Практическая работа №1

Квадратичные алгоритмы сортировки

В работе рассматриваются алгоритмы сортировки вставки и выбором. Рассматривается целесообразность использования этих алгоритмов, их преимущества и недостатки.

Вопросы:

1. Общие сведения о задачах сортировки, понятие временной сложности
2. Сортировка выбором
3. Сортировка вставками
4. Задание
5. Оформление отчета
6. Литература.

1. Общие сведения о задачах сортировки, понятие временной сложности алгоритмов сортировки.

Пусть есть последовательность a_0, a_1... a_n и функция сравнения, которая на любых двух элементах последовательности принимает одно из трех значений: меньше, больше или равно. Задача сортировки состоит в перестановке членов последовательности таким образом, чтобы выполнялось условие: $a_i <= a_{i+1}$, для всех i от 0 до n.

Возможна ситуация, когда элементы последовательности состоят из нескольких полей:

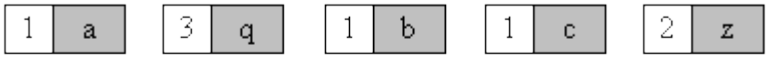
Рис. 1

```
struct element {
  field x;
  field y;
}
```

Если значение функции сравнения зависит только от поля x, то x называют ключом, по которому производится сортировка. На практике, в качестве x часто выступает число, а поле y хранит какие-либо данные, никак не влияющие на работу алгоритма.

Рассмотрим параметры, по которым производится оценка алгоритмов сортировки

1. *Время* сортировки - основной параметр, характеризующий быстродействие алгоритма.
2. *Память* - ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных..
3. *Устойчивость* - устойчивая сортировка не меняет взаимного расположения равных элементов. Такое свойство может быть очень полезным, если они состоят из нескольких полей, а сортировка происходит по одному из них, например, по x. (рис. 2 и 3)
4. *Естественность поведения* - эффективность метода при обработке уже отсортированных, или частично отсортированных данных. Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.



Исходные данные

Пример работы устойчивой сортировки.

Рис. 2

Взаимное расположение равных элементов с ключом 1 и дополнительными полями "a", "b", "c" осталось прежним: элемент с полем "a", затем - с "b", затем - с "c".

Пример работы неустойчивой сортировки.

Рис. 3

Взаимное расположение равных элементов с ключом 1 и дополнительными полями "a", "b", "c" изменилось.

Временная сложность алгоритма определяет время работы, используемое алгоритмом, как функции от длины строки, представляющей входные данные. Временная сложность алгоритма обычно выражается с использованием нотации «O» большое, которая исключает коэффициенты и члены меньшего порядка. Если сложность выражена таким способом, говорят об *асимптотическом* описании временной сложности, т.е. при стремлении размера входа к бесконечности. Например, если время, которое нужно алгоритму для выполнения работы, для всех входов длины n не превосходит $5n^3 + 3n$ для некоторого n (большего некоторого n_0), асимптотическая временная сложность равна O (n^3).

Временная сложность зачастую оценивается путём подсчёта числа элементарных операций, осуществляемых алгоритмом, где элементарная операция занимает для выполнения фиксированное время. Тогда полное время выполнения и число элементарных операций, выполненных алгоритмом, отличаются максимум на постоянный множитель.

Алгоритмы сортировки пузырьком, вставками и выбором относятся к сортировкам с квадратичным временем сложности $O(n^2)$

2. Сортировка выбором

Мы начинаем *сортировку выбором* с поиска наименьшего элемента в списке и обмена его с первым элементом (таким образом, наименьший элемент помещается в окончательную позицию в отсортированном списке). Затем мы сканируем список, начиная со второго элемента, в поисках наименьшего среди оставшихся n-1 элементов и обмениваем найденный наименьший элемент со вторым, т.е. помещаем второй наименьший элемент в окончательную позицию в отсортированном списке. В общем случае, при i-ом проходе по списку ($0 \le i \le n$-2) алгоритм находит наименьший элемент среди последних n-i элементов и обменивает его с A_i. (рис. 4). После выполнения n-1 проходов список оказывается отсортирован.

