2023CSP-S 复赛模测1

日记和最短路

题解

题意简述

求字符串最短路。最短的定义有两种：

- 字典序最小。
- 长度最小，之后比较字典序。

知识点

最短路、贪心、拆点

解题思路

网格图 $n \leq 2000$

网格图下，所有 1 到 n 的路径长度都相同，因此两种定义下最短路是一致的。

考虑一个简单的贪心：当前走到一个点，其两条出边里选择字典序较小的那个。

这样的贪心在大部分情况下是正确的。不正确时，两条出边字典序一定相同。

对一个点，其最短路是可以确定的。因此我们转而维护每个点的最短路。注意，这样的话空间复杂度很差，只能跑 2000。

网格图

考虑优化。有的点永远不会成为最短路的一部分，因此考虑剪枝。对若干最短路长度相等的点，它们之中只有最短路字典序最小的若干个点可能成为最短路的一个前缀。

因此，我们维护当前字典序最小的路径，并维护有哪些点的最短路是它（我们称为有效点）。

转移：遍历所有有效点的所有出边，并取它们之中字典序最小的出边。只有这些出边的终点能成为下一时刻的有效点。**注意要对有效点去重。**

保证边权长度为 1

网格图的结论仍部分适用。

现在，一个点可以有很多出边，所以不去重很容易超时。去重也不能说就不超时了，以下分析复杂度：

每条边只会被遍历一次。证明：若一条边被遍历两次，说明一个点（这条边的起点）成为了有效点至少两次。这说明原图有环，不成立。

因此，总的遍历次数是 $O(m)$ 的。

与网格图的区别：两种最短路的定义不再等价。字典序最小仍然可以直接做；而保证路径最短就需要我们先跑两次常规最短路，求出 1 到 u、u 到 n 的距离。之后，在取有效点时，要保证当前层数 + 到终点距离 = 起点到终点的最短距离。

保证边权长度为 2

也许可以考虑直接套用。

完整做法

边权长度不固定时，算法存在的漏洞是遍历出边时我们不知道哪条边是更优的。

因此，考虑拆点。利用中间点，把每条边权都变为一个字符，就仍然可以套用了。具体怎么变呢？考虑边 (u, v, w)，对每个 w_i，加边 (p_i, p_{i+1}, w_i)，其中 $p_1 = u, p_{|w|+1} = v$。

具体实现上，可以维护实际的总点数 `nc`，并在拆点时动态生成新的点。

标准代码

C++

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2000005;

struct Edge {
  int u, v, next;
  char w;
} e[maxn << 1], f[maxn << 1];
vector<tuple<int, int, string>> E;
int head[maxn], cnt;
void add(int u, int v, char w) {
  e[++cnt].v = v;
  e[cnt].w = w;
  e[cnt].next = head[u];
  head[u] = cnt;
}
int headf[maxn], cntf;
void addf(int u, int v, char w) {
  f[++cntf].v = v;
  f[cntf].w = w;
  f[cntf].next = headf[u];
  headf[u] = cntf;
}

int n, m, nc;

void make_graph() {
  for (auto P : E) {
    int u = get<0>(P);
    int v = get<1>(P);
    string w = get<2>(P);
    int lst = u;
    for (int i = 0; i < w.length(); ++i) {
      int x = u, y = v;
      if (i != 0) x = lst;
      if (i != w.length() - 1) {
        y = lst = ++nc;
      }
      add(x, y, w[i]);
```

```
      addf(y, x, w[i]);
    }
  }
}

int dis1[maxn], disn[maxn];
void solve_dis1(int s) {
  queue<int> q;
  fill(dis1 + 1, dis1 + 1 + nc, 0x3f3f3f3f);
  dis1[s] = 0;
  q.push(s);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int i = head[u]; i; i = e[i].next) {
      int v = e[i].v;
      if (dis1[v] > dis1[u] + 1) {
        dis1[v] = dis1[u] + 1;
        q.push(v);
      }
    }
  }
  return;
}
void solve_disn(int s) {
  queue<int> q;
  fill(disn + 1, disn + 1 + nc, 0x3f3f3f3f);
  disn[s] = 0;
  q.push(s);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int i = headf[u]; i; i = f[i].next) {
      int v = f[i].v;
      if (disn[v] > disn[u] + 1) {
        disn[v] = disn[u] + 1;
        q.push(v);
      }
    }
```

