字符串 0921补

目录

Rabin-Karp 最大重复子矩阵

```
typedef long long LL;
const int maxn = 510;
const LL P = 337;
int N, M;
char source[maxn][maxn];
LL col[maxn][maxn];
LL hash[maxn][maxn];
LL value[maxn];
void buildValue()
{
    value[0] = 1;
    for (int i = 1; i < maxn; i++)
    {
        value[i] = value[i - 1] * P;
    }
}

void init()
{
    scanf("%d%d", &N, &M);
    for (int i = 1; i <= N; i++)
    {
        scanf("%s", source[i] + 1);
    }
    for (int i = 1; i <= M; i++)
    {
        for (int j = 1; j <= N; j++)
        {
            col[i][j] = col[i][j - 1] * P + source[j][i];
        }
    }
    for (int i = 1; i <= N; i++)
    {
        for (int j = 1; j <= M; j++)
        {
            hash[i][j] = hash[i][j - 1] * P + col[j][i];
        }
    }
}

LL getHash(int x1, int y1, int x2, int y2)
{
    return hash[x2][y2] - hash[x1 - 1][y2] * value[x2 - x1 + 1] -
hash[x2][y1
            - 1] * value[y2 - y1 + 1] + hash[x1 - 1][y1 - 1] *
value[x2 - x1
            + 1] * value[y2 - y1 + 1];
}

int verify(int x1, int y1, int x2, int y2, int k)
{
    for (int i = 0; i < k; i++)
    {
        for (int j = 0; j < k; j++)
        {
            if (source[x1 + i][y1 + j] != source[x2 + i][y2 + j])
                return 0;
        }
    }
    return 1;
}
```

```
struct Data
{
    int x, y;
    LL value;

    bool operator<(const Data& p) const
    {
        return value < p.value;
    }
};

Data tmp[maxn * maxn];

int check(int val)
{
    int cnt = 0;
    for (int i = 1; i + val - 1 <= N; i++)
    {
        for (int j = 1; j + val - 1 <= M; j++)
        {
            tmp[cnt].x = i;
            tmp[cnt].y = j;
            tmp[cnt].value = getHash(i, j, i + val - 1, j + val - 1);
            cnt++;
        }
    }
    sort(tmp, tmp + cnt);
    for (int i = 0; i < cnt;)
    {
        LL current = tmp[i].value;
        int begin = i;
        for (; i < cnt && current == tmp[i].value; i++)
            ;
        int end = i - 1;
        if (end - begin + 1 < 2)
            continue;
        for (int j = begin; j <= end; j++)
        {
            for (int k = j + 1; k <= end; k++)
            {
                if (verify(tmp[j].x, tmp[j].y, tmp[k].x, tmp[k].y,
val))
                    return 1;
            }
        }
    }
    return 0;
}

void work()
{
    int low = 1, high = N, ans = -1;
    while (low <= high)
    {
        int mid = (low + high) >> 1;
        if (check(mid))
        {
            ans = mid;
            low = mid + 1;
        }
        else
            high = mid - 1;
    }
    if (ans == -1)
        puts("0");
    else
    {
        int cnt = 0;
        for (int i = 1; i + ans - 1 <= N; i++)
        {
            for (int j = 1; j + ans - 1 <= M; j++)
            {
                tmp[cnt].x = i;
                tmp[cnt].y = j;
                tmp[cnt].value = getHash(i, j, i + ans - 1, j + ans -
1);
                cnt++;
            }
        }
        sort(tmp, tmp + cnt);
        for (int i = 0; i < cnt;)
        {
            LL current = tmp[i].value;
            int begin = i;
            for (; i < cnt && current == tmp[i].value; i++)
                ;
            int end = i - 1;
            if (end - begin + 1 >= 2)
            {
                for (int j = begin; j <= end; j++)
                {
                    for (int k = j + 1; k <= end; k++)
                    {
                        if (verify(tmp[j].x, tmp[j].y, tmp[k].x,
tmp[k].y, ans))
                        {
                            printf("%d\n", ans);
                            printf("%d %d\n", tmp[j].x, tmp[j].y);
                            printf("%d %d\n", tmp[k].x, tmp[k].y);
                            return;
                        }
                    }
                }
            }
        }
    }
}
```

一维可修改目标串匹配多模式串

```
typedef long long LL;
const LL P = 337;
LL value[2000010];

void buildValue()
{
    value[0] = 1;
    for (int i = 1; i <= 2000000; i++)
