UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação

Disciplina	Curso	Turmas	Período
Algoritmos e Estruturas de Dados II			2°

Professores
Erickson Nascimento, William Schwartz

Trabalho Prático 3 - Árvores de Pesquisa/Hashing

Data de entrega: 30/06/2013 **(não haverá prorrogação do prazo devido ao final do semestre)**
Valor: 15 **pontos**

1 Introdução

Uma editora possui diversos títulos em seu acervo, porém estão desatualizados. Um dos itens que precisa ser atualizado é o índice remissivo de cada livro. O índice remissivo consiste em uma lista alfabética de palavras-chave ou palavras relevantes do texto com a indicação dos locais no texto onde cada palavra-chave ocorre.

A empresa precisa atualizar o índice remissivo de diversos livros até 30/06/2013. Para isso, contrataram você para automatizar o sistema. A empresa precisará que você faça a leitura dos índices que ela possui e permita operações de inserção de novos registros e busca por elemento e intervalos. Ela também gostaria de calcular o número de ocorrências de cada registro do índice, dentre outras operações.

Este trabalho é dividido em três partes. Na primeira parte, você deverá representar a hierarquia dos registros por meio de árvores com balanceamento SBB aninhadas, além de implementar as operações de inserção e busca por um único elemento. Na segunda parte, além das etapas anteriores, deverá ser implementado busca por intervalo e um método para obter todas as chaves que aparecem em páginas ímpares. Finalmente, na terceira parte pede-se ao aluno para implementar uma busca por chaves em um intervalo de páginas e um contador do número de ocorrências de cada um dos registros utilizando técnicas de hashing.

2 O que deve ser feito

Dado um índice remissivo como entrada, deve-se propor uma solução em que as operações de busca e inserção possuam complexidade $O(\log_2 n)$, onde n é o número de elementos. Para isso, deverá ser utilizada uma árvore balanceada SBB para cada índice/chave. Cada palavra-chave possuirá nós filhos correspondendo aos seus subíndices. Destes subíndices que possuírem outros índices aninhados, deverão ser representados por outra árvore; e assim sucessivamente. Deverão ser implementadas as seguintes operações:

Inserção de um novo registro Inserir o registro em sua posição correspondente. Recebe como parâmetro a `chave` a ser inserida, o nome da `chave-pai` onde ela deverá ser inserida e as páginas a serem inseridas (pelo menos uma). A inserção de um novo registro deve ser feita em complexidade $O(log_2(n))$.

Busca por elemento Possibilitar ao usuário buscar por um registro. Recebe como parâmetro o nome da `chave` a ser buscada. Deve-se imprimir a primeira chave encontrada e o caminho percorrido para alcançá-la. Caso a chave não seja encontrada, imprimir `-1`. A busca por um elemento deve possuir complexidade $O(\log_2(n))$ (assumindo que o número de níveis do índice é constante).

Busca por intervalo Executar buscas por intervalo, como elementos que possuem chave maior ou igual a X e menor ou igual a Y, e imprimir os elementos que se encontram nesta restrição. Recebe como parâmetro a `chave_menor` e a `chave_maior` que delimitam o intervalo de busca. Todas as ocorrências deverão ser retornadas em ordem.

Número de ocorrências Outra operação que deverá ser implementada é um contador de ocorrências do número de palavras-chave. Para isso, deverá ser utilizado um algoritmo de hashing. Caso não seja implementado utilizando hashing, a pontuação no Pratico não será contabilizada. O vetor para hashing é restrito ao tamanho máximo de $2 \cdot N$, onde N é o número de chaves total. Esta operação não recebe nenhum parâmetro e a saída deve ser ordenada por palavra-chave.

Número de chaves em páginas ímpares Contabilizar o número de palavras que se encontram em páginas ímpares. Também não recebe nenhum parâmetro.

Entradas em um conjunto de páginas Encontrar todas as entradas dentro de um intervalo de páginas dado. Por exemplo, no intervalo de páginas 2 e 15 — retornar todas as chaves que contém entradas nesse intervalo de páginas (incluindo a página 2 e a página 15). Recebe como parâmetros o primeiro e o segundo número da página desejados. Essa operação não precisa retornar o caminho percorrido para chegar em cada chave. No entanto, é necessário ordenar as palavras-chave.

A execução do programa será feita da seguinte forma:

```
./exec entrada.txt saida.txt
```

2.1 Estruturas de Dados

Como sugestão para representar árvores aninhadas, acrescente ao TAD `sbb` um apontador para a raiz da árvore contida dentro de um nó da árvore. Assim, cada nó da árvore possui uma subárvore interna.

```
struct sbb {
    struct registro reg;
    struct sbb *esq;
    struct sbb *dir;
    int esqtipo;
    int dirtipo;
    struct sbb *subarvore; // especifica uma nova subárvore
}
```

O tipo `registro` deve conter um campo para a `chave`, do tipo `char*`, que representará cada palavra-chave do índice remissivo. O segundo campo consiste em um tipo de dados para armazenar o número das páginas onde cada palavra-chave ocorre. Exemplo:

```
struct registro {
    char* chave;
    TipoDados dados;
}
```

onde `TipoDados` contém uma lista para armazenar os números das páginas em que uma palavra-chave ocorre.

```
typedef struct {
    TipoLista paginas;
} TipoDados;
```

3 Entrada

A entrada consistirá em um arquivo contendo o índice remissivo. Todas as palavras serão compostas por um único termo, i.e, não existirá chaves com palavras compostas ou chaves com múltiplas palavras, como `algoritmos e estruturas de dados`. Após a palavra-chave, haverá, pelo menos, um número indicando a página em que cada palavra ocorre. Não há um limite máximo de páginas que podem ocorrer.

