板子

update: 2025/11/7

Contents

1 KMP

1.1 Fail

```cpp
// input: 0-based, output: 1-based
auto getFail (const std::string& s) {
  int n = s.size();
  std::vector<int> fail(n + 1);
  for (int i = 1, j = 0; i < n; ++i) {
    while (j && s[i] != s[j]) j = fail[j];
    fail[i + 1] = j += (s[i] == s[j]);
  }
  return fail;
}
```

1.2 Trans

```cpp
// input: 0-based, output: 1-based
auto getTrans(const std::string& s) {
  int n = s.size();
  auto fail = getFail(s);

  std::vector<int> trans(n + 1);
  for (int i = 1, j = 0; i < n; ++i) {
    while (j && s[i] != s[j]) j = fail[j];
    j += s[i] == s[j];
    while (2 * j > i + 1) j = fail[j];
    trans[i + 1] = j;
  }
  return trans;
}
```

1.3 occurrence

```cpp
// input: 0-based, output: 0-based
auto occurrence(const std::string& s, const std::string& t) {
  int n = s.size(), m = t.size();
  auto fail = getFail(t);

  std::vector<int> occur;
  for (int i = 0, j = 0; i < n; ++i) {
    while (j && s[i] != t[j]) j = fail[j];
    j += s[i] == t[j];
    if (j == m) {
      occur.push_back(i - m + 1);
      j = fail[j];
    }
  }
  return occur;
}
```

2 Manacher

2.1

```cpp
// input: 0-based, idx(s[i]) = 2i (i: 1-based)
std::vector<int> Manacher(const std::string& t) {
  std::string s = "#";
```

```
  for (auto& ch : t) {
    s += '$', s += ch;
  } s += '$';

  int n = s.size() - 1;
  std::vector<int> d(n + 1);
  for (int i = 1, j = 1; i <= n; ++i) {
    d[i] = i < j + d[j] ? std::min(d[2 * j - i], j + d[j] - i) : 1;
    while (i + d[i] <= n && i - d[i] >= 1 && s[i - d[i]] == s[i + d[i]]) ++d[i];
    if (i + d[i] > j + d[j]) j = i;
  }

  return d;
}
```

2.2

题意：找最长的子串满足存在一个字符串 S 使得该子串可以被表示成 S+$\overline{\text{S}}$+S+$\overline{\text{S}}$. 显然一个字符串只有 $O(n)$ 个本质不同的回文串，暴力 check 即可

```
// input: 0-based, idx(s[i]) = 2i (i: 1-based)
int Manacher(const std::string& t) {
  std::string s = "#";
  for (auto& ch : t) {
    s += '$', s += ch;
  } s += '$';

  int n = s.size() - 1, ans = 0;
  std::vector<int> d(n + 1);


  for (int i = 1, j = 1; i <= n; ++i) {
    d[i] = i < j + d[j] ? std::min(d[2 * j - i], j + d[j] - i) : 1;
    while (i + d[i] <= n && i - d[i] >= 1 && s[i - d[i]] == s[i + d[i]]) ++d[i];
    if (i + d[i] > j + d[j]) {
      if (s[i] == '$') {
        for (int k = j + d[j]; k < i + d[i]; ++k) {
          if (s[k] == '$') continue;
          int l = (2 * i - k) >> 1, r = k >> 1;
          if ((r - l + 1) % 4 != 0) continue;
          int x = i >> 1;
          if (d[l + x] - 1 >= (r - l + 1) / 2) {
            ans = std::max(ans, r - l + 1);
          }
        }
      }
      j = i;
    }
  }

  return ans;
}
```

3 Z Function

```
// input: 0-based, output: 0-based
auto ZFunction(const std::string& s) {
```

```
  int n = s.size();
  std::vector<int> z(n + 1);
  z[0] = n;
  for (int i = 1, j = 1; i < n; ++i) {
    z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
    while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
    if (i + z[i] > j + z[j]) j = i;
  }
  return z;
}
```

4 Suffix Array

要学会使用 sa,rk,height 数组，多观察性质，转化成 sa,rk,height 能做的

```
// input: 0-based, output: 1-based
auto SuffixArray(const std::string& s) {
  int n = s.size();
  std::vector<int> sa(n + 1), rk(n + 1);
  std::iota(sa.begin() + 1, sa.end(), 1);
  std::sort(sa.begin() + 1, sa.end(), [&](int x, int y) {
    return s[x - 1] < s[y - 1];
  });

  rk[sa[1]] = 1;
  for (int i = 1; i < n; ++i) {
    rk[sa[i + 1]] = rk[sa[i]] + (s[sa[i + 1] - 1] != s[sa[i] - 1]);
  }

  std::vector<int> tmp(n + 1), cnt(n + 1);
  for (int k = 1; rk[sa[n]] != n; k <<= 1) {
    for (int i = n - k + 1, j = 1; i <= n; ++i, ++j) {
      tmp[j] = i;
    }
    for (int i = 1, j = k; i <= n; ++i) {
      if (sa[i] <= k) continue;
      tmp[++j] = sa[i] - k;
    }

    for (int i = 1; i <= n; ++i) {
      cnt[rk[i]]++;
    }
    for (int i = 1; i < rk[sa[n]]; ++i) {
      cnt[i + 1] += cnt[i];
    }
    for (int i = n; i >= 1; --i) {
      sa[cnt[rk[tmp[i]]]--] = tmp[i];
    }

    std::swap(rk, tmp);
    rk[sa[1]] = 1, cnt[tmp[sa[n]]] = 0;
    for (int i = 1; i < n; ++i) {
      cnt[tmp[sa[i]]] = 0;

      rk[sa[i + 1]] = rk[sa[i]] + (
        tmp[sa[i + 1]] != tmp[sa[i]] ||
```

```
        sa[i] + k - 1 == n ||
        tmp[sa[i + 1] + k] != tmp[sa[i] + k]
      );
    }
  }

  std::vector<int> height(n + 1);
  for (int i = 1, lcp = 0; i <= n; ++i) {
    if (rk[i] == 1) continue;
    if (lcp != 0) lcp--;
    while (
      i + lcp <= n &&
      sa[rk[i] - 1] + lcp <= n &&
      s[i + lcp - 1] == s[sa[rk[i] - 1] + lcp - 1]
    ) ++lcp;
    height[rk[i]] = lcp;
  }

  return std::tuple {
    std::move(sa),
    std::move(rk),
    std::move(height)
  };
}
```

4.1 求 Longest Common Substring

```
int n = s.size(), m = t.size();
auto [sa, rk, height] = SuffixArray(s + '$' + t);

std::array<int, 3> ans { 0, 0, 0 };
for (int i = 1; i <= n + m; ++i) {
  int x = sa[i], y = sa[i + 1];
  int len = height[i + 1];
  if (len <= ans[0]) continue;
  if (x <= n && y >= n + 2) {
    ans = { len, x - 1, y - n - 2 };
  }
  if (y <= n && x >= n + 2) {
    ans = { len, y - 1, x - n - 2 };
  }
}
```

5 Suffix Automaton

```
// Node: 1-based "" 为 1 号节点
struct SAM {
  static constexpr int N = 26;
  struct Node {
    int len;
    int link;
    std::array<int, N> next;
    Node() : len(), link(), next() {}
  };

  i64 substr;
```

```
std::vector<Node> t;

SAM (int n = 0) {
  t.reserve(n);
  t.assign(2, Node());
  t[0].next.fill(1);
  t[0].len = -1;
  substr = 0;
}

int newNode() {
  t.push_back();
  return t.size() - 1;
}

int extend(int p, int c) {
  int cur = newNode();
  t[cur].len = t[p].len + 1;

  while (t[p].next[c] == 0) {
    t[p].next[c] = cur;
    p = t[p].link;
  }


  int q = t[p].next[c];
  if (t[q].len == t[p].len + 1) {
    t[cur].link = q;
  } else {
    int r = newNode();
    t[r].len = t[p].len + 1;
    t[r].link = t[q].link;
    t[r].next = t[q].next;
    t[q].link = r;
    while (t[p].next[c] == q) {
      t[p].next[c] = r;
      p = t[p].link;
    }
    t[cur].link = r;
  }

  substr += t[cur].len - t[t[cur].link].len;
  return cur;
}

int len(int p)          const {return t[p].len; }
int link(int p)         const {return t[p].link; }
int next(int p, int x) const {return t[p].next[x]; }
int size()              const {return t.size(); }
i64 count ()            const {return substr; }

// [ SAM 节点的个数 (不含空节点), 后缀树 ]
auto getTree() {
  int n = t.size();
  std::vector<std::vector<int>> adj(n);
```

```
    for (int i = 2; i < n; ++i) {
      adj[t[i].link].push_back(i);
    }
    return std::pair { n - 1, std::move(adj) };
  }
};
```

5.1 弦论

计算 kth 子串，t == 1 时多次出现需要多次计算

```
int n = s.size();
SAM sam(n);
vector<int> p(n + 1);
p[0] = 1;
for (int i = 0; i < n; ++i) {
  p[i + 1] = sam.extend(p[i], s[i] - 'a');
}

auto [m, adj] = sam.getTree();
vector<i64> siz(m + 1);
if (t == 1) { // 考虑 endpos.size()
  for (int i = 1; i <= n; ++i) {
    siz[p[i]]++;
  }
  auto dfs = [&](auto &&dfs, int u) -> void {
    for (auto v : adj[u]) {
      dfs(dfs, v);
      siz[u] += siz[v];
    }
  }; dfs(dfs, 1);
} else { // 否则一个集合只算一次
  for (int i = 1; i <= m; ++i) {
    siz[i] = 1;
  }
}
siz[1] = 0;

vector<int> deg(m + 1);
adj.assign(m + 1, {});
for (int u = 1; u <= m; ++u) {
  for (int ch = 0; ch < 26; ++ch) {
    int v = sam.next(u, ch);
    if (v == 0) continue;
    adj[v].push_back(u);
    deg[u]++;
  }
}

i64 substr = 0;
vector<i64> dp = siz;

for (int u = 2; u <= m; ++u) {
  substr += (sam.len(u) - sam.len(sam.link(u))) * siz[u];
}
if (substr < k) {
```

```
  cout << "-1\n";
  return;
}

queue<int> que;
for (int u = 1; u <= m; ++u) {
  if (deg[u] == 0) {
    que.push(u);
  }
}

while (!que.empty()) {
  int u = que.front();
  que.pop();
  for (auto v : adj[u]) {
    dp[v] += dp[u];
    if (--deg[v] == 0) {
      que.push(v);
    }
  }
}

int u = 1;
string ans;

while (k > siz[u]) {
  ans.push_back('$');
  k -= siz[u];
  for (int ch = 0; ch < 26; ++ch) {
    int v = sam.next(u, ch);
    if (v == 0) continue;
    ans.back() = 'a' + ch;
    if (k > dp[v]) {
      k -= dp[v];
    } else break;
  }
  u = sam.next(u, ans.back() - 'a');
}
cout << ans << "\n";
```

6 Palindromic Automaton

```
//  odd root: 1
// even root: 0
struct PAM {
  static constexpr int N = 26;
  struct Node {
    int len, fail;
    std::array<int, N> next;
    Node() : len(0), fail(0), next{} {}
  };

  int cur;
  std::vector<int> s;
  std::vector<Node> t;
  PAM(int n = 0) {
```

```
    s.reserve(n);
    t.reserve(n + 2);
    t.assign(2, Node());
    t[t[0].fail = 1].len = -1;
  }

  int newNode() {
    t.emplace_back();
    return t.size() - 1;
  }

  int append(int p, int ch) {
    int n = s.size();
    s.push_back(ch);

    auto get = [&](int p) {
      while (n - t[p].len - 1 < 0 || ch != s[n - t[p].len - 1]) {
        p = t[p].fail;
      }
      return p;
    };

    p = get(p);
    if (t[p].next[ch] == 0) {
      int cur = newNode();
      t[cur].len = t[p].len + 2;
      t[p].next[ch] = cur;
      if (t[cur].len != 1) {
        t[cur].fail = t[get(t[p].fail)].next[ch];
      }
    }
    return t[p].next[ch];
  }

  int len(int p)         const { return t[p].len; }
  int fail(int p)        const { return t[p].fail; }
  int next(int p, int x) const { return t[p].next[x]; }
  int size()             const { return t.size(); }
};
```

7 Aho Corasick Automaton

```
// 记得 work
// 树根 "" 为 1 号节点
// adj 为失配树的子节点
struct ACAM {
  static constexpr int N = 26;
  struct Node {
    int len, fail;
    std::vector<int> adj;
    std::array<int, N> next;
    Node() : len(0), fail(0), adj{}, next{} {}
  };

  std::vector<Node> t;
  ACAM (int n = 0) {
```

```
    t.reserve(n);
    t.assign(2, Node());
    t[0].next.fill(1);
    t[0].len = -1;
    t[0].adj.push_back(1);
  }

  int newNode() {
    t.emplace_back();
    return t.size() - 1;
  }

  int insert(const std::string& s) {
    int p = 1;
    for (auto c : s) {
      int x = c - 'a';
      if (t[p].next[x] == 0) {
        t[p].next[x] = newNode();
        t[t[p].next[x]].len = t[p].len + 1;
      }
      p = t[p].next[x];
    }
    return p;
  }
  void work() {
    std::queue<int> q;
    q.push(1);

    while (!q.empty()) {
      int u = q.front();
      q.pop();

      for (int i = 0; i < N; i++) {
        if (t[u].next[i] == 0) {
          t[u].next[i] = t[t[u].fail].next[i];
        } else {
          t[t[u].next[i]].fail = t[t[u].fail].next[i];
          t[t[t[u].fail].next[i]].adj.push_back(t[u].next[i]);
          q.push(t[u].next[i]);
        }
      }
    }
  }

  int len(int p)                    const { return t[p].len; }
  int fail(int p)                   const { return t[p].fail; }
  const std::vector<int>& adj(int p) const { return t[p].adj; }
  int next(int p, int x)            const { return t[p].next[x]; }
  int size()                        const { return t.size(); }
};
```

