Universidade Federal de Santa Catarina
INE5408-03208A (20221) - Estruturas de Dados
Aluna: Samantha Costa de Sousa (21106156)

Relatório do Projeto II - identificação de prefixos e indexação de dicionários

Para concluir o projeto, utilizei a estrutura de dados Trie, implementada em uma classe para este uso baseado nos algoritmos referenciados pelo professor. Todos os testes passaram com sucesso.

Na **main()**, o que acontece primeiramente é a leitura do nome do arquivo que será lido, a criação de uma variável *file* do tipo “ifstream” e a abertura desse arquivo utilizando *file*. Um objeto da classe Trie é criado, e a leitura do arquivo é iniciada, linha por linha.

```
while (getline(file, line))
{
  size_t end_position = line.find(']') - 1;
  string word = line.substr(1, end_position);
  my_trie.insert(word, i, line.length());
  i += line.length() + 1; // pos da próxima palavra
}
file.close();
```

Esse while adquire as palavras do arquivo, diferenciadas por estarem protegidas por colchetes, e salva na Trie: a palavra, a posição da palavra e o comprimento da linha, como requisitados nos problemas do projeto.

A estrutura da Trie

```
char letter;
struct TrieNode *children[ALPHABET_SIZE];
unsigned long position;
unsigned long length;
bool is_leaf;
```

Trie é uma classe representante da estrutura de dados Trie (árvore de prefixos) que é iniciada com uma raiz *root*, cuja estrutura é um TrieNode. O TrieNode possui a letra, a posição, comprimento (que existem caso aquele nó seja o fim de uma palavra) e o booleano *is_leaf* que diz se esse nó é o fim de uma palavra. Também possui um vetor children de nós do tamanho do alfabeto (todos os nós iniciados como nullptr). O destrutor da classe deleta a *root*, e o destrutor do TrieNode deleta todos seus filhos não-nulos.

Método insert

```
// insere nova palavra
void insert(std::string key, size_t position, size_t length)
{
  struct TrieNode *crawler = root; // percursor

  // para todas letras da palavra
  for (std::size_t i = 0; i < key.length(); i++)
  {
    int index = key[i] - 'a';                          // ajuste ascii, consigo pos letra letra
    if (crawler->children[index] == nullptr)           // não existe ainda
      crawler->children[index] = new TrieNode(key[i]); // salvo a letra
    // a partir do percursor, continua percurso
    crawler = crawler->children[index];
  }
  // crawler agora é a última letra da palavra
  crawler->is_leaf = true;
  crawler->position = position;
  crawler->length = length;
}
```

Chamado durante a leitura do arquivo para inserir palavras, ele começa a percorrer tendo a root como *crawler*:

- Percorre cada letra da palavra, adquirindo o index (com ajuste para a ser 0, b ser 1 respectivamente para todas as letras do alfabeto).
- Se no crawler na posição no array children para aquela palavra não tem um nó ainda, o nó é criado utilizando new;
- O crawler agora é o nó para aquela letra no children do crawler anterior (criado anteriormente se não existia), continuando o percurso.

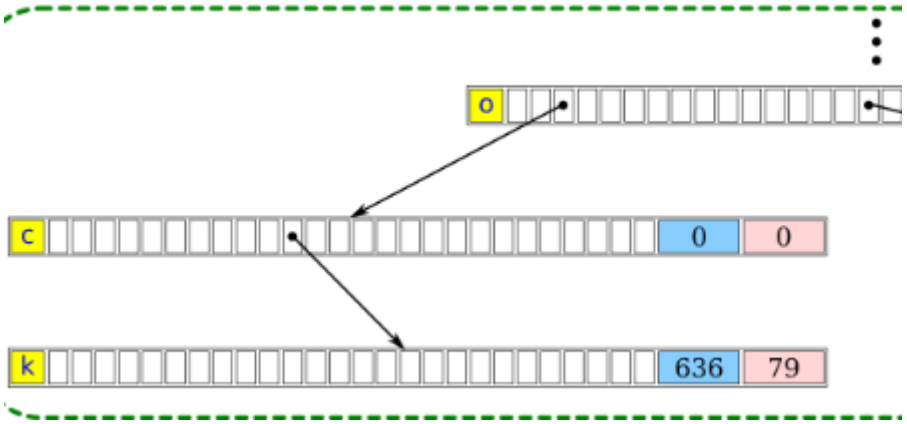

Imagem entregue pelo professor que demonstra o percurso de uma palavra/prefixo de uma letra para outra via o TrieNode e seu vetor children. Cada posição de children é para uma letra, ou seja, se o crawler é este com a letra o e estou lendo a letra c na palavra, o próximo crawler será a posição 2 (referente a c) no children do crawler com o.

- Ao finalizar o for percorrendo a palavra, o último crawler é com certeza o nó que finaliza a palavra. Assim, ele recebe a posição e comprimento entregues ao chamar o método, e a variável is_leaf recebe true.

