**Сбалансированные деревья, хэш–таблицы**

**Цель работы** – построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнить эффективность устранения коллизий при внешнем и внутреннем хешировании

**Входные данные:**

Строка букв, длиною не более 279.

**Выходные данные:**

Двоичное дерево поиска, составленное из букв данной строки, представленное в графическом виде; узлы, значения которых встречается в данной строке более одного раза, выделены цветом. Двоичное дерево поиска, балансированное двоичное дерево без без повторяющихся букв строки .

Хэш-таблицы, коллизии в которых разрешены при помощи метода цепочек и метода внутреннего хэширования.

**Особенность работы программы**

Для деревьев, состоящих только из корня, отсутствует графическая реализация. Под хэш-таблицу с открытой адресации выделена статическая область памяти.

**Внутренние структуры данных**

Двоичное дерево поиска

**struct** tree\_node

{

**char** **name**; -ключ

**struct** tree\_node \***left**; - указатель на левый потомок

**struct** tree\_node \***right**; -указатель на правый потомок

};

Сбалансированное дерево

**struct** Node

{

**char** **key**;

**char** **height**; - высота элемента

**struct** Node \***right**;

**struct** Node \***left**;

};

Хэш таблицы

**typedef** **struct** T1

{

**char** **data**; - ключ

**int** **number**; - число повторений буквы в строке

} T

**typedef** **struct** Node\_ {

**struct** Node\_ \***next**;

T **key**;

} Node1;

**Основные алгоритмы**

**Добавление элемента в хэш-таблицу с открытой адресацией(ОА)**

**void** put(**char** letter,**int** number, Node1 z[], **int** size)

{

**int** i=hash1(letter);

**while**((z[i].**key**.**data**!=0)&&(i<size)) i++;

**if** (i==size)

{

i=0;

**while**(z[i].**key**.**data**!=0) i++;

}

z[i].**key**.**data**=letter;

z[i].**key**.**number**=number;

**return**;

}

**Поиск элемента в ОА хэш-таблице**

Node1 search\_op(**char** letter, Node1 z[], **int** size)

{

Node1 \*p;

**int** i=hash1(letter);

**while**((z[i].**key**.**data**!=letter)&&(i<size))

{

i++;

hash\_op++;

}

**if**(z[i].**key**.**data**==letter) hash\_op++;

**if** (i==size)

{

i=0;

**while**(z[i].**key**.**data**!=letter)

i++;

}

**return** z[i];

}

**Добавление элемента в хэш-таблицу МЦ(метод цепочек)**

Node1 \*insertNode(**char** data1,**int** number) {

Node1 \*p, \*p0;

hashTableIndex bucket;

bucket = hash(data1);

**if** ((p = malloc(**sizeof**(Node1))) == 0) {

fprintf (**stderr**, **"out** **of** **memory** **(insertNode)\n"**);

exit(1);

}

p0 = hashTable[bucket];

hashTable[bucket] = p;

p->**next** = p0;

p->**key**.**data** = data1;

p->**key**.**number**=number;

**return** p;

}

**Поиск элемента в МЦ хэш-таблице**

Node1 \*findNode (**char** data) {

Node1 \*p;

p = hashTable[hash(data)];

**while** (p && !**compEQ**(p->**key**.**data**, data))

{

hash\_open++;

p = p->**next**;

}

**if** (p) hash\_open++;

**return** p;

}

**Сравнение эффективности поиска**

В двоичном дереве поиска после удаления повторяющих элементов не может содержаться более 26 элементов.

**Двоичное дерево поиска:** количество элементов- 26, память 234 байта

|  |  |  |
| --- | --- | --- |
| Время поиска(cекунд) | Количество сравнений | Искомый элемент |
| 2.913793e-004 | 1 | a |
| 3.503448e-004 | 2 | b |
| 5.296552e-004 | 3 | c |
| 5.862069e-004 | 4 | d |
| 8.548276e-004 | 5 | e |
| 8.882759e-004 | 6 | f |
| 8.900000e-004 | 7 | g |
| 1.030000e-003 | 8 | h |
| 1.034828e-003 | 9 | I |
| 1.154828e-003 | 10 | J |
| 1.235172e-003 | 11 | k |
| 1.290000e-003 | 12 | l |
| 1.305172e-003 | 13 | m |
| 1.320000e-003 | 14 | n |
| 1.340000e-003 | 15 | o |
| 1.444828e-003 | 16 | p |
| 1.470000e-003 | 17 | q |
| 1.471724e-003 | 18 | r |
| 1.595172e-003 | 19 | s |
| 1.634828e-003 | 20 | t |
| 1.745172e-003 | 21 | u |
| 1.755172e-003 | 22 | v |
| 1.775172e-003 | 23 | w |
| 1.884828e-003 | 24 | x |
| 1.925172e-003 | 25 | y |
| 2.150000e-003 | 26 | z |

