Алгоритмы сортировки

Сортировка пузырьком

```
bubble_sort(a, n):
      for i = 1..n:
            for j = 1..n - i - 1:
                  if a[j] > a[j + 1]:
                        swap(a[j], a[j + 1])
```

Усовершенствованная версия (ранний выход при отсортированном массиве):

```
bubble_sort_optimized(a, n):
      i = 1
      swapped = true
      while (swapped)
            swapped = false
            for j = 1..n - i - 1:
                  if a[j] > a[j + 1]:
                        swap(a[j], a[j + 1])
                        swapped = true
            i++
```

Сортировка вставками

```
insertion_sort(a, n):
      for i = 2..n:
            j = i
            while (j > 1) && (a[j] < a[j - 1]):
                  swap(a[j], a[j - 1])
                  j--
```

Сортировка выбором

```
selection_sort(a, n):
      for i = 1..n:
            m = index_of_min(a, i, n)
            if i != m:
                  swap(a[i], a[m])

index_of_min(a, i, n):
      min = i
      for k = i + 1..n:
            if a[k] < a[min]:
                  min = k
      return min
```

“Разделяй и властвуй”

Метод разработки алгоритмов “разделяй и властвуй” (метод декомпозиции, метод разбиения) предполагает такое разбиение (декомпозицию) задачи размера n на более мелкие подзадачи, что на основе решений этих подзадач можно получить решение всей задачи.

Обычно при применении метода декомпозиции решение задачи производится рекурсивно, каждый уровень рекурсии состоит из трех шагов:

1. **Разделение** задачи на несколько меньших подзадач (чаще всего экземпляры той же задачи)
2. **Решение** подзадач (“властвование”) с помощью рекурсии (повторного применения метода декомпозиции). Если размеры подзадач достаточно малы, такие подзадачи могут решаться непосредственно без применения рекурсии
3. **Комбинирование** решение подзадач в решение исходной задачи

Примеры применения метода “разделяй и властвуй” для разработки алгоритмов сортировки - сортировка слиянием и быстрая сортировка.

Сортировка слиянием

```
merge_sort(a, begin, end):
      if begin < end:
            middle = (begin + end)/2
            merge_sort(a, begin, middle)
            merge_sort(a, middle + 1, end)
            merge(a, begin, middle, end)

merge(a, begin, middle, end):
      a1 = a[begin..middle] + [INFINITY]
      a2 = a[middle + 1..end] + [INFINITY]
      i = 1, j = 1
      for k = begin..end:
            if a1[i] < a2[j]:
                  a[k] = a1[i]
                  i++
            else:
                  a[k] = a2[j]
                  j++
```

В процедуре merge используется специальное значение `INFINITY` - максимальное значение для сортируемого типа данных - для корректной обработки завершения одного из объединяемых подмассивов.

Быстрая сортировка

```
quick_sort(a, begin, end):
      if begin < end:
            p = partition(a, begin, end)
            quick_sort(a, begin, p - 1)
            quick_sort(a, p, end)

partition(a, begin, end):
      pivot = find_pivot(a, begin, end)
      i = begin, j = end
      do:
            swap(a[i], a[j])
            while a[i] < pivot:
                  i++
            while a[j] >= pivot:
                  j--
      while i <= j
      return i
```

В качестве опорного элемента (`find_pivot`) может браться первый, последний, средний и т.п. элемент подмассива.

Альтернативная реализация быстрой сортировки:

```
quick_sort(a, begin, end):
      if begin < end:
            p = partition(a, begin, end)
            quick_sort(a, begin, p - 1)
            quick_sort(a, p + 1, end)

partition(a, begin, end):
      x = a[end]
      i = begin - 1
      for j = begin..end - 1:
            if a[j] <= x:
                  i++
                  swap(a[i], a[j])
      swap(a[i + 1], a[end])
      return i + 1
```

Здесь в качестве опорного берется последний элемент подмассива.

Сортировки за линейное время

Считаем, что сортируемые элементы сравниваются по ключу, где ключ - значение простого или составного типа данных (число, строка, комбинация строк и/или чисел и т.д.). На множестве ключей определено отношение порядка (сравнения), элемент с меньшим ключом считается меньше элемента элемента с большим ключом.

Если известно множество всех возможных значений ключа и оно конечно, сортировку можно ускорить за счет создания массива для помещения результата (размером равным размеру множества значений ключа) и использования ключей сортируемых элементов для индексации в этом массиве.

