算法学习：AcWing

Algorithm Learning: AcWing

吴小强 编著

深圳 • SHENZHEN

前言

目　录

PART I

第一部分

算法基础

第一章

算法基础

1.1 快速排序

1.1.1 AcWing 785. 快速排序

AcWing 785. 快速排序

给定你一个长度为 n 的整数数列。请你使用快速排序对这个数列按照从小到大进行排序。并将排好序的数列按顺序输出。

输入格式：

输入共两行，第一行包含整数 n。第二行包含 n 个整数（所有整数均在 $1 \sim 10^9$ 范围内），表示整个数列。

输出格式：

输出共一行，包含 n 个整数，表示排好序的数列。

数据范围：$1 \leq n \leq 100000$

输入样例：	输出样例：
5	1 2 3 4 5
3 1 2 4 5	

快速排序算法基于**分治**算法，以一个数来作为分治的节点。

随机选取数组中的某个元素 x 作为分界点，操作数组中的元素使得数组被分割为两个部分，左边一侧的元素小于等于 x，右边一侧则大于等于 x。接下来递归的对左右两侧数组进行操作，直到最小数组只有一个元素则完成排序。

主要步骤如下：

1. 确定分界点 x，可取值：q[l], q[r], q[(l + r) >> 1], random value
2. 调整数组，使得左边小于等于 x，右边大于等于 x
3. 递归处理左右两段

quick sort

```
#include <stdio.h>
#include <stdlib.h>

void swap(int *q, int a, int b) {
    int tmp = q[a];
    q[a] = q[b];
    q[b] = tmp;
}

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

int main() {
    int n;
    scanf("%d", &n);
    int *q = (int *)calloc(sizeof(int), n);
    if (q == NULL) {
        return -1;
    }
    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]);
    }
    free(q);
    return 0;
}
```

从上述代码段中可以清晰看到递归处理的过程，每次选取分界点，之后将左右两侧的元素进行调整，此处采用双指针算法。

这里有两个问题：

1. 在选择 x 时选择q[l]则在递归是不能选用i，会出现边界问题 | i − 1, i
2. 在选择 x 时选择q[r]则在递归是不能选用j，会出现边界问题 | j, j + 1

边界用例可使用1, 2这个例子，会有递归不结束的问题

该算法**不稳定**，因为q[i]和q[j]相等的时候会发生交换。
这里调整数组的部分是难点，怎么优雅的调整数组？暴力做法可以开辟两个辅助数组来存储。
双指针做法优雅简洁。

时间复杂度分析：

1.1.2 AcWing 786. 第 k 个数

快速选择算法可选出有序数组中的第 k 个数，与快排中逻辑相同，左侧的元素都小于 x 右侧元素都大于 x。如果左侧元素的数量大于等于 k 则表示第 k 个元素在左侧数组中，反之则在右侧数组中寻找 $k-$left length 的元素。

AcWing 786. 第 k 个数

给定一个长度为 n 的整数数列，以及一个整数 k，请用快速选择算法求出数列从小到大排序后的第 k 个数。

输入格式：

第一行包含两个整数 n 和 k。第二行包含 n 个整数（所有整数均在 $1 \sim 10^9$ 范围内），表示整数数列。

输出格式：

输出一个整数，表示数列的第 k 小数。

数据范围：

$1 \leq n \leq 100000, 1 \leq k \leq n$

输入样例：

5 3

2 4 1 5 3

输出样例：

3

find kth smallest number

```c
#include <stdio.h>
#include <stdlib.h>

int q_select(int *q, int l,
             int r, int k) {
    if (r <= l) {
        return q[l];
    }

    int x = q[(l + r) >> 1];
    int i = l - 1;
    int j = r + 1;

    while (i < j) {
        do i++; while(q[i] < x);
        do j--; while(q[j] > x);

        if (i < j) {
            int tmp = q[i];
            q[i] = q[j];
            q[j] = tmp;
        }
    }

    int length = j - l + 1;
    if (length < k) {
        return q_select(q, j + 1, r,
            k - length);
    } else {
        return q_select(q, l, j, k);
    }
}

int main()
{
    int n;
    int k;
    scanf("%d %d", &n, &k);
    int *q = (int *)calloc(sizeof(int), n);
    if (q == NULL) {
        return -1;
    }
    for (int i = 0; i < n; i++) {
        scanf("%d", q + i);
    }
    int ret = q_select(q, 0, n - 1, k);
    printf("%d", ret);
    return 0;
}
```

