Algoritmos e Estruturas de Dados II

EP1: Tabelas de Símbolos Ordenadas

Profº Carlos Ferreira
Atenágoras Souza Silva (nºUSP: 5447262)
athenagoras@gmail.com

1 Introdução

Neste Trabalho, foram implementadas as seguintes tabelas de símbolo ordenadas (TSO)

- Vetor Desordenado (VD)
- Vetor Ordenado (VO)
- Lista Desordenada (LD)
- Lista Ordenada (LO)
- Árvore Binária de Busca (AB)
- Árvore Heap, Treap (TR)

Deveriam ser implementadas também as árvores 2-3 e Tabelas Hash, mas gastei muito tempo tentando implementar o algoritmo de inserir da árvore 2-3 sem sucesso, o que impossibilitou completar todas as estruturas em tempo hábil.

2 Compilação e Modo de uso

Para compilar, basta digitar no prompt:

$ make

O nome do executável gerado é tso. Para rodar o programa com um livro e uma estrutura, é necessário digitar:

$ ./tso livro TSO,

onde livro é o arquivo contendo o livro ou o texto a ser utilizado e TSO é a estrutura a ser utilizada, podendo ser VD, VO, LD, LO, AB ou TR.

O script run.sh, se executado ($ ./run.sh), permite rodar o código com todos os livros textados e também todas as estruturas implementadas

3 Descrição das estruturas para tabela de símbolos

Apresenta-se uma breve descrição teórica de cada estrutura de dados.

Este autor não obteve sucesso na implementação da Árvore 2-3, e gastou tanto tempo nisso que não foi possível implementar as estruturas Árvore Rubro-Negra nem Tabela Hash. Árvore 2-3 foi implementada parcialmente, mas sem resultados satisfatórios devido a um bug de difícil resolução.

3.1 Vetor Desordenadado

Em um vetor desordenado, o custo de inserção é, teoricamente, sempre constante, porque pode o elemento pode ser inserido sempre na última posição do vetor.

Entretanto, como esta é uma tabela de símbolos de frequência (e mesmo que fosse de outro tipo), antes de inserir por último, é necessário verificar se o objeto, no caso, a palavra, já foi inserida antes, realizando-se para isto uma busca, o que neste caso, pode penalizar a performance se os elementos forem inseridos em ordem aleatória.

3.2 Vetor Ordenado

Em um vetor ordenadado, os elementos devem ser inseridos de forma ordenada, mesmo que a ordem dos elementos a inserir seja aleatória.

Isto faz com que seja necessário deslocar parte dos elementos já inseridos para inserir um novo elemento entre eles, o que é um processo custoso. Em compensação, obtém-se resultados melhores na busca, já que esta pode ser realizada em O(log(N)) (busca binária).

3.3 Lista Desordenada

Em uma lista desordenada, o elemento é inserido sempre no fim da fila. O tempo necessário para encontrar o elemento inserido é linear.

3.4 Lista Ordenada

Em uma lista ordenadada, os elementos são inseridos sempre de maneira ordenada. A vantagem em relação a um vetor é que não é necessário fazer deslocamentos, sendo um processo mais barato de ordenação. O tempo para realizar uma busca também é linear, embora possivelmente mais rápido que uma lista desordenada.

3.5 Árvore Binária de Busca

A árvore Binária de Busca é uma estrutura de dados que permite inserção e busca de maneira muito rápida (complexidade O(log(N)) desde que a inserçãoseja de elementos aleatórios. Se os elementos a serem inseridos apresentam uma forte ordenação, a estrutra de dados pode se tornar mais parecida com uma lista, na prática, sendo demorada tanto a inserção quanto a busca.

Neste pior caso, a complexidade torna-se linear.

3.6 Árvore Heap (Treap)

A árvore Heap utiliza Heaps e rotação para manter a árvore sempre balanceada (evita que a árvore se torne parecida com uma lista no caso de inserção de elementos com forte ordenação), através do sorteio de um número auxiliar que auxilia em seu balanceamento obedecendo max heap ou mini heap.

Isto faz com que a inserção seja mais cara que a da Árvore Binária de Busca para o caso em que os elementos a serem inseridos são aleatórios, ou apresentam pouca ordenação, já que o algoritmo heap e a rotação apresentam um custo computacional a mais, mas mantém a árvore balanceada e eficiente mesmo no pior caso.

3.7 Árvore 2-3

A árvore 2-3, como o próprio nome sugere, não é uma árvore binária, já que pode ter até 3 nós.

