普及T1

签到题。

反转串后按字符开头 sort，然后相同字符开头的按评分 sort。

或者读入时按字符结尾归类，然后一类的按评分 sort。

$O(1)$ 查询输出。

注意空间。

写法一：

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

const int A = 1e5 + 5;
int n, m;
struct node {
  int val, id;
  string x;
  inline void reverse() {
    int len = x.size() - 1;
    for (int i = 0; i <= len / 2; i++) swap(x[i], x[len - i]);
    return;
  }
} a[A];
int w[A];

inline bool cmp1(node u, node v) { return u.x < v.x; }

inline bool cmp2(node u, node v) {
  if (u.val != v.val) return u.val > v.val;
  return u.id < v.id;
}

signed main() {
  cin >> n >> m;
  for (int i = 1; i <= n; i++) {
    cin >> a[i].x >> a[i].val;
    a[i].id = i;
    a[i].reverse();
  }
  sort(a + 1, a + 1 + n, cmp1);
  for (int i = 1; i <= n; i++) {
    w[a[i].x[0] - 'a'] = i;
    if (a[i].x[0] != a[i - 1].x[0]) {
```

```
      int pos = i;
      while (a[pos + 1].x[0] == a[pos].x[0]) pos++;
      sort(a + i, a + pos + 1, cmp2);
      i = pos;
    }
  }
  w[26] = n + 1;
  for (int i = 26; ~i; i--)
    if (!w[i]) w[i] = w[i + 1];
  for (int i = 1; i <= n; i++) a[i].reverse();
  while (m--) {
    char x;
    cin >> x;
    int k;
    cin >> k;
    int num = x - 'a';
    if (w[num + 1] - w[num] < k)
      puts("Orz YYR tql");
    else
      cout << a[w[num] + k - 1].x << '\n';
  }
  return 0;
}
```

写法二:

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int n, m;
struct pr {
  int id, sc;
  string c;
};
vector<pr> a[28];
char s[55];
bool cmp(pr x, pr y) { return x.sc != y.sc ? x.sc > y.sc : x.id < y.id; }
int main() {
  scanf("%d%d", &n, &m);
  for (int x, i = 1; i <= n; ++i) {
    scanf("%s%d", s, &x);
    a[s[strlen(s) - 1] - 'a'].push_back(pr{i, x, s});
  }
  for (int i = 0; i < 26; ++i) sort(a[i].begin(), a[i].end(), cmp);
  int p;
  while (m--) {
    scanf("%s%d", s, &p);
    s[0] -= 'a';
```

```
    if (p > a[s[0]].size())
      printf("Orz YYR tql\n");
    else
      cout << (a[s[0]][p - 1].c) << endl;
  }
  return 0;
}
```

普及T2

首先枚举 a, b，预处理出每个 $\gcd$ 的出现次数，然后枚举每个 $\gcd$ 与 $[1, n]$ 中的数，将贡献相加即可。

时间复杂度 $O(n^2 \log n)$

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch = getchar();}
    return x * f;
}
int tong[5005], ans;
int gcd(int a, int b) {return (a % b == 0) ? b : gcd(b, a % b);}
signed main() {
    int n = Read();
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            ++tong[gcd(i, j)];
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            ans += tong[i] * gcd(i, j);
    cout << ans << endl;
    return 0;
}
```

普及T3

subtask 1

$\mathcal{O}(n^3)$ 暴力随便怎么写都可过，这里不详讲。

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

```cpp
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch = getchar();}
    return x * f;
}
int first[1000005], nxt[1000005], to[1000005], tot = 0;
void Add(int x, int y) {
    nxt[++tot] = first[x];
    first[x] = tot;
    to[tot] = y;
}
int n, m, k, l, r, fa[500005], ts[500005], dep[500005], yts[500005],
dis[500005];
void dfs(int u, int f) {
    fa[u] = f;  dep[u] = dep[f] + 1;
    for(int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if(v == f)  continue;
        dfs(v, u);
    }
}
int getlca(int x, int y) {
    if(dep[x] < dep[y])  swap(x, y);
    while(dep[x] != dep[y])  x = fa[x];
    while(x != y)  x = fa[x], y = fa[y];
    return x;
}
int getdis(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[getlca(x, y)];
}
signed main() {
    freopen("Tree41.in", "r", stdin);
    freopen("Tree41.out", "w", stdout);
    double st = clock();
    memset(dis, 0x7f, sizeof(dis));
    n = Read(), m = Read(), k = Read(), l = Read(), r = Read();
    for(int i = 1; i < n; i++) {
        int x = Read(), y = Read();
        Add(x, y); Add(y, x);
    }
    dfs(1, 0);
    for(int i = 1; i <= m; i++) {
        int x = Read();
        ts[x] = 1;
        for(int j = 1; j <= n; j++)
            dis[j] = min(dis[j], getdis(x, j));
    }
    for(int i = 1; i <= n; i++)
```

