ПРАКТИЧЕСКАЯ РАБОТА №1

ЭМПИРИЧЕСКИЙ АНАЛИЗ СЛОЖНОСТИ АЛГОРИТМОВ

Часть 1.1: ПРАКТИЧЕСКИЙ МЕТОД ОЦЕНКИ СЛОЖНОСТИ

Цель: актуализация знаний и приобретение практических умений и навыков по определению вычислительной сложности алгоритмов (эмпирический подход).

Задание 1. Определить эффективный алгоритм из двух предложенных, используя оценку практической сложности каждого из алгоритмов и их ёмкостную сложность. Доказать корректность алгоритма на основе инварианта цикла.

Пусть имеется задача: дан массив из n элементов символьного типа (С-строка), удалить из массива все значения (символы), равные заданному пользователем (ключевому).

Примечание: удаление состоит в уменьшении размера массива («эффективного» размера). Удаление осуществляется путем сдвига элементов массива к его началу с сохранением порядка следования (как до удаляемого, так и следующих после удаляемого).

Например, нужно удалить из массива все значения, равные '_' (т.е. **key** = '_').
Исходный массив: **n** = 12; **x**(n) = {'_', 'M', 'I', '_', '_', 'R', '_', '_', 'E', 'A', '_', '_'}.
Результат: n=5; x(n) = {'M', 'I', 'R', 'E', 'A'}.
Два алгоритма решения этой задачи:

x-массив, n – количество элементов в массиве, key – удаляемое значение

Алгоритм 1.

```
delFirstMetod(x,n,key) {
i←1
while (i<=n) do
        if x[i]=key then
                //удаление
                 for j←i to n-1 do
                 x[j] ←x[j+1]
               od
               n←n-1
      else
             i←i+1
     endif
od
}
```

Алгоритм 2

```
delOtherMetod(x,n,key) {
j←1
for i←1 to n do
  x[j]=x[i];
  if x[i]!=key then
     j++
  endif
od
if x[1]=key then
  n←0
else
  n←j
endif
}
```

Требования к выполнению задания 1:

1. Для каждого алгоритма отдельно привести в отчёте следующие результаты:
   1.1. Мат. модель решения поставленной задачи (блок-схему алгоритма).
   1.2. Реализовать алгоритмы в виде функций на языке C++ и отладить на динамическом символьном массиве **new/delete** небольшого размера для ситуаций лучшего, среднего и худшего случаев.
   1.3. Реализовать практический подход в оценке сложности алгоритма, включив в код операторы, подсчитывающие число выполненных сравнений ($C_п$) + перемещений элементов в памяти ($M_п$), а также суммарное число критических операций $T_п = C_п + M_п$.
   1.4. Реализовать дополнительные возможности:
      a. заполнение массива:
         - датчиком случайных чисел в заданном диапазоне,
         - заданным значением (например, ключевым);
      b. вывод массива на экран;
      c. вывод значений $T_п$, $C_п$, $M_п$, а также времени выполнения алгоритма в миллисекундах.
   1.5. Протестировать алгоритм при n = 100, 200, 500, 1000, 2000, 5000, 10000 в трёх случаях: а) все элементы должны быть удалены, б) случайное заполнение и в) ни один элемент не удаляется.
   Какая ситуация будет лучшим, средним и худшим случаем в работа алгоритма?

Примечание: для обеспечения равных условий при сравнении работы двух алгоритмов в среднем случае (случайное заполнение) следует предусмотреть, чтобы они обрабатывали один и тот же входной массив.

   1.6. Представить результаты эмпирического исследования в таблице (по примеру табл. 2), указав для каждого n время работы алгоритмов в мс и практически полученное количество критических операций при выполнении алгоритма (по счётчикам).
   1.7. Определить ёмкостную сложность алгоритмов в количестве ячеек памяти, необходимых сверх памяти под входной массив.
   1.8. По данным п. 1.6 построить на одном графике зависимость времени решения задачи (в мс) для трёх случаев, описанных в п.1.5. Предположите визуально характер зависимости (линейный, квадратичный, логарифмический, экспоненциальный)[1].

[1] Для построения графика можно использовать любое вспомогательное ПО.

График должен быть подписан, быть наглядным, с подписанными и размеченными координатными осями[2].