8 String Hash

这几条式子 `0-based` 和 `1-based` 都能用

8.1 Basic String Hash

$$\boxed{\operatorname{Hash}(s,x)=\sum_{i=1}^{|s|}s_i x^{|s|-i}}$$

$$\boxed{\operatorname{Hash}(s[l,r],x)=\operatorname{Hash}(s[1,r],x)-\operatorname{Hash}(s[1,l-1],x)\cdot x^{r-l+1}}$$

8.2 Reverse String Hash

$$\boxed{\operatorname{Hash}(\overline{s},x)=x^{|s|-1}\cdot\operatorname{Hash}(s,x^{-1})}$$

$$\boxed{\operatorname{Hash}\left(\overline{s[l,r]},x\right)=x^{r-l}\ \operatorname{Hash}(s[l,r],x^{-1})}$$

8.3 2D String Hash

$$\boxed{\operatorname{Hash}(A,x,y)=\sum_{i=1}^{n}\sum_{j=1}^{m}A_{ij}x^{n-i}y^{m-j}}$$

$$\begin{aligned}\boxed{\operatorname{Hash}\big(A[l_x,r_x]\big[l_y,r_y\big],x,y\big)}&=\operatorname{Hash}\big(A[1,r_x]\big[1,r_y\big],x,y\big)\\&-\operatorname{Hash}\big(A[1,l_x-1]\big[1,r_y\big],x,y\big)\cdot x^{r_x-l_x+1}\\&-\operatorname{Hash}\big(A[1,r_x]\big[1,l_y-1\big],x,y\big)\cdot y^{r_y-l_y+1}\\&+\operatorname{Hash}\big(A[1,l_x-1]\big[1,l_y-1\big],x,y\big)\cdot x^{r_x-l_x+1}y^{r_y-l_y+1}\end{aligned}$$

如果需要求 Reverse 的哈希值， 哪个方向反转就把对应的 base 改成逆元即可.

随机生成了一些素数（不保证质量），可能会用到：

3×10^2 : 179 191 211 227 251 311 313 347 349 353 379 389 397 419
1×10^6 : 950569 959449 960703 961531 972623 1016681 1063619
1×10^8 : 123634409 224247619 566424149 687587993 775262303 872340281
1×10^{12} : 992345236997 995678562787 1023452343671 1045674564469
1×10^{13} : 10123412340917 10123412346533 10234523455957 10567856781973
2×10^{13} : 19234523459539 19345634567977 20234523454021 20567856785261
1×10^{15} : 995678567851157 1045674567457081 1045674567459773
1×10^{17} : 101234123412348037 103456345634562587 105678567856789793
5×10^{17} : 491234123412346679 493456345634561563 502345234523452883
1×10^{18} : 992345234523451717 994567456745676007 1045674567456745241
2×10^{18} : 1956785678567854391 1956785678567855843 2056785678567853529

9 Diameter

性质：对于任意一个点 u，距离 u 最远的点一定是直径的一个端点。

但是路径不满足这个性质。hack: 考虑路径是直径本身。涉及到路径的时候，需要更细致的讨论。

9.1

前提条件：$w \geq 0$，但是这个方法找到直径比较方便。

```
vector<i64> dep(n + 1);
auto dfs = [&](auto&& dfs, int u, int fa) -> void {
  for (auto v : adj[u]) {
    if (v == fa) continue;
    dep[v] = dep[u] + w;
    dfs(dfs, v, u);
  }
};

dep[1] = 0, dfs(dfs, 1, 0);
int x = max_element(dep.begin() + 1, dep.end()) - dep.begin();
dep[x] = 0, dfs(dfs, x, 0);

i64 ret = *max_element(dep.begin() + 1, dep.end());
```

9.2

```
i64 ret = 0;
vector<array<i64, 2>> dp(n + 1);

auto dfs = [&](auto&& dfs, int u, int fa) -> void {
  dp[u] = { 0, 0 };
  for (auto v : adj[u]) {
    if (v == fa) continue;
    dfs(dfs, v, u);

    i64 get = dp[v][0] + w;
    if (get > dp[u][0]) {
      dp[u][1] = dp[u][0];
      dp[u][0] = get;
    } else if (get > dp[u][1]) {
      dp[u][1] = get;
    }
  }
  ret = max(ret, dp[u][0] + dp[u][1]);
}; dfs(dfs, 1, 0);
```

9.3 例题

给你一棵树，选择一条长度不超过 k 的路径，最小化树上任意一点到这条路径的距离的最大值。

通过上面的性质，合理地放缩，不容易证明：选择的路径一定是直径的一部分。于是双指针、单调队列一下就做完了。你也可以使用 ST 表和二分大力草过去。

10 Center

感觉这个东西屁用没有啊

最小化 $\max \mathrm{dep}_{\mathrm{center}}(u)$ 的点。

性质：

- 中心最多有两个。如果有两个，他们相邻（和重心一样）。
- 中心一定位于树的直径上
- 对于任意一个点 u，其最长路径一定经过中心。
- 当通过在两棵树间连一条边以合并为一棵树时，连接两棵树的中心可以使新树的直径最小。

- 树的中心到其他任意节点的距离不超过树直径的一半。

换根 dp 维护最大值和次大值即可实现。

```
vector<array<i64, 2>> dp(n + 1);
auto dfs1 = [&](auto&& dfs, int u, int fa) -> void {
  dp[u] = { 0, 0 };
  for (auto [v, w] : adj[u]) {
    if (v == fa) continue;
    dfs(dfs, v, u);

    i64 get = dp[v][0] + w;
    if (get > dp[u][0]) {
      dp[u][1] = dp[u][0];
      dp[u][0] = get;
    } else if (get > dp[u][1]) {
      dp[u][1] = get;
    }
  }
}; dfs1(dfs1, 1, 0);

auto dfs2 = [&](auto&& dfs, int u, int fa) -> void {
  for (auto [v, w] : adj[u]) {
    if (v == fa) continue;
    i64 get;
    if (dp[v][0] + w == dp[u][0]) {
      get = dp[u][1] + w;
    } else get = dp[u][0] + w;

    if (get > dp[v][0]) {
      dp[v][1] = dp[v][0];
      dp[v][0] = get;
    } else if (get > dp[v][1]) {
      dp[v][1] = get;
    }

    dfs(dfs, v, u);
  }
}; dfs2(dfs2, 1, 0);
```

11 Centroid

感觉这个比较重要。这是把分治思想应用到树上的基础，性质也比较多。

性质：

- 删除重心后，任意一棵子树的大小小于等于 $\lfloor \frac{n}{2} \rfloor$
- 考虑以重心为根，那么节点的**深度和最小**
- 最多存在两个重心。如果存在，两个重心相邻，且**删掉这条边后连通块大小相等**
- 删除一个点，如果删的是重心，那么剩余的子树的最大值最小
- 两棵树合并的时候，新树的重心一定在两棵树重心的路径上
- 重心一定在根节点所在重链上。根节点的重子树的重心一定是整棵树重心的后代。

实现代码参考**点分治**部分的 `get_root()` 部分

11.1 例题

其实我也不知道放什么题了。

对于所有 u 求 u 的子树的重心。u 的子树的重心一定是 u 的重儿子的重心的祖先，那么考虑从重儿子的重心暴力往上跳。

注意到跳的边一定是重边，而重边最多也就 $n-1$ 条，所以时间复杂度是 $O(n)$ 的.

```
vector<int> ans(n + 1), son(n + 1), siz(n + 1);
auto dfs = [&](auto&& dfs, int u) -> void {
  siz[u] = 1;
  for (auto v : adj[u]) {
    dfs(dfs, v);
    siz[u] += siz[v];
    if (siz[v] > siz[son[u]]) {
      son[u] = v;
    }
  }
  if (son[u] == 0) {
    assert(adj[u].empty());
    ans[u] = u;
    return;
  }

  ans[u] = ans[son[u]];
  for (int& x = ans[u]; x != u; x = fa[x]) {
    int y = fa[x];
    int now = max(siz[son[x]], siz[u] - siz[x]);
    int nxt = max(siz[son[y]], siz[u] - siz[y]);
    if (nxt >= now) break;
  }
}; dfs(dfs, 1);
```

12 点分治

好像确实就是：如果我能快速求解过某个点的情况，那么就可以用点分治。类似于分治的：如果我能快速求解过区间中点的答案，那么就可以用分治。

```
vector<int> vis(n + 1), siz(n + 1);
auto get_root = [&](int u, int size) -> int {
  int root = 0, Min = size;
  auto dfs = [&](auto dfs, int u, int fa) -> void {
    siz[u] = 1;
    int Max = 0;
    for (auto [v, w] : adj[u]) {
      if (vis[v] || v == fa) continue;
      dfs(dfs, v, u);
      siz[u] += siz[v];
      Max = max(Max, siz[v]);
    }
    Max = max(Max, size - siz[u]);
    if (Max < Min) {
      Min = Max, root = u;
    }
  }; dfs(dfs, u, 0);
  return root;
};

auto dfz = [&](auto&& dfz, int u, int size) -> void {
  u = get_root(u, size);

// 这段是求答案的
// 维护每条链的长度以及它是从哪个儿子出发的，就可以 O(nlogn) 求解
// 时间复杂度 O(nlog^2n)
// Map<int, vector<int>> d;
// d[0].push_back(u);
// auto dfs = [&](auto&& dfs, int u, int fa, int dep, int from) -> void {
//  d[dep].push_back(from);
//  for (auto [v, w] : adj[u]) {
```

```
//    if (vis[v] || v == fa) continue;
//    dfs(dfs, v, u, dep + w, from);
//  }
// };
// for (auto [v, w] : adj[u]) {
//  if (vis[v]) continue;
//  dfs(dfs, v, u, w, v);
// }
// for (auto& [x, pos] : d) {
//  sort(pos.begin(), pos.end());
//  pos.erase(unique(pos.begin(), pos.end()), pos.end());
// }
// for (int i = 1; i <= m; ++i) {
//  if (ans[i]) continue;
//  for (auto& [x, pos] : d) {
//    auto it = d.find(query[i] - x);
//    if (it == d.end()) continue;
//    ans[i] |= pos.size() > 1 | (it->second).size() > 1 | pos[0] != (it->second)[0];
//  }
// }
  vis[u] = 1;
  for (auto [v, w] : adj[u]) {
    if (vis[v]) continue;
    if (siz[v] < siz[u]) {
      dfz(dfz, v, siz[v]);
    } else {
      dfz(dfz, v, size - siz[u]);
    }
  }
}; dfz(dfz, 1, n);
```

13 LCA

13.1 倍增

```
void solve() {
  int n, q, rt;
  std::vector<std::vector<int>> adj(n + 1);

  std::vector<int> dep(n + 1);
  std::vector fa(std::__lg(n) + 1, std::vector<int> (n + 1));

  auto dfs = [&](auto&& dfs, int u) -> void {
    dep[u] = dep[fa[0][u]] + 1;
    for (auto& v : adj[u]) {
      if (v == fa[0][u]) {
        continue;
      }
      fa[0][v] = u;
      dfs(dfs, v);
    }
  }; dfs(dfs, rt);

  for (int i = 1; i <= std::__lg(n); ++i) {
    for (int u = 1; u <= n; ++u) {
      fa[i][u] = fa[i - 1][fa[i - 1][u]];
    }
  }

  auto lca = [&](int u, int v) -> int {
    if (dep[u] < dep[v]) std::swap(u, v);
    while (dep[u] != dep[v]) {
      u = fa[std::__lg(dep[u] - dep[v])][u];
    }
    if (u == v) return u;
```

```
    for (int i = std::__lg(n); i >= 0; --i) {
      if (fa[i][u] != fa[i][v]) {
        u = fa[i][u], v = fa[i][v];
      }
    }
    return fa[0][u];
  };
}
```

13.2 树剖

```
struct HLD {
  int n, cur;
  std::vector<std::vector<int>> adj;
  std::vector<int> dfn, idfn, siz, fa, top, dep;

  HLD() = default;
  HLD(int n) { init(n); }

  void init(int n) {
    this -> n = n;
    cur = 0;
    adj.assign(n + 1, {});
    dfn.assign(n + 1, 0);
    idfn.assign(n + 1, 0);
    siz.assign(n + 1, 0);
    fa.assign(n + 1, 0);
    top.assign(n + 1, 0);
    dep.assign(n + 1, 0);
  }

  void add(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }

  void work(int root = 1) {
    dfs1(root);
    top[root] = root;
    dfs2(root);
  }

  void dfs1(int u) {
    if (fa[u] != 0) {
      adj[u].erase(find(adj[u].begin(), adj[u].end(), fa[u]));
    }
    siz[u] = 1;
    for (auto& v : adj[u]) {
      dep[v] = dep[fa[v] = u] + 1;
      dfs1(v);
      siz[u] += siz[v];
      if (siz[v] > siz[adj[u][0]]) {
        std::swap(v, adj[u][0]);
      }
    }
  }

  void dfs2(int u) {
    dfn[u] = ++cur;
    idfn[cur] = u;
    for (auto v : adj[u]) {
      top[v] = v == adj[u][0] ? top[u] : v;
      dfs2(v);
    }
  }
```

```
  int lca(int u, int v) {
    while (top[u] != top[v]) {
      if (dep[top[u]] > dep[top[v]]) {
        u = fa[top[u]];
      } else {
        v = fa[top[v]];
      }
    }
    return dep[u] < dep[v] ? u : v;
  }

  int jump(int u, int k) {
    assert(dep[u] >= k);
    int d = dep[u] - k;
    while (dep[top[u]] > d)
      u = fa[top[u]];
    return idfn[dfn[u] - dep[u] + d];
  }
};
```

