字符串(strings,3s,1024MB)

【算法分析】

设 S为$s_2, s_3, \dots, s_m$的所有后缀的集合，记max_lcp_t表示串s_1的后缀t与S中字符串的lcp的最大值，答案即为：

$$\min_t max\_lcp_t + 1(max\_lcp_t < |\mathrm{t}|)$$

但每次取出一个后缀对S集中的字符串求lcp并不现实。考虑对于s_1的每个后缀t，若我们将其加入S并按字典序排列，那么max_lcp_t显然为t与排在它前一位/后一位的字符串的lcp的较大值。

考虑用一个特殊字符将$s_2, s_3, \dots, s_m$串联起来，并将s_1连在最后（同样要用特殊字符连接），得到一个新的字符串s，则对于s_1的每个后缀t，它对s中非s_1的后缀的后缀求lcp和它对S中字符串求lcp显然是等效的（求得的lcp一定不会跨过特殊字符）。

于是我们可以对s求它的sa、$rank$和$height$数组。对于s_1的每个后缀t，我们找到$rank$离它最近的字典序小于/大于它的字符串，利用$RMQ + height$数组求出max_lcp_t即可。

时间复杂度$O(n\log n)$。

【参考程序】

```
//潘恩宁
#include <bits/stdc++.h>
#define For(i, a, b) for (int i = a, bb = b; i <= bb; ++i)
#define Rof(i, a, b) for (int i = a, bb = b; i >= bb; --i)
#define s64 long long

template <class T>
inline void get(T &res)
{
    char ch;
    bool bo = false;
    while ((ch = getchar()) < '0' || ch > '9')
        if (ch == '-') bo = true;

    res = ch - '0';
    while ((ch = getchar()) >= '0' && ch <= '9')
        res = (res << 1) + (res << 3) + ch - '0';

    if (bo) res = ~ res + 1;
    return;
}

template <class T>
inline void _put(T x)
{
    if (x > 9) _put(x / 10);
    putchar(x % 10 + '0');
    return;
}

template <class T>
inline void put(T x, char ch)
{
    if (x < 0)
    {
        putchar('-');
        x = ~ x + 1;
    }
    _put(x);
    putchar(ch);

    return;
}

const int MaxN = 3e6 + 5, INF = 0x3f3f3f3f;
int s[MaxN], T, n;
int rmq[MaxN][20], logn[MaxN];
int b[MaxN], w[MaxN], rnk[MaxN], sa[MaxN], hght[MaxN], lcp[MaxN], len;
char s1[MaxN], t[MaxN];

inline void init_in()
{
    n = 0;
    memset(s, 0, sizeof(s));

    int m;
    get(m);
    scanf("%s", s1);
    For(i, 2, m)
    {
        scanf("%s", t);
        For(j, 0, strlen(t) - 1)
            s[n++] = t[j] - 'a';
        s[n++] = 26;
    }
    For(j, 0, (len = strlen(s1)) - 1)
        s[n++] = s1[j] - 'a';

    return;
}
```

```
inline bool check(int x, int y, int k)
{
    int _x = x + k < n ? b[x + k] : -1, _y = y + k < n ? b[y + k] : -1;
    return b[x] == b[y] && _x == _y;
}

inline void init_sa() //求 sa 数组和 rank 数组
{
    int num;
    For(i, 0, 26) w[i] = 0;
    For(i, 0, n - 1) w[s[i]]++;
    For(i, 1, 26) w[i] += w[i - 1];
    Rof(i, n - 1, 0) sa[--w[s[i]]] = i;
    rnk[sa[0]] = 0, num = 1;
    For(i, 1, n - 1)
        rnk[sa[i]] = s[sa[i]] == s[sa[i - 1]] ? num - 1 : num++;

    int m;
    for (int k = 1; num < n; k <<= 1)
    {
        m = 0;
        For(i, n - k, n - 1) b[m++] = i;
        For(i, 0, n - 1)
            if (sa[i] >= k) b[m++] = sa[i] - k;
        For(i, 0, num - 1) w[i] = 0;
        For(i, 0, n - 1) w[rnk[b[i]]]++;
        For(i, 1, num - 1) w[i] += w[i - 1];
        Rof(i, n - 1, 0) sa[--w[rnk[b[i]]]] = b[i];
        For(i, 0, n - 1) b[i] = rnk[i];
        rnk[sa[0]] = 0, num = 1;
        For(i, 1, n - 1)
            rnk[sa[i]] = check(sa[i], sa[i - 1], k) ? num - 1 : num++;
    }
    return;
}

inline void init_height() //求 height 数组
{
    int l = 0, x;
    For(i, 0, n - 1)
    {
        l ? l-- : 0;
        if (!rnk[i]) continue;
        x = sa[rnk[i] - 1];
        while (s[x + l] == s[i + l] && std::max(x, i) + l < n) l++;
        hght[rnk[i]] = l;
    }

    return;
}

inline void ckmin(int &x, const int &y)
{
    if (x > y) x = y;
    return;
}

inline void init_lcp()
{
    logn[0] = -1;
    For(i, 1, n) logn[i] = logn[i >> 1] + 1;
    For(i, 1, n - 1)
        rmq[i][0] = hght[i];

    For(j, 1, 18)
    For(i, 1, n - 1 - (1 << j))
        ckmin(rmq[i][j] = rmq[i][j - 1], rmq[i + (1 << j - 1)][j - 1]);
    return;
}

inline int query_lcp(int l, int r) //求 suffix(l-1)和 suffix(r)的 lcp
{
    int k = logn[r - l + 1];
    return std::min(rmq[l][k], rmq[r - (1 << k) + 1][k]);
}

inline void ckmax(int &x, const int &y)
{
    if (x < y) x = y;
    return;
}

inline void query()
{
    int ans = INF, pos, lst;

    For(i, 0, n - 1)
```

```
        lcp[i] = 0;

    lst = 0;
    while (lst < n && sa[lst] + len >= n) lst++; //求串 s1 的后缀与其余串的
后缀的 lcp 的最大值的最小值
    For(i, lst + 1, n - 1)
        if (sa[i] + len < n) lst = i;
        else ckmax(lcp[i], query_lcp(lst + 1, i));
    lst = n - 1;
    while (lst >= 0 && sa[lst] + len >= n) lst--;
    Rof(i, lst - 1, 0)
        if (sa[i] + len < n) lst = i;
        else ckmax(lcp[i], query_lcp(i + 1, lst));
    For(i, 0, n - 1)
    if (sa[i] + len >= n && sa[i] + lcp[i] < n && lcp[i] + 1 < ans)
        ans = lcp[i] + 1, pos = sa[i];

    if (ans ^ INF)
    {
        For(i, pos, pos + ans - 1)
            putchar(s[i] + 'a');
        putchar('\n');
    }
    else
        puts("Impossible");
    return;
}

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

    get(T);
    For(tsk, 1, T)
    {
        init_in();
        init_sa();
        init_height();
        init_lcp();

        printf("Case #%d: ", tsk);
        query();
    }
    fclose(stdin), fclose(stdout);
    return 0;
}
```