```
        if(dis[i] >= l && dis[i] <= r && !ts[i])  yts[i] = 1;
    for(int i = 1; i <= k; i++) {
        int x = Read(), ans = 0;
        for(int i = 1; i <= n; i++) {
            if(ts[i])  ans += getdis(x, i) * getdis(x, i);
            else if(yts[i])  ans += getdis(x, i);
        }
        printf("%d\n", ans);
    }
    double ed = clock();
    cerr << ed - st << endl;
    return 0;
}
```

subtask 2

观察到上述做法在求 lca 方面还可以优化，使用倍增求 lca 可在 $\mathcal{O}(n^2 \log n)$的时间复杂度内通过。

```
#include<bits/stdc++.h>
using namespace std;
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch = getchar();}
    return x * f;
}
int first[1000005], nxt[1000005], to[1000005], tot = 0;
void Add(int x, int y) {
    nxt[++tot] = first[x];
    first[x] = tot;
    to[tot] = y;
}
int n, m, k, l, r, ts[500005], dep[500005], yts[500005], dis[500005],
fa[500005][21];
void dfs(int u, int f) {
    dep[u] = dep[f] + 1;  fa[u][0] = f;
    for(int i = 1; i <= 20; i++)  fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for(int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if(v == f)  continue;
        dfs(v, u);
    }
}
int getlca(int x, int y) {
    if(dep[x] < dep[y])  swap(x, y);
    for(int i = 20; i >= 0; i--)
        if(dep[x] - (1 << i) >= dep[y])  x = fa[x][i];
    if(x == y)  return x;
```

```
    for(int i = 20; i >= 0; i--)
        if(fa[x][i] != fa[y][i])  x = fa[x][i], y = fa[y][i];
    return fa[x][0];
}
int getdis(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[getlca(x, y)];
}
signed main() {
    memset(dis, 0x7f, sizeof(dis));
    n = Read(), m = Read(), k = Read(), l = Read(), r = Read();
    for(int i = 1; i < n; i++) {
        int x = Read(), y = Read();
        Add(x, y); Add(y, x);
    }
    dfs(1, 0);
    for(int i = 1; i <= m; i++) {
        int x = Read();
        ts[x] = 1;
        for(int j = 1; j <= n; j++)
            dis[j] = min(dis[j], getdis(x, j));
    }
    for(int i = 1; i <= n; i++)
        if(dis[i] >= l && dis[i] <= r && !ts[i])  yts[i] = 1;
    for(int i = 1; i <= k; i++) {
        int x = Read(), ans = 0;
        for(int i = 1; i <= n; i++) {
            if(ts[i])  ans += getdis(x, i) * getdis(x, i);
            if(yts[i])  ans += getdis(x, i);
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

subtask 3

由于数据是一条链，所以可以用线段树在 $\mathcal{O}(n\log n)$ 的复杂度内预处理出所有叶超能力，具体是先标记距离每个点 $[1,l)$ 的点为不可能为叶超能力，所有点标记完后再处理每个超能力周围 $[l,r]$ 的叶超能力，最后 $\mathcal{O}(n\log n)$ 内扫一遍每个点，判断是否为叶超能力。

处理完后对于点 i，如果点 i 有超能力，那么就对 $[1,i-1]$ 执行区间加 $(i-1)^2,\dots,2^2,1^2$，对 $[i+1,n]$ 执行区间加 $1^2,2^2,\dots,(n-i)^2$，叶超能力同理，可以用线段树解决。

subtask 4

由于菊花图深度只有 2，所以预处理叶超能力可以暴力。

然后我们对于 1 与其它点分开处理，1 的贡献直接暴力加，对于其它点，我们按照标号顺序建一棵线段树，计算一个点对其它点的影响时，我们判断它是否为 1 。如果是 1 的话，我们就对线段树执行整体加 1 或 1^2 ~~（其实还是 1）~~， 如果是其它点的话，我们就将线段树其它点进行区间加 2 或 2^2，对于 1 号点我们直接加 1 。

