TRABALHO PRÁTICO 0:

QUADTREE

Marlon Junior Barbosa Marques – 2012049839 – mmarques@dcc.ufmg.br

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais

25 de março de 2013

Resumo. *Esse trabalho tem como objetivo o estudo de uma estrutura de dados conhecida como Quadtree, utilizada, entre outros, para a indexação de pontos bidimensionais no espaço. A documentação do trabalho explica os algoritmos utilizados para a implementação dessa estrutura, além de procurar explicar os benefícios e defeitos de suas principais funções, a saber, inserção e consulta de pontos, através de análises de complexidade e experimentos.*

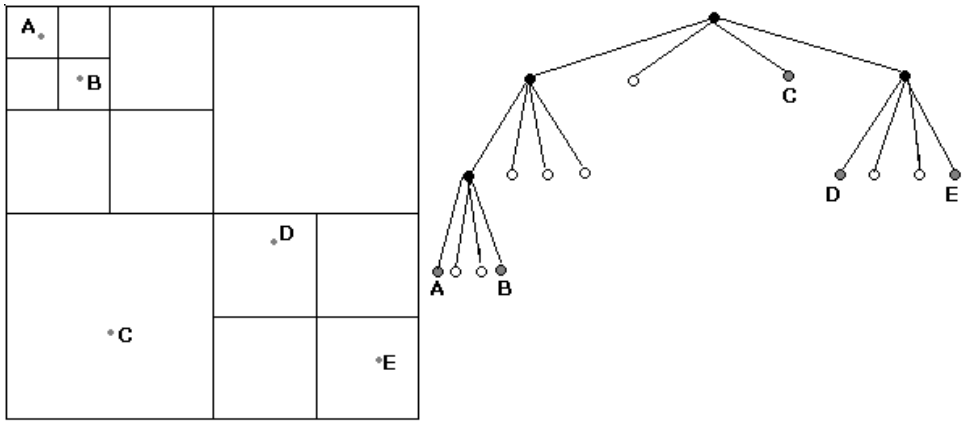
1. INTRODUÇÃO

As Quadtrees são definidas como estruturas de dados cujo princípio vital é a decomposição recursiva do espaço. Tais estruturas são utilizadas principalmente para particionar espaços bidimensionais, dividindo-os recursivamente em quatro regiões, ou quadrantes. Essa técnica pode ser útil, por exemplo, na consulta de coordenadas em um mapa ou na detecção de colisões de objetos em videogames. Neste trabalho, será apresentada a Point Region Quadtree, ou simplesmente PR Quadtree, que nada mais é do que uma adaptação da árvore binária para pontos do espaço bidimensional. Seus principais métodos terão suas complexidades analisadas, e espera-se, através de tal análise, descobrir seus principais benefícios e falhas.

2. MODELAGEM E SOLUÇÃO PROPOSTA

A Quadtree deste trabalho funcionará dividindo seus quadrantes sempre em regiões de áreas iguais, ou seja, em quadrados. Começa-se com uma região do plano cartesiano, que vai de 0 até X e de 0 até Y. Nesta região serão inseridos pontos que, obviamente, estão contidos no plano, até que se alcance o limite máximo de pontos por quadrante (valor pré-estabelecido), quando a região original é subdivida em quatro

quadrados. Tal processo se repete recursivamente quantas vezes forem necessárias. A figura abaixo ilustra o procedimento.



Na parte esquerda da figura, observamos a descrição de uma Quadtree no plano. Tomou-se 1 como o limite de pontos por quadrante, assim cada vez que um segundo ponto é inserido em um quadrante, ele se fragmenta em quatro quadrados. Na parte da direita, observamos a representação de uma Quadtree através de nós, método mais comum na computação. É fácil observar que os cinco pontos (A,B,C,D,E) foram alocados em nós (ou quadrantes) diferentes e que o número de nós folhas (nós que não tem nenhum filho) corresponde ao número de quadrantes. No exemplo acima, após a inserção de todos os pontos, temos 13 nós folhas, ou seja, 13 quadrantes.