Ela é uma árvore que não precisa ser rotacionada para manter o balanceamento, já que por conta do próprio algoritmo, ela não se desbalanceia, permitindo uma manutenção de pouco custo.

3.8 Árvore Rubro-Negra

A exemplo da árvore 2-3, a árvore rubro-negra também é uma árvore B, e tem como principal propósito manter-se balanceada.

Deste modo, também obtém-se um bom pior caso, de complexidade O(log(N)).

3.9 Tabelas Hash

Em uma tabela Hash, utilizamos algum algoritmo para associar o elemento inserido a algum número do vetor onde são inseridos os elementos. Se o algoritmo for bom, não há colisões, e obtém-se inserção sem a necessidade de deslocamentos. O tempo de busca, no caso em que não há colisões é constante.

4 Métodos

4.1 Interface

Para permitir maior automação, a interface sugerida pelos monitores da disciplina foi substituida por outra criada por este autor, de modo a automatizar os testes.

Todas as palavras utilizadas nos métodos devolve(), remove() e rank() de todas as estruturas foram sorteadas antes mesmo da inserção do conteúdo do livro nestas.

O método seleciona() selecionou sempre a última palavra inserida na estrutura, para analisar como cada estrutura lida com o pior caso na utilização deste método.

O método insere() principal de cada estrutura insere todo o conteúdo do livro de uma vez, recorrendo a métodos auxiliares para inserir cada palavra de acordo com o tipo de estrutura em que o livro é armazenado.

4.2 Funcionamento geral do programa

Uma vez que o programa tem os argumentos lidos corretamente (o livro ou o texto a ser incluido na TSO, bem como a própria TSO escolhida), o fluxo do programa consiste em, nesta ordem:

1. Extrair as palavras do livro
2. Chamar a "interface" para incluir as palavras do livro na estrutura, e executar os testes nos métodos, recebendo os resultados (TestesTSO())
3. Imprimir o resultado dos testes em arquivo com a sigla da estrutura
4. Liberar a memória utilizada

Como o código que chama as funções para realizar este fluxo é pequeno, pode ser mostrado aqui:

```
int main(int argc, char** argv){
  // Processando entrada
  if (argc < 3)
    return erroPadrao();
  entrada *In = analIN(argv[1], argv[2]);
  if (!In)
    return erroPadrao();

  // Carregando livro na memória
  long n;
  palavra *livro = ExtraiPalavras(In->f, &n);
  fclose(In->f); // fechando arquivo

  tabTempo resultado; // para armazenar resultados dos testes
  /* Realiza os testes com a estrutura selecionada */
  resultado = TestesTSO(livro, n, In->Ttso);
  printTeste(resultado, argv[1], n);
  // Liberando memória
  free(In);
  Delpalavras(livro, n); // libera memória utilizada por array de palavras livro
  DoomDay(resultado);

  return 0;
}
```

Para a análise do resultado, este é impresso em um arquivo .csv, podendo ser tabelado ou dele ser extraído um gráfico a partir de outras ferramentas.

Para melhor análise do resultado, além do tempo total para executar todas as operações, foram incluidos os tempos para realizar cada um dos métodos (insere(), devolve(), remove(), rank() e seleciona()), permitindo discutir em que operações cada estrutura apresenta vantagens e desvantagens.

O programa contou com estruturas e funções em cabeçalhos auxiliares para representar cada palavra inserida nas TSOs, bem como para processar os métodos mensurados em cada uma delas, como por exemplo, medir o tempo.

As TSOs foram implementadas em classes.

4.3 Estrutura palavra

A estrutura palavra foi criada para armazenar cada palavra dos livros em um vetor de palavras, bem como para inserí-las nas TSOs.

Foram implemantadas funções de gerenciamento e remoção da estrutura, sorteio de palavras em um vetor e comparação.

```
typedef struct{
  char *String;
} palavra;

/* ExtraiPalavras: Extrai palavras de um livro até encontrar o caractere nulo
   parâmetros: FILE: *fp     -> ponteiro para o arquivo que contém o livro
               long : *n      -> ponteiro para variável a contar o número de palavras extraídas
   retorno: Ponteiro para as palavras extraídas da string */
palavra *ExtraiPalavras(FILE *fp, long *n);
palavra *sorteiaPalavra(palavra* palavras, long n);
void Delpalavra(palavra *p); // Apaga string dentro de palavra
void Delpalavras(palavra *p, long n); // Apaga array de palavras
// retorna 1 se p1 > p2
int maior(palavra p1, palavra p2);
// retorna 1 se p1 >= p2
int maiorigual(palavra p1, palavra p2);
// retorna 1 se p1 == p2
int igual(palavra p1, palavra p2);
// retorna 1 se p1 < p2
int menor(palavra p1, palavra p2);
// retorna 1 se p1 <= p2
int menorigual(palavra p1, palavra p2);
```

4.4 Estrutura infoTSO

A estrutura infoTSO foi criada para inserção das palavras em cada TSO. As TSOs, basicamente, são formadas por um membro do tipo infoTSO e outras variáveis relevantes para sua manutenção.

infoTSO armazena a palavra oriunda do livro junto a frequência com a qual ela ocorre e a última posição em que ela foi encontrada no texto. infoTSO é a informação sobre a palavra.

