ACM 模板

rogeryoungh

2021 年 5 月 5 日

目录

第一章　上号

1.1　头文件

</> 代码 1.1:　/上号/头文件.hpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
#define _fora(i,a,n) for(ll i=(a);i<=(n);i++)
#define _forz(i,a,n) for(ll i=(a);i>=(n);i--)
#define _forb(i,a)   for(ll i=(a);i>0;i-=i&(-i))
#define _fore(i,a)   for(int i=head[(a)];i;i=edge[i].nxt)
#define _in(i,min,max) ( ((i)-(min)) | ((max)-(i)) )
#define _dbg(...) printf(__VA_ARGS__)
#define LN putchar('\n')

inline ll rr() {
    ll s = 0, w = 1; char c = getchar();
    while(c<'0'||c>'9') { if(c=='-')w=-1; c=getchar(); }
    while(c>='0'&&c<='9') { s=s*10+c-'0'; c=getchar(); }
    return s*w;
}
```

1.2　预编译

头文件引入方式改为如下，可以把头文件放入 `lab.hpp`，然后使用 `clang++ lab.hpp` 预编译。实际编译使用 `clang++ lab.cpp -D RYLOCAL` 添加条件编译参数。

```cpp
#ifdef RYLOCAL
#include "lab.hpp"
#else  //#include <bits/stdc++.h>
#include <iostream>   #include <cstring>    #include <functional>
#include <cmath>      #include <cstdio>     #include <algorithm>
#endif
```

1.3 进制转换

</> 代码 1.2: /上号/进制转换.cpp

```
void pr(ll n, ll x) {
    if(n >= x)  pr(n / x, x);
    putchar(n%x + (x>10 ? 'A'-10 : '0'));
}
void pr(ll n) {
    if(n >= 10)  pr(n / 10);
    putchar(n % 10 + '0');
}
```

1.4 常见技巧

向上取整 p/q 为 (p-1)/q+1 。

预计算 $\log_n$，只需 `_fora(i, n, MN) logn[i] = logn[i/n] + 1;`。

字典序 `strcmp(x,y) < 0`。

1.5 二分查找

STL 二分 在 $[l,r)$ 查找 $\geqslant value$ 中最前的一个，找不到则返回 r 。支持 cmp 函数。

```
ForwardIt lower_bound(ForwardIt l, ForwardIt r, const T& value);
```

在 $[l,r)$ 查找 $> value$ 中最前的一个，找不到则返回 r 。支持 cmp 函数。

```
ForwardIt upper_bound(ForwardIt l, ForwardIt r, const T& value);
```

手写二分，在单增（单减）数组中查找 $\geqslant x (\leqslant x)$ 的数中最前的一个。

</> 代码 1.3: /上号/二分/01.cpp

```
while (l < r) {
    int mid = (l + r) >> 1;
    if (aa[mid] >= x) // <=
        r = mid;
    else
        l = mid + 1;
}
return l;
```

在单增（单减）数组中查找 $\leqslant x (\geqslant x)$ 的数中最后的一个。

</> 代码 1.4: /上号/二分/02.cpp

```
while (l < r) {
    int mid = (l + r + 1) >> 1;
```

```
    if (aa[mid] <= x) //  >=
        l = mid;
    else
        r = mid - 1;
}
return l;
```

对于上凸（∧ 形）函数，可以使用三分法来查找最大值。对于下凸（∨ 形）变号即可

</> 代码 1.5: /上号/三分法.cpp

```
while(r - l > eps) {
    ld mid = (r + l) / 2;
    if(f(mid + eps) > f(mid - eps))
        l = mid;
    else
        r = mid;
}
```

1.6 矩阵乘法

构建一个 p 行 q 列的矩阵。

</> 代码 1.6: /上号/矩阵乘法.cpp

