|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика**

**О Т Ч Е Т**

по лабораторной работе №1

Название: Исследование структур и методов обработки данных

Дисциплина: Технология разработки программных систем

**Вариант 7**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-45Б |  |  | В. И. Мамыкин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Е. К. Пугачёв |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2024 г.*

1. **Цель лабораторной работы**

Определить основные критерии оценки структуры данных и методов ее обработки применительно к конкретной задаче.

1. **Описание задания**
   1. **Задание**

Дана таблица материальных нормативов, состоящая из K записей

фиксированной длины вида: код детали; код материала; единица измерения; номер цеха; норма расхода.

* 1. **Основные требования**

Основной вариант задания включает в себя следующие требования:

* Структура данных ­– таблица;
* Поиск – метод дихотомии (двоичный поиск);
* Упорядочение – вставка;
* Корректировка – удаление сдвигом.

1. **Описание основного варианта задания**
   1. **Структура данных**

Структуру данных по заданию (таблицу) предлагается реализовать с по­мощью одномерного массива записей, структура которого представлена на рисунке 1.

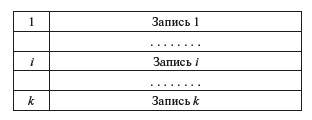


Рисунок 1 – Структура массива записей

Ниже представлена диаграмма статического массива и его элементов:

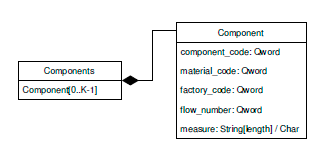


Рисунок 2 – Диаграмма статического массива и его элементов

Структура Component содержит следующие поля:

* component\_code – код детали;
* material\_code – код материала;
* factory\_code – номер цеха;
* flow\_number – норма расхода;
* measure – единица измерения.

Для первых четырех полей используется тип данных Qword – беззнаковый формат, занимает 8 байт.

Для поля «единица измерения» используется тип String, длиной 255 символов, занимает 256 байт.

* + 1. **Реализация структуры на языке Pascal**

|  |
| --- |
| type Component = record  component\_code: QWord; // код детали  material\_code: QWord; // код материала  factory\_code: QWord; // номер цеха  flow\_number: QWord; // норма расхода  measure: String[255]; // единица измерения  end;  type Components = Array [0 .. K-1] of Component; |

Переопределенные методы (структура Component) для сравнения по коду детали:

|  |
| --- |
| // Оператор сравнения “=” по коду детали  operator < ( a, b: Component ) R: Boolean;  begin  R := a.component\_code < b.component\_code;  end;  // Оператор сравнения “>” по коду детали  operator > ( a, b: Component ) R: Boolean;  begin  R := a.component\_code > b.component\_code;  end;  // Оператор сравнения “<” по коду детали  operator = ( a, b: Component ) R: Boolean;  begin  R := a.component\_code = b.component\_code;  end; |

* + 1. **Расчет памяти, занимаемой массивом**

Максимальный объем памяти, занимаемой одной записью:

*VST = V(QWord) \* 4 + V(String[255]) = 8 \* 4 + 256 = 288 Байт.*

Максимальный объем памяти, занимаемой массивом, содержащим K записей:

*Varr = K \* VST = K \* 288 Байт.*

Заметим, что если диапазон значений кода детали, кода материала, номера цеха; нормы расхода находится: в пределах от 0 до 28-1 включительно, то данные поля записи можно определить типом «Byte», который займет в памяти 1 байт на каждое поле; в пределах от 0 до 216-1 включительно, то данные поля записи можно определить типом «Word», который займет в памяти 2 байта на каждое поле; в пределах от 0 до 232-1 включительно, то данные поля записи можно определить типом «LongWord», который займет в памяти 4 байта на каждое поле; в пределах от 0 до 264-1 включительно, то данные поля записи можно определить типом «QWord», который займет в памяти 8 байта на каждое поле.

Поле measure (единица измерения), если его значение записывается одним символом, можно определить типом «Сhar», который займет в памяти 1 байт, если диапазон его значений находится в интервале от 2 до 255 символов, то данное поле записи можно определить типом «String[length]», где length — длина, который занимает в памяти 1 + length Байт.

В любом из возможных случаев определения типа полей записи размер массива линейно зависит от размера его элементов и определяется по формуле Varr = K \* VST, где VST — объем памяти, занимаемой одной записью, K — целочисленная константа, определяющая размер массива записей. Заметим, что объем памяти, который будет выделен операционной системой для хранения переменной типа Components, является постоянным на всём протяжении времени жизни данной переменной и не зависит от количества инициализированных переменных типа Component рассматриваемого массива.

