УДК 004.42(076)

СОРТИРОВКА КАК СРЕДСТВО ДЛЯ РАЗРАБОТКИ СЛОЖНЫХ И ВАЖНЫХ АЛГОРИТМОВ ПРИ ИЗУЧЕНИИ КУРСА ПРОГРАММИРОВАНИЯ В ВУЗе

Бегаришева Г.Г., Тасмухамбетова Г.М.

Мақалада іріктеудің әр түрлі алгоритмдері жүйелі түрде баяндалып, іріктеудің әр әдісіне көрнекі түсінік берілген және олардың тәжірибеде жүзеге асырылуы қарастырылған.

In article various algorithms of sorting are described, the evident explanation to each method of sorting and their practical realisation is offered.

Сортировка – один из наиболее сложных и важных для изучения алгоритмов при изучении курса программирования в ВУЗе.

Во-первых, сортировка – это общая задача многих компьютерных приложений. Практически любой список ценнее, когда он отсортирован по какому-либо определенному принципу.

Во-вторых, многие алгоритмы сортировки являются интересными примерами программирования, демонстрирующих важные методы: частное упорядочивание, рекурсия, объединение списков, сохранение двоичных деревьев в массивах.

«Методы сортировки служат великолепной иллюстрацией базовых концепций анализа алгоритмов, т.е. оценки качества алгоритмов, что в свою очередь, позволяет разумно делать выбор среди, казалось бы, равноценных методов»[1].

Сортировка – одна из немногих задач с точными теоретическими границами производительности. Любой алгоритм сортировки, который использует сравнения, занимает по крайней мере, O(N*logN) времени.

Сортировка выбором

Сортировка выбором – это простой алгоритм $O(N^2)$. Его задача – искать наименьший элемент, который затем меняется местами с элементом из начала списка. Затем находится из оставшихся элементов и меняется местами со вторым элементом. Процесс продолжается до тех пор, пока все элементы не займут свое конечное положение.

2 4 5 3 10 8 1

a_1 a_2 a_3 a_4 a_5 a_6 a_7

1. Определяется наименьший элемент a_7=1
2. Меняется местами с первым 1 4 5 3 10 8 2
3. В оставшемся снова ищется минимальный и меняется со вторым 1 2 5 3 10 8
4. и т. д.

1 2 3 5 10 8 4
1 2 3 4 10 8 5
1 2 3 4 5 8 10
1 2 3 4 5 8 10

Всего проходов n-1.

For i = 1 To n - 1
Min = a(i)
k_min = i
For j = i + 1 To n
If a(j) < Min Then
Min = a(j)
k_min = j
End If
Next
a(k_min) = a(i)
a(i) = Min
Next
	Для реализации этого алгоритма необходимы вложенные циклы For. Внешний цикл (по I) предназначен для последовательного фиксирования элементов массива, внутренний (по J) - осуществляет поиск минимума (максимума) и его позиции. После выхода из внутреннего цикла следует перестановка элементов. Последний элемент во внешнем цикле не рассматривается: он сам встанет на свое место.

При поиске i-го наименьшего элемента алгоритм должен проверить каждый из N-I оставшихся. Время выполнения алгоритма равно N+(N-1)+(N-2)+…+1 или $O(N^2)$.

Сортировка выбором работает достаточно хорошо со списками, где элементы расположены случайно или в прямом порядке, но для обратно сортированных списков производительность этого алгоритма немного хуже. Для поиска минимального элемента списка сортировка выбором выполняет следующую последовательность операторов:

```
If a(j) < Min Then
 Min = a(j)
 k_min = j
 End If
```

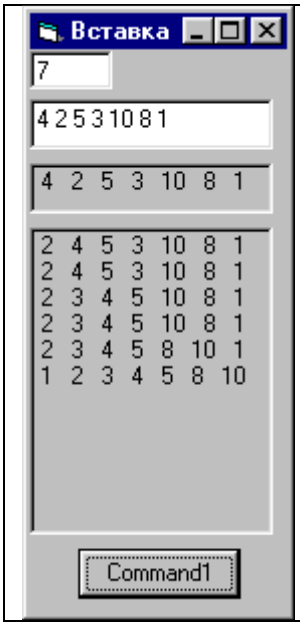
Если список отсортирован в обратном порядке, условие a(j) < Min выполняется большую часть времени. Во время первого прохода через список элементов, оно будет истинно для всех элементов, потому что каждый элемент меньше, чем предыдущий. Программа выполняет сравнение много раз, что приводит к некоторому замедлению работы алгоритма [2].

Это не самый быстрый алгоритм, но он очень прост, а также быстро сортирует небольшие списки.

Сортировка вставкой

Сортировка вставкой – еще один алгоритм сложности $O(N^2)$. Основан на внедрении в отсортированную часть массива элемента, следующего за этой частью, если он удовлетворяет условию сортировки. Алгоритм просматривает исходный список в порядке возрастания и ищет место, где необходимо вставить новый элемент. Затем он помещает новый элемент в найденную позицию.