subtask 5

我们考虑以每个点为根时的 DP 方程式，我们记 $d2_x$ 表示 $\sum dis(a_i,x)^2$，d_x 表示 $\sum dis(a_i,x)$，g_x 表示 $\sum dis(b_i,x)$，cnt_x 表示子树内超能力个数，$ycnt_x$ 表示子树内叶超能力个数，那么我们有：

$$g_u=\sum_{v\in son\{u\}} g_v+ycnt_v$$

$$d_u=\sum_{v\in son\{u\}} d_v+cnt_v$$

直接展开 $(x+1)^2$ 得

$$d2_u=\sum_{v\in son\{u\}} d2_v+2\cdot d_v+cnt_v$$

由于每个点都可以为根，所以写一个 DP 就好了。

我们再考虑如何处理叶超能力，由于树的边权均为 1 ，所以再 $\mathcal{O}(n)$ bfs 即可，也可以再写一个 DP 来处理叶超能力。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch = getchar();}
    return x * f;
}
int first[1000005], nxt[1000005], to[1000005], tot = 0;
void Add(int x, int y) {
    nxt[++tot] = first[x];
    first[x] = tot;
    to[tot] = y;
}
int n, m, k, l, r, dis[500005], ts[500005], yts[500005], vis[500005];
void bfs() {
    queue<int> q;
    for(int i = 1; i <= n; i++)
        if(ts[i])  q.push(i), q.push(0), vis[i] = 1;
    while(!q.empty()) {
        int u = q.front(); q.pop();
        int t = q.front(); q.pop();
        dis[u] = t;
```

```
        for(int e = first[u]; e; e = nxt[e]) {
            int v = to[e];
            if(!vis[v]) {
                q.push(v); vis[v] = 1;
                q.push(t + 1);
            }
        }
    }
    for(int i = 1; i <= n; i++)
        if(dis[i] >= l && dis[i] <= r)  yts[i] = 1;
    memset(dis, 0, sizeof(dis));
}
int cnt[500005], ycnt[500005], ydis[500005], dis2[500005], ans[500005];
void dfs1(int u, int fa) {
    cnt[u] = ts[u];  ycnt[u] = yts[u];
    for(int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if(v == fa)  continue;
        dfs1(v, u);
        cnt[u] += cnt[v];
        ycnt[u] += ycnt[v];
        dis2[u] += dis2[v] + dis[v] * 2 + cnt[v];
        dis[u] += dis[v] + cnt[v];
        ydis[u] += ydis[v] + ycnt[v];
    }
}
void dfs2(int u, int fa) {
    for(int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if(v == fa)  continue;
        if(cnt[u] > cnt[v])  dis2[v] += (dis2[u] - dis2[v] - dis[v] * 2 -
cnt[v]) + (dis[u] - dis[v] - cnt[v]) * 2 + (cnt[u] - cnt[v]);
        dis[v] += cnt[u] - cnt[v] + (dis[u] - dis[v] - cnt[v]);
        ydis[v] += ycnt[u] - ycnt[v] + (ydis[u] - ydis[v] - ycnt[v]);
        cnt[v] = cnt[u], ycnt[v] = ycnt[u];
        dfs2(v, u);
    }
}
signed main() {
    n = Read(), m = Read(), k = Read(); l = Read(), r = Read();
    for(int i = 1; i < n; i++) {
        int x = Read(), y = Read();
        Add(x, y); Add(y, x);
    }
    for(int i = 1; i <= m; i++)
        ts[Read()] = 1;
    bfs(); dfs1(1, 0); dfs2(1, 0);
    for(int i = 1; i <= n; i++)  ans[i] = dis2[i] + ydis[i];
    for(int i = 1; i <= k; i++)  printf("%lld\n", ans[Read()]);
```