```
struct Mtx {
    ll m[MN][MN], p, q;
    Mtx(ll p, ll q) : p(p), q(q) {
        memset(m, 0, sizeof(m));
    }
    Mtx operator * (Mtx& mtx) {
        Mtx c(p, mtx.q);
        _fora(i, 1, p) { _fora(k, 1, q) {
            ll t = m[i][k];
            _fora(j, 1, mtx.q) {
                c.m[i][j] += t * mtx.m[k][j];
                c.m[i][j] %= MOD;
            }
        } }
        return c;
    }
};
```

矩阵的输入、输出。

```
void read(Mtx& mtx) {
```

```
    _fora(i, 1, mtx.p) _fora(j, 1, mtx.q)
        mtx.m[i][j] = (MOD + rr()) % MOD;
}
void pr(Mtx mtx) {
    _fora(i, 1, mtx.p) {
        printf("%lld",mtx.m[i][1]);
        _fora(j, 2, mtx.q)
            printf(" %lld",mtx.m[i][j]);
        putchar('\n');
    }
}
```

1.7 快速幂

</> 代码 1.7: /上号/快速幂.cpp

```
ll qpow(ll a, ll b, ll p) {
    ll rst = 1 % p;
    for(; b > 0; b >>= 1, a = a * a % p)
        if(b & 1) rst = a * rst % p;
    return rst;
}
```

</> 代码 1.8: /上号/矩阵快速幂.cpp

```
struct QMtx {
    ll m[5][5], p;
    QMtx(ll p) : p(p) {
        memset(m, 0, sizeof(m));
    }
    QMtx operator * (QMtx& mtx) {
        QMtx c(p);
        _fora(i, 1, p) { _fora(k, 1, p) {
            ll t = m[i][k];
            _fora(j,1,p) {
                c.m[i][j] += t * mtx.m[k][j];
                c.m[i][j] %= MOD;
            }
        } }
        return c;
    }
};
QMtx base(ll p) {
```

```
    QMtx rst(p);
    _fora(i, 1, p)
        rst.m[i][i] = 1;
    return rst;
}
QMtx operator ^ (QMtx m, ll n) {
    QMtx rst = base(3);
    for(; n > 0; n >>= 1, m = m * m)
        if(n & 1) rst = m * rst;
    return rst;
}
```

1.8 快速排序

</> 代码 1.9: /上号/快速排序.cpp

```
void quick_sort(ll* nn, ll l, ll r) {
    if(l >= r) return;
    ll i = l, j = r;
    ll x = nn[(l+r)/2];
    while(i <= j) {
        while(nn[j] > x) j--;
        while(nn[i] < x) i++;
        if(i <= j) swap(nn[i++], nn[j--]);
    }
    quick_sort(l, j); quick_sort(i, r);
}
```

1.9 第 k 大数

</> 代码 1.10: /上号/第 k 大数.cpp

```
ll q_sort(ll* nn, ll l, ll r,) {
    ll i=l, j=r, x=nn[(l+r)/2];
    while(i <= j) {
        while(nn[j] > x) j--;
        while(nn[i] < x) i++;
        if(i <= j) swap(nn[i++], nn[j--]);
    } //l <= j <= i <= r
    if(k <= j) return q_sort(l, j);
    else if(k >= i) return q_sort(i, r);
    else return nn[k+1];
```

```
}
```

第二章　数学

2.1　GCD 和 LCM

</> 代码 2.1:　/数学/gcdlcm.cpp

```
ll gcd(ll a, ll b) { return a ? gcd(b%a,a) : b; }
ll lcm(ll a, ll b) { return a / gcd(b%a,a) * b; }
ll gcd(ll a, ll b) { while(b) { ll t=a%b; a=b; b=t; } return a; }
```

2.2　EXGCD

对于方程

$$ax + by = \gcd(a, b)$$

可通过 exgcd 求出一个整数解。

</> 代码 2.2:　/数学/exgcd.cpp

```
void exgcd(ll a, ll b, ll& x, ll& y) {
    if(!b) { y=0; x=1; return; /* gcd = a */ }
    exgcd(b, a%b, y, x);  y -= a/b*x;
}
```

方程 $ax + by = c$ 有解的充要条件是 $\gcd(a, b) \mid c$。

</> 代码 2.3:　/数学/liEu.cpp

