
72         } else {
73             rank -= tr[tr[u].ch[0]].siz + tr[u].cnt;
74             if (rank <= 0) {
75                 opt(u, 0);
76                 return u;
77             } else {
78                 u = tr[u].ch[1];
79             }
80         }
81     }
82 }
83
84 // 返回排名 //
85 int nlt(int key) {
86     int rank = 0, u = root;
87     while (1) {
88         if (tr[u].key > key) {
89             u = tr[u].ch[0];
90         } else {
91             rank += tr[tr[u].ch[0]].siz;
92             if (tr[u].key == key) {
93                 opt(u, 0);
94                 return rank + 1;
95             }
96             rank += tr[u].cnt;
97             if (tr[u].ch[1]) {
98                 u = tr[u].ch[1];
99             } else {
100                 return rank + 1;
101             }
102         }
103     }
104 }
105
106 int get_prev(int key) { return kth(nlt(key) - 1); }
107
108 int get_next(int key) { return kth(nlt(key) + 1); }
109
110 void remove(int key) {
111     nlt(key);
112     if (tr[root].cnt > 1) {
113         tr[root].cnt--;
114         pushup(root);
115         return;
116     }
117     int u = root, l = get_prev(key);
118     tr[tr[u].ch[1]].fa = l;
119     tr[l].ch[1] = tr[u].ch[1];
120     tr[u].clear();
121     pushup(root);
122 }
123
124 void output(int u) {
125     if (tr[u].ch[0]) output(tr[u].ch[0]);
126     std::cout << tr[u].key << ' ';
127     if (tr[u].ch[1]) output(tr[u].ch[1]);
128 }
129

```

```

130 } splay;
131
132 int n, op, x;
133 int main() {
134     std::ios::sync_with_stdio(false);
135     std::cin.tie(0);
136     std::cout.tie(0);
137
138     splay.insert(-inf), splay.insert(inf);
139
140     std::cin >> n;
141     for (int i = 1; i <= n; i++) {
142         std::cin >> op >> x;
143         if (op == 1) {
144             splay.insert(x);
145         } else if (op == 2) {
146             splay.remove(x);
147         } else if (op == 3) {
148             std::cout << splay.nlt(x) - 1 << '\n';
149         } else if (op == 4) {
150             std::cout << splay.tr[splay.kth(x + 1)].key << '\n';
151         } else if (op == 5) {
152             std::cout << splay.tr[splay.get_prev(x)].key << '\n';
153         } else if (op == 6) {
154             std::cout << splay.tr[splay.get_next(x)].key << '\n';
155         }
156     }
157     return 0;
158 }

```

### 3.7 Link Cut Tree

### 3.8 ODT

### 3.9 tree in tree

## 4 string

### 4.1 KMP

```

1 auto getNext = [&](const std::string& s) -> vi {
2     int n = s.length();
3     vi next(n);
4     for (int i = 1; i < n; i++) {
5         int j = next[i - 1];
6         while (j > 0 and s[i] != s[j]) j = next[j - 1];
7         if (s[i] == s[j]) j++;
8         next[i] = j;
9     }
10    return next;
11 };

```

### 4.2 z-function

```

1 auto zFunction = [&](const std::string& s) -> vi {
2     int n = s.size();
3     vi z(n);
4     for (int i = 1, l = 0, r = 0; i < n; i++) {
5         if (i <= r and z[i - l] < r - i + 1) {
6             z[i] = z[i - l];
7         } else {
8             z[i] = std::max(0, r - i + 1);
9             while (z[i] + i < n and s[z[i]] == s[z[i] + i]) z[i]++;
10        }
11        if (z[i] + i - 1 > r) {
12            l = i;
13            r = z[i] + i - 1;
14        }
15    }
16    return z;

```

