|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 4** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Нелинейные структуры данных. Бинарное дерево.»** | |
|  | |
| Выполнил студент группы ИКБО-10-21 | Черномуров С.А. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2022

# **Цель работы**

Получение умений и навыков разработки и реализаций операций над структурой данных бинарное дерево.

# **Постановка задачи**

**Задание.**

1. Разработать программу в соответствии с требованиями варианта. Выполнить реализацию средствами ООП.

Вид дерева: идеально сбалансированное из n узлов (не AVL).

1. Реализовать операции общие для вариантов c 1 по 7
   1. Создать идеально сбалансированное бинарное дерево из n узлов. Структура узла дерева включает: информационная часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла определена вариантом.
   2. Отобразить дерево на экране, повернув его против часовой стрелки.
2. Реализовать операции варианта.
3. Разработать программу, демонстрирующую выполнение всех операций.
4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант №7. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | **Значение информационной части:**  Символьное значение  **Операции варианта:**  Определить уровень, на котором находится заданное значение.  Определить количество цифр в левом поддереве исходного дерева.  Вывести дерево располагая элементы вертикально. |

# **Решение**

**Объяснение алгоритмов применяемых функций**

**Задание 1**

Конструктор Tree создает массив длины n и записывает в него все указатели на записи, которые будут находиться в дереве, после чего вызывает метод построения структуры дерева.

|  |
| --- |
| //Метод конструктора дерева  Tree::Tree(int n) {  this->nodes = new Node\*[n];  this->amount = n;  for (int i = 0; i < amount; i++) {  nodes[i] = new Node();  }  insertItem(0, 1, 1, 2, 0);  } |

Метод insertItem берет исходный массив nodes, содержащий все записи дерева (еще не сформированные в формат дерева), и конструирует связи между элементами этого массива так, чтобы полученный ассоциативный массив можно было представить в форме идеально сбалансированного бинарного дерева, а также задает записям их уровень глубины. Метод построения связей основан на индексах записей в массиве, а также особенностях построения идеально сбалансированного бинарного дерева в данной реализации, а именно на постоянстве (в пределах уровня глубины дерева) разности индексов между левым, правым поддеревьями и их родительской записью:

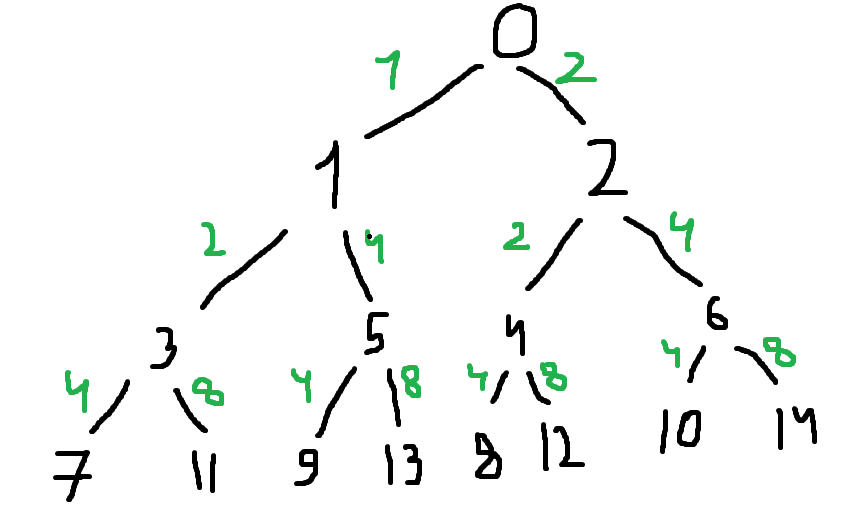


Рисунок 1. Демонстрация описанного выше метода

|  |
| --- |
| //Метод построения структуры дерева  void Tree::insertItem(int indexStart, int indexEl, int indexLeft, int indexRight, int deep) {  for (int i = indexStart; i < indexEl; i++) {  nodes[i]->deep = deep;  if (i + indexLeft < amount) {  nodes[i]->left = nodes[i + indexLeft]; nodes[i]->left->deep = deep + 1;  }  else {  if (i + indexLeft > amount) continue;  if (i + indexLeft == amount) return;  }  if (i + indexRight < amount) {nodes[i]->right = nodes[i + indexRight]; nodes[i]->right->deep = deep + 1;}  else {  if (i + indexRight > amount) continue;  if (i + indexRight == amount) return;  }  }  insertItem(indexStart + indexLeft, indexEl + indexRight, indexLeft \* 2, indexRight \* 2, deep+1);} |