    {
        value[i] = value[i - 1] * P;
    }
}
const int maxn = 100010;
template<class T>
struct SegNode
{
    T key;
    int flag;
    int left, right;

    int mid()
    {
        return (left + right) >> 1;
    }
};

template<class T>
struct SegTree
{
    SegNode<T> tree[5 * maxn];

    void init(int left, int right, int idx, T value[])
    {
        tree[idx].left = left;
        tree[idx].right = right;
        tree[idx].flag = 0;
        if (left == right)
        {
            tree[idx].key = value[left];
            return;
        }
        int mid = tree[idx].mid();
        init(left, mid, idx << 1, value);
        init(mid + 1, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    void update(int left, int right, int idx, T value)
    {
        //It's a sub-interval, update it here.
        if (left <= tree[idx].left && right >= tree[idx].right)
        {
            tree[idx].key = value;
            return;
        }
        push_down(idx);
        int mid = tree[idx].mid();
        if (left <= mid)
            update(left, right, idx << 1, value);
        if (mid < right)
            update(left, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    T query(int left, int right, int idx)
    {
        //Query result here.
        if (left == tree[idx].left && right == tree[idx].right)
        {
            return tree[idx].key;
        }
        push_down(idx);
        int mid = tree[idx].mid();
        if (right <= mid)
            return query(left, right, idx << 1);
        else if (left > mid)
            return query(left, right, (idx << 1) + 1);
        else
        {
            int len = right - mid;
            return query(left, mid, idx << 1) * value[len] + query(mid
+ 1,
                    right, (idx << 1) + 1);
        }
    }

    void push_down(int idx)
    {
        if (tree[idx].flag)
        {
            tree[idx].flag = 0;
            //left, right, respectively.
        }
    }
    void push_up(int idx)
    {
        int len = tree[(idx << 1) + 1].right - tree[(idx << 1) +
1].left + 1;
        tree[idx].key = tree[idx << 1].key * value[len]
                + tree[(idx << 1) + 1].key;
    }
};

LL hash(char str[])
{
    LL result = 0;
    for (int i = 0; str[i]; i++)
    {
        result *= P;
        result += str[i];
    }
    return result;
}

int N, M;

LL strHash[10010];

LL initValue[100010];

char source[100010];

int BS(LL key)
{
    int low = 1, high = N;
    while (low <= high)
    {
        int mid = (low + high) >> 1;
        if (strHash[mid] == key)
            return mid;
        else if (strHash[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

SegTree<LL> tree;

void init()
{
    scanf("%d", &N);
    for (int i = 1; i <= N; i++)
    {
        scanf("%s", source);
        strHash[i] = hash(source);
    }
    sort(strHash + 1, strHash + 1 + N);
    scanf("%s", source);
    int len = strlen(source);
    for (int i = 0; i < len; i++)
    {
        initValue[i] = source[i];
    }
    tree.init(0, len - 1, 1, initValue);
    scanf("%d", &M);
}

int query(int from, int to)
{
    LL value = tree.query(from, to, 1);
    if (BS(value) != -1)
        return 1;
    return 0;
}

void update(int idx, char value)
{
    tree.update(idx, idx, 1, value);
}
```

二维子串计数

```
typedef unsigned long long LL;
const int maxn = 65536;
const LL P = 397;
int A[maxn], B[maxn], C[maxn], D[maxn], sa[maxn], *rank, *height;
void sortAndRank(int *a1, int *a2, int n, int &m, int j)
{
    int i;
    memset(C, 0, sizeof(C));
    for (i = 0; i < n; i++)
        C[a1[i]]++;
    for (i = 1; i <= m; i++)
        C[i] += C[i - 1];
    for (i = n - 1; i >= 0; i--)
        sa[--C[a1[a2[i]]]] = a2[i];
    a2[sa[0]] = m = 0;
    for (i = 1; i < n; i++)
        a2[sa[i]] = a1[sa[i - 1]] == a1[sa[i]] && a1[sa[i - 1] + j] ==
a1[sa[i]
                + j] ? m : ++m;
}
void da(int* str, int n, int m)
{
    int *a1 = A, *a2 = B, *tmp;