13.3 O(1) LCA

```
struct LCA {
  int n, cur;
  std::vector<std::vector<int>> adj, st;
  std::vector<int> fa, dep, dfn, siz;

  LCA() = default;
  LCA (int n) { init(n); }

  void init(int n) {
    this -> n = n;
    cur = 0;
    adj.assign(n + 1, {});
    st.assign(std::__lg(n) + 1, std::vector<int> (n + 1));
    fa.assign(n + 1, 0);
    dep.assign(n + 1, 0);
    dfn.assign(n + 1, 0);
    siz.assign(n + 1, 0);
  }

  void add(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }

  void dfs(int u) {
    if(fa[u] != 0) adj[u].erase(find(adj[u].begin(), adj[u].end(), fa[u]));
    siz[u] = 1;
    st[0][dfn[u] = ++cur] = u;
    for(auto v : adj[u]) {
      dep[v] = dep[fa[v] = u] + 1;
      dfs(v);
      siz[u] += siz[v];
    }
  }
  int merge(int x, int y) {
    return dep[x] < dep[y] ? x : y;
  }

  void work(int rt = 1) {
    dep[rt] = 1; dfs(rt);
    for(int i = 1; i <= std::__lg(n); ++i) {
```

```
      for(int j = 1; j + (1 << i) - 1 <= n; ++j) {
        st[i][j] = merge(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
      }
    }
  }

  int lca(int u, int v) {
    if (u == v) return u;
    u = dfn[u], v = dfn[v];
    if (u > v) std::swap(u, v);
    int k = std::__lg(v - u);
    return fa[merge(st[k][u + 1], st[k][v - (1 << k) + 1])];
  }
};
```

14 rooted functions

```
bool isAncester(int f, int u) {
  return dfn[f] <= dfn[u] && dfn[u] <= dfn[f] + siz[f] - 1;
}

int rootedParent(int rt, int u) {
  if (rt == u) return rt;
  if (!isAncester(u, rt)) return fa[u];
  // 需要保证 adj[u] 里面只有子节点，不能包含父节点
  auto it = std::upper_bound(adj[u].begin(), adj[u].end(), rt, [&](int x, int y) {
    return dfn[x] < dfn[y];
  }) - 1;
  return *it;
}

int rootedSize(int rt, int u) {
  if (rt == u) return n;
  if (!isAncester(u, rt)) return siz[u];
  return n - siz[rootedParent(rt, u)];
}

int rootedLca(int rt, int u, int v) {
  return lca(rt, u) ^ lca(u, v) ^ lca(v, rt);
}
```

15 Tarjan

15.1 强连通分量

```
struct SCC {
  int n;
  std::vector<std::vector<int>> adj;
  std::vector<int> stk;
  std::vector<int> dfn, low, bel;
  int cur, cnt;

  SCC() = default;
  SCC(int n) { init(n); }

  void init(int n) {
    this -> n = n;
    adj.assign(n + 1, {});
    dfn.assign(n + 1, 0);
    low.assign(n + 1, 0);
    bel.assign(n + 1, 0);
    stk.clear();
```

```
    cur = cnt = 0;
  }

  void add(int u, int v) {
    adj[u].push_back(v);
  }

  void dfs(int u) {
    dfn[u] = low[u] = ++cur;
    stk.push_back(u);

    for (auto v : adj[u]) {
      if (dfn[v] == 0) {
        dfs(v);
        low[u] = std::min(low[u], low[v]);
      } else if (bel[v] == 0) {
        low[u] = std::min(low[u], dfn[v]);
      }
    }

    if (dfn[u] == low[u]) {
      int x = cnt++;
      do {
        bel[x = stk.back()] = cnt;
        stk.pop_back();
      } while (x != u);
    }
  }

  void work() {
    for (int i = 1; i <= n; i++) {
      if (dfn[i] == 0) dfs(i);
    }
  }

  auto getGraph() {
    work();
    std::vector<std::vector<int>> adj(cnt + 1);
    for (int u = 1; u <= n; ++u) {
      for (auto v : adj[u]) {
        if (bel[u] != bel[v]) {
          adj[bel[u]].push_back(bel[v]);
        }
      }
    }
    return std::pair { cnt, std::move(adj) };
  }
};
```

15.2 边双连通分量

```
struct EDCC {
  int n;
  int cur, cnt, edges;
  std::vector<int> stk, dfn, low, bel;
  std::vector<std::array<int, 2>> cut;
  std::vector<std::vector<std::array<int, 2>>> adj;

  EDCC() = default;
  EDCC(int n) { init(n); }

  void init(int n) {
    this -> n = n;
    adj.assign(n + 1, {});
    dfn.assign(n + 1, 0);
```

```
    low.assign(n + 1, 0);
    bel.assign(n + 1, 0);
    stk.clear();
    cut.clear();
    cur = cnt = edges = 0;
  }
  void add(int u, int v) {
    ++edges;
    adj[u].push_back({ v, edges });
    adj[v].push_back({ u, edges });
  }
  void dfs(int u, int fa) {
    dfn[u] = low[u] = ++cur;
    stk.push_back(u);
    for (auto [v, w] : adj[u]) {
      if (w == fa) continue;
      if (dfn[v] == 0) {
        dfs(v, w);
        low[u] = std::min(low[u], low[v]);
        if (dfn[u] < low[v]) {
          cut.push_back({ u, v });
        }
      } else {
        low[u] = std::min(low[u], dfn[v]);
      }
    }
    if (low[u] == dfn[u]) {
      int x = ++cnt;
      do {
        bel[x = stk.back()] = cnt;
        stk.pop_back();
      } while (x != u);
    }
  }

  void work() {
    for (int i = 1; i <= n; ++i) {
      if (dfn[i] == 0) dfs(i, 0);
    }
  }

  auto getTree() {
    work();
    std::vector<std::vector<int>> adj(cnt + 1);
    for (auto [u, v] : cut) {
      adj[bel[u]].push_back(bel[v]);
      adj[bel[v]].push_back(bel[u]);
    }
    return std::pair { cnt, std::move(adj) };
  }
};
```

15.3 点双连通分量

```
struct VDCC {
  int n, cur;
  std::vector<std::vector<int>> adj, vdcc;
  std::vector<int> dfn, low, stk, cut;

  VDCC() = default;
  VDCC (int n) { init(n); }

  void init(int n) {
    this -> n = n;
    cur = 0;
```

```
    adj.assign(n + 1, {});
    dfn.assign(n + 1, 0);
    low.assign(n + 1, 0);
    cut.assign(n + 1, 0);
    vdcc.clear();
    stk.clear();
  }

  void add(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }




  void dfs(int u, int fa) {
    int son = 0;
    stk.push_back(u);
    low[u] = dfn[u] = ++cur;
    for (auto v : adj[u]) {
      if (dfn[v] == 0) {
        son += 1;
        dfs(v, u);
        low[u] = std::min(low[u], low[v]);
        if(low[v] >= dfn[u]) {
          if (fa != 0) cut[u] = 1;
          std::vector<int> cc { u };
          int x;
          do {
            cc.push_back(x = stk.back());
            stk.pop_back();
          } while(x != v);
          vdcc.push_back(cc);
        }
      } else if (v != fa) {
        low[u] = std::min(low[u], dfn[v]);
      }
    }

    if (fa == 0 && son == 0) vdcc.push_back({u});
    if (fa == 0 && son >= 2) cut[u] = 1;
  }

  void work() {
    for (int i = 1; i <= n; ++i) {
      if (dfn[i] == 0) dfs(i, 0);
    }
  }

  auto getTree() {
    work();
    int m = vdcc.size();
    std::vector<std::vector<int>> adj(n + m + 1);
    for (int i = 1; i <= m; ++i) {
      for (auto x : vdcc[i - 1]) {
        adj[x].push_back(i + n);
        adj[i + n].push_back(x);
      }
    }
    return std::pair { m, std::move(adj) };
  }
};
```

16 Virtual Tree

```cpp
struct VirtualTree {
  int n, tot;
  std::vector<std::vector<int>> adj, st;
  std::vector<int> fa, dep, dfn, siz;

  VirtualTree (int n) { init(n); }

  void init (int n) {
    this -> n = n;
    tot = 0;
    adj.assign(n + 1, {});
    st.assign(std::__lg(n) + 1, std::vector<int> (n + 1));
    fa.assign(n + 1, 0);
    dep.assign(n + 1, 0);
    dfn.assign(n + 1, 0);
    siz.assign(n + 1, 0);
  }

  void add (int u, int v) {
    adj[u].emplace_back(v);
    adj[v].emplace_back(u);
  }

  void dfs (int u) {
    if(fa[u] != 0) adj[u].erase(find(adj[u].begin(), adj[u].end(), fa[u]));
    siz[u] = 1;
    st[0][dfn[u] = ++tot] = u;
    for(auto v : adj[u]) {
      dep[v] = dep[fa[v] = u] + 1;
      dfs(v);
      siz[u] += siz[v];
    }
  }

  int merge (int x, int y) {
    return dep[x] < dep[y] ? x : y;
  }

  void work (int rt = 1) {
    dep[rt] = 1; dfs(rt);
    for(int i = 1; i <= std::__lg(n); ++i) {
      for(int j = 1; j + (1 << i) - 1 <= n; ++j) {
        st[i][j] = merge(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
      }
    }
  }

  int lca (int u, int v) {
    if(u == v) return u;
    u = dfn[u], v = dfn[v];
    if(u > v) std::swap(u, v);
    int len = std::__lg(v - u);
    return fa[merge(st[len][u + 1], st[len][v - (1 << len) + 1])];
  }
  template<typename Iter, typename Func>
  int build(Iter l, Iter r, Func&& link) {
    std::vector p(l, r);
```

```
    std::sort(p.begin(), p.end(), [&](int x, int y) {
      return dfn[x] < dfn[y];
    });

    std::vector<int> stk;
    stk.push_back(1);

    int len = p.size(), x = p[0] == 1;
    for (int i = p[0] == 1; i < len; ++i) {
      int u = p[i];
      int w = lca(u, stk.back());
      if (w != stk.back()) {
        while (stk.size() >= 2 && dep[w] <= dep[stk[stk.size() - 2]]) {
          link(stk[stk.size() - 2], stk.back());
          x |= stk[stk.size() - 2] == 1;
          stk.pop_back();
        }

        if (stk.back() != w) {
          link(w, stk.back());
          stk.pop_back();
          stk.push_back(w);
        }
      }
      stk.push_back(u);
    }

    len = stk.size();
    for (int i = !x; i < len - 1; ++i) {
      link(stk[i], stk[i + 1]);
    }
    return stk[!x];
  }
};
```

17 最大流

```
template<typename T>
struct Flow {
  struct edge {
    int v; T cap;
    edge(int v, T cap) : v(v), cap(cap) {}
  };

  int n;
  std::vector<edge> e;
  std::vector<std::vector<int>> g;
  std::vector<int> cur, h;

  Flow() = default;
  Flow(int n) { init(n); }

  void init(int n) {
    this -> n = n;
    g.assign(n + 1, {});
  }

  void add(int u, int v, T cap) {
    g[u].push_back(e.size());
    e.emplace_back(v, cap);
    g[v].push_back(e.size());
    e.emplace_back(u, 0);
  }

  bool bfs(int s, int t) {
```

```
  std::queue<int> que;
  h.assign(n + 1, 0);
  h[s] = 1;
  que.push(s);
  while (!que.empty()) {
    int u = que.front();
    que.pop();
    for (auto i : g[u]) {
      auto [v, cap] = e[i];
      if (cap > 0 && h[v] == 0) {
        h[v] = h[u] + 1;
        if (v == t) return true;
        que.push(v);
      }
    }
  }
  return false;
}

T dfs(int u, int t, T f) {
  if (u == t) return f;
  T r = f;
  for (int& i = cur[u]; i < (int) g[u].size(); ++i) {
    int j = g[u][i];
    auto& [v, cap] = e[j];
    if (cap > 0 && h[v] == h[u] + 1) {
      T aug = dfs(v, t, std::min(r, cap));
      r -= aug;
      e[j].cap -= aug;
      e[j ^ 1].cap += aug;
      if (r == 0) break;
    }
  }
  return f - r;
}

T flow(int s, int t) {
  T ans = 0;
  while (bfs(s, t)) {
    cur.assign(n + 1, 0);
    ans += dfs(s, t, std::numeric_limits<T>::max());
  }
  return ans;
}

std::vector<int> cut() {
  std::vector<int> x(n + 1);
  for (int i = 1; i <= n; ++i) {
    x[i] = h[i] != 0;
  }
  return x;
}

using Edge = std::tuple<int, int, T, T>;
auto edges() -> std::vector<Edge> {
  std::vector<Edge> E;
  for (int i = 0; i < (int) e.size(); i += 2) {
    E.emplace_back(
      e[i + 1].v,
      e[i].v,
      e[i].cap + e[i + 1].cap,
      e[i + 1].cap
    );
  }
```

```
    return E;
  }
};
```