Метод getRoot возвращает первый элемент массива nodes, который в данной реализации всегда является корневым для дерева.

|  |
| --- |
| //Метод получения корневой записи дерева  Node\* Tree::getRoot() {  return nodes[0];  } |

Метод printHor рекурсивно выводит дерево на экран (горизонтально). Это делается с помощью строки-префикса, которая сдвигает элементы вправо по мере заглубления в дерево, а также различных символов «палочек», выбор которых производится тернарным оператором.

|  |
| --- |
| //Метод вывода дерева по горизонтали  void Tree::printHor(const std::string& prefix, Node\* node, bool isLeft)  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|--" : "L--");  // print the value of the node  std::cout << node->value << std::endl;  // enter the next tree level - left and right branch  printHor(prefix + (isLeft ? "| " : " "), node->left, true);  printHor(prefix + (isLeft ? "| " : " "), node->right, false);  }  } |

Перегрузка метода printHor нужна для более удобного первого вызова этого метода.

|  |
| --- |
| //Перегрузка метода вывода дерева на экран по горизонтали  void Tree::printHor(Node\* node)  {  printHor("", node, false);  } |

Метод pasteValues присваивает каждой записи значения из переданного в него символьного массива arr.

|  |
| --- |
| //Метод заполнения записей дерева значениями  void Tree::pasteValues(char\* arr) {  for (int i = 0; i < amount; i++) {  nodes[i]->value = arr[i];  }  } |

Метод search обходит дерево в глубину и ищет запись, в которой значение value совпадает с переданным значением key. В случае нахождения совпадения найденная запись записывается в переменную found, переданную по ссылке.

|  |
| --- |
| //Метод поиска уровня записи с заданным значением  void Tree::search(Node\* root, char key, Node& found) {  if (root) {  if (root->value == key) found=\*root;  search(root->left, key, found);    search(root->right,key, found);  }  } |

Метод findNumbers обходит левое поддерево в глубину и считает количество записей, в которых значением является цифра. Результат записывается в переданную по ссылке переменную res.

|  |
| --- |
| //Метод подсчета количества цифр в левом поддереве  void Tree::findNumbers(Node\* root, int& res) {  if (root) {  if (root->value >= '0' && root->value <= '9') res++;  findNumbers(root->left, res);  findNumbers(root->right, res);  }  } |

Метод getHeight находит максимальную глубину, встречающуюся в дереве, и возвращает значение этой глубины.

|  |
| --- |
| //Метод получения высоты дерева  int Tree::getHeight() {  int max = -1;  for (int i = 0; i < amount; i++) {  if (nodes[i]->deep > max) max = nodes[i]->deep;  }  return max;  } |

Метод printVert циклично вызывает метод printRow выводящий дерево на экран построчно.

|  |
| --- |
| //Метод вертикального вывода дерева на экран  void Tree::printVert(Node\* p)  {  int height = getHeight();  for (int i = 0; i < height; i++) {  printRow(p, height, i);  }  } |

Метод printRow выводит один уровень дерева на экран и соединяет его с предыдущим уровнем с помощью символов «/» и «\».

|  |
| --- |
| //Метод вывода строки-уровня для вертикального вывода дерева на экран  void Tree::printRow(const Node\* p, const int height, int depth)  {  vector<char> vec;  getLine(p, depth, vec);  cout << setw((height - depth) \* 2);  bool toggle = true;  if (vec.size() > 1) {  for (char v : vec) {  if (v != '\_') {  if (toggle)  cout << "/" << " ";  else  cout << "\\" << " ";  }  toggle = !toggle;  }  cout << endl;  cout << setw((height - depth) \* 2);  }  for (char v : vec) {  if (v != '\_')  cout << v << " ";  }  cout << endl;  } |

Метод getLine формирует вектор символов, который обрабатывается методом printRow для вертикального вывода дерева на экран.

|  |
| --- |
| //Метод формирования строки для вертикального вывода дерева на экран  void Tree::getLine(const Node \* root, int depth, vector<char>&vals)  {  if (depth <= 0 && root != nullptr) {  vals.push\_back(root->value);  return;  }  if (root->left != nullptr)  getLine(root->left, depth - 1, vals);  else if (depth - 1 <= 0)  vals.push\_back('\_');  if (root->right != nullptr)  getLine(root->right, depth - 1, vals);  else if (depth - 1 <= 0)  vals.push\_back('\_');  } |

**Описание работы пользовательского интерфейса**

Сначала выводится выбор задания, пользователь вводом с клавиатуры выбирает задание. После чего консоль очищается и отображается ввод в зависимости от выбранного задания. Введенные данные обрабатываются, результат обработки выводится на экран, программа автоматически перезапускается.