```
bool liEu(ll a, ll b, ll c, ll &x, ll &y) {
    exgcd(a, b, x, y);
    if(c % gcd != 0) return false;
    ll k = c / gcd;
    x *= k, y *= k;
    return true;
}
```

2.3　乘法逆元

方程 $ax \equiv 1 \pmod p$ 有解的充要条件是 $\gcd(a, p) = 1$。

容易想到它与方程 $ax + py = c$ 等价，于是可以利用 exgcd 求最小正解。

</> 代码 2.4: /数学/逆元/exgcd 法.cpp

```
ll inv(ll a, ll p) {
    ll x, y;
    exgcd(a, p, x, y);
    return (x % p + p) % p;
}
```

仅当 p 为质数时，由 Fermat 小定理知 $x \equiv a^{p-2} \pmod{p}$。

</> 代码 2.5: /数学/逆元/快速幂法.cpp

```
ll inv(ll a, ll p) {
    return qpow(a, p - 2, p);
}
```

2.4 筛法

Eratosthenes 筛 复杂度 $O(n \log\log n)$。

</> 代码 2.6: /数学/筛法/Eratosthenes.cpp

```
bool notp[100000001];
int prime[20000001], cnt;
void pre_eratosthenes(int n) {
    _fora(i,2,n) { if(!notp[i]) {
        prime[++cnt] = i;
        int tn = n/i;
        _fora(j, i, tn) notp[i*j] = true;
    } }
}
```

Eular 筛 复杂度 $O(n)$，每个合数只会被筛一次。

</> 代码 2.7: /数学/筛法/Eular.cpp

```
bool notp[100000001];
int prime[20000001], cnt;
void pre_eular(int n){
    _fora(i, 2, n) {
        if(!notp[i])  prime[++cnt] = i;
        int t = n / i;
        _fora(j,1,cnt) {
            if(prime[j] > t)  break;
            notp[i * prime[j]] = true;
```

```
            if(i % prime[j] == 0)  break;
        }
    }
}
```

2.5 素性测试

2.5.1 试除法

</> 代码 2.8: /数学/试除法.cpp

```
bool isprime(ll n) {
    if(n < 3) return n == 2;
    if(n & 1 == 0) return false;
    ll sn = (ll) sqrt(n*1.0);
    for(ll i = 3; i <= sn; i += 2)
        if(n % i == 0) return false;
    return true;
}
```

2.5.2 Miller Rabbin

如果 $n \leqslant 2^{32}$，那么 ppp 取 $2,7,61$；如果 ppp 选择 $2,3,7,61,24251$，那么 10^{16} 内只有唯一的例外。如果莫名 WA 了，就多取点素数吧。

</> 代码 2.9: /数学/Miller_Rabbin.cpp

```
bool Miller_Rabbin(ll n) {
    if(n < 3)  return n == 2;
    if(n & 1 == 0)  return false;
    int a = n-1, b=0;
    while(1 - a & 1)  a/=2, ++b;
    int ppp[10] = {2,7,61};
    _fora(i, 0, 2) {
        ll x = ppp[i], j;
        if(n == x) return true;
        ll v = qpow(x,a,n);
        if(v == 1 || v == n-1)  continue;
        for(j = 0; j < b; ++j) {
            v = v*v%n; if(v == n-1) break;
        }
        if(j >= b) return false;
    } return true;
}
```

2.6 Lucas 定理

当 n, m 很大而 p 较小的时候，有

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

</> 代码 2.10: /数学/Lucas.cpp

```
ll Lucas(ll n, ll m, int p){
    return m ? Lucas(n/p, m/p, p) * comb(n%p, m%p, p) % p : 1;
}
```

2.7 约瑟夫 Josephus 问题

对 n 个人进行标号 $0, \cdots, n-1$，顺时针站一圈。从 0 号开始，每一次从当前的人继续顺时针数 k 个，然后让这个人出局，如此反复。

设最后剩下的人的编号为 $J(n, k)$，有递推式

$$J(n+1, k) = (J(n, k) + k) \bmod (n+1)$$

踢出第一个人 k 后，剩下就转化为 $J(n, k)$ 的情景，还原编号只需增加相对位移 k。

</> 代码 2.11: /数学/Josephus.cpp

