1. 课程介绍

* 1. 什么是排序,排序的稳定性;（了解）
* 2. 常见的排序法;（了解）
* 3.排序方法的实现;（掌握）
* 4. 排序算法的效率; (了解)
* 5. 二分查找的实现; (掌握)
* 6. 二分查找的时间复杂度.(了解)

1. 排序的基本概念

**排序算法**（英语：Sorting Algorithm）是一种能将一串数据依照特定顺序进行排列的一种算法。

* 1. 排序算法的稳定性

**稳定排序**：让原本有相等键值的记录维持相对次序不变的排序。[1,2,4,2]

**不稳定排序**：可能会让原本有相等键值的记录维持相对次序改变的排序

如果一个排序算法是稳定的，当两个相等键值的记录R和S在原序列中R在S之前，那么排序后的序列中R也会在S之前。

例1，假设将以下元组以其第一个元素由小到大排序:

(3, 1) (1,1) (3, 2)（5, 10）

有可能产生两种不同的结果：

(1,1) (3, 1) (3, 2) (5, 10) （稳定）

(1,1) (3, 2) (3, 1) (5, 10) （不稳定）

1. 常见的排序算法
   1. 冒泡排序

**冒泡排序**（Bubble Sort）是一种简单的交换排序算法。

**其基本思路**:

* + - 1. 对于待排序的序列，一次比较两个元素，若为逆序就把它们交换,
      2. 重复地进行步骤1,直到没有再需要交换时，排序完成。

例2. 有一组关键字序列为

{90,60,30,65,25,75,40}.

冒泡排序过程如下(高亮为下一趟冒泡区间):

冒泡次数 下一趟冒泡区间

初始 [6, 0] 90 60 30 65 25 75 40

一趟 [6, 1] 25 60 90 30 65 75 40

二趟 [6, 2] 25 30 60 90 65 75 40

三趟 [6, 3] 25 30 40 60 90 65 75

四趟 [6, 4] 25 30 40 60 90 65 75

五趟 [6, 5] 25 30 40 60 65 90 75

六趟 序列有序 25 30 40 60 65 75 90

**冒泡排序的实现**:

1. **def** bubble\_sort(list):
2. """
3. 冒泡排序方法1:大的元素向右冒泡
4. :param list: 待排序的记录(列表)
5. """
6. **for** j **in** range(len(list)-1,0,-1):
7. # j表示每次遍历需要比较的次数，是逐渐减小的
8. **for** i **in** range(j):
9. **if** list[i] > list[i+1]:
10. list[i], list[i+1] = list[i+1], list[i]
12. li = [90,60,30,65,25,75,40]
13. bubble\_sort(li)
15. **print** li
16. **def** bubble\_sort(list):
17. """
18. 冒泡排序方法2:小的元素向左冒泡
19. :param list: 待排序的记录(列表)
20. """
21. **for** j **in** range(len(list),0,-1):
22. # j表示每次遍历需要比较的次数，是逐渐减小的
23. **for** i **in** range(1,j):
24. **if** list[-i] < list[-i-1]:
25. list[-i], list[-i-1] = list[-i-1], list[-i]
27. li = [90,60,30,65,25,75,40]
28. # 75 40
29. bubble\_sort(li)
31. **print**(li)

**时间复杂度**

时间复杂度：O()

稳定性：稳定

**使用情形:**

当待排序序列**原顺序较好**时,冒泡排序效率较高.

n较小时可以采用.

* 1. 选择排序

**选择排序**（Selection Sort）的基本思路:

* + - 1. 首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，
      2. 然后，从剩余未排序元素中继续寻找最小（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。

选择排序的主要优点与数据移动有关。如果某个元素位于正确的最终位置上，则它不会被移动。选择排序每次交换一对元素，它们当中至少有一个将被移到其最终位置上，因此对n个元素的序列排序总共至多进行n-1次交换。在完全依靠交换去移动元素的排序方法中，选择排序是非常好的一种。

**例3**. 有一组关键字序列为

{90,60,30,65,25,75,40}.