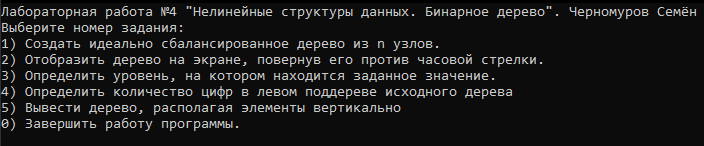


Рисунок 2. Интерфейс программы

# **Тестирование**

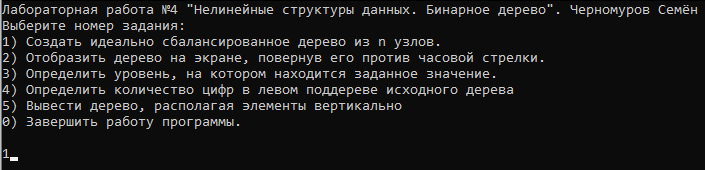


Рисунок 3. Выбор задания 1



Рисунок 4. Создание дерева с 10 узлами

Программа получает на вход количество узлов и создает идеально сбалансированное дерево с этим количеством узлов.

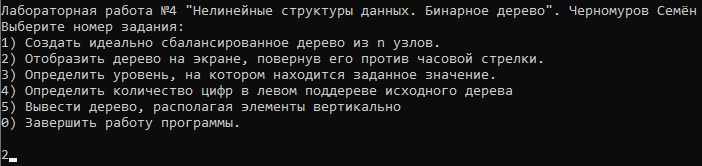


Рисунок 5. Выбор задания 2

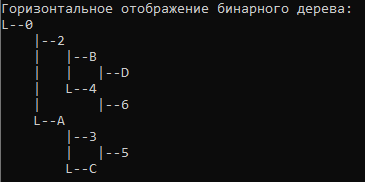


Рисунок 6. Результат работы программы (задание 2)

Программа выводит идеально сбалансированное дерево в горизонтальном формате на экран, причем левая ветвь находится сверху, а правая – снизу.

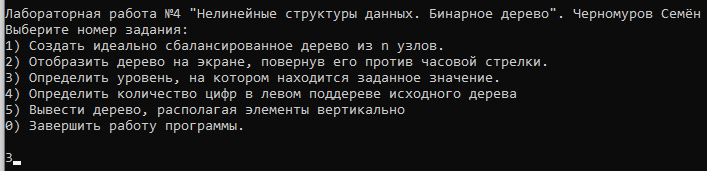


Рисунок 7. Выбор задания 3



Рисунок 8. Результат работы программы (задание 3)

Программа получает на вход символьное значение (3) и выводит на экран глубину, на которой находится это значение.

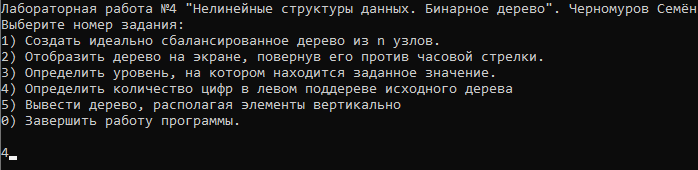


Рисунок 9. Выбор задания 4



Рисунок 10. Результат работы программы (задание 4)

Программа получает на вход корень левого поддерева (root -> left), совершает обход этого поддерева в глубину и считает число встретившихся цифр, после чего выводит их количество на экран.

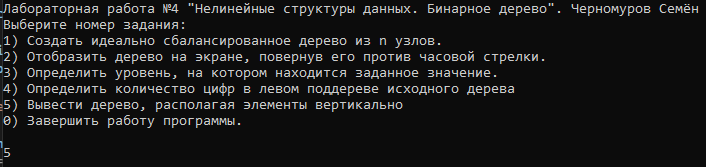


Рисунок 11. Выбор задания 5

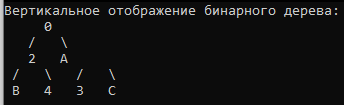


Рисунок 12. Результат работы программы (задание 5) для дерева из 7 узлов

Программа выводит идеально сбалансированное дерево в вертикальном формате на экран.