18 费用流

```
template< typename T, typename F>
struct CostFlow {
  struct edge {
    int v; T cap; F cost;
    edge(int v, T cap, F cost) : v(v), cap(cap), cost(cost) {}
  };

  int n;
  std::vector<edge> e;
  std::vector<std::vector<int>> g;
  std::vector<F> h, dp;
  std::vector<int> pre;

  CostFlow() = default;
  CostFlow(int n) { init(n); }

  void init(int n) {
    this -> n = n;
    g.assign(n + 1, {});
    h.assign(n + 1, 0);
    dp.assign(n + 1, 0);
    pre.assign(n + 1, 0);
  }

  void add(int u, int v, T cap, F cost) {
    g[u].push_back(e.size());
    e.emplace_back(v, cap, cost);
    g[v].push_back(e.size());
    e.emplace_back(u, 0,  -cost);
  }

  bool dijkstra(int s, int t) {
    using node = std::pair<F, int>;
    fill( dp.begin(),  dp.end(), std::numeric_limits<F>::max());
    fill(pre.begin(), pre.end(), -1);

    std::priority_queue<node, std::vector<node>, std::greater<node>> q;
    dp[s] = 0;
    q.emplace(dp[s], s);
    while (!q.empty()) {
      auto [w, u] = q.top();
      q.pop();
      if (dp[u] != w) continue;
      for (auto i : g[u]) {
        auto [v, cap, cost] = e[i];
        if (cap > 0 && dp[u] + h[u] - h[v] + cost < dp[v]) {
          dp[v] = dp[u] + h[u] - h[v] + cost;
          pre[v] = i;
          q.emplace(dp[v], v);
        }
      }
    }
    return dp[t] != std::numeric_limits<F>::max();
  }

  std::pair<T, F> flow(int s, int t) {
    T flow = 0;
    F cost = 0;
    while (dijkstra(s, t)) {
```

```
      for (int i = 1; i <= n; ++i) {
        h[i] += dp[i];
      }
      // 最小费用可行流：if (g[t] >= 0) break;
      T aug = std::numeric_limits<T>::max();
      for (int u = t; u != s; u = e[pre[u] ^ 1].v) {
        aug = std::min(aug, e[pre[u]].cap);
      }
      for (int u = t; u != s; u = e[pre[u] ^ 1].v) {
        e[pre[u]].cap -= aug;
        e[pre[u] ^ 1].cap += aug;
      }
      flow += aug;
      cost += aug * h[t];
    }

    return std::pair(flow, cost);
  }

  using Edge = std::tuple<int, int, T, T, F>;
  auto edges() -> std::vector<Edge> {
    std::vector<Edge> E;
    for(int i = 0; i < (int)e.size(); i += 2) {
      E.emplace(
        e[i + 1].v,
        e[i].v,
        e[i].cap + e[i + 1].cap,
        e[i + 1].cap,
        e[i].cost
      );
    }
    return E;
  }
};
```

19 二分图

19.1 二分图判定

一个图为二分图当且仅当不存在奇环。可以通过染色判断。当然，如果不需要二分图，只需要判断是否存在奇环，也是可以用染色来做的。

19.2 二分图匹配

求解：考虑网络流，连边：$S \to L \to R \to T$。时间复杂度 $O(m\sqrt{n})$。使用最大流中的 `edges()` 函数找到 $L \to R$ 的边可以找到构造方案。

19.3 最小点覆盖

定义：每条边都存在至少一个端点被选择。

König 定理：最大匹配数等于最小点覆盖数。

19.4 最大独立集

性质：对于一般图：$C \subset V$ 是点覆盖当且仅当 $V - C$ 是独立集。

所以最大独立集 = $|V|$ - 最小点覆盖。

19.5 最小路径覆盖

定义：选择最少的路径，使得每个点都出现在恰好一条路径中。

由于一个点恰好出现在一条路径上，我们给路径的起点连上源点，终点连上汇点。这样每个点都是恰好 1 入度，1 出度。

考虑拆点。对于每个点，拆成 u_{in} 和 u_{out}，然后对于每条边 (u,v)，考虑在二分图上连边：$u_{\text{out}} \to v_{\text{in}}$ 这个时候最大匹配就是路径上边的数量，所以最小路径覆盖就是 $n-|f(g)|$。

如果一个点可以出现在多条路径中呢？传递闭包一下即可。

20 最小斯坦纳树

```
vector dp(n + 1, vector<i64> (1 << k, inf));
for (int i = 0; i < k; ++i) {
  dp[node[i]][1 << i] = 0;
}
for (int s = 1; s < (1 << k); ++s) {
  using node = std::pair<int, i64>;
  std::vector<int> vis(n + 1);
  std::priority_queue<node, std::vector<node>, std::greater<node>> q;
  for (int i = 1; i <= n; ++i) {
    for (int t = (s - 1) & s; t; t = (t - 1) & s) {
      dp[i][s] = min(dp[i][s], dp[i][s ^ t] + dp[i][s]);
    }
    q.emplace(dp[i][s], i);
  }
  while (!q.empty()) {
    auto [_, u] = q.top();
    q.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (auto [v, w] : adj[u]) {
      if (dp[v][s] < dp[u][s] + w) {
        dp[v][s] = dp[u][s] + w;
        q.emplace(dp[v][s], v);
      }
    }
  }
}
```

21 Kruskal 重构树

还没学，先放着看看能不能抄（

```
#include<bits/stdc++.h>
// #pragma GCC optimize("Ofast")
// #pragma GCC optimize("unroll-loops")
// #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
using namespace std;
#define int long long
inline int read(){
   int s=0,w=1;
   char ch=getchar();
   while(ch<'0'||ch>'9'){if(ch=='-')w=-1;ch=getchar();}
   while(ch>='0'&&ch<='9') s=s*10+ch-'0',ch=getchar();
   return s*w;
}
int fa[1<<21];
int find(int x)
{
  return (fa[x]==x)?x:(fa[x]=find(fa[x]));
}
int a[1<<21],b[1<<21],ans;
vector<int> e[1<<21];
void dfs(int x)
{
  for(int y:e[x]) a[y]=min(a[y],a[x]),dfs(y),b[x]+=b[y];
  if(b[x]>=2)
  {
    // printf("go %lld %lld\n",x,a[x]);
```

```
    ans+=(b[x]/2)*a[x];
    b[x]%=2;
  }
  return ;
}
signed main()
{
  for(int T=read();T--;)
  {
    int n=read(),m=read();
    ans=0;
    for(int i=1; i<=n+m; ++i) fa[i]=i,a[i]=0,e[i].clear(),b[i]=0;
    // for(int i=1; i<=n; ++i) b[i]=1;
    for(int i=1; i<=m; ++i)
    {
      int u=read(),v=read();
      a[n+i]=read();
      ans+=a[n+i];
      b[u]^=1;b[v]^=1;
      u=find(u);v=find(v);
      fa[u]=fa[v]=n+i;
      e[n+i].push_back(u);
      if(u!=v) e[n+i].push_back(v);
    }
    // int s=0;
    // for(int i=1; i<=n; ++i) s+=b[i];
    dfs(n+m);
    printf("%lld\n",ans);
  }
  return 0;
}
```

22 DSU

22.1

```
struct DSU {
  int n, cnt;
  std::vector<int> f, siz;

  DSU() = default;
  DSU(int n) { init(n); }

  void init(int n) {
    this -> n = n;
    cnt = n;
    f.resize(n + 1);
    siz.assign(n + 1, 1);
    std::iota(f.begin(), f.end(), 0);
  }

  int find(int x) {
    if (f[x] == x) return x;
    return f[x] = find(f[x]);
  }

  bool merge(int x, int y) {
    x = find(x), y = find(y);
    if (x == y) return false;
    siz[f[y] = x] += siz[y], cnt--;
    return true;
  }

  bool same(int x, int y) { return find(x) == find(y); }
  int size(int x) { return siz[find(x)]; }
```

```
  int count() const { return cnt; }
};
```

22.2

```
template<typename Info> struct WeightedDSU {
  int n, cnt;
  std::vector<int> f, siz;
  std::vector<Info> info;

  WeightedDSU() = default;
  WeightedDSU(int n) { init(n); }

  void init(int n, Info e = {}) {
    this -> n = n;
    cnt = n;
    f.resize(n + 1);
    siz.assign(n + 1, 1);
    info.assign(n + 1, e);
    std::iota(f.begin(), f.end(), 0);
  }

  int find(int x) {
    if (f[x] == x) return x;
    int p = find(f[x]);
    info[x] = info[x] + info[f[x]];
    return f[x] = p;
  }

  bool merge(int x, int y, Info w) {
    w = w + info[x] - info[y];
    x = find(x), y = find(y);
    if (x == y) return false;
    info[y] = w;
    siz[f[y] = x] += siz[y], cnt--;
    return true;
  }

  bool same(int x, int y) { return find(x) == find(y); }
  int size(int x) { return siz[find(x)]; }
  int count() const { return cnt; }
  Info query(int x) const { return info[x]; }
};
```

22.3

```
struct DrawbackDSU {
  int n, cnt;
  std::vector<int> f, siz, dep;
  std::vector<std::pair<int&, int>> his;

  DrawbackDSU() = default;

  DrawbackDSU(int n) { init(n); }

  void init(int n) {
    this -> n = n;
    cnt = n;
    f.resize(n + 1);
    siz.assign(n + 1, 1);
    dep.assign(n + 1, 1);
    std::iota(f.begin(), f.end(), 0);
  }

  int find (int x) const {
    if (f[x] == x) return x;
```

```
    return find(f[x]);
  }

  bool merge(int x, int y) {
    x = find(x), y = find(y);
    if (dep[x] < dep[y]) std::swap(x, y);
    his.push_back({f[y], f[y]});
    his.push_back({cnt, cnt});
    his.push_back({dep[x], dep[x]});
    his.push_back({siz[x], siz[x]});
    if (x == y) return false;
    dep[x] += dep[x] == dep[y];
    siz[f[y] = x] += siz[y], cnt--;
    return true;
  }

  void roll() {
    assert(his.size() >= 4);
    for (int i = 1; i <= 4; ++i) {
      auto [x, y] = his.back();
      x = y, his.pop_back();
    }
  }

  bool same(int x, int y) const { return find(x) == find(y); }
  int size(int x) const { return siz[find(x)]; }
  int count() const { return cnt; }
};
```

23 SparseTanle

23.1

```
template<typename T, typename Func>
struct SparseTable {
  int n;
  Func op;
  std::vector<std::vector<T>> st;

  SparseTable () = default;

  template<typename Iter, typename = std::_RequireInputIter<Iter>>
  SparseTable (const Iter& l, const Iter& r, Func&& f) : n(r - l), op(f) {
    st.assign(std::__lg(n) + 1, std::vector<T> (n + 1));
    for (int i = 1; i <= n; ++i) {
      st[0][i] = l[i - 1];
    }
    for (int i = 1; i <= std::__lg(n); ++i) {
      for (int j = 1; j + (1 << i) - 1 <= n; ++j) {
        st[i][j] = op(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
      }
    }
  }

  template<typename Array>
  SparseTable (int n, Array&& a, Func&& f) : n(n), op(f) {
    st.assign(std::__lg(n) + 1, std::vector<T> (n + 1));
    for (int i = 1; i <= n; ++i) {
      st[0][i] = a(i);
    }
    for (int i = 1; i <= std::__lg(n); ++i) {
      for (int j = 1; j + (1 << i) - 1 <= n; ++j) {
        st[i][j] = op(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
      }
    }
```

```
  }

  T operator () (int l, int r) {
    assert(l <= r);
    int k = std::__lg(r - l + 1);
    return op(st[k][l], st[k][r - (1 << k) + 1]);
  }
};

template<typename Iter, typename Func>
SparseTable (const Iter&, const Iter&, Func&&) ->
SparseTable<typename std::iterator_traits<Iter>::value_type, Func>;

template<typename Array, typename Func>
SparseTable (int, Array&&, Func&&) ->
SparseTable<std::invoke_result_t<Array, int>, Func>;
```

23.2

```
template<typename T, typename Func>
struct SparseTable {
  int n;
  Func op;
  std::vector<std::vector<T>> st;
  SparseTable () = default;

  template<typename Iter, typename = std::_RequireInputIter<Iter>>
  SparseTable (const Iter& l, const Iter& r, Func&& f) : n(r - l), op(f) {
    st.assign(std::__lg(n) / 2 + 1, std::vector<T> (n + 1));
    for (int i = 1; i <= n; ++i) {
      st[0][i] = l[i - 1];
    }
    for (int i = 1; i <= std::__lg(n) / 2; ++i) {
      for (int j = 1; j + (1 << (i << 1)) - 1 <= n; ++j) {
        st[i][j] = op(
          op(
            st[i - 1][j + (0 << ((i - 1) << 1))],
            st[i - 1][j + (1 << ((i - 1) << 1))]
          ),
          op(
            st[i - 1][j + (2 << ((i - 1) << 1))],
            st[i - 1][j + (3 << ((i - 1) << 1))]
          )
        );
      }
    }
  }

  T operator () (int l, int r) {
    assert(l <= r);
    int k = std::__lg(r - l + 1) / 2;
    if ((2 << (k << 1)) >= r - l + 1) {
      return op(st[k][l], st[k][r - (1 << (k << 1)) + 1]);
    } else {
      return op(st[k][l], operator()(l + (1 << (k << 1)), r));
    }
  }
};
template<typename Iter, typename Func>
SparseTable (const Iter&, const Iter&, Func&&) ->
SparseTable<typename std::iterator_traits<Iter>::value_type, Func>;
```