**Среднее время поиска:** 0.001350

**Среднее количество сравнений**: 14.5

**АВЛ дерево**: количество элементов- 26, память 260 байт

|  |  |  |
| --- | --- | --- |
| Время Поиска(cекунд) | Количество сравнений | Искомый элемент |
| 1.215172e-003 | 5 | a |
| 7.910345e-004 | 4 | b |
| 6.179310e-004 | 5 | c |
| 7.310345e-004 | 3 | d |
| 1.260000e-003 | 5 | e |
| 1.240000e-003 | 4 | f |
| 1.154828e-003 | 5 | g |
| 8.700000e-004 | 2 | h |
| 1.180000e-003 | 5 | I |
| 1.175172e-003 | 4 | J |
| 1.245172e-003 | 5 | k |
| 9.075862e-004 | 3 | l |
| 1.254828e-003 | 5 | m |
| 1.005172e-003 | 4 | n |
| 1.034828e-003 | 5 | o |
| 6.000000e-004 | 1 | p |
| 1.060000e-003 | 4 | q |
| 9.800000e-004 | 3 | r |
| 1.020000e-003 | 4 | s |
| 7.200000e-004 | 2 | t |
| 1.074828e-003 | 5 | u |
| 9.548276e-004 | 4 | v |
| 1.030000e-003 | 5 | w |
| 6.489655e-004 | 3 | x |
| 9.651724e-004 | 4 | y |
| 9.951724e-004 | 5 | z |

**Среднее время поиска:** 0.001028

**Среднее количество сравнений**: 4.192308

**Хэш-таблица, метод цепочек:**

Количество строчек в хэш-таблице - 10. Хэш-функция askii\_kod mod 10.

Память -352 байта

Количество элементов – 26.

|  |  |  |
| --- | --- | --- |
| Время Поиска(cекунд) | Количество сравнений | Искомый элемент |
| 1.075172e-003 | 3 | a |
| 1.125172e-003 | 3 | b |
| 1.000000e-003 | 3 | c |
| 1.060000e-003 | 3 | d |
| 1.190000e-003 | 3 | e |
| 1.190000e-003 | 3 | f |
| 8.206897e-004 | 2 | g |
| 4.582759e-004 | 2 | h |
| 8.900000e-004 | 2 | I |
| 5.282759e-004 | 2 | J |
| 8.262069e-004 | 2 | k |
| 7.600000e-004 | 2 | l |
| 8.206897e-004 | 2 | m |
| 7.748276e-004 | 2 | n |
| 7.551724e-004 | 2 | o |
| 6.037931e-004 | 2 | p |
| 3.351724e-004 | 1 | q |
| 3.800000e-004 | 1 | r |
| 3.800000e-004 | 1 | s |
| 2.386207e-004 | 1 | t |
| 4.400000e-004 | 1 | u |
| 3.700000e-004 | 1 | v |
| 3.700000e-004 | 1 | w |
| 3.700000e-004 | 1 | x |
| 3.800000e-004 | 1 | y |
| 2.386207e-004 | 1 | z |

**Среднее время поиска:** 0.000678 c

**Среднее количество сравнений**: 1.884615

**Хэш-таблица, открытая адресация:**

Хэш-функция askii\_kod mod 20 +10.

Память -420 байта

Количество элементов – 26.

|  |  |  |
| --- | --- | --- |
| Время Поиска(cекунд) | Количество сравнений | Искомый элемент |
| 2.300000e-004 | 1 | a |
| 2.248276e-004 | 1 | b |
| 2.251724e-004 | 1 | c |
| 2.051724e-004 | 1 | d |
| 2.251724e-004 | 1 | e |
| 2.248276e-004 | 1 | f |
| 2.251724e-004 | 1 | g |
| 2.300000e-004 | 1 | h |
| 2.300000e-004 | 1 | I |
| 2.248276e-004 | 1 | J |
| 2.051724e-004 | 1 | k |
| 2.248276e-004 | 1 | l |
| 2.300000e-004 | 1 | m |
| 2.248276e-004 | 1 | n |
| 2.248276e-004 | 1 | o |
| 2.300000e-004 | 1 | p |
| 2.248276e-004 | 1 | q |
| 2.048276e-004 | 1 | r |
| 2.300000e-004 | 1 | s |
| 2.600000e-004 | 1 | t |
| 3.748276e-004 | 4 | u |
| 3.200000e-004 | 4 | v |
| 3.200000e-004 | 4 | w |
| 5.351724e-004 | 24 | x |
| 5.200000e-004 | 24 | y |
| 5.200000e-004 | 24 | z |

**Среднее время поиска: 0.000293** c

**Среднее количество сравнений**: 4.923077

**Вывод:**

Хеш-таблицы показывают наименьшее время поиска, несбалансированное ДДП – наибольшее. Сбалансированное ДДП даёт средний результат. Однако, хэш-таблицы проигрывают по памяти деревьям.

Тем не менее, при использовании ХТ, результат напрямую зависит от используемой хеш-функции.

Выбрать наиболее оптимальную структуру невозможно, поскольку все они обладают своими недостатками. В сбалансированном дереве каждое добавление нового элемента может приводить к реструктуризации дерева. Хеш-таблицы требуют подбора оптимальной хеш-функции, и также могут требовать реструктуризации. Также при закрытом хешировании размер ХТ, фактически, строго ограничен параметрами хеш-функции