    int i, j, p;
    for (i = 0; i < n; i++)
    {
        a1[i] = i;
        a2[i] = str[i];
    }
    a1[n] = a2[n] = -1;
    sortAndRank(a2, a1, n, m, 0);
    for (j = 1; m < n - 1; j <<= 1)
    {
        p = 0;
        for (i = n - j; i < n; i++)
            a2[p++] = i;
        for (i = 0; i < n; i++)
            if (sa[i] >= j)
                a2[p++] = sa[i] - j;
        sortAndRank(a1, a2, n, m, j);
        tmp = a1;
        a1 = a2;
        a2 = tmp;
    }
    rank = a1;
    height = a2;
}
void calHeight(int *str, int n)
{
    int i, j, k;
    sa[-1] = n;
    for (height[0] = k = i = 0; i < n; i++)
    {
        for (k ? k-- : 0, j = sa[rank[i] - 1]; str[i + k] == str[j +
k]; k++)
            ;
        height[rank[i]] = k;
    }
}
int N, M;
```

```
char source[200][200];
LL pw[200];
void buildPw()
{
    pw[0] = 1;
    for (int i = 1; i < 200; i++)
        pw[i] = pw[i - 1] * P;
}

void init()
{
    scanf("%d%d", &N, &M);
    for (int i = 1; i <= N; i++)
    {
        scanf("%s", source[i] + 1);
    }
}

LL h[200][200];

LL val[40010];

int BS(LL key, int len)
{
    int low = 1, high = len;
    while (low <= high)
    {
        int mid = (low + high) >> 1;
        if (val[mid] == key)
            return mid;
        else if (val[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

void unique(LL arr[], int &len)
{
    int c = len;
    len = 0;
    for (int i = 1; i <= c;)
    {
        LL current = arr[i];
        arr[++len] = current;
        for (; i <= c && current == arr[i]; i++)
            ;
    }
}

int str[maxn];

void work()
{
    LL result = 0;
    for (int w = 1; w <= M; w++)
    {
        int C = 0;
        for (int i = 1; i <= N; i++)
        {
            h[i][1] = 0;
            for (int j = 1; j <= w; j++)
            {
                h[i][1] += source[i][j] * pw[w - j];
            }
            val[++C] = h[i][1];
        }
        for (int i = 1; i <= N; i++)
        {
            for (int j = 2; j + w - 1 <= M; j++)
            {
                h[i][j] = (h[i][j - 1] - source[i][j - 1] * pw[w - 1])
* P
                        + source[i][j + w - 1];
                val[++C] = h[i][j];
            }
        }
        sort(val + 1, val + 1 + C);
        unique(val, C);
        for (int i = 1; i <= N; i++)
        {
            for (int j = 1; j + w - 1 <= M; j++)
            {
                h[i][j] = BS(h[i][j], C);
            }
        }
        LL sum = 0;
        int cnt = 0;
        for (int j = 1; j + w - 1 <= M; j++)
        {
            for (int i = 1; i <= N; i++)
            {
                str[cnt++] = h[i][j];
            }
            str[cnt++] = C + j;
        }
        str[cnt] = 0;
        da(str, cnt, C + M);
        calHeight(str, cnt);
        LL tmp = 0;
        for (int i = 0; i < cnt; i++)
        {
            tmp += height[i];
        }
        result += N * (N + 1) / 2 * (M - w + 1) - tmp;
    }
    cout << result << endl;
}

int main()
{
#ifndef ONLINE_JUDGE
    freopen("p4029", "r", stdin);
#endif
    buildPw();
    int t;
    scanf("%d", &t);
    for (int i = 1; i <= t; i++)
    {
        printf("Case #%d: ", i);
        init();
        work();
    }
    return 0;
}
```

AC自动机

```
typedef long long LL;
const int maxK = 4;
const int maxM = 110;
struct TreeNode
{
    TreeNode *next[maxK];
    TreeNode *fail;
    bool accept;
    int count;
    int id;
    void init(TreeNode *fl, int i)
    {
        accept = false;
        fail = fl;
        id = i;
        count = 0;
        memset(next, 0, sizeof(next));
    }
};

//buildHash()
//init(()
//insert()
//finish()
//match(), buildMat()

template<class T>
struct AC
{
    TreeNode *root, *nodes[maxM];
    TreeNode *queue[maxM];
    bool visit[maxM];
    int hash[256];
    int C;
    TreeNode* newNode()
    {
        TreeNode *res = new TreeNode;
        res->init(root, C);
        nodes[C++] = res;
        return res;
    }
    void init()
    {
        C = 0;