24 BIT

```
template<typename T>
struct BIT {
```

```
int n;
std::vector<T> tr;
constexpr int lowbit(int x) { return ; }

BIT () = default;
BIT (int n, const T& e = T()) { init(n, e); }

template<typename Array> BIT (int n, Array&& a) { init(n, a); }

template<typename Iter, typename = std::_RequireInputIter<Iter>>
BIT (const Iter& l, const Iter& r) { init(l, r); }

void init (int n, const T& e = T()) {
  init(n, [&](int) { return e; });
}

template<typename Iter, typename = std::_RequireInputIter<Iter>>
void init (const Iter& l, const Iter& r) {
  init(r - l, [&](int p) { return l[p - 1]; });
}

template<typename Array>
void init (int n, Array&& a) {
  this -> n = n;
  tr.assign(n + 1, T {});
  for(int i = 1; i <= n; ++i) {
    tr[i] += a(i);
    if(i + (i & -i) <= n) {
      tr[i + (i & -i)] += tr[i];
    }
  }
}

void modify (int p, const T& v) {
  for(; p <= n; p += p & -p) tr[p] += v;
}

T query (int p) {
  T res = T();
  for(; p; p -= p & -p) res += tr[p];
  return res;
}

T query (int l, int r) {
  return query(r) - query(l - 1);
}


int lower_bound (T k) {
  int x = 0;
  for(int i = 1 << std::__lg(n); i; i >>= 1) {
    if(x + i <= n && tr[x + i] < k) {
      k -= tr[x += i];
    }
  }
  return x + 1;
}

int upper_bound (T k) {
  int x = 0;
  for(int i = 1 << std::__lg(n); i; i >>= 1) {
    if(x + i <= n && tr[x + i] <= k) {
      k -= tr[x += i];
    }
```

```
    }
    return x + 1;
  }
};
```

25 CartesionTree

```
// f(a[fa], a[son]) = true
// return: 1-based [rt, ls, rs, fa]
template<typename Iter, typename Func, typename = std::_RequireInputIter<Iter>>
auto CartesianTree(const Iter& l, const Iter& r, Func&& f) {
  int n = r - l;
  std::vector<int> ls(n + 1), rs(n + 1), fa(n + 1);
  std::vector<int> stk;

  auto a = [&](int p) {
    return *(l + p - 1);
  };

  for(int i = 1; i <= n; ++i) {
    while(!stk.empty() && f(a(i), a(stk.back()))) {
      ls[i] = stk.back();
      stk.pop_back();
    }
    fa[ls[i]] = i;
    if(!stk.empty()) {
      rs[stk.back()] = i;
      fa[i] = stk.back();
    }
    stk.push_back(i);
  }

  return std::tuple { stk[0], std::move(ls), std::move(rs), std::move(fa) };
}
```

26 MooreVote

```
#define v first
#define c second
template<typename T, int _N>
struct MooreVote {
  using Vote = pair<T, i64>;
  constexpr static int N = _N - 1;

  array<Vote, N> d;

  MooreVote () { d.fill(Vote { T(), 0 }); }
  MooreVote (const T& e, i64 cnt = 1) {
    d.fill(Vote { T(), 0 });
    d[0] = Vote { e, cnt };
  }

  void merge(const Vote& vote) {
    for (auto& [v, c] : d) {
      if (v == vote.v) {
        c += vote.c;
        return;
      }
    }

    auto p = min_element(d.begin(), d.end(), [](const Vote& x, const Vote& y) {
      return x.c < y.c;
    }) - d.begin();
```

```
    int del = min(d[p].c, vote.c);
    if (d[p].c < vote.c) {
      d[p] = vote;
    }
    for(auto& [v, c] : d) {
      if (c > 0) c -= del;
    }
  }

  friend MooreVote operator+(MooreVote x, MooreVote y) {
    for (Vote& vote : y.d) x.merge(vote);
    return x;
  }
};
#undef v
#undef c

template<typename T>
struct MooreVote<T, 2> {
  T v; i64 c;

  MooreVote () { v = T(), c = 0; }
  MooreVote (const T& e, i64 cnt = 1) {
    v = e, c = cnt;
  }


  friend MooreVote operator+(MooreVote x, MooreVote y) {
    if (x.v == y.v) {
      return { x.v, x.c + y.c };
    } else if (x.c <= y.c) {
      return { y.v, y.c - x.c };
    } else {
      return { x.v, x.c - y.c };
    }
  }
};
```

27 ChthollyTree

```
// Requires: Fn.offset(int)
template<typename Fn>
struct ChthollyTree {
  struct Node {
    i64 l, r;
    mutable Fn v;
    Node (i64 l, i64 r, Fn v) {
      this -> l = l;
      this -> r = r;
      this -> v = v;
    }
    bool operator< (const Node& o) const {
      return l < o.l;
    }
  };

  i64 n;
  std::set<Node> odt;

  ChthollyTree (i64 n) { init(n); }

  void init (i64 n) {
    this -> n = n;
    odt.clear();
```

```
    odt.emplace(1, n, Fn());
  }

  std::set<Node>::iterator split(i64 p) {
    if (p == n + 1) return odt.end();
    auto it = odt.lower_bound(Node {p, 0, Fn()});
    if (it != odt.end() && it -> l == p) return it;
    --it;
    auto [l, r, v] = *it;
    odt.erase(it);

    odt.emplace(l, p - 1, v);
    v.offset(r - l + 1);
    return odt.emplace(p, r, v).first;
  }

  void assign(int l, int r, const Fn& val) {
    assert(1 <= l && l <= r && r <= n);
    auto y = split(r + 1);
    auto x = split(l);
    odt.erase(x, y);
    odt.emplace(l, r, val);
  }

  template<typename F>
  void perform(int l, int r, F&& f) {
    assert(1 <= l && l <= r && r <= n);
    auto y = split(r + 1);
    auto x = split(l);
    for (auto it = x; it != y; ++it) {
      auto& [l, r, v] = *it;
      f(l, r, v);
    }
  }
};
```

28 莫队

28.1 普通莫队

维护二维信息，块长 $\sqrt{n}$。

28.2 带修莫队

维护三维信息，块长 $n^{\frac{1}{3}}$，按第一个所在块为第一关键字，第二个所在块为第二关键字，第三个的值为第三关键字排序

28.3 回滚莫队

给定一个序列以及若干次询问，求区间众数，可离线。

这个问题是个经典的问题，目前有许多的解法。但从思维难度上来讲，回滚莫队可能是最简单的。

考虑莫队。我们维护一个 f 数组代表出现次数，当我们要插入一个数时，我们只需要更新这个数的次数，然后和最大值作比较。这是大家都能想到的。

但要删除一个数就显得比较复杂，于是我们可以对莫队作一点变化。

首先，对于左右端点位于一个块内的询问，我们直接暴力，这样的复杂度显然正确。

接着，我们依然按照左端点排序，将左端点在同一个块内的数拿出来一起处理。

我们直接将右指针移到该左端点所在块的右端点处，然后暴力向右插入。这样，在块外的部分的贡献我们就可以统计了。

然后我们将状态保存，然后将左指针移到左端点所在块的右端点 $+1$ 处，向左插入，将块内的部分的贡献加入。

然后，我们再将左端点逐个移回，并撤销左区间的贡献即可。注意这里是撤销，而不是删除。

具体的实现方法很多，比如我们在右端点时将区间众数答案保存（定义一个变量 lst=ans），然后左端点插入时修改 f 数组，撤回时依然修改 f 数组，然后将答案变回（ans=lst）。

这样，我们的 f 数组和 ans 最终都没有发生变化，而左指针又回到了左端点所在块的右端点 +1，就好像什么也没有发生。

29 HLD

$$\mathbf{ds}(\mathbf{dfn}(*)) = a(*)$$
$$\mathbf{ds}(*) = a(\mathbf{idfn}(*))$$

```
struct HLD {
  int n = 0, tot = 0;
  std::vector<std::vector<int>> adj;
  std::vector<int> dfn, idfn, siz, fa, top, dep;

  HLD() = default;
  HLD(int n) { init(n); }
  void init(int n) {
    this -> n = n;
    tot = 0;
    adj.assign(n + 1, {});
    dfn.assign(n + 1, 0);
    idfn.assign(n + 1, 0);
    siz.assign(n + 1, 0);
    fa.assign(n + 1, 0);
    top.assign(n + 1, 0);
    dep.assign(n + 1, 0);
  }

  void add(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }

  void work(int root = 1) {
    dfs1(root);
    top[root] = root;
    dfs2(root);
  }

  void dfs1(int u) {
    if (fa[u] != 0) {
      adj[u].erase(find(adj[u].begin(), adj[u].end(), fa[u]));
    }
    siz[u] = 1;
    for (auto& v : adj[u]) {
      dep[v] = dep[fa[v] = u] + 1;
      dfs1(v);
      siz[u] += siz[v];
      if (siz[v] > siz[adj[u][0]]) {
        std::swap(v, adj[u][0]);
      }
    }
  }

  void dfs2(int u) {
    dfn[u] = ++tot;
    idfn[tot] = u;
    for (auto v : adj[u]) {
      top[v] = v == adj[u][0] ? top[u] : v;
      dfs2(v);
```

```
  }
}

int lca(int u, int v) {
  while (top[u] != top[v]) {
    if (dep[top[u]] > dep[top[v]]) {
      u = fa[top[u]];
    } else {
      v = fa[top[v]];
    }
  }
  return dep[u] < dep[v] ? u : v;
}

int jump(int u, int k) {
  assert(dep[u] >= k);
  int d = dep[u] - k;
  while (dep[top[u]] > d) {
    u = fa[top[u]];
  }
  return idfn[dfn[u] - dep[u] + d];
}

template<typename Func>
void modify(int u, int v, Func modify) {
  while (top[u] != top[v]) {
    if (dep[top[u]] > dep[top[v]]) {
      modify(dfn[top[u]], dfn[u]);
      u = fa[top[u]];
    } else {
      modify(dfn[top[v]], dfn[v]);
      v = fa[top[v]];
    }
  }
  if (dep[u] < dep[v]) {
    modify(dfn[u], dfn[v]);
  } else {
    modify(dfn[v], dfn[u]);
  }
}

template<typename Func>
void modify(int u, Func modify) {
  modify(dfn[u], dfn[u] + siz[u] - 1);
}

template<typename Func, typename T = std::invoke_result_t<Func, int, int>>
T query(int u, int v, Func query) {
  T res = T();
  while (top[u] != top[v]) {
    if (dep[top[u]] > dep[top[v]]) {
      res = query(dfn[top[u]], dfn[u]) + res;
      u = fa[top[u]];
    } else {
      res = query(dfn[top[v]], dfn[v]) + res;
      v = fa[top[v]];
    }
  }
  if (dep[u] < dep[v]) {
    return query(dfn[u], dfn[v]) + res;
  } else {
    return query(dfn[v], dfn[u]) + res;
  }
}
```

```
  template<typename Func, typename T = std::invoke_result_t<Func, int, int>>
  T query (int u, Func query) {
    return query(dfn[u], dfn[u] + siz[u] - 1);
  }
};
```

30 SegmentTree ALL

可能包含：

- 普通线段树
- 区间修改线段树
- 动态开点线段树 & 主席树 & 线段树合并

30.1

```
#define ls (u << 1)
#define rs (u << 1 | 1)
template<typename Info>
struct SegmentTree {
  int n;
  std::vector<Info> info;

  SegmentTree() = default;

  SegmentTree(int n) { init(n); }

  template<typename Array>
  SegmentTree(int n, Array&& a) { init(n, a); }

  template<typename Iter>
  SegmentTree(const Iter& l, const Iter& r) { init(l, r); }

  void init(int n) {
    init(n, [](int) { return Info(); });
  }

  template<typename Array>
  void init(int n, Array&& a) {
    this -> n = n;
    info.assign(4 << std::__lg(n), Info {});
    build(1, 1, n, a);
  }

  template<typename Iter>
  void init(const Iter& l, const Iter& r) {
    init(r - l, [&](int p) { return l[p - 1]; });
  }

  void pull(int u) {
    info[u] = info[ls] + info[rs];
  }

  template<typename Array>
  void build(int u, int l, int r, Array&& a) {
    if (l == r) {
      info[u] = a(l);
      return;
    }
    int m = (l + r) >> 1;
    build(ls, l, m, a);
    build(rs, m + 1, r, a);
    pull(u);
  }
```

```cpp
template<typename Func>
void modify(int p, Func&& op, int u, int l, int r) {
  if (l == r) {
    op(info[u]);
    return;
  }
  int m = (l + r) >> 1;
  if (p <= m) {
    modify(p, op, ls, l, m);
  } else {
    modify(p, op, rs, m + 1, r);
  }
  pull(u);
}

template<typename Func>
void modify(int p, Func&& op) {
  modify(p, op, 1, 1, n);
}

Info query(int L, int R, int u, int l, int r) {
  if (L <= l && r <= R) {
    return info[u];
  }
  int m = (l + r) >> 1;
  if (R <= m) {
    return query(L, R, ls, l, m);
  } else if (L > m) {
    return query(L, R, rs, m + 1, r);
  } else {
    return query(L, R, ls, l, m) + query(L, R, rs, m + 1, r);
  }
}

Info query(int l, int r) {
  assert(l <= r);
  return query(l, r, 1, 1, n);
}

template<typename Func>
int findL(int L, int R, Func&& f, int u, int l, int r) {
  if (l > R || r < L || !f(info[u])) return n + 1;
  if (l == r) return l;
  int m = (l + r) >> 1;
  int p = findL(L, R, f, ls, l, m);
  if (p > n) p = findL(L, R, f, rs, m + 1, r);
  return p;
}

template<typename Func>
int findL(int l, int r, Func&& f) {
  assert(l <= r);
  return findL(l, r, f, 1, 1, n);
}

template<typename Func>
int findR(int L, int R, Func&& f, int u, int l, int r) {
  if (l > R || r < L || !f(info[u])) return 0;
  if (l == r) return l;
  int m = (l + r) >> 1;
  int p = findR(L, R, f, rs, m + 1, r);
  if (p < 1) p = findR(L, R, f, ls, l, m);
  return p;
```

```
  }

  template<typename Func>
  auto findR(int l, int r, Func&& f) {
    assert(l <= r);
    return findR(l, r, f, 1, 1, n);
  }
};
#undef ls
#undef rs
```

