MC202GH - ESTRUTURAS DE DADOS

2º Semestre de 2017

Professor: Rafael C. S. Schouery

Monitores: Yulle Glebbyo Felipe Borges (PED)

Caio Vinícius Piologo Véras Fernandes (PAD)

Lab. 08 - Busca Bidimensional

Peso 4

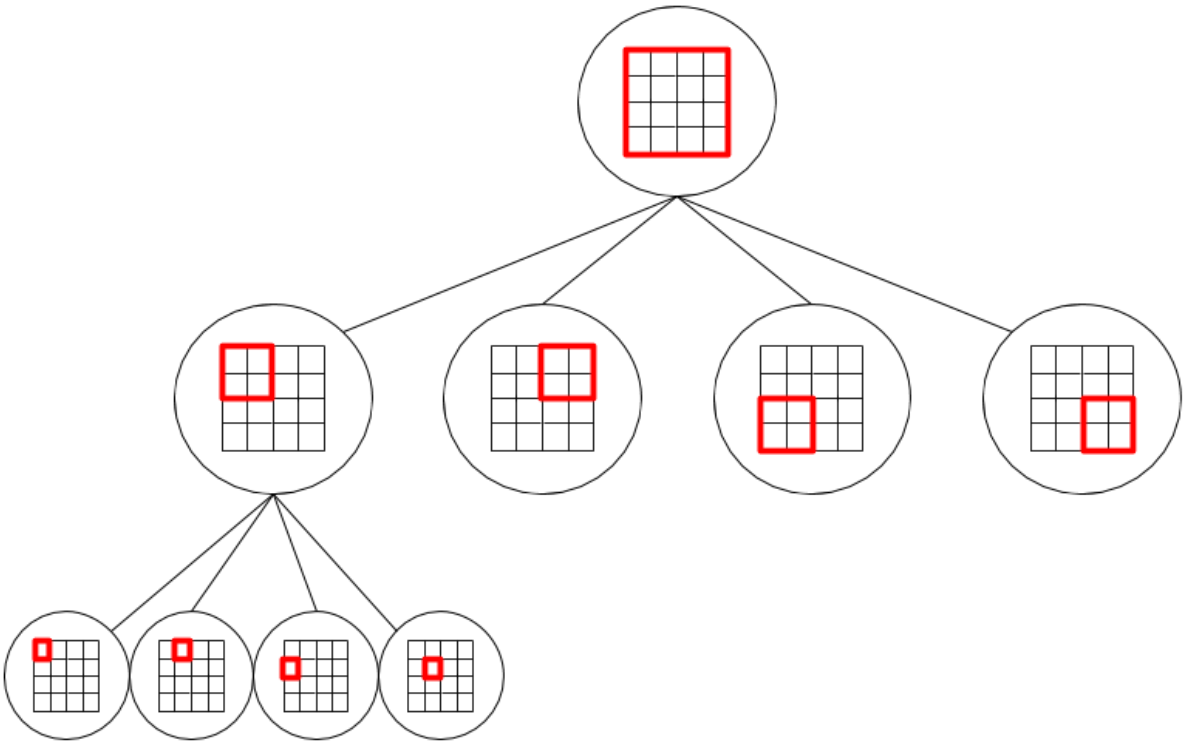
1. O Problema

Até então, temos estudado mecanismos que facilitam acesso de informações baseados em uma chave (como filas de prioridades e árvores binárias de busca). Entretanto, é comum a existência de entradas multidimensionais, isto é, cada entrada possui múltiplos campos que podem ser utilizados como chaves de uma busca. Assim, podemos representar as entradas como pontos em um espaço multidimensional, o valor de sua coordenada em cada dimensão corresponde ao valor de um de seus campos. Estes tipos de dados aparecem em diversas aplicações de banco de dados, computação gráfica, geometria computacional, processamento de imagens, etc.