```
int josephus(int n, int k) {
    int rst = 0;
    _fora(i, 1, n)
        rst = (rst + k) % i;
    return rst;
}
```

2.8 中国剩余定理

若 n_i 中任意两个互质，求方程组的解

$$\begin{cases} x \equiv a_1 & \pmod{n_1} \\ x \equiv a_2 & \pmod{n_2} \\ & \vdots \\ x \equiv a_k & \pmod{n_k} \end{cases}$$

</> 代码 2.12: /数学/china.cpp

```
ll china(ll* aa, ll* nn) {
    ll prod = 1;
    ll rst = 0;
```

```
    _fora(i, 1, n)
        prod *= nn[i];
    _fora(i, 1, n) {
        ll m = prod / nn[i];
        rst += aa[i] * m * inv(m, nn[i]);
        rst %= prod;
    }
    return rst;
}
```

2.9 博弈

下面都是石子游戏，轮流取走物品。方便起见，称场上 n 堆石子 $a_1, \cdots, a_n$ 为局势。先手必输的局势称为奇异局势

2.9.1 Nim 博弈

有 n 堆分别有 a_i 个物品，两人轮流取走任意一堆的任意个物品，不能不取，最后取光者获胜。奇异局势判定

$$a_1 \oplus \cdots \oplus a_n = 0$$

2.9.2 Wythoff 博奕

两堆分别有 a, b 各物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，不可不取，最后取光者获胜。

</> 代码 2.13: /数学/博弈/Wythoff.cpp

```
const ld phi = 1.61803398874989484820458683436563;
int wythoff(ll a, ll b) {
    if(a > b)
        swap(a, b);
    ll t = (ll) (b - a) * phi;
    if(t == a)
        return false;
    return true;
} // 判先手输赢
```

特点：所有自然数都出现在奇异局势中，不重不漏。

第三章 图论

3.1 链式前项星

</> 代码 3.1: /图论/链式前项星.cpp

```cpp
const int MN = 10005; int head[MN];
struct Edge { int too,nxt,len; } edge[MN*2];
void add(int frm, int too, int len) {
    static int cnt = 0;
    edge[++cnt] = { too, head[frm], len };
    head[frm] = cnt;
}
void dfs(int x,int fa) {
    _fore(i,x)  if(edge[i].too != fa)
        dfs(edge[i].too,x);
}
```

3.2 最短路

3.2.1 Dijkstra

权值必须是非负，复杂度 $O(E\log E)$。

</> 代码 3.2: /图论/最短路/Dijkstra.cpp

```cpp
int dis[MN];
struct Dis {
    int dis, pos;
    bool operator < (const Dis& x) const
        { return x.dis<dis; }
};
void dijkstra(int ss) {
    memset(dis, 0x3f, sizeof(dis));
    dis[ss] = 0;
    priority_queue<Dis> pq; pq.push({0,ss});
    while(!pq.empty()) {
        Dis td = pq.top();  pq.pop();
```

```
        int d = td.dis, x = td.pos;
        if(d != dis[x]) continue;
        _fore(i, x) {
            int y = edge[i].too, z = dis[x] + edge[i].len;
            if(dis[y] > z)
                dis[y] = z, pq.push({dis[y],y});
        }
    }
}
```

3.2.2 Bellman-Ford

复杂度 $O(VE)$。

</> 代码 3.3: /图论/最短路/Bellman-Ford.cpp

