CSP 复赛复习 - 基础算法(5)

5. 排序算法

排序的基本概念

排序定义：将一组数据按照特定顺序重新排列的过程

排序稳定性：如果两个相等的元素在排序前后的相对位置不变，则称该排序算法是稳定的

常见排序分类：

- 比较排序：基于元素比较的排序
- 非比较排序：基于其他方法的排序

时间复杂度对比：

排序算法	平均时间复杂度	最坏时间复杂度	空间复杂度	稳定性
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(k)$	稳定

冒泡排序

基本思想：重复遍历数组，比较相邻元素，如果顺序错误就交换

特点：

- 每轮遍历将最大的元素"冒泡"到正确位置
- 稳定排序算法
- 可以优化：如果某轮没有交换，说明已经有序

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // 优化：标记是否发生交换
        bool swapped = false;

        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // 交换相邻元素
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        // 如果没有发生交换，说明已经有序
        if (!swapped) break;
    }
}
```

冒泡排序示例

```
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "原始数组: ";
    printArray(arr, n);

    bubbleSort(arr, n);

    cout << "排序后数组: ";
    printArray(arr, n);

    return 0;
}
```

执行过程：

```
第1轮: 34 25 12 22 11 64 90
第2轮: 25 12 22 11 34 64 90
第3轮: 12 22 11 25 34 64 90
第4轮: 12 11 22 25 34 64 90
第5轮: 11 12 22 25 34 64 90
```

选择排序

基本思想：每次从未排序部分选择最小（或最大）元素，放到已排序部分的末尾

特点：

- 不稳定排序算法
- 交换次数较少
- 无论数据如何，比较次数固定

```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // 找到未排序部分的最小元素索引
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        // 将最小元素交换到当前位置
        if (minIndex != i) {
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}
```

选择排序示例

```
int main() {
    int arr[] = {29, 10, 14, 37, 13};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "原始数组: ";
    printArray(arr, n);

    selectionSort(arr, n);

    cout << "排序后数组: ";
    printArray(arr, n);

    return 0;
}
```

执行过程：

```
第1轮: 10 29 14 37 13  (找到最小值10)
第2轮: 10 13 14 37 29  (找到最小值13)
第3轮: 10 13 14 37 29  (找到最小值14)
第4轮: 10 13 14 29 37  (找到最小值29)
```

插入排序

基本思想：将每个元素插入到已排序部分的正确位置

特点：

- 稳定排序算法
- 对小规模数据或基本有序数据效率高
- 适合在线排序（数据逐个到达）

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];  // 当前要插入的元素
        int j = i - 1;

        // 将比key大的元素向后移动
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        // 插入key到正确位置
        arr[j + 1] = key;
    }
}
```

插入排序示例

```
int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "原始数组: ";
    printArray(arr, n);

    insertionSort(arr, n);

    cout << "排序后数组: ";
    printArray(arr, n);

    return 0;
}
```

执行过程：

```
第1轮: 11 12 13 5 6   (插入11)
第2轮: 11 12 13 5 6   (插入13)
第3轮: 5 11 12 13 6   (插入5)
第4轮: 5 6 11 12 13   (插入6)
```

计数排序

基本思想：统计每个元素出现的次数，然后根据计数重构有序数组

适用条件：

- 元素范围已知且不大
- 元素为非负整数
- 数据范围 k 远小于数据量 n

```
void countingSort(int arr[], int n) {
    // 找到最大值，确定计数数组大小
    int maxVal = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > maxVal)
            maxVal = arr[i];
    // 创建计数数组并初始化
    int count[maxVal + 1] = {0};
    // 统计每个元素出现次数
    for (int i = 0; i < n; i++)
    // 计算前缀和（累计计数）
        count[arr[i]]++;
    for (int i = 1; i <= maxVal; i++)
        count[i] += count[i - 1];
    // 创建临时数组存放排序结果
    int output[n];
    // 从后往前遍历，保证稳定性
    for (int i = n - 1; i >= 0; i--) {
        output[count[arr[i]] - 1] = arr[i];
        count[arr[i]]--;
    }
    // 将结果复制回原数组
    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}
```

计数排序简化版

```
// 简化版计数排序（适用于小范围非负整数）
void countingSortSimple(int arr[], int n) {
    // 假设数据范围在0-100
    const int RANGE = 101;
    int count[RANGE] = {0};

    // 统计频率
    for (int i = 0; i < n; i++)
        count[arr[i]]++;

    // 重构数组
    int index = 0;
    for (int i = 0; i < RANGE; i++) {
        while (count[i] > 0) {
            arr[index++] = i;
            count[i]--;
        }
    }
}
```

计数排序示例

```
int main() {
    int arr[] = {4, 2, 2, 8, 3, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "原始数组: ";
    printArray(arr, n);

    countingSort(arr, n);

    cout << "排序后数组: ";
    printArray(arr, n);

    return 0;
}
```

执行过程：

```
计数数组: [0,1,2,2,1,0,0,0,1] (对应0-8)
前缀和:   [0,1,3,5,6,6,6,6,7]
重构数组: [1,2,2,3,3,4,8]
```

排序算法性能对比

排序算法	最佳情况	平均情况	最坏情况	空间复杂度	稳定性	适用场景
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定	教学用途
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定	交换次数少
插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定	小数据、基本有序
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(k)$	稳定	小范围整数

STL 排序函数

```cpp
#include <algorithm>
using namespace std;

int main() {
    int arr[] = {5, 2, 8, 1, 9};
    int n = sizeof(arr) / sizeof(arr[0]);

    // 使用STL排序（快速排序实现）
    sort(arr, arr + n);

    // 降序排序
    sort(arr, arr + n, greater<int>());

    // 自定义比较函数
    struct Point {
        int x, y;
    };
    Point points[5] = {{1,2}, {3,1}, {2,3}, {4,0}, {0,4}};

    // 按x坐标排序
    sort(points, points + 5, [](Point a, Point b) {
        return a.x < b.x;
    });

    return 0;
}
```