串(BZOJ3277)

【算法分析】

首先在所有串的末尾加上一个特殊的分隔字符（字典序比串中的所有字符都要小），然后把所有串拼接起来，对这个拼接起来的新串进行后缀排序。由于这个特殊分隔字符的存在，原来所有串后缀的字典序大小关系不会互相影响，这一过程也就相当于把每个串的后缀全部取出按字典序大小排序。

如果一个串的某个子串满足条件，那么这个子串的子串也必然满足条件。我们枚举原来每个串的每个后缀，二分出这个后缀最长的满足条件的前缀，那么这个后缀对答案的贡献就是最长的满足条件的前缀的长度。

考虑怎样检查二分的答案是否合法。我们用 ST 表预处理出 height 数组的区间最小值。对于二分的前缀长度 len，先找到这个后缀在排序之后的位置 x，然后从 x 分别向左和向右二分得到长度最长的区间[L,R]，满足[L,R]内的每个后缀与我们枚举的后缀的最长公共前缀的长度大于等于 len。那么现在我们只要检查[L,R]内是否包含了至少 k 个串的后缀，这相当于查询区间内的数字种数。

对于这类问题，我们有一个单次询问$O(\log n)$的经典在线做法，即在主席树中记录数字上一次出现的位置。但我们注意到这道题中只需要确定大小关系，并且 k 是给定的，实际上有一个更为简单的判断方法。我们可以再预处理一个数组 left[i]，表示满足[j,i]内包含了至少 k 个串后缀的最大的 j（找不到合法的 j 时 left[i] = 0），这可以通过使用双指针并记录区间内每种数字的出现次数以及数字种数$O(n)$得到。那么[L,R]合法当且仅当 left[R]≥L，总的时间复杂度$O(n\log^2 n)$。