30.2

```
#define ls (u << 1)
#define rs (u << 1 | 1)
template<typename Info, typename Tag>
struct LazySegmentTree {
  int n;
  std::vector<Info> info;
  std::vector<Tag>  tag;

  LazySegmentTree() = default;

  LazySegmentTree(int n) { init(n); }

  template<typename Array>
  LazySegmentTree(int n, Array&& a) { init(n, a); }

  template<typename Iter>
  LazySegmentTree(const Iter& l, const Iter& r) { init(l, r); }

  void init(int n) {
    init(n, [](int) { return Info(); });
  }

  template<typename Array>
  void init(int n, Array&& a) {
    this -> n = n;
    info.assign(4 << __lg(n), Info());
     tag.assign(4 << __lg(n),  Tag());
    build(1, 1, n, a);
  }

  template<typename Iter>
  void init(const Iter& l, const Iter& r) {
    init(r - l, [](int p) { return l[p - 1]; });
  }

  void pull(int u) {
    info[u] = info[ls] + info[rs];
  }

  void apply(int u, const Tag& v) {
    info[u].apply(v);
     tag[u].apply(v);
  }

  void push(int u) {
    apply(ls, tag[u]);
    apply(rs, tag[u]);
    tag[u] = Tag();
  }

  template<typename Array>
  void build(int u, int l, int r, Array&& a) {
```

```
  if (l == r) {
    info[u] = a(l);
    return;
  }
  int m = (l + r) >> 1;
  build(ls, l, m, a);
  build(rs, m + 1, r, a);
  pull(u);
}

template<typename Func>
void modify(int p, Func&& op, int u, int l, int r) {
  if (l == r) {
    op(info[u]);
    return;
  }
  push(u);
  int m = (l + r) >> 1;
  if (p <= m) {
    modify(p, op, ls, l, m);
  } else {
    modify(p, op, rs, m + 1, r);
  }
  pull(u);
}

template<typename Func>
void modify(int p, Func&& op) {
  modify(p, op, 1, 1, n);
}

void modify(int L, int R, const Tag& v, int u, int l, int r) {
  if (L <= l && r <= R) {
    apply(u, v);
    return;
  }
  push(u);
  int m = (l + r) >> 1;
  if (L <= m) {
    modify(L, R, v, ls, l, m);
  }
  if (R > m) {
    modify(L, R, v, rs, m + 1, r);
  }
  pull(u);
}

void modify(int l, int r, const Tag& v) {
  assert(l <= r)
  modify(l, r, v, 1, 1, n);
}

Info query(int L, int R, int u, int l, int r) {
  if (L <= l && r <= R) {
    return info[u];
  }
  push(u);
  int m = (l + r) >> 1;
  if (R <= m) {
    return query(L, R, ls, l, m);
  } else if (L > m) {
    return query(L, R, rs, m + 1, r);
  } else {
    return query(L, R, ls, l, m) + query(L, R, rs, m + 1, r);
```

```
    }
  }

  Info query(int l, int r) {
    assert(l <= r);
    return query(l, r, 1, 1, n);
  }

  template<typename Func>
  int findL(int L, int R, Func&& f, int u, int l, int r) {
    if (l > R || r < L || !f(info[u])) return n + 1;
    if (l == r) return l;
    push(u);
    int m = (l + r) >> 1;
    int p = findL(L, R, f, ls, l, m);
    if (p > n) p = findL(L, R, f, rs, m + 1, r);
    return p;
  }

  template<typename Func>
  int findL(int l, int r, Func&& f) {
    assert(l <= r);
    return findL(l, r, f, 1, 1, n);
  }

  template<typename Func>
  int findR(int L, int R, Func&& f, int u, int l, int r) {
    if (l > R || r < L || !f(info[u])) return 0;
    if (l == r) return l;
    push(u);
    int m = (l + r) >> 1;
    int p = findR(L, R, f, rs, m + 1, r);
    if (p < 1) p = findR(L, R, f, ls, l, m);
    return p;
  }

  template<typename Func>
  auto findR(int l, int r, Func&& f) {
    assert(l <= r);
    return findR(l, r, f, 1, 1, n);
  }
};
#undef ls
#undef rs
```

30.3

```
template<typename Info>
struct SegmentTreePool {
  int x, y;
  std::vector<Info> info;
  std::vector<int> ls, rs;

  void reserve(int n) {
    info.reserve(n);
    ls.reserve(n);
    rs.reserve(n);
  }

  SegmentTreePool(int x, int y, Info e = {}) {
    init(x, y, e);
  }

  void init(int x, int y, Info e = {}) {
    this -> x = x, this -> y = y;
```

```
  info.assign(1, e);
  ls.assign(1, 0);
  rs.assign(1, 0);
}

void pull(int u) {
  info[u] = info[ls[u]] + info[rs[u]];
}

int Node(int u) {
  info.push_back(info[u]);
  ls.push_back(ls[u]);
  rs.push_back(rs[u]);
  return info.size() - 1;
}

template<typename Func>
int build(int l, int r, Func&& f) {
  int u = Node(0);
  if (l == r) {
    info[u] = f(l);
    return u;
  }
  int m = (l + r) >> 1;
  ls[u] = build(l, m, f);
  rs[u] = build(m + 1, r, f);
  pull(u);
  return u;
}

template<typename Func>
[[nodiscard]] int build(Func&& f) {
  return build(x, y, f);
}

template<typename Func>
int modify(int p, Func&& op, int u, int l, int r) {
  if (u == 0) u = Node(0);
  if (l == r) {
    op(info[u]);
    return u;
  }
  int m = (l + r) >> 1;
  if (p <= m) ls[u] = modify(p, op, ls[u], l, m);
  else rs[u] = modify(p, op, rs[u], m + 1, r);
  pull(u);
  return u;
}

template<typename Func>
[[nodiscard]] int modify(int u, int p, Func&& op) {
  return modify(p, op, u, x, y);
}

template<typename Func>
int extend(int p, Func&& op, int u, int l, int r) {
  u = Node(u);
  if (l == r) {
    op(info[u]);
    return u;
  }
  int m = (l + r) >> 1;
  if (p <= m) {
    ls[u] = extend(p, op, ls[u], l, m);
```

```
  } else {
    rs[u] = extend(p, op, rs[u], m + 1, r);
  }
  pull(u);
  return u;
}

template<typename Func>
[[nodiscard]] int extend(int u, int p, Func&& op) {
  return extend(p, op, u, x, y);
}

Info query(int L, int R, int u, int l, int r) {
  if (L <= l && r <= R) {
    return info[u];
  }
  int m = (l + r) >> 1;
  if (R <= m) return query(L, R, ls[u], l, m);
  if (L > m)  return query(L, R, rs[u], m + 1, r);
  return query(L, R, ls[u], l, m) + query(L, R, rs[u], m + 1, r);
}

Info query(int u, int l, int r) {
  assert(l <= r);
  return query(l, r, u, x, y);
}

template<typename Func>
int findL(int L, int R, Func&& f, int u, int l, int r) {
  if (l > R || r < L || !f(info[u])) {
    return y + 1;
  }
  if (l == r) return l;
  int m = (l + r) >> 1;
  int p = findL(L, R, f, ls[u], l, m);
  if (p > y) {
    p = findL(L, R, f, rs[u], m + 1, r);
  }
  return p;
}

template<typename Func>
int findL(int u, int l, int r, Func&& f) {
  assert(l <= r);
  return findL(l, r, f, u, x, y);
}

template<typename Func>
int findR(int L, int R, Func&& f, int u, int l, int r) {
  if (l > R || r < L || !f(info[u])) {
    return x - 1;
  }
  if (l == r) return l;
  int m = (l + r) >> 1;
  int p = findR(L, R, f, rs[u], m + 1, r);
  if (p < x) {
    p = findR(L, R, f, ls[u], l, m);
  }
  return p;
}

template<typename Func>
auto findR(int u, int l, int r, Func&& f) {
  assert(l <= r);
```

```
    return findR(l, r, f, u, x, y);
  }

  template<typename Func>
  int merge(int u, int v, int l, int r, Func&& op) {
    if (u == 0 || v == 0) return u | v;
    if (l == r) {
      info[u] = op(info[u], info[v]);
      return u;
    }
    int m = (l + r) >> 1;
    ls[u] = merge(ls[u], ls[v], l, m, op);
    rs[u] = merge(rs[u], rs[v], m + 1, r, op);
    pull(u);
    return u;
  }

  template<typename Func>
  [[nodiscard]] int merge(int u, int v, Func&& op) {
    return merge(u, v, x, y, op);
  }
};
```

31 FHQ-Treap

这个是没有封装的，瞎写，仅供参考。没加懒标记，自己加吧（

```
namespace treap {
  constexpr int maxn = 2e6;
  struct Info {
    ll val;
    Info(ll x = 0) { val = x; }
  };

  auto operator<=>(Info x, Info y) {
    return x.val <=> y.val;
  }
  Info operator+(Info x, Info y) {
    return Info(min(x.val, y.val));
  }

  mt19937 sj(time(0));
  Info val[maxn];
  Info info[maxn];
  int siz[maxn], key[maxn], ls[maxn], rs[maxn], fa[maxn];
  int rev[maxn];
  int cnt = 1;

  void init() {
    cnt = 1;
    val[0] = info[0] = Info(INF);
  }
  int add(Info x) {
    info[cnt] = val[cnt] = x;
    key[cnt] = (int) sj();
    siz[cnt] = 1, ls[cnt] = 0, rs[cnt] = 0, fa[cnt] = 0;
    rev[cnt] = 0;
    return cnt++;
  }
  void pull(int u) {
    siz[u] = siz[ls[u]] + 1 + siz[rs[u]];
    info[u] = info[ls[u]] + val[u] + info[rs[u]];
  }
  void push(int u) {
    if(rev[u]) {
```

```
    swap(ls[u], rs[u]);
    rev[ls[u]] ^= 1, rev[rs[u]] ^= 1;
    rev[u] = 0;
  }
}
void split_val(int u, Info k, int& x, int& y, int fx = 0, int fy = 0) {
  if(!(x = y = u)) return;
  push(u);
  if(val[u] <= k) {
    fa[u] = fx;
    split_val(rs[u], k, rs[u], y, u, fy);
  } else {
    fa[u] = fy;
    split_val(ls[u], k, x, ls[u], fx, u);
  }
  pull(u);
}
void split_rnk(int u, int k, int& x, int& y, int fx = 0, int fy = 0) {
  if(!(x = y = u)) return;
  push(u);
  if(siz[ls[u]] + 1 <= k) {
    fa[u] = fx;
    split_rnk(rs[u], k - siz[ls[u]] - 1, rs[u], y, u, fy);
  } else {
    fa[u] = fy;
    split_rnk(ls[u], k, x, ls[u], fx, u);
  }
  pull(u);
}
int merge(int x, int y) {
  if(!x || !y) return x | y;
  push(x), push(y);
  if(key[x] < key[y]) {
    rs[x] = merge(rs[x], y);
    if(rs[x]) fa[rs[x]] = x;
    pull(x);
    return x;
  } else {
    ls[y] = merge(x, ls[y]);
    if(ls[y]) fa[ls[y]] = y;
    pull(y);
    return y;
  }
}
template<class... Args>
[[nodiscard]] int mergeall(Args... args) {
  int rt = 0;
  ((rt = merge(rt, args)), ...);
  return rt;
}
template<class T>
int findleft(int rt, T&& f) {
  if(!f(info[rt])) return inf;
  int u = rt, k = 0;
  while(1) {
    push(u);
    if (ls[u] && f(info[ls[u]])) {
      u = ls[u];
    } else if(f(val[u])) {
      return k + siz[ls[u]] + 1;
    } else {
      k += siz[ls[u]] + 1, u = rs[u];
    }
  }
```

```
  }
  template<class T>
  int findright(int rt, T&& f) {
    if(!f(info[rt])) return -inf;
    int u = rt, k = 0;
    while(1) {
      push(u);
      if (rs[u] && f(info[rs[u]])) {
        k += siz[ls[u]] + 1, u = rs[u];
      } else if(f(val[u])) {
        return k + siz[ls[u]] + 1;
      } else {
        u = ls[u];
      }
    }
  }
  int rank(int x) {
    auto dfs = [&](auto dfs, int u) -> void {
      if(fa[u]) dfs(dfs, fa[u]);
      push(u);
    };
    dfs(dfs, x);
    int res = siz[ls[x]] + 1;
    for(int u = x; fa[u]; u = fa[u])
      if(u == rs[fa[u]])
        res += siz[ls[fa[u]]] + 1;
    return res;
  }
  int find(int x) {
    while(fa[x]) x = fa[x];
    return x;
  }
}
```