No cabeçalho em que se apresenta infoTSO, a estrutura “palavra” foi redefinida em “Chave” com um typedef por simplicidade.

```
typedef palavra Chave;

typedef struct{
  Chave chave;
  long freq; // frequência com que a chave é encontrada no texto de origem
  long pos; // posição da chave na última inserção a partir do texto
} infoTSO; // Estrutura com a informação de um nó de informação da TSO

infoTSO *novaInfo(); // cria nova informação TSO
bool vazia(infoTSO *info); // verifica se está vazia uma estrutura já criada e não deletada
```

```
void insereInfo(Chave chave, long pos, infoTSO *info);
infoTSO *infoCopia(infoTSO *info); // faz uma cópia da informação
void DelInfo(infoTSO *info); // Apaga informação TSO
```

4.5 Estrutura tabTempo

Esta estrutura foi criada para armazenar os resultados das medições de cada método nas TSOs.

Foram criados também protótipos de função construidos para cada TSO para que possam ser auferidos os tempos de processamento de cada método mensurado.

```
// Estrutura para armazenar os resultados das medições
typedef struct{
  double Tinsere, Tdevolve, Tremove, Trank, Tseleciona, Ttotal;
  long Ttam; // tamanho da estrutura em número de palavras armazenadas
  char *nomeTSO;
} tabTempo;

// Cria Variável de medição de tempo dos testes da experiência para um tipo de TSO
tabTempo NovoTempo(char *nomeTSO, long Ttam, double Tinsere, double Tdevolve, double Tremove, double
// Imprime Testes em Arquivo
void printTeste(tabTempo t, char *nomearq, long n);
// Apaga variável do tipo TabTempo
void DoomDay(tabTempo var);
/*TestesTSO: */
tabTempo TestesTSO(palavra *livro, long n, char *nomeTSO);

/* Funções de teste de performancce para cada operação das TSO */
template <class TSO>
double insere(palavra *livro, long n, TSO *tso);
template <class TSO>
double devolve(Chave chave, TSO *tso);
template <class TSO>
double remove(Chave chave, TSO *tso);
template <class TSO>
double rank(Chave chave, TSO *tso);
template <class TSO>
double seleciona(TSO *tso);
template <class TSO>
long tam(TSO *tso);
```

4.6 Classe VD

Nesta classe implementou-se a Tabela de Símbolos Ordenada (TSO) Vetor Desordenado (VD).

Para fazer a busca de uma palavra no método devolve(), cria-se internamente um vetor auxiliar ordenado com quicksort(), e é feita uma busca binária.

```
class VD{
private:
  // variável auxiliar para rank() e seleciona(). Liberar após o uso
  infoTSO **aux;
  infoTSO **info;
  long N; // tamanho do vetor
  // Funções auxiliares para o algoritmo Qsort()
  void Troca(infoTSO **info1, infoTSO ** info2);
  void Particiona(long ini, long fim, long *i, long *j);
  /* Qsort() será usada para ordenação DEPOIS que todas as palavras forem inseridas na em aux. Depen
  void Qsort(long ini, long fim);
```

```
  /* bb(): Busca binária
     Parâmetro:
     Chave chave: Palavra a ser procurada na tabela
     Retorno: posição da chave em info, -1 para não encontrado */
  long bb(Chave chave);
  void criaAux(){
    aux = (infoTSO **) malloc(sizeof(infoTSO *) * N);
    long i;
    for (i = 0; i < N; i++)
      aux[i] = info[i];
  }
  void destroiAux(){
    free(aux);
    aux = NULL;
  }

public:
  VD(){
    info = NULL;
    aux = NULL;
    N = 0;
  }
  ~VD(){
    long i;
    for (i = 0; i < N; i++){
      DelInfo(info[i]);
      // free(info[i]);
    }
    free(info);
  }

  bool vazia();
  //infoTSO *busca(Chave chave);
  /* busca(): busca linearmente para achar a chave. Para ajudar insere()
   Parâmetros:
   Chave chave: Palavra a buscar
   Retorno: Ponteiro para a infoTSO* encontrada, ou NULL em caso de fracasso
  infoTSO /*VD::*/ *busca(Chave chave){
  long i;
  for (i = 0; i < N; i++)
    if (!strcmp(chave.String, info[i]->chave.String))
      return info[i];
  return NULL;
}

  /* Busca(): busca linearmente para achar a chave. Para ajudar remove() e devolve()
   Parâmetros:
   Chave chave: Palavra a buscar
   Retorno: (long) Posição da chave em info. -1 em caso de fracasso */
  long Busca(Chave chave);
  void insere(Chave *chave, long n);
  long devolve(Chave chave); // devolve a frequência da chave no texto em que foi colhida
  void remove(Chave chave);
  long rank(Chave chave);
  Chave seleciona(long k);
  long tam(){
```

```
    return N;
  }
};
```