        root = NULL;
        root = newNode();
    }
    void insert(char str[])
    {
        TreeNode *current = root;
        for (int i = 0; str[i]; i++)
        {
            if (!current->next[hash[str[i]]])
                current->next[hash[str[i]]] = newNode();
            current = current->next[hash[str[i]]];
        }
        current->accept = true;//be careful of the repetation
        current->count++;
    }
    void finish()//Build Fail
```

```
{
    int head = 0, tail = 0;
    queue[tail++] = root;
    while (head != tail)
    {
        TreeNode *current = queue[head++];
        for (int i = 0; i < maxK; i++)
        {
            if (!current->next[i])
                continue;
            queue[tail++] = current->next[i];
            if (current == root)
                continue;
            for (TreeNode *t = current->fail; t; t = t->fail)
            {
                if (t->next[i])
                {
                    current->next[i]->fail = t->next[i];
                    current->next[i]->accept |= t->next[i]->accept;
                    break;
                }
            }
        }
    }
}
void buildMat(T mat[maxM][maxM])//all legal
{
    for (int i = 0; i < C; i++)
        for (int j = 0; j < C; j++)
            mat[i][j] = 0;
    for (int i = 0; i < C; i++)
    {
        for (int j = 0; j < maxK; j++)
        {
            int flag = 1;
            for (TreeNode *t = nodes[i]; t; t = t->fail)
            {
                if (t->accept)
                    break;
                if (t->next[j])
                {
                    flag = 0;
                    mat[i][t->next[j]->id] += !t->next[j]->accept;
                    break;
                }
            }
            mat[i][0] += flag;
        }
    }
}
void match(char str[])
{
    for (int i = 0; i < C; i++)
        visit[i] = 0;
    TreeNode *current = root;
    for (int i = 0; str[i]; i++)
    {
        int flag = 1;
        for (TreeNode *t = current; t; t = t->fail)
        {
            TreeNode *c = t->next[hash[str[i]]];
            if (c)
            {
                    if (flag)
                    {
                        flag = 0;
                        current = c;
                    }
                    if (visit[c->id])
                        break;
                    visit[c->id] = true;
                    if (c->accept)
                    {
                        //works here

                        //break;
                    }
                    else
                        break;
                }
            }
            current = flag ? root : current;
        }
    }
};
```

后缀数组_不同回文子串数

```
const int maxn = 200010;

//-------------------------此处后缀数组以及RMQ---------------------------

template<class T>
struct SegNode
{
    T key;
    int flag;
    int left, right;

    int mid()
    {
        return (left + right) >> 1;
    }
};
template<class T>
struct SegTree
{
    SegNode<T> tree[5 * maxn];
    void init(int left, int right, int idx)
    {
        tree[idx].left = left;
        tree[idx].right = right;
        tree[idx].flag = 0;
        if (left == right)
        {
            tree[idx].key = 0;
            return;
        }
        int mid = tree[idx].mid();
        init(left, mid, idx << 1);
        init(mid + 1, right, (idx << 1) + 1);
        push_up(idx);
    }
    void init1(int left, int right, int idx, T value[])
    {
        tree[idx].left = left;
        tree[idx].right = right;
        tree[idx].flag = 0;
        if (left == right)
        {
            tree[idx].key = value[left];
            return;
        }
        int mid = tree[idx].mid();
        init1(left, mid, idx << 1, value);
        init1(mid + 1, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    void update(int left, int right, int idx, T value)
    {
        if (left > right)
            return;
        //It's a sub-interval, update it here.
