# Relatório - EP2 AED2

Lourenço Henrique Moinheiro Martins Sborz Bogo - 11208005

Nesse EP implementei 9 symbol tables , cada uma usando uma estrutura de dado diferente.

**Como usar o EP**

Para usar o EP, deve-se usar `./main.out <arquivo_de_texto> <estrutura_de_dado>`, por exemplo: `./main.out teste.txt RN`.

Os tipos de estrutura são:

- VD
- VO
- LD
- LO
- AB
- TR
- A23
- RN
- HS

Além disso, pode-se usar alguma combinação dos seguintes argumentos extras, para uma execução diferente:

- Ao chamar o EP com `s` (ex: `./main.out teste.txt RN s`), ele irá mostrar as todas as inserções feitas ao carregar o arquivo `teste.txt`.

- Ao chamar o EP com `t` (ex: `./main.out teste.txt RN t`), ele irá mostrar os tempos de cada operação feita durante o modo iterativo.

- Ao chamar o EP com `e` (ex: `./main.out teste.txt RN e`), ele não irá entrar no modo iterativo e irá inserir todas as palavras do arquivo, depois realizar todas as operações em todas as chaves, calcular e printar o tempo médio e total de cada operação e terminar.

## Escolhas de Implementação

**Vetores:** Aqui não foi usado nada de especial. No caso do vetor ordenado utilizei uma implementação de busca binária que caso não encontre um certo elemento, ela devolve o índice onde esse elemento deve ser inserido.

**Listas:** Aqui eu optei por usar nodos com ponteiros para o anterior e, também, um nodo como cabeça e um como cauda (lista duplamente ligada com cabeça). Decidi fazer dessa maneira pois é mais rápido para achar os elementos, já que percorro a tabela pelos dois lados. Na lista ordenada as buscas são bem mais rápidas, pois caso o ponteiro que percorre pela esquerda ache um elemento maior que o q estou procurando, ou o ponteiro da direita ache um menor, pode-se concluir que o elemento que procuro não está na lista e terminamos a busca.

**Árvore de Busca Binária:** Aqui foi feita uma árvore de busca binária comum, não balanceada.

**Treap:** Usei uma implementação de treap com uma remoção um pouco diferente. Para remover um elemento, com rotações vou levando-o até folha e ao chegar eu simplesmente deleto ele, mantendo todas as propriedades de ABB. A propriedade de treap se mantém caso as rotações sejam feitas para o lado certo, o que é garantido na minha implementação.

**Árvore 2-3:** Minha implementação da 2-3 foi feita baseada em uma implementação da Princeton University.

**Árvore Rubro Negra:** Optei por fazer a implementação do Sedgewick da Árvore Rubro Negra.

**Hash Table:** Utilizei a `Rolling Hash` como função de hash e optei pelo método de encadeamento separado para tratar as colisões. O tamanho do vetor está setado para `MAXHASH` que é uma constante definida na `main` (por padrão seu valor é 10000).

**Experimentos**

Todas as operações foram testadas com o valgrind e não houve nenhum erro/memory leak em qualquer symbol table.

Com os experimentos pude concluir na prática algo que já conhecia na teoria. A diferença de tempo das symbol tables que têm operações quadráticas para as lineares, é tão gritante quanto a diferença das lineares para as logarítmicas. A tabela feita a partir de uma hash table é praticamente inutilizável para as operações `rank` e `seleciona`, e as duas desordenadas (vetor e lista), também são muito piores que as outras. O vetor ordenado e a lista ordenada têm algumas operações muito rápidas mas em média perdem para as árvores que têm operações em $O(\log N)$, com exceção da árvore de busca binária, que pode ser linear em alguns casos.

Testei o EP para casos pequenos (100 - 10000) e para casos grandes (100000 - 1000000). Nos casos menores, o vetor ordenado foi o que se saiu melhor, pois o seu tempo para calcular `rank` é $O(1)$, e a operação mais demorada é o `seleciona`, que, no caso dessa estrutura, utiliza a função `rank`. Já para casos maiores, as funções `remove` e `insere` começam a exigir muito, fazendo com que as árvores, que as executam em tempo $O(\log N)$, sejam mais rápidas que o vetor ordenado, que as executa em $O(N)$.

**OBSERVAÇÃO:** Todas as árvores, com exceção da 2-3 guardam em cada nó o tamanho da sub-árvore cuja raiz é esse nó, otimizando **muito** as operações `rank` e `seleciona`. Na 2-3 eu tentei fazer isso, mas não consegui, logo ela acaba sendo pior que as outras nessas operações.