1.2 归并排序

归并排序同样是基于**分治**算法，不过是以整个数组的中间位置来分。

将数组分割成两个已经分别排序好的有序数组，再将其二者合并即可。此方法需要有单独的空间来存放合并的临时结果，再将临时结果写入到原始区域中。

主要步骤如下：

1. 确定分界点,mid = (l + r) >> 1

2. 递归排序左右两边

3. 归并,将两个有序的子数组合二为一

1.2.1 AcWing 787. 归并排序

AcWing 787. 归并排序

给定你一个长度为 n 的整数数列。请你使用归并排序对这个数列按照从小到大进行排序。并将排好序的数列按顺序输出。

输入格式:

输入共两行,第一行包含整数 n。第二行包含 n 个整数(所有整数均在 $1 \sim 10^9$ 范围内),表示整个数列。

输出格式:

输出共一行,包含 n 个整数,表示排好序的数列。

数据范围:$1 \leq n \leq 100000$

输入样例:

5

3 1 2 4 5

输出样例:

1 2 3 4 5

merge sort

```c
#include <stdio.h>
#include <stdlib.h>

#define N 100010

int backup[N];

void merge_sort(int *q, int l, int r)
{
    if (r <= l) {
        return;
    }
    int mid = (l + r) >> 1;
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);
    int k = 0;
    int i = l;
    int j = mid + 1;
    while (i <= mid && j <= r) {
        if (q[i] <= q[j]) {
            backup[k++] = q[i++];
        }
        if (q[j] < q[i]) {
            backup[k++] = q[j++];
        }
    }
    while (i <= mid) {
        backup[k++] = q[i++];
    }
    while (j <= r) {
        backup[k++] = q[j++];
    }

    for (i = l, j = 0; j < k; i++, j++) {
        q[i] = backup[j];
    }
}

int main()
{
    int n;
    scanf("%d", &n);
    int *q = (int *)calloc(sizeof(int), n);
    if (q == NULL) {
        return -1;
    }
    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;
}
```

双指针算法做归并

这里归并两个子数组之后要写回去，backup数组只是临时存储使用。

1.2.2 AcWing 788. 逆序对的数量

AcWing 788. 逆序对的数量

给定一个长度为 n 的整数数列，请你计算数列中的逆序对的数量。逆序对的定义如下：对于数列的第 i 个和第 j 个元素，如果满足 $i < j$ 且 $a[i] > a[j]$，则其为一个逆序对；否则不是。

输入格式：

第一行包含整数 n，表示数列的长度。第二行包含 n 个整数，表示整个数列。

输出格式：

输出一个整数，表示逆序对的个数。

数据范围

$1 \leq n \leq 100000$，数列中的元素的取值范围 $[1, 10^9]$。

输入样例：

2 3 4 5 6 1

输出样例：

5

分治思路，将整个区间一分为二。考虑到归并排序的时候需要将两个有序数组合并，此时恰好可以做逆序对的统计。假设有一种算法，可以将数组排序的过程中统计该数组中的逆序对数量，则问题变为怎么统计两个有序数组中合起来的逆序对。