4.7 Classe VO

Nesta classe foi implementada a TSO Vetor Ordenado.

```
class VO{
private:
  infoTSO **info;
  long N; // tamanho do vetor
  /* antes(): Insere chave no vetor na posição posV, e desloca todos os elementos posteriores para f
     Parâmetros:
     Chave chave: chave a ser inserida
     long posT  : posição em que a chave foi encontrada no texto de origem
     long posV  : Posição em que a chave vai ser inserida no vetor */
  void antes(Chave chave, long posT, long posV);
  // insere chave depois da posição posV e desloca todos os elementos posteriores para frente, menos
  void depois(Chave, long posT, long posV);
public:
  VO(){
    info = NULL;
    N = 0;
  }
  ~VO(){
    long i;
    for (i = 0; i < N; i++)
    DelInfo(info[i]);
    //free(info);
  }
  bool vazia();
  void insere(Chave *chave, long n);
  long bb(Chave chave); // busca binária
  long devolve(Chave chave); // devolve a frequência da chave no texto em que foi colhida
  void remove(Chave chave);
  long rank(Chave chave);
  Chave seleciona(long k);
  long tam(){
    return N;
  }
};
```

4.8 Classe Nolst

Esta classe implementa a estrutura Nó, que é utilizado nas TSOs Lista Desordenada (LD) e Lista Ordenada (LO).

```
// Classe nó para as listas desordenada e ordenada
class Nolst{
public:
  infoTSO *info;
  Nolst *ant, *prox;
  Nolst(infoTSO *Info);
  Nolst(infoTSO *Info, Nolst *an, Nolst *pr);
  ~Nolst();
};
```

4.9 Classe LD

Esta classe implementa a TSO Lista Desordenada.

Para a utilização do método rank(), apartir da lista desordenada, é criada uma lista auxiliar ordenada.

```
// Lista Desordenada
class LD{
public:
  LD(){
    ini = NULL;
    fim = NULL;
    N = 0;
  }
  ~LD();
  bool vazia();
  void insere(Chave *chave, long n);
  long devolve(Chave chave); // devolve a frequência da palavra no texto
  void remove(Chave chave);
  long rank(Chave chave);
  Chave seleciona(long k);
  long tam();
  Nolst /*LD::*/ *INI(){
    return ini;
  }

  Nolst /*LD::*/ *FIM(){
    return fim;
  }
  // Nolst *busca(Chave chave);
private:
  Nolst *ini, *fim;
  long N; // tamanho da TSO em palavras
  // void antes(Chave chave, long pos, Nolst *no);
  // void depois(Chave chave, long pos, Nolst *no);
  // auxílio às funções inserir(), devolve() e remove()
  Nolst /*LD::*/ *busca(Chave chave){
    Nolst *p;
    for (p = ini; p; p = p->prox){
      if (igual(chave, p->info->chave))
        return p;
    }
    return NULL;
  }
};
```

4.10 Classe LO

Esta classe implementa a TSO Lista Ordenada.

```
// Lista Desordenada
class LO{
public:
  LO(){
    ini = NULL;
    fim = NULL;
    N = 0;
  }
```

```
  // Cria lista ordenada a partir de lista desordenada
  LO(LD *l);
  ~LO();
  void insere(Chave *chave, long n);
  long devolve(Chave chave); // devolve a frequência da palavra no texto
  void remove(Chave chave);
  long rank(Chave chave);
  Chave seleciona(long k);
  long tam();
  Nolst /*LO::*/ *INI(){
    return ini;
  }

  Nolst /*LO::*/ *FIM(){
    return fim;
  }

  // Nolst *loBusca(Chave chave);
private:
  Nolst *ini, *fim;
  long N; // tamanho da TSO em palavras
  // Auxílio de inserir()
  void antes(Chave chave, long pos, Nolst *no);
  void depois(Chave chave, long pos, Nolst *no);
  // auxílio às funções devolve() e remove()
  Nolst /* LO:: */ *loBusca(Chave chave){
    Nolst *p;
    for (p = ini; p; p = p->prox){
      if (igual(chave, p->info->chave))
        return p;
    }
    return NULL;
  }
};
```