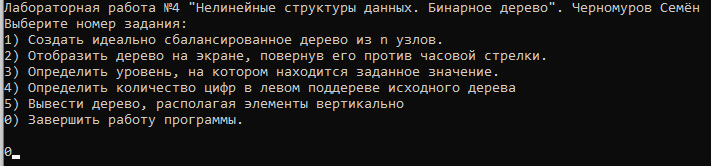


Рисунок 13. Выбор завершения работы программы

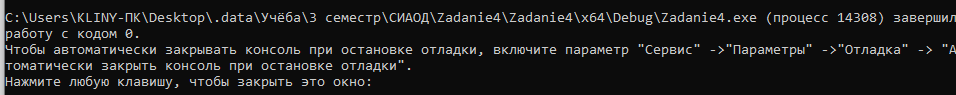


Рисунок 14. Завершение работы программы

При выборе функции завершения работы программы программа завершает свою работу.

Из результатов выполнения программы видно, что программа работает корректно.

# **Вывод**

В результате выполнения работы я:

1. Получил умения и навыки разработки и реализации операций над структурой данных бинарное дерево.
2. Закрепил знания по тестированию корректности работы программы.
3. Закрепил навыки создания пользовательских интерфейсов.

# **Исходный код программы**

**Файл main.cpp (основной алгоритм программы)**

|  |
| --- |
| #include "ideallyBalancedTree.h"  using namespace std;  int main() {  setlocale(LC\_ALL, "");  cout << "Лабораторная работа №4 \"Нелинейные структуры данных. Бинарное дерево\". Черномуров Семён\n";  cout << "Выберите номер задания:\n1) Создать идеально сбалансированное дерево из n узлов.\n" <<  "2) Отобразить дерево на экране, повернув его против часовой стрелки.\n" <<  "3) Определить уровень, на котором находится заданное значение.\n" <<  "4) Определить количество цифр в левом поддереве исходного дерева\n" <<  "5) Вывести дерево, располагая элементы вертикально\n" <<  "0) Завершить работу программы.\n\n";  int choice1;  do {  cin >> choice1;  if (choice1 != 1 && choice1 != 2 && choice1 != 3 && choice1 != 4 && choice1 != 5 && choice1 != 0) cout << "Введено неверное значение, попробуйте снова.\n";  } while (choice1 != 1 && choice1 != 2 && choice1 != 3 && choice1 != 4 && choice1 != 5 && choice1 != 0);  if (choice1 == 0) return 0;  cout << "Введите количество узлов: ";  int amount;  cin >> amount;  Tree\* tree = new Tree(amount);  char arr[16] = { '0','2','A','B','3','4','C','D','5','6','E','F','7','8','G','H' };  tree->pasteValues(arr);  system("cls");  switch (choice1)  {  case 1: {  break;  }  case 2: {  cout << "Горизонтальное отображение бинарного дерева:\n";  tree->printHor(tree->getRoot());  break;  }  case 3: {  cout << "Введите искомое значение: ";  char c;  cin >> c;  cout << "Уровень нахождения искомого значения(начиная с 0): ";  Node node;  tree->search(tree->getRoot(), c, node);  cout << node.deep;  break;  }  case 4: {  cout << "Кол-во цифр в левом поддереве исходного дерева: ";  int res = 0;  tree->findNumbers(tree->getRoot()->left, res);  cout << res;  break;  }  case 5: {  cout << "Вертикальное отображение бинарного дерева:\n";  tree->printVert(tree->getRoot());  break;  }  case 0:  return 0;  }  cout << "\n\n";  main();  } |

**Файл ideallyBalancedTree.h (заголовочный файл класса Tree идеально сбалансированного бинарного дерева)**

|  |
| --- |
| #pragma once  #ifndef BIN\_TREE  #define BIN\_TREE  #include <iostream>  #include <vector>  using namespace std;  struct Node {  char value;  Node\* left=nullptr;  Node\* right=nullptr;    int deep = 0;  };  class Tree  {  private:  Node\*\* nodes;  int amount = 0;  public:  Tree( int n);  Node\* getRoot();  void insertItem(int indexStart, int indexEl, int indexLeft, int indexRight, int deep);  void printHor(const std::string& prefix, Node\* node, bool isLeft);  void printHor(Node\* node);  void search(Node\* root, char key, Node& found);  void pasteValues(char\* arr);  void findNumbers(Node\* root, int& res);  void printVert(Node\* p);  void printRow(const Node\* p, const int height, int depth);  void getLine(const Node\* root, int depth, vector<char>& vals);  int getHeight();  };  #endif |