```
17 |};
```

## 4.3 Manacher

## 4.4 AC automaton

## 4.5 PAM

```
1  /* PAM @ ddl */
2  std::vector<node> tr;
3  std::vector<int> stk;
4  auto newnode = [&](int len) {
5      tr.emplace_back();
6      tr.back().len = len;
7      return (int) tr.size() - 1;
8  };
9  auto PAMinit = [&]() {
10     newnode(0), tr.back().fail = 1;
11     newnode(-1), tr.back().fail = 0;
12     stk.push_back(-1);
13 };
14 PAMinit();
15 auto getfail = [&](int v) {
16     while (stk.end()[-2 - tr[v].len] != stk.back()) {
17         v = tr[v].fail;
18     }
19     return v;
20 };
21 auto insert = [&](int last, int c, int cnt) {
22     stk.emplace_back(c);
23     int x = getfail(last);
24     if (!tr[x].ch[c]) {
25         int u = newnode(tr[x].len + 2);
26         tr[u].fail = tr[getfail(tr[x].fail)].ch[c];
27         tr[x].ch[c] = u;
28         /* tr[u].size = tr[tr[u].fail].size + 1; */
29         /* Can be used to count the number of types of palindromic strings ending at the current
30          * position */
31     }
32     tr[tr[x].ch[c]].size += cnt;
33     return tr[x].ch[c];
34 };
35 auto build = [&]() { /* DP on fail tree */
36     int ans = 0;
37     for (int i = (int) tr.size() - 1; i > 1; i--) {
38         tr[tr[i].fail].size += tr[i].size;
39         /* options */
40     }
41     return ans;
42 };
43 /* PAM */
44 int ans = 0, last = 0;
45 for (int i = 0; i < n; i++) {
46     last = insert(last, s[i] - 'a', 1);
47 }
```

## 4.6 suffix array

```
1  /* suffix array and ST table @ jiangly */
2  auto suffixArray = [&](const std::string& s) {
3      int n = s.length();
4      vi sa(n), rk(n);
5      std::iota(all(sa), 0);
6      std::sort(all(sa), [&](int a, int b) { return s[a] < s[b]; });
7      rk[sa[0]] = 0;
8      for (int i = 1; i < n; ++i) {
9          rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
10     }
11     int k = 1;
12     vi tmp(n), cnt(n);
```

```

13 tmp.reserve(n);
14 while (rk[sa[n - 1]] < n - 1) {
15     tmp.clear();
16     for (int i = 0; i < k; ++i) tmp.push_back(n - k + i);
17     for (const auto& i : sa) {
18         if (i >= k) tmp.push_back(i - k);
19     }
20     std::fill(all(cnt), 0);
21     for (int i = 0; i < n; i++) cnt[rk[i]]++;
22     for (int i = 1; i < n; i++) cnt[i] += cnt[i - 1];
23     for (int i = n - 1; i >= 0; i--) sa[--cnt[rk[tmp[i]]]] = tmp[i];
24     std::swap(rk, tmp);
25     rk[sa[0]] = 0;
26     for (int i = 1; i < n; i++) {
27         rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] or sa[i - 1] + k == n or
28                                     tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
29     }
30     k *= 2;
31 }
32 vi height(n);
33 for (int i = 0, j = 0; i < n; ++i) {
34     if (rk[i] == 0) continue;
35     if (j) --j;
36     while (s[i + j] == s[sa[rk[i] - 1] + j]) ++j;
37     height[rk[i]] = j;
38 }
39 return std::make_tuple(sa, rk, height);
40 };
41 auto [sa, rk, height] = suffixArray(s);
42 vvi f(n, vi(30, inf));
43 vi Log2(n);
44 auto init = [&]() -> void {
45     for (int i = 0; i < n; i++) {
46         f[i][0] = height[i];
47         if (i > 1) Log2[i] = Log2[i / 2] + 1;
48     };
49     int t = Log2.back();
50     for (int j = 1; j <= t; j++) {
51         for (int i = 0; i <= n - (1 << j); i++) {
52             f[i][j] = std::min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
53         }
54     }
55 };
56 init();
57 auto query = [&](int l, int r) -> int {
58     int t = Log2[r - l + 1];
59     return std::min(f[l][t], f[r - (1 << t) + 1][t]);
60 };
61 auto lcp = [&](int i, int j) {
62     i = rk[i], j = rk[j];
63     if (i > j) std::swap(i, j);
64     return query(i + 1, j);
65 };

```

## 5 graph

- 5.1 shortest path
- 5.2 minimum spanning tree
- 5.3 SCC
- 5.4 DCC
- 5.5 2-SAT
- 5.6 minimum ring
- 5.7 tree - center of gravity
- 5.8 tree - DSU on tree
- 5.9 tree - AHU
- 5.10 tree - LCA
- 5.11 tree - LLD
- 5.12 tree - HLD
- 5.13 tree - virtual tree
- 5.14 tree - pseudo tree
- 5.15 tree - prufer sequence
- 5.16 tree - divide and conquer on tree
- 5.17 network flow - maximal flow
- 5.18 network flow - minimal cost flow
- 5.19 network flow - minimal cut
- 5.20 matching - matching on bipartite graph
- 5.21 matching - general on general graph

## 6 math - number theory

## 7 math - polynomial

## 8 math - numerical analysis