****

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 Комисарик М.А.

группа ИКБО-20-23

**Москва 2024**

**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 i-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:

1. void BinaryTree::Insert(const std::string& value)
2. {
3. BinaryTree\* newNode = new BinaryTree(value);
4. BinaryTree\* currentNode = this;
5. while (true)
6. {
7. if (CompareStrings(newNode->Value(), currentNode->Value()) < 0)
8. {
9. if (currentNode->Left() == nullptr)
10. {
11. currentNode->SetLeft(newNode);
12. return;
13. }
14. currentNode = currentNode->Left();
15. continue;
16. }
17. if (currentNode->Right() == nullptr)
18. {
19. currentNode->SetRight(newNode);
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. }
7. result.push\_back(m\_value);
8. if (m\_right != nullptr)
9. {
10. m\_right->TraverseInOrder(result);
11. }
12. }

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. {
16. queue.push(currentNode->Right());
17. result.push\_back(currentNode->Right()->Value());
18. }
19. queue.pop();
20. }
21. }

Код среднего арифметического:

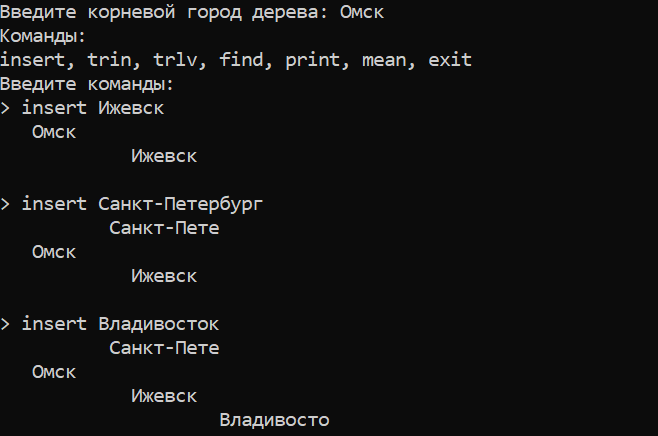
1. std::string BinaryTree::Average() const
2. {
3. std::string out;
4. std::vector<float> average;
5. std::vector<std::string> values;
6. TraverseInOrder(values);
7. for (const std::string& value : values)
8. {
9. if (average.size() < value.length())
10. average.resize(value.length());
11. for (unsigned int i = 0; i < value.length(); ++i)
12. {
13. average[i] += value[i];
14. }
15. }
16. for (float& a : average)
17. {
18. a /= values.size();
19. out.push\_back(static\_cast<char>(a));
20. }
21. return out;
22. }

Код вывода дерева в консоль:

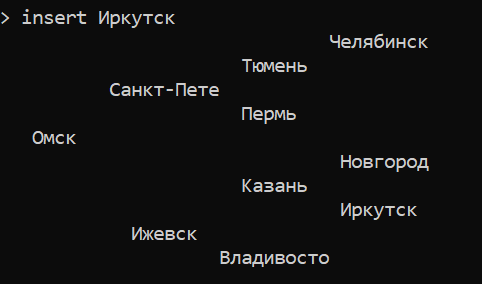
1. void BinaryTree::Print(unsigned int space, unsigned int level) const
2. {
3. if (m\_right != nullptr)
4. {
5. m\_right->Print(space, level + 1);
6. }
7. std::string line(level \* space, ' ');
8. if (m\_value.length() > space)
9. {
10. line += m\_value.substr(0, space) + '\n';
11. std::cout << line;
12. }
13. else
14. {
15. unsigned int edge\_space = (space - m\_value.length()) / 2;
16. line += std::string(edge\_space, ' ') + m\_value + '\n';
17. std::cout << line;
18. }
19. if (m\_left != nullptr)
20. {
21. m\_left->Print(space, level + 1);
22. }
23. }

Результаты тестирования

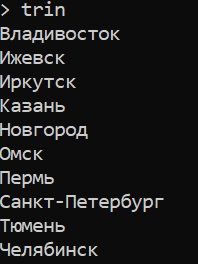
Тестирование вставки элемента в дерево:



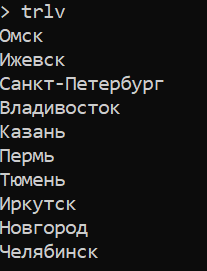
Для дальнейших тестов было создано данное дерево:



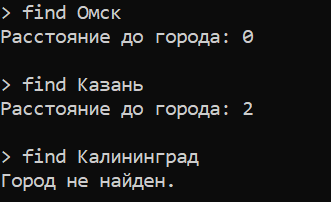
Тестирование симметричного обхода дерева:



Тестирование обхода дерева в ширину:



Тестирование поиска узла по ключу:



Тестирование среднего арифметического:



Вывод

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