该库用来记录在 acwing 的代码模板

####

快速排序

```
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int q[N];

void quick_sort(int q[], int l, int r) {
    if (l >= r) return ;

    int x = q[l + r >> 1], i = l - 1, j = r + 1; //如果超时的话建议修改一下中值 x

    while (i < j) {
        do i ++; while (q[i] < x);
        do j --; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    quick_sort(q, l, j);
    quick_sort(q, j + 1, r);
}

int main() {
    int n;

    scanf("%d", &n);

    for (int i = 0; i < n; i ++) scanf("%d", &q[i]);

    quick_sort(q, 0, n - 1);

    for (int i = 0; i < n; i ++) printf("%d ", q[i]);

    return 0;
}
```

归并排序

```
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int n;
```

```
int q[N], temp[N];

void merge_sort(int q[], int l, int r)
{
    //递归的终止情况
    if(l >= r) return;

    //第一步：分成子问题
    int mid = l + r >> 1;

    //第二步：递归处理子问题
    merge_sort(q, l, mid ), merge_sort(q, mid + 1, r);

    //第三步：合并子问题
    int k = 0, i = l, j = mid + 1, tmp[r - l + 1];
    while(i <= mid && j <= r) {
        if(q[i] <= q[j]) tmp[k++] = q[i++];
        else tmp[k++] = q[j++];
    }
    while(i <= mid) tmp[k++] = q[i++];
    while(j <= r) tmp[k++] = q[j++];

    for(k = 0, i = l; i <= r; k++, i++) q[i] = tmp[k];
}

int main() {
    scanf("%d", &n);

    for (int i = 0; i < n; i ++)
        scanf("%d", &q[i]);

    merge_sort(q, 0, n - 1);

    for (int i = 0; i < n; i ++)
        printf("%d ", q[i]);

    return 0;
}
```

第二章数据结构(一)

单链表

```
#include <iostream>

using namespace std;

const int N = 1000010;

// head 表示头节点 e[i] 表示节点 i 的值 ne[i] 表示节点 i 的 next 指针是多少
// idx 存储当前已经用到哪个节点

int head, e[N], ne[N], idx;

// 初始化
```

```
init () {
    head = -1;
    idx = 0;
}

// 将x插到头节点
void add_to_head (int x) {
    e[idx] = x, ne[idx] = head, head = idx, idx ++;
}

// 将 x 插到下标是 k 的点后面
void add (int k, int x) {
    e[idx] = x;
    ne[idx] = ne[k];
    ne[k] = idx;
    indx ++;
}

// 将下标是 k 的点后面的点删掉
remove (int k) {
    ne[k] = ne[ne[k]]; // 删除的时候并没有关联到 idx
}

int main () {
    int m;
    cin >> m;

    init();

    while (m --) {
        int k, x;
        char op;

        cin >> op;

        if (op == 'H') {
            cin >> x;
            add_to_head(x);
        } else if (op == 'D') {
            cin >> k;
            if (!k) head = ne[head]; // 对头节点的一个特判
            remove(k - 1);
        } else {
            cin >> k >> x;
            add(k - 1, x);
        }
    }

    for (int i = head; i != -1; i = ne[i]) cout << e[i] << " ";

    cout << endl;

    return 0;
}
```

双链表

```
#include <iostream>

using namespace std;

const int N = 100010;

int m;
int e[N], l[N], r[N], idx;

// 初始化
void init () {
    // 0表示左端点 1表示右端点
    r[0] = 1, l[0] = 0;
    idx = 2;
}

// 在下标是 k 的点的右边插入 x (如果是在左边插入 其实也可以直接调用这个 但参数需要换一下)
void add (int k, int x) { //顺序别写反了
    e[idx] = x;
    r[idx] = r[k];
    l[idx] = k;
    l[r[k]] = idx;
    r[k] = idx;
}

// 删除操作
void remove (int k, int x) {
    r[l[k]] = r[k];
    l[r[k]] = l[k];

}
```

模拟栈

```
#include <iostream>

using namespace std;

const int N = 100010;

int stk[N], tt; // tt 表示栈顶元素

// 插入
stk[++ tt] = x;

// 弹出
tt --;

// 判断栈是否为空
if (tt > 0) not empty
esle empty

// 栈顶
```

```
stk[tt];
```

模拟队列

-普通队列

```
#include <iostream>

using namespace std;

const int N = 100010;

int q[N], hh, tt = -1; // hh 表示的是队头 tt 表示的是队尾 (注意这里栈初始的是 -1 而栈初始的是0)

// 插入一个元素
q[ ++ tt] = x;

// 弹出一个元素
hh ++;

// 判断是否为空
if (hh <= tt) not empty
else empty

// 取出队头元素
q[hh]
```

-循环队列

```
// hh 表示队头，tt表示队尾的后一个位置
int q[N], hh = 0, tt = 0;

// 向队尾插入一个数
q[tt ++ ] = x;
if (tt == N) tt = 0;

// 从队头弹出一个数
hh ++ ;
if (hh == N) hh = 0;

// 队头的值
q[hh];

// 判断队列是否为空
if (hh != tt) {

}
```

单调栈

```
#include <iostream>

using namespace std;

const int N = 100010;

int n;
int stk[N], tt;

int main () {
    cin >> n;

    for (int i = 0; i < n; i ++) {
        int x;
        cin >> x;

        while (tt && stk[tt] >= x) tt --;

        if (tt) cout << stk[tt] << " ";
        else cout << -1 << " ";

        stk[ ++ tt] = x;
    }
}
```

单调队列

```
#include <iostream>

using namespace std;

const int N = 1000010;

int n;
int a[N], q[N];

int main () {
    scanf ("%d%d", &n, &k);

    for (int i = 0; i < n; i ++) scanf("%d", &a[i]);


    int hh = 0, tt = -1;

    for (int i = 0; i < n; i ++) {
        // 判断队头是否滑出窗口 q[hh] 里存的是数组下标
        if (hh <= tt && i - k + 1 > q[hh]) hh ++;

        while (hh < = tt && a[q[tt]] >= a[i]) tt --;

        q[ ++ tt] = i;

        if (i >= k - 1) print("%d", a[q[hh]]);
```

```
    }

    puts(" ");

    /*
    如果是求窗口里的最大值

    int hh = 0, tt = -1;

    for (int i = 0; i < n; i ++) {
        // 判断队头是否滑出窗口 q[hh] 里存的是数组下标
        if (hh <= tt && i - k + 1 > q[hh]) hh ++;

        while (hh < = tt && a[q[tt]] <= a[i]) tt --; // 就把这里的符号改一下

        q[ ++ tt] = i;

        if (i >= k - 1) print("%d", a[q[hh]]);
    }

    puts(" ");
    */

    return 0;
}
```

KMP字符串

```
#include <iostream>

using namespace std;

const int N = 10010, M = 1000010;

int n, m;

char P[N], s[M];
int ne[N]; // next 数组

int main () {
    cin >> n >> p + 1 >> m >> s + 1; // 下标从 1 开始

    // 求 next 过程
    for (int i = 2, j = 0; i <= n; i ++) {
        while (j && p[i] != p[j + 1]) j = ne[j];

        if (p[i] == p[j + 1]) j ++;

        ne[i] = j;
    }

    // kmp 匹配过程
    for (int i = 1, j = 0; i <= m; i ++) {

        while (j && s[i] != p[j + 1]) j = ne[j];
```

```
        if (s[i] == p[j + 1]) j ++;

        if (j == n) { // 匹配成功
            printf("%d", i - n);

            j = ne[j]; //
        }
    }
}
```