# 1 线性基

## 性质

1. 原序列里面的任意一个数都可以由线性基里面的一些数异或得到。
2. 线性基里面的任意一些数异或起来都不能得到 0
3. 线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的

## 用法

1. 线性基可以套在线段树上使用，线段树区间合并的时候将一个区间的所有值插入到另外一个区间即可。
2. 不支持删除操作。遇到需要删除操作的线性基考虑用时间分治。

```
struct Linear_Basis {
LL b[63],nb[63],tot;
void init() {
    tot=0;
    memset(b,0,sizeof(b));
    memset(nb,0,sizeof(nb));
}
bool ins(LL x) {
    for(int i=62;i>=0;i--)
        if (x&(1LL<<i)){
            if (!b[i]) {b[i]=x;break;}
            x^=b[i];
        }
    return x>0;
}
LL Max(LL x){
    LL res=x;
    for(int i=62;i>=0;i--)
        res=max(res,res^b[i]);
    return res;
}
LL Min(LL x){
    LL res=x;
    for(int i=0;i<=62;i++)
        if (b[i]) res^=b[i];
    return res;
}
 void rebuild(){
    for(int i=62;i>=0;i--)
        for(int j=i-1;j>=0;j--)
            if (b[i]&(1LL<<j)) b[i]^=b[j];
```

```
        for(int i=0;i<=62;i++)
            if (b[i]) nb[tot++]=b[i];
    }
    LL Kth_Max(LL k){
        LL res=0;
        for(int i=62;i>=0;i--)
            if (k&(1LL<<i)) res^=nb[i];
        return res;
    }
} LB;
```

# 2 线段树

用法:

1. 区间查询最值、和等等
2. 权值线段树。一般配合离散化使用（离散化注意区间处理为左闭右开）
3. 可持久化线段树可以处理版本问题。
4. 可以遍历线段树处理一些问题 (类似于分治的思想)。

## 2.1 带 LAZY 的朴素线段树

```
//线段树 区间加 区间求和
//带lazy
#include <iostream>
#include <cstring>
using namespace std;
#define MAXN 100100
#define lc o * 2
#define rc o * 2 + 1
#define mid (l + r) / 2
#define LL long long
struct Segment_Tree {
    LL tag[MAXN * 4 + 10];
    LL sum[MAXN * 4 + 10];
    Segment_Tree() {
        memset(tag, 0, sizeof(tag));
        memset(sum, 0, sizeof(sum));
    }
    void update(int o, int l, int r, LL del) {
        sum[o] += (r - l + 1) * del;
        tag[o] += del;
```

```
    }
    void push_down(int o, int l, int r) {
        update(lc, l, mid, tag[o]);
        update(rc, mid + 1, r, tag[o]);
        tag[o] = 0;
    }
    void insert(int o, int l, int r, int qx, int qy, LL del) {
        if (qx <= l && r <= qy) {
            sum[o] += (r - l + 1) * del;
            tag[o] += del;
            return;
        }
        push_down(o, l, r);
        if (qx <= mid)
            insert(lc, l, mid, qx, qy, del);
        if (qy > mid)
            insert(rc, mid + 1, r, qx, qy, del);
        sum[o] = sum[lc] + sum[rc];
    }
    LL query(int o, int l, int r, int qx, int qy) {
        if (qx <= l && r <= qy)
            return sum[o];
        push_down(o, l, r);
        LL t = 0;
        if (qx <= mid)
            t += query(lc, l, mid, qx, qy);
        if (qy > mid)
            t += query(rc, mid + 1, r, qx, qy);
        return t;
    }
} ST;
```