As palavras-chave ocorrerão apenas em minúsculas. Além disso, uma mesma palavra-chave pode repetir em subníveis diferentes, mas não em um mesmo nível. As palavras-chave do índice não estão necessariamente

ordenadas e vão sendo inseridas nas árvores na mesma ordem em que são lidas do arquivo. O caracter ␣ representa um espaço no arquivo de entrada.

Ao final do arquivo, saltando uma linha, virão as operações que deverão ser realizadas sobre este índice.

```
<chave01>,␣<n1>,␣<n2>,␣<n3>
␣<chave11>,␣<n1>
␣<chave12>,␣<n1>,␣<n2>
␣<chave13>,␣<n1>
<chave02>,␣<n1>,␣<n2>,␣<n3>,␣<n4>
␣<chave21>,␣<n1>
␣<chave22>,␣<n1>,␣<n2>
␣␣<chave221>,␣<n1>,␣<n2>
␣␣<chave222>,␣<n1>,␣<n2>
␣␣<chave223>,␣<n1>
␣␣<chave224>,␣<n1>
␣␣<chave225>,␣<n1>,␣<n2>,␣<n3>
␣<chave23>,␣<n1>
␣␣<chave231>,␣<n1>
␣␣<chave232>,␣<n1>
␣␣<chave233>,␣<n1>
<chave03>,␣<n1>

inserir <chave> <chave_pai> <pagina 1>                          // inserção de um novo registro
inserir <chave> <chave_pai> <pagina 1>, <pagina 2>, <pagina 3>  // ... pelo menos uma pagina

busca <chave>
busca_intervalo <chave_menor> <chave_maior>
contar
contar_impares
buscar_chaves_paginas <pagina 1> <pagina 2>
```

Parte do índice de entrada acima está representado abaixo.

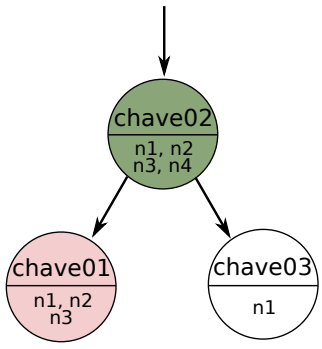
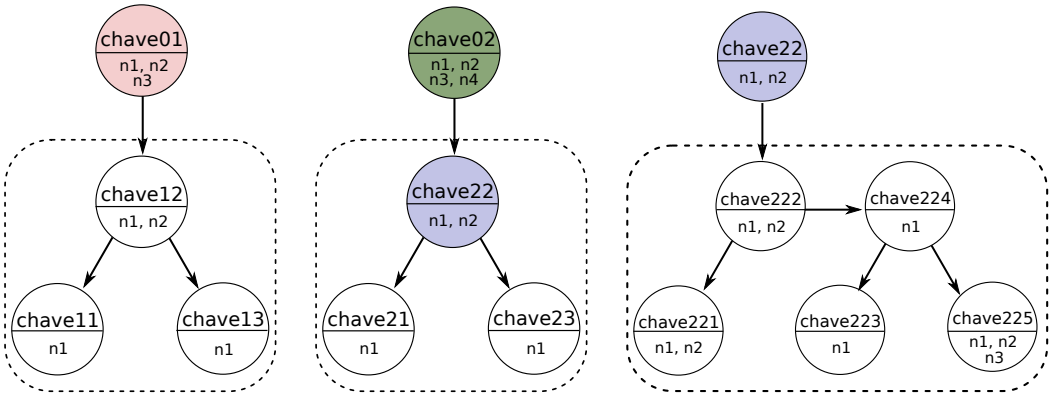


Figura 1: Primeira árvore. Desenho ilustrando o conceito de árvores aninhadas. Dentro de cada nó existe uma lista contendo as páginas nas quais a palavra-chave ocorre.

Note que existem árvores dentro de outras árvores. Cada árvore deverá manter seu balanceamento próprio—que é diferente de fazer um único balanceamento global para a árvore inteira.

4 Saída

A saída deverá ser impressa na mesma ordem de chamada de cada operação. Deve-se saltar uma linha entre duas operações distintas. Se uma determinada operação retornar mais que uma resposta, elas devem se



(a) Subárvore pertencente a `chave01`.

(b) Subárvore pertencente a `chave02`.

(c) Subsubárvore pertencente a `chave22` pertencente a `chave02`.