* + 1. **Оценка времени доступа к i-му элементу**

В массиве доступ выполняется по индексу. Соответственно, количество тактов, необходимых для доступа к *i*-му элементу, складывается из количе­ства тактов, необходимых для умножения и для сложения с постоянной:

*Tд = t++ + t→ = 1 + 1 = 2 такта*

* 1. **Анализ метода поиска**

По заданию необходимо реализовать и оценить способ поиска методом дихотомии. Двоичный (бинарный) поиск (или дихотомия) — классический алгоритм поиска элемента в отсортированном массива, использующий дробление массива на половины.

* + 1. **Реализация метода поиска**

|  |
| --- |
| // Двоичный поиск по сортированному массиву  function find ( arr: Components; elem: Component; size: QWord ): Int64;  var m, i, j: QWord;  begin  i := 0;  j := size - 1;  m := (i + j) >> 1;  while (arr[m] <> elem) and (i <= j) do  begin  if elem > arr[m] then  i := m + 1  else  j := m - 1;  m := (i + j) >> 1;  end;  if i > j then  find := -1  else  find := m;  end; |

* + 1. **Среднее количество сравнений**

Среднее количество сравнений для операции поиска равно:

****

* + 1. **Оценка времени поиска**

*T = tуст + tцикла + tif= (t= + t+C + t= + t+ + t****→****) +*

*+ C \* ((ti + t+ + t+) + (ti + t+) + 1 + (0.5 \* (t+C + t=)) + 0.5\*(t+C + t= + t+ + t****→****)) + (t+ + 1 + 0.5 \* (t=) + 0.5 \* (t=)) = (2 + 1 + 2 + 2 + 1) + (C ∗ ((2 + 2 + 2) + (2 + 2) + 1 + (0.5 \* (1 + 2)) + (0.5 \* (1 + 2 + 2 + 1))) + (2 + 1 + 0.5 \* 2 + 0.5 \* 2) = 8 + 16 \* ([(k + 1)log2(k + 1)] / k – 1) + 5 = 13 + 16 \* ([(k + 1)log2(k + 1)] / k – 1)*

* 1. **Анализ метода упорядочивания**

По заданию необходимо реализовать и оценить упорядочение массива методом вставки.

Элемент массива *a*i (начиная со второго) сравнивают по­следовательно с предшествующими *a*j, где *j* = *i* – 1, *i* – 2, …, до тех пор, пока не будет найден элемент со значением, меньшим *a*i. Пусть этот элемент име­ет номер *j*, где *j* < *i*. Тогда все элементы с номерами *j* + 1, …, *i* – 1 сдвигают на одну позицию, а *i*-й элемент ставят на место (*j* + 1)-го элемента. Если все впереди стоящие элементы больше *i*-го, то их сдвигают на одну пози­цию, а *i*-й элемент ставят на первое место.

Общая суть сортировок вставками такова:

1. Перебираются элементы в неотсортированной части массива.
2. Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находиться.
   * 1. **Реализация метода упорядочивания**

|  |
| --- |
| // Сортировка вставками  procedure sort ( var arr: Components; size: QWord );  var  i, j: QWord;  t: Component;  begin  for i := 1 to size - 1 do  begin  j: = i;  while (j > 0) and (arr[j] < arr[j - 1]) do  begin  // counter +=1;  t := arr[j];  arr[j] := arr[j - 1];  arr[j - 1] := t;  j -= 1;  end  end;  end; |
|  |

* + 1. **Оценка количества сравнений**

Наименьшее количество сравнений определяется по формуле:

*C = (K-1) \* 1*

Наибольшее количество сравнений определяется по формуле:

*C = (K-1) \* K/2*

Среднее количество операций сравнения:

*C = (K-1) \* K/4*

* + 1. **Оценка времени упорядочивания**

Наименьшее время работы алгоритма:

*T = tуст + tпр + 1 + (K – 1) \* (tтела + tпр + 2) = 2 + 2 + 1 + (K – 1) \* (tтела + 2 + 2);*

*tтела = t= + (K) \* (t+ + ti + ti + t+C + 1 + t= + ti + ti + t= + t+C + ti + t+C + t= + t++) = 2 + 1 \* (2 + 2 + 2 + 1 + 1 + 2 + 2 + 2 + 2 + 1 + 2 + 1 + 2 + 1) = 2 + K \* 23*

*T = 5 + (K – 1) \* (2 + 1 \* 23 + 4) = 5 + (K – 1) \* (6 + 1 \* 23)*

Наибольшее время работы алгоритма:

*T = 5 + (K – 1) \* (6 + (K / 2) \* 23)*

Среднее время работы алгоритма:

*T = 5 + (K – 1) \* (6 + (K / 4) \* 23)*

* 1. **Анализ метода корректировки**

По заданию необходимо реализовать и оценить удаление сдвигом.