## 2.2 可持久化线段树 (不带修)

1. 注意内存开 MAXN « 5
2. 核心思想是对 1...n 做 n 个线段树，每两个线段树只有 height 个节点不同

```
//可持久化线段树求第k大
#include <algorithm>
#include <cstring>
#include <iostream>
using namespace std;
#define MAXN 200010
```

```
#define mid (l + r) / 2
#define LL long long
int n, m;
int asize;
int a[MAXN];
int b[MAXN];

int get_id(int num) { return lower_bound(a + 1, a + asize + 1, num) - a; }

int root[MAXN];

struct SegmentTree {

    struct node {
        int lc, rc;
        int sum;
        node() { sum = 0; }
    };

    node ST[MAXN << 5];
    int cnt = 0;

    int build(int l, int r) {
        int now = ++cnt;
        if (l == r)
            return now;
        ST[cnt].lc = build(l, mid);
        ST[cnt].rc = build(mid + 1, r);
        return now;
    }

    void init() { build(1, asize); }

    int insert(int o, int l, int r, int k) {
        int now = ++cnt;
        ST[now] = ST[o];
        ST[now].sum++;
        if(l == r)
            return now;
        if (k <= mid)
            ST[now].lc = insert(ST[o].lc, l, mid, k);
        else
            ST[now].rc = insert(ST[o].rc, mid + 1, r, k);
```

```
        return now;
    }

    int query(int u, int v, int l, int r, int k) {
        int num = ST[ST[v].lc].sum - ST[ST[u].lc].sum;
        if (l == r)
            return l;
        if (k <= num)
            return query(ST[u].lc, ST[v].lc, l, mid, k);
        else
            return query(ST[u].rc, ST[v].rc, mid + 1, r, k - num);
    }
} ST;

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++){
        scanf("%d", &a[i]);
        b[i] = a[i];
    }
    sort(a + 1, a + n + 1);
    asize = unique(a + 1, a + n + 1) - a - 1;
    ST.init();
    for (int i = 1; i <= n; i++) {
        root[i] = ST.insert(root[i - 1], 1, asize, get_id(b[i]));
    }
    for (int i = 1; i <= m; i++) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        int ans = ST.query(root[l - 1], root[r], 1, asize, k);
        printf("%d\n", a[ans]);
    }
}
```

# 3 后缀数组

## 定义

1. SA[i] 排名为 i 的后缀在原串组对应的下标
2. Rank[i] 原数组 i 开始的后缀的排名
3. height[i] 表示 sa[i-1] 和 sa[i] 的最长公共前缀长度。
4. h[i] 后缀 Suffix[i] 和它前一名的后缀的最长公共前缀

5. 时间复杂度：$O(nlog_n)$

## 应用

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXN 10000000
char s[MAXN];
int n, sa[MAXN], rank[MAXN], height[MAXN], t1[MAXN], t2[MAXN], c[MAXN];
int a[MAXN];
void SAsort(int n, int m) {
    int *x = t1, *y = t2;
    for (int i = 1; i <= m; i++)
        c[i] = 0;
    for (int i = 1; i <= n; i++)
        c[x[i] = a[i]]++;
    for (int i = 1; i <= m; i++)
        c[i] += c[i - 1];
    for (int i = n; i >= 1; i--)
        sa[c[x[i]]--] = i;
    for (int k = 1, p = 0; k <= n && p <= n; k <<= 1) {
        p = 0;
        for (int i = n - k + 1; i <= n; i++)
            y[++p] = i;
        for (int i = 1; i <= n; i++)
            if (sa[i] > k)
                y[++p] = sa[i] - k;
        for (int i = 1; i <= m; i++)
            c[i] = 0;
        for (int i = 1; i <= n; i++)
            c[x[y[i]]]++;
        for (int i = 1; i <= m; i++)
            c[i] += c[i - 1];
        for (int i = n; i >= 1; i--)
            sa[c[x[y[i]]]--] = y[i];
        std::swap(x, y);
        p = 2;
        x[sa[1]] = 1;
        for (int i = 2; i <= n; i++)
            x[sa[i]] =
                y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k]
                    ? p - 1
                    : p++;
```

```
        m = p;
    }
    for (int i = 1; i <= n; i++)
        rank[sa[i]] = i;
    int f = 0;
    for (int i = 1; i <= n; i++) {
        int j = sa[rank[i - 1]];
        if (f)
            f--;
        while (s[j + f] == s[i + f])
            f++;
        height[rank[i]] = f;
    }
}
int main() {
    scanf("%s", s);
    n = strlen(s);
    for (int i = 1; i <= n; i++)
        a[i] = s[i - 1];
    SAsort(n, 10000);
    for (int i = 1; i <= n; i++)
        printf("%d%c", sa[i], i == n ? '\n' : '□');
    return 0;
}
```