Para ilustrar, imagine uma base de dados que armazena cidades e suas respectivas coordenadas X e Y em um mapa plano. Nesta base, deve ser possível pesquisar qual cidade está em uma determinada coordenada (x,y). A maneira mais simples de se armazenar os pontos desta base seria utilizando uma lista sequencial sem ordenação nenhuma, porém para buscar uma determinada cidade deve-se procurar por todas as cidades cadastradas checando cada um de seus valores de coordenada.

É possível melhorar a implementação desta base utilizando uma estrutura de dados mais sofisticada para armazenar os pontos. De maneira semelhante às árvores binárias de busca, existem as árvores quaternárias de busca bidimensional que lidam com entradas de duas dimensões (no nosso caso, coordenadas x e y, pois assumimos que não serão feitas buscas por nome da cidade).

Com árvores binárias de busca, dividimos o espaço da busca em duas regiões menores (menor que o valor da chave, e maior que o valor da chave). Com árvores quaternárias, dividimos o espaço bidimensional de busca em quatro quadrantes de tamanhos iguais: noroeste do ponto central (NO), nordeste do ponto central (NE), sudoeste do ponto central (SO), e sudeste do ponto central (SE). Esta divisão é feita recursivamente até que tenha-se apenas um ponto em cada quadrante. Assim, nós internos representam uma região e possui necessariamente 4 filhos, cada um representando um dos quadrantes da região do nó pai. A figura abaixo mostra como é feita a subdivisão de uma área:



Desta forma, os nós de uma árvore quaternária podem ser:

- **Nós internos**: têm necessariamente 4 filhos;
- **Nós folha vazios**: representa uma região que não contém nenhum ponto, logo não precisa ser subdividido;
- **Nós folha com conteúdo**: representa uma região que contém exatamente um ponto, e guarda as coordenadas deste ponto.

Nesta representação de árvores quaternárias, os pontos serão armazenados somente nos nós folhas, e os nós internos representam regiões que contém mais que um ponto, portanto deve ser subdividido para que o ponto seja encontrado.

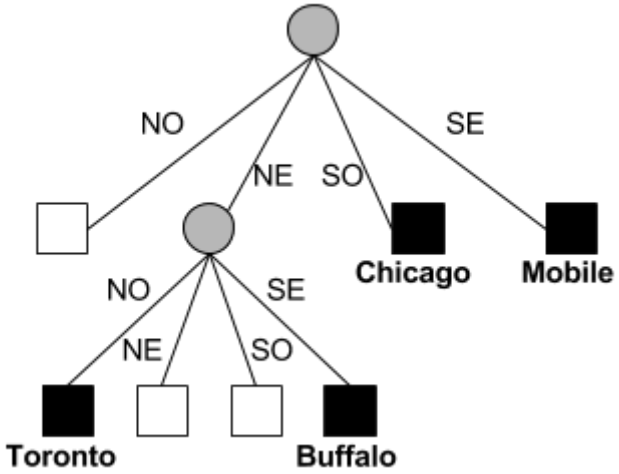
1.1 Inserção

Uma cidade C de coordenadas (a,b) é inserido na árvore quaternárias através da busca pelo quadrante que contém suas coordenadas. Para isso, a árvore é percorrida obedecendo às seguintes condições de inserção:

- Caso a raiz seja vazia, o nó com o ponto C passa a ser a raiz;
- Caso a raiz seja um nó interno, determine qual o quadrante C pertence e continue a inserção recursivamente a partir deste quadrante;
- Caso a raiz seja uma folha com conteúdo, a região desse nó deve ser dividida e o ponto que o ocupava deve ser movido para um dos 4 nós filhos criados a partir da divisão. Determine qual quadrante C pertence e continue a inserção recursivamente a partir deste quadrante.

Para ilustrar, podemos ver a subdivisão do espaço conforme inserimos pontos de acordo com a ordem Chicago (35,42), Mobile (52,10), Toronto (62,77) e Buffalo (82,65).