```
int dis[MN];
void bellman_ford(int ss) {
    memset(dis, 0x3f, sizeof(dis)); dis[ss] = 0;
    _fora(iia, 1, n-1) { int flag = 1;
        _fora(x, 1, n) { _fore(i, x) {
            int y = edge[i].too, z = dis[x] + edge[i].len;
            if(dis[y] > z)
                dis[y] = z, flag = 0;
        } } if(flag) return;
    }
}
```

3.2.3 Floyd

起始条件 $f(i,j) = \text{edge}(i,j)$, $f(i,i) = 0$。

</> 代码 3.4: /图论/最短路/Floyd.cpp

```
void floyd() {
    _fora(k, 1, n) { _fora(i, 1, n) {
        if(i == k || f[i][k] == 0x3f3f3f3f)
            continue;
        _fora(j, 1, n)
            f[i][j] = min(f[i][j], f[i][k]+f[k][j]);
    } }
}
```

3.3 最近公共祖先 LCA

如果数据小，可以不用求 $\log_2$，直接莽 20。

</> 代码 3.5: 图论/LCA.cpp

```cpp
int pa[MN][30], lgb[MN], dep[MN];
void lca_dfs(int u, int fa) {
    _fora(i, 1, n) lgb[i] = lgb[i>>1] + 1;
    lgb[1] = 0; pa[u][0] = fa;
    dep[u] = dep[fa] + 1;
    _fora(i, 1, lgb[dep[u]])
        pa[u][i] = pa[pa[u][i-1]][i-1];
    _fore(i, u)  if(edge[i].too != fa)
        lca_dfs(edge[i].too, u);
}
int lca(int x, int y) {
    if(dep[x] < dep[y])  swap(x,y);
    while(dep[x] > dep[y])
        x = pa[x][lgb[dep[x]-dep[y]]];
    if(x == y) return x;
    _forz(k, lgb[dep[x]]-1, 0)
        if(pa[x][k] != pa[y][k])
            x = pa[x][k], y = pa[y][k];
    return pa[x][0];
}
```

第四章　动态规划

4.1 背包

4.1.1 01 背包

给定体积为 v_i，价值 w_i 的 N 个物品，背包容积为 M，每个物品只能取 1 个，求最大价值。

</> 代码 4.1: /动态规划/01 背包.cpp

```
_fora(i, 1, n)  _forz(j, m, v[i])
    dp[j] = max(dp[j], dp[j-v[i]]+w[i]);
_fora(j, 1, m)  ans = max(ans, dp[j]);
```

4.1.2 完全背包

给定体积为 v_i，价值 w_i 的 n 个物品，背包容积为 v，每个物品任意取，求最大价值。

</> 代码 4.2: /动态规划/完全背包.cpp

```
_fora(i, 1, n)  _fora(j, v[i], m)
    dp[j] = max(dp[j], dp[j-v[i]]+w[i]);
_fora(j, 1, m)  ans = max(ans, dp[j]);
```

4.1.3 多重背包

给定体积为 v_i，价值 w_i 的 N 个物品，背包容积为 M，每个物品有 c_i 个，求最大价值。

如各种背包组合（如洛谷 P1833 樱花），通常把完全背包转为 99999 个（适当调节）多重背包，再按 01 背包来。

</> 代码 4.3: /动态规划/多重背包.cpp

```
int tm=1,vv[ ],ww[ ];
_fora(i, 1, n) {
    int tc = c[i];
    for(int b=1;b<p;b<<=1,tc-=b,++tm)
        vv[tm] = v[i] * b, ww[tm] = w[i] * b;
    vv[tm] = v[i] * tc, ww[tm] = w[i] * tc;
    ++tm;
}
```

```
_fora(i, 1, n)  _forz(j, m, v[i])
    dp[j] = max(dp[j], dp[j-v[i]]+w[i]);
_fora(j, 1, m)  ans = max(ans, dp[j]);
```

4.2 最长公共上升序列

给出 $1, 2, \ldots, n$ 的两个排列 a 和 b，求它们的最长公共子序列。

</> 代码 4.4: /动态规划/最长公共上升序列.cpp