4.11 Classe noArv

Esta classe implementa o nó utilizados nas TSO que são árvores binárias, como a própria árvore binária, a Treap e a árvore rubro-negra (porém esta última não foi implementada devido ao tempo dedicado a tentar implementar sem sucesso a árvore 2-3).

Esta classe tem funções de construção de nós para cada tipo de árvore implementada.

```
// A idéia dessa classe é ser um nó genérico que serve para todo o tipo de árvore Binária (ABB, RB,
class noArv{
public:
  noArv *esq, *dir;
  infoTSO *info;
  char cor; // caso a árvore seja rubro-negra
  int b; // balanceamento da árvore, se necessário
  long prioridade; // Para Treaps. prioridade é um número aleatório utilizado para manter o balancea
  long N; // nº de nós das subárvores que compoem o nó incluindo ele próprio
  // etc
  noArv(infoTSO *Info, long n = 1){ // Criação para árvore binária de Busca
    info = Info;
    cor = '\0';
    b = 0;
```

```
    esq = dir = NULL;
    N = n;
  }

  noArv(infoTSO *Info, long n, long Prioridade){
    info = Info;
    cor = '\0';
    b = 0;
    esq = dir = NULL;
    N = n;
    prioridade = Prioridade;
  }

  ~noArv(){
    DelInfo(info);
    info = NULL;
    N = 0;
    esq = dir = NULL;
    b = 0;
    cor = '\0';
    prioridade = 0;
  }
};
```

4.12 Classe AB

Esta classe implementa a TSO Árvore Binária de Busca (AB).

```
// Classe de Árvore Binária
class AB{
public:
  AB(){
    //h = 0;
    N = 0;
    raiz = NULL;
  }
  ~AB();
  void insere(Chave *chave, long n);
  void insere(Chave chave, long pos);
  long devolve(Chave chave);
  void remove(Chave chave);
  long rank(Chave chave);
  long tam();
  Chave seleciona(long k);
  long tamanhoSub();
  Chave min();
  void deleteMin();
  void Destroi();
private:
  // dados
  noArv *raiz;
  long N; // nº nós
  // long h; // altura da árvore

  // funções auxiliares
  long tamanhoSub(noArv *no);
```

```
  long Devolve(noArv *no, Chave chave);
  noArv *Insere(noArv *no, Chave chave, long pos);
  noArv *Seleciona(noArv *no, long k);
  long rank(Chave chave, noArv *no);
  noArv *min(noArv * no);
  noArv *deleteMin(noArv *no);
  noArv *remove(noArv *no, Chave chave);
  void destroi(noArv *no);
};
```

4.13 Classe TR

Esta classe implementa a TSO Árvore Heap, ou Treap (TR).

```
// Classe de TREAPs
/* Treaps são Árvores binárias que utilizam um número de prioridade definido aleatoriamente em cada
*/

class TR{
public:
  TR();
  ~TR();
  void insere(Chave *chave, long n);
  void insere(Chave chave, long pos);
  long devolve(Chave chave);
  void remove(Chave chave);
  long rank(Chave chave);
  Chave seleciona(long k);
  long tamanhoSub();
  Chave min();
  long tam();
  void deleteMin();
  void Destroi();
private:
  // Dados
  noArv *raiz;
  long N; // nº de nós
  // Funções auxiliares
  long tamanhoSub(noArv *no);
  long devolve(noArv *no, Chave chave);
  noArv *insere(noArv *no, Chave chave, long pos);
  noArv *seleciona(noArv *no, long k);
  long rank(Chave chave, noArv *no);
  noArv *min(noArv * no);
  noArv *deleteMin(noArv *no);
  noArv *remove(noArv *no, Chave chave);
  void destroi(noArv *no);
  noArv *rodaDir(noArv *no);
  noArv *rodaEsq(noArv *no);
};
```