E a árvore resultante destas inserções:

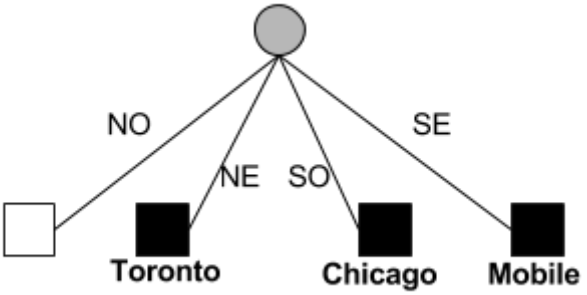


1.1 Remoção

O processo de remoção é para esta implementação de árvores quaternárias é simples, uma vez que os pontos estão sempre nas folhas, logo não será necessário fazer

nenhum tipo de reorganização da árvore. Para remover uma cidade C de coordenadas (a,b), deve-se navegar a árvore até encontrar um nó folha de coordenadas (a,b) (assume-se que não haverão coordenadas duplicadas) e liberá-lo de forma que ele passe a ser um nó folha vazio.

Entretanto, é importante observar que ao remover um ponto, é possível que o resultado seja um nó interno com 4 filhos folhas, e 3 deles são vazios. Neste caso, devemos passar o ponto contido no filho não vazio para o pai e remover os 3 filhos vazios, já que esta subdivisão passa a ser desnecessária. Por exemplo, se utilizarmos a árvore da Seção 1.1, e removermos o ponto de coordenada (82,65) temos a seguinte árvore como resultado:



1.3 Busca

A busca por um ponto C de coordenadas (a,b) em uma árvore quaternária deve simplesmente seguir recursivamente no quadrante contém o ponto (a,b), até que seja encontrado um nó folha com estas coordenadas. Caso tal nó não seja encontrado, isso quer dizer que o ponto C de coordenadas (a,b) não estão na base de dados. Utilizamos este mecanismo de busca por um ponto nos processos de inserção e remoção de um ponto.

As árvore quaternárias se destacam por permitirem busca regional de maneira relativamente simples. Uma região pode ser definida por um ponto central P e um raio r, logo a busca por região deve retornar todos os pontos que estão a uma distância até r do ponto P. Para executar esta busca a partir de uma raiz, deve-se obter quais de seus quadrantes filhos intercepta a área formada por P e r, e continuar a busca recursivamente em todos eles. Ao encontrar um nó folha com conteúdo, deve-se checar se suas coordenadas estão contidas na área formada por P e r, e imprimi-lo caso positivo.

1.4 Objetivo

O objetivo deste trabalho é implementar uma base de dados que armazena cidades com seus respectivos nomes e coordenadas (x,y) em um mapa plano. Esta base de dados **deve obrigatoriamente ser implementada utilizando árvores quaternárias**. Para implementar a árvore quaternária, **recomendamos** que você utilize a seguinte estrutura de nó:

```
typedef struct No {
  char dado;
  int x,y;
  short tipo; /* 0: nó interno; 1: nó folha conteúdo; 2: nó folha vazio.
              */
```

```
  struct No * filhos[4]; /* vetor com os 4 filhos NO, NE, SO, SE
                          * respectivamente */
} No;
typedef No * p_no;
```

A base de dado deve suportar operações de inserção, remoção, busca por ponto e busca por região:

- Para inserção, considere uma função para inserir um Nó P, em uma árvore de raíz R que correspondendo a um retângulo de dimensões L x A, com centro em (x,y).
- Para remoção, considere uma função para remover um ponto de coordenadas (px,py) da árvore de raiz R correspondente a um retângulo de dimensões L x A com centro em (x,y).
- Para busca por ponto, considere uma função para buscar por um ponto de coordenadas (px,py) na árvore de raiz R correspondente a um retângulo de dimensões L x A com centro em (x,y), e imprimir seu conteúdo caso ele exista. **É importante ressaltar que esta busca deve ser "inteligente", ou seja não deve procurar em quadrantes que o ponto não pertence.**
- Para busca por região, considere uma função para buscar por todos os pontos contidos na região definida por um ponto (px,py) e um raio r, na árvore de raiz R correspondente a um retângulo de dimensões L x A com centro em (x,y), e imprimir o conteúdo de cada um deles. **~~É importante ressaltar que esta busca deve ser "inteligente", ou seja não deve procurar em quadrantes que não interceptam a região procurada.~~ É importante ressaltar que esta busca pode procurar no quadrante mesmo se o quadrante não interceptar a região procurada.**