        if (left <= tree[idx].left && right >= tree[idx].right)
        {
            tree[idx].key = value;
            return;
        }
        push_down(idx);
        int mid = tree[idx].mid();
        if (left <= mid)
            update(left, right, idx << 1, value);
        if (mid < right)
            update(left, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    T query(int left, int right, int idx)
    {
        if (left > right)
            return 0;
        //Query result here.
        if (left == tree[idx].left && right == tree[idx].right)
        {
            return tree[idx].key;
        }
        push_down(idx);
        int mid = tree[idx].mid();
        if (right <= mid)
            return query(left, right, idx << 1);
        else if (left > mid)
            return query(left, right, (idx << 1) + 1);
        else
        {
            return max(query(left, mid, idx << 1), query(mid + 1,
right, (idx
                    << 1) + 1));
        }
    }
    void push_down(int idx)
    {
        if (tree[idx].flag)
        {
            tree[idx].flag = 0;
            //left, right, respectively.
        }
    }
    void push_up(int idx)
    {
        tree[idx].key = max(tree[idx << 1].key, tree[(idx << 1) +
1].key);
    }
};

char source[maxn / 2];
char str[maxn];

int N, Len;

int evenLen[maxn];
int oddLen[maxn];

int hash[maxn];

RMQ<int> rmq;

int getLCP(int p, int q)
{
    if (p > q)
        return getLCP(q, p);
    return height[rmq.query(p + 1, q)];
}
```

```
void buildOdd()
{
    for (int i = 0; source[i]; i++)
    {
        oddLen[i] = getLCP(rank[i], rank[hash[i]]);
        //oddLen[i]=min(oddLen[i],i+1);
        //oddLen[i]=min(oddLen[i],N-i);
    }
}

void buildEven()
{
    for (int i = 1; source[i]; i++)
    {
        evenLen[i] = getLCP(rank[i], rank[hash[i - 1]]);
        //evenLen[i]=min(evenLen[i],i);
        //evenLen[i]=min(evenLen[i],N-i);
    }
}

//SegTree<int> tree;
SegTree<int> lenTree;

int stack[maxn], top;

void init()
{
    scanf("%s", source);
    N = strlen(source);
    for (int i = 0; source[i]; i++)
    {
        str[i] = source[i];
        str[i + N + 1] = source[N - i - 1];
        hash[N - 1 - i] = i + N + 1;
    }
    str[N] = 1;
    Len = 2 * N + 1;
    str[Len] = 0;
    da(str, Len, 300);
    calHeight(str, Len);

    rmq.init(height, Len);

    buildOdd();
    buildEven();

    da(source, N, 300);
    calHeight(source, N);
    rmq.init(height, N);
}

void print()
{
    for (int i = 0; source[i]; i++)
    {
        printf("%s\n", source + sa[i]);
    }
}

int mxQueue[maxn];

long long getResult(int arr[])
```

```
{
    mxQueue[0] = 0;
    long long result = 0;
    top = 0;
    //tree.init(0,N,1);
    lenTree.init1(0, N - 1, 1, arr);

    stack[0] = 0;
    result += arr[0];
    for (int i = 1; i < N; i++)
    {
        for (int j = top; j > 0; j--)
        {
            if (height[i] <= height[stack[j]])
            {
                mxQueue[stack[j]] = 0;
                //tree.update(stack[j],stack[j],1,0);
                top--;
            }
            else
                break;
        }
        stack[++top] = i;
        int from = stack[top - 1];
        int to = i - 1;

        int value;

        value = mxQueue[stack[top - 1]];
        //value=tree.query(0,i-1,1);
        int tmp = lenTree.query(from, to, 1);
        int mx = tmp;
        tmp = min(tmp, height[i]);
        value = max(value, tmp);

        int current = arr[i];
        if (current > value)
            result += current - value;

        value = max(arr[i], tmp);
        value = min(value, height[i]);
        mxQueue[i] = max(value, mxQueue[stack[top - 1]]);
        //tree.update(i,i,1,value);