考虑能否做进一步的优化，注意到如果一个以 L 为左端点的后缀最后得到的最长前缀所在的区间为[L,R]，那么以 L+1 为左端点的后缀最后得到的区间至少为[L+1,R]。我们只要正序枚举每个串后缀的左端点，用一个指针记录当前的右端点，每次不断尝试将右端点右移即可去除外层的二分，最后总的时间复杂度$O(n\log n)$。

【参考程序】

```
//陈贤
#include <bits/stdc++.h>

typedef long long ll;
const int N = 2e5 + 5;
int sa[N], f[20][N], rank[N], w[N], Log[N], height[N];
char a[N], s[N];
int n, K, r, sm;
int bel[N], tl[N], tr[N], lef[N], cnt[N];
```

```
template <class T>
inline void put(T x)
{
    if (x > 9) put(x / 10);
    putchar(x % 10 + 48);
}

template <class T>
inline T Min(T x, T y) {return x < y ? x : y;}
template <class T>
inline void CkMax(T &x, T y) {x < y ? x = y : 0;}

inline bool in_equal(int *x, int i, int j, int k)
{
    if (x[i] != x[j])
        return false;
    else
    {
        int p = i + k > n ? -1 : x[i + k],
            q = j + k > n ? -1 : x[j + k];
        return p == q;
    }
}

inline int queryMin(int l, int r)
{
    int k = Log[r - l + 1];
    return Min(f[k][l], f[k][r - (1 << k) + 1]);
}

inline void buildSA()
{ //建立后缀数组
    int *x = rank, *y = height;
    r = 255;
    for (int i = 1; i <= r; ++i)
        w[i] = 0;
    for (int i = 1; i <= n; ++i)
        ++w[s[i]];
    for (int i = 2; i <= r; ++i)
        w[i] += w[i - 1];
    for (int i = n; i >= 1; --i)
        sa[w[s[i]]--] = i;
    x[sa[1]] = r = 1;
```

```
for (int i = 2; i <= n; ++i)
    x[sa[i]] = s[sa[i - 1]] == s[sa[i]] ? r : ++r;
for (int k = 1; r < n; k <<= 1)
{
    int yn = 0;
    for (int i = 1; i <= n; ++i)
        if (sa[i] + k > n)
            y[++yn] = sa[i];
    for (int i = 1; i <= n; ++i)
        if (sa[i] > k)
            y[++yn] = sa[i] - k;

    for (int i = 1; i <= r; ++i)
        w[i] = 0;
    for (int i = 1; i <= n; ++i)
        ++w[x[i]];
    for (int i = 2; i <= r; ++i)
        w[i] += w[i - 1];
    for (int i = n; i >= 1; --i)
        sa[w[x[y[i]]]--] = y[i];

    std::swap(x, y);
    x[sa[1]] = r = 1;
    for (int i = 2; i <= n; ++i)
        x[sa[i]] = in_equal(y, sa[i - 1], sa[i], k) ? r : ++r;
}
for (int i = 1; i <= n; ++i)
    rank[i] = x[i];

height[1] = 0;
for (int i = 1, j, k; i <= n; ++i)
{
    if (rank[i] == 1)
        continue;
    k ? --k : 0;
    j = sa[rank[i] - 1];
    while (i + k <= n && j + k <= n && s[i + k] == s[j + k])
        ++k;
    height[rank[i]] = k;
}
Log[0] = -1;
for (int i = 1; i <= n; ++i)
    f[0][i] = height[i], Log[i] = Log[i >> 1] + 1;
for (int j = 1, jm = Log[n]; j <= jm; ++j)
```

```
        for (int i = 1; i + (1 << j) - 1 <= n; ++i)
            f[j][i] = Min(f[j - 1][i], f[j - 1][i + (1 << j - 1)]);
}

inline bool check(int L, int R)
{ //检验[L,R]代表的子串是否合法
    int x = rank[L], y = R - L + 1;

    int tl = 1, tr = x - 1, xl = x, xr = x;
    while (tl <= tr)
    {
        int mid = tl + tr >> 1;
        if (queryMin(mid + 1, x) >= y)
            xl = mid, tr = mid - 1;
        else
            tl = mid + 1;
    }

    tl = x + 1; tr = n;
    while (tl <= tr)
    {
        int mid = tl + tr >> 1;
        if (queryMin(x + 1, mid) >= y)
            xr = mid, tl = mid + 1;
        else
            tr = mid - 1;
    }
    return xl <= lef[xr];
}

int main()
{
    scanf("%d%d", &n, &K);
    int tn = n;
    for (int i = 1; i <= n; ++i)
    {
        scanf("%s", a + 1);
        int len = strlen(a + 1);
        tl[i] = sm + 1;
        for (int j = 1; j <= len; ++j)
            s[++sm] = a[j], bel[sm] = i;
        tr[i] = sm;
        s[++sm] = 'a' - 1;
    }
```

```
n = sm;
buildSA();
int j = 1, num = 0;
for (int i = 1; i <= n; ++i)
{  //two pointers 预处理 left 数组
    if (bel[sa[i]])
    {
        if (++cnt[bel[sa[i]]] == 1)
            ++num;
    }
    while (num > K)
    {
        int x = bel[sa[j]];
        if (x)
            --cnt[x] == 0 ? --num : 0;
        ++j;
    }
    while (num >= K)
    {
        int x = bel[sa[j]];
        if (x)
        {
            if (cnt[x] > 1)
                --cnt[x];
            else
                break;
        }
        ++j;
    }
    if (num >= K)
        lef[i] = j;
}
for (int i = 1; i <= tn; ++i)
{
    int r = 0;
    ll res = 0;
    for (int l = tl[i]; l <= tr[i]; ++l)
    {
        CkMax(r, l - 1);
        while (r < tr[i] && check(l, r + 1))
            ++r;
        res += r - l + 1;
    }
```

```
        put(res), putchar(' ');
    }
    return 0;
}
```