```
int f[MN], ma[MN], b[MN], n, len=0;
memset(f,0x3f,sizeof(f)); f[0]=0;
_fora(i, 1, n) { ma[rr()] = i; } _fora(i, 1, n) { b[i] = rr(); }
_fora(i, 1, n) {
    int l = 0,r = len;
    if(ma[b[i]] > f[len])
        f[++len] = ma[b[i]];
    else { while(l < r) {
        int mid = (l + r) / 2;
        if(f[mid] > ma[b[i]])  r = mid;
        else l = mid+1;
    } }
    f[l] = min(ma[b[i]], f[l]);
}
```

4.3 数字计数

试计算在区间 1 到 n 的所有整数中，数码 $x(0 \leqslant x \leqslant 9)$ 共出现了多少次?

</> 代码 4.5: /动态规划/数字计数/01.cpp

```
int addup(int n, int x) {
    int ans = 0, m = 1;
    while(m <= n){
        int a = n/(m*10), b = n/m%10, c = n%m;
        ans += a * m;
        if(x > 0){
            if(b > x) ans += m;
            else if(b == x) ans += c + 1;
        } else if(b == 0) {
            ans += c + 1 - m;
        }
        m *= 10;
    }
```

```
    return ans;
}
```

试计算在区间 1 到 n 的所有整数中，出现数码 $x(0 \leqslant x \leqslant 9)$ 的数字有多少?

</> 代码 4.6: /动态规划/数字计数/02.cpp

```
// 预计算
ll dp[20], m = 1;
_fora(i,1,9) {
    dp[i] = dp[i-1] * 9 + m;
    m *= 10;
}
ll addup(ll n, ll x) {
    ll m = 1, i = 0;
    while(m <= n)
        i++, m *= 10;
    ll ans = 0;
    while(m) {
        ll t = n / m;
        n = n % m;
        ans += t * dp[i];
        if(t == x) {
            ans += n + 1;
            break;
        } else if(t > x) {
            ans += - dp[i] + m;
        }
        i--; m /= 10;
    }
    return ans;
}
```

第五章　数据结构

5.1　链表

</> 代码 5.1:　/数据结构/链表.cpp

```cpp
struct Node { int val; Node *prev, *next; };
struct List {
    Node *head, *tail; int len;
    List() {
        head = new Node(); tail = new Node();
        head->next = tail; tail->prev - head;
        len = 0;
    } // 在节点后 p 后插入值 v
    void insert(Node *p,int v) {
        Node *q = new Node(); q->val = v;
        p->next->prev = q; q->next = p->next;
        p->next = q; q->prev = p;
        len++;
    } // 删除节点 p
    void erase(Node *p) {
        p->prev->next = p->next;
        p->next->prev = p->prev;
        delete p; len--;
    } // 清空链表
    ~List() {
        while(head != tail) {
            head = head->next;
            delete head->prev;
        } delete tail; len = 0;
    }
};
```

链表的遍历。

```cpp
for(Node *p=l->head->next; p!=l->tail; p=p->next)
    //正序遍历
for(Node *p=l->tail->prev; p!=l->head; p=p->prev)
```

```
    //逆序遍历
```

5.2 树状数组

树状数组可以维护数组 a 实现（1）将某个数加上 x。（2）求前缀和。

</> 代码 5.2: /数据结构/树状数组/01.cpp

```cpp
ll aa[MN], cc[MN], n;
void build() {
    _fora(i,1,n) {
        cc[i] += aa[i];
        ll j = i + (i&(-i));
        if(j <= n)
            cc[j] += cc[i];
    }
}
ll ask(ll *cc, ll x) {
    ll sum = 0;
    while(x >= 1) {
        sum += cc[x];
        x -= x&(-x);
    }
    return sum;
}
void add(ll *cc, ll x, ll k) {
    while(x <= n) {
        cc[x] += k;
        x += x&(-x);
    }
}
```

区间加 & 单点查询 维护数组 a 的额外差分数组 b，那么 a 的区间加就被转化为 b 的单点增加，且 a 单点查询就被转化为 b 的区间查询。

</> 代码 5.3: /数据结构/树状数组/02.cpp

