Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа № 1

По дисциплине ЯПИС

За шестой семестр

Тема: «Работа с таблицей идентификаторов»

Вариант 8

Выполнил:

Студент 3 курса

Группы ИИ-16(2)

Пилипук

Проверил:

Слинко

Брест, 2021

*Цель работы:* изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц символов (идентификаторов).

Для выполнения лабораторной работы требуется написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить поиск идентификатора в этой таблице.

Список идентификаторов задан в виде текстового файла. Длина идентификаторов ограничена 32 символами.

Вариант 8: Таблица строится с использованием хеш-функции из варианта №6. Одинаковые элементы помещаются в одну ячейку, внутри которой используется логарифмический поиск. Хэш-функция - сумма трех первых букв идентификатора. При этом все буквы переводятся в заглавные (большие).

Поиск может быть выполнен более эффективно, если элементы таблицы упорядочены (отсортированы) согласно некоторому естественному порядку.

Эффективным методом поиска в упорядоченном списке из N элементов является бинарный или логарифмический поиск. Идентификатор, который требуется найти, сравнивается с элементом (N+1)/2 в середине таблицы. Если этот элемент не является искомым, то мы должны просмотреть только блок элементов, пронумерованных от 1 до (N+1)/2-1, или блок элементов от (N+1)/2+1 до N в зависимости от того, меньше или больше искомый элемент того, с которым его сравнили. Затем процесс повторяется над нужным блоком в два раза меньшего размера. Так продолжается до тех пор, пока либо элемент не будет найден, либо алгоритм не дойдет до очередного блока, содержащего один или два элемента (с которыми уже можно выполнить прямое сравнение искомого элемента).

Так как на каждом шаге число элементов, которые могут содержать искомый элемент, сокращается наполовину, то максимальное число сравнений равно 1+log2(N).

Тогда время поиска элемента в таблице идентификаторов можно оценить как ТП = O(log2N). Для сравнения: при N=128 бинарный поиск требует самое большее 8 сравнений, а поиск в неупорядоченной таблице — в среднем 64 сравнения. Метод называют «бинарным поиском», поскольку на каждом шаге объем рассматриваемой информации сокращается в два раза, а «логарифмическим» — поскольку время, затрачиваемое на поиск нужного элемента в массиве, имеет логарифмическую зависимость от общего количества элементов в нем.

Недостатком метода является требование упорядочивания таблицы идентификаторов. Так как массив информации, в котором выполняется поиск, должен быть упорядочен, то время его заполнения уже будет зависеть от числа элементов в массиве. Таблица идентификаторов зачастую просматривается еще до того, как она заполнена полностью, поэтому требуется, чтобы условие упорядоченности выполнялось на всех этапах обращения к ней.

Следовательно, для построения таблицы можно пользоваться только алгоритмом прямого упорядоченного включения элементов.

При добавлении каждого нового элемента в таблицу сначала надо определить место, куда поместить новый элемент, а потом выполнить перенос части информации в таблице, если элемент добавляется не в ее конец.

В итоге при организации логарифмического поиска в таблице идентификаторов мы добиваемся существенного сокращения времени поиска нужного элемента за счет увеличения времени на помещение нового элемента в таблицу. Поскольку добавление новых элементов в таблицу идентификаторов происходит существенно реже, чем обращение к ним, то этот метод следует признать более эффективным, чем метод организации неупорядоченной таблицы.

Проект с исходным кодом можно найти в репозитории по ссылке: https://github.com/onelxj/labs

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

class table

{

public:

table() {

std::ifstream file("test.txt");

while (!file.eof())

{

std::string temp;

file >> temp;

insertId(0, data.size(), temp);

}

file.close();

};

~table() {};

void print()

{

for (auto& x : data)

{

std::cout << "hash: " << x.first << "\n";

for (auto& s : x.second)

{

std::cout << s << " ";

}

std::cout << "\n";

}

}

private:

// methods

unsigned int hash(const std::string& str)

{

return std::toupper(str[0]) + std::toupper(str[1]) + std::toupper(str[2]);

}

void insertId(size\_t f, size\_t l, const std::string& str)

{

unsigned int h = hash(str);

int n = l - f;

if (n == 1 || n == 0)

{

std::pair<unsigned int, std::vector<std::string>> newPair;

newPair.second.push\_back(str);

newPair.first = h;

if (n == 0)

{

data.push\_back(newPair);

return;

}

if (h == data[f].first)

{

data[f].second.push\_back(str);

return;

}

if (h < data[f].first)

{

data.insert(data.begin() + f, newPair);

return;

}

else

{

data.insert(data.begin() + f + 1, newPair);

return;

}

}

if (h == data[(l+f + 1) / 2].first)

{

data[(l+f + 1) / 2].second.push\_back(str);

return;

}

if (h < data[(l+f + 1) / 2].first)

{

return insertId(0, (l+f + 1) / 2, str);

}

else

{

return insertId((l + f) / 2, l, str);

}

}

//variables

std::vector<std::pair<unsigned int, std::vector<std::string>>> data;

};

int main()

{

table obj;

obj.print();

return 0;

}



Вывод: научился строить хэш-таблицы. Разобрал и реализовал метод при котором одинаковые элементы помещаются в одну ячейку, внутри которой используется логарифмический поиск.