徐老师的假期奖金

题解

首先要求区间和，那么必然是需要用到前缀和

可以发现对于 r 来说，所有满足 $sum[r] - sum[l-1] >= k$ 的 l 都是合法的

那么转换一下，也就是枚举到 r 的时候，需要求出前面有多少 $sum[l-1] <= sum[r] - k$

那么将 sum 数组离散化，然后直接树状数组或者线段树求个前缀和即可

非常基础的送分题

标程

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 300005;
LL a[N], b[N];
int c[N];
inline int lowbit(int x) { return x & -x; }
int sum(int x) {
    int res = 0;
    while (x > 0) {
        res += c[x];
        x -= lowbit(x);
    }
    return res;
}
void add(int x, int d, int n) {
    while (x <= n) {
        c[x] += d;
        x += lowbit(x);
    }
}
int main() {
    int n;
    LL m;
    scanf("%d%lld", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        a[i] += a[i - 1];
        b[i] = a[i];
    }
    sort(b, b + n);
    int ct = unique(b, b + n) - b;
    LL ans = 0;
    int t = lower_bound(b, b + ct, 0) - b + 1;
    add(t, 1, ct);
    for (int i = 1; i <= n; i++) {
        t = upper_bound(b, b + ct, a[i] - m) - b;
        ans += sum(t);
        t = lower_bound(b, b + ct, a[i]) - b + 1;
```

```
        add(t, 1, ct);
    }
    if (m <= 0){
        ans++;
    }
    printf("%lld\n", ans);
    return 0;
}
```

徐老师的装备强化

题解

60% 的数据可以用 n^3 通过，把 `*` 当成 1，提前处理出每一行前缀和 $sum[i][j]$ 枚举顶点 (i,j)，枚举高度 k，可以用前缀和快速判断对应的 $[l,r]$ 区间内是否都是 `*`

对于 100% 的数据考虑 dp

从上述暴力的思路中可以发现，对于 (i,j) 为顶点往下能否拓展取决于 $(i+1,j-1),(i+1,j),(i+1,j+1)$

而对于 (i,j) 为顶点最多能拓展几层，主要取决于 $(i+1,j-1),(i+1,j),(i+1,j+1)$ 这三个顶点最多能拓展的层数

可以发现，(i,j) 为顶点向下拓展的最多层数是 $(i+1,j-1),(i+1,j),(i+1,j+1)$ 三个点分别为顶点往下拓展层数的最小值 $+1$

例如如下图

```
..*..
.***.
.****
```

可以发现每个点为顶点的最大拓展层数为

```
00200
01220
01111
```

那么这里可以发现，我们只需要设 $dp[i][j]$ 为以 (i,j) 为顶点的最大高度，那么很容易得到方程 $dp[i][j]=min(dp[i+1][j-1],dp[i+1][j],dp[i+1][j+1])+1$

倒过来递推即可

标程

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e5 + 100;
typedef long long LL;

char str[1050][1050];
int dp[1050][1050];
int n, m;
int main() {
```

```
    scanf("%d%d", &n, &m);
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= n; ++i) {
        scanf("%s", str[i] + 1);
        for (int j = 1; j <= m; ++j) {
            if (str[i][j] == '*') {
                dp[i][j] = 1;
            }
        }
    }
    LL ans = 0;
    for (int i = n; i  >= 1; --i) {
        for (int j = 1; j <= m; ++j) {
            if(str[i][j] == '*') {
                dp[i][j] = min({dp[i + 1][j - 1],
                                dp[i + 1][j],
                                dp[i + 1][j + 1]}) + 1;
                ans += dp[i][j];
            }
        }
    }
    printf("%lld\n", ans);
    return 0;
}
```

徐老师的窗外风景

题解

这题的关键就是考虑到一个问题，那就是如果存在 x 条道路的 gcd 是 y，那么必然存在 $1 \sim x-1$ 条道路的 $gcd \geq y$

有了这个思路以后就会发现，这道题实际上是个预处理的题目，我们应该先预处理出所有 $1 \sim n$ 条边的答案

考虑枚举答案 x，计算出对于这个 $gcd = x$ 而言的最长路径 $maxi$，那么所有 $i <= maxi$ 的答案都至少是 x

而由于我们希望 gcd 为 x，那么就只能用 x 倍数为边权的边才能组合出这样的路径

所以我们可以在存边时进行处理，以边权将边分类，枚举 x 的时候就仅用 x 倍数的所有边建树计算最长路径即可

对于最长路径，直接 dfs 暴力计算每个节点子树中的最大值和次大值，相加就是经过当前节点的最长路径，然后直接递归即可 $O(n)$ 计算答案，这里注意，由于我们是挑了一些边出来跑，所以大概率这个图是不连通的，需要枚举一下每个联通块跑一下 dfs，即可求出对于 x 而言的最长路径长度 $maxi$