**Файл ideallyBalancedTree.cpp (реализации методов класса Tree идеально сбалансированного бинарного дерева)**

|  |
| --- |
| #include "ideallyBalancedTree.h"  #include<string>  #include<vector>  #include <iomanip>  using namespace std;  //Метод конструктора дерева  Tree::Tree(int n) {  this->nodes = new Node\*[n];  this->amount = n;  for (int i = 0; i < amount; i++) {  nodes[i] = new Node();  }  insertItem(0, 1, 1, 2, 0);  }  //Метод построения струтуры дерева  void Tree::insertItem(int indexStart, int indexEl, int indexLeft, int indexRight, int deep) {  for (int i = indexStart; i < indexEl; i++) {  nodes[i]->deep = deep;  if (i + indexLeft < amount) {  nodes[i]->left = nodes[i + indexLeft]; nodes[i]->left->deep = deep + 1;  }  else {  if (i + indexLeft > amount) continue;  if (i + indexLeft == amount) return;  }  if (i + indexRight < amount) {nodes[i]->right = nodes[i + indexRight]; nodes[i]->right->deep = deep + 1;}  else {  if (i + indexRight > amount) continue;  if (i + indexRight == amount) return;  }  }  insertItem(indexStart + indexLeft, indexEl + indexRight, indexLeft \* 2, indexRight \* 2, deep+1);  }  //Метод получения корневой записи дерева  Node\* Tree::getRoot() {  return nodes[0];  }  //Метод вывода дерева по горизонтали  void Tree::printHor(const std::string& prefix, Node\* node, bool isLeft)  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|--" : "L--");  // print the value of the node  std::cout << node->value << std::endl;  // enter the next tree level - left and right branch  printHor(prefix + (isLeft ? "| " : " "), node->left, true);  printHor(prefix + (isLeft ? "| " : " "), node->right, false);  }  }  //Перегрузка метода вывода дерева на экран по горизонтали  void Tree::printHor(Node\* node)  {  printHor("", node, false);  }  //Метод заполнения записей дерева значениями  void Tree::pasteValues(char\* arr) {  for (int i = 0; i < amount; i++) {  nodes[i]->value = arr[i];  }  }  //Метод поиска уровня записи с заданным значением  void Tree::search(Node\* root, char key, Node& found) {  if (root) {  if (root->value == key) found=\*root;  search(root->left, key, found);    search(root->right,key, found);  }  }  //Метод подсчета количества цифр в левом поддереве  void Tree::findNumbers(Node\* root, int& res) {  if (root) {  if (root->value >= '0' && root->value <= '9') res++;  findNumbers(root->left, res);  findNumbers(root->right, res);  }  }  //Метод получения высоты дерева  int Tree::getHeight() {  int max = -1;  for (int i = 0; i < amount; i++) {  if (nodes[i]->deep > max) max = nodes[i]->deep;  }  return max;  }  //Метод вертикального вывода дерева на экран  void Tree::printVert(Node\* p)  {  int height = getHeight();  for (int i = 0; i < height; i++) {  printRow(p, height, i);  }  }  //Метод вывода строки-уровня для вертикального вывода дерева на экран  void Tree::printRow(const Node\* p, const int height, int depth)  {  vector<char> vec;  getLine(p, depth, vec);  cout << setw((height - depth) \* 2); // scale setw with depth  bool toggle = true; // start with left  if (vec.size() > 1) {  for (char v : vec) {  if (v != '\_') {  if (toggle)  cout << "/" << " ";  else  cout << "\\" << " ";  }  toggle = !toggle;  }  cout << endl;  cout << setw((height - depth) \* 2);  }  for (char v : vec) {  if (v != '\_')  cout << v << " ";  }  cout << endl;  }  //Метод формирования строки для вертикального вывода дерева на экран  void Tree::getLine(const Node \* root, int depth, vector<char>&vals)  {  if (depth <= 0 && root != nullptr) {  vals.push\_back(root->value);  return;  }  if (root->left != nullptr)  getLine(root->left, depth - 1, vals);  else if (depth - 1 <= 0)  vals.push\_back('\_');  if (root->right != nullptr)  getLine(root->right, depth - 1, vals);  else if (depth - 1 <= 0)  vals.push\_back('\_');  } |

1. **Ответы на вопросы (чётные, вариант №2)**
2. **Какова степень сильноветвящегося дерева?**

У сильноветвящегося дерева степень дерева произвольная, у бинарного дерева – не более, чем 2.