Para implementar a busca por região, **você deve implementar e utilizar a seguinte função** (com o uso do cabeçalho `math.h`) para determinar a região a ser buscada:

```
int distancia(int x1, int y1, int x2, int y2) {
  return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}
```

OBS:

- Por convenção, temos que a origem, ou seja o ponto (0,0) está no canto inferior esquerdo do retângulo formado pelo espaço de busca, logo não teremos pontos de coordenadas negativas.
- Assumimos também que todos os intervalos de subdivisão são fechados para o início e abertos para o fim, ou seja em um retângulo 10 x 10 o ponto (5,5) deve ser considerado como parte do quadrante Nordeste (NE) e o ponto (10,10) não pertence a região.
- **Todos as operações matemáticas que resultem em valores reais devem ser imediatamente arredondadas para baixo para manter coerência**.
- Na busca por região, deve-se seguir a ordem NO, NE, SO, SE (buscando apenas os quadrantes que interceptam a região) para que os pontos sejam impressos na ordem correta.

1.5 Informações Adicionais

Esta seção contém um detalhamento da forma utilizada para navegar a árvore recursivamente, e como calcular os centros dos quadrantes de maneira mais clara.

Considere que uma função de inserção na árvore, recebe um ponteiro pra raiz da árvore, um ponteiro para o nó a ser inserido (com nome da cidade, valor de x e valor y), largura máxima do quadrante representado, altura máxima do quadrante representado, e as coordenadas x e y do ponto que está no centro deste quadrante. Com isso teríamos uma função de inserção com uma assinatura parecida com essa:

```
ponteiro_no insere(ponteiro_no raiz, ponteiro_no novo, inteiro L, inteiro A, inteiro centro_x, inteiro centro_y);
```

Com isso podemos utilizar os valores de `centro_x` e `centro_y` diretamente para descobrir onde a recursão deve continuar, e utilizaremos os valores de L e A para calcular os valores de `centro_x` e `centro_y` do próximo nível da recursão.

Assim, para fazer uma chamada recursiva da função inserir, usamos uma comparação simples:

```
se (novo.x < centro_x)
  se (novo.y >= centro_y)
      responda NO
  senão se (novo.y < centro_y)
      responda SO
senão se (novo.x >= centro_x)
  se (novo.y >= centro_y)
      responda NE
  senão se (novo.y < centro_y)
      responda SE
```

Com isso saberemos em qual quadrante devemos chamar o algoritmo recursivamente, entretanto ainda precisamos calcular as dimensões deste quadrante filho e as coordenadas do seu ponto central. Neste ponto, como estamos considerando apenas valores inteiros, podem haver divergências. A forma correta (que está implementada no SuSy) assume que qualquer divisão deve ser arredondada para baixo imediatamente para não propagar o erro de arredondamento.

Para calcular as dimensões do quadrante filho, basta dividir as dimensões do quadrante pai por 2, arredondando para baixo:

```
inteiro filho_L = ((inteiro) L/2)
inteiro filho_A = ((inteiro) A/2)
```

Para calcular o ponto central do quadrante filho, teremos quatro casos possíveis (NO, NE, SO, SE) dependendo de qual quadrante filho contém o ponto a ser inserido:

```
caso NO:
```

```
  inteiro filho_centro_x = centro_x - ((inteiro) (L/4))
  inteiro filho_centro_y = centro_y + ((inteiro) (A/4))
caso NE:
  inteiro filho_centro_x = centro_x + ((inteiro) (L/4))
  inteiro filho_centro_y = centro_y + ((inteiro) (A/4))
caso SO
  inteiro filho_centro_x = centro_x - ((inteiro) (L/4))
  inteiro filho_centro_y = centro_y - ((inteiro) (A/4))
caso SE:
  inteiro filho_centro_x = centro_x + ((inteiro) (L/4))
  inteiro filho_centro_y = centro_y - ((inteiro) (A/4))
```