4.14 Classe A23

Esta classe não foi completamente implementada, e o pouco que foi está errado. Os esforços de implementação dessa TSO, que é a árvore 2-3 são apresentados.

```
/* Árvores 2-3 podem ter 2 ou três nós, eventualmente e já são balanceadas por natureza
   - Quando têm 2 nós (esquerda e direita) possuem uma chave, e obedecem as regras de árvores binári
   - Quando têm 3 nós, (esquerda, centro, direita) possui chaves esquerda e direita
     + As regras são semelhantes às das árvores binárias, mas os filhos esquerdos são menores que a
     + O filho do centro está entre a chave esquerda e a chave direita;
     + O filho direito é maior que o a chave direta
   - As inclusões podem envolver tornar um nó binário em um ternário (com 3 filhos)
   - As remoções podem envolver um desmembramento, e um nó ternário torna-se binário (com 2 filhos) */

class No{
public:
  infoTSO *infoEsq, *infoDir; // armazenar chaves esquerda e direita, eventualmente
  int nchaves; // nº de chaves (determina se é um nó binário ou ternário)
  long N; // Total de nós na subárvore
  No *pai; // para tornar mais fácil o algoritmo de subida recursiva do nó a ser incluido
  No *esq, *centro, *dir;
  No(infoTSO *info, long n, No *pai);
  No(infoTSO *info, long n, No *Esq, No *Dir, No *Pai);
  No(No* no); // construtor de cópia
  ~No();
};

class A23{
public:
  A23();
  ~A23();
  void insere(Chave *chave, long n);
  void insere(Chave chave, long pos);
  No *busca(No *no, Chave chave);
  long devolve(Chave chave);
  void remove(Chave chave);
  long rank(Chave chave);
  Chave seleciona(long k);
  long tam();
private:
  No *raiz;
  long N; // nº nós
  // long h; // altura da árvore
  long tamSub(No *no);
  void insere(No *no, No* novo);
  No *buscaPaiDoNulo(No *no, Chave chave, No *pai);
  No *quebraNo(No *no, No* novo);
  void destroi(No *no);
};
```

5 Análise e Resultados

Tabela 1: Tempos de execução dos métodos do Vetor Desorganizado para cada livro (s)

Nome do arquivo	TSO	Total de palavras no arquivo	Palavras armazenadas na TSO	Tempo insere()	Tempo devolve()	Tempo remove()	Tempo rank()	Tempo seleciona()	Tempo Total
criticaeconomiapoliticaIngles.txt	VD	87693	6496	5.645480e-01	1.000000e-05	4.700000e-05	2.477000e-03	2.511000e-03	5.695560e-01
lusiadasIngles.txt	VD	171099	14460	2.685672e+00	2.800000e-05	4.500000e-05	6.119000e-03	6.023000e-03	2.697870e+00
manifestocomunistaIngles.txt	VD	14627	2624	6.421700e-02	1.000000e-06	9.000000e-06	8.660000e-04	8.340000e-04	6.591900e-02
mariliadedirceo.txt	VD	26389	4957	2.010080e-01	2.900000e-05	1.500000e-05	1.832000e-03	1.850000e-03	2.047480e-01
ocapitalGrego.txt	VD	3717	986	9.322000e-03	6.000000e-06	4.000000e-06	2.880000e-04	2.860000e-04	9.908000e-03
oslusiadas.txt	VD	63475	9042	7.243360e-01	2.000000e-06	2.800000e-05	3.617000e-03	3.590000e-03	7.315470e-01
parodiaLusiadas.txt	VD	10589	2881	7.213800e-02	1.000000e-06	1.200000e-05	1.019000e-03	9.690000e-04	7.412800e-02
sherlock.txt	VD	108833	8067	8.071270e-01	6.000000e-06	0.000000e+00	3.093000e-03	3.083000e-03	8.133150e-01

Tabela 2: Tempos de execução dos métodos do Vetor Organizado para cada livro (s)

Nome do arquivo	TSO	Total de palavras no arquivo	Palavras armazenadas na TSO	Tempo insere()	Tempo devolve()	Tempo remove()	Tempo rank()	Tempo seleciona()	Tempo Total
criticaeconomiapoliticaIngles.txt	VO	87693	6496	5.440711e+00	1.000000e-06	1.000000e-05	6.600000e-05	0.000000e+00	5.440779e+00
lusiadasIngles.txt	VO	171099	14460	2.909146e+01	1.000000e-06	2.900000e-05	1.530000e-04	1.000000e-06	2.909162e+01
manifestocomunistaIngles.txt	VO	14627	2624	3.755810e-01	1.000000e-06	4.000000e-06	1.100000e-05	1.000000e-06	3.755950e-01
mariliadedirceo.txt	VO	26389	4957	1.227298e+00	1.000000e-06	7.000000e-06	2.500000e-05	0.000000e+00	1.227325e+00
ocapitalGrego.txt	VO	3717	986	3.561000e-02	1.000000e-06	2.000000e-06	9.000000e-06	1.000000e-06	3.562200e-02
oslusiadas.txt	VO	63475	9042	6.246481e+00	2.000000e-06	1.700000e-05	6.000000e-05	0.000000e+00	6.246545e+00
parodiaLusiadas.txt	VO	10589	2881	3.013030e-01	1.000000e-06	5.000000e-06	1.900000e-05	0.000000e+00	3.013240e-01
sherlock.txt	VO	108833	8066	1.073662e+01	1.000000e-06	9.000000e-06	4.900000e-05	1.000000e-06	1.073667e+01