历史的行程(history,3s,1024MB)

【算法分析】

很容易想到把串倒过来之后，建立后缀数组。

我们的询问相当于求

$$\max_{i,j\in[l,r],rank_i<rank_j}\ \min_{k\in(rank_i,rank_j]} height[k]$$

直接求这个式子的值，复杂度难以承受。

不妨考虑这个式子的值$\geq$ val的充要条件：设集合$\mathrm{S}=\{\mathrm{rank_i}+1|\mathrm{l}\leq \mathrm{i}\leq \mathrm{r}\}$，$\mathrm{T}=\{\mathrm{rank_i}|\mathrm{l}\leq \mathrm{i}\leq \mathrm{r}\}$，那么这个条件就是我们能够选出$\mathrm{u}\in \mathrm{S},\mathrm{v}\in \mathrm{T},\mathrm{u}\leq \mathrm{v}$使得区间$[\mathrm{u},\mathrm{v}]$不包含小于val的数。

这给我们的启示是，我们可以把询问离线，从大到小把height值插入并用并查集维护连续已经被插入的段，再次过程中处理询问。我们需要在一个段$\mathrm{height}[\mathrm{x}\ldots \mathrm{y}]$上维护一个询问集合：

$S_{[x,y]}=\{\mathrm{i}|\mathrm{sa_j}\in[\mathrm{l_i},\mathrm{r_i}],\mathrm{j}\in[\mathrm{x}-1,\mathrm{y}]\}$，即如果一个询问区间$[\mathrm{l_i},\mathrm{r_i}]$中存在一个p使得$\mathrm{rank_p}\in[x-1,y]$则$\mathrm{i}\in \mathrm{S}_{[x,y]}$，反之亦然。

当$\mathrm{height}[\mathrm{x}]$被插入时，x会和$\mathrm{x}-1$所在的段以及$\mathrm{x}+1$所在的段合并成一段

根据上面提到的询问结果$\geq$ val的充要条件，可以发现如果一个询问同时出现在$\mathrm{x}-1$和$\mathrm{x}+1$所在段对应的集合内，那么这个询问的结果$\geq height[x]$。此外，由于我们从大到小插入了height值，故如果一个询问出现在$\mathrm{x}-1$和$\mathrm{x}+1$所在段对应的集合交内，且这个询问的答案还没被确定下来，那么就可以把这个询问的结果定为$height[x]$，然后把$\mathrm{x}-1$和$\mathrm{x}+1$所在段合并起来，合并后段的S为这两个段S的并集。注意$\mathrm{x}-1$或$\mathrm{x}+1$还未被插入的情况。

