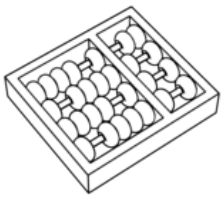


Instituto de Computação
Universidade Estadual de campinas

MC202 - Estruturas de Dados

Laboratório 6

Árvores Binárias de Busca

Data de publicação: Sexta feira, 13 de maio de 2016
Prazo máximo de submissão: Domingo, 5 de junho de 2016 às 23h59m

Professor: Neucimar J. Leite <neucimar@ic.unciamp.br>

Monitores:
- Juan Hernández (PED) <juan.albarracin@students.ic.unicamp.br>
- Leonardo Yvens (PAD) <leoyvens@gmail.com>

Grupo do curso: https://groups.google.com/d/forum/mc202bc_2016s1
Sítio eletrônico da submissão do código: https://susy.ic.unicamp.br:9999/mc202bc

Enunciado

Represente os registros de uma tabela de um banco de dados relacional como uma árvore binária de busca e implemente as operações de criação, atualização e remoção de registros.

Descrição do problema

Em bancos de dados é muito útil criar estruturas para tornar mais eficiente o acesso aos dados, considerando que eles têm uma locação física em disco. Uma dessas estruturas é a árvore binária de busca para construção de índices.

PESSOA			
Id	**Nome**	**Sobrenome**	**Idade**
35	Ernani	Gama	31
36	Yuri	Leite	49
37	Deise	Tomazelli	27

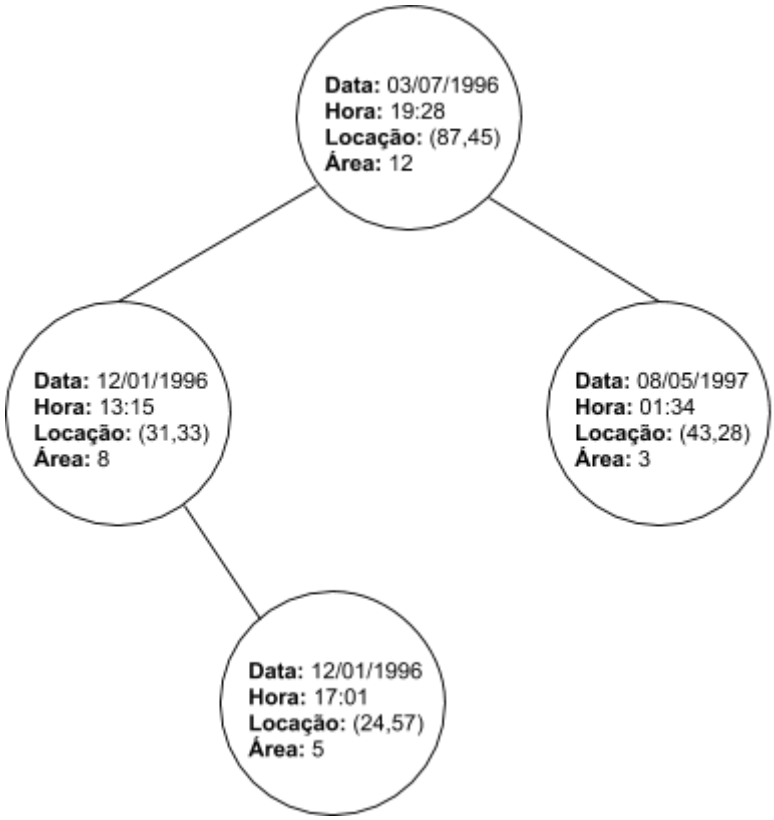
O exemplo acima é uma **tabela** de um banco de dados que armazena informação de pessoas, neste caso, nome, sobrenome e idade. Cada linha (pessoa) dessa tabela é um **registro**. A coluna “Id” é um identificador único de cada registro (chave primária). Suponha que a tabela tem dez milhões de registros, ao invés de três, e que constantemente é consultada pelo nome da pessoa, para saber sua idade. Como não sabemos onde se encontra o registro da pessoa que buscamos, teríamos de percorrer todos os registros até encontrá-la, a não ser que tenhamos um índice que ordene os registros em ordem alfabética e se, por exemplo, procuramos Yuri, poderíamos saltar, como num dicionário, todos os registros cujos nomes começam por letras anteriores a Y. Isto reduz de maneira significativa o tempo de operações em bases de dados. Neste exemplo, o índice é armazenado numa árvore binária de busca cuja chave (informação de comparação) é formada pela concatenação do nome e sobrenome da pessoa.
Para este laboratório, vamos trabalhar com incêndios florestais em uma área rural determinada:

INCÊNDIO							
Ano	**Mês**	**Dia**	**Hora**	**Minuto**	**Latitude**	**Longitude**	**Área**
1996	Julho	3	19	28	87	45	12
1997	Maio	8	1	34	43	28	3
1996	Janeiro	12	13	15	31	33	8
1996	Janeiro	12	17	1	24	57	5

Para facilitar, as informações de latitude e longitude serão apenas inteiros. Da mesma forma, a área pode ser em hectares.. A informação temporal (ano, mês, dia, hora, minuto) indica o momento da ocorrência do incêndio.
As operações realizadas na tabela seguem as seguintes regras:

- Um incêndio pode mudar de área e local com o tempo, mas nunca mudará o instante em que ocorreu.
- Assim que um incêndio é apagado, o registro correspondente é removido da tabela.
- Nem sempre novos incêndios serão registrados respeitando a ordem cronológica.

Suponha que a agência de preservação florestal tenha como identificar um incêndio pelo tempo da ocorrência. Portanto, você deve representar cada registro como um nó da sua árvore, através de um struct que armazene esta informação. A chave (informação de comparação) será o instante de ocorrência do incêndio: a árvore deve estar ordenada segundo esse critério.
Assumindo uma construção da árvore seguindo a ordem dos registros na tabela acima, a árvore resultante será:



Especificação de entrada e saída

- A primeira linha contém o número de registros *n* a serem inseridos inicialmente.
- As seguintes *n* linhas contêm 8 inteiros separados por espaço, representando a seguinte informação:

```
<ano> <mês> <dia> <hora> <minuto> <latitude> <longitude> <area>
```

- A linha seguinte contém o número *m* de operações realizadas sobre os dados.
- As *m* linhas seguintes contêm uma operação cada. Para identificá-las, o primeiro caractere da linha pode ser "**i**" para inserção, "**a**" para atualização, "**s**" para remoção e substituição por sucessor e "**p**" para remoção e substituição predecessor. A estrutura de cada operação é:
    - Inserção: cria um novo nó com todos os dados e o insere na árvore.

```
i <ano> <mês> <dia> <hora> <minuto> <latitude> <longitude> <area>
```

- Atualização: procura o nó com a informação temporal fornecida, e atualiza os campos latitude, longitude e área, segundo os valores fornecidos na linha.

```
a <ano> <mês> <dia> <hora> <minuto> <latitude> <longitude> <area>
```

- Remoção e substituição por predecessor: procura o nó com a informação temporal fornecida e o apaga, substituindo-o por seu predecessor em in-ordem (o mais à direita da subárvore esquerda).

```
p <ano> <mês> <dia> <hora> <minuto>
```

- Remoção e substituição por sucessor: procura o nó com a informação temporal fornecida e o apaga, substituindo-o por seu sucessor (o mais à esquerda da subárvore direita).

```
s <ano> <mês> <dia> <hora> <minuto>
```

Caso haja mais de um registro com a mesma informação temporal, as operações de atualização ou remoção serão feitas unicamente no primeiro registro encontrado. Caso uma atualização ou remoção seja feita em um nó que não exista, a operação é ignorada.

As regras para inserir na árvore são as mesmas apresentadas no laboratório anterior:

1. O primeiro registro da lista será a raiz.
2. Para cada um dos registros seguintes, parte-se da raiz da árvore até atingir uma folha, fazendo o seguinte:
    a. Se o valor a inserir for menor ou igual ao valor do nó atual, desloque-se para o esquerdo. Caso contrário, desloque-se para o filho direito.
3. Ao atingir uma folha, crie um novo nó com os valores a inserirem e, se o valor da chave for menor ou igual ao valor da chave da folha, aloque o nó como filho esquerdo da folha. Caso contrário, aloque-o como filho direito.

A entrada referente à árvore do exemplo anterior seria, entre outra opções possíveis:

```
4
1996 7 3 19 28 87 45 12
1997 5 8 1 34 43 28 3
1996 1 12 13 15 31 33 8
1996 1 12 17 1 24 57 5
```

Cabe a você entender por que a entrada é esta e por que pode haver outras ordens de valores na construção da mesma árvore. Seguem alguns exemplos de entradas com suas respectivas saídas.

