UFAM

FT – FACULDADE DE TECNOLOGIA
ENGENHARIA DA COMPUTAÇÃO

JEAN CLEISON BRAGA GUIMARÃES – 21601227

TRABALHO 3: ÁRVORE TRIE/PATRICIA
INSERÇÃO, REMOÇÃO E BUSCA

MANAUS, AM
2019

JEAN CLEISON BRAGA GUIMARÃES

TRABALHO 3: ÁRVORE TRIE/PATRICIA
INSERÇÃO, REMOÇÃO E BUSCA

O trabalho foi solicitado pelo professor de Algoritmos e Estrutura de Dados II, Edson Nascimento para obtenção de nota parcial por parte dos alunos no primeiro semestre de 2019.

MANAUS, AM
2019

Introdução

Em ciência da computação, uma trie, ou árvore de prefixos, é uma estrutura de dados do tipo árvore ordenada, que pode ser usada para armazenar um array associativo em que as chaves são normalmente cadeias de caracteres. Edward Fredkin, o inventor, usa o termo trie (do inglês "retrieval") (recuperação), porque essa estrutura é basicamente usada na recuperação de dados. De acordo com essa etimologia ele é pronunciado ("tree"), embora alguns encoragem o uso de ("try") de modo a diferenciá-lo do termo mais geral tree.

A árvore PATRICIA é uma representação compacta de uma Trie onde os nós que teriam apenas um filho são agrupados nos seus antecessores. É comum que as árvores trie possuam um grupo disperso de chaves, desse modo, muitos nós possuem apenas um descendente. Isto faz com que as Trie tenham um custo grande de espaço.

Uma Trie usa cada uma das partes de uma chave, por vez, para determinar a sub-árvore. Por outro lado, a árvore PATRICIA escolhe um elemento da chave (armazenando a sua posição) para determinar a sub-árvore. Isso remove a necessidade de nós com apenas um descendente.

Implementação

- O programa foi construído para fazer inserção, remoção e busca de árvore trie e inserção e busca de árvore Patricia.

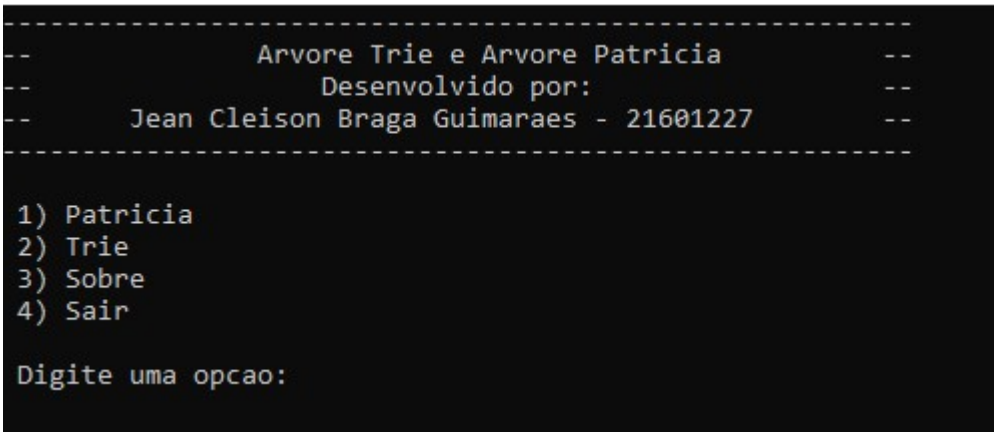
Busca: A busca por uma string em uma arvore PATRICIA é similar a busca em uma Trie, com a diferença de que ao chegar em um nó, é comparado apenas um caractere, contra a comparação de substrings inteiras que acontece na Trie. No pior caso, a complexidade de tempo é O(|s|), onde s é a string procurada.

Inserção: Inserir uma string em uma árvore Patricia é similar a pesquisar por essa string até o ponto onde a busca é encerrada, pois a string não é encontrada na árvore. Se a busca é encerrada em uma aresta, um novo nó é criado nessa aresta. Esse nó armazena a posição do caractere que distingue a chave destino daquela aresta e a chave que se deseja inserir, e tem como filhos o nó que estava na extremidade seguinte da aresta e um novo nó com a parte restante da nova chave. Se a busca for encerrada em um nó, então um nó filho é criado e o restante da nova chave é usado como rótulo para aresta entre os dois. Ambos os casos tem complexidade de tempo de O(|s| + |E|), onde s é a string que será inserida e E é o alfabeto suportado pela árvore.