    }
    return result;
}

int tmp[maxn];

void work()
{
    long long result = 0;
    for (int i = 0; i < N; i++)
        tmp[i] = oddLen[sa[i]];
    result += getResult(tmp);
    for (int i = 0; i < N; i++)
        tmp[i] = evenLen[sa[i]];
    result += getResult(tmp);
    cout << result << endl;
}
```

带’?’与’*’的通配符匹配

```
const int maxn = 100010;
const int maxK = 26;
const int maxM = 100010;
struct TreeNode
{
    TreeNode *next[maxK];
    TreeNode *fail;
    bool accept;
    vector<int> len;
    int id;

    void init(TreeNode *fl, int i)
    {
        accept = false;
        fail = fl;
        id = i;
        len.clear();
        memset(next, 0, sizeof(next));
    }
};

//buildHash()
//init(()
//insert()
//finish()
//match(), buildMat()

template<class T>
struct AC
{
//-----------------------此处AC自动机具体实现----------------------------
    int match(char str[], int cnt[], int n)
    {
        TreeNode *current = root;
        int count = 0;
        for (int i = 0; str[i]; i++)
        {
            count++;
            int flag = 1;
            for (TreeNode *t = current; t; t = t->fail)
            {
                TreeNode *c = t->next[str[i] - 'a'];
                if (c)
                {
                    if (flag)
                    {
                        flag = 0;
                        current = c;
                    }
                    if (c->accept)
                    {
                        //works here
                        if (c->len.size())
                        {
                            for (int j = 0; j < c->len.size(); j++)
                            {
                                if (i - c->len[j] < 0)
                                    continue;
                                cnt[i - c->len[j]]++;
                            }
                        }
                        //break;
                    }
                    else
                        break;
                }
            }
            current = flag ? root : current;
        }
        for (int i = 0; i < count; i++)
        {
            if (cnt[i] >= n)
            {
                return i;
            }
        }
        return -1;
    }
};

AC<int> ac;

char source[maxn];
char dest[maxn];
int cnt[maxn];

int match(char source[], char dest[])
{
    int len = 0;
    for (int i = 0; dest[i];)
    {
        dest[len++] = dest[i];
        if (dest[i] != '*')
        {
            i++;
            continue;
        }
        for (; dest[i] && dest[i] == '*'; i++)
            ;
    }
    dest[len] = 0;

    int deltaStart = 0;
    for (int i = 0, j = 0; source[i] && dest[j] && (source[i] ==
dest[j]
            || dest[j] == '?'); i++, j++)
        deltaStart++;
    source = source + deltaStart;
    dest = dest + deltaStart;
    int deltaEnd = 0;
    int N = strlen(source);
    int M = strlen(dest);
    for (int i = N - 1, j = M - 1; i >= 0 && j >= 0 && (source[i] ==
dest[j]
            || dest[j] == '?'); i--, j--)
        deltaEnd++;
    source[N - deltaEnd] = 0;
    dest[M - deltaEnd] = 0;
    N -= deltaEnd;
    M -= deltaEnd;

    if (M)
    {
        if (dest[0] != '*' || dest[M - 1] != '*')
            return 0;
        dest++;
        M--;
        if (!M)
            return 1;
    }
    else
    {
        if (N)
            return 0;
        return 1;
    }

    int i, j;
    for (i = 0, j = 0; source[i] && dest[j]; j++)
    {
        ac.init();
        int begin = j;
        for (; dest[j] && dest[j] != '*'; j++)
            ;
        int end = j - 1;
        memset(cnt, 0, sizeof(cnt));
        int strCnt = 0;
        for (int k = begin; k <= end;)
        {
            if (dest[k] == '?')