具体地，可以使用 bitset 维护集合，再维护一个集合$haveAns$表示还未被确定的询问集合，那么这一步已经确定答案的询问集合就是$\mathrm{S}_{[\ldots,x-1]}\cap S_{[x+1,\ldots]}\cap haveAns$。

还有一个问题：开O(n)个 bitset 的空间无法承受。

考虑取一个小参数T，[2,8]左右，把所有的询问分成T组，对每一组询问进行处理，那么这样空间就只需要开到$\mathrm{O}(\frac{nm}{T\omega})$即可。

时间$\mathrm{O}(\frac{\mathrm{nm}}{\omega})$，空间$\mathrm{O}(\frac{\mathrm{nm}}{\mathrm{T}\omega})$，但T取得越大，程序的时间常数越大。

【参考程序】

```
//陈栉旷
#include <bits/stdc++.h>

inline int read()
{
    int res = 0; bool bo = 0; char c;
    while (((c = getchar()) < '0' || c > '9') && c != '-');
    if (c == '-') bo = 1; else res = c - 48;
    while ((c = getchar()) >= '0' && c <= '9')
        res = (res << 3) + (res << 1) + (c - 48);
    return bo ? ~res + 1 : res;
}

template <class T>
inline T Min(const T &a, const T &b) {return a < b ? a : b;}

typedef std::set<int>::iterator it;

const int N = 1e5 + 5, M = 25005;

int n, m, sa[N], rank[N], height[N], w[N], ans[N], fa[N], le[N], ri[N];
char t[N], s[N];
bool vis[N];

std::bitset<M>unr[N], uer[N], haveAns, zzq, yjz, wxh;

struct node
{
    int h, pos;
} a[N];

inline bool comp(node a, node b)
{
    return a.h>b.h;
}

int cx(int x)
{
    if (fa[x] != x) fa[x] = cx(fa[x]);
    return fa[x];
}

void zm(int x, int y)
{
    int ix = cx(x), iy = cx(y);
    if (ix != iy) fa[iy] = ix, uer[ix] |= uer[iy];
}

void buildsa()
{
    int m = 2, *x = rank, *y = height;
    for (int i = 1; i<= n; i++) w[x[i] = s[i] - '0' + 1]++;
    w[2] += w[1];
    for (int i = 1; i<= n; i++) sa[w[x[i]]--] = i;
    for (int k = 1; k < n; k <<= 1, std::swap(x, y))
    {
        int tt = 0;
        for (int i = n - k + 1; i<= n; i++) y[++tt] = i;
        for (int i = 1; i<= n; i++)
            if (sa[i] > k) y[++tt] = sa[i] - k;
        memset(w, 0, sizeof(w));
        for (int i = 1; i<= n; i++) w[x[i]]++;
        for (int i = 2; i<= m; i++) w[i] += w[i - 1];
        for (int i = n; i>= 1; i--) sa[w[x[y[i]]]--] = y[i];
        m = 0;
        for (int i = 1; i<= n; i++)
        {
            int u = sa[i], v = sa[i - 1];
            y[u] = x[u] != x[v] || x[u + k] != x[v + k] ? ++m : m;
        }
        if (m == n) break;
    }
    for (int i = 1; i<= n; i++) rank[sa[i]] = i;
    for (int i = 1, k = 0; i<= n; i++)
    {
        if (k) k--;
        while (s[i + k] == s[sa[rank[i] - 1] + k]) k++;
        height[rank[i]] = k;
    }
}

void divi(int l, int r)
{
    for (int i = 1; i<= 25000; i++) haveAns[i] = 1;
    //haveAns[i]表示第 i 个询问是否还未确定答案
    for (int i = 1; i<= n; i++) fa[i] = i;
    for (int i = 1; i<= n; i++) unr[i].reset();
    memset(vis, 0, sizeof(vis));
    for (int i = l; i<= r; i++)
    {
        unr[le[i]][i - l + 1] = 1;
        if (ri[i] < n) unr[ri[i] + 1][i - l + 1] = 1;
    }
    for (int i = 2; i<= n; i++) unr[i] ^= unr[i - 1];
      //利用差分快速求包含每个位置的询问
    for (int i = 1; i< n; i++)
    {
        int u = a[i].pos;
        vis[u] = 1;
        uer[u] = unr[sa[u]] | unr[sa[u - 1]];
        if (vis[u - 1]) zzq = uer[cx(u - 1)], zm(u - 1, u);
        else zzq = unr[sa[u - 1]];
        if (vis[u + 1]) wxh = uer[cx(u + 1)], zm(u + 1, u);
        else wxh = unr[sa[u]];
        yjz = zzq&wxh&haveAns; //对可以确定答案的询问计算答案
        for (int j = yjz._Find_first(); j < M; j = yjz._Find_next(j))
            haveAns[j] = 0, ans[j + l - 1] = a[i].h;
    }
}

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

    int l, r;
    n = read(); m = read();
    scanf("%s", t + 1);
    for (int i = 1; i<= n; i++) s[n - i + 1] = t[i];
    buildsa();
    for (int i = 1; i<= m; i++)
        ri[i] = n - read() + 1, le[i] = n - read() + 1;
    for (int i = 2; i<= n; i++) a[i - 1] = (node) {height[i], i};
    std::sort(a + 1, a + n, comp);
    for (int i = 1; i<= m; i += 25000) //分段压空间
        divi(i, Min(i + 24999, m));
    for (int i = 1; i<= m; i++) printf("%d\n", ans[i]);
    return 0;
}
```