На первом проходе второй элемент сравнивается с первым, на втором – третий элемент сравнивается с первым и вторым и т.д. Если проверяемый i+1-ый элемент удовлетворяет условию сортировки среди i элементов, то он вставляется на j-е место без нарушения порядка, т.е. элементы с индексами >=j и <=i-1 увеличивают свой индекс на 1.



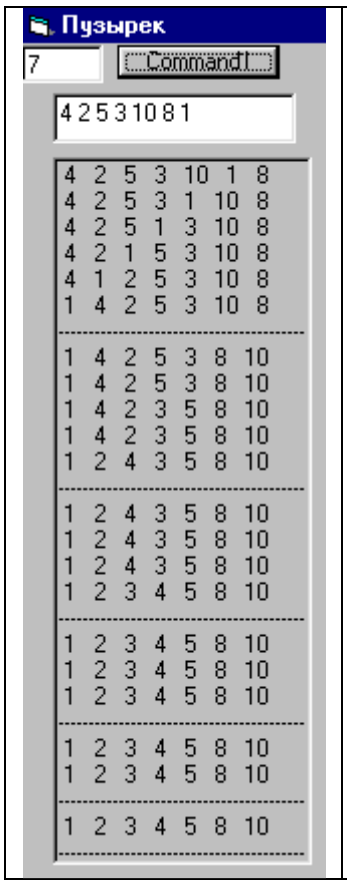

```
For i = 2 To n   'сравнение всегда начинается
 b = a(i): j = 1 'с первого
 Do While b > a(j) 'определяется номер
  j = j + 1        'для вставки
 Loop
  For k = i To j + 1 Step –1 'освобождается
   a(k) = a(k - 1)           'место для вставки
  Next k
  a(j) = b                   'осуществляется вставка
 For l = 1 To n
  Picture2.Print a(l); " ";
 Next
 Picture2.Print
Next I
End Sub
Всего проходов n-1.
```

Подобный алгоритм много времени тратит на поиск правильной позиции для нового элемента. Работает медленнее, чем сортировка выбором.

Пузырьковая сортировка

Пузырьковая сортировка – это алгоритм, предназначенный для сортировки списков, которые уже находятся в почти упорядоченном состоянии. Если это условие выполнено, то алгоритм выполняется очень быстро, за время порядка O(N). Если элементы изначально расположены в произвольном порядке, алгоритм выполняется за $O(N^2)$ шагов.

При пузырьковой сортировке список просматривается до тех пор, пока не найдутся два смежных элемента, которые следуют не по порядку. Они меняются местами, и список продолжает исследоваться дальше. После первого прохода на первом месте (по возрастанию) окажется самый маленький элемент. Следующий проход будет осуществляться до этого элемента, т.е. сортируется оставшаяся часть массива. Алгоритм повторяет этот процесс, пока не упорядочит все элементы.

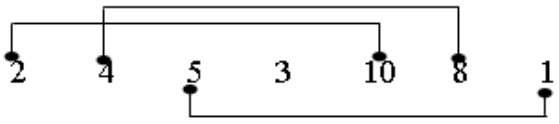



```
Dim a(1 To 10)
Private Sub Command1_Click()
n = Val(Text2.Text)
av = Split(Text1.Text)
For i = 1 To n
a(i) = Val(av(i - 1))
Next
For i = 1 To n - 1           'Цикл по числу проходов
For j = n To i + 1 Step -1   'Цикл сравнения элементов
If a(j - 1) > a(j) Then      'в оставшейся части массива
p = a(j - 1)                 'и продвижение вперед
a(j - 1) = a(j)              '"легких" элементов
a(j) = p
End If
For l = 1 To n
Picture2.Print a(l); " ";
Next
Picture2.Print
Next
Picture2.Print "-----------"
Next i
End Sub
```

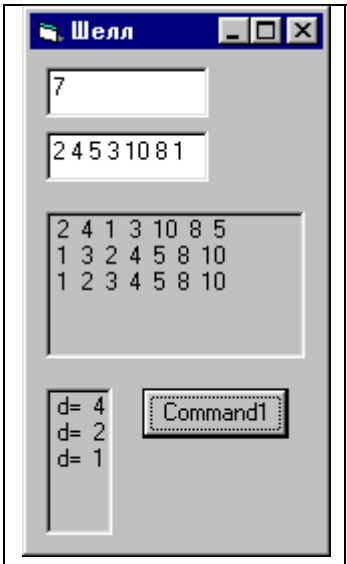
Сортировка методом Шелла

Суть сортировки:

1. N элементов массива разбивается на K групп, причем в группе не более 2 элементов, элементы располагаются на расстоянии D друг от друга. D – шаг группы.
2. Производится сортировка в группах.
3. Шаг группы D уменьшается вдвое.
4. Повторяются пункты 2 – 3 до тех пор, пока шаг не станет меньше 1.