选择排序过程如下(高亮为有序序列):

第1步:在7个元素中,找出最小元素,放在位置

Min\_index

初始 0 90 60 30 65 25 75 40

比较 1 90 60 30 65 25 75 40

比较 2 90 60 30 65 25 75 40

比较 2 90 60 30 65 25 75 40

比较 4 90 60 30 65 25 75 40

比较 4 90 60 30 65 25 75 40

比较 4 90 60 30 65 25 75 40

交换 2 25 60 30 65 90 75 40 {25}已是有序序列

第2步:在未排序的6个元素中,找出最小元素,放在位置

Min\_index

初始 1 25 60 30 65 90 75 40

比较 2 25 60 30 65 90 75 40

比较 2 25 60 30 65 90 75 40

比较 2 25 60 30 65 90 75 40

比较 2 25 60 30 65 90 75 40

比较 2 25 60 30 65 90 75 40

交换 2 25 30 60 65 90 75 40 {25, 30}已是有序序列

第3步:在未排序的5个元素中,找出最小元素,放在位置

Min\_index

初始 2 25 30 60 65 90 75 40

比较 2 25 30 60 65 90 75 40

比较 2 25 30 60 65 90 75 40

比较 2 25 30 60 65 90 75 40

比较 6 25 30 60 65 90 75 40

交换 2 25 30 40 65 90 75 60 {25, 30,40}已是有序序列

…….

第6步:在未排序的2个元素中,找出最小元素,放在位置

Min\_index

初始 5 25 30 40 60 65 75 90

比较 5 25 30 40 60 65 75 90

交换 5 25 30 40 60 65 75 90 排序完成.

**选择排序实现**:

1. **def** selection\_sort(list):
2. """
3. 选择排序
4. :param list: 待排序序列
5. """
6. n = len(list)
7. # 需要进行n-1次选择操作
8. **for** i **in** range(n-1):
9. # 记录最小位置
10. min\_index = i
11. # 从i+1位置到末尾两两比较,选出最小数据
12. **for** j **in** range(i+1, n):
13. **if** list[j] < list[min\_index]:
14. min\_index = j
15. # 如果选择出的数据不在正确位置，进行交换
16. **if** min\_index != i:
17. list[i], list[min\_index] = list[min\_index], list[i]
19. list = [54,26,93,17,77,31,44,55,20]
20. selection\_sort(list)
22. **print** (list)

**时间复杂度**

最优时间复杂度: O()

最坏时间复杂度：O()

稳定性：不稳定（考虑升序每次选择最大的情况）

(N-1)\*(N-1)= N^2

**使用场景**

N较小时使用.

* 1. 插入排序
     1. 直接插入排序

**直接插入排序**（Insertion Sort）:

直接插入排序的基本操作就是将一个数据插入到有序表{}中，从而得到一个新的、长度为的有序表.

换句话说: 在处理第个记录时,前面的个记录已排成有序,将第个记录插入到该有序表中,得到一个新的长度为的有序表.

**当表中只有一个元素时,该表是一个有序表**.

**例4**:有一组关键字序列为

{90,60,30,65,25,75,40}.

直接插入排序过程如下: 高亮部分为有序表.

处理记录号 插入位置

i=1 j = 0 90 60 30 65 25 75 40

i=2 j = 0 60 90 30 65 25 75 40

i=3 j = 0 30 60 90 65 25 75 40

i=4 j = 2 30 60 65 90 25 75 40

i=5 j = 0 25 30 60 65 90 75 40

i=6 j = 4 25 30 60 65 75 90 40

i=7 j = 2 25 30 40 60 65 75 90

**插入排序实现**:

1. **def** insert\_sort(list):
2. """
3. 直接插入排序: 从第二个位置(即下标为1的元素)开始向前插入
4. """
5. **for** i **in** range(1, len(list)):
6. # 从第i个元素开始向前比较，如果小于前一个元素，交换位置
7. **for** j **in** range(i, 0, -1):
8. **if** list[j] < list[j-1]:
9. list[j], list[j-1] = list[j-1], list[j]
11. list = [90,60,30,65,25,75,40]
12. insert\_sort(list)
14. **print**(list)

**时间复杂度**

最优时间复杂度：O(n) （升序排列，序列已经处于升序状态）

最坏时间复杂度： O()

稳定性：稳定

说明:

* + - 1. 该算法适用于少量数据的排序.
      2. 通过一趟冒泡排序后,待排序序列的长度减1.
      3. 大部分已经排好序时,该算法较好.
    1. 希尔排序

**希尔排序**(Shell Sort)是插入排序的一种,也称**缩小增量排序**，是直接插入排序算法的一种更高效的改进版本.希尔排序是非稳定排序算法.

**基本思想**:

* + - 1. 把待排序的序列按某个增量分组(一般取为序列中元素的个数)，对**每组**进行直接插入排序；
      2. 然后按增量分组 (一般取), 对每组再进行直接插入排序;
      3. 重复第二步,当增量减至1时，对序列进行直接插入排序,排序完成。

