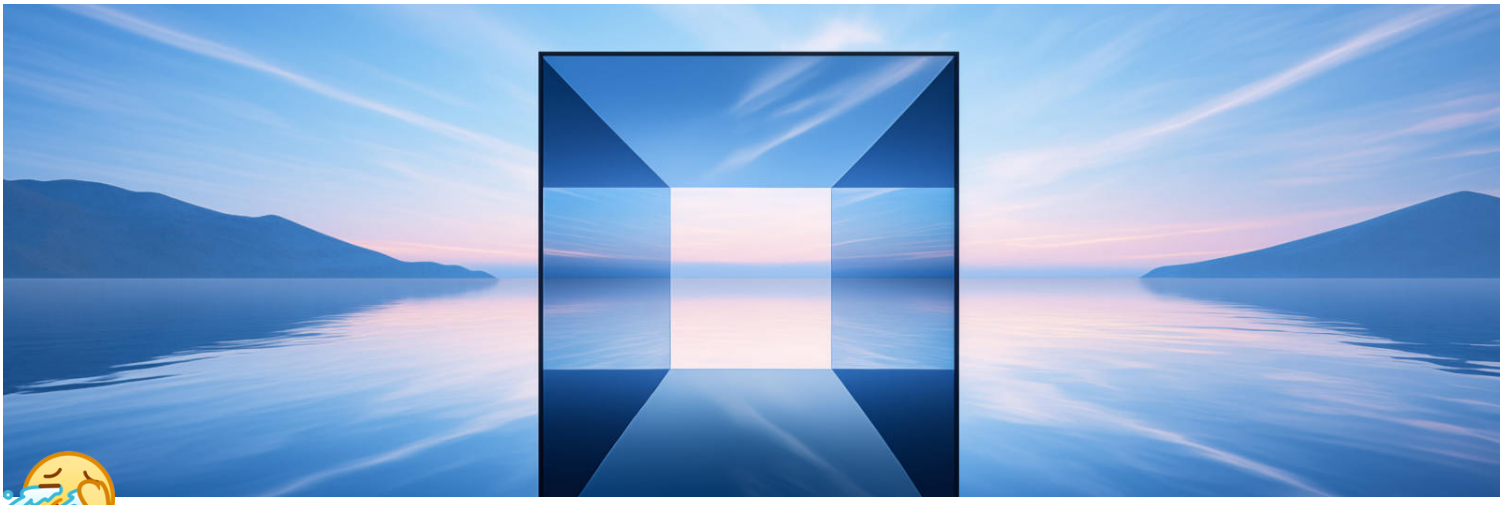

算法

一、排序算法

1、冒泡排序

`步骤`：假设有n个元素，比较相邻的元素，把更大的元素放在后面，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数；针对前n-1个元素重复以上的步骤。

- 时间复杂度：`O(n^2)`。
- 优点：实现简单，容易理解。
- 缺点：效率低下，不适合大型数据集。

2、快速排序

采用分治策略来对一个序列进行排序。

`步骤`：选择一个基准，随后遍历数列，将小于基准的元素移到基准的左边，大于基准的元素移到基准的右边，这个过程可以使用双指针。递归地对左右子数组执行上述快速排序过程。

- 时间复杂度：平均为`O(n·logn)`，最坏情况为`O(n^2)`。
- 优点：平均时间复杂度低，效率高，适用于大规模数据集。
- 缺点：最坏情况下时间复杂度较高，并且不适用于小规模数据或基本有序的数据集。

3、插入排序

`工作原理`：通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

- 时间复杂度：`O(n^2)`。

4、选择排序

`工作原理`：每次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余未排序元素中继续寻找最小（或最大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。

- 时间复杂度：`O(n^2)`。

5、归并排序

1. **分解**：将待排序的序列不断拆分为子序列，直到每个子序列只包含一个元素。
2. **合并**：将两个有序的子序列合并成一个有序的子序列。
3. **重复合并**：将合并后的子序列再次合并，直到得到一个完整的有序序列。