2. По данным п.1.6 постройте на одном сравнительном графике зависимость $T_п(n)$ сразу для обоих алгоритмов (только для случая, когда все элементы – ключевые). Сделайте вывод, какой алгоритм эффективнее в этом случае.
3. Повторите п.2 отдельно для ситуаций а) со случайным заполнением и, б) когда ни один элемент не удаляется.
4. Предложите способ как привести длину массива после удаления ключевых элементов к «эффективной» (для экономного расходования памяти).
5. Сделайте в отчёте вывод – какой из двух алгоритмов в каком случае эффективнее по времени. Какой эффективнее по расходу памяти?

Задание 2. Оценить сложность алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай).

1. Составить функцию простой сортировки одномерного целочисленного массива A[n], используя алгоритм согласно варианту индивидуального задания (столбец «*Алгоритм заданий 2 и 3»* в табл. 1*)*. Провести тестирование программы на исходном массиве n=10 случайных значений.

Таблица 1. Варианты индивидуальных заданий

№	Алгоритм заданий 2 и 3	Алгоритм задания 4
1	Простой вставки (***Insertion sort***)	Простого выбора ***(Selection sort)***
2	Простого обмена («пузырек», ***Exchange sort***)	Простой вставки (***Insertion sort***)
3	Простого выбора ***(Selection sort)***	Простого обмена («пузырек», ***Exchange sort***)

2. Провести контрольные прогоны программы массивов случайных чисел при n = 100, 200, 500, 1000, 2000, 5000, 10000, 100000, 200000, 500000 и 1000000 элементов с вычислением количества сравнений, перемещений, суммарного числа критических операций и времени выполнения (в миллисекундах/секундах). Полученные результаты свести в сводную табл. 2.
3. Провести эмпирическую оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества критических операций $T_п$ как сумму сравнений $C_п$ и перемещений $M_п$. Полученные результаты вставить в сводную табл. 2.
4. Построить графики для этого алгоритма аналогично п.1.8 в задании 1.
5. Определить ёмкостную сложность алгоритма.
6. Сделать вывод об эмпирической вычислительной сложности алгоритма в среднем случае.

[2] Здесь и в последующих заданиях с графиками настраивайте равномерный масштаб осей, а не логарифмический.

Таблица 2. Сводная таблица результатов

n	**время, мс**	**$C_п$**	**$M_п$**	**$T_п=C_п+M_п$**
100				
200				
500				
1000				
2000				
5000				
10000				
100000				
200000				
500000				
1000000				

Задание 3. Оценить вычислительную сложность алгоритма простой сортировки в наихудшем и наилучшем случаях.

1. Провести дополнительные прогоны программы на массивах при n = 100, 200, 500, 1000, 2000, 5000, 10000, 100000, 200000, 500000 и 1000000 элементов, отсортированных:
   a. строго в убывающем порядке значений, результаты представить в сводной таблице по формату табл. 2;
   b. строго в возрастающем порядке значений, результаты представить в сводной таблице по формату табл. 2.
2. Сделать вывод о зависимости (или независимости) алгоритма сортировки от исходной упорядоченности массива.

Задание 4. Сравнить эффективность алгоритмов простых сортировок

1. Выполнить разработку и программную реализацию второго алгоритма согласно индивидуальному варианту в столбце *«Алгоритм задания 4»* из табл. 1.
2. Аналогично заданиям 2 и 3 сформировать таблицы с результатами эмпирического исследования второго алгоритма в среднем, лучшем и худшем случаях в соответствии с форматом табл. 2 (на тех же массивах, что и в заданиях 2 и 3).
3. Определить ёмкостную сложность алгоритма от n.
4. На одном сравнительном графике отобразить зависимости $T_п(n)$ работы двух алгоритмов сортировки в худшем случае.
5. Аналогично на другом общем графике отобразить зависимости для лучшего случая.

6. Сделать вывод о том, какой из алгоритмов простой сортировки эффективнее на основе данных эмпирического анализа.

По результатам выполнения всех заданий сформируйте отчёт, сохраните его в формате pdf и загрузите в систему СДО.

ЛИТЕРАТУРА:

1. Бхаргава А. Грокаем алгоритмы, 2-е изд. – СПб: Питер, 2024. – 352 с.
2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.
3. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.
4. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.
5. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 05.02.2025).