Tendo visto como se dá a inserção de pontos em uma Quadtree, pode-se agora explicar, através de um pseudocódigo, as estruturas e as funções utilizadas a fim de se implementá-la. Tem-se três estruturas essenciais:

```
Struct ListadePontos

//Lista encadeada que contem o(s) ponto(s) de cada quadrante

{

   Double x, y;

   ListadePontos* ProximoPonto;

}

Struct Node

//Lembrando que cada nó é considerado um quadrante e / /
/ //pode apontar para quatros outros quadrantes (se não for um nó folha)

{

  Struct Node*Q1, *Q2, *Q3, *Q4;

  Int numMaxPontos;
```

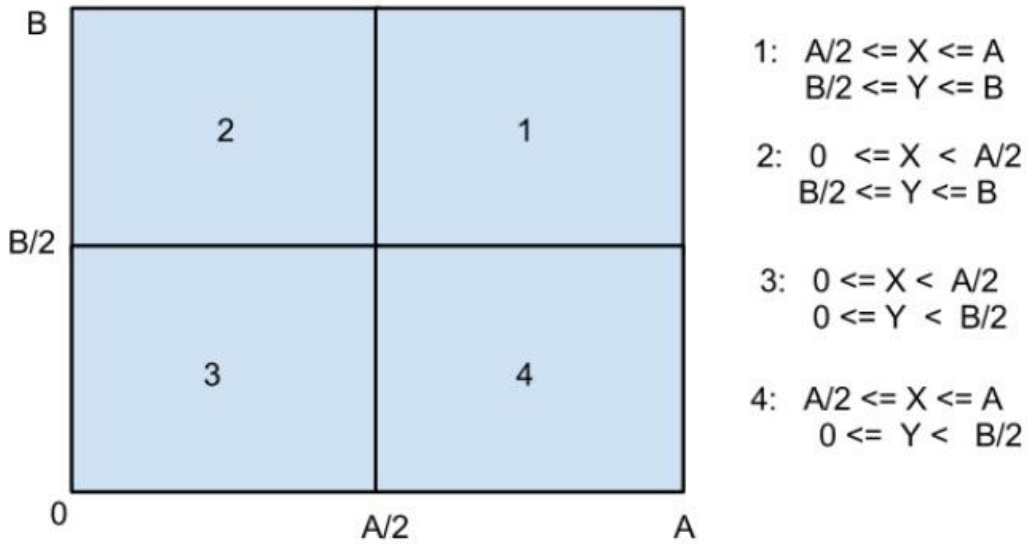
```
    Int numPontos;
    int estáDividido;
    ListaDePontos* lista;
  }
  Struct Arvore
  //Raiz aponta para o primeiro nó, e NúmerosQuadrante é uma variável auxiliar
////  //que nos diz quantos nós folhas a árvore tem
  {
    Struct Node*Raiz;
     Int NúmerosQuadrante;
  }
```

E dois métodos (funções) principais:

```
  Função InserePonto (Node N, double x, double y)
  {
     Se (N->numPontos<numMaxPontos) InserePontoNaLista(N->lista);
     SeNão
    {
          Se (!N->estáDividido) DivideQuad(N);
          Se (PontoPertenceaQ1) InserePonto(N->Q1, x,y);
          SeNão
          Se (PontoPertenceaQ2) InserePonto(N->Q2, x,y);
          SeNão
          Se (PontoPertenceaQ3) InserePonto(N->Q3, x,y);
          SeNão
          Se (PontoPertenceaQ4) InserePonto(N->Q4, x,y);
    }
```