Remoção: Remover uma string de uma árvore PATRICIA é o oposto da operação de inserção. Primeiro, localiza-se a folha correspondente a string e remove-se ela da árvore. Como o pai terá apenas um filho, os nós pai e irmão do nó removido são agrupados em um único nó. A complexidade de tempo depende diretamente do tempo para remover 2 nós da árvore, se essa remoção for considerada linear, então a complexidade de tempo da operação é O(|s|), onde s é a string que será removida, se essa remoção tiver complexidade O(N), então a complexidade de tempo da operação é O(|s| + N), onde N é o tamanho total de todas as strings armazenadas na árvore.

Execução do Algoritmo

O programa, ao ser executado, está encarregado de imprimir na tela um menu, que por sua vez, irá mostrar as opções disponíveis ao usuário.

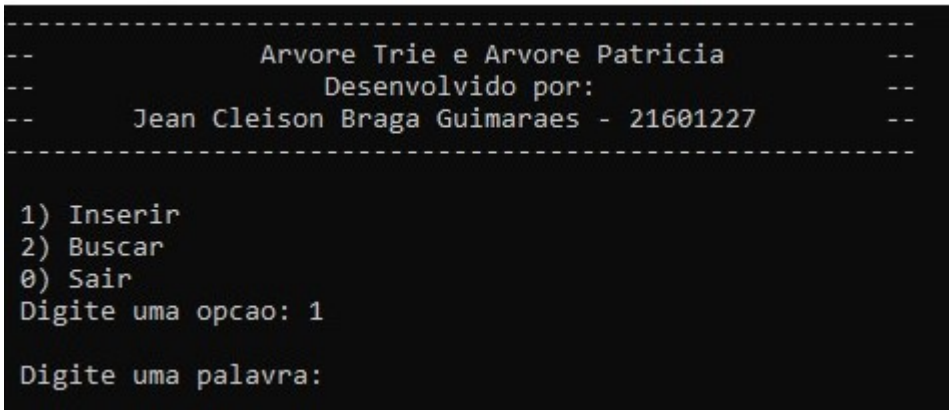


A seguir há a explicação de cada opção:

1 - Patricia:

- Inicia o programa com a construção de uma árvore Patricia.

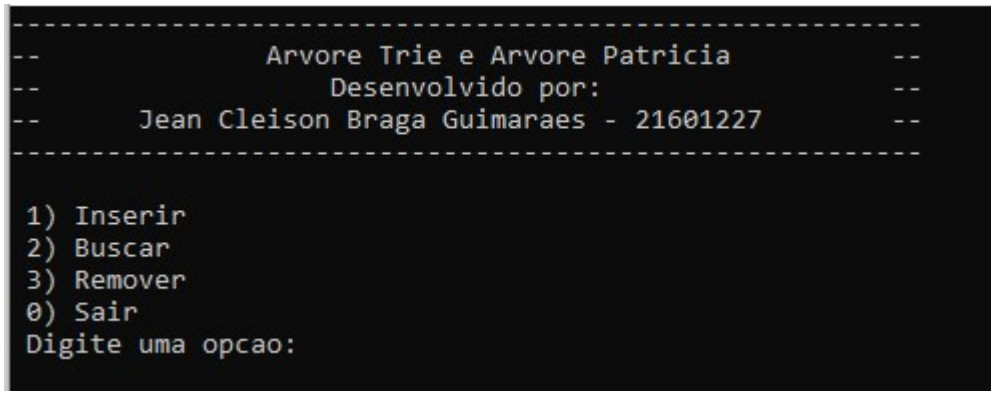
Tela de execução:



2 - Trie:

- Inicia o programa com a construção de uma árvore Trie.

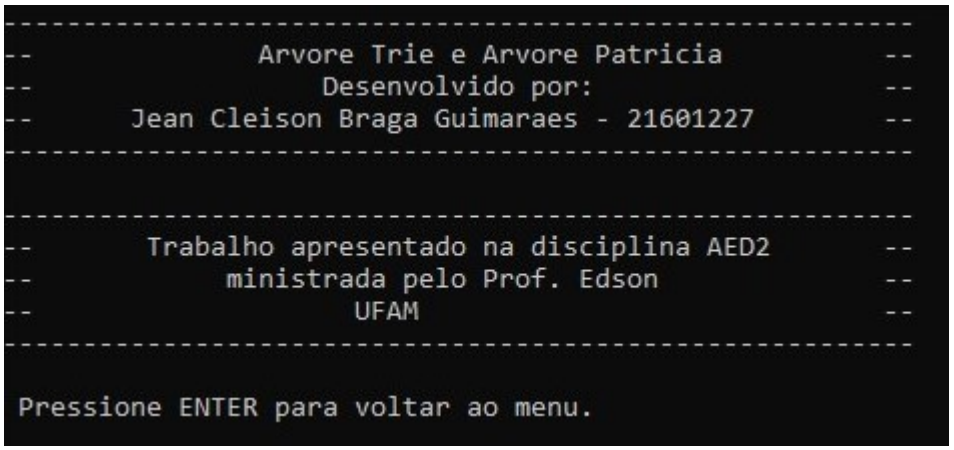
Tela de execução:



3 - Sobre:

- Mostra uma pequena informação sobre o programa.