```
  }
  return;
}

bool vis[maxn];
string bfs(bool min_dis) {
  fill(vis, vis + maxn, false);
  string ans;
  vector<int> now, nxt;
  now.push_back(1);
  while (true) {
    char mn = 127;
    for (auto u : now) {
      for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].v;
        if (min_dis) if (ans.length() + disn[v] + 1 > dis1[n])
          continue;
        if (!min_dis) if (disn[v] >= 0x3f3f3f3f)
          continue;
        if (e[i].w < mn) {
          for (auto x : nxt) vis[x] = false;
          nxt.clear();
          mn = e[i].w;
          vis[v] = true;
          nxt.emplace_back(v);
        } else if (e[i].w == mn) {
          if (!vis[v]) {
            vis[v] = true;
            nxt.emplace_back(v);
          }
        }
      }
    }
    ans += mn;
    if (vis[n]) break;
    for (auto x : nxt) vis[x] = false;
    now.swap(nxt);
    nxt.clear();
  }
```

```
  return ans;
}

int main() {
  //freopen("path.in", "r", stdin);
  //freopen("path.out", "w", stdout);
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  cin >> n >> m;
  for (int i = 1; i <= m; ++i) {
    int u, v;
    string w;
    cin >> u >> v >> w;
    E.emplace_back(u, v, w);
  }
  nc = n;
  make_graph();
  solve_dis1(1), solve_disn(n);
  cout << bfs(true) << ' ' << bfs(false) << endl;
  return 0;
}
```

日记和欧拉函数

题解

题意简述

给定 B，多次求 \sum\limits_{i=L}^R \varphi^{(\max_{j=1}i \varphi(j) - B)}(i)。

> 日记不会这种科技，数据怎么造的？
> 你不会真的认为我写的是暴力吧 /mm

知识点

质因数分解、欧拉函数、（可选）线性筛

解题思路

前置知识

为解决此题，我们首先要证明一些东西。

> 对一个正整数 x，满足 $\varphi^{(k)}(x) = 1$ 的最小正整数 k 不超过 $2\lceil \log x \rceil$。

证明：分类讨论。

- 当 $x \equiv 0 \pmod 2$ 时，根据 φ 的定义式，所有形如 $2i$ 的数均不与 x 互质。因此，$\varphi(x) \le \dfrac{x}{2}$。
- 当 $x \equiv 1 \pmod 2$ 且 $x \neq 1$ 时，$\gcd(x, x) = x \neq 1$。因此，$\varphi(x) \le x - 1$。

额外地，可以证明奇数的 φ 函数一定为偶数。证明：若 i 与 x 互质，$(x - i)$ 也与 x 互质。特例在 $\dfrac{x}{2}$ 处，而奇数不存在这一处。

综上，每两次迭代，x 必然缩小到 $\dfrac{x}{2}$。因此，

- 事实上，可以进一步地缩小下界到 $\lceil \log x \rceil$，但此题不需要。

> 对一个正整数 $x \le B$，$f(x) = x$。

证明：$\max_{i=1}^{x} \varphi(x) \le x \le B$，因此迭代次数非正。

> 对一个正整数 $x \ge B + k$，$f(x) = 1$，其中 k 为一个较小常数（可视为 500）。

这并非一个严谨证明。考虑最小的 $p \ge B + l$（其中 l 为一个更小的常数，可视为 60），则对一个正整数 $x \ge p$，有 $\max_{i=1}^{x} \varphi(x) \ge \varphi(p) = p - 1 \ge B + l - 1$，因此迭代次数至少为 $l - 1$。由于 l 足够大（$> 2\lceil \log B \rceil$），迭代结果一定为 1（见第一条证明）。

下一步目标，是找到 $p - B$ 的上界。它相当于两个相邻质数差的最大值。万幸，10^9 范围内，这一最大值不超过 400。（日记：暴力跑的）

完整做法

以上已经把题说得差不多了。我们已经对 $x \le B$ 和 $x \ge B + k$ 的情况给出了能够达到 $O(1)$ 的做法。

剩余的数一共有 k 个。考虑暴力求。利用质因数分解求 φ 值是容易做到 $O(\sqrt{n})$ 的。而迭代次数是 $O(\log n)$ 的，这部分的复杂度为 $O(k\sqrt{n}\log n)$。

而 B 固定时，这一部分是不变的。因此只需要一次预处理。总复杂度：$O(k\sqrt{n}\log n + T)$。