```
    return 0;
}
```

普及T4

首先勇士只会增加防，于是打每只怪的回合数是不变的。然后又因为在任何时候防都不可能大于怪物的攻，所以每时每刻都一定有伤害，所以1防对每只怪的效果是不变的。效果即是降低伤害，以下称作减伤。

可以这么考虑，最小化受到的伤害，相当于**最大化减伤**。

定义怪物 i 的回合数为 h_i ，拿到的蓝宝石数量为 b_i ，定义 $\frac{b_i}{h_i}$ 为一只怪的性价比，设为 t_i。

首先考虑**菊花图**的情况：考虑一个最优的打怪序列 $\{p_1, p_2 \dots, p_n\}$ ，若交换 p_i 和 p_{i+1} ，目前减伤的变化为 $b_{i+1} * h_i - b_i * h_{i+1}$ ，因为交换后的序列一定不更优，

于是有：$b_{i+1} * h_i - b_i * h_{i+1} \le 0$

移项得：$\frac{b_i}{h_i} \ge \frac{b_{i+1}}{h_{i+1}}$

于是只需要按性价比排序，依次打即可。

然后考虑**菊花图加强版**的情况：用到了以下一个结论：**如果一只怪a挡在b前面（必须打a才能打b），如果$t_b > t_a$，则打完a后立即打b一定最优。**

证明：假设存在一个最优的打法为：打完a后又打了一连串的怪$\{s1, s2 ... sm\}$后才打b，根据前面的证明，所有t_{s_i}一定大于t_b，（否则不会在b前面打），又因为$t_b > t_a$，所以所有$t_{s_i} > t_a$，那这一连串的怪应该**在a之前打会更优**，矛盾，于是不存在任何怪会在打了a之后打，然后打b，即打a之后会立即打b。

于是可以从叶子开始，如果此节点b比父节点a的性价比高，就将两个节点缩为一个节点（$b_a + = b_b$，$h_a + = h_b$），缩完后整棵树就成了一个以性价比为关键字的大根堆。然后将当前能达到的节点的性价比为关键字放入堆中，依次取出最大的，并更新当前能达到的节点。最终得到的序列即是打怪顺序。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int Read() {
  int x = 0, f = 1; char ch = getchar();
  while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
  while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch = getchar();}
  return x * f;
}
int first[200005], nxt[200005], to[200005], tot = 0;
void Add(int x, int y) {nxt[++tot] = first[x]; first[x] = tot; to[tot] = y;}
int fa[100005], b[100005], a[100005], d[100005], hh[100005], val[100005],
HH[100005], Val[100005], tim[100005];
int vis[100005], sc[100005];
int ffa[500005];
int findfa(int x) {return (ffa[x] == x) ? x : ffa[x] = findfa(ffa[x]);}
```

```
void fight(int x) {
  //cout << x << endl;
  b[1] -= (a[x] - d[1]) * hh[x];
  d[1] += val[x];
}
void dfs(int u, int F) {
  fa[u] = F;
  for(int e = first[u]; e; e = nxt[e]) {
    int v = to[e];
    if(v == F)  continue;
    dfs(v, u);
  }
}
vector<int> Nxt[100005];
void Do(int u) {
  fight(u); sc[u] = 1;
  for(int i = 0; i < Nxt[u].size(); i++) {
    Do(Nxt[u][i]);
  }
}
signed main() {
  priority_queue<pair<double, int> > q;
  int n; scanf("%lld", &n);
  for(int i = 1; i < n; i++) {
    int x, y;
    scanf("%lld%lld", &x, &y);
    Add(x, y); Add(y, x);
  }
  dfs(1, 0);
    scanf("%lld%lld%lld", &b[1], &a[1], &d[1]);
  for(int i = 2; i <= n; i++) {
        scanf("%lld%lld%lld%lld", &b[i], &a[i], &d[i], &val[i]);
    hh[i] = b[i] / (a[1] - d[i]);  HH[i] = hh[i]; Val[i] = val[i];
    if(b[i] % (a[1] - d[i]) == 0)  --hh[i], --HH[i];
    q.push(make_pair(1.0 * val[i] / hh[i], i));
  }
  sc[1] = 1;
  for(int i = 1; i <= n; i++)  ffa[i] = i;
  while(!q.empty()) {
    int u = q.top().second; q.pop();
    if(vis[u])  continue;  vis[u] = 1;
    if(sc[fa[u]])  {Do(u); continue;}
    HH[findfa(fa[u])] += HH[u], Val[findfa(fa[u])] += Val[u];
    Nxt[ffa[fa[u]]].push_back(u);
    ffa[u] = ffa[fa[u]];
    q.push(make_pair(1.0 * Val[ffa[fa[u]]] / HH[ffa[fa[u]]], ffa[fa[u]]));
  }
  cout << b[1] << endl;
  return 0;
```

```
}
```