Tela de execução:



3 - Sair:

- Finaliza o programa.

Estrutura do Algoritmo

Trie

```
17  struct Trie* getNewTrieNode()
18  {
19      struct Trie* node = (struct Trie*)malloc(sizeof(struct Trie));
20      node->isLeaf = 0;
21      int i;
22      for (i = 0; i < CHAR_SIZE; i++)
23          node->character[i] = NULL;
24
25      return node;
26  }
27
28  // Funçao iterativa para inserir uma string na trie
29  void insert(struct Trie* *head, char* str)
30  {
31      // Começa da raiz
32      struct Trie* curr = *head;
33      while (*str)
34      {
35
36          if (curr->character[*str - 'a'] == NULL)
37              curr->character[*str - 'a'] = getNewTrieNode();
38          curr = curr->character[*str - 'a'];
39          str++;
40      }
41
42      // marca que o no eh folha
43      curr->isLeaf = 1;
44  }
45
46  // Funçao iterativa para buscar uma palavra. Retorna 1 se existir
47  int searchTrie(struct Trie* head, char* str)
48  {
49      // return 0 se Trie esta vazia
50      if (head == NULL)
51          return 0;
52
53      struct Trie* curr = head;
54      while (*str)
55      {
56          curr = curr->character[*str - 'a'];
57          if (curr == NULL)
58              return 0;
59          str++;
60      }
61      return curr->isLeaf;
62  }
```

Patricia

```
220
221 static struct ptrie_node* new_node(struct ptrie_node *node,int sibl
222 {
223     struct list_head *list;
224     struct ptrie_node *new_node;
225     if(!node) {
226         new_node = calloc(1,sizeof(struct ptrie_node));
227         if(!new_node)
228             return NULL;
229         INIT_LIST_HEAD(&new_node->next);
230         INIT_LIST_HEAD(&new_node->sibling_list);
231         return new_node;
232     }
233     if(sibling)
234         list = &node->sibling_list;
235     else
236         list = &node->next;
237     new_node = calloc(1,sizeof(struct ptrie_node));
238     if (!new_node) {
239         return NULL;
240     }
241     INIT_LIST_HEAD(&new_node->next);
242     INIT_LIST_HEAD(&new_node->sibling_list);
243     list_add_tail(&new_node->sibling_list,list);
244 return new_node;
245 }
246
247 static int __add_word(struct ptrie_node *node,const char *word)
248 {
249
250     if (node->substr && strlen(word)<=strlen(node->substr)) {
251         strcpy(node->substr,word);
252         return;
253     }
254     if (node->substr) {
255         free(node->substr);
256         node->substr = NULL;
257     }
258     node->substr = calloc(1,strlen(word)+1);
259     if (!node->substr)
260         return ENOMEM;
261     strcpy(node->substr,word);
262 return 0;
263 }
```

Corpo da Função main()

```
1301  int main() {
1302      screen next_screen = HOME;
1303      snprintf(msg, MSG_LEN, "-");
1304      n_keys = 0;
1305
1306      while (next_screen != EXIT) {
1307          switch (next_screen) {
1308          case HOME:
1309              next_screen = home_screen();
1310              break;
1311          case ABOUT:
1312              next_screen = about_screen();
1313              break;
1314          case RUN:
1315              next_screen = run_screen();
1316              break;
1317          case RUNNING:
1318              next_screen = running_screen();
1319              break;
1320          case EXIT:
1321              break;
1322          }
1323      }
1324
1325      snprintf(msg, MSG_LEN, " Saindo da aplicacao");
1326      goodbye();
1327
1328      return 0;
1329  }
1330
```

Referências Bibliográficas

1. https://pt.wikipedia.org/wiki/Trie
2. https://pt.wikipedia.org/wiki/%C3%81rvore_Patricia
3. https://cs.stackexchange.com/questions/63048/what-is-the-difference-between-radix-trees-and-patricia-tries/63060
4. https://www.quora.com/unanswered/Can-you-give-me-a-good-reference-of-a-C-implementation-insert-delete-search-operations-of-Patricia-Trie-Or-suggestions-on-how-to-convert-a-basic-Trie-to-Patricia-Trie