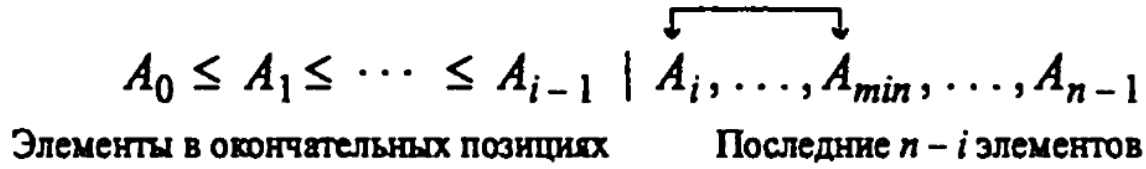


Рис.4 Алгоритм сортировки выбором

Алгоритм *Selectionsort* (*A* [0..*n* - 1])
// Сортировка массива методом выбора
// Входные данные: Массив *A* [0.. *n* - 1] упорядочиваемых элементов
// Выходные данные: Массив *A* [0.. *n* - 1], отсортированный в неубывающем
//порядке
for *i*← 0 to *n*-2 do
min ← *i*
for *j* ← *i*+1 to *n* -1 do
if *A*[j] < *A*[*min*]
min← *j*
Обмен *A*[*i*] и *A* [*min*]

В качестве примера на рис. 5 приведена сортировка выбором следующего списка: 89, 45, 68, 90, 29, 34, 17.

| 89	45	68	90	29	34	**17**
17 |	45	68	90	**29**	34	89
17	29 |	68	90	45	**34**	89
17	29	34 |	90	**45**	68	89
17	29	34	45 |	90	**68**	89
17	29	34	45	68 |	90	**89**
17	29	34	45	68	89 |	90

Рис. 5 Пример сортировки выбором

На рисунке введены следующие обозначения. Каждая строка соответствует одной итерации алгоритма, т.е. сканированию части списка справа от вертикальной черты. Полужирным шрифтом выделены наименьшие элементы, обнаруживаемые при сканировании. Элементы справа от вертикальной черты находятся в окончательных позициях и не сканируются
Результат численного моделирования приведенного псевдокода.
Исходные данные A=[89, 45, 68, 90, 29, 34, 17] n=7

1. i=0; *min*=0; *j*=0+1=1; A[1]<A[0] (45<89) min=1
    a. j=2 A[2]<A[1] (68<45) min=1
    b. j=3 A[3]<A[1] (90<45) min=1
    c. j=4 A[4]<A[1] (29<45) min=4
    d. j=5 A[5]<A[4] (34<29) min=4

e. j=6 A[6]<A[4] (17<29) min=6

2. b=A[0] A[0]=A[6] A[6]=b A[0]=17 A[6]=89
3. i=1; *min=1; j=1+1=2;* A[2]<A[1] (68<45) min=1
    1. j=3 A[3]<A[1] (90<45) min=1
    2. j=4 A[4]<A[1] (29<45) min=4
    3. j=5 A[5]<A[4] (34<29) min=4
    4. j=6 A[6]<A[4] (89<29) min=4
4. b=A[1] A[1]=A[4] A[4]=b A[1]=29 A[4]=45
5. i=2; *min=2; j=2+1=3;* A[3]<A[2] (68<90) min=2
    1. j=4 A[4]<A[1] (45<68) min=4
    2. j=5 A[5]<A[4] (34<45) min=5
    3. j=6 A[6]<A[4] (89<34) min=5
6. b=A[2] A[2]=A[5] A[5]=b A[2]=34 A[5]=68
7. i=3; *min=3; j=3+1=4;* A[4]<A[3] (45<90) min=4
    1. j=5 A[5]<A[4] (68<45) min=4
    2. j=6 A[6]<A[4] (89<45) min=4
8. b=A[3] A[3]=A[4] A[3]=b A[3]=45 A[4]=90
9. i=4; *min=4; j=4+1=5;* A[5]<A[4] (68<90) min=5
    1. j=6 A[6]<A[5] (89<68) min=5
10. b=A[4] A[4]=A[5] A[4]=b A[4]=68 A[5]=90
11. i=5; *min=5; j=5+1=6;* A[6]<A[5] (89<90) min=6
12. b=A[5] A[5]=A[6] A[5]=b A[5]=89 A[6]=90

3. Сортировка вставками

Будем разбирать алгоритм вставками, рассматривая его действия на i-м шаге. Последовательность A к этому моменту разделена на две части: готовую A[0]...A[i] и неупорядоченную A[i+1]...A[n-1]. На следующем, (i+1)-м каждом шаге алгоритма берем a[i+1] и вставляем на нужное место в готовую часть массива.

Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним.

В зависимости от результата сравнения элемент либо остается на текущем месте(вставка завершена), либо они меняются местами и процесс повторяется (Рис. 6).