            {
                k++;
                continue;
            }
            int bg = k;
            for (; k <= end && dest[k] != '?'; k++)
                ;
            char tmp = dest[k];
            dest[k] = 0;
            ac.insert(dest + bg, bg - begin);
            dest[k] = tmp;
            strCnt++;
        }
        ac.finish();
        int tmp = ac.match(source + i, cnt, strCnt);
        if (tmp == -1)
            return 0;
        if (i + tmp + (end - begin) >= N)
            return 0;
        i += tmp + end - begin + 1;
    }
    if (dest[j])
        return 0;
    return 1;
}

void work()
{
    if (match(source, dest))
        puts("YES");
    else
        puts("NO");
}

int main()
{
    while (scanf("%s%s", source, dest) != EOF)
    {
        work();
    }
    return 0;
}
```

后缀数组+栈扫描求长度不小于K的子串数

```
typedef long long LL;
const int maxn = 200010;
//---------------------------此处后缀数组------------------------------
char P[maxn / 2], Q[maxn / 2], str[maxn];
int Lp, Lq, Len;
int K;

LL l[maxn], r[maxn];
LL sum[maxn];

void init()
{
    scanf("%s%s", P, Q);
    Lp = strlen(P);
    Lq = strlen(Q);
    strcpy(str, P);
    strcat(str, "\001");
    strcat(str, Q);
    Len = Lp + Lq + 1;
    da(str, Len, 300);
    calHeight(str, Len);
    for (int i = 0; i < Len; i++)
        height[i] -= K - 1;
    for (int i = 0; i < Len; i++)
        if (height[i] < 0)
            height[i] = 0;
}

int stack[maxn], top;

void work()
{
    sum[0] = 0;
    for (int i = 1; i < Len; i++)
    {
        sum[i] = sum[i - 1];
        if (sa[i - 1] >= Lp + 1)
            sum[i]++;
    }

    top = 0;
    stack[0] = 0;
    l[0] = 0;
    for (int i = 0; i < Len; i++)
    {
        for (int j = top; j > 0; j--)
        {
            if (height[i] <= height[stack[j]])
                top--;
            else
                break;
        }
        stack[++top] = i;
        l[i] = l[stack[top - 1]] + (sum[i] - sum[stack[top - 1]]) *
(height[i]);
    }

    sum[0] = 0;
    if (sa[0] >= Lp + 1)
        sum[0] = 1;
    for (int i = 1; i < Len; i++)
    {
        sum[i] = sum[i - 1];
        if (sa[i] >= Lp + 1)
            sum[i]++;
    }

    top = 0;
    stack[0] = Len;
    r[Len] = 0;
    r[Len - 1] = 0;
    for (int i = Len - 1; i >= 0; i--)
    {
        for (int j = top; j > 0; j--)
        {
            if (height[i] <= height[stack[j]])
                top--;
            else
                break;
        }
        stack[++top] = i;
        r[i] = r[stack[top - 1]] + (sum[stack[top - 1] - 1] - sum[i -
1])
                * height[i];
    }
    long long result = 0;
    for (int i = 0; i < Len; i++)
    {
        if (sa[i] < Lp)
        {
            result += l[i] + r[i + 1];
        }
    }
    cout << result << endl;
}
```

数位统计_666

```
typedef long long LL;
/*
 * dp[n+1][1] = 9*dp[n][1]+9*dp[n][2]+9*dp[n][3]
 * dp[n+1][2] = dp[n][1]
 * dp[n+1][3] = dp[n][2]