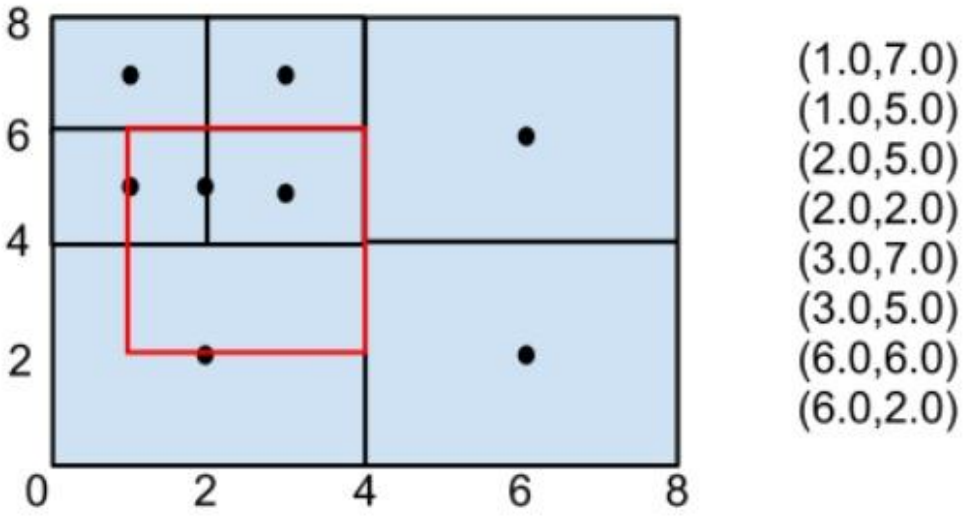
```
Função DivideQuad(arvore A, node N)
{
        A->NumerosQuadrante = NumerosQuadrante + 3;
        N->Q1 = NovoNode();
        N->Q2 = NovoNode();
        N->Q3 = NovoNode();
        N->Q4 = NovoNode();
    For (N->ListaPontos(Inicio)) To N->(ListaPontos(Fim))
        {
        Se (PontoPertenceaQ1) InserePonto(N->Q1, x,y);
        SeNão
        Se (PontoPertenceaQ2) InserePonto(N->Q2, x,y);
        SeNão
        Se (PontoPertenceaQ3) InserePonto(N->Q3, x,y);
        SeNão
        Se (PontoPertenceaQ4) InserePonto(N->Q4, x,y);
}
```

Observando que as condições PontoPertenceaQx (onde x=1, x=2, x=3 ou x=4) são simples funções que verificam em qual quadrante o ponto se encaixa. Tal critério está presente na especificação do TP e pode ser visto na figura abaixo.



Ainda vale ressaltar que nessa implementação não é permitida a inserção de pontos repetidos, e, caso isso ocorra, haverá um erro. Mas como está escrito na especificação do trabalho que tal caso não será cobrado, ele não foi tratado.

Falou-se de como inserir elementos em uma Quadtree e como dividir seus quadrantes de forma recursiva. Entretanto, há ainda outra função importante que não foi explorada: a consulta. A busca em uma Quadtree pode retornar não só um ponto, mas uma lista de pontos, que são varridos por um retângulo delimitados pelas coordenadas de seus vértices. A figura abaixo, tirada da especificação do TP, ilustra o processo.



Observa-se o retângulo na figura acima, delimitado em vermelho, e os pontos que ele engloba: 1.0 5.0, 2.0 2.0, 2.0 5.0, 3.0 5.0.

O algoritmo para implementar a função Consulta é muito parecido com o algoritmo para dividir e inserir da Quadtree, em relação a sua natureza recursiva:

```
Função ConsultaArvore(node N, coordenadas do retângulo: X1Y1, X2Y2)
{
    ListadePontos resposta = CriaListaVazia();
    Se (NÉFolha)
        {
        For (ListaPontos(Inicio)) To (ListaPontos(Fim))
         Se(PontoEstáContidoemRetângulo) InserePontoLista(resposta);
        }
    SeNão
    {
       Se(FazIntersecao(Retangulo,N->Q1)) ConsultaArvore(N->Q1,X1Y1,X2Y2);
       Se(FazIntersecao(Retangulo,N->Q2)) ConsultaArvore(N->Q2,X1Y1,X2Y2);
       Se(FazIntersecao(Retangulo,N->Q3)) ConsultaArvore(N->Q3,X1Y1,X2Y2);
       Se(FazIntersecao(Retangulo,N->Q4)) ConsultaArvore(N->Q4,X1Y1,X2Y2);
    }
```

Uma observação importante, neste caso, é de que o algoritmo implementado só funciona se forem passados os vértices sudoeste e nordeste, como demonstrado na especificação do trabalho.