然后我们可以直接标记 $ans[maxi] = x$，最后对 ans 数组求一个后缀最大值即可

关于复杂度问题，可以发现每条边最多被计算的次数就是 a_i 的因子数，而 $100w$ 内的数字最大因子数量大概在 200 左右，也就是每条边最多被算 200 次，那么总体复杂度就是 $O(200n)$，显然这就是 $n \leq 400000$ 的由来

P.S. 这道题还存在一个特殊的贪心方式，如果做题经验足够多，那么数据里的“数据随机生成”是一个很有趣的捷径，由于数据是随机的，那么意味着这棵树的结构是真正的树而不会是一条链，那么也就是意味着存在大量的 $ans[i] = -1$，所以也可以直接暴力从 1 开始枚举边数，然后计算答案，只要碰到答案为 -1，后面的就都不用算了，全都是 -1

标程

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1000100;
int head[N], to[N], Next[N], sta[N],cnt;
struct node{
    int u,v;
} t;
int n, m, maxi, used[N], ans[N];
vector <node> e[N];
void add(int u, int v){
    to[++cnt] = v;
    Next[cnt] = head[u];
    head[u] = cnt;
    sta[cnt] = u;
}
int dfs(int now, int fa){
    used[now] = 1;
    int mx0 = 0, mx1 = 0;
    for(int i = head[now]; i ; i = Next[i]){
        int v = to[i];
        if(v != fa){
            int dis = dfs(v, now);
            if(dis >= mx0){
                mx1 = mx0;
                mx0 = dis;
            } else if(dis > mx1){
                mx1 = dis;
            }
        }
    }
    maxi = max(maxi, mx0 + mx1);
    return mx0 + 1;
}
int main(){
    memset(ans, -1, sizeof(ans));
    scanf("%d",&n);
    int len = 0;
    for(int i = 1; i < n; i++){
        int w;
        scanf("%d%d%d", &t.u, &t.v, &w);
        e[w].push_back(t);
        len = max(len, w);
    }
    for(int i = 1; i <= len; i++){
        for(int j = i; j <= len; j += i){
            for(int k = 0; k < e[j].size(); k++){
                add(e[j][k].u, e[j][k].v);
                add(e[j][k].v, e[j][k].u);
            }
        }
```