Карманная сортировка

Если известно, что все возможные значения ключа встречаются в сортируемой последовательности ровно один раз:

```
bucket_sort_unique(a, n):
      r = array of n elements
      for i = 1..n:
            r[index(a[i].key)] = a[i]
      return r
```

Здесь `index` - функция, осуществляющая отображение множества значений ключа на множество индексов массива (обычно путем масштабирования, сдвига и т.п.). Примеры функций отображения:

- `index(k): return k` - ключ используется как индекс
- `index(k): return k - 200` - ключ больше 200
- `index(k): return k[1]*4 + k[2]` - ключ состоит из двух чисел, каждое число находится в диапазоне [0..3]

В общем случае в сортируемой последовательности значения ключей могут повторяться или присутствовать не полностью. Тогда множество возможных значений ключа разбивается на поддиапазоны (“карманы”), создается массив карманов и сортируемые элементы помещаются по значению ключа в соответствующие карманы. Каждый карман обычно реализуется списком. Финальный результат получается путем объединения этих списков по порядку возрастания ключей.

В случае, если число карманов равно размеру множества возможных значений ключа (в каждом кармане содержатся элементы с одинаковыми ключами):

```
bucket_sort(a, n, m):
      temp = array of m lists
      result = empty array or list
      for i = 1..n:
            temp[index(a[i].key)].push(a[i])
      for k = 1..m:
```

```
            result.append(temp[k])
      return result
```

В случае, если число карманов меньше числа возможных значений ключа (каждый карман может содержать элементы с разными значениями ключа), при построении финального результата требуется произвести дополнительную сортировку элементов в каждом кармане. Для этого в предудущем алгоритме строку `result.append(temp[k])` нужно заменить на `result.append(sort(temp[k]))` .

Сортировка подсчетом

m - число возможных значений ключа. Ключи могут повторяться.

```
counting_sort(a, n, m):
      b = array of n elements
      c = array of m elements
      for i=1..m:
            c[i] = 0
      for i = 1..n:
            c[index(a[i].key)]++
      for i = 2..m:
            c[i] = c[i] + c[i - 1]
      for i = n downto 1:
            ind = index(a[i].key)
            if c[ind] > 0:
                  b[c[ind]] = a[i]
                  c[ind]--
      return b
```

Сортировка подсчетом является стабильной, что дает возможность использовать ее как основу для порязрядной сортировки.

Поразрядная сортировка

Ключ состоит из r полей k_1, k_2, …, k_r.

```
radix_sort(a, n, m, r):
      for i = 1..r:
            counting sort of a by key k i
```

Альтернативная версия: карманная сортировка a по первому полю ключа, затем при сборке результата вызов карманной сортировки для каждого кармана по второму полю ключа и т.д:

```
radix_sort(a, n, m, key_num):
      temp = array of m lists
      result = empty array or list
```

```
for i = 1..n:
      temp[index(a[i].key[key_num])].push(a[i])
if key_num < r:
      next_key = key_num + 1
      next_m = num of key values for key[next_key]
      for k = 1..m:
            len = length(temp[k])
            srtd = radix_sort(temp[k], len, next_m, next_key)
            result.append(srtd)
else:
      for k = 1..m:
            result.append(temp[k])
return result
```

Домашнее задание

Требуется реализовать указанные в задании алгоритмы сортировки (использовать язык программирования по выбору) для сортировки записей из входного файла по указанному ключу. Для тестов использовать файлы из вспомогательных материалов (см. следующий пункт).

Формат записи во входном файле (по одной записи на строку): `Имя Фамилия Год рождения Месяц рождения День рождения`. `Имя`, `фамилия` - строка, `год рождения` - целое число от 1900 до 2000, `месяц рождения` - строка из множества `Jan`, `Feb`, `Mar`, `Apr`, `May`, `June`, `July`, `Aug`, `Sept`, `Oct`, `Nov`, `Dec` (сокращенное название месяца), `день рождения` - целое число от 1 до 31.

Пример:
```
John Smith 1980 July 4
Robert Paulson 1964 Aug 12
```

Если ключ сортировки состоит из нескольких полей (сложный ключ), то в результате записи должны быть отсортированы по первому полю ключа, записи с совпадающим первым полем ключа должны быть в свою очередь отсортированы по второму полю и т.д. Сортировка по месяцу рождения подразумевает сортировку по номеру месяца.

Пример:
Ключ: фамилия, день рождения (ключ состоит из двух полей).

Вход:
John Smith 1980 July 4
Adam Smith 1954 Sep 1

Henry Rodger 1988 Jan 31

Результат (сначала сортировка по имени, затем по дню рождения):
Henry Rodger 1988 Jan 31
Adam Smith 1954 Sep 1
John Smith 1980 July 4

Для каждого алгоритма сортировки из задания построить график/таблицу зависимости времени выполнения от числа записей в файле. Сравнить результаты времени работы алгоритмов между собой, а также со временем работы стандартной функции сортировки в языке (std::sort в C++). Для проверки корректности сортировки выводить отсортированную последовательность в файл.