A saída esperada consiste em *k* linhas, onde *k* é o número de nós na árvore após a operações forem feitas. Cada linha deve conter latitude, longitude e área dos nós separadas por espaço, e a ordem de impressão deve ser *pré-ordem*, já que é preciso conferir a estrutura da árvore final.

Entrada	**Saída**
4 1996 7 3 19 28 87 45 12 1997 5 8 1 34 43 28 3 1996 1 12 13 15 31 33 8 1996 1 12 17 1 24 57 5 3 p 1996 7 3 19 28 i 1996 2 6 14 27 42 36 9 a 1996 1 12 17 1 24 57 8	1996 1 12 17 1 24 57 8 1996 1 12 13 15 31 33 8 1997 5 8 1 34 43 28 3 1996 2 6 14 27 42 36 9
4 1996 7 3 19 28 87 45 12 1997 5 8 1 34 43 28 3 1996 1 12 13 15 31 33 8 1996 1 12 17 1 24 57 5 3 s 1996 7 3 19 28 i 1996 2 6 14 27 42 36 9 a 1996 1 12 17 1 24 57 8	1997 5 8 1 34 43 28 3 1996 1 12 13 15 31 33 8 1996 1 12 17 1 24 57 8 1996 2 6 14 27 42 36 9

Estrutura da submissão

O código fonte deve ser submetido no sistema SuSy para ser executado e testado. O sistema receberá três arquivos:

Nome	**Função**
main.c	Programa principal que lê os dados de entrada, faz as chamadas às funções e escreve a saída.
bstree.h	Contém unicamente a declaração do TAD e de funções para operações em árvores binárias de busca. Não será fornecida uma estrutura de arquivo, portanto você tem a liberdade de fazer a sua implementação para demonstrar seu domínio sobre o tema. Espera-se, no mínimo, uma função de criação da árvore, uma função de liberação da árvore e funções para adicionar, remover e atualizar os nós da árvore.
bstree.c	Implementação das funções declaradas em **bstree.h**.

row.h	Contém unicamente a declaração do TAD do registro.
row.c	Implementação das funções declaradas em **row.h**.

Observações

- A operação de remoção de um nó numa árvore binária de busca não é trivial. Sugere-se que você consulte a literatura e as notas de aula sobre como fazê-la. Encontrará que, quando o nó a ser apagado tem dois filhos, existem duas maneira de substituí-lo: pelo filho esquerdo ou pelo filho direito. Ambas a operações resultam em uma árvore diferente cada. Como indicado acima, você deve implementar as duas formas e aplicá-las corretamente em cada caso.

- É mandatório liberar memória dinamicamente alocada.

- Arquivos de teste serão fornecidos para a validação do programa. A avaliação no sistema será feita com esses e outros testes privados.

- O limite de submissões no SuSy é **15**. Recomenda-se executar de forma local seu programa, usando tanto os testes abertos quanto os exemplos aqui fornecidos.
  Submeta seu programa somente após haver testado localmente e de maneira bem sucedida todos os casos.

- A clareza do código e a sua documentação (por meio de comentários) serão avaliadas. O esforço dos monitores para entender o seu código deve ser mínimo. Com a documentação, você demonstra que compreendeu de maneira efetiva os conteúdos do curso referentes ao tema árvores binárias.

- Dúvidas podem ser esclarecidas nas aulas de laboratório ou no grupo do curso indicado no cabeçalho deste documento.

Critério de avaliação

$$nota = 5\frac{n_c}{c} + QD + AC$$

$$0 \leq QD \leq 1$$
$$0 \leq AC \leq 4$$

onde,

n	:	Número total de testes no SuSy
n_c	:	Número de testes corretos
QD	:	Qualidade do código
AC	:	Indicador de quanto o estudante tem demostrado dominar o tema de árvores binárias

A qualidade do código (QD) dependerá tanto da legibilidade quanto da documentação. Deve ser fácil para os monitores entender o que foi feito. Já o AC é uma nota que depende de quanto o estudante conseguiu convencer o revisor de que sabe implementar corretamente os TADs e operar com a estrutura em questão.

Considerações finais

- Embora haja várias maneiras de resolver os problemas indicados nos laboratórios, o estudante deve optar pela maneira que melhor exercite os conceitos trabalhados em sala de aula. **Não atender a esta advertência invalidará a parte da nota referente aos testes do SuSy**.

- Casos de plágio acarretam **média final zero na disciplina** para todos os envolvidos, sem exceção. O SuSy pode detectar casos de plágio no código, portanto evite compartilhar seu código com outros colegas, mesmo que seja apenas pequenos trechos.