|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 6** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Рекурсивные алгоритмы и их реализация.»** | |
|  | |
| Выполнил студент группы ИКБО-07-22 | Козлов К.И. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

Получить знания и практические навыки по разработке и реализации рекурсивных процессов.

**Задание.**

Разработать и протестировать рекурсивные функции в соответствии с задачами варианта. Составить отчет.

Требования к выполнению первой задачи варианта:

* приведите итерационный алгоритм решения задачи
* реализуйте алгоритм в виде функции и отладьте его
* определите теоретическую сложность алгоритма
* реализуйте и отладьте рекурсивную функцию решения задачи
* определите глубину рекурсии, изменяя исходные данные
* определите сложность рекурсивного алгоритма, используя метод подстановки и дерево рекурсии
* приведите для одного из значений схему рекурсивных вызовов
* разработайте программу, демонстрирующую выполнение обеих функций, и покажите результаты тестирования.

Требования к выполнению второй задачи варианта:

* рекурсивную функцию для обработки списковой структуры согласно варианту. Информационная часть узла – простого типа – целого;
* определите глубину рекурсии
* определите теоретическую сложность алгоритма
* разработайте программу, демонстрирующую работу функций и покажите результаты тестов.

Требования к упражнениям:

Вариант №9. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | 1 Бинарный поиск элемента в массиве  2 Создание двунаправленного списка. |

# **Решение**

В данной практической работе была описана структуры node и list. Эти структуры выполняют функции узла в списке и самого списка – набора узлов соответственно.

Сведения по двунаправленному динамическому списку

Двунаправленный список – это совокупность узлов. Узел в односвязном динамическом списке – это его часть. В узле хранится информация о нем самом (какие-то данные, переменные), информация о следующем узле и о предыдущем узле (указатель на следующий узел. Указатель на следующий узел у последнего объекта равен нулевому указателю, у первого указатель на предыдущий равен нулевому указателю). Добавление элемента происходит в конце списка(push\_back).

В данной задаче нужно было спроектировать и отладить метод рекуррентного ввода двунаправленного списка. Сложность создания двунаправленного линейного списка из n элементов - O(n), так как для каждого элемента выполняется постоянное число элементарных операций.

Рекурсивный алгоритм – алгоритм, в процессе выполнения которого он вызывает сам себя. У рекурсивного алгоритма обязательно должна присутствовать точка выхода. У рекурсивного алгоритма есть прямой и обратный ход. Прямым ходом называется последовательность рекуррентных вызовов алгоритма самого себя, а обратных ходом называется “сборка” от результата последнего вызова рекурсии к самому первому

В данной задаче нужно было спроектировать и отладить алгоритм бинарного поиска как инерционный, так и рекурсивный.

Теоретическая сложность бинарного поиска составляет O(log2n). Это

связано с тем, что на каждой итерации пространство поиска сужается

примерно наполовину. Таким образом, в худшем случае нам потребуется

log2n итераций, чтобы сузить массив от полного размера n до одного

элемента.

Сложность log2n предполагает, что входной массив отсортирован —

бинарный поиск работает только с отсортированными массивами.

Рекуррентное соотношение задается формулой: T(n) = a\*T(n/b)+O(1),

где a>=1, b>1. Для бинарного поиска выполняется только одна из двух

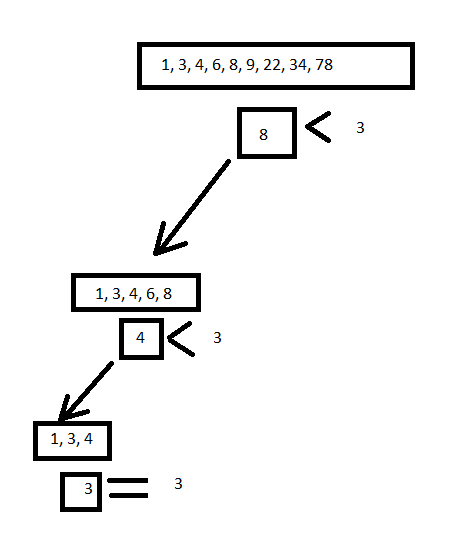
подзадач, т.е. a=1 и b=2. Получаем формулу T(n)=T(n/2)+O(1).