跳蚤(flea,2s,512MB)

【参考程序】

```
//陈予菲
#include <bits/stdc++.h>

using namespace std;

#define ll long long
```

```
const int e = 2e5 + 5;
int m, n, a[e], b[e], tmp[e], sa[e], he[e], w[e], tot, val, r1[e];
char s[e];
ll mx, ans;

inline void init()
{
    int i, k, r = 0; //后缀排序
    for (i = 1; i<= n; i++) w[s[i]]++;
    for (i = 1; i<= 255; i++) w[i] += w[i - 1];
    for (i = n; i>= 1; i--) sa[w[s[i]]--] = i;
    for (i = 1; i<= n; i++)
    if (s[sa[i]] == s[sa[i - 1]]) a[sa[i]] = r;
    else a[sa[i]] = ++r;
    for (k = 1; r < n; k <<= 1)
    {
        tot = 0;
        for (i = n - k + 1; i<= n; i++) b[++tot] = i;
        for (i = 1; i<= n; i++)
        if (sa[i] > k) b[++tot] = sa[i] - k;
        for (i = 1; i<= n; i++) w[a[b[i]]] = 0;
        for (i = 1; i<= n; i++) w[a[b[i]]]++;
        for (i = 2; i<= n; i++) w[i] += w[i - 1];
        for (i = n; i>= 1; i--) sa[w[a[b[i]]]--] = b[i];
        r = 0;
        for (i = 1; i<= n; i++) tmp[i] = a[i];
        for (i = 1; i<= n; i++)
        if (tmp[sa[i]] == tmp[sa[i - 1]] &&tmp[sa[i]+k] == tmp[sa[i-1]+k])
        a[sa[i]] = r; else a[sa[i]] = ++r;
    }
    int j = 0;
    for (i = 1; i<= n; i++) //预处理 height 数组
    {
        int x = sa[a[i] - 1];
        if (!x)
        {
            j = 0;
            continue;
        }
        j = max(0, j - 1);
        while (s[x + j] == s[i + j] && max(i, x) + j <= n) j++;
        he[a[i]] = j;
    }
}
```

```
inline bool check(ll mid)
{
    int i, st, ed, j, t = 1, pos; ll now = 0;
    for (i = 1; i<= n; i++)
    {
        int cnt = n - sa[i] + 1 - he[i];
        if (now + 1 <= mid && mid <= now + cnt)
        {
            st = sa[i];
            ed = sa[i] + he[i] + mid - now - 1;
            pos = i;
            r1[st] = ed;
            break;
        }
        now += cnt;
        r1[sa[i]] = n;
    }
    if (s[st] != val) return 0;
    int mi = ed - st + 1;   //S[st,ed]是排名为 mid 的子串
    for (i = pos + 1; i<= n; i++)
    {
        int x = sa[i];
        mi = min(mi, he[i]);
        r1[x] = x + mi - 1;
    }
    int nr = n; int lst = 1;
    for (i = 1; i<= n; i++)
    {
        if (nr <i)
        {
            t++;
            if (t > m) return 0; //非法
            nr = r1[i];
            continue;
        }
        nr = min(nr, r1[i]);
    }
    return 1;
}

int main()
{
    freopen("flea.in", "r", stdin);
    freopen("flea.out", "w", stdout);
    scanf("%d\n", &m);
    scanf("%s", s + 1);
    n = strlen(s + 1);
    init();
    int i, st, ed, j = 0;
    for (i = 1; i<= n; i++) mx += n - sa[i] + 1 - he[i], val = max(val,
(int)s[i]);
    ll l = 1, r = mx, now = 0;  //mx是本质不同的子串个数
    while (l <= r)
    {
        ll mid = l + r >> 1;
        if (check(mid))
        {
            ans = mid;
            r = mid - 1;
        }
        else l = mid + 1;
    }
    for (i = 1; i<= n; i++)
    {
        int cnt = n - sa[i] + 1 - he[i];
        if (now + 1 <= ans&&ans<= now + cnt)
        {
            st = sa[i];
            ed = sa[i] + he[i] + ans - now - 1;
            break;
        }
        now += cnt;
    }
    for (i = st; i<= ed; i++) putchar(s[i]);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

往事树(recollection,2s,512MB)

【算法分析】

首先考虑 Lcs，显然 Lcs(r(u), r(v))就是 Lca(u, v)的深度。那么问题转化为对于每个点 u，找出 u 子树中 Lcp 最大的两个点 x,y，并用 deep(u)+Lcp(r(x),r(y))更新答案。注意可能找出来的 x,y 可能不满足 Lca(x,y)=u，但这并不影响答案。因为在这种情况下，deep(u) < Lcs(r(x),r(y))，一定不是最优解。

考虑怎么找 Lcp 最大的两个点 x,y：首先求树上 SA 以及 height 数组，并用 st 表预处理

height 的区间最小值。显然 r(x),r(y)一定是子树内排名相邻的两个后缀。

考虑对每个点 u 维护一棵下标为排名的线段树，线段树的区间[l, r]记录 u 子树内排名在[l, r]内的最大排名和最小排名。显然将所有子树对应的线段树合并，然后再插入 u 这个点，就是 u 的线段树了。可以在合并线段树的时候，对于线段树上的每个点 x，都用 x 左子节点的排名最大的点和右子节点排名最小的点更新答案。

时间复杂度和空间复杂度都是 O(n log n)。

【参考程序】