Алгоритм удаления элемента сводится к сдвигу на одну позицию влево всех элементов, расположенных правее удаляемого. При этом нужно сохранить новый размер в переменную, чтобы далее работать уже с новой областью массива.

* + 1. **Реализация метода корректировки**

|  |
| --- |
| // Удаление сдвигом  procedure del ( var arr: Components; index: QWord; var size: QWord );  var  i: QWord;  begin  for i := index to size - 2 do  arr[i] := arr[i+1];  size -= 1;  end; |

* + 1. **Расчет времени удаления элемента**

*T = tуст + tпр + 1 + (K / 2) \* (t+C + ti +ti + t=) = 2 + 2 + 1 + (K / 2) \* 7 = 5 + (K / 2) \* 7*

*(K / 2)* – среднее количество прохождение по массиву (половина массива)

1. **Альтернативный вариант**
   1. **Выбор альтернативной структуры данных**

Для хранения записи Component использован динамический массив Components.

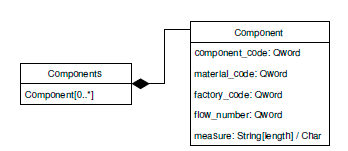


Рисунок 3 – Диаграмма динамического массива и его элементов

* + 1. **Реализация структуры на языке Pascal**

|  |
| --- |
| type Component = record  component\_code: QWord;  material\_code: QWord;  factory\_code: QWord;  flow\_number: QWord;  measure: String[255];  end;  type  Components = Array of Component; |

Переопределенные методы для сравнения по коду детали:

|  |
| --- |
| // Оператор сравнения “=” по коду детали  operator < ( a, b: Component ) R: Boolean;  begin  R := a.component\_code < b.component\_code;  end;  // Оператор сравнения “>” по коду детали  operator > ( a, b: Component ) R: Boolean;  begin  R := a.component\_code > b.component\_code;  end;  // Оператор сравнения “<” по коду детали  operator = ( a, b: Component ) R: Boolean;  begin  R := a.component\_code = b.component\_code;  end; |

* + 1. **Расчет памяти, занимаемой массивом**

Максимальный объем памяти, занимаемой одной записью:

*VST = V(QWord) \* 4 + V(String[255]) = 8 \* 4 + 256 = 288 Байт.*

Максимальный объем памяти, занимаемой массивом, содержащим K записей:

*Varr = N \* VST = N \* 288 Байт.*

Размер памяти, занимаемой одной записью, определяется по тем же правилам, что и в исходном варианте структуры.

Размер массива линейно зависит от размера его элементов и определяется по формуле Varr = N \* VST, где VST — объем памяти, занимаемой одной записью, N — целочисленная переменная, определяющая размер массива записей и способная изменяться в течение времени работы программы. Данный вариант описания структуры данных полезен для таких случаев, когда разработчик не знает на момент компиляции программного кода точное количество элементов массива, которые будут определены пользователем.

* + 1. **Оценка времени доступа к i-му элементу**

В массиве доступ выполняется по индексу. Соответственно, количество тактов, необходимых для доступа к *i*-му элементу, складывается из количе­ства тактов, необходимых для умножения и для сложения с постоянной:

*Tд = t++ + t→ = 1 + 1 = 2 такта*

* 1. **Анализ метода поиска**

Выбран последовательный метод поиска.

При реализации последовательного метода поиска необходимо учиты­вать дополнительные условия: данные не отсортированы, и неизвестно коли­чество элементов, удовлетворяющих ключу поиска; данные не отсортирова­ны, но известно число элементов, удовлетворяющих ключу поиска; данные отсортированы по ключевому полю.

* + 1. **Реализация метода поиска**

|  |
| --- |
| function find(arr: Components; elem: Component): Int64;  var  i: QWord;  len: QWord;  flag: Boolean;  begin  flag:= false;  len := Length(arr) - 1;  for i := 0 to len do  begin  if arr[i] = elem then  begin  find := i;  flag := true;  end;  end;  if not flag then  find := -1;  end; |

* + 1. **Среднее количество сравнений**

Среднее количество сравнений для реализации данного метода вычисляют по формуле:

*С = (N + 1) / 2,*

где *N* — число записей в массиве.