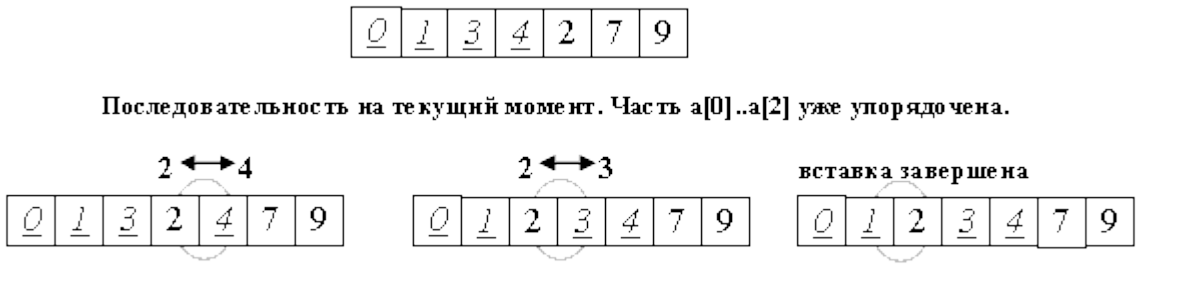


Рис. 6 Пример сортировки вставками

Таким образом, в процессе вставки мы "просеиваем" элемент *A* к началу массива, останавливаясь в случае, когда: найден элемент, меньший *A* или достигнуто начало последовательности.
Алгоритм сортировки вставками будет состоять из *n*-1-го проходов (*n* – размерность массива). Каждый из проходов включает четыре действия:

1. взятие очередного *i*-го неотсортированного элемента и сохранение его в дополнительной переменной;
2. поиск позиции *j* в отсортированной части массива, в которой присутствие взятого элемента не нарушит упорядоченности элементов;
3. сдвиг элементов массива от *i*-1-го до *j*-1-го вправо, чтобы освободить найденную позицию вставки;
4. вставка взятого элемента в найденную j-ю позицию.

Псевдокод сортировки методом вставок представлен ниже под названием *InsertionSort*.
На вход подается массив A [1..n], содержащий последовательность из n сортируемых чисел (количество элементов массива A обозначено в этом коде как length[A].) Входные числа сортируются без использования дополнительной памяти: их перестановка производится в пределах массива, и объем используемой при этом дополнительной памяти не превышает некоторую постоянную величину. По окончании работы алгоритма InsertionSort входной массив содержит отсортированную последовательность:

```
InsertionSort
for j← 2 to length[A]
do key← A[j]
►Вставка элемента A[j] в отсортированную последовательность A[1..j-1]
i←j-l
while i > 0 и A[i] > key
do A[i + 1]← A[i]
i ← i-1
A[i + 1] ← key
```

Проверьте представленный псевдокод на численном примере из сортировки выбором
Исходные данные A=[89, 45, 68, 90, 29, 34, 17] n=7

Еще один вариант псевдокода сортировки вставками.

```
SelectionSortCheckExchanges(A)
Input: A, a array of items to be sorted
Result: A is sorted
1 for i ← 0 to |A| − 1 do
2 m ← i
3 for j ← i + 1 to |A| do
4 if Compare(A[j],A[m]) < 0 then
5 m ← j
6 if i m then
7 Swap(A[i],A[m])
```

Задание:

1. Варианты заданий

Вариант	Последовательность
1	54, 65, 7, 33, 86, 29, 11, 91, 12
2	50, 80, 19, 86, 35, 7, 60, 48, 51
3	53, 12, 68, 63, 3, 47, 50, 61, 41
4	98, 40, 42, 88, 61, 87, 79, 97, 82
5	70, 80, 61, 92, 12, 23, 67, 65, 50
6	29, 1, 24, 69, 52, 97, 27, 10, 88
7	95, 43, 98, 41, 68, 67, 10, 7, 69
8	61, 32, 27, 45, 75, 58, 5, 50, 99
9	78, 77, 4, 62, 8, 69, 46, 11, 49
10	28, 76, 27, 10, 5, 35, 95, 16, 33
11	31, 40, 22, 50, 53, 68, 97, 12, 15
12	18, 20, 2, 88, 61, 17, 79, 97, 82
13	41, 52, 47, 65, 95, 38, 15, 50, 99
14	11, 42, 67, 55, 65, 78, 25, 50, 69
15	53, 5, 44, 47, 35, 83, 82, 85, 28
16	77, 89, 74, 68, 70, 49, 5, 62, 51
17	19, 75, 43, 31, 5, 66, 62, 34, 76
18	50, 57, 78, 34, 41, 68, 47, 61, 38
19	86, 36, 14, 50, 64, 21, 2, 83, 82
20	80, 27, 37, 36, 91, 53, 86, 66, 98
21	21, 44, 22, 50, 63, 68, 97, 12, 15
22	7, 89, 4, 68, 70, 49, 10, 62, 51

2. Отсортировать с помощью сортировки выбором
3. Отсортировать с помощью сортировки вставками
4. Сравнить между собой алгоритмы сортировки

Отчет должен содержать:

1. Псевдокод, результат численного моделирования и контрольный пример по вариантам заданий 1 и 2, выводы по практической работе №1