诈骗：答案显然不会超过 n^2，因此开个 `long long` 就行。

以上 n 均指 $\max R$，即数据范围的第三列值。

标准代码

C++

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

#define int long long
bool inp[1000005];
int pri[1000005], pc;
void sieve(int mx) {
  inp[0] = inp[1] = true;
  for (int i = 2; i <= mx; ++i) {
    if (!inp[i]) pri[++pc] = i;
    for (int j = 1; i * pri[j] <= mx && j <= pc; ++j) {
      inp[i * pri[j]] = true;
      if (i % pri[j] == 0) break;
    }
  }
  return;
}

bool is_prime(int x) {
  if (x <= 1) return false;
  for (int i = 1; pri[i] * pri[i] <= x; ++i) {
    if (x % pri[i] == 0) return false;
  }
  return true;
}
int phi(int x) {
  int ans = x;
  for (int i = 1; pri[i] * pri[i] <= x; ++i) {
    if (x % pri[i] == 0) {
      while (x % pri[i] == 0) {
        x /= pri[i];
      }
      ans = ans / pri[i] * (pri[i] - 1);
    }
  }
  if (x > 1) ans = ans / x * (x - 1);
  return ans;
}
```

```
int T, B;
int f(int x, int max_phi) {
  max_phi -= B;
  if (max_phi <= 0) return x;
  while (max_phi--) {
    x = phi(x);
    if (x == 1) return 1;
  }
  return x;
}

int p1, p2;
int val[1000005], ps[1000005];
int a(int x) {
  return x * (x + 1) / 2; // 1 + 2 + ... + x
}
int b(int x) {
  return ps[x - p1];
}
int c(int x) {
  return x - p2; // p2+1 p2+2 .. x
}
void pre() {
  for (int i = B; i; --i) {
    if (is_prime(i)) {
      p1 = i;
      break;
    }
  }
  for (int i = B + 60; i <= 1100000000; ++i) {
    if (is_prime(i)) {
      p2 = i;
      break;
    }
  }

  int mx = phi(p1);
  for (int i = p1; i <= p2; ++i) {
    mx = max(mx, phi(i));
```

```
    val[i - p1] = f(i, mx);
    if (i != p1) ps[i - p1] = ps[i - p1 - 1] + val[i - p1];
    else ps[i - p1] = val[i - p1];
  }
}
int solve(int x) {
  if (x < p1) return a(x);
  else if (x <= p2) return a(p1 - 1) + b(x);
  else return a(p1 - 1) + b(p2) + c(x);
}

signed main() {
  //freopen("phi.in", "r", stdin);
  //freopen("phi.out", "w", stdout);
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  sieve(1000000);
  cin >> T >> B;
  pre();
  while (T--) {
    int L, R;
    cin >> L >> R;
    cout << solve(R) - solve(L - 1) << '\n';
  }
  cout.flush();
  return 0;
}
```

日记和二叉搜索树

题解

题意简述

给一棵有根树的每一个节点赋一个不同的权值 w_i，使得满足 $w_u < w_{\mathrm{lca}(u,v)} < w_v$ 的点对数最

多。

显然，由于我们只关心大小关系，每个节点权值不同等价于权值是一个排列。

知识点

二叉搜索树、0-1 背包、多重背包、（可选）STL 模板

解题思路

$n \leq 10$

考虑暴力枚举每个节点的权值。

期望得分：8

保证儿子节点个数不超过 2

考虑贪心。首先，题目名称已经给了足够多的提示：二叉搜索树。回忆一下二叉搜索树的结构，对一个节点 u，其满足 $w_{\mathrm{lson}_u} < w_u < w_{\mathrm{rson}_u}$，因此 lca 为 u 的贡献即为 $\mathrm{size}_{\mathrm{lson}_u} \times \mathrm{size}_{\mathrm{rson}_u}$。