* + 1. **Оценка времени поиска**

*T = t= + 1 + t+ + tуст + tпр + 1 + С \* (tтела + tпр + 2) + tif = 1 + t+ + tуст + tпр + 1 + С \* (t+ + ti + 1 + 0.5 \* (t= + t=)) + t+ + 1 + 0.5 \* t= = 2 + 1 + 2 + 2 + 2 + 1 + C \* (2 + 2 + 1 + 0.5 \* (2 + 2)) + 2 + 1 + 0.5 \* 2 = 10 + C \* 7 + 4 = 14 + C \* 7 = 14 + (N + 1) / 2 \* 7*

* 1. **Анализ метода упорядочивания**

В качестве альтернативного варианта метода сортировки выберу метод сортировки слиянием.

Метод основан на принципе «разделяй и властвуй». Мы берем половину массива, потом делим еще и так пока не останутся подмассивы длиной 1-2 элемента. Потом поэлементно сравниваем и получаем отсортированный массив.

* + 1. **Реализация метода упорядочивания**

|  |
| --- |
| procedure Merge(var A: array of Component; low, mid, high: QWord);  var  B: array of Component;  i, j, k: QWord;  begin  B := nil;  SetLength(B, high - low + 1);  i := low;  j := mid + 1;  k := 0;  while (i <= mid) and (j <= high) do  begin  if A[i] < A[j] then  begin  B[k] := A[i];  Inc(i);  end  else  begin  B[k] := A[j];  Inc(j);  end;  Inc(k);  end;  while i <= mid do  begin  B[k] := A[i];  Inc(i);  Inc(k);  end;  while j <= high do  begin  B[k] := A[j];  Inc(j);  Inc(k);  end;  for k := low to high do  A[k] := B[k - low];  end;  procedure MergeSort(var A: array of Component; low, high: QWord);  var  mid: QWord;  begin  if low < high then  begin  mid := (low + high) div 2;  MergeSort(A, low, mid);  MergeSort(A, mid + 1, high);  Merge(A, low, mid, high);  end;  end; |

* + 1. **Оценка количества сравнений**

*С = N \* log2N*

* + 1. **Оценка времени упорядочивания**

При выполнении слияния в памяти инициализируются два массива. Также при рекурсивном вызове функции разделения образуются фреймы активации, которые остаются в памяти до выхода из рекурсии.

Память, занимаемая массивами:

Память, выделяемая под рекурсию:

В итоге:

* 1. **Анализ метода корректировки**

Буду использовать метод сдвига с другой реализацией. За счет того, что массив динамический – получится освободить место компонента с помощью функции setlegth().

* + 1. **Реализация метода корректировки**

|  |
| --- |
| // Процедура удаления записи из массива  procedure del ( var arr: Components; index: QWord );  var  i: QWord;  len: QWord;  begin  len := high(arr) - 1;  for i := index to len do  arr[i] := arr[i+1];  setlength(arr, length(arr) - 1);  end; |

* + 1. **Расчет времени удаления элемента**

*T = t= + 1 + t+C + tуст + tпр + 1 + (N / 2) \* (ti + ti + t+C + t=) + 2 = 2 + 1 + 1 + 2 + 2 + 1 + (N / 2) \* (2 + 2 + 1 + 2) + 2 = 11 + (N / 2) \* 7*

1. **Таблица результатов**

Таблица 1 – таблица результатов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Варианты | Структура данных | Метод поиска | Метод упорядочения | Метод корректировки |
| Основной | Статический массив  *K \* 288 Байт*  *K – константа*  *(жесткое ограничение)* | Бинарный  C = *\* ([(k + 1)log2(k + 1)] / k – 1)*  *T = 13 + 16 \* ([(k + 1)log2(k + 1)] / k – 1)* | Сортировка вставками  *C = (K-1) \* K/4*  *T = 5 + (K – 1) \* (6 + (K / 4) \* 23)* | Удаление сдвигом  *T = 5 + (K / 2) \* 7* |
| Альтернативный | Динамический массив  *N \* 288 Байт*  *N – изменяемое число (гибкость работы с массивом)* | Последовательный  *С = (N + 1) / 2*  *T = 14 + (N + 1) / 2 \* 7* | Сортировка слиянием  *С = N \* log2N*  *V = 310N* | Удаление сдвигом (другая реализация)  *T = 11 + (N / 2) \* 7* |

Как видно из сравнительной таблицы, к недостаткам использования динамического массива можно отнести большой объем памяти за счет того, что не освобождается, что дополнительно приводит к необходимости хранения текущего размера массива в переменной.

1. **Заключение**

В результате выполнения лабораторной работы были проведены качественные и количественные оценки структур данных и методов их обработки в соответствии с вариантом задания. В альтернативном варианте предложены решения, которые обеспечат более эффективную работу с занимаемой памятью, быструю сортировку элементов и их удаление.