Para ilustrar, considere a subdivisão de um quadrante de 50 x 50 com centro em (25,25). O quadrante filho NO terá o seu ponto central em:

```
filho_centro_x = 25 - ((inteiro) (50/4))
               = 25 - ((inteiro) 12.5)
               = 25 - 12
               = 13
filho_centro_y = 25 + ((inteiro) (50/4))
               = 25 + ((inteiro) 12.5)
               = 25 + 12
               = 37
```

Calculando de maneira semelhante, temos o centro dos quadrantes `NE = (37,37), SO = (13,13)` e `SE = (37,13).`

Esta mesma forma para encontrar os pontos centrais dos quadrantes filhos deve ser utilizada na busca e remoção também. Cada pessoa implementa uma lógica diferente para percorrer a árvore, o que já é esperado, porém pedimos que utilizem esta forma para calcular os pontos centrais dos quadrantes filhos para que não de problemas de imprecisão e arredondamento com a saída do SuSy.

2. Entrada

Ao iniciar o programa, a primeira coisa a ser feita deve ser a leitura das dimensões do espaço de busca. Estes serão dados na forma de um inteiro L indicando o limite na coordenada x, e A indicando o limite na coordenada y. Assim, pontos com valores de x entre 0 e L-1, e valores de y entre 0 e A-1. Este processo não resulta em uma saída, ou seja, nada deve ser impresso na tela.

Uma vez que a árvore for criada, o programa deve entrar em um laço de comandos para adicionar, remover e buscar pontos na árvore. Estes comandos e suas entradas são descritos na tabela abaixo.

ID	Nome	Descrição
1	Adicionar Ponto	Dados inteiros **x** e **y**, e uma string **D** de no máximo 20 caracteres, insere o nó de coordenadas **x**, **y** com o

		conteúdo **D** na árvore de busca.
2	Remover Ponto	Dados inteiros **x** e **y**, remove o ponto de coordenadas **x**, **y** da árvore de busca, consequentemente removendo nós internos tornados desnecessários pela remoção.
3	Buscar Ponto	Dados inteiros x e y, procura o ponto de coordenadas **x**, **y** e imprime seu conteúdo.
4	Buscar Por Região	Dados inteiros **x** e **y**, representando as coordenadas de um ponto **P**, e um inteiro **r**, imprime o conteúdo de todos os pontos que estão a uma distância até **r** do ponto **P**, seguindo a ordem de busca NO, NE, SO e SE.
5	Imprimir Árvore	Imprime o percurso em pré-ordem da árvore atual. Os nós filhos devem ser impressos na ordem: NO, NE, SO, SE. Para nós internos, imprima apenas a letra I, para nós vazios imprima a letra V, e para nós folha com um ponto imprima suas coordenadas e seu conteúdo. Veja a seção de saída para mais informações sobre a formatação da resposta.
0	Finalização	Saia do sistema, certificando-se que toda a memória alocada foi liberada.

Exemplo de entrada:

```
100 100
1 35 42 Chicago
1 52 10 Mobile
1 62 77 Toronto
1 82 65 Buffalo
1 5 45 Denver
1 27 35 Omaha
1 85 15 Atlanta
1 90 5 Miami
3 62 77
5
4 75 75 15
2 90 5
0
```

3. Saída

A tabela abaixo descreve o formato esperado da saída de cada comando definido na seção anterior.

