

Факультет программной инженерии и компьютерной техники

*Системы искусственного интеллекта*

№1. Изучение работы Prolog программы

|  |  |
| --- | --- |
|  | **Работу выполнили**  Сыюань Хуан  **Преподаватель**  Игорь Бессмертный |

Санкт-Петербург

2021

### 

### **Текст программы с комментариями**

1. Сортировка методом вставки
2. *% If there is nothing, then sort is over*
3. insrtsort([],[]).
5. insrtsort([Head | Tail], ListSorted) :-
6. *% We use parameter "Tail", because we need to sort from the end to*
7. *% the beginning.*
8. insrtsort(Tail, TailSorted),
10. *% After recursion, we will sort the last two number.*
11. *% Then we sort the rest of numbers from the end to the beginning.*
12. *% This can make sure the "TailSorted" is 100% sorted.*
13. insrt(Head, TailSorted, ListSorted).
15. insrt(X, [Y | ListSorted], [Y | ListSorted1]) :-
16. *% If X > Y, we put Y in front of X.*
17. *% Then we recurse X with the rest of the array.*
18. *%*
19. *% If we find X < Y, method will stop.*
20. *% So this method can't sort the "ListSorted",*
21. *% that's why we need to make sure the list parameter is sorted.*
22. X > Y,
24. *% "!" is used to interrupt the backtracking. If we don't use it,*
25. *% the program won't be end.*
26. !,
27. insrt(X, ListSorted, ListSorted1).
29. *% Put X in front of the sorted list.*
30. insrt(X, ListSorted, [X | ListSorted]).
31. Быстрая сортировка
32. fastsort([],[]).
33. fastsort([Head | Tail], ListSorted) :-
35. *% We pick the first number for comparison.*
36. split(Head, Tail, TailLess, TailGreater),
38. *% Sort "TailLess" with recursion.*
39. fastsort(TailLess, TailLessSorted),
40. *% Sort "TailGreater" with recursion.*
41. fastsort(TailGreater, TailGreaterSorted),
43. *% We will get TailLessSorted + TailGreaterSorted.*
44. append(TailLessSorted, [Head | TailGreaterSorted], ListSorted).
46. *% This method can split the head of the tail.*
47. *% If split null, then get null.*
48. split(\_, [], [], []).
50. *% If Head is less than X, then we put the H to the "TailLess".*
51. split(X, [H | T], [H | TL], TG) :- H < X, !, split(X, T, TL, TG).
53. *% If Head is greater than X, then we put the H to the "TailGreater".*
54. split(X, [H | T], TL, [H | TG]) :- split(X, T, TL, TG).

### **Cтруктурированная трасса хода выполнения 1.** Сортировка методом вставки

### 表格 中度可信度描述已自动生成

**2.** Сортировка методом вставки

文本

描述已自动生成

**3. Описание логики**

* + - 1. Сортировка методом вставки

The logic is we sort 2 numbers to get the sorted list. Then we compare the new number with the sorted list. We insert the new number to the correct place. We continue to perform repeated operations. Finally we get the sorted list.

信件

描述已自动生成

* + - 1. Быстрая сортировка

We pick the first number as the pivot key. Then we compare it with the rest of the list. Split them to the “LessTail” and the “GreatTail”. For these two arrays we do the same operation and recurse. After the recursion is over, we need to backtrack. Through method “append”, we will append the sub array of the traceback to the answers. When the backtracking is complete, we get the answer.

图示

中度可信度描述已自动生成

**4. Cравнение методов**

The average time complexity for Insertion Sort is O(n2) and for Quicksort is O(nlogn). And the space complexity are O(1) and O(logn).

If we need to sort a large amount of data, the Quicksort is better than the insertion sort. Because even if we encounter the worst case, the time complexity is O(n2). It’s as fast as the average case of Insertion Sort.

But if we need to sort a few numbers, the Insertion Sort is better. Because it doesn’t matter which method we choose, it will only speed a little time. And the Insertion Sort takes up less space and more stable. So, we should pick Insertion Sort.

And if the list is already ordered or almost ordered. The Insertion Sort is better than Quicksort. Because it’s the best case for Insertion Sort. The time complexity is close to O(n). For quick sort, the situation is just the opposite.

It will be like Bubble Sort. The time complexity will be close to O(n2).