归并排序计算逆序对数量

```c
#include <stdio.h>
#include <stdlib.h>

long long merge_sort(int *q, int *tmp,
                     int l, int r) {
    if (l >= r) {
        return 0;
    }
    int mid = (l + r) >> 1;
    // 左侧的数组已统计逆序对且已经排序，右侧同样
    long long res = merge_sort(q, tmp, l, mid) + merge_sort(q, tmp, mid + 1, r);
    // 统计两个有序数组合起来的逆序对数量
    int i = l;
    int j = mid + 1;
    int k = 0;
    while (i <= mid && j <= r) {
        if (q[i] > q[j]) {
            res += mid - i + 1;
            tmp[k++] = q[j++];
        } else {
            tmp[k++] = q[i++];
        }
    }

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

    return res;
}

int main()
{
    int n;
    scanf("%d", &n);
    int *q = (int *)calloc(n, sizeof(int));
    if (q == NULL) {
        return -1;
    }
    int *tmp = (int *)calloc(n, sizeof(int));
    if (tmp == NULL) {
        free(q);
        return -1;
    }
    for(int i = 0; i < n; i++) {
        scanf("%d", &q[i]);
    }

    printf("%ld", merge_sort(q, tmp, 0, n - 1));

    return 0;
}
```

1.3 二分

整数二分和浮点数二分，二分即查找一个边界值，在左侧满足某种性质，右侧不满足。

二分用模版如下：

二分模版

```c
// 区间[l, r]被划分为[l, mid] 和
// [mid + 1, r]时使用，往左找
int bsearch_1(int l, int r)
{
    while (l < r) {
        mid = (l + r) / 2;
        if (Check(mid)) {
            r = mid;
        } else {
            l = mid + 1;
        }
    }
}
```

```c
// 区间[l, r]被划分为[l, mid - 1] 和
// [mid, r]时使用，往右找
int bsearch_2(int l, int r)
{
    while (l < r) {
        mid = (l + r + 1) / 2;
        if (Check(mid)) {
            l = mid;
        } else {
            r = mid - 1;
        }
    }
}
```

每次要保证答案在区间中。

第二个模版加一的原因在于，如果某次循环结束后，l = r − 1，如果不加 1，此时因为向下取整的缘故mid = l，check 成功后l被再次赋值为mid即l，则此时进入死循环。

1.3.1 AcWing 789. 数的范围

AcWing 789. 数的范围

给定一个按照升序排列的长度为 n 的整数数组，以及 q 个查询。对于每个查询，返回一个元素 k 的起始位置和终止位置（位置从 0 开始计数）。如果数组中不存在该元素，则返回−1 −1。

输入格式：

第一行包含整数 n 和 q，表示数组长度和询问个数。第二行包含 n 个整数（均在 $1 \sim 10000$ 范围内），表示完整数组。接下来 q 行，每行包含一个整数 k，表示一个询问元素。

输出格式：

共 q 行，每行包含两个整数，表示所求元素的起始位置和终止位置。

数据范围 $1 \leq n \leq 100000$，$1 \leq q \leq 10000$，$1 \leq k \leq 10000$

输入样例：

6 3

1 2 2 3 3 4

3

4

5

输出样例：

3 4

5 5

-1 -1

binary search

```c
#include <stdio.h>
#include <stdlib.h>

int b_search_l(int *q, int l,
               int r, int t) {
    while (l < r) {
        int mid = (l + r) >> 1;
        if (q[mid] >= t) {
            r = mid;
        } else {
            l = mid + 1;
        }
    }
    if (q[l] != t) {
        return -1;
    }
    return l;
}

int b_search_r(int *q, int l,
               int r, int t) {
    while (l < r) {
        int mid = (l + r) / 2 + 1;
        if (q[mid] <= t) {
            l = mid;
        } else {
            r = mid - 1;
        }
    }
    if (q[l] != t) {
        return -1;
    }
    return l;
}

int main()
{
    int n;
    int q;
    scanf("%d %d", &n, &q);
    int *q = (int *)calloc(sizeof(int), n);
    if (q == NULL) {
        return -1;
    }
    for (int i = 0; i < n; i++) {
        scanf("%d", q + i);
    }
    while(q--) {
        int t;
        scanf("%d", &t);
        printf("%d %d\n",
        b_search_l(q, 0, n - 1, t),
        b_search_r(q, 0, n - 1, t));
    }
    return 0;
}
```