1. **Как рассчитать длину пути в дереве?**

Длиной пути дерева называется сумма длин (весов) всех ребер, которые соединяют путевые узлы.

1. **Может ли дерево быть пустым?**

Дерево не может быть пустым, так как пустой граф по определению не может быть связным.

1. **Дайте определение алгоритму обхода.**

Обход графа – это процесс посещения (проверки и/или изменения) каждой вершины графа. Обходы графов классифицируются по порядку посещения вершин, а обход дерева является частным случаем обхода графа.

1. **Изобразите бинарное дерево, корень которого имеет индекс 6, и которое представлено в памяти таблицей вида**

|  |  |  |  |
| --- | --- | --- | --- |
| *Индекс* | *key* | *left* | *right* |
| 1 | 12 | 7 | 3 |
| 2 | 15 | 8 | NULL |
| 3 | 4 | 10 | NULL |
| 4 | 10 | 5 | 9 |
| 5 | 2 | NULL | NULL |
| 6 | 18 | 1 | 4 |
| 7 | 7 | NULL | NULL |
| 8 | 14 | 6 | 2 |
| 9 | 21 | NULL | NULL |
| 10 | 5 | NULL | NULL |

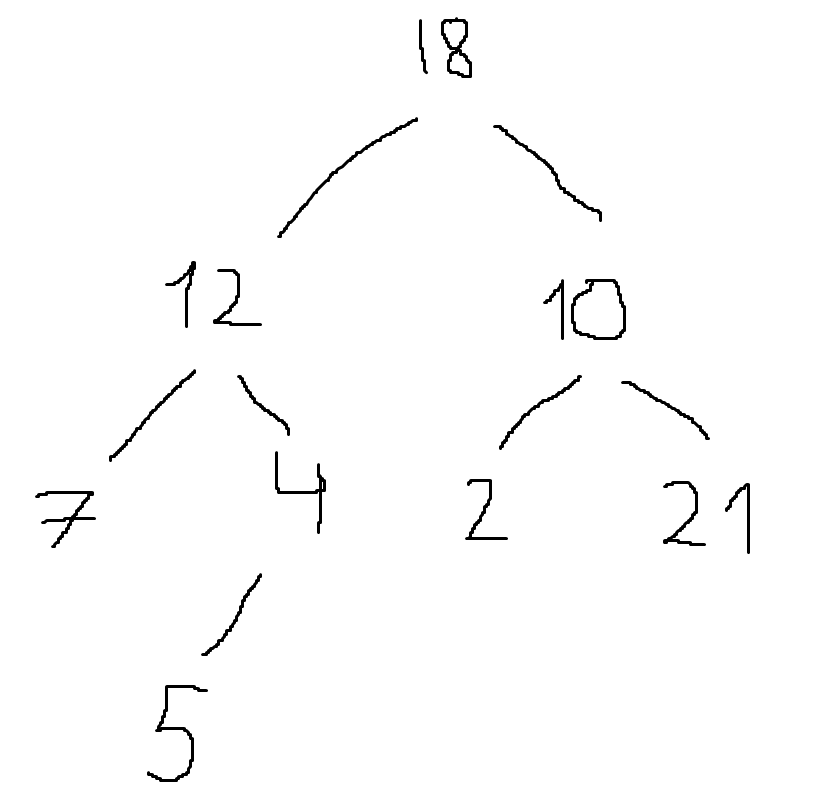


Рисунок 15. Полученное бинарное дерево

Две записи (2 и 8) не используются так как находятся выше по уровню, чем корень.

1. **Какая структура используется в алгоритме обхода дерева методом в «ширину»?**

При обходе дерева в ширину вершины посещаются слева-направо (горизонтально) последовательно по уровням сверху-вниз.

Пусть дан узел, в контейнер нужно поместить сначала его, а потом его наследников, при этом следующий узел должен быть обработан первым. Для этого можно воспользоваться очередью (элемент, вошедший в нее первым, первым будет и обработан).