В отчет включить:

- условие задачи
- описание использованных алгортмов (псевдокод, оценки временной сложности)
- результаты тестов (графики/таблицы)

Код программы в отчет включать не обязательно. Требуется показать работающую программу и ее исходный код (а также знание исходного кода) на занятии.

Вспомогательные материалы

Вспомогательные материалы можно использовать как основу для выполнения задания. Они находятся в репозитории https://gitlab.com/georgy-schukin/mpiaa, директория sorting.

В файлах .h и .cpp находится исходный код тестового примера на языке C++. При реализации своей сортировки достаточно изменить код функции doSort в файле sort.cpp.

Для генерации входных файлов нужно запустить скрипт inputgen.py (должен быть установлен язык программирования Python). Запуск скрипта в Linux, в командной строке (в директории со скриптом): `./inputgen.py` или `python inputgen.py`, в Windows: `python.exe inputgen.py` или двойной щелчок на файле скрипта.

Тестовый пример принимает на вход два аргумента: имя входного файла с записями и имя выходного файла для сохранения записей в отсортированном порядке. При отсутствии аргументов будут использованые значения по умолчанию. Примеры запуска (sort.exe - имя исполняемого файла):

- `sort.exe` - будут использованы входной и выходной файлы по умолчанию
- `sort.exe input_1e5.txt` - указан входной файл, выходной файл по умолчнию
- `sort.exe input_1e4.txt my_output.txt` - указаны входной и выходной файлы

Варианты заданий

Номер варианта для выполнения = свой номер в списке группы % количество вариантов

1. Сортировка пузырьком, сортировка слиянием. Ключ: имя, фамилия, год рождения.
2. Сортировка вставками, быстрая сортировка. Ключ: имя, фамилия, месяц рождения.
3. Сортировка выбором, сортировка слиянием. Ключ: фамилия, год рождения, день рождения.
4. Сортировка пузырьком, быстрая сортировка. Ключ: фамилия, год рождения, месяц рождения.
5. Сортировка вставками, сортировка слиянием. Ключ: имя, фамилия, день рождения.
6. Сортировка выбором, быстрая сортировка. Ключ: имя, год рождения, месяц рождения.
7. Сортировка слиянием, карманная сортировка. Ключ: год рождения, месяц рождения.
8. Быстрая сортировка, карманная сортировка. Ключ: месяц рождения, день рождения.
9. Сортировка слиянием, поразрядная сортировка. Ключ: год рождения, день рождения.
10. Быстрая сортировка, поразрядная сортировка. Ключ: год рождения, месяц рождения.
11. Сортировка пузырьком, карманная сортировка. Ключ: месяц рождения, день рождения.
12. Сортировка вставками, карманная сортировка. Ключ: имя.
13. Сортировка выбором, карманная сортировка. Ключ: фамилия.
14. Сортировка пузырьком, поразрядная сортировка. Ключ: месяц рождения, день рождения.
15. Сортировка вставками, порязрядная сортировка. Ключ: год рождения, день рождения.
16. Сортировка выбором, поразрядная сортировка. Ключ: год рождения, месяц рождения.
17. Порязрядная сортировка (обе версии). Ключ: год рождения, месяц рождения, день рождения.
18. Сортировка слиянием, сортировка подсчетом. Ключ: год рождения.
19. Быстрая сортировка, карманная сортировка. Ключ: день рождения, имя.
20. Сортировка слиянием, карманная сортировка. Ключ: месяц рождения, фамилия.
21. Быстрая сортировка, сортировка подсчетом. Ключ: год рождения, месяц рождения.
22. Сортировка выбором, сортировка подсчетом. Ключ: месяц рождения, день рождения.

Контрольные вопросы

1. Пузырьковая сортировка.
2. Сортировка вставками.
3. Сортировка выбором.
4. Метод “разделяй и властвуй”.
5. Сортировка слиянием.
6. Быстрая сортировка.
7. Карманная сортировка.
8. Поразрядная сортировка.

Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: Построение и анализ, 3-е издание:
    - Глава 2, Приступаем к изучению
    - Глава 7, Быстрая сортировка
    - Глава 8, Сортировка за линейное время
2. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы:
    - Глава 8, Сортировка
3. Седжвик Р. Фундаментальные алгоритмы на C++:
    - Часть 3, Сортировка, главы 6-10