32 Sieve

32.1 线性筛

```
std::vector<int> minp, P;

void sieve(int n) {
  minp.assign(n + 1, 0);
  P.clear();
  for (int i = 2; i <= n; ++i) {
    if (minp[i] == 0) {
      P.push_back(minp[i] = i);
    }
    for (i64 p : P) {
      if(i * p > n) break;
      minp[i * p] = p;
      if(i % p == 0) break;
    }
  }
}
```

32.2 线性筛求积性函数

```
std::vector<int> minp, P;
std::vector<i64> f;

void sieve(int n) {
  minp.assign(n + 1, 0);
  P.clear();
  f.assign(n + 1, 0); f[1] = 1;
```

```
  std::vector<int> h(n + 1, 0);
  for(int i = 2; i <= n; ++i) {
    if(minp[i] == 0) {
      P.push_back(h[i] = minp[i] = i);
      f[i] = ??????;
    }
    for(i64 p : P) {
      i64 x = i * p;
      if(x > n) break;
      h[x] = (minp[x] = p) * (i % p ? 1 : h[i]);
      if(x == h[x]) {
        f[x] = ??????;
      } else {
        f[x] = f[h[x]] * f[x / h[x]];
      }
      if(i % p == 0) break;
    }
  }
}
```

函数	初始值	递推	含义
d	2	$d(i)+1$	因子个数
σ	$x+1$	$\sigma(i)+x$	因子和
φ	$x-1$	$\varphi(i)\times p$	因子和
μ	-1	0	莫比乌斯反演

32.3 区间筛

```
std::vector<int> P;

std::vector<int> interval_sieve(i64 l, i64 r) {
  assert (l + 1 <= r);
  std::vector<int> ok(r - l + 1, 1);
  if (l == 1) ok[0] = 0;
  for (i64 p : P) {
    i64 s = std::max(p * p, (l + p - 1) / p * p);
    for (i64 i = s; i <= r; i += p) {
      ok[i - l] = 0;
    }
  }
  return ok;
}
```

32.4 min_25 筛

```
template<typename T, typename F_Tp, typename sum_Tp>
struct Min_25 {
public:
  vector<i64> minp, P;
  Min_25(i64 n, F_Tp&& F, sum_Tp&& sum) {
    this -> F = F;
    this -> sum = sum;
    init(n);
  }
  T operator()(i64 n) { return g(n); }
private:
  F_Tp F;
  sum_Tp sum;
```

```
  i64 n, sq, m;
  vector<T> pre;
  vector<T> g1, g2;

  T& g(i64 k) { return k <= sq ? g1[k] : g2[n / k]; }

  void sieve(i64 n) {
    minp.assign(n + 1, 0);
    P.clear();
    for (i64 i = 2; i <= n; ++i) {
      if (minp[i] == 0) {
        P.push_back(minp[i] = i);
      }
      for (i64& p : P) {
        if(i * p > n) break;
        minp[i * p] = p;
        if(i % p == 0) break;
      }
    }
  }


  void init(i64 n) {
    this -> n = n;
    this -> sq = [n]() {
      i64 t = std::sqrt(n);
      while (t * t > n) --t;
      while ((t + 1) * (t + 1) < n) ++t;
      return t;
    } ();

    g1.assign(sq + 1, T());
    g2.assign(sq + 1, T());

    sieve(sq);
    m = P.size();

    pre.assign(m + 1, T());
    for (int i = 0; i < m; ++i) {
      pre[i + 1] = pre[i] + F(P[i]);
    }

    vector<i64> x;
    for (i64 k = 1; k <= n; ++k) {
      g(n / k) = sum(n / k) - F(1);
      x.push_back(n / k);
      k = n / (n / k);
    }

    for (i64 j = 0; j < m; ++j) {
      i64 p = P[j];
      for (i64 x : x) {
        if (p * p > x) break;
        g(x) -= F(p) * (g(x / p) - pre[j]);
      }
    }
  }
};

template<typename F_Tp, typename sum_Tp>
Min_25(int, F_Tp, sum_Tp) -> Min_25<std::invoke_result_t<F_Tp, i64>, F_Tp, sum_Tp>;

T dfs (i64 n, int x) {
  T res = g(n) - pre[x];
```

```
  for (int i = x; i < m && P[i] * P[i] <= n; ++i) {
    for (i64 j = P[i]; j * P[i] <= n; j *= P[i]) {
      res += f(j) * dfs(n / j, i + 1) + f(j * P[i]);
    }
    res += (Z(j) * j - j);
  }
  return res;
}
```

33 数论分块

```
for (i64 x = 1; x <= n; ++x) {
  i64 l = x;
  i64 r = min(n, m / (m / x));
  ans += f(m / x) * (r - l + 1);
  x = r;
}
for (i64 x = 1; x <= n; ++x) {
  i64 l = x;
  i64 r = min(n, x >= m ? INF : (m - 1) / ((m - 1) / x));
  ans += f((m + x - 1) / x) * (r - l + 1);
  x = r;
}
```

34 常用常数表

n	$\log_{\{10\}} n$	$n!$	$C(n, \frac{n}{2})$
2	0.30102999	2	2
3	0.47712125	6	3
4	0.60205999	24	6
5	0.69897000	120	10
6	0.77815125	720	20
7	0.84509804	5040	70
8	0.90308998	40320	70
9	0.95424251	362880	126
10	1	3628800	252
11	1.04139269	39916800	462
12	1.07918125	479001600	924
15	1.17609126	1.31×10^12	6435
20	1.30103	2.43×10^18	184756
25	1.39794	1.55×10^25	5200300
30	1.47712	2.65×10^32	155117520

nle	max ω(n)	max d(n)	π(n)
10	2	4	4
100	3	12	25
1e3	4	32	168
1e4	5	64	1229
1e5	6	128	9592
1e6	7	240	78498
1e7	8	448	664579
1e8	8	768	5761455
1e9	9	1344	50847534
1e10	10	2304	455052511
1e11	10	4032	4120308870
1e12	11	6720	37076114526
1e13	12	10752	$\pi(n) \sim \frac{n}{\log n}$
1e14	12	17280	$\pi(n) \sim \frac{n}{\log n}$
1e15	13	26880	$\pi(n) \sim \frac{n}{\log n}$
1e16	13	41472	$\pi(n) \sim \frac{n}{\log n}$
1e17	14	64512	$\pi(n) \sim \frac{n}{\log n}$
1e18	15	103680	$\pi(n) \sim \frac{n}{\log n}$

35 NTT

```cpp
template<typename T>
std::vector<T> NTT(std::vector<T> a, std::vector<T> b) {
  static constexpr u64 P = T::mod();
  static constexpr  T  g = 3;
  static std::vector<T> rt { 0, 1 };

  int tot = a.size() + b.size() - 1;
  int k = std::__lg(tot), n = 1 << (k + 1);

  std::vector<int> rev(n);
  for (int i = 0; i < n; ++i) {
    rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
  }

  if ((int) rt.size() < n) {
    int k = std::countr_zero(rt.size());
    rt.resize(n);
    for (; (1 << k) < n; ++k) {
      auto e = g.pow((P - 1) >> (k + 1));
      for (int i = 1 << (k - 1); i < (1 << k); ++i) {
        rt[i << 1] = rt[i];
        rt[i << 1 | 1] = rt[i] * e;
      }
    }
  }

  auto dft = [&](std::vector<T>& f) {
    for (int i = 0; i < n; ++i) {
      if (i < rev[i]) {
        std::swap(f[i], f[rev[i]]);
      }
    }
    for (int i = 1; i < n; i <<= 1) {
      for (int j = 0; j < n; j += 2 * i) {
        for (int k = 0; k < i; ++k) {
          T fx = f[j + k], fy = f[i + j + k] * rt[i + k];
          f[j + k] = fx + fy;
          f[i + j + k] = fx - fy;
        }
      }
    }
  };

  a.resize(n), b.resize(n);
  dft(a), dft(b);

  for (int i = 0; i < n; ++i) {
    a[i] *= b[i];
  }

  std::reverse(a.begin() + 1, a.end());
  dft(a);
  a.resize(tot);
  T inv = T(n).inv();
  for (int i = 0; i < tot; ++i) {
    a[i] *= inv;
  }
  return a;
}
```

36 拉格朗日插值

```cpp
// f(x) = x!, g(x) = inv(x!)
template<typename T>
T lagrange_iota(T x, const std::vector<T>& y) {
  int n = y.size();
  if (x < n) return y[x.val()];

  std::vector<T> lp(n + 1), rp(n + 1);
  lp[0] = rp[n] = 1;
  for (int i = 0; i < n; ++i) {
    lp[i + 1] = lp[i] * (x - i);
  }
  for (int i = n - 1; i >= 0; --i) {
    rp[i] = rp[i + 1] * (x - i);
  }

  T ans = 0;
  for (int i = 0; i < n; ++i) {
    if ((n - 1 - i) & 1) {
      ans -= y[i] * lp[i] * rp[i + 1] * g<T>[i] * g<T>[n - 1 - i];
    } else {
      ans += y[i] * lp[i] * rp[i + 1] * g<T>[i] * g<T>[n - 1 - i];
    }
  }
  return ans;
}

template<typename T>
T lagrange_any(T x, const std::vector<std::pair<T, T>>& f) {
  T ans = 0; int n = f.size();
  for (int i = 0; i < n; ++i) {
    T u = 1, d = 1;
    auto& [xi, yi] = f[i];
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      auto& [xj, yj] = f[j];
      if (x == xj) return yj;
      u *=  x - xj;
      d *= xi - xj;
    }
    ans += yi * u / d;
  }
  return ans;
}
```

37 Miller Rabin

```cpp
template<typename T>
bool is_prime(const T& x) {
  if (x == 2 || x == 3 || x == 5 || x == 7) return true;
  if (x < 2 || x % 2 == 0 || x % 3 == 0 || x % 5 == 0 || x % 7 == 0) return false;
  if (x < 121) return x > 1;
  const T d = (x - 1) >> __builtin_ctzll(x - 1);
  auto isok = [&](T a) -> bool {
    T y = 1, t = d;
    for (T i = a, j = d; j; j >>= 1) {
      if (j & 1) y = (i128) y * i % x;
      i = (i128) i * i % x;
    }
    while (y != 1 && y != x - 1 && t != x - 1) {
      y = (i128) y * y % x;
      t <<= 1;
    }
    return y == x - 1 || t % 2 == 1;
```

```
  };

  if (x < (1LL << 32)) for (T a : { 2, 7, 61 }) {
    if (!isok(a)) return false;
  } else for (T a : { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 }) {
    if (!isok(a)) return false;
  }
  return true;
}
```

38 Pollard Rho

```
std::mt19937_64 rng_64((std::random_device())());
auto rng(i64 L, i64 R) { return rng_64() % (R - L + 1) + L; }

template<typename T>
T pollard_rho(T n) {
  if (n <= 1) return n;
  const T c = rng(1, n - 1);
  auto f = [&](T x) { return ((i128) x * x % n + c) % n; };
  T x = 1, y = 2, z = 1, q = 1, g = 1;
  const T m = 1LL << (__lg(n) / 5);
  for (T r = 1; g == 1; r <<= 1) {
    x = y;
    for (T i = 0; i < r; ++i) y = f(y);
    for (T k = 0; k < r && g == 1; k += m) {
      z = y;
      for (T i = 0; i < min(m, r - k); ++i) {
        y = f(y);
        q = (i128) q * abs(x - y) % n;
      }
      g = gcd(q, n);
    }
  }
  if (g == n) do {
    z = f(z);
    g = gcd(abs(x - z), n);
  } while (g == 1);
  return g;
}

template<typename T>
T find_prime_factor(T n) {
  if (is_prime(n)) return n;
  for (int i = 0; i < 100; ++i) {
    T p = pollard_rho(n);
    if (is_prime(p)) return p;
    n = p;
  }
  return -1;
}

template<typename T>
auto factor(T n) {
  std::vector<std::pair<T, int>> res;
  for (int p = 2; p * p <= n && p < 100; ++p) {
    if (n % p) continue;
    int e = 0;
    do { n /= p, e++; } while (n % p == 0);
    res.emplace_back(p, e);
  }
  while (n > 1) {
    T p = find_prime_factor(n);
    int e = 0;
    do { n /= p, e++; } while (n % p == 0);
```

```
    res.emplace_back(p, e);
  }
  return res;
}
```

39 分解因子

```
template<typename T>
std::vector<T> divisor(std::vector<std::pair<T, int>> pf) {
  std::vector<T> res = { 1 };
  for (auto& [p, e] : pf) {
    int siz = res.size();
    for (int i = 0; i < siz; ++i) {
      T x = 1;
      for (int j = 0; j < e; ++j) {
        x *= p;
        res.push_back(res[i] * x);
      }
    }
  }
  // std::sort(res.begin(), res.end());
  return res;
}
```

40 Barrett 模乘

```
struct Barrett {
public:
  Barrett(u32 x) : m(x), im((u64)(-1) / x + 1) {}
  constexpr u32 Mod() const { return m; }
  constexpr u32 mul(u32 a, u32 b) const {
    u64 z = 1LL * a * b;
    u64 x = u64((u128(z) * im) >> 64);
    u32 v = u32(z - x * m);
    if (m <= v) v += m;
    return v;
  }
private:
  u32 m;
  u64 im;
};
```

