

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет» РТУ МИРЭА

Отчет по выполнению практического задания 7.1

Тема: «НЕЛИНЕЙНЫЕ СТРУКТУРЫ»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент группа

Комисарик М.А. ИКБО-20-23

Цель работы: изучить и реализовать бинарное дерево поиска.

Задание 1

Формулировка задачи

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом.

Тип дерева: бинарное дерево поиска.

Тип данных узла: Строка – город.

Операции:

- Вставка элемента
- Симметричный обход дерева
- Обход дерева в ширину
- Среднее арифметическое всех узлов
- Длина пути от корня до заданного значения

Модель решения

Бинарное дерево поиска — это бинарное дерево, узлы которого упорядочены по значениям ключей по правилу:

- слева от корня узлы со значением ключа меньше, чем ключ в корне
- справа со значением ключа большими, чем в корне

Для представления дерева в памяти был выбран способ с применением указателей. Каждый узел дерева содержит два указателя на корневые узлы левого и правого поддерева.

Для вставки узла в бинарное дерево поиска требуется найти элемент в соответствии с определением дерева. Рекурсивно выполняется сравнение с корнем поддерева: если ключ корня меньше ключа нового узла, то поиск продолжается в поддереве с корнем в левом узле поддерева, в противном случае — в правом. Когда при проверке один из узлов пуст, вставка осуществляется на его место.

Алгоритм симметричного обхода:

• обойти в симметричном порядке левое поддерево

- посетить корневой узел (добавить в результирующий массив)
- обойти в симметричном порядке правое поддерево

Обход в ширину. Такой обход выводит дерево по уровням. При обходе, пройденные узлы записываются в очередь. Затем из очереди извлекается узел и обрабатывается, а в очередь отправляются его сыновья (узлы левого и правого деревьев).

Для нахождения среднего арифметического городов дерева требуется найти среднее арифметическое символов для каждой і-й позиции строки и собрать эти символы в результирующую строку. Для этого требуется обойти дерево.

Для нахождения длины пути от корня до значения можно выполнить обход дерева в глубину, в каждой итерации которого параметр глубины будет увеличиваться на 1. При нахождении узла вывести этот параметр. Если узел не был найден вывести -1.

Код программы

Код алгоритма вставки:

```
void BinaryTree::Insert(const std::string& value)
2:
         BinaryTree* newNode = new BinaryTree(value);
3:
4:
         BinaryTree* currentNode = this;
         while (true)
5:
6:
              if (CompareStrings(newNode->Value(), currentNode->Value()) < 0)</pre>
7:
8:
                  if (currentNode->Left() == nullptr)
9:
10:
                      currentNode->SetLeft(newNode);
11:
12:
                      return:
13:
14:
                  currentNode = currentNode->Left();
15:
                  continue:
16:
             if (currentNode->Right() == nullptr)
17:
18:
                  currentNode->SetRight(newNode);
19:
20:
                  return:
21:
22:
             currentNode = currentNode->Right();
23:
         }
24:
```

Код симметричного обхода:

```
1: void BinaryTree::TraverseInOrder(std::vector<std::string>& result) const
2: {
3:    if (m_left != nullptr)
4:    {
5:        m_left->TraverseInOrder(result);
6: }
```

Код обхода в ширину:

```
1:
     void BinaryTree::TraverseLevelOrder(std::vector<std::string>& result) const
2:
3:
         std::queue<const BinaryTree*> queue;
4:
         queue.push(this);
5:
         result.push_back(m_value);
6:
         while (!queue.empty())
7:
8:
             const BinaryTree* currentNode = queue.front();
9:
             if (currentNode->Left() != nullptr)
10.
11 .
                  queue.push(currentNode->Left());
12:
                  result.push_back(currentNode->Left()->Value());
13:
14:
             if (currentNode->Right() != nullptr)
15:
                  queue.push(currentNode->Right());
16:
17:
                  result.push_back(currentNode->Right()->Value());
18:
19:
             queue.pop();
         }
20:
21:
```

Код среднего арифметического:

```
22:
     std::string BinaryTree::Average() const
23:
24:
         std::string out;
25:
         std::vector<float> average;
26:
         std::vector<std::string> values;
27:
         TraverseInOrder(values);
28:
         for (const std::string& value : values)
29:
30:
              if (average.size() < value.length())</pre>
31:
                  average.resize(value.length());
32:
33:
              for (unsigned int i = 0; i < value.length(); ++i)</pre>
34:
              {
                  average[i] += value[i];
35:
36:
37:
         for (float& a : average)
38:
39:
40:
              a /= values.size();
41:
              out.push_back(static_cast<char>(a));
42:
43:
         return out;
44:
```

Код вывода дерева в консоль:

```
1: void BinaryTree::Print(unsigned int space, unsigned int level) const
2: {
3: if (m_right != nullptr)
```

```
4:
5:
                m_right->Print(space, level + 1);
           }
6:
           std::string line(level * space, ' ');
7:
           if (m_value.length() > space)
8:
9:
                line += m_value.substr(0, space) + '\n';
10:
11:
                std::cout << line;</pre>
12:
           }
13:
           else
14:
                unsigned int edge_space = (space - m_value.length()) / 2;
line += std::string(edge_space, ' ') + m_value + '\n';
15:
16:
                std::cout << line;</pre>
17:
18:
19:
           if (m_left != nullptr)
20:
21:
                m_left->Print(space, level + 1);
22:
           }
23:
      }
```

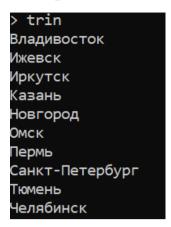
Результаты тестирования

Тестирование вставки элемента в дерево:

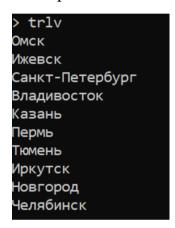
```
Введите корневой город дерева: Омск
Команды:
insert, trin, trlv, find, print, mean, exit
Введите команды:
insert Ижевск
  Омск
            Ижевск
 insert Санкт-Петербург
          Санкт-Пете
  Омск
            Ижевск
 insert Владивосток
          Санкт-Пете
  Омск
            Ижевск
                    Владивосто
```

Для дальнейших тестов было создано данное дерево:

Тестирование симметричного обхода дерева:



Тестирование обхода дерева в ширину:



Тестирование поиска узла по ключу:

```
> find Омск
Расстояние до города: 0
> find Казань
Расстояние до города: 2
> find Калининград
Город не найден.
```

Тестирование среднего арифметического:

```
> mean
Нккксщфшыьээяяю
```

Вывод

В рамках задания было реализовано и протестировано бинарное дерево поиска. Были реализованы алгоритмы обхода дерева, вставки элемента, поиска элемента.