这里不能使用bsearch函数来完成左端点的搜索，因为该函数在面对重复值时返回值不确定，是未定义行为。

1.3.2 AcWing 790. 数的三次方根

AcWing 790. 数的三次方根

给定一个浮点数 n，求它的三次方根。

输入格式：

共一行，包含一个浮点数 n。

输出格式：

共一行，包含一个浮点数，表示问题的解。

注意，结果保留 6 位小数。

数据范围：

$-10000 \leq n \leq 10000$

输入样例：

1000.00

输出样例：

10.000000

数的三次方根

```
#include <stdio.h>

#define N 10000

int main()
{
    double n;
    scanf("%lf", &n);
    double l = 0 - N;
    double r = N;

    while (r - l > 1e-8) {
        double mid = (l + r) / 2;
        if (mid * mid * mid < n) {
            l = mid;
        } else {
            r = mid;
        }
    }
    printf("%.6f", l);
    return 0;
}
```

1.4 高精度

1.5 前缀和与差分

1. 前缀和可方便地求取数组中某个区间的元素和，或者矩阵的子矩阵的和 O(1)

2. 差分和方便的将数组某个区间的所有元素加上一个数，O(1) 时间复杂度

1.5.1 AcWing 795. 前缀和

AcWing 795. 前缀和

输入一个长度为 n 的整数序列。接下来再输入 m 个询问，每个询问输入一对 l, r。对于每个询问，输出原序列中从第 l 个数到第 r 个数的和。

输入格式：

第一行包含两个整数 n 和 m。第二行包含 n 个整数，表示整数数列。接下来 m 行，每行包含两个整数 l 和 r，表示一个询问的区间范围。

输出格式：

共 m 行，每行输出一个询问的结果。

数据范围：

$1 \leq l \leq r \leq n$, $1 \leq n, m \leq 100000$, $-1000 \leq$ 数列中元素的值 ≤ 1000

输入样例：

```
5 3
2 1 3 6 4
1 2
1 3
2 4
```

输出样例：

```
3
6
10
```

前缀和

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
    int m;
    scanf("%d %d", &n, &m);
    int *q = (int *)calloc(n + 1, sizeof(int));
    int *preSum = (int *)calloc(n + 1, sizeof(int));
    for (int i = 1; i <= n; i++) {
        scanf("%d", q + i);
        preSum[i] = preSum[i - 1] + q[i];
    }
    while (m--) {
        int l;
        int r;
        scanf("%d %d", &l, &r);
        printf("%d\n", preSum[r] - preSum[l - 1]);
    }
    return 0;
}
```

1.5.2 AcWing 796. 子矩阵的和

子矩阵

AcWing 796. 子矩阵的和

子矩阵的和 输入一个 n 行 m 列的整数矩阵，再输入 q 个询问，每个询问包含四个整数 x_1, y_1, x_2, y_2，表示一个子矩阵的左上角坐标和右下角坐标。对于每个询问输出子矩阵中所有数的和。

输入格式：

第一行包含三个整数 n, m, q。接下来 n 行，每行包含 m 个整数，表示整数矩阵。接下来 q 行，每行包含四个整数 x_1, y_1, x_2, y_2，表示一组询问。

输出格式：

共 q 行，每行输出一个询问的结果。

数据范围：

$1 \le n, m \le 1000$, $1 \le q \le 200000$, $1 \le x_1 \le x_2 \le n$, $1 \le y_1 \le y_2 \le m$, $-1000 \le \ \le 1000$

输入样例：	输出样例：
3 4 3	17
1 7 2 4	27
3 6 2 8	21
2 1 2 3	
1 1 2 2	
2 1 3 4	
1 3 3 4	

子矩阵的和

