数据结构

主席树

```
#include <bits/stdc++.h>
#define ll long long
const int mod = 1e9 + 7;
const int maxn = 1e5 + 100;
struct Node
{
    Node *left,*right;
    int sum;
    void init(Node *l=NULL,Node *r=NULL,int s=0)
    {
        left = l,right = r,sum=s;
    }
}node[maxn*20],*root[maxn];
int num[maxn],a[maxn],p[maxn],cnt,n;
Node *modify(int left,int right,int v,Node *p)
{
    if (v<left || v>right) return p;
    Node *t = &node[cnt++];
    *t = *p;
    ++t->sum;
    if (left==right) return t;
    int mid = (left+right)/2;
    t->left = modify(left,mid,v,p->left);
    t->right = modify(mid+1,right,v,p->right);
    return t;
}
inline void build()
{
    cnt = 0;
    root[0] = &node[cnt++];
    root[0]->init(root[0],root[0],0);
    for (int i=1;i<=n;++i)
        root[i] = modify(1,n,num[i],root[i-1]);
}
int query(int left,int right,int k,Node *x,Node *y)
{
    if (left==right) return left;
    int mid = (left+right)/2;
    int sum = y->left->sum - x->left->sum;
    if (sum>=k) return query(left,mid,k,x->left,y->left);
    return query(mid+1,right,k-sum,x->right,y->right);
}
int main()
{
    return 0;
}
```