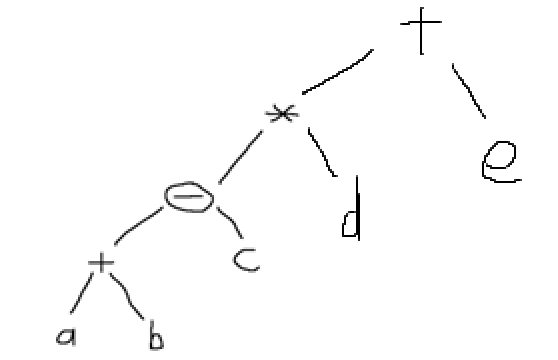
1. **Какая структура используется в нерекурсивном обходе дерева методом в «глубину»?**

Обход дерева начинается с того, что сначала в контейнер помещается стартовая вершина, потом при входе в цикл извлечения и обработки она извлекается из него, "обрабатывается" и после этого в контейнер помещаются все ее дочерние вершины, последняя из которых будет обработана на последующей итерации, и т.д.

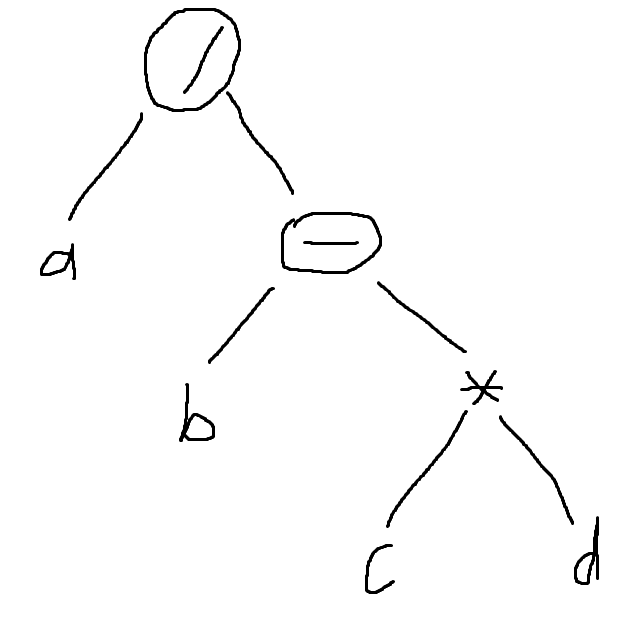
В конце концов, когда новые вершины перестанут добавляться в контейнер, оставшиеся в контейнере вершины будут только последовательно из него извлекаться и обрабатываться, до тех пор пока контейнер не станет пустым. Для такого алгоритма можно использовать структуру данных стек (последний зашедший в него элемент будет обработан первым).

1. **Для каждого заданного арифметического выражения постройте бинарное дерево выражений:**