É claro que os algoritmos acima explicitados foram escritos de forma bem simples, e várias funções auxiliares são necessárias para criar a Quadtree de fato. No código fonte deste trabalho, escrito na linguagem C de programação, encontram-se todas as funções necessárias para esse fim. Convém lembrar que, no código, todas as estruturas foram alocadas dinamicamente e existem funções que liberam a memória alocada para elas. A fim de se compilar e executar o programa, este trabalho incluiu um arquivo Makefile, que automatiza e facilita essas operações. Basta localizar a pasta do trabalho no terminal e digitar um desses comandos:

1. make: Isto compilará o programa, gerando os arquivos objeto e binário, para executar o programa depois pode-se digitar **./tp0 input.txt output.txt** ou digitar o comando abaixo.

2. make run: Este comando compila o programa e automaticamente o roda, com os parâmetros input.txt e output.txt

3. make clean: Remove todos os arquivos objeto e executáveis.

Observando que o programa requer que os arquivos input.txt e output.txt sejam passados como argumentos na hora de execução, como explicado na especificação.

3. ANÁLISE DE COMPLEXIDADE

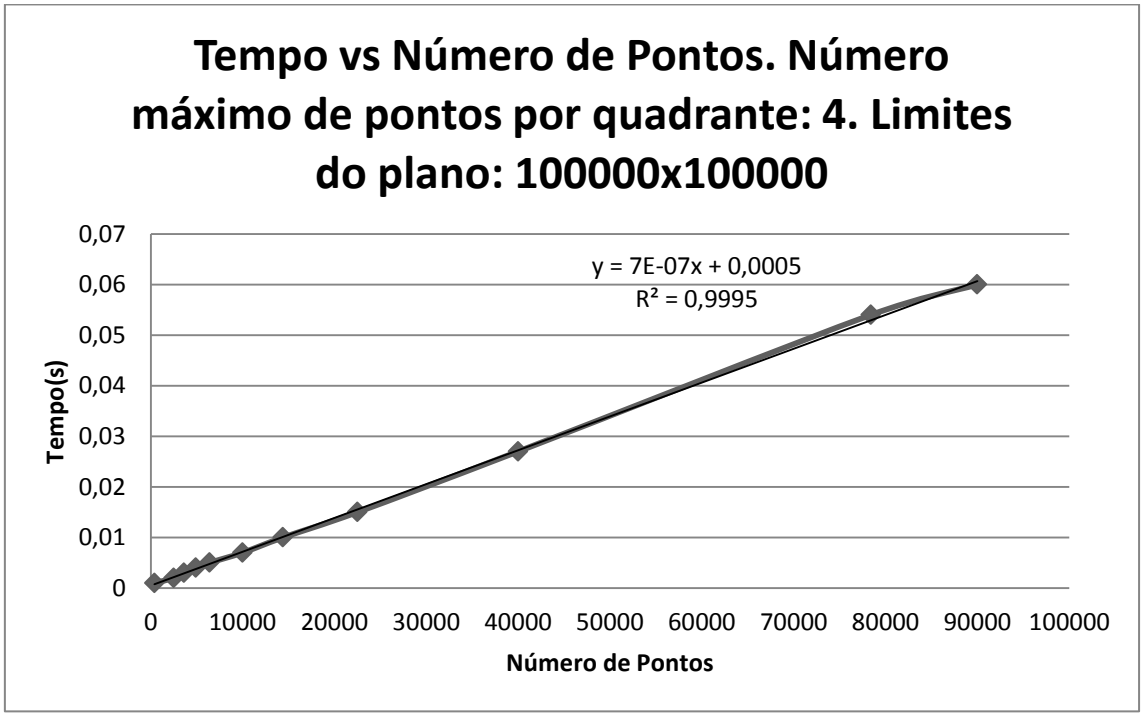
Verifica-se importante fazer uma análise de complexidade dos algoritmos apresentados, isto é, estudar o custo temporal das funções em função do número de entradas. Entretanto, não somente o número é importante, como também a distribuição da entrada em questão, que é neste caso, os pontos. Quando se geram pontos aleatórios, obtém-se o caso médio, onde a Quadtree é homogênea e seus pontos são bem distribuídos pelos quadrantes. Mas quando se geram pontos concentrados em uma determinada região da Quadtree, obtém-se seu pior caso, pois a árvore só cresce em determinada direção, funcionando praticamente como uma lista encadeada. Tem-se assim que o custo no caso médio para inserir um ponto é $O(N\log_4 N)$, onde N é o número de pontos da árvore, e $\log_4 N$ corresponde a sua altura. Tal constatação é de fácil entendimento, uma vez que a PR Quadtree tem natureza muito similar às Árvores Binárias de Busca. Enquanto nesta, a cada chamada, a função descarta $\frac{1}{2}$ de possibilidades, naquela a função descarta $\frac{3}{4}$, portanto a proporcionalidade logarítmica. Entretanto, o seu custo depende do número de pontos que já estão inseridos na árvore, o que causa muitas subdivisões, daí a proporcionalidade linear. Como a cada N pontos, tem-se c.N divisões, onde $0 \leq c \leq 1$, chega-se a complexidade de $O(N\log_4 N)$.