但这不一定是最优的，我们需要证明。

试证：若一个节点的儿子个数为 2，钦定 一个子树权值 < 此节点权值 < 另一个子树权值 是最优的。

证明：先考虑 LCA 为此节点这部分贡献。此时，一对点 (u, v) 产生贡献的必要不充分条件是它们分属两个子树。若钦定 一个子树权值 < 此节点权值 < 另一个子树权值，分属两个子树的任意一对点都会产生贡献，因此一定是最优的。

再考虑两子树内部的贡献。由于我们只考虑大小关系，子树与外部是独立的。

再考虑儿子个数为 1 的情况。这种情况下，任意钦定都不会产生额外贡献。

儿子个数为 0 的情况显然不需要讨论。

至此，我们得到了儿子节点个数不超过 2 时的最优构造。

期望得分：24

$n \leq 5000$

继续考虑贪心。

试证：当一个节点的儿子个数超过 2 时，将这若干个子树分成两部分，钦定 第一部分权值 < 此节点权值 < 第二部分权值 是最优的。额外地，这两部分的总大小越接近，构造越优。

证明：第一个命题可套用上述证法。第二个命题，考虑设两部分大小之和为 s，第一部分大小为 a，则我们要最大化 $a(s-a)$。显然 $a=(s-a)=\dfrac{s}{2}$ 时表达式取最大值，且值随 $|a-(s-a)|$ 的增大而增大。

这个结论如何用于贪心呢？我们相当于控制一部分大小 $\leq \dfrac{s}{2}$，并最大化这个大小。这可以化归到 0-1 背包问题，可以解决。

时间复杂度：$O(n^2)$，跑不满。

期望得分：44，实现得好并且加上各种特判有机会拿到 64

保证儿子节点个数不超过 3/4

节点个数不超过 3 时，使最大的一个单独成组是最优的，因此可以特判掉。

节点个数不超过 4 时，暴力枚举大概率会比背包跑得快，所以可以换成暴力。

期望得分：60~80。

保证树高不超过 10

这里说一个乱搞想法。树高不高，想塞下 10^6 个节点，必然会有一堆大小重复的子树。

考虑多重背包的二进制优化。对 c 个大小为 k 的子树，我们只需要对若干个 $k\times 2^x$ 形式的东西（以及一个余项，具体可以搜多重背包了解）执行转移，而不需要对 c 个 k 进行转移。

这样会跑的快些。

期望得分：68~88。

$n \leq 30000$

考虑优化。0-1 背包中，我们只需要知道一个值是否能取到，之后暴力从大往小查找即可。而选择一棵大小为 k 的子树，就相当于让每个值 a 对 $a+k$ 产生贡献。

这是否让你想到了什么数据结构？bitset！

我们要维护的相当于是 `operator <<, operator |, _Find_last`。前两个操作 bitset 原生支持，第三个呢？bitset 是有 `_Find_first` 操作的，因此可以把整个序列翻转着存进 bitset 里，再用 `_Find_first` 实现。

如果你对 bitset 不够了解，也可以手写。这几个操作写起来都不算很难。

期望得分：88。

bitset 倍增优化

如果对每个节点 DP 时，都维护一个 10^6 的 bitset，是会直接 TLE 的。

如果你手写 bitset，不会有这个问题。但如果你在用 STL，就不得不解决它了。

出题人的写法是模板元编程，利用 `template <int mx> solve()` 来解决这个问题，每次执行 `solve` 时，若当前的 `mx` 还不够大，就调用 `solve<mx * 2>()`，否则就开一个大小为 `mx` 的 bitset 并做背包。