 * dp[n+1][4] = dp[n][3]+10*dp[n][4]
 */
LL dp[30][5];
LL DP(int n, int k)
{
    if (n < 0)
        return 0;
    if (n == 0 && k == 1)
        return 1;
    else if (n == 0)
        return 0;
    else
    {
        if (dp[n][k])
            return dp[n][k];
        if (k == 1)
        {
            dp[n][k] = 9 * DP(n - 1, 1) + 9 * DP(n - 1, 2) + 9 * DP(n
- 1, 3);
        }
        else if (k == 2)
        {
            dp[n][k] = DP(n - 1, 1);
        }
        else if (k == 3)
        {
            dp[n][k] = DP(n - 1, 2);
        }
        else
        {
            dp[n][k] = DP(n - 1, 3) + 10 * DP(n - 1, 4);
        }
    }
    return dp[n][k];
}

int getLen(LL n)
{
    int cnt = 0;
    while (n)
    {
        n /= 10;
        cnt++;
    }
    return cnt;
}

int first(LL n)
{
    int res = 0;
    while (n)
    {
        res = n;
        n /= 10;
    }
    return res;
}

int second(LL n)
{
    int len = getLen(n);
    if (len <= 1)
        return 0;
    LL pw = 1;
    for (int i = 1; i <= len - 2; i++)
        pw *= 10;
    return n / pw % 10;
}

char *delFirst(char str[], int &l)
{
    char *result = str + 1;
    l = 1;
    //for (;*result && *result=='0';result++,l++) ;
    return result;
}

LL pow[18];

void buildPow()
{
    pow[0] = 1;
    for (int i = 1; i <= 17; i++)
        pow[i] = pow[i - 1] * 10;
}

LL below(char str[], int sixCount, int len)
{
    LL result = 0;
    if (sixCount == 3)
    {
        LL n;
        if (len == 0)
            return 1;
        else
            sscanf(str, "%I64d", &n);
        return n + 1;
    }
    if (len <= 0)
        return 0;
    else if (len == 1)
    {
        if (sixCount < 2)
            return 0;
        else
        {
            LL n;
            sscanf(str, "%I64d", &n);
            return n >= 6;
        }
    }
    int firstBit = str[0] - '0';
    for (int i = 0; i < firstBit; i++)
    {
        if (i == 6)
        {
            char ss[20];
            sprintf(ss, "%I64d", pow[len - 1] - 1);
            result += below(ss, sixCount + 1, strlen(ss));
        }
        else
        {
            result += DP(len - 1, 4);
        }
    }
    if (firstBit == 6)
        result += below(str + 1, sixCount + 1, len - 1);
    else
        result += below(str + 1, 0, len - 1);
    return result;
}

LL N;

void work()
{
    LL low = 1, high = 10000000000LL, ans = -1;
    char tmp[20];
    while (low <= high)
    {
        LL mid = (low + high) >> 1;
        sprintf(tmp, "%I64d", mid);
        int len = getLen(mid);
        if (below(tmp, 0, len) >= N)
        {
            ans = mid;
            high = mid - 1;
        }
        else
            low = mid + 1;
    }
    printf("%I64d\n", ans);
}

void init()
{
    scanf("%I64d", &N);
}

int main()
{
    buildPow();   int t;
    scanf("%d", &t);
    for (int i = 1; i <= t; i++)
    {
        init();
        work();
    }
    return 0;
}
```