二维树状数组

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e3 + 100;
int sum[maxn][maxn];
int n;
int lowbit(int x){return x&(-x);}
void update(int x,int y,int value)
{
    for (int i=x;i<=n;i+=lowbit(i))
        for (int j=y;j<=n;j+=lowbit(j))
            sum[i][j]+=value;
}
LL query(int x,int y)
{
    LL ret=0;
    for (int i=x;i>0;i-=lowbit(i))
        for (int j=y;j>0;j-=lowbit(j))
            ret+=sum[i][j];
    return ret;
}
int main()
{
    return 0;
}
```

Splay

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int mod = 1e9 + 7;
const int maxn = 1e6 + 100;
struct Node
{
    int v,size;
    bool reverse;
    Node *son[2],*father;
}node[maxn],*stk[maxn];
int id,top;
Node *createNode(int v=0,Node *f=NULL){
    Node *t = top>=0? stk[top--] : &node[++id];
    t->v = v;
    t->father = f;
    t->size = 1;
    t->reverse = false;
    t->son[0] = t->son[1] = NULL;
    return t;
}
void freeNode(Node *t) { stk[++top] = t; }

class Splay
{
private:
    Node *root;
    bool son(Node *f,Node *s);
    int size(Node *x);
    void push_up(Node *x);
    void push_down(Node *x);
    void rotate(Node *x);
    void splay(Node *x,Node *y = NULL);
public:
    Splay();
    inline Node* Root();
    void clear();
    void init(int l,int r,int *a);
    void insert(int x,int v);
    void insert(int x,int *a,int n);
    void insert(int x,Node *y);
    void vis(Node *p);
    bool empty();
    Node* erase(int l,int r);
    Node* build(int l,int r,int *a,Node *f=NULL);
    Node* kth(int k,Node *p);
    Node* min(Node *p=NULL);
    Node* max(Node *p=NULL);
    Node* next(Node *x);
    Node* pre(Node *x);
    Node* select(int l,int r);
    Node* reverse(int l,int r);
}splay;
int main()
{

    return 0;
}

bool Splay::son(Node *f,Node *s) { return f->son[1] == s;}
int Splay::size(Node *x){return x? x->size:0;}
void Splay::push_up(Node *x)
{
    x->size = 1+size(x->son[0])+size(x->son[1]);
}
void Splay::push_down(Node *x)
{
    if (x->reverse){
        if (x->son[0]) x->son[0]->reverse ^= 1;
        if (x->son[1]) x->son[1]->reverse ^= 1;
        swap(x->son[0],x->son[1]);
        x->reverse = false;
    }
}
```

```
void Splay::rotate(Node *x)
{
    Node *f = x->father;
    Node *g = f->father;
    push_down(f);
    push_down(x);
    int a = son(f,x);
    int b = !a;
    f->son[a] = x->son[b];
    if (x->son[b]) x->son[b]->father = f;
    x->son[b] = f;
    f->father = x;
    x->father = g;
    if (g) g->son[son(g,f)] = x;
    else root = x;
    push_up(f);
    push_up(x);
}
void Splay::splay(Node *x,Node *y)
{
    while (x->father != y){
        Node *f = x->father;
        Node *g = f->father;
        if (g == y) {rotate(x);}
        else {
            if (son(f,x) ^ son(g,f)) rotate(x),rotate(x);
            else rotate(f),rotate(x);
        }
    }
    //push_up(x);
}
Splay::Splay(){ id=top=-1,root = NULL;}
void Splay::clear(){id=top=-1,root=NULL;}
bool Splay::empty() {return root == NULL;}
inline Node* Splay::Root(){return root;}
Node* Splay::build(int l,int r,int *a,Node *f)
{
    if (l>r) return NULL;
    int mid = (l+r)>>1;
    Node *t = createNode(a[mid],f);
    t->son[0] = build(l,mid-1,a,t);
    t->son[1] = build(mid+1,r,a,t);
    push_up(t);
    return t;
}
void Splay::init(int l,int r,int *a) {root = build(l,r,a,NULL);}
Node* Splay::kth(int k,Node *p)
{
    push_down(p);
    if (k <= size(p->son[0])) return kth(k,p->son[0]);
    int t = size(p->son[0])+1;
    if (k > t) return kth(k-t,p->son[1]);
    splay(p,NULL);
    return p;
}
void Splay::insert(int x,Node *y)
{
    Node* p = kth(x,root);
    push_down(p);
    if (!p->son[1]){
        p->son[1] = y;
        y->father = p;
        p = p->son[1];
    }
    else{
        p = p->son[1];
        push_down(p);
        while (p->son[0]){
            p = p->son[0];
            push_down(p);
        }
        p->son[0] = y;
        y->father = p;
        p = p->son[0];
    }
    splay(p,NULL);
}
void Splay::insert(int x,int v){ insert(x,createNode(v)); }
void Splay::insert(int x,int *a,int n) { insert(x,build(1,n,a)); }
Node* Splay::min(Node *p)
```

```
{
    if (!p) p = root;
    push_down(p);
    while (p && p->son[0]) {
        p = p->son[0];
        push_down(p);
    }
    return p;
}
Node* Splay::max(Node *p)
{
    if (!p) p = root;
    push_down(p);
    while (p && p->son[1]){
        p = p->son[1];
        push_down(p);
    }
    return p;
}
Node* Splay::next(Node *x)
{
    push_down(x);
    if (x->son[1]){
        x = x->son[1];
        push_down(x);
        while (x->son[0]){
            x = x->son[0];
            push_down(x);
        }
        return x;
    }
    Node *y = x->father;
    while (y && son(y,x)){ x = y;  y = x->father; }
    return y;
}
Node* Splay::pre(Node *x)
{
    push_down(x);
    if (x->son[0]){
        x = x->son[0];
        push_down(x);
        while (x->son[1]) {
            x = x->son[1];
            push_down(x);
        }
        return x;
    }
    Node *y = x->father;
    while (y && !son(y,x)){ x = y; y = x->father; }
    return y;
}
Node* Splay::select(int l,int r)
{
    Node *x = kth(l,root);
    Node *y = kth(r,root);
    Node *p = pre(x);
    splay(p,NULL);
    Node *t = next(y);
    splay(t,root);
    return t->son[0];
}
Node* Splay::reverse(int l,int r)
{
    Node *x = select(l,r);
    x->reverse ^= 1;
    return x;
}
void Splay::vis(Node *p)
{
    push_down(p);
    if (p->son[0])  vis(p->son[0]);
    printf("%d ",p->v);
    if (p->son[1]) vis(p->son[1]);
}
Node* Splay::erase(int l,int r)
{
    Node* x = select(l,r);
    Node* f = x->father;
    f->son[son(f,x)] = NULL;
    splay(f,NULL);
    return x;
}
```