**例5**. 把以下序列由小到大排序.

57 68 59 51 70 27 90 30 25 18 (n=10)

第一步: 取间隔 (相同颜色高亮元素分别属于同一组)

57 68 59 51 70 27 90 30 25 18

27 57

每组分别用直接插入排序:

27 68 30 25 18 57 90 59 51 70

第二步: 取间隔 (取奇数)

27 68 30 25 18 57 90 59 51 70

每组分别用直接插入排序:

25 27 70 90

25 18 30 27 59 51 70 68 57 90

第三步: 取间隔 (取奇数)

25 18 30 27 59 51 70 68 57 90

用直接插入排序: (高亮为有序序列)

18 25 30 27 59 51 70 68 57 90

18 25 30 27 59 51 70 68 57 90

18 25 27 30 59 51 70 68 57 90

18 25 27 30 59 51 70 68 57 90

18 25 27 30 51 59 70 68 57 90

18 25 27 30 51 59 70 68 57 90

18 25 27 30 51 59 68 70 57 90

18 25 27 30 51 57 59 68 70 90

18 25 27 30 51 57 59 68 70 90

**希尔排序的实现**:

1. **def** shell\_sort(list):
2. """希尔排序"""
3. n = len(list)
4. # 初始步长
5. gap = n / 2
6. **while** gap > 0:
7. # 按步长进行插入排序
8. **for** i **in** range(gap, n):
9. j = i
10. # 插入排序
11. **while** j>=gap **and** list[j-gap] > list[j]:
12. list[j-gap], list[j] = list[j], list[j-gap]
13. j -= gap
14. # 得到新的步长
15. gap = gap / 2
17. list = [57, 68, 59, 51, 70, 27, 90, 30, 25, 18]
18. shell\_sort(list)
20. **print** (list)

**时间复杂度**

最优时间复杂度：根据步长序列的不同而不同

最坏时间复杂度：O()

稳定性：不稳定

* 1. 快速排序

**快速排序**(Quick sort），又称**划分交换排序**（partition-exchange sort），是对冒泡排序的一种改进.

**基本思想**:

通过一趟冒泡排序后,要使待排序序列的长度大幅减少.

具体地,通过一趟排序将要排序的数据分割成**独立的两部分**，其中一部分的所有数据都比另外一部分的所有数据**都要小**，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

**步骤**：

* + - 1. 从序列中挑出一个元素(通常为第一个)，称为"**基准**"（pivot）;
      2. 将所有比基准值小的元素放在基准之前，所有比基准值大的元素放在基准之后. 在这个操作结束后，该基准就处于序列的中间位置。这个操作称为**划分**。
      3. 对小于基准值元素的子序列和大于基准值元素的子序列**重复步骤1,2**,直到原序列变成有序序列. (这个操作称为递归)

**例6a**. 有一组关键字序列为

{29,60,30,15,25,75,40}.

15, 25， 29， 60，30，75， 40

1. 40， **60**， 75

快速排序过程如下:

初始 29 60 30 15 25 75 40 (选序列的第一个元素29为基准)

Δ

=0

从j=6所在位置向左扫描,发现25<=29, 将25放到所在位置, 现在

25 29 60 30 15 75 40

Δ

i=1

从j=6所在位置向左扫描,发现15<=29, 将15放到所在位置, 现在

25 15 29 60 30 75 40

Δ

i=2

-----------------------------------一次划分---------------------------------------------

对子序列{25,15,29}操作

25 15 29 60 30 75 40

Δ

从j=2所在位置向左扫描,发现15<=25, 将15放到所在位置, 现在,子序列{15,25,29}有序.

15 25 29 60 30 75 40

Δ

i=1

对子序列{60,30,75,40}操作

15 25 29 60 30 75 40

Δ

从j=6所在位置向左扫描,发现40<=60, 将40放到所在位置, 现在

15 25 29 40 60 30 75

Δ

从j=6所在位置向左扫描,发现30<=60, 将30放到所在位置, 现在

15 25 29 40 30 60 75

Δ

对子序列[40,30]操作

15 25 29 40 30 60 75

Δ

从j=4所在位置向左扫描,发现30<=40, 将30放到所在位置, 现在, 该子序列有序.