```
#include<stdio.h>

#define N 1010

int mat[N][N];
int preMat[N][N];

int main()
{
    int n, m, q;
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            scanf("%d", &mat[i][j]);
            preMat[i][j] = preMat[i - 1][j] + preMat[i][j - 1] - preMat[i - 1][j - 1] + mat[i
    ][j];
        }
    }
    while (q--) {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        printf("%d\n", preMat[x2][y2] - preMat[x2][y1 - 1] - preMat[x1 - 1][y2] + preMat[x1 -
     1][y1 - 1]);
    }
    return 0;
}
```

1.6 双指针算法

1.7 位运算

1.8 离散化

1.9 区间合并

第二章

数据结构

2.1 单链表

2.1.1 AcWing 826. 单链表

linked list

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define N 100010
int idx; // 标识节点
int head; // 头节点指向的元素
int e[N]; // element array
int ne[N]; // next array

void Init()
{
    idx = -1;
    head = -1;
}

// 在头节点后插入元素
void AddHead(int x) {
    e[++idx] = x;
    ne[idx] = head;
    head = idx;
}

// 在k节点之后插入元素
void Add(int k, int x)
{
    e[++idx] = x;
    ne[idx] = ne[k];
    ne[k] = idx;
}

// 删除k之后的那个元素
void Remove(int k) {
    ne[k] = ne[ne[k]];
}

int main()
{
    Init();
    int n;
    scanf("%d", &n);
    while (n--) {
        int x;
        int k;
        char op;
        scanf(" %c", &op);
        if (op == 'H') {
            scanf("%d", &x);
            AddHead(x);
        }
        if (op == 'I') {
            scanf("%d %d", &k, &x);
            Add(k - 1, x);
        }
        if (op == 'D') {
            scanf("%d", &k);
            if (k == 0) {
                head = ne[head];
            }
            Remove(k - 1);
        }
    }

    int tmp = head;
    while (tmp != -1) {
        printf("%d ", e[tmp]);
        tmp = ne[tmp];
    }
    return 0;
}
```

2.2 双链表

2.2.1 AcWing 827. 双链表

2.3 栈

2.3.1 AcWing 828. 模拟栈

2.3.2 AcWing 3302. 表达式求值

2.4 队列

2.4.1 AcWing 829. 模拟队列

2.5 单调栈

2.5.1 AcWing 830. 单调栈

2.6 单调队列

2.6.1 AcWing 154. 滑动窗口

2.7 KMP

2.7.1 AcWing 831. KMP 字符串

2.8 Trie

2.8.1 AcWing 835. Trie 字符串统计

2.8.2 AcWing 143. 最大异或对

2.9 并查集

2.9.1 AcWing 836. 合并集合

2.9.2 AcWing 837. 连通块中点的数量

2.9.3 AcWing 240. 食物链

2.10 堆

2.10.1 AcWing 838. 堆排序

2.10.2 AcWing 839. 模拟堆

2.11 哈希表

2.11.1 AcWing 840. 模拟散列表

2.11.2 AcWing 841. 字符串哈希

第三章

搜索和图论

3.1 DFS

3.1.1 AcWing 842. 排列数字

3.1.2 AcWing 843. n-皇后问题

3.2 BFS

3.2.1 AcWing 844. 走迷宫

3.2.2 AcWing 845. 八数码

3.3 树与图的深度优先遍历

3.3.1 AcWing 846. 树的重心

3.4 树与图的广度优先遍历

3.4.1 AcWing 847. 图中点的层次

3.5 拓扑排序

3.5.1 AcWing 848. 有向图的拓扑序列

3.6 Dijkstra

3.6.1 AcWing 849. Dijkstra 求最短路 I

3.6.2 AcWing 850. Dijkstra 求最短路 II

3.7 bellman-ford

3.7.1 AcWing 853. 有边数限制的最短路

3.8 spfa

3.8.1 AcWing 851. spfa 求最短路

3.8.2 AcWing 852. spfa 判断负环

3.9 Floyd

3.9.1 AcWing 854. Floyd 求最短路

3.10 Prim

3.10.1 AcWing 858. Prim 算法求最小生成树

第四章

数学知识

第五章

动态规划

PART II

第二部分

算法提高

PART III

第三部分

算法进阶

PART IV

第四部分

LeetCode 究极班