这里有一个小问题：子树大小并非编译期确定，因此编译器会不断倍增 `mx`，直到溢出。可以在 `mx * 2` 时将其与 `1048576` 或一个足够大值取 `min`。

倍增优化的替代

如果你不会元编程，也不想手写，那么可以考虑“根号分治”。

伪代码如下：

```
if (siz[u] <= 10) solve10();
else if (siz[u] <= 100) solve100();
else if (siz[u] <= 1000) solve1000();
else if (siz[u] <= 10000) solve10000();
else if (siz[u] <= 100000) solve100000();
else solve1000000();
```

它实际上与元编程进行了相近的工作。但是这样写，你需要将 `solve` 函数复制若干遍，并调整内部 `bitset` 的大小，非常麻烦。

元编程事实上就是让编译器完成这一工作，减少码量。

再提多重背包

这样大部分情况是正确的了。但是菊花图和类似的东西仍然能卡飞你的程序。此时多重背包二进制优化就显得十分有效了。

复杂度不会证，反正应该能过。

期望得分：100。

标准代码

C++

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1000005;
int n, a[maxn], deg[maxn];
struct Edge {
  int v, next;
} e[maxn << 1];
int head[maxn], cnt;
void add(int u, int v) {
  e[++cnt].v = v;
  e[cnt].next = head[u];
  head[u] = cnt;
}

int siz[maxn];
long long dp[maxn];
vector<int> s[maxn], t;
template<int mx>
long long solve(int sz) {
  if (mx < sz + 5) {
    return solve<((mx << 1) > 1048576 ? 1048576 : (mx << 1))>(sz);
  } else {
    bitset<mx + 5> vis;
    vis.set(mx);
    for (auto i : t) vis |= (vis >> i);
    int mid = (sz - 1) / 2;
    int i = vis._Find_next(mx - mid - 1);
    return 1ll * (mx - i) * (sz - 1 - (mx - i));
  }
}
void make(vector<int> &a) {
  sort(a.begin(), a.end());
  a.emplace_back(-1);
  t.clear();
  int cnt = 1;
  for (int i = 1; i < a.size(); ++i) {
    if (a[i] != a[i - 1]) {
      long long x = a[i - 1], j = 1;
      while (j < cnt) {
```

```
        t.emplace_back(x * j);
        cnt -= j;
        j <<= 1;
      }
      t.emplace_back(x * cnt);
      cnt = 1;
    } else ++cnt;
  }
}
void dfs(int u, int fa) {
  siz[u] = 1;
  int son = 0, mx = 0;
  for (int i = head[u]; i; i = e[i].next) {
    int v = e[i].v;
    if (v == fa) continue;
    dfs(v, u);
    siz[u] += siz[v];
    dp[u] += dp[v];
    s[u].push_back(siz[v]);
    mx = max(mx, siz[v]);
    ++son;
  }
  if (siz[u] == 1) {
    s[u].clear();
    return;
  } else if (son == 1) {
    s[u].clear();
    return;
  } else if (son == 2) {
    dp[u] += 1ll * s[u][0] * s[u][1];
    s[u].clear();
    return;
  }
  int mid = (siz[u] - 1) / 2;
  if (mx >= mid) {
    dp[u] += 1ll * mx * (siz[u] - 1 - mx);
    s[u].clear();
    return;
  }
```

```
  make(s[u]);
  dp[u] += solve<8>(siz[u]);
  s[u].clear();
  return;
}

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

  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);

  cin >> n;
  for (int i = 1; i < n; ++i) {
    int u, v;
    cin >> u >> v;
    ++deg[u], ++deg[v];
    add(u, v);
    add(v, u);
  }
  int chain_border = 0, leaf = 0;
  for (int i = 1; i <= n; ++i) {
    if (deg[i] == 1) {
      ++chain_border;
      if (chain_border == 2) leaf = i;
    } else if (deg[i] != 2) {
      chain_border = -1;
      break;
    }
  }
  if (chain_border != 2) dfs(1, 0);
  cout << dp[1] << '\n';
  // for (int i = 1; i <= n; ++i) clog << dp[i] << ' ';