Tabela 3: Tempos de execução dos métodos da Lista Desorganizada para cada livro (s)

Nome do arquivo	TSO	Total de palavras no arquivo	Palavras armazenadas na TSO	Tempo insere()	Tempo devolve()	Tempo remove()	Tempo rank()	Tempo seleciona()	Tempo Total
criticaeconomiapoliticaIngles.txt	LD	87693	6496	8.364580e-01	1.300000e-05	6.900000e-05	1.779050e-01	2.132590e-01	1.227648e+00
lusiadasIngles.txt	LD	171099	14460	4.080817e+00	4.000000e-05	1.100000e-05	9.231150e-01	1.117849e+00	6.121861e+00
manifestocomunistaIngles.txt	LD	14627	2624	9.520500e-02	1.000000e-06	4.000000e-06	2.615600e-02	2.965900e-02	1.510220e-01
mariliadedirceo.txt	LD	26389	4957	2.997230e-01	5.800000e-05	4.000000e-06	1.040850e-01	1.236840e-01	5.276080e-01
ocapitalGrego.txt	LD	3717	986	1.311900e-02	7.000000e-06	3.000000e-06	3.793000e-03	3.993000e-03	2.091900e-02
oslusiadas.txt	LD	63475	9042	1.091917e+00	2.000000e-06	8.000000e-06	3.527650e-01	4.259130e-01	1.870599e+00
parodiaLusiadas.txt	LD	10589	2881	1.043330e-01	0.000000e+00	1.000000e-05	3.334500e-02	3.833000e-02	1.760080e-01
sherlock.txt	LD	108833	8066	1.215729e+00	8.000000e-06	7.000000e-06	2.824910e-01	3.404890e-01	1.838725e+00

Tabela 4: Tempos de execução dos métodos da Lista Organizada para cada livro (s)

Nome do arquivo	TSO	Total de palavras no arquivo	Palavras armazenadas na TSO	Tempo insere()	Tempo devolve()	Tempo remove()	Tempo rank()	Tempo seleciona()	Tempo Total
criticaeconomiapoliticaIngles.txt	LO	87693	6496	5.260295e+00	1.000000e-05	9.300000e-05	1.010000e-04	6.100000e-05	5.260477e+00
lusiadasIngles.txt	LO	171099	14460	2.726081e+01	2.140000e-04	1.150000e-04	2.330000e-04	1.560000e-04	2.726162e+01
manifestocomunistaIngles.txt	LO	14627	2624	3.428710e-01	6.000000e-06	2.900000e-05	1.600000e-05	1.800000e-05	3.429170e-01
mariliadedirceo.txt	LO	26389	4957	1.158591e+00	6.000000e-06	5.600000e-05	3.900000e-05	4.400000e-05	1.158686e+00
ocapitalGrego.txt	LO	3717	986	3.380200e-02	1.400000e-05	1.500000e-05	1.400000e-05	7.000000e-06	3.385100e-02
oslusiadas.txt	LO	63475	9042	5.821786e+00	1.000000e-04	1.090000e-04	8.800000e-05	9.000000e-05	5.822164e+00
parodiaLusiadas.txt	LO	10589	2881	2.789200e-01	2.500000e-05	2.900000e-05	2.800000e-05	2.100000e-05	2.790190e-01
sherlock.txt	LO	108833	8066	1.015170e+01	5.900000e-05	1.320000e-04	7.100000e-05	7.900000e-05	1.015197e+01

Tabela 5: Tempos de execução dos métodos da Árvore Binária de Busca para cada livro (s)

