Relatório - EP2 de MAC0323

Nome: Gabriel Haruo Hanai Takeuchi NUSP: 13671636

Como compilar e executar

Para compilar, na pasta raíz do projeto, execute:

```
make main.out
```

Para executar, na pasta raíz do projeto, execute:

```
./main.out < <nomedoarquivo>
```

Funções e estruturas auxiliares

Os arquivos `lib.cpp` e `lib.h` contêm estruturas de dados e funções auxiliares para o programa. Além disso, há uma função `void separaPalavras()` adaptada para VO, ABB, etc, em seus respectivos arquivos `vo.cpp`, `abb.cpp`, etc.

As estruturas auxiliares são:

```
class Item {
public:
  int numOcorrencias;
  int numLetras;
  int numVogais; // número de vogais não-repetidas

  Item(std::string key) {
    this->numOcorrencias = 1;
    this->numLetras = contaNumLetras(key);
    this->numVogais = contaNumVogaisUnicas(key);
  }
};
```

As funções auxiliares são:

- `void separaPalavras(unsigned int n, <T>& t);` Lê a entrada e adiciona cada palavra filtrada na estrutura imposta; diferencia letras maiúsculas de minúsculas.
- `int contaNumLetras(const std::string& s);` Lê uma string e retorna o número de letras que ela contém.
- `int contaNumVogaisUnicas(const std::string& s);` Lê uma string e retorna o número de vogais não-repetidas que ela contém.

- `bool repeteLetras(std::string key);` Lê uma string e retorna um booleano indicando se ela contém letras repetidas.
- `bool repeteVogais(std::string key);` Lê uma string e retorna um booleano indicando se ela contém vogais repetidas.

Vetor ordenado

A implementação do vetor ordenado está localizada nos arquivos `vo.cpp` e `vo.h`. A estrutura utilizada para armazenar os dados, localizada no arquivo `lib.h`, é a struct `Palavra` descrita abaixo:

```
struct Palavra {
  // Key
    std::string key;
  // Item
    int numOcorrencias;
    int numLetras;
    int numVogais; // número de vogais não-repetidas

    Palavra(std::string key) {
        this->key = key;
        this->numOcorrencias = 1;
        this->numLetras = contaNumLetras(key);
        this->numVogais = contaNumVogaisUnicas(key);
    }
};
```

As propriedades `Key` e `Item` já estão combinadas em uma única estrutura `Palavra`. O VO é um `std::vector<Palavras*>`, que é ordenado em ordem alfabética. A cada inserção, usa-se de uma busca binária para encontrar a posição correta da palavra. Caso a palavra já exista, o número de ocorrências é incrementado. Caso contrário, a palavra é inserida na posição correta.

Árvore binária de busca

A implementação da árvore binária de busca está localizada nos arquivos `abb.cpp` e `abb.h`. A estrutura utilizada para armazenar os dados é um `NoABB` abaixo:

```
class NoABB {
public:
    std::string key;
    Item* value;
    NoABB* esq;
    NoABB* dir;

    NoABB(std::string key) {
        this->key = key;
        this->value = new Item(key);
        this->esq = nullptr;
        this->dir = nullptr;
```

```
    }
};
```

As propriedades `Key` e `Item` não estão combinadas em uma única estrutura, havendo um ponteiro para `Item*`.

Treap

A implementação da treap está localizada nos arquivos `tr.cpp` e `tr.h`. A estrutura utilizada para armazenar os dados é um `NoTR` abaixo:

```
class NoTR {
public:
    NoTR* esq; NoTR* dir; NoTR* pai;
    int prioridade;
    std::string key;
    Item* value;

    NoTR(std::string key) {
        this->esq = this->dir = this->pai = nullptr;
        this->prioridade = rand()%MAXKEYS;
        this->key = key;
        this->value = new Item(key);
    }
};
```

As propriedades `Key` e `Item` não estão combinadas em uma única estrutura, havendo um ponteiro para `Item*`.

IMPORTANTE: A constante `MAXKEYS`, localizada no arquivo `lib.h`, é definida como 1000000 e serve para limitar o valor da prioridade de cada nó da treap. Edite essa constante de acordo com o máximo de palavras que serão lidas.

Árvore 2-3

A implementação da árvore 2-3 está localizada nos arquivos `ar23.cpp` e `ar23.h`. A estrutura utilizada para armazenar os dados é um `No23` abaixo:

```
class NoA23 {
public:
    std::string key1, key2;
    Item* value1; Item* value2;
    NoA23* esq; NoA23* mid; NoA23* dir;
    int nKeys;
    NoA23() {
        key1 = ""; key2 = "";
        value1 = value2 = nullptr;
        esq = mid = dir = nullptr;
        nKeys = 0;
    }
};
```

As propriedades `Key` e `Item` não estão combinadas em uma única estrutura, havendo um ponteiro para `Item*`.

Árvore rubro-negra

A implementação da árvore rubro-negra está localizada nos arquivos `ar23.cpp` e `ar23.h`. A estrutura utilizada para armazenar os dados é um `NoRN` abaixo:

```
class NoARN{
public:
    std::string key;
    int numOcorrencias, numLetras, numVogaisUnicas;
    NoARN* pai; NoARN* esq; NoARN* dir;
    bool cor; // true = vermelho, false = preto

    NoARN(std::string key, bool cor, NoARN* pai, NoARN* esq, NoARN* dir) {
        this->key = key;
        this->numOcorrencias = 1;
        this->numLetras = contaNumLetras(key);
        this->numVogaisUnicas = contaNumVogaisUnicas(key);
        this->pai = pai; this->esq = esq; this->dir = dir;
        this->cor = cor;
    }
};
```

As propriedades `Key` e `Item` já estão combinadas em uma única estrutura.

Testes

Os testes foram feitos com o arquivo `mobydick.txt`, que contém 110731 palavras, e as consultas foram as seguintes:

```
5
F
O your
L
SR
VD
```

Os testes foram feitos com o comando `time ./main.out < mobydick.txt`.

Estrutura de dado	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Teste 6	Teste 7	Teste 8	Teste 9	Teste 10	Media
VO	0.106	0.105	0.110	0.102	0.103	0.102	0.104	0.101	0.103	0.102	0.1038
ABB	0.103	0.103	0.111	0.108	0.107	0.106	0.108	0.104	0.105	0.110	0.1065

Estrutura de dado	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Teste 6	Teste 7	Teste 8	Teste 9	Teste 10	Media
TR	0.139	0.130	0.130	0.129	0.133	0.138	0.127	0.134	0.131	0.133	0.1324
ARN	0.097	0.104	0.097	0.102	0.105	0.098	0.107	0.102	0.101	0.097	0.101
A23	0.145	0.149	0.148	0.144	0.139	0.144	0.145	0.142	0.152	0.151	0.1458

Do mais rápido para o mais lento, temos:

1. ARN
2. VO
3. ABB
4. TR
5. A23

Conclusão

As expectativas das estruturas mais performáticas estavam na ARN, A23, TR, ABB e VO, respectivamente. Entretanto, não foi o caso. A pior performance sendo a A23 não foi esperada, o que mostra que a implementação da mesma não foi eficiente. A melhor performance sendo a ARN foi esperada e condiz com o esperado. A VO e ABB tiveram uma performance muito próxima, o que também era esperado. A TR teve uma performance pior que a VO e ABB, o que não era esperado, mas pode ser explicado pelo fato de que a implementação da mesma não foi eficiente.