- 时间复杂度：`O(n·logn)`。

6、sort算法

`sort` 默认使用 Timsort 算法，这是一种混合排序算法，结合了归并排序和插入排序的特点。

对于平均情况和大于 100 个元素的数组，`sort` 的时间复杂度是 `O(n·logn)`，其中 n 是数组中的元素数量。这是因为 Timsort 在合并和插入步骤中的操作是 O(n·log n)。

对于小于 100 个元素的数组，Timsort 使用了插入排序，因此其时间复杂度是 `O(n^2)`。这是因为插入排序在小数组上比大数组更有效率。

二、查找算法

1、线性查找

在线性查找中，我们从头到尾遍历数组或列表，逐个检查每个元素，直到找到我们正在寻找的目标元素

- 时间复杂度：`O(n)`。

2、二分查找

适用于已经排序的数组，基本思想是，将待查找的元素与有序数组中间元素进行比较，如果中间元素正好是要查找的元素，则搜索过程结束；如果待查找的元素大于中间元素，则在数组大于中间元素的那部分继续查找，否则在数组小于中间元素的那部分查找。以此类推，直到找到要查找的元素。

- 时间复杂度：`O(logn)`。

三、图论算法

1、深度优先搜索（DFS）

DFS通常使用递归来实现，它会递归地深入到图的一个分支。当一个分支走到尽头时，DFS会回溯到上一个分叉点，尝试其他路径。会尽可能深地遍历图中的一个分支，直到不能再深入为止。

`优点`

- 能够找到从源点到目标点的最短路径（如果图中没有环）。
- 在树形结构中，DFS可以用于计算节点的深度。

缺点

- 在图中进行DFS时，可能会遇到回溯问题，需要大量的重复搜索。
- 深度优先可能导致搜索路径过长，特别是在大型图中。

使用场景

- 当图的结构比较深或者包含大量的分支时。
- 当需要找到从源点到目标点的最短路径时（无环图）。

2、广度优先搜索（BFS）

BFS通常使用队列来实现，它会按照从最近到最远的顺序访问节点。BFS会首先访问起始节点，然后访问它的所有邻居，然后再访问这些邻居的邻居，以此类推。会先访问最近的节点，然后逐渐向外扩展，确保每个节点都被访问到。

优点

- 能够找到从源点到目标点的最短路径（在无权图中）。
- 能够有效地遍历图中的所有节点。

缺点

- 需要额外的空间来存储队列，特别是在大型图中。
- 在图中进行BFS时，需要额外的步骤来找到从源点到目标点的最短路径。

使用场景

- 当图的结构比较浅或者包含大量的节点时。
- 当需要找到从源点到目标点的最短路径时（无权图）。

四、算法设计

1、分治

分治：将问题分解成几个较小的相同问题，递归地解决这些子问题，然后将子问题的解合并为原始问题的解。分治算法在每一步都将问题分成两部分，然后分别解决，最后将两部分的结果合并。

分治算法的示例

- **快速排序**：将数组分为两部分，分别对这两部分递归排序。
- **归并排序**：将数组分成多个子数组，然后合并这些子数组。

2、回溯

回溯是一种在搜索过程中，当发现当前路径不满足解的条件时，退回上一步，尝试其他路径的算法。

回溯算法的特点

- **试探**：从第一个元素开始，试探每个元素是否可以作为解的一部分。
- **递归**：递归地尝试所有可能的解。
- **剪枝**：在搜索过程中，剪除不可能成为解的路径。

3、动态规划

动态规划：将复杂问题分解为更小的子问题，并存储这些子问题的解，以避免重复计算。

重点：`转移方程` + `边界条件`

`动态规划的特点`

- **最优子结构**：问题的最优解包含其子问题的最优解。
- **重叠子问题**：问题分解出的子问题有重叠部分。
- **存储子问题的解**：避免重复计算，提高效率。
- **边界条件**：确定子问题的解。

4、贪心

贪心算法是一种在每一步选择中都采取当前状态下最优（或看似最优）的选择，从而希望导致全局最优解的算法。贪心算法并不保证得到最优解，但在很多情况下，贪心算法能够得到接近最优解，且实现简单。