ID	Saída	Comentário
1	Ponto **NOME** adicionado com coordenadas (**X**,**Y**).	**NOME** é o conteúdo do nó adicionado, e **X**,**Y** são suas coordenadas x e y respectivamente
2	Ponto **NOME** removido!	**NOME** é o conteúdo do nó removido
3	**NOME**	**NOME** é o conteúdo do nó correspondente às coordenadas de entrada
4	**NOME1 NOME2 … NOMEN**	**NOME1**, **NOME2** e **NOMEN** são os conteúdos dos nós cujas coordenadas estão dentro da região formada pelos dados da entrada. Os conteúdos devem ser separados por um espaço em branco, e ao fim do processo deve-se imprimir um \n. Obs: Deve existir um espaço após o conteúdo do último nó encontrado, antes do \n
5	**I; V; NOME (X,Y); I; …;**	**I** representa um nó interno; **V** representa um nó vazio; para nós folhas imprime-se o seu conteúdo (**NOME**), e suas coordenadas x e y entre parêntesis separadas por uma vírgula. Toda a sequência deve ser impressa em uma única linha, separando o conteúdo de cada nó com um ponto-e-vírgula (;) seguido de um espaço, e terminada em \n. Obs: Deve existir um espaço após o último ponto e vírgula, antes do \n
0	Sistema encerrado.	Antes de encerrar o sistema, é necessário liberar toda a memória alocada

Exemplo de saída:

```
Ponto Chicago adicionado com coordenadas (35,42).
```

```
Ponto Mobile adicionado com coordenadas (52,10).
Ponto Toronto adicionado com coordenadas (62,77).
Ponto Buffalo adicionado com coordenadas (82,65).
Ponto Denver adicionado com coordenadas (5,45).
Ponto Omaha adicionado com coordenadas (27,35).
Ponto Atlanta adicionado com coordenadas (85,15).
Ponto Miami adicionado com coordenadas (90,5).
Toronto
I; V; I; Toronto (62,77); V; V; Buffalo (82,65); I; Denver (5,45);
I; Chicago (35,42); V; Omaha (27,35); V; V; V; I; V; V; Mobile
(52,10); I; Atlanta (85,15); V; V; Miami (90,5);
Toronto Buffalo
Ponto Miami removido!
Sistema encerrado.
```

4. Informações

- Este laboratório possui **peso 4**.
- **Não** há um número máximo de submissões.
- No início de cada arquivo a ser submetido, insira um comentário com seu nome, RA e uma breve descrição do conteúdo do arquivo.
- Apenas comentários no formato `/* comentário */` serão aceitos. Comentários com `//` serão acusados como erros pelo SuSy.
- Os cabeçalhos permitidos são:
  - `stdio.h`
  - `stdlib.h`
  - `string.h`
  - `math.h`
- A submissão da sua solução deverá conter múltiplos arquivos:
  - **arvore_quaternaria.h**: interface da estrutura de árvore quaternária
  - **arvore_quaternaria.c**: implementação da estrutura de árvore quaternária
  - **lab08.c**: cliente que utiliza a estrutura
- Será disponibilizado um **Makefile** para este trabalho na página do laboratório:
  - Para compilar seu projeto, basta navegar até o diretório do projeto e utilizar o comando ‘make’ no terminal do Linux.
  - Veja mais instruções na página da disciplina.

- Toda a memória alocada deve ser liberada adequadamente ao fim do programa. Isso pode te ajudar a evitar a ocorrência de falhas relacionadas a memória.
- Indente corretamente todo o seu código. Escolha entre espaços ou tabs para indentação e seja coerente em todos os arquivos. Indentação é fundamental para a legibilidade do seu código, e ajuda no entendimento do fluxo de controle do seu programa.

5. Critérios de Avaliação

- Seu código deverá passar por todos os casos de teste do SuSy definidos para este laboratório. Caso positivo, utilizaremos os seguintes critérios adicionais:
    - A **não utilização/implementação da estrutura árvore quaternária** resultará em **nota zero**
    - Falha na implementação do comportamento esperado para árvores quaternárias acarretará em uma penalização
    - Implementação de uma busca exaustiva, isto é procurar pelo ponto (~~ou região~~) em todos os quadrantes possíveis, resultará em uma penalização
    - Utilização de bibliotecas ou funções não permitidas pelo enunciado resultará em uma **penalização severa** na nota