Primeiro problema: identificação de prefixos

De volta a main(), após ter salvo todas as palavras na Trie com suas respectivas informações, ela inicia um while que lê as entradas até a palavra ser 0, caso seja 0, o while recebe um *break*. Para cada palavra lida, se faz a resolução dos problemas. O primeiro problema pede de quantas palavras aquela palavra é um prefixo. Isso é feito chamando o método **prefix_count** da Trie.

```
int prefixes = my_trie.prefix_count(word);
```

O método prefix_count percorre a Trie de modo semelhante ao insert, iniciando um crawler que é a root; Uma contagem é iniciada com 0 e um for percorre a palavra e a Trie (o exato mesmo for de insert, exceto que ele não aloca novos TrieNode). Se em algum momento o percurso falhar (uma letra que tentei acessar não foi alocada), retorna a contagem. Se isso não acontecer, o for finaliza. Há um teste para saber se ele é o fim de uma palavra, e se sim, a contagem recebe um incremento (uma palavra é prefixo dela mesma). Depois disso, a contagem recebe outro incremento de acordo com o retorno do método **children_count** da Trie.

```
if (crawler->is_leaf)
  count++;
count += children_count(crawler);
```

Chamada de children_count em prefix_count

O método children_count conta, a partir de um nó (crawler), quantos children ele tem no total. Ele faz isso:

- Percorrendo todo o array “children” do nó em um for. Se é nulo, continua o percurso. Se não:
- Se é leaf (fim de uma palavra), incremento a contagem que será retornada. Sendo leaf ou não, já que não é um ponteiro nulo, faz a chamada de children_count novamente com aquele nó (chamada recursiva)

```
int
children_count(TrieNode *crawler)
{
  int count = 0;
  for (int i = 0; i < ALPHABET_SIZE; i++)
  {
    if (crawler->children[i] != nullptr)
    {
      if (crawler->children[i]->is_leaf)
      {
        count++;
      }
      count += children_count(crawler->children[i]);
    }
  }
  return count;
}
```

A children_count, chamando ela mesma ao encontrar um filho não-nulo do crawler;

No fim, finalizando o funcionamento de children_count, a prefix_count recebe o incremento correto de quantas palavras são finalizadas abaixo do último nó do prefixo, e retorna para main a solução do problema, que faz o print de 'P is prefix of N words', onde P é o nome da palavra e N é a quantidade de palavras) ou a mensagem 'P is not prefix', caso o retorno da prefix_count seja 0.

Segundo problema: indexação de arquivo de dicionário

O segundo problema pede, se a palavra está na Trie (como uma palavra do dicionário), que seja printado sua posição no arquivo e o comprimento da linha onde essa palavra se encontra. Ou seja, as informações salvas no TrieNode que possui is_leaf como true. Para resolver isso, foi criada uma struct de nome *search_return,* que possui 1. um booleano que responde se a palavra está na Trie ou não, 2. O TrieNode da última letra da palavra, que possui as informações de posição e comprimento. A main chama o método **search** da Trie.

```
search_return sr = my_trie.search(word);
```

```
search_return search(std::string key)
{
  struct TrieNode *crawler = root; // percursor
  struct search_return returning;
```

O método search:

- Percorre a palavra pela Trie de modo semelhante ao método insert e prefix_count, mas, ao encontrar um children nulo no caminho, coloca o booleano do search_return como false e retorna.
- Caso consiga finalizar o for, preenche a struct search_return com o booleano sendo is_leaf e o nó sendo o último crawler, e retorna ela.

De volta à main, ela checa a estrutura retornada. Se o booleano for true, faz um print com a posição e comprimento (que estão no TrieNode na struct retornada), solucionando o problema.

```
  if (sr.search_result)
  {
    TrieNode *ll = sr.last_letter;
    cout << word << " is at (" << ll->position << "," << ll->length << ")" << endl;
  }
}
```

A execução termina após os dois problemas terem sido solucionados para todas as palavras testadas, finalizando com sucesso.

Conclusão (dificuldades e referências)

O projeto foi relativamente tranquilo, visto que a lógica de organização do dicionário na Trie é toda exemplificada na página do moodle. Tive um pouco de problema com retorno na search (porque queria retornar duas coisas, então preferi criar uma struct para isso ao invés de modificar ponteiros), e a parte de contar quantos “filhos” existem abaixo de nó também foi um pouco confusa no começo, já que utilizei recursividade, mas também ficou tranquilo de entender ao observar as imagens do moodle e referências na internet.

Referências utilizadas:

https://www.geeksforgeeks.org/trie-insert-and-search/
https://www.geeksforgeeks.org/counting-number-words-trie/
https://kalkicode.com/count-the-number-of-words-with-given-prefix-using-trie
https://www.tutorialspoint.com/read-file-line-by-line-using-cplusplus