15 25 29 30 40 60 75

Δ

-----------------------------------------------排序完成-------------------------------------------------------------------------

15 25 29 30 40 60 75

**例6b. 有一组关键字序列为**

**{29,60,30,15,25,75,40}.**

**快速排序过程如下:**

初始 29 60 30 15 25 75 40 (选序列的第一个元素29为基准)

Δ

=0

i =0 j =5 (j减1)

i =0 j=4 (j减1)

25 60 30 15 29 75 40

i =1 j=4

25 29 30 15 60 75 40

i=1 j =3

25 15 30 29 60 75 40

i=2 j=3

25 15 29 30 60 75 40

i=2

j =2 (第一次划分.结果： 基准左边的所有元素都比基准小，基准右边的元素都比基准大.)

------------------------------------------第一次划分-----------------------------------------------------------------------

对左子序列做快速排序： **25** 15

i=0 j=1

15 **25**

i=1

j=1 (第二次划分： 基准为25。左子序列排序完成)

对右子序列做快速排序： **30** 60 75 40

i=3 j=6

i=3 j=5

i=3 j=4

i=3

j=3 (第二次划分：基准30.)

------------------------------------------------------第二次划分------------------------------------------------------------

**60** 75 40

i=4 j=6

40 75 **60**

i=5 j=6

40 **60** 75

i =5

j=5 (第三次划分：基准为60.)

15 25 29 30 40 60 75 （**排序完成！**）

**快速排序的实现:**

1. def quick\_sort(list, start, end):
2. """快速排序"""
3. # 递归的退出条件
4. **if** start >= end:
5. **return**
6. # 设定起始元素为要寻找位置的基准元素
7. mid = list[start]
8. # low为序列左边的由左向右移动的游标
9. low = start
10. # high为序列右边的由右向左移动的游标
11. high = end
13. **while** low < high:
14. # 如果low与high未重合，high指向的元素不比基准元素小，则high向左移动
15. **while** low < high **and** list[high] >= mid:
16. high -= 1
17. # 将high指向的元素放到low的位置上
18. list[low] = list[high]
20. # 如果low与high未重合，low指向的元素比基准元素小，则low向右移动
21. **while** low < high **and** list[low] < mid:
22. low += 1
23. # 将low指向的元素放到high的位置上
24. list[high] = list[low]
26. # 退出循环后，low与high重合，此时所指位置为基准元素的正确位置
27. # 将基准元素放到该位置
28. list[low] = mid
30. # 对基准元素左边的子序列进行快速排序
31. quick\_sort(list, start, low-1)
33. # 对基准元素右边的子序列进行快速排序
34. quick\_sort(list, low+1, end)
36. li = [90,60,30,65,25,75,40]
37. quick\_sort(li,0,len(li)-1)
39. **print** （li）

**时间复杂度**

最坏时间复杂度：O()

稳定性：不稳定

从一开始快速排序平均需要花费O(n log n)时间的描述并不明显。但是不难观察到的是分区运算，数组的元素都会在每次循环中走访过一次，使用O(n)的时间。

**使用场景**

N较大时的情形

记忆: **一次划分得到两个小序列**.

* 1. 归并排序

**归并**是指将两个有序序列合为一个有序序列. **归并排序**是采用分治法的一个非常典型的应用。

**基本思想**:

* + - 1. 把n个记录都看成长度为1的有序表,将它们两两归并,得到长度为2的若干个有序表;
      2. 重复步骤1,直到得到长度为n的有序表为止.

**例7**. 有一组关键字序列为{50, 25,13,17,44,39,2}.

归并排序过程如下:

初始关键字 [50] [25] [13] [17] [44] [39] [2]

一趟归并之后 [25 50] [13 17] [39 44] [2] [13,17, 25, 50 ]

二趟归并之后 [13 17 25 50] [2 39 44]

二趟归并之后 [2 13 17 25 39 44 50]

**-------------------------------**完成排序**-----------------------------------**

**归并排序的实现**:

1. **def** merge\_sort(list):
2. **if** len(list) <= 1:
3. **return** list
4. # 二分分解
5. num = len(list)/2
6. left = merge\_sort(list[:num])
7. right = merge\_sort(list[num:])
8. # 合并
9. **return** merge(left,right)
11. **def** merge(left, right):
12. """
13. 合并操作，将两个有序数组left[]和right[]合并成一个大的有序数组
14. """
15. #left与right的下标指针
16. l, r = 0, 0
17. result = []
18. **while** l<len(left) **and** r<len(right):
19. **if** left[l] < right[r]:
20. result.append(left[l])
21. l += 1
22. **else**:
23. result.append(right[r])
24. r += 1
25. result += left[l:]
26. result += right[r:]
27. **return** result
29. list = [50,25,13, 17,44,39,2]
30. sorted\_list = merge\_sort(list)
32. **print**(sorted\_list)