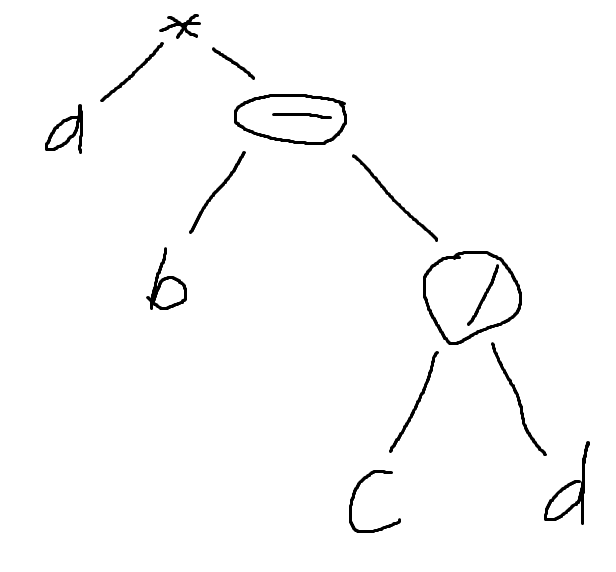
**a+b-c\*d+e**

****

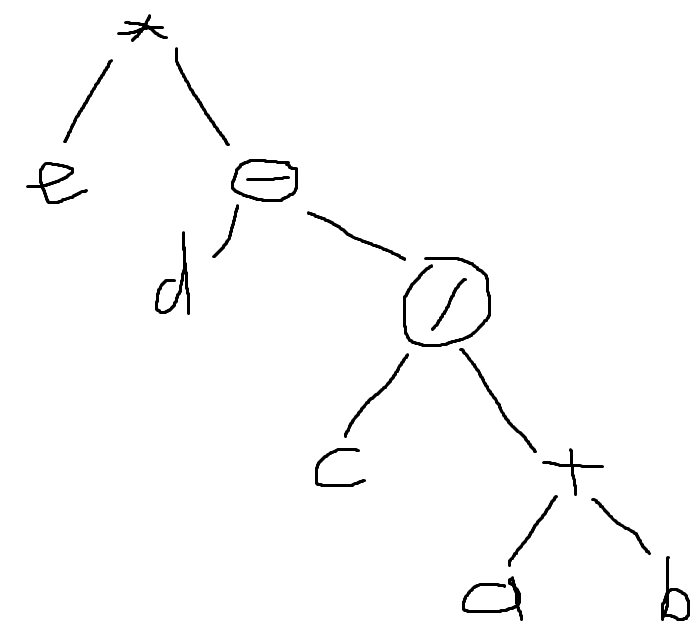
**/a-b\*c d**

****

**a b c d / - \***

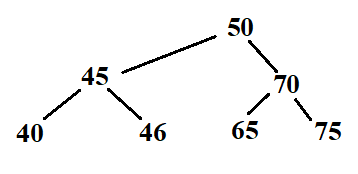
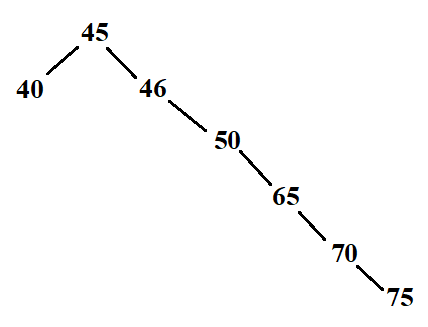
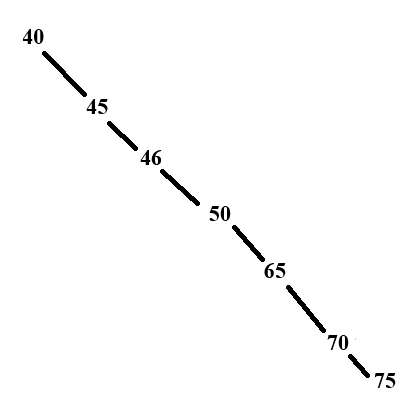
****

**\* - / + a b c d e**

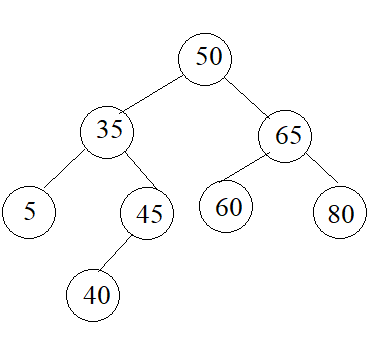
****

1. **Постройте бинарное дерево поиска, которое в результате симметричного обхода дало бы следующую последовательность узлов?**

**40 45 46 50 65 70 75**

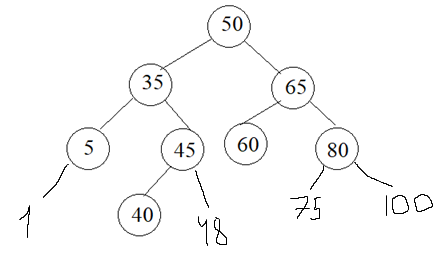
****

1. **Дано следующее бинарное дерево поиска**

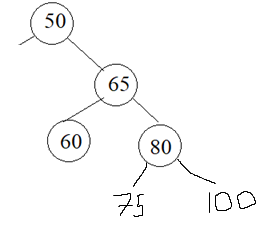


Покажите дерево:

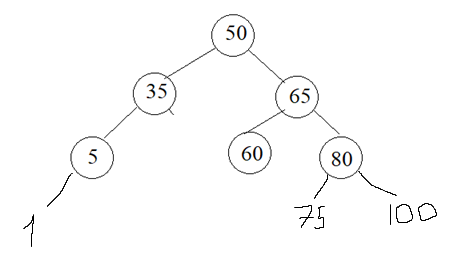
1. **после включения узлов 1 48 75 100**

****

1. **после удаления узлов 5, 35**

****

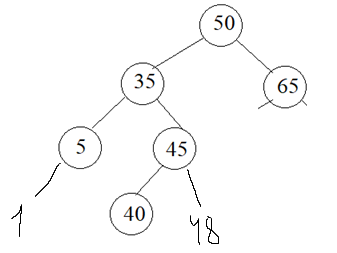
1. **после удаления узла 45**

****

1. **после удаления узла 50**

Граф станет пустым

1. **после удаления узла 65 и вставки его снова**

****