```
//陈予菲
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <queue>
#include <cctype>

using namespace std;

#define ll long long

template <class t>
inline void read(t & res)
{
    char ch;
    while (ch = getchar(), ch < '0' || ch > '9');
    res = ch - 48;
    while (ch = getchar(), ch >= '0' && ch <= '9')
    res = res * 10 + ch - 48;
}

const int e = 2e5 + 5, inf = 0x3f3f3f3f;
int f[e][18], logn, nxt[e], go[e], adj[e], len[e], n, num, dep[e],
knar[e][18], pool;
int ans, fav[e], sum[e], tmp[e], sa[e], nowt, rt[e], val[e][18], h[e],
rk[e];
int lo[e], nowu;
struct point
{
    int l, r, ld, rd;
}c[e * 19];

inline void add(int x, int y, int z)
{
    nxt[++num] = adj[x];
    adj[x] = num;
    go[num] = y;
    len[num] = z;
}

inline void dfs1(int u)
{
    int i;
    for (i = 0; i < logn; i++) f[u][i + 1] = f[f[u][i]][i];
    for (i = adj[u]; i; i = nxt[i])
    {
        int v = go[i];
        f[v][0] = u;
        fav[v] = len[i];
        dep[v] = dep[u] + 1;
        dfs1(v);
    }
}