Начальное значение шага D можно задать, как степень числа 2, таким образом, что $D \geq N/2$. При больших значениях N время сортировки методом Шелла значительно меньше, чем в методе пузырька.




```
Dim a(1 To 20)
Private Sub Command1_Click()
n = Val(Text1)
av = Split(Text2, " ")
For i = 1 To n
a(i) = Val(av(i - 1))
Next
d = 1
Do While n \ 2 > d          'Определение шага группы
 d = 2 * d
Loop
Do While d <> 0
 Picture2.Print " d= "; d
 For i = 1 To n - d         'Цикл по элементам группы
  If a(i) > a(i + d) Then 'Сравнение элементов группы
    buf = a(i)              'Перестановка элементов группы
    a(i) = a(i + d)
    a(i + d) = buf
  End If
 Next
 d = d \ 2                  'Уменьшение шага группы
 For k = 1 To n
  Picture1.Print a(k);
 Next
 Picture1.Print
Loop
```

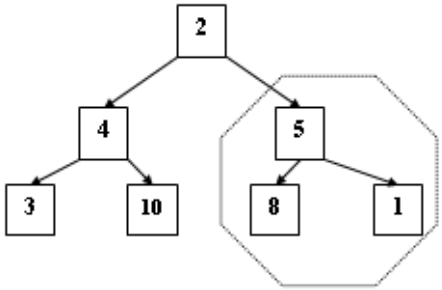
```
d = 1
Do While n > d
d = 2 * d
Loop
d = Int((d - 1) / 2)
Do While d <> 0
Picture2.Print " d= "; d
For i = 1 To n - d
  For j = i To 1 Step -d
  l = j + d
  If a(l) <= a(j) Then
   buf = a(j)
    a(j) = a(l)
    a(l) = buf
  End If
  Next
  Picture1.Print a(i);
Next
Picture1.Print
d = Int((d - 1) / 2)
Loop
For i = 1 To n
Picture1.Print a(i);
Next
```

Пирамидальная сортировка (бинарные деревья)

Пирамида – полное двоичное дерево, в котором каждый узел больше, чем его два дочерних. Оба дочерних узла, должны быть меньше, чем родительский, но любой из них может быть больше другого. Корневой узел всегда самый большой в пирамиде [3].

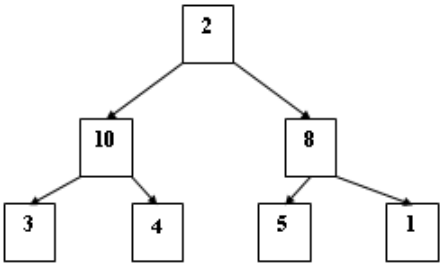
Любой массив можно представить в виде пирамиды, поместив первый элемент массива в корень пирамиды.

Например, 2 4 5 3 10 8 1

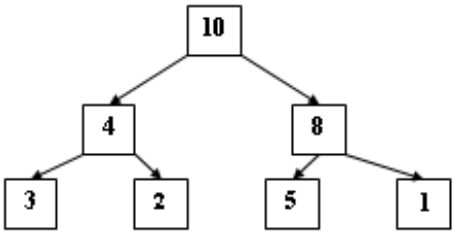


Поддерево, начинающееся в любом узле пирамиды, тоже является пирамидой.

Используя этот факт, можно построить пирамиду снизу вверх. Из каждого из поддеревьев с тремя узлами нужно сформировать пирамиду. Для этого нужно сравнить верхний узел с двумя его дочерними. Если любой из дочерних узлов больше, нужно поменять его с верхним узлом. Если оба дочерних узла больше, нужно поменять с родительский с большим из дочерних. Этот шаг повторяется до тех пор, пока все поддеревья не станут пирамидами:



Затем, создаются более крупные пирамиды: маленькие пирамиды с вершинами 10 и 8 объединяются с элементом 2.



В итоге на самой вершине пирамиды оказывается максимальный элемент последовательности. Выкидываем его из пирамиды, а на его место ставим крайний справа элемент на нижнем уровне. И снова строим пирамиду. На вершине получаем следующий наибольший элемент и т. д.

Подводя итог, можно сказать, что у каждого алгоритма сортировки есть свои преимущества и недостатки. Производительность различных алгоритмов зависит от типа данных, начального расположения, размера и значений. Важно выбрать тот алгоритм, который лучше всего подходит для решения конкретной задачи.

Литература:

1. Д. Э. Кнут. Искусство программирования для ЭВМ, Т3 М.: 1976 г.
2. Евангелос Петрусос. Visual Basic 6. Руководство разработчика: В 2 т.: Пер. с анг. – К.: Издательская группа BHV, 2000. Т.1. -576 с., ил.
3. Евангелос Петрусос. Visual Basic 6. Руководство разработчика: В 2 т.: Пер. с анг. – К.: Издательская группа BHV, 2000. Т.2. -560с., ил.