41 高斯消元

```
// 要求向量 V 对域 F 构成向量空间
std::vector<std::vector<F>> A;
std::vector<V> B

for (int i = j; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    if (A[i][j] == F()) continue;
    if (i != j) {
      swap(A[i], A[j]);
      swap(B[i], B[j]);
    }
    break;
  }

  F c = A[j][j];
  for (int k = j; k < n; ++k) {
    A[j][k] /= c;
  } B[j] /= c;

  for (int i = 0; i < n; ++i) {
    if (i == j || A[i][j] == 0) continue;
```

```
    F c = A[i][j];
    for (int k = j; k < n; ++k) {
      A[i][k] -= A[j][k] * c;
    } B[i] -= B[j] * c;
  }
}
```

42 异或线性基

```
std::array<u64, N> basis {};

bool insert(u64 x) {
  for (int i = N - 1; i >= 0; --i) {
    if (x >> i & 1) {
      if (basis[i] == 0) {
        basis[i] = x;
        return true;
      }
      x ^= basis[i];
    }
  }
  return false;
}

bool contains(u64 x) {
  for (int i = N - 1; i >= 0; --i) {
    if (x >> i & 1) {
      if (basis[i] == 0) {
        basis[i] = x;
        return false;
      }
      x ^= basis[i];
    }
  }
  return true;
}
```

43 公式

43.1 二项式反演

$$g(n)=\sum_{i=0}^{n}(-1)^i\binom{n}{i}f(i)\Leftrightarrow f(n)=\sum_{i=0}^{n}(-1)^i\binom{n}{i}g(i)$$

$$g(n)=\sum_{i=0}^{n}\binom{n}{i}f(i)\Leftrightarrow f(n)=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}g(i)$$

$$g(n)=\sum_{i=n}^{N}(-1)^i\binom{i}{n}f(i)\Leftrightarrow f(n)=\sum_{i=n}^{N}(-1)^i\binom{i}{n}g(i)$$

$$g(n)=\sum_{i=n}^{N}\binom{i}{n}f(i)\Leftrightarrow f(n)=\sum_{i=n}^{N}(-1)^{i-n}\binom{i}{n}g(i)$$

43.2 莫比乌斯反演

$$f(n)=\sum_{d\mid n}g(d)\Leftrightarrow g(n)=\sum_{d\mid n}\mu\left(\frac{n}{d}\right)f(d)$$

取 $n=\gcd(x,y),g=\varepsilon$, 则

$$[\gcd(x,y)=1] \Leftrightarrow= \sum_{d \mid \gcd(x,y)} \mu(d)$$

43.3 欧拉函数的神秘公式

$$n=\prod_{i=1}^{s} p_i^{k_i} \Rightarrow \varphi(n)=n\times\prod_{i=1}^{s}\frac{p_i-1}{p_i}$$

43.4 欧拉定理 & ex 欧拉定理

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)} & \gcd(a,m)=1 \\ a^b & \gcd(a,m)\neq 1, b<\varphi(m)=1,) \\ a^{b \bmod \varphi(m)+\varphi(m)} & \gcd(a,m)\neq 1, b\geq\varphi(m) \end{cases} \pmod m$$

43.5 Lucas

$$\binom{n}{m} \equiv \binom{\lfloor \frac{n}{p} \rfloor}{\lfloor \frac{m}{p} \rfloor} \cdot \binom{n \bmod p}{m \bmod p}$$

44 一直想学但是还没学明白的（(

44.1 区间本质不同子串个数

SAM + LCT.

子串计数大概率需要一个 SAM.

首先这个问题和区间数颜色很类似，回忆一下怎么数颜色。

一种常见的方法是考虑扫描线。令 p(i) 为第 i 种颜色最后一次出现的位置，则 [l,r] 的答案为 $\sum[p(i)\geq l]$ 。用数据结构动态维护就行。

这个题同样考虑扫描线，不过维护的是子串开头最后一次出现的位置。考虑把询问离线下来，从前往后加入字符。

加入下标 x 会导致 [1,x] 的所有后缀最后出现的位置改变，由 parent tree 易知这里的后缀等价于 parent tree 上一条从根到某个结点的路径。

于是只需要考虑把这条路径原本的贡献去掉，再加入新的贡献。

这里的操作类似于 LCT 的 access，不妨用 LCT 维护 parent tree 的结构。一种暴力的思路是直接在 LCT 上暴力跳，但是复杂度是假的。

但是我们发现这条路径是连续的，所以可以在 access 的时候顺便做一下。

LCT 不能维护贡献，另外再搞一棵线段树，维护以每个位置为开头的子串数量。因为 parent tree 知，这里路径上的子串长度是连续的。在固定右端点的情况下，它们的开头显然也是一段连续的区间。所以只需要在线段树上大力修改就行。

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long ll;

const int maxn = 1e5 + 5;
```

```
const int maxq = 2e5 + 5;
const int sam_sz = maxn << 1;
const int sgt_sz = maxn << 2;

int n, q;
int pos[maxn];
ll ans[maxq];
char s[maxn];

namespace SAM
{
  int lst = 1, cur = 1;
  int len[sam_sz], fa[sam_sz], son[sam_sz][26];

  void cpy_node(int to, int from)
  {
    len[to] = len[from], fa[to] = fa[from];
    memcpy(son[to], son[from], sizeof(son[to]));
  }

  void insert(int c)
  {
    int p = lst, np = lst = ++cur;
    len[np] = len[p] + 1;
    for ( ; p  && (!son[p][c]); p = fa[p]) son[p][c] = np;
    if (!p) fa[np] = 1;
    else
    {
      int q = son[p][c];
      if (len[q] == len[p] + 1) fa[np] = q;
      else
      {
        int nq = ++cur;
        cpy_node(nq, q), len[nq] = len[p] + 1;
        fa[q] = fa[np] = nq;
        for ( ; p && (son[p][c] == q); p = fa[p]) son[p][c] = nq;
      }
    }
  }

  void build()
  {
    for (int i = 1; i <= n; i++)
    {
      insert(s[i] - 'a');
      pos[i] = lst;
    }
  }
}

namespace SGT
{
  ll sum[sgt_sz], lazy[sgt_sz];

  void push_up(int k) { sum[k] = sum[k << 1] + sum[k << 1 | 1]; }
```

```
  void push_down(int k, int l, int r)
  {
    if (!lazy[k]) return;
    int mid = (l + r) >> 1;
    sum[k << 1] += 1ll * (mid - l + 1) * lazy[k];
    sum[k << 1 | 1] += 1ll * (r - mid) * lazy[k];
    lazy[k << 1] += lazy[k], lazy[k << 1 | 1] += lazy[k];
    lazy[k] = 0ll;
  }

  void update(int k, int l, int r, int ql, int qr, int w)
  {
    if ((l >= ql) && (r <= qr))
    {
      sum[k] += 1ll * (r - l + 1) * w, lazy[k] += w;
      return;
    }
    push_down(k, l, r);
    int mid = (l + r) >> 1;
    if (ql <= mid) update(k << 1, l, mid, ql, qr, w);
    if (qr > mid) update(k << 1 | 1, mid + 1, r, ql, qr, w);
    push_up(k);
  }

  ll query(int k, int l, int r, int ql, int qr)
  {
    if ((l >= ql) && (r <= qr)) return sum[k];
    push_down(k, l, r);
    int mid = (l + r) >> 1;
    ll res = 0;
    if (ql <= mid) res += query(k << 1, l, mid, ql, qr);
    if (qr > mid) res += query(k << 1 | 1, mid + 1, r, ql, qr);
    return res;
  }
}

namespace LCT
{
  #define ls son[x][0]
  #define rs son[x][1]

  int fa[sam_sz], son[sam_sz][2];
  int top, q[sam_sz], val[sam_sz], lazy[sam_sz];
  bool rev[sam_sz];

  bool get(int x) { return (son[fa[x]][1] == x); }

  bool is_root(int x) { return (son[fa[x]][0] != x) && (son[fa[x]][1] != x); }

  void push_down(int x)
  {
    if (!lazy[x]) return;
    if (ls) val[ls] = lazy[ls] = lazy[x];
    if (rs) val[rs] = lazy[rs] = lazy[x];
    lazy[x] = 0;
  }
```

```
  void rotate(int x)
  {
    int y = fa[x], z = fa[y], k = get(x);
    son[y][k] = son[x][k ^ 1], fa[son[x][k ^ 1]] = y;
    son[x][k ^ 1] = y;
    if (!is_root(y)) son[z][son[z][1] == y] = x;
    fa[x] = z, fa[y] = x;
  }

  void splay(int x)
  {
    q[top = 1] = x;
    for (int i = x; !is_root(i); i = fa[i]) q[++top] = fa[i];
    for (int i = top; i; i--) push_down(q[i]);
    while (!is_root(x))
    {
      int y = fa[x], z = fa[y];
      if (!is_root(y)) rotate(get(x) == get(y) ? y : x);
      rotate(x);
    }
  }

  void access(int x, int cp)
  {
    int nd = 0;
    for (int t = 0; x; t = x, x = fa[x])
    {
      splay(x);
      if (int p = val[x]) SGT::update(1, 1, n, p - SAM::len[x] + 1, p - SAM::len[fa[x]],
-1);
      rs = t, nd = x;
    }
    val[nd] = lazy[nd] = cp;
    SGT::update(1, 1, n, 1, cp, 1);
  }

  void build() { for (int i = 2; i <= SAM::cur; i++) fa[i] = SAM::fa[i]; }
}

struct Query
{
  int l, r, idx;

  bool operator < (const Query& rhs) const { return (r < rhs.r); }
} qry[maxq];

int main()
{
  scanf("%s%d", s + 1, &q);
  n = strlen(s + 1);
  SAM::build();
  LCT::build();
  for (int i = 1; i <= q; i++)
  {
    qry[i].idx = i;
```

```
    scanf("%d%d", &qry[i].l, &qry[i].r);
  }
  sort(qry + 1, qry + q + 1);
  for (int i = 1, cur = 1; i <= q; i++)
  {
    while (cur <= qry[i].r) LCT::access(pos[cur], cur), cur++;
    ans[qry[i].idx] = SGT::query(1, 1, n, qry[i].l, qry[i].r);
  }
  for (int i = 1; i <= q; i++) printf("%lld\n", ans[i]);
  return 0;
}
```

44.2 Link-Cut-Tree

```
#include<bits/stdc++.h>
#define il inline
using namespace std;
il int read()
{
    int xr=0,F=1; char cr;
    while(cr=getchar(),cr<'0'||cr>'9') if(cr=='-') F=-1;
    while(cr>='0'&&cr<='9')
        xr=(xr<<3)+(xr<<1)+(cr^48),cr=getchar();
    return xr*F;
}
const int N=3e5+5;
struct node
{
    int key,sum,fa,lz;
    int s[2];
    node() {key=sum=fa=lz=s[0]=s[1]=0;}
}tr[N];
#define ls(x) tr[(x)].s[0]
#define rs(x) tr[(x)].s[1]
#define fa(x) tr[(x)].fa
il bool get(int x) {return rs(fa(x))==x;}
il bool isroot(int x) {return ls(fa(x))!=x&&rs(fa(x))!=x;}
il void pushup(int x) {tr[x].sum=tr[ls(x)].sum^tr[rs(x)].sum^tr[x].key;}
il void pushdown(int x)
{
    if(!tr[x].lz) return;
    swap(ls(x),rs(x)),tr[ls(x)].lz^=1,tr[rs(x)].lz^=1;
    tr[x].lz=0;
}
il void update(int x)
{
    if(!isroot(x)) update(fa(x));
    pushdown(x);
}
il void rotate(int x)
{
    int y=fa(x),z=fa(y),c=get(x);
    if(!isroot(y)) tr[z].s[get(y)]=x;
    fa(tr[x].s[c^1])=y,tr[y].s[c]=tr[x].s[c^1],tr[x].s[c^1]=y;
    fa(y)=x,fa(x)=z; pushup(y),pushup(x);
}
il void splay(int x)
```

```
{
    update(x);
    for(int f=fa(x);f=fa(x),!isroot(x);rotate(x))
        if(!isroot(f)) rotate(get(f)==get(x)?f:x);
}
il void access(int x) {for(int p=0;x;p=x,x=fa(x)) splay(x),rs(x)=p,pushup(x);}
il void makeroot(int x) {access(x),splay(x),tr[x].lz^=1;}
il int find(int x)
{
    access(x),splay(x);
    while(pushdown(x),ls(x)) x=ls(x);
    return splay(x),x;
}
il void link(int x,int y) {makeroot(x);if(find(y)!=x) fa(x)=y;}
il void split(int x,int y) {makeroot(x),access(y),splay(y);}
il void cut(int x,int y)
{
    makeroot(x);
    if(find(y)==x&&fa(y)==x&&!ls(y)) fa(y)=rs(x)=0;
}
int main()
{
    int n=read(),m=read();
    for(int i=1;i<=n;i++) tr[i].key=read();
    while(m--)
    {
        int op=read(),x=read(),y=read();
        if(op==0) split(x,y),printf("%d\n",tr[y].sum);
        if(op==1) link(x,y);
        if(op==2) cut(x,y);
        if(op==3) makeroot(x),splay(x),tr[x].key=y;
    }
    return 0;
}
```