  cout.flush();
  return 0;
}
```

日记和编辑器

题解

题意简述

有一个字符串 S 和一个模式串 P，每次：

- 在 S 的指定位置插入一个给定字符串；
- 删除 S 的一个子串；
- 将 S 的一个子串替换为一个给定字符串；
- 求 S 的一个子串里字符 c 的出现次数；
- 求 S 的一个子串里模式串 P 的匹配次数。

知识点

KMP、FHQ-Treap

解题思路

暴力

前四个操作，考虑用 `string` 维护。第五个操作，可以利用 `substr` 提取子串后暴力判断。

复杂度：$O(n\sum|s||P|)$

期望得分：常数很小，应该有 52

暴力优化

第五个操作可以 KMP 做。

复杂度：$O(n\sum|s|)$

期望得分：60

引入平衡树

我们首先引入“位置平衡树”这一概念（出题人编的名字）。

如同位置线段树（普通线段树）对于权值线段树，前者适合做区间修改和查询，后者适合做权值相关的查询。

平衡树呢？平衡树一般只做权值相关的查询，因此可以称为权值平衡树。

相对应地，我们有“位置平衡树”——用平衡树来维护序列。

具体怎么维护呢？如果你利用 FHQ 做过文艺平衡树，你应该对此有比较深刻的理解。

FHQ 是由分裂和合并维护 Treap 性质的一种 Treap 变种。它的分裂有两种——按值和按排名。

按值分裂的前提是值满足二叉搜索树性质，但用它维护序列，它不再具有这一性质。因此，只能按排名分裂。

我们钦定任意时刻，整棵树的中序遍历为原序列。同时，每一个节点 u 的子树维护的是一个完整区间，我们称之为 u 代表的区间。

位置平衡树上，按排名分裂会发生什么？会将原序列从某位置处断开，分裂后的两棵树根代表的区间分别为序列左半部分和右半部分。

用平衡树解决操作 1-3

构建：以给定字符串 s 构建一棵平衡树，返回树根。这可以通过分治实现：选定当前串的中点为根，并分治构建左半部分和右半部分。

插入：将序列在 p 处断开为 x, y，并构建串 s 为节点 u，顺次合并 x, u, y。

删除：将序列在 $l-1$、r 处分别断开为 x, y, z，并合并 x, z。

替换：将序列在 $l-1$、r 处分别断开为 x, y, z，构建串 s 为节点 u，并顺次合并 x, u, z。

用平衡树解决操作 4

如果平衡树上，每个节点维护了它代表的区间里每个字符出现的次数，那就很方便了！

万幸，这个值很好维护！`pushup` 时，将两子树的对应值相加，最后加上节点本身的贡献。

用平衡树解决操作 5

KMP 是避免不了的。先预处理模式串的 `fail`，之后，还是考虑如何用平衡树维护。

考虑序列上的整个 KMP 过程。我们先走了一段，然后走进了某个节点 u 代表的区间，再走出去。

走进这个区间时，会有一个 j 指针的值；走出时，又会有一个值。这两个值有什么关系呢？

如果节点 u 及其子树未发生变化，走入时的 j 和走出时的 j' 是构成单射的。换句话说，只要预处理得当，给一个走入时的 j，我们可以直接查表得到走出时的 j'。

问题变为了预处理。`pushup` 时，我们剖析一下 KMP 的流程：

（走入左子树前，指针的值为 j）先走入左子树，再走 u，再走右子树。

那么，先按照左子树的表查到走出左子树时的指针 j_1；再单独判断 u 节点自身对指针的影响（类似执行一次常规 KMP 过程的循环体），并取到新的指针 j_2；再按照右子树的表查到走出右子树时的指针 j'。

大体的流程是这样的。还要维护匹配次数——只要在构造表的时候顺带构造即可。在走到 u 节点自身时，考虑是否构成匹配（如果构成，则对答案 +1），并加上左右子树分别的匹配次数。

具体可以阅读代码。主要的逻辑在 `pushup` 中进行。

期望得分：100。

标准代码

C++

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

const int maxn = 2000005;
#define ls t[u].lson
#define rs t[u].rson
struct Node {
  int lson, rson, size;
  char ch;
  int p;
  int count[26], next[22], match[22];
} t[maxn];
mt19937 rng;
char p[22];
int root, cnt, pl, fail[22];

void pushup(int u) {
  if (u == 0) {
    t[u].size = 0;
    return;
  }
  t[u].size = t[ls].size + t[rs].size + 1;
  { for (int i = 0; i < 26; ++i) t[u].count[i] = t[ls].count[i] + t[rs].count[i];
    ++t[u].count[t[u].ch - 'a']; }
  for (int i = 0; i < pl; ++i) {
    int j = t[ls].next[i], m = 0;
    while (j != 0 && p[j + 1] != t[u].ch) j = fail[j];
    if (p[j + 1] == t[u].ch) ++j;
    if (j == pl) ++m, j = fail[j];
    t[u].next[i] = t[rs].next[j];