```cpp
void badd(ll l, ll r, ll k) {
    add(bb, l, k);
    add(bb, r+1, -k);
}
ll bask(ll x) {
    return ask(bb, x) + aa[x];
}
```

区间加 & 区间求和 维护数组 a 的额外差分数组 b，当我们对 a 的前缀 r 求和时有

$$\sum_{i=1}^{r}\sum_{j=1}^{i} b_j = \sum_{i=1}^{r} b_i(r-i+1) = (r+1)\sum_{i=1}^{r} b_i - \sum_{i=1}^{r} b_i i$$

因此还需要两个树状数组来维护 $\sum b_i$ 和 $\sum b_i i$。查询前缀和 cask。

</> 代码 5.4: /数据结构/树状数组/03.cpp

```
ll bb1[MN], bb2[MN];
void cadd(ll l, ll r, ll k) {
    add(bb1, l, k);
    add(bb1, r+1, -k);
    add(bb2, l, l*k);
    add(bb2, r+1, -(r+1)*k);
}
ll cask(ll x) {
    return (x+1) * ask(bb1, x) + ask(cc,x) - ask(bb2,x);
}
```

5.3 ST 表

需要预处理 $\log_2$。令 $st(i,j)$ 表示区间 $[i, i+2^j-1]$ 的最大值，显然 $ST(i,0)=a_i$。状态转移方程

$$ST(i,j+1) = \max(f(i,j), f(i+2^j,j))$$

</> 代码 5.5: /数据结构/ST 表/01.cpp

```
_fora(j,0,lg2n-1) {
    ll tj = 1 << j;
    ll ti = n - (1<<(j+1)) + 1;
    _fora(i,1,ti)
        ST[i][j+1] = max(ST[i][j], ST[i+tj][j]);
}
```

对于 RMQ 问题，记 $s=\lfloor\log_2(r-l+1)\rfloor$，我们总是可以用两个区间 $[l, l+2^s-1]$ 和 $[r-2^s+1, r]$ 来覆盖所查询区间。

</> 代码 5.6: /数据结构/ST 表/02.cpp

```
ll s = lg2[y-x+1];
return max(ST[x][s], ST[y-(1<<s)+1][s]);
```

第六章　字符串

6.1　KMP

前缀函数　对于长为 n 的字符串 s，定义每个位置的前缀函数 $\pi(i)$，值为右端在 i 的相等真后缀与真前缀中最长的长度。

设最长的长度为 $j_1 = \pi(i)$，如何找到其次长 j_2?

注意到后缀 j_1 位与前缀 j_1 位完全相同，故 j_2 为前缀 j_1 中相等真前缀与真后缀中最长的，即

$$j_{n+1} = \pi(j_n - 1)$$

</> 代码 6.1:　/字符串/KMP/01.cpp

```
ll pi[MN];
void pre_kmp(char* s, ll lens) {
    _fora(i,1,lens - 1) {
        ll j = pi[i-1];
        while(j && s[i] != s[j])
            j = pi[j-1];
        pi[i] = j + (s[i] == s[j]);
    }
}
```

Knuth - Morris - Pratt　给定一个文本 t 和一个字符串 s （模式串），尝试找到 s 在 t 中所有出现。

构造字符串 $s + * + t$，其中 $*$ 为不出现在两个字符串中的特殊字符，此时字符串 t 的前缀恰为 s，$\pi(i)$ 的意义为 s 在此处的出现长度。

当 $\pi(i) = |s|$ 时，s 在此处完全出现。

当字符串已经合并时，直接计算 $\pi(i)$ 函数即可，字符串出现位置是 $i - 2|s|$。

</> 代码 6.2:　/字符串/KMP/02.cpp

```
void kmp(char* s, ll lens, char* t, ll lent) {
    pre_kmp(s, lens);
    ll p = 0;
    _fora(i, 0, lent-1) {
        ll j = p;
        while(j && t[i] != s[j])
            j = pi[j-1];
```

```
        p = j + (t[i] == s[j]);
        if(p == lens)
            // 出现起始 i-lens+2
    }
}
```