**时间复杂度**

最优时间复杂度：O( n\*log(n) )

最坏时间复杂度：O( n\*log(n) )

稳定性：稳定

**使用场景**

N较大时可以采用

* 1. 常见排序算法效率比较

**排序算法 时间复杂度 稳定性**

-----------------------------------------------------

冒泡排序 O() 稳定

选择排序 O() 不稳定

直接插入排序 O() 稳定

希尔排序 O() 不稳定

快速排序 O() 不稳定

归并排序 O(nlogn) 稳定

1. 搜索
   1. 为什么要学习搜索?

很多人工智能问题都是用搜索方法来求解. 比如对抗搜索,约束条件问题(CSP).

* 1. 什么是搜索?

**搜索**(search):

检查给定数据对象的每个结点,寻求一条从开始结点到答案结点的**路径**,并输出问题的解.

通常,搜索的答案是真或假(或者说某个目标条件被满足).

几种常见的搜索方法：

顺序查找、**二分法查找**、二叉树查找、哈希查找.

**状态空间**(state space):

所求问题的各种可能情况的**集合**.

注意:

1. 这是搜索最基本的含义.

2. 高级的搜索算法,并不需要寻找到一条从初始结点到答案结点的路径,而是给出一个策略.比如,最典型的时对抗搜索中,搜索空间太大,对手的决定我们不可控制,故只能搜索出一个策略.

1. 常见的搜索方法
   1. 折半查找

**折半查找**，又称**二分查找.**其过程是:先确定待查找记录所在范围,然后逐步缩小范围,直到得到结果.

折半查找适用于有序列表。

**折半查找的具体过程:**

* + - 1. 先假设表中元素是按升序排列，将表中间位置记录的关键字与查找关键字比较，若两者相等，则查找成功.
      2. 中间位置记录的关键字与查找关键字不相等,则利用中间位置记录将表分成前、后两个子表，若中间位置记录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。
      3. 重复以上过程，直到查找成功，或子表不存在(查找失败)。

**折半查找的实现:**

1. # 非递归实现
2. **def** binary\_search(alist, item):
3. first = 0
4. last = len(alist)-1
5. **while** first<=last:
6. midpoint = (first + last)/2
7. **if** alist[midpoint] == item:
8. **return** True
9. **elif** item < alist[midpoint]:
10. last = midpoint-1
11. **else**:
12. first = midpoint+1
13. **return** False
15. testlist = [0, 5,  8, 13, 17, 19, 21, 34, 55]
16. **print**(binary\_search(testlist, 3))
17. **print**(binary\_search(testlist, 17))
19. # 递归实现
20. **def** binary\_search(alist, item):
21. **if** len(alist) == 0:
22. **return** False
23. **else**:
24. midpoint = len(alist)//2
25. **if** alist[midpoint]==item:
26. **return** True
27. **else**:
28. **if** item<alist[midpoint]:
29. **return** binary\_search(alist[:midpoint],item)
30. **else**:
31. **return** binary\_search(alist[midpoint+1:],item)
33. testlist = [0, 5,  8, 13, 17, 19, 21,  34, 55]
34. **print**(binary\_search(testlist, 3))
35. **print**(binary\_search(testlist, 17))

优点:比较次数少，查找速度快，平均性能好；

缺点:要求待查表为有序表，且插入删除困难。

**时间复杂度**

最优时间复杂度：O(1)

最坏时间复杂度：O(log)

1. 课程总结
   1. 重点
      * 1. 掌握常见排序算法的实现
        2. 掌握折半查找的实现
   2. 难点
      * 1. Shell排序算法
        2. 快速排序的实现
        3. 状态空间
2. 习题

一组记录的关键值为(45, 75, 56, 35, 40, 90). 利用快速排序方法,得到的第一趟排序结果为\_\_\_\_.

1. 35,40,45,56,75,90
2. 40,35,45,75,56,90
3. 40,35,45,56,75,90
4. 40,35,45,90,56,75
5. 扩展知识或课外阅读推荐
   1. 课外阅读

刘清,张小勇,王琼 <<数据结构>>(第三版),电子工业出版社 (2010)

裘宗燕,<<数据结构与算法>>,机械工业出版社 (2015)