    t[u].match[i] = t[ls].match[i] + m + t[rs].match[j];
  }
}
void split(int u, int k, int &x, int &y) {
  if (!u) { x = y = 0; return; }
  if (k <= t[ls].size) {
    y = u;
    split(ls, k, x, ls);
  } else {
```

```
    x = u;
    split(rs, k - t[ls].size - 1, rs, y);
  }
  pushup(u);
}
int merge(int u, int v) {
  if (!u || !v) return u + v;
  if (t[u].p < t[v].p) {
    t[u].rson = merge(t[u].rson, v);
    pushup(u); return u;
  } else {
    t[v].lson = merge(u, t[v].lson);
    pushup(v); return v;
  }
}
int newnode(char ch) {
  t[++cnt].ch = ch;
  t[cnt].p = rng();
  pushup(cnt);
  return cnt;
}
int build(int l, int r, const string &s) {
  if (l > r) return 0;
  int mid = l + r >> 1, u = newnode(s[mid]);
  t[u].lson = build(l, mid - 1, s);
  t[u].rson = build(mid + 1, r, s);
  pushup(u);
  return u;
}

// void print(int u) {
//   if (!u) return;
//   clog << u << ' ' << ls << ' ' << rs << ' ' << t[u].size << ' ' << t[u].count[0] << endl;
//   print(ls);
//   cout << t[u].ch;
//   print(rs);
// }

int n;
```

```cpp
int main() {
  //freopen("edit.in", "r", stdin);
  //freopen("edit.out", "w", stdout);
  rng.seed(random_device{}());
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  cin >> n >> (p + 1);
  pl = std::strlen(p + 1);
  for (int i = 2, j = 0; i <= pl; ++i) {
    while (j != 0 && p[j + 1] != p[i]) j = fail[j];
    if (p[j + 1] == p[i]) ++j;
    fail[i] = j;
  }
  for (int i = 0; i < pl; ++i) t[0].next[i] = i;
  while (n--) {
    string op, s;
    char ch;
    int l, r, x, y, z;
    cin >> op;
    if (op == "Insert") {
      cin >> l >> s;
      split(root, l, x, y);
      root = merge(merge(x, build(0, s.length() - 1, s)), y);
    } else if (op == "Delete") {
      cin >> l >> r;
      split(root, r, x, z); split(x, l - 1, x, y);
      root = merge(x, z);
    } else if (op == "Replace") {
      cin >> l >> r >> s;
      split(root, r, x, z); split(x, l - 1, x, y);
      root = merge(merge(x, build(0, s.length() - 1, s)), z);
    } else if (op == "Count") {
      cin >> l >> r >> ch;
      split(root, r, x, z); split(x, l - 1, x, y);
      cout << t[y].count[ch - 'a'] << '\n';
      root = merge(merge(x, y), z);
    } else if (op == "Search") {
      cin >> l >> r;
```

```
      split(root, r, x, z); split(x, l - 1, x, y);
      cout << t[y].match[0] << '\n';
      root = merge(merge(x, y), z);
    } // else print(root), cout << '\n';
  }
  cout.flush();
  return 0;
}
```