```
        maxi = 0;
        for(int j = 1; j <= cnt; ++j){
            if(!used[sta[j]]){
                dfs(sta[j], 0);
            }
        }
        ans[maxi] = i;
        for(int j = 1; j <= cnt; j++){
            used[sta[j]] = 0;
            head[sta[j]] = 0;
        }
        cnt = 0;
    }
    for(int i = n; i ; i--){
        ans[i] = max(ans[i + 1], ans[i]);
    }
    int T;scanf("%d", &T);
    while (T--){
        int x;scanf("%d", &x);
        printf("%d\n", ans[x]);
    }
    return 0;
}
```

徐老师的弱循环节

题解

暴力可以获得 20 分

然后考虑 40 分的部分，这里是在强调每次询问区间都不大，那么也就是说对于这部分而言，关键的复杂度优化是在暴力匹配循环节上

那么这部分题面里已经提示的很清楚了——Hash

这里有一个非常简单的判断弱循环节的方法，对于一个字符串 S，弱循环节 x

只要判断 $[0, len - x]$ 和 $[x, len]$ 是否相等就可以判断 x 是不是 S 的弱循环节

那么我们直接将所有字符串 $Hash$，就可以 $O(1)$ 判断弱循环节了，那么整体复杂度就来到了 $O(n * 100)$

但是这里要注意，由于可能生成的字符串数量过于庞大，所以可能会出现 Hash 冲突，最好是使用双哈希的方式防止冲突

对于满分做法，难度相对比较高，首先我们不可能在枚举出弱循环节后暴力判断，这个判断过程没有办法优化，所以我们只能换一个角度思考

那么考虑一个弱循环节 x 它有什么特点，由于题目给出的字符串生成方式，使得 A_{i+1} 一定就是在 A_i 的基础上左侧或者右侧添加了一个字母

那么也就是意味着，如果 x 不是 A_i 的弱循环节，那么它一定不会是 A_{i+1} 的弱循环节

也就是说对于一个弱循环节 x 而言，它一定是 $A_x \sim A_y$ 连续一段字符串的弱循环节，那么我们可以直接二分预处理一组 $cnt[x]$ 用于表示 x 的弱循环节持续的长度，也就是 x 是 $A_x \sim A_{cnt[x]}$ 这一段字符串的弱循环节

有了这个思路以后，剩下的就简单了，对于一次询问 l, r, pos 来说，我们要找的答案就是在 $i = a_l \sim a_r$ 中满足 $cnt[a[i]] \geq pos$ 的最小值

那么问题就变成了区间更新，区间查询最小值，但是这里有一个更加简单的做法，也就是对于 60 分部分的特殊数据

我们可以尝试离线这些询问，由于每次询问的一定是 $\leq pos$ 的值，那我们完全可以将所有询问和 $cnt[i]$ 均进行排序

从大到小依次添加 $cnt[i]$，然后每次询问的就是区间内已经存在数值的最小值

那么就变成了单点更新，区间查询，而且这里的单点更新是非常简单的，一定是从某个叶子更新到根节点，可以直接简化更新的过程

这样整体复杂度就是 $O((m+T)logm))$

标程

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int>PI;
const int N = 500005, S1 = 13331, P1 = 1000000007, S2 = 233, P2 = 998244353;
int n, m, q, f[N], st[N], head, tail, cur, que[N][2], ans[N], pos[N],
v[2001000];
char op[N], a[N * 2];
vector<PI> e[N];
vector<int> g[N];
int p1[N], h1[N * 2], p2[N], h2[N * 2];
int geth1(int l, int r){
    return((h1[r] - 1LL * h1[l - 1] * p1[r - l + 1]) % P1 + P1) % P1;
}
int geth2(int l, int r){
    return((h2[r] - 1LL * h2[l - 1] * p2[r - l + 1]) % P2 + P2) % P2;
}
int getper(int x){
    int l = x + 1, r = n, ans = x;
    while(l <= r){
        int mid = (l + r) >> 1;
        int L = st[mid], R = L + mid - 1;
        if(geth1(L, R - x) == geth1(L + x, R) &&
           geth2(L, R - x) == geth2(L + x, R)){
            ans = mid;
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    return ans;
}
void build(int id, int l, int r){
    v[id] = N;
    if(l == r){
        pos[l] = id;
        return;
    }
    int mid = (l + r) >> 1;
    build(id * 2, l, mid);
    build(id * 2 + 1, mid + 1, r);
```

```
}
void ins(int x, int p){
    for(x = pos[x]; x ; x >>= 1){
        v[x] = min(v[x], p);
    }
}
void query(int id, int l, int r, int x, int y){
    if(x <= l && r <= y){
        cur = min(cur, v[id]);
        return;
    }
    int mid = (l + r) >> 1;
    if(x <= mid){
        query(id * 2, l, mid, x, y);
    }
    if(y > mid){
        query(id * 2 + 1, mid + 1, r, x, y);
    }
}
int main(){
    scanf("%s", op);
    n = strlen(op);
    head = n + 1;
    tail = n;
    for(int i = 1; i <= n; i++){
        if(op[i - 1] >= 'a' && op[i - 1] <= 'z'){
            a[--head] = op[i - 1];
        }else{
            a[++tail] = op[i - 1] + 32;
        }
        st[i] = head;
    }
    p1[0] = 1;
    p2[0] = 1;
    for(int i = 1; i <= n; i++){
        p1[i] = 1LL * p1[i - 1] * S1 % P1;
        p2[i] = 1LL * p2[i - 1] * S2 % P2;
    }
    for(int i = head; i <= tail; i++){
        h1[i] = (1LL * h1[i - 1] * S1 + a[i]) % P1;
        h2[i] = (1LL * h2[i - 1] * S2 + a[i]) % P2;
    }
    for(int i = 1; i <= n; i++){
        f[i] = getper(i);
    }
    scanf("%d", &m);
    for(int i = 1; i <= m; i++){
        int x;
        scanf("%d", &x);
        e[f[x]].push_back(PI(i, x));
    }
    scanf("%d", &q);
    for(int i = 1; i <= q; i++){
        int x;
        scanf("%d%d%d", &x, &que[i][0], &que[i][1]);
        g[x].push_back(i);
    }
    build(1, 1, m);
```

```
    for(int i = n; i ; i--){
        for(int j = 0; j < e[i].size(); j++){
            PI t = e[i][j];
            ins(t.first, t.second);
        }
        for(int j = 0; j < g[i].size(); j++){
            int x = g[i][j];
            cur = N;
            query(1, 1, m, que[x][0], que[x][1]);
            if(cur > i){
                cur = -1;
            }
            ans[x] = cur;
        }
    }
    for(int i = 1; i <= q; i++){
        if (ans[i] == -1){
            printf("No answer!\n");
        } else {
            printf("%d\n", ans[i]);
        }
    }
    return 0;
}
```