Figura 2: Subárvores da árvore

expandir por várias linhas, com um espaçamento no início de cada linha de resposta. Cada operação tem um formato de saída específico a ser seguido. A seguir, o formato específico de cada operação:

```
busca(chaveX): (chave0(chave1(chave2,chave3(chave5(chaveX))))) // cada parêntesis identifica
                                                               // ...  um subnível da árvore
                                                               // ... ',' representam nós em
                                                               // ... um mesmo nível da árvore.

busca(chaveX): -1                                              // Caso a chave não seja
                                                               // encontrada, imprima '-1'

busca_intervalo(chave_m0,chave_m9):
␣chave_m0                  // chaves dentro do intervalo especificado
␣chave_m1                  // deve-se imprimir as chaves que são
␣chave_m2                  //  >= chave_m0 && <= chave_m9
␣chave_m8
␣chave_m9

contar:
␣(chave1,␣<numero_ocorrencias>) // chaves repetidas em níveis diferentes devem ser
␣(chave2,␣<numero_ocorrencias>) // contadas múltiplas vezes. Ordenar por
␣(chave3,␣<numero_ocorrencias>) // palavra-chave.
␣(chave4,␣<numero_ocorrencias>)

contar_impares:␣<numero de palavras-chave em paginas impares>

buscar_chaves_paginas(<numero_pagina_1>,<numero_pagina_2>):
␣chaveXX                   // ordenar por palavra-chave
␣chaveXY
␣chaveXZ
␣chaveYZ
```

5 Exemplo

Exemplo de funcionamento do programa. Um exemplo mais completo será acrescentado posteriormente, pois é necessário a execução do algoritmo para obter a resposta correta. Observe que, neste exemplo simplificado, o índice remissivo está em ordem alfabética, mas isto não é necessariamente verdadeiro.

O seguinte arquivo de entrada

```
estruturas, 4, 81
 arrays, 283, 284
 fila, 283, 284
 pilha, 300
ordenacao, 250, 340, 483
 bubblesort, 266
  complexidade, 388
 mergesort, 206, 335, 359, 436
  bottomup, 348
  topdown, 341, 342, 343, 344, 345
 quicksort, 118, 305
  basico, 409
  complexidade, 455
  particionamento, 326
  selecao, 327, 328
pesquisa, 53, 54, 55
 hashing, 595, 597
  lineares, 697
  funcoes, 611
  duplas, 595, 600
 radix, 403
  digital, 431, 433
  tries, 596
  patricia, 623, 624
recursao, 187, 188, 189, 248
 buscabinaria, 498
 quicksort, 321
 selection, 330

busca complexidade
busca_intervalo basico selecao
contar
```

produz a seguinte saída:

```
busca(complexidade): (ordenacao(bubblesort(complexidade)))

busca_intervalo(basico,selecao):
 basico
 complexidade
 particionamento
 selecao

contar:
 (arrays, 1)
 (basico, 1)
 (bottomup, 1)
 (bubblesort, 1)
 (buscabinaria, 1)
 (complexidade, 2)
 (digital, 1)
 (duplas, 1)
 (estruturas, 1)
```

```
(fila, 1)
(funcoes, 1)
(hashing, 1)
(lineares, 1)
(mergesort, 1)
(ordenacao, 1)
(particionamento, 1)
(patricia, 1)
(pesquisa, 1)
(pilha, 1)
(quicksort, 2)
(radix, 1)
(recursao, 1)
(selecao, 1)
(selection, 1)
(topdown, 1)
(tries, 1)
```

O que deve ser entregue

- Código fonte do programa em C (todos os arquivos .c e .h), devidamente comentado e documentado.
- Documentação sobre o trabalho (máximo de 10 páginas em formato pdf), com as seguintes seções:

  1. Introdução: descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa. Cópias idênticas da especificação do trabalho nesta seção terão nota ZERO.
  2. Implementação: descrição sobre a implementação do programa. Deve ser explicada a estrutura de dados utilizada, o funcionamento de todas as funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como todas as decisões tomadas na implementação do trabalho.
  3. Estudo de Complexidade: estudo da complexidade DETALHADO do tempo de execução dos procedimentos implementados e do programa como um todo (notação O).
  4. Testes: devem ser realizados diversos testes para cada operação variando o número de registros na árvore e apresentados os gráficos de tempos de execução correspondentes. Faça também uma análise dos resultados dos testes comparando com o estudo de complexidade realizado na Seção anterior.
  5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.

- O trabalho será entregue através do sistema de submissão de trabalhos práticos: `http://aeds.dcc.ufmg.br`, em um arquivo compactado (.zip) contendo os arquivos fonte. A documentação deve ser enviada em pdf, no link específico para sua submissão no sistema. Caso tenha algum problema na submissão pelo Prático, envie o trabalho por email para todos os monitores.

Comentários gerais

- Todos os TADs, vetores e matrizes deverão ser sempre declarados como ponteiros e instanciados dinamicamente.
- Técnica, clareza, legibilidade, organização, coesão, indentação, comentários, nomes adequados a variáveis, atributos e funções também vão valer pontos.

- Trabalhos copiados serão penalizados com a nota zero.
- Os trabalhos não serão aceitos, em hipótese nenhuma, fora do prazo.