Nome do arquivo	TSO	Total de palavras no arquivo	Palavras armazenadas na TSO	Tempo insere()	Tempo devolve()	Tempo remove()	Tempo rank()	Tempo seleciona()	Tempo Total
criticaeconomiapoliticaIngles.txt	AB	87693	6496	3.981400e-02	1.000000e-06	1.200000e-05	1.000000e-06	1.000000e-06	3.981800e-02
lusiadasIngles.txt	AB	171099	14460	8.925800e-02	1.000000e-06	9.000000e-06	0.000000e+00	0.000000e+00	8.926000e-02
manifestocomunistaIngles.txt	AB	14627	2624	6.420000e-03	1.000000e-06	4.000000e-06	0.000000e+00	0.000000e+00	6.422000e-03
mariliadedirceo.txt	AB	26389	4957	1.360000e-02	1.000000e-06	4.000000e-06	1.000000e-06	0.000000e+00	1.360300e-02
ocapitalGrego.txt	AB	3717	986	1.530000e-03	1.000000e-06	3.000000e-06	1.000000e-06	0.000000e+00	1.533000e-03
oslusiadas.txt	AB	63475	9042	3.370200e-02	1.000000e-06	6.000000e-06	1.000000e-06	1.000000e-06	3.370600e-02
parodiaLusiadas.txt	AB	10589	2881	8.079000e-03	1.000000e-06	4.000000e-06	1.000000e-06	1.000000e-06	8.083000e-03
sherlock.txt	AB	108833	8066	4.651400e-02	0.000000e+00	6.000000e-06	1.000000e-06	1.000000e-06	4.651600e-02

Tabela 6: Tempos de execução dos métodos da Árvore Heap (Treap) para cada livro (s)

Nome do arquivo	TSO	Total de palavras no arquivo	Palavras armazenadas na TSO	Tempo insere()	Tempo devolve()	Tempo remove()	Tempo rank()	Tempo seleciona()	Tempo Total
criticaeconomiapoliticaIngles.txt	TR	87693	6496	5.224700e-02	1.000000e-06	6.000000e-06	1.000000e-06	1.000000e-06	5.225100e-02
lusiadasIngles.txt	TR	171099	14460	1.272790e-01	2.000000e-06	1.000000e-05	1.000000e-06	0.000000e+00	1.272840e-01
manifestocomunistaIngles.txt	TR	14627	2624	1.007300e-02	1.000000e-06	4.000000e-06	1.000000e-06	1.000000e-06	1.007700e-02
mariliadedirceo.txt	TR	26389	4957	1.734100e-02	1.000000e-06	5.000000e-06	1.000000e-06	1.000000e-06	1.734500e-02
ocapitalGrego.txt	TR	3717	986	2.124000e-03	1.000000e-06	4.000000e-06	1.000000e-06	1.000000e-06	2.128000e-03
oslusiadas.txt	TR	63475	9042	4.601300e-02	2.000000e-06	6.000000e-06	1.000000e-06	0.000000e+00	4.601800e-02
parodiaLusiadas.txt	TR	10589	2881	5.997000e-03	0.000000e+00	3.000000e-06	1.000000e-06	0.000000e+00	5.998000e-03
sherlock.txt	TR	108833	8066	7.525000e-02	1.000000e-06	5.000000e-06	1.000000e-06	1.000000e-06	7.525400e-02

A análise dos resultados expostos nas tabelas 1 à 6 mostra que as estruturas AB (árvore binária) e TR (árvore heap) foram as mais rápidas na execução dos métodos. Isto era esperado, uma vez que são métodos em que o caso médio (inserção de palavras em ordem aleatória) tem complexidade O(log(N)).

A Treap garante complexidade O(log(N)) mesmo no pior caso, mas ao custo de operações de manutenção da árvore para mantê-la sempre balanceada. Isto explica porque no presente caso, os resultados foram piores que a árvore binária. Porém, se as palavras já estivessem relativamente ordenadas, a árvore binária teria se transformado em algo parecido com uma lista, perdendo performance tanto na inserção quanto na busca.

Nas estruturas lineares, como os vetores e listas observou-se que a as versões Desordenadas foram mais rápidas na inserção, mas muito mais lentas nos métodos em que alguma busca e ordenação é importante (devolve(), remove(), rank() e seleciona()). Ainda assim, no geral, as contrapartes foram mais rápidas observando-se o tempo total, porque houve muito mais utiliação do método insere() do que os outros, que só foram utilizados uma vez para cada livro, do contrário, os resultados poderiam ser diferentes.

6 Conclusão

A melhor estrutura para utilização de tabelas de símbolos, entre as implementadas, é a árvore binária de busca (AB), pois livros, em geral, apresentam uma "aleatoriedade" suficiente para que evitar o pior caso. Entretanto, se o objetivo é garantir que a árvore sempre estará balanceada, a árvore Heap (TR) é uma boa alternativa.