A complexidade da consulta com retângulo, descrita na seção acima, não é difícil de ser calculada. Seja N o número de pontos e $\log_4 N$ a altura da árvore. Tem-se que, no pior caso, o algoritmo terá que ir até a parte mais baixa da Quadtree, o que nos dá uma complexidade de $O(\log_4 N)$.

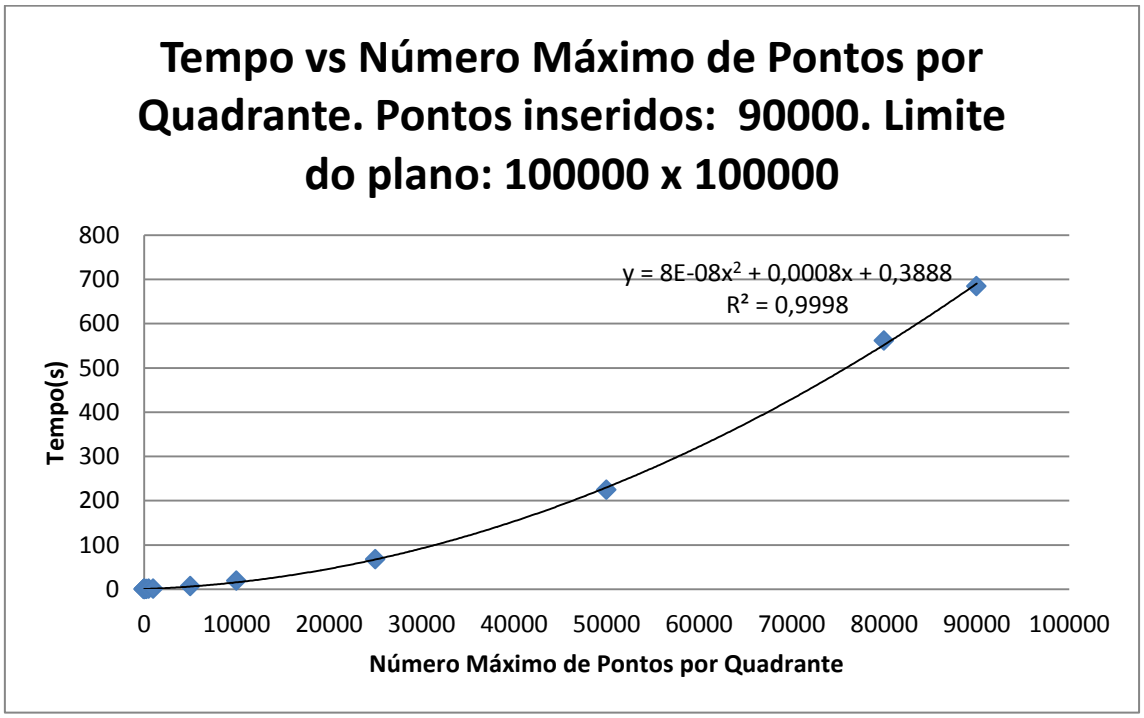
4. EXPERIMENTOS

A medida real do tempo para se executar os algoritmos presentes neste trabalho é de fundamental importância para entender o desempenho do algoritmo em termos práticos. Para tal, realizaram-se diversos experimentos, os quais consistem, basicamente, em aumentar o número de pontos a serem inseridos até o limite máximo determinado na especificação do trabalho (100000 pontos). Também se variou o número máximo de quadrantes e o limite da região plana. Entretanto, apenas o primeiro resultou em mudança significativa do tempo para grandes quantidades de pontos.

Todos os pontos inseridos na árvore para os experimentos foram gerados através de um laço duplo, que a cada iteração aumenta a coordenada por uma unidade. Por exemplo, quando se geram 90000 pontos, inserem-se pontos inteiros desde (0,0) até (300,300). Como visto na seção anterior, o algoritmo de inserção tem seu pior caso quando os pontos a serem inseridos estão condensados em uma certa região da árvore. Usando este modelo para o experimento, procuramos retratar seu pior caso, uma vez que os pontos a serem inseridos são bem próximos uns dos outros, com relação aos limites da região. Se os pontos fossem gerados de maneira aleatória, certamente ocorreria o caso médio, e os tempos observados seriam menores. Entretanto, na especificação do TP consta que pontos iguais não