inline void init_sa() //树上 sa
{
    int i, j, r = 1;
    for (i = 1; i <= n; i++) sum[fav[i]]++;
    for (i = 1; i <= 301; i++) sum[i] += sum[i - 1];
    for (i = 1; i <= n; i++) sa[sum[fav[i]]--] = i;
    knar[sa[1]][0] = 1;
    for (i = 2; i <= n; i++)
    knar[sa[i]][0] = (fav[sa[i]] == fav[sa[i - 1]] ? r : ++r);
    for (j = 0; j < logn; j++)
    {
        for (i = 0; i <= r; i++) sum[i] = 0;
        for (i = 1; i <= n; i++) sum[knar[f[i][j]][j]]++;
        for (i = 1; i <= r; i++) sum[i] += sum[i - 1];
        for (i = 1; i <= n; i++)
        tmp[sum[knar[f[i][j]][j]]--] = i;
        for (i = 0; i <= r; i++) sum[i] = 0;
        for (i = 1; i <= n; i++) sum[knar[i][j]]++;
        for (i = 1; i <= r; i++) sum[i] += sum[i - 1];
        for (i = n; i >= 1; i--)
        sa[sum[knar[tmp[i]][j]]--] = tmp[i];
        r = 1;
        knar[sa[1]][j + 1] = 1;
        for (i = 2; i <= n; i++)
        knar[sa[i]][j + 1] = (knar[sa[i - 1]][j] != knar[sa[i]][j] ||
        knar[f[sa[i - 1]][j]][j] != knar[f[sa[i]][j]][j] ? ++r : r);
    }
    for (i = 1; i <= n; i++) rk[sa[i]] = i;
}

inline int jump(int x, int y) //倍增求 height
{
    int i, res = 0;
    for (i = logn; i >= 0; i--)
    if (f[x][i] && f[y][i] && knar[x][i] == knar[y][i])
    {
        x = f[x][i];
        y = f[y][i];
        res += 1 << i;
    }
    return res;
}

inline void init_rmq() //预处理 height 区间最小值
{
    int i, j;
    lo[0] = -1;
    for (i = 1; i < n; i++) lo[i] = lo[i >> 1] + 1;
    for (i = 1; i < n; i++) val[i][0] = jump(sa[i], sa[i + 1]);
    for (j = 1; j <= logn; j++)
    for (i = 1; i + (1 << j) <= n; i++)
    val[i][j] = min(val[i][j - 1], val[i + (1 << j - 1)][j - 1]);
}

inline int query(int x, int y) // 查 lcp(sa[x],sa[y])
{
    if (x > y) swap(x, y);
    y--;
    int k = lo[y - x + 1], res = min(val[x][k], val[y - (1 << k) + 1][k]);
    return res;
}

inline void collect(int x)
{
    int l = c[x].l, r = c[x].r;
    c[x].ld = inf; //ld,rd 分别代表区间排名最小、最大的后缀
    c[x].rd = 0;
    if (l)
    {
        c[x].ld = min(c[x].ld, c[l].ld);
        c[x].rd = max(c[x].rd, c[l].rd);
    }
    if (r)
    {
        c[x].ld = min(c[x].ld, c[r].ld);
        c[x].rd = max(c[x].rd, c[r].rd);
    }
}

inline void update(int l, int r, int &x, int s)
{
    if (!x) x = ++pool;
    if (l == r)
    {
        c[x].ld = c[x].rd = s;
        return;
    }
    int mid = l + r >> 1;
    if (s <= mid) update(l, mid, c[x].l, s);
    else update(mid + 1, r, c[x].r, s);
    collect(x);
}

inline int merge(int x, int y)
{
    if (!x || !y) return x ^ y;
    c[x].ld = min(c[x].ld, c[y].ld);
    c[x].rd = max(c[x].rd, c[y].rd);
    c[x].l = merge(c[x].l, c[y].l);
    c[x].r = merge(c[x].r, c[y].r);
    collect(x);
    ans = max(ans,nowt+query(c[c[x].l].rd,c[c[x].r].ld));//合并时更新答案
    return x;
}

inline void dfs2(int u)
{
    if (u != 1) update(1, n, rt[u], rk[u]);
    int i;
    for (i = adj[u]; i; i = nxt[i])
    {
        int v = go[i];
        dfs2(v);
        nowu = u;
        nowt = dep[u];
        rt[u] = merge(rt[u], rt[v]);
    }
}

int main()
{
    freopen("recollection.in", "r", stdin);
    freopen("recollection.out", "w", stdout);
    read(n);
    logn = log(n) / log(2.0);
    int i, x, y, j;
    for (i = 2; i <= n; i++)
    {
        read(x);
        read(y);
        y++;
        add(x, i, y);
    }
    dfs1(1);
    init_sa();
    init_rmq();
    dfs2(1);
    cout << ans << endl;
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```