Оценив сложность методом подстановки, так же получаем O(log2n):

T(n)=T(n/2)+O(1)=T(n/4)+O(1)+O(1)=T(n/8)+O(1)+O(1)+O(1)=…

…=T(1)+O(log2n)

Предположим, что задан массив из 9 элементов: 1, 3, 4, 6, 8, 9, 22, 34, 78. Нужно найти позицию элемента со значением 3, тогда схема рекуррентных вызовов будет выглядеть следующим образом

  
Рисунок 1 Схема рекурсивных вызовов функции бинарного поиска

Для решения упражнения были описаны несколько структур и написано несколько функций: структура node, реализующая узел списка.

|  |
| --- |
| struct node  {  char data;  node\* nextNodePtr;  node\* prevNodePtr;  node(char userData)  {  data = userData;  nextNodePtr = nullptr;  prevNodePtr = nullptr;  }  }; |

Также была описана структура list, реализующая однонаправленный динамический список. List имеет методы: Empty() – проверка на пустой список, PushBack – добавление элемента в конец списка, ShowList() – вывод спика на экран.

|  |
| --- |
| struct list  {  node\* head;  node\* tail;  list()  {  head = nullptr;  tail = nullptr;  }  void PushFront(char userDataAdd)  {  node\* add = new node(userDataAdd);  if (head == nullptr)  {  head = add;  return;  }  else if (tail == nullptr)  {  node\* temp = head;  head = add;  head->nextNodePtr = temp;  temp->prevNodePtr = head;  tail = temp;  return;  }  else  {  node\* temp = head;  head = add;  head->nextNodePtr = temp;  temp->prevNodePtr = add;  }  }  void PushBack(char userDataAdd)  {  node\* add = new node(userDataAdd);  if (tail == nullptr)  {  tail = add;  return;  }  else if (head == nullptr)  {  node\* temp = tail;  tail = add;  tail->prevNodePtr = temp;  temp->nextNodePtr = tail;  head = temp;  return;  }  else  {  node\* temp = tail;  tail = add;  tail->prevNodePtr = temp;  temp->nextNodePtr = add;  }  }  void ShowList()  {  node\* temp = head;  while (temp != nullptr)  {  std::cout << temp->data;  temp = temp->nextNodePtr;  }  }  }; |

Функция inputArray() имеет 2 парамер: array – указатель на массив целочисленных значений и его длинна.

Эта функция реализует запись массива пользователем.

|  |
| --- |
| void inputArray(int\* array, int n)  {  for (int i = 0; i < n; i++)  {  std::cin >> array[i];  }  } |

Функция showArray() имеет все параметры функции inputArray().

Функция выводит введенный массив на экран.

|  |
| --- |
| void showArray(int\* array, int n)  {  for (int i = 0; i < n; i++) std::cout << array[i] << ' ';  } |

Функция sortByChoise() имеет такие же параметры как и функция inputArray().

Функция реализует сортировку введенного массива.

|  |
| --- |
| void sortByChoise(int\* array, int n)  {  for (int i = 0; i < n; i++)  {  int min = i;  for (int j = i + 1; j < n; j++)  {  if (array[j] < array[i])  {  min = j;  std::swap(array[j], array[i]);  }  }  }  } |
|  |

Функция rescueBinar() имеет такие же параметры как и функция inputArray().

Функция реализует бинарную сортировку массива, выполнена рекурентно.

|  |
| --- |
| int recurseBinar(int\* array, int left, int rigth, int answer)  {  int mid = (left + rigth) / 2;  if (array[mid] == answer) return(mid);  if (array[mid] > answer) recurseBinar(array, left, mid - 1, answer);  else recurseBinar(array, mid + 1, rigth, answer);  } |

При запуске программы пользователь увидит начальный интерфейс программы



Рисунок 2. Интерфейс программы

# **4. Тестирование**

Протестируем работу программы. Для этого введем 1 – работа с массивом и тест бинарного поиска. Длаее введем 2 и 1. Так мы запишем предложение в виде двусвязного списка добавляя все элементы в конец массива слова в конец списка. Далее слова введем 2, но теперь выберем 2 как режим заполнения двусвязного списка и предложение запишется “наоборот”.

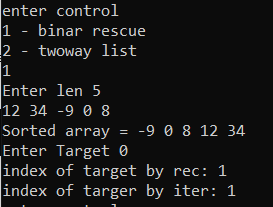


Рисунок 3. Проверка работы программы c функциями бинарного поиска

Вывод и поиск индекса искомого элемента работает корректно как в случае с итерационным алгоритмом, так и в случае с рекурсивным.

Далее проверим работу программы с двунаправленным списком.

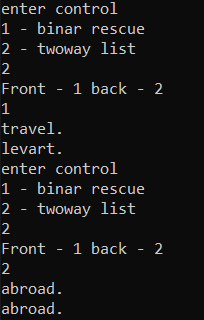


Рисунок 4. Проверка работы программы с двунаправленным списком.

Как видно, корректно проработали записи как с функцией PushFront так и с PushBack

Все функции и методы работают корректно.

