Семестровая работа

Структура данных- Бор (префиксное дерево).

Выполнил:

Юсупов М.Р. 11-002

2021г

**Определение:**

Бор, или префиксное дерево, - структура данных для эффективного хранения и обработки строк, представляющая из себя корневое дерево с символами на рёбрах. Вершины в боре соответствуют отдельным символам. Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной. Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца.

Авторы: Briandais, 1959; Fredkin, 1960.Происхождение слова “trie” –retrieval (поиск, извлечение, выборка, возврат).Альтернативные названия: бор(Д. Кнут, Т. 3, 1978, выборка), луч(Д. Кнут, Т. 3, 2000, получение), нагруженное дерево (А. Ахои др., 2000).

**Принцип устройства. Особенности:**

* Префиксное дерево (Бор) содержит n ключей (строк).
* Ключ(key)–это набор символов (c1, c2, …, cm) из алфавита A= {a1, a2, …, ad}.
* Каждый узел содержит от 1 до d дочерних узлов
* За каждым ребром закреплен символ алфавита
* В каждой вершине есть логическая переменная, которая показывает, является ли вершина терминальной. То есть от корня до этой вершины можно составить слово, имеющееся в этом дереве.
* Хранит строки, символы, либо числа, представленные как строки

**Оценка временной сложности:**

Временная сложность всех трех операций: добавление, удаление и поиск выполняются за O(|Key|), где |Key| - это длинна слова. Алгоритм данных операций заключается в том, что мы будем спускаться из корня дерева на нижние уровни, каждый раз переходя в узел, чья метка совпадает с очередным символом ключа, пробегаясь по размеру алфавита, чтобы найти данный узел (в случаи добавления, если узла не существует, мы его создаем). Тем самым каждый спуск выполняется за O(1) – так как мы переходим за константное время по ссылкам от одного узла к другому, более нижнему. Следовательно, мы совершим действий равное длине слова умноженное на константу.

**Графики:**

Время представлено в миллисекундах\*.

В графиках для функции add(), сложность различается от O(|Key|), так как при создании новой вершины мой алгоритм создает массив размер n, где n – размер алфавита. То есть сложность алгоритма add() – O(n\*|Key|).

Функции remove() и contain() отрабатывают приблизительно за сложность O(|Key|), небольшие различия во времени происходят из-за сторонних процессов, обрабатывающиеся на моем устройстве.

**Вывод:**

Префиксное дерево – это эффективный способ хранить словарь и искать в нем слова. Его преимуществом является то, что все операции с любой данной строкой длины (поиск, вставка, удаление) выполняются за время O(|Key|), не зависящее от числа элементов в хранимом множестве. А также для хранения ключей (строк) не используется дополнительной памяти (ключи не хранятся в узлах).

Недостатками является то, что Бор хранит строки, символы, либо числа, представленные как строки, и поэтому неприменим к другим типам данных. Если реализовывать ассоциативный массив на обычном боре, а ключами будут являться строки, то будет использоваться слишком много памяти. А также много памяти будет затрачиваться, если алфавит имеет большое значение, тогда даже если слова, хранимые в префиксном дереве, покрывают только часть алфавита, то все равно будет создаваться под каждый узел количество ссылок, равное значению алфавита.

Область применения префиксных деревьев огромна – их можно использовать для хранения словарей, для исправлений орфографических ошибок в слове, для предложения дальнейших запросов в поисковиках, для решения задач, связанных с построением автоматов — то есть в задачах, где необходимо быстро получать наборы ключей с заданным префиксом.

Список литературы:

1. Кормен. Алгоритмы - построение и анализа.
2. Седжвик. Фундаментальные алгоритмы на C++
3. Ахо, А.В., Хопкрофт, Д., Ульман Д.Д. Структуры данных и алгоритмы. –М.: Вильямс, 2001.

**Код струтуры:**

#include <iostream>

**using** **namespace** std**;**

const int alphabet **=** 26**;**

struct PrefixTree **{**

PrefixTree **\***next**[**alphabet**]** **=** **{**0**};**

bool is\_terminal **=** **false;**

**};**

void add**(**string s**,**PrefixTree**\*** root**)** **{** //добавление слова

PrefixTree **\***head **=** root**;**

**for(**int i **=** 0**;**i**<**s**.**length**();**i**++)** **{**

char c **=** s**[**i**];**

**if(**head**->**next**[**c**-**'a'**]==nullptr)** **{**

head**->**next**[**c**-**'a'**]** **=** **new** PrefixTree**;**

**}**

head **=** head**->**next**[**c**-**'a'**];**

**}**

head**->**is\_terminal **=** **true;**

**}**

bool contains**(**string s**,**PrefixTree**\*** root**)** **{** //содержание слова в структуре

PrefixTree **\***head **=** root**;**

bool flag **=** **true;**

**for(**int i **=** 0**;**i**<**s**.**length**();**i**++)** **{**

char c **=** s**[**i**];**

**if(**head**->**next**[**c**-**'a'**]==nullptr)** **{**

flag **=** **false;**

**return** flag**;**

**}**

head **=** head**->**next**[**c**-**'a'**];**

**}**

flag **&=** head**->**is\_terminal**;**

**return** flag**;**

**}**

void remove**(**string s**,**PrefixTree**\*** root**)** **{** // удаление слова

PrefixTree **\***head **=** root**;**

**for(**int i**=**0**;**i**<**s**.**length**();**i**++)** **{**

char c **=** s**[**i**];**

**if(**head**->**next**[**c**-**'a'**]==nullptr)** **{**

**break;**

**}**

head **=** head**->**next**[**c**-**'a'**];**

**}**

head**->**is\_terminal **=** **false;**

**}**

int main**()**

**{**

PrefixTree **\***head **=** **new** PrefixTree**();**//создание структуры

**return** 0**;**

**}**

**Приложение**

1.Таблица времени для длинны слова 10(add):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time(мс) | 0,3 | 1 | 2 | 11 | 22 |
| Size(кол.элем) | 100 | 500 | 1000 | 5000 | 10000 |

2. Таблица времени для длинны слова 100(add):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time(мс) | 2 | 11 | 29 | 112 | 226 |
| Size(кол.элем) | 100 | 500 | 1000 | 5000 | 10000 |

3. Таблица времени для длинны слова 1000(add):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time(мс) | 23 | 113 | 241 | 1091 | 2090 |
| Size(кол.элем) | 100 | 500 | 1000 | 5000 | 10000 |

4. Таблица времени для длинны слова 100(contain/remove):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time(мс) | 0,4 | 3 | 6 | 26 | 53 |
| Size(кол.элем) | 100 | 500 | 1000 | 5000 | 10000 |

5. Таблица времени для длинны слова 1000(contain/remove):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time(мс) | 6 | 25 | 49 | 260 | 538 |
| Size(кол.элем) | 100 | 500 | 1000 | 5000 | 10000 |

\*для каждого из размера данных бралось округленное среднее значение от 50 наборов входных данных.

\*https://github.com/MarselYsup/Itis-aicd/tree/master/semesterTask