são inseridos na árvore, e isso pode acontecer com números aleatórios sendo gerados; possibilidade que não é admitida nesta implementação. Observou-se ainda que, ao aumentar o número máximo de pontos por quadrante, o tempo demandado aumenta significativamente. Os gráficos abaixo demonstram os resultados.



Podemos observar do gráfico que para valores muito grandes, a função se torna linear. Tal fato é compatível da análise de complexidade, que nos deu um algoritmo O(nlgn).



A variação de número máximo de pontos por quadrante se mostra quadrática para valores elevados.

5. ESPECIFICAÇÃO

A compilação do programa e os experimentos acima descritos foram realizados em uma máquina cuja CPU é um Intel Core 2 Duo E7500 @2.93GHz, com memória RAM de 3.0 GiB e o sistema operacional Linux Ubuntu 10.04 (Lucid).

6. CONCLUSÃO

Quadtrees são estruturas de dados recursivas utilizadas principalmente para indexar espaços bidimensionais. A PR Quadtree, abordada neste trabalho, sempre particiona o espaço em quadrados, com cada nó folha contendo referências a pontos espaciais. Observou-se que este tipo de estrutura pode ser muito eficaz quando os dados são homogêneos, ou seja, igualmente distribuídos pelo plano. Entretanto, seu

desempenho cai quando os pontos ficam concentrados em uma determinada região, e despenca bruscamente quando o número máximo de pontos por quadrante é elevado.

7. REFERÊNCIAS

Winder, Ransom. Examples of a Point-Region Quadtree with moving points or a “Dynamic PR Quadtree”. 29 Semptember 2000.

< http://www.cs.umd.edu/~mount/Indep/Ransom/>

Voorsluys, Bárbara L. Voorsluys, William. Quadtree.

<http://www.lcad.icmc.usp.br/~nonato/ED/Quadtree/quadtree.htm>