# **5. Выводы**

В результате выполнения работы я:

# 1. Научился программировать автоматическое тестирование более

# сложных программ

# 2. Научился реализовывать рекурсивные алгоритмы и познакомился с

# управлением двунаправленного линейного списка.

# **6. Исходный код программы**

|  |
| --- |
| #include <iostream>  int recurseBinar(int\* array, int left, int rigth, int answer)  {  int mid = (left + rigth) / 2;  if (array[mid] == answer) return(mid);  if (left == rigth) return (-1);  if (array[mid] > answer) recurseBinar(array, left, mid - 1, answer);  else recurseBinar(array, mid + 1, rigth, answer);  };  void sortByChoise(int\* array, int n)  {  for (int i = 0; i < n; i++)  {  int min = i;  for (int j = i + 1; j < n; j++)  {  if (array[j] < array[i])  {  min = j;  std::swap(array[j], array[i]);  }  }  }  }  void inputArray(int\* array, int n)  {  for (int i = 0; i < n; i++)  {  std::cin >> array[i];  }  }  void showArray(int\* array, int n)  {  for (int i = 0; i < n; i++) std::cout << array[i] << ' ';  }  void rescueBinarIter(int\* array, int answer, int length)  {  int left = 0;  int right = length - 1;  int mid;  bool find = false;  while ((left <= right) && (find == false))  {  mid = (left + right) / 2;  if (array[mid] == answer)  {  find = true;  }  if (array[mid] > answer)  {  right = mid - 1;  }  else  {  left = mid + 1;  }  }  if (find)  {  std::cout << "index of targer by iter: " << mid << std::endl;  }  else  {  std::cout << "No" << std::endl;  }  }  struct node  {  char data;  node\* nextNodePtr;  node\* prevNodePtr;  node(char userData)  {  data = userData;  nextNodePtr = nullptr;  prevNodePtr = nullptr;  }  };  struct list  {  node\* head;  node\* tail;  list()  {  head = nullptr;  tail = nullptr;  }  list PushFront(char userDataAdd)  {  node\* add = new node(userDataAdd);  if (head == nullptr)  {  head = add;  return(\*this);  }  else if (tail == nullptr)  {  node\* temp = head;  head = add;  head->nextNodePtr = temp;  temp->prevNodePtr = head;  tail = temp;  return(\*this);  }  else  {  node\* temp = head;  head = add;  head->nextNodePtr = temp;  temp->prevNodePtr = add;  return(\*this);  }  }  list PushBack(char userDataAdd)  {  node\* add = new node(userDataAdd);  if (tail == nullptr)  {  tail = add;  return(\*this);  }  else if (head == nullptr)  {  node\* temp = tail;  tail = add;  tail->prevNodePtr = temp;  temp->nextNodePtr = tail;  head = temp;  return(\*this);  }  else  {  node\* temp = tail;  tail = add;  tail->prevNodePtr = temp;  temp->nextNodePtr = add;  return(\*this);  }  }  void ShowList()  {  node\* temp = head;  while (temp != nullptr)  {  std::cout << temp->data;  temp = temp->nextNodePtr;  }  }  };  list RecursePushBack(list Sentence)  {  char buff;  std::cin >> buff;  if (buff == '.')  {  Sentence.PushBack('.');  return (Sentence);  }  else  {  RecursePushBack(Sentence.PushBack(buff));  }  }  list RecursePushFront(list Sentence)  {  char buff;  std::cin >> buff;  if (buff == '.')  {  Sentence.PushBack('.');  return (Sentence);  }  else  {  RecursePushFront(Sentence.PushFront(buff));  }  }  int main()  {  int length;  int searchTarget;  int control;  do  {  std::cout << "enter control" << std::endl;  std::cout << "1 - binar rescue" << std::endl;  std::cout << "2 - twoway list" << std::endl;  std::cin >> control;  switch (control)  {  case 1:  {  std::cout << "Enter len ";  std::cin >> length;  int\* array = new int[length];  inputArray(array, length);  sortByChoise(array, length);  std::cout << "Sorted array = ";  showArray(array, length);  std::cout << std::endl;  std::cout << "Enter Target ";  std::cin >> searchTarget;  if (recurseBinar(array, 0, length - 1, searchTarget) == -1)  {  std::cout << "No" << std::endl;  }  else  {  std::cout << "index of target by rec: " << recurseBinar(array, 0, length - 1, searchTarget) << std::endl;  }  rescueBinarIter(array, searchTarget, length);  break;  }  case 2:  {  list Sentence;  char buff;  int controlBackFront;  std::cout << "Front - 1 back - 2" << std::endl;  std::cin >> controlBackFront;  if (controlBackFront == 1)  {  Sentence = RecursePushFront(Sentence);  }  if (controlBackFront == 2)  {  Sentence = RecursePushBack(Sentence);  }  Sentence.ShowList();  std::cout << std::endl;  }  }  } while (control == 1 || control == 2);  } |