字符串

KMP

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e6 + 100;
int next[maxn];
char s[maxn], str[maxn];
void getNext(char *s,int *next)
{
    int k=-1;
    int len=strlen(s);
    next[0]=-1;
    for (int i=1;i<len;++i){
        while (k>=0&&s[k+1]!=s[i]) k=next[k];
        if (s[k+1]==s[i]) ++k;
        next[i]=k;
    }
}
int KMPmatch(char *s,char *str,int *next)
{
    getNext(s,next);
    int len=strlen(s);
    int k=-1;
    for (int i=0;str[i];++i){
        while (k>=0&&s[k+1]!=str[i]) k=next[k];
        if (s[k+1]==str[i]) ++k;
        if (k+1==len)
            return i-len+1;
    }
}
int main()
{
    cin>>str>>s;
    cout<<KMPmatch(s,str,next)<<endl;
    return 0;
}
```

AC自动机

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int mod = 1e9 + 7;
const int maxn = 1e6 + 100;
#define N 26
struct Node
{
    int count;
    Node *fail,*next[N];
    void init(int c=0,Node *f=NULL)
    {
        CL(next);
        fail = f;
        count = c;
    }
}node[maxn];
int cnt;
char s[maxn];

inline void insert(Node *p,char *s)
{
    for (int i=0;s[i];++i){
        int id = s[i]-'a';
        if (!p->next[id]){
            p->next[id] = &node[cnt++];
            p->next[id]->init();
        }
        p = p->next[id];
    }
    ++p->count;
}
inline void buildAC(Node *root)
{
    queue<Node*> Q;
    Node *p = root;
    p->fail = NULL;
    Q.push(p);
    Node *t;
    while (!Q.empty()){
        p = Q.front();
        Q.pop();
        for (int i=0;i<N;++i){
            if (!p->next[i]) continue;
            Q.push(p->next[i]);
            if (p == root) p->next[i]->fail = root;
            else {
                t = p->fail;
                while (t){
                    if (t->next[i]){
                        p->next[i]->fail = t->next[i];
                        break;
                    }
                    t = t->fail;
                }
                if (!t) p->next[i]->fail = root;
            }
        }
    }
}
inline int query(Node *root,char *s)
{
    Node *p = root;
    Node *t;
    int ans = 0;
    for (int i=0;s[i];++i){
        int id = s[i]-'a';
        while (p!=root && p->next[id]==NULL)
            p = p->fail;
        p = p->next[id];
        if (!p) p = root;
        t = p;
        while (t!=root && t->count!=-1){
            ans += t->count;
            t->count = -1;
            t=t->fail;
        }
    }
    return ans;
}
int main()
{
    return 0;
}
```

LCA

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
const int mod = 1e9 + 7;
const int maxn = 1e5 + 100;
vector<int> G[maxn];
int grand[maxn][17],depth[maxn];
int Log[maxn],a[maxn],b[maxn],tmp[maxn];
int n,u,v;
inline void init(int n)
{
    for (int i=1;i<=n;++i)
        G[i].clear();
    CL(depth);
    CL(grand);
}
void dfs(int from=0,int u=1)
{
    if (depth[u]) return ;
    depth[u] = depth[from]+1;
    grand[u][0] = from;
    for (int i=1;depth[u]-(1<<i) > 0; ++i)
```

```
        grand[u][i] = grand[grand[u][i-1]][i-1];
    for (int i=0;i<G[u].size();++i)
        dfs(u,G[u][i]);
}

inline int lca(int a,int b)
{
    if (depth[a]<depth[b])
        swap(a,b);

    int t=depth[a]-depth[b];
    for (int i=0;t;t>>=1,++i)
        if (t&1) a = grand[a][i];
    if (a==b) return a;

    for (int i=Log[n];i>=0;--i){
        if (grand[a][i]!=grand[b][i]){
            a = grand[a][i];
            b = grand[b][i];
        }
    }
    return grand[a][0];
}


int main()
{

    return 0;
}
```

后缀数组

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int mod = 1e9 + 7;
const int maxn = 1e6 + 100;
const int maxm = 1e2 + 100;
int wa[maxn],wb[maxn],wu[maxn],wv[maxn],sa[maxn];
char s[maxn];
bool cmp(int *y,int a,int b,int j)
{
    return y[a] == y[b] && y[a+j] == y[b+j];
}
void Suffix(char *s,int *sa,int n,int m)
{
    int *x = wa, *y = wb,i,j,k,p;
    for (i = 0;i < m;++i) wu[i] = 0;
    for (i = 0;i < n;++i) ++wu[x[i]=s[i]];
    for (i = 1;i < m;++i) wu[i] += wu[i-1];
    for (i = n-1;i >= 0;--i) sa[--wu[x[i]]] = i;
    for (p = 1,j = 1;p < n;j <<= 1,m = p) {
        for (p=0,i=n-j;i < n;++i) y[p++] = i;
        for (i = 0;i < n;++i) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (i = 0;i < m;++i) wu[i] = 0;
        for (i = 0;i < n;++i) wv[i] = x[y[i]];
        for (i = 0;i < n;++i) ++wu[wv[i]];
        for (i = 1;i < m;++i) wu[i] += wu[i-1];
        for (i = n-1;i >= 0;--i) sa[--wu[wv[i]]] = y[i];
        swap(x,y);
        for (x[sa[0]]=0,p=1,i=1;i < n;++i)
            x[sa[i]] = cmp(y,sa[i-1],sa[i],j)? p-1:p++;
    }
}
int height[maxn],Rank[maxn];
void calheight(char *s,int *sa,int n)
{
    int j,k;
    for (int i = 1;i <= n;++i) Rank[sa[i]] = i;
    for (int i = 0,k=0;i < n;height[Rank[i++]]=k)
        for (k? k--:0,j=sa[Rank[i]-1];s[i+k] == s[j+k];++k);
}
int main()
{

    return 0;
}
```