Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Algoritmos e Estruturas de Dados III
1º Semestre de 2013

Trabalho Prático 0 - Quadtree

Esse trabalho prático temo como objetivo familiarizar o aluno com primitivas básicas da linguagem C (estrutura de dados, modularizadão, alocação dinâmica de memória e compilação do código através da ferramenta Makefile) e padrões de documentação da disciplina. Neste trabalho, o aluno deve mapear pontos em um espaço bidimensional utilizando a estrutura de dados conhecida como *Quadtree* e dado um retângulo $(X_1, Y_1; X_2, Y_2)$ retornar os pontos que ele abrange.

Problema

Quadtree é uma técnica simples de indexação espacial. Ela é um grafo onde cada vértice representa um quadrante do espaço que esta sendo mapeado, sendo que o vértice raiz cobre toda a área representada. Cada vértice ou é uma folha, neste caso contendo um ou mais pontos e nenhum vértice filho, ou é um vértice interno, neste caso possuindo exatamente quatro filhos, um para cada quadrante obtido por dividir sua área responsável em quatro sub quadrantes de mesma área.

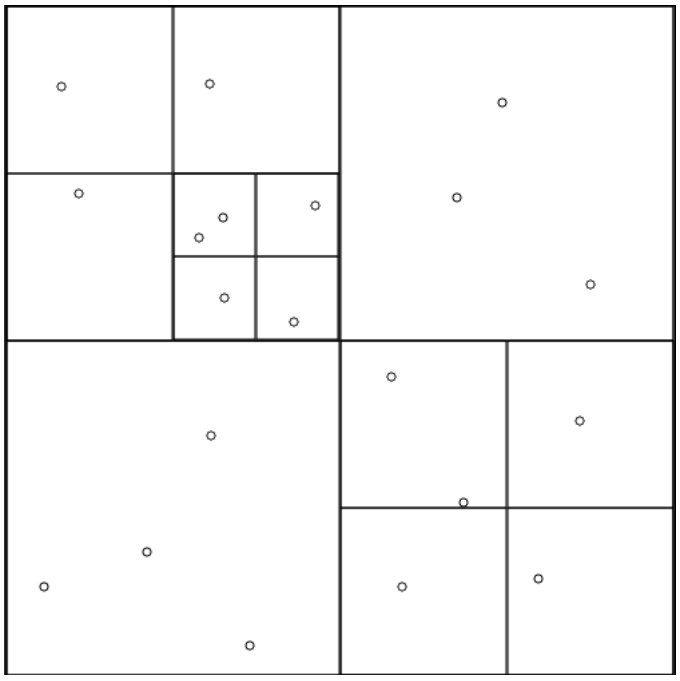

Figura 1: Quadtree com no máximo 4 pontos por quadrante

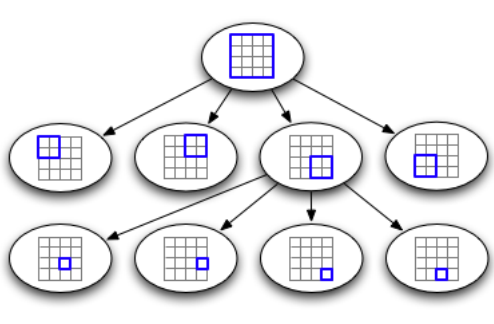

Figura 2: Representação de como uma quadtree é organizada internamente

A inserção de dados em uma quadtree é simples: Começando pelo vértice raiz, descubra em qual quadrante o ponto a ser inserido ocuparia. Reuse esta estrategia ate encontrar um vértice folha. Quando encontrar, adicione o ponto à lista de pontos deste vértice. Caso a lista exceda o limite máximo pré-determinado de elementos, divida o vértice, e mova seus antigos pontos para os respectivos sub-vertices.

Afim de definir a qual quadrante o ponto pertence, utilize a seguinte regra:

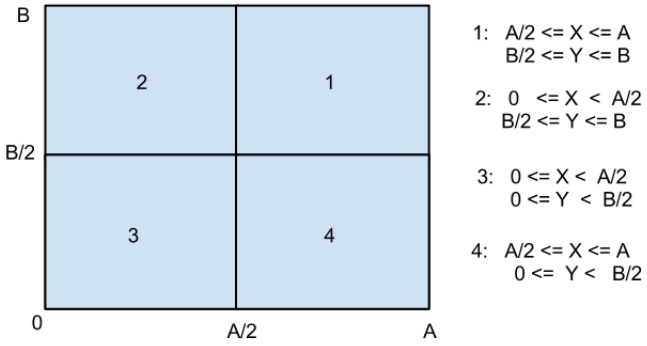

Figura 3: Critério de escolha de quadrante

Apos mapeado todos os pontos na *Quadtree*, o problema se resume em, dado retângulos (X_1, Y_1, X_2, Y_2) especificados no arquivo de entrada, consultar quais pontos estão contidos em cada um deles.

Entrada e Saída

O programa devera solucionar múltiplas instancias do problema em uma única execução. Os pontos a serem inseridos na *Quadtree* devem ser lidos de um arquivo de entrada e o resultado do programa deve ser impresso em um arquivo de saída. Ambos arquivos devem ser passados por parâmetro na chamada do executável:

./tp3 input.txt output.txt

O arquivo de entrada possui um inteiro K na primeira linha onde K é o número de instâncias a serem simuladas. Em seguida as K instâncias são definidas da seguinte forma. A primeira linha possui dois inteiros, representando o limite x,y do espaço estudado (lembrando que consideraremos

apenas o primeiro quadrante do plano cartesiano: $0 \leq X, 0 \leq Y$). A segunda linha possui um inteiro V indicando o máximo de pontos permitidos em cada quadrante. A terceira linha possui um inteiro N indicando quantos pontos distintos serão utilizados ($0 \leq N \leq 100.000$). As próximas N linhas possuirão dois números de ponto flutuante [**double**] (separados por espaço representando as coordenadas x,y de cada ponto. Apos isto, haverá uma nova linha contendo um inteiro M, que representa o número de consultas que deve ser feitas na *Quadtree*. As próximas M linhas, possuem dois pares números de ponto flutante [**double**] (x_1y_1, x_2y_2) representando o retângulo, que será utilizado na consulta, da seguinte maneira:

Figura 4: Representação do retângulo

Para cada instância, deve ser impresso no arquivo de saída o número de quadrantes gerados pela *Quadtree* e M linhas, cada linha contendo todos os pontos que cada retângulo englobou. Lembre-se de ordená-los em ordem crescente em X. Em caso de empate utilize o valor de Y para desempatar. Pode-se utilizar a função *qsort()*.

Exemplo

A seguir temos um exemplo de funcionamento do programa:

Entrada:

```
1
8 8
4
8
1.0 7.0
1.0 5.0
2.0 5.0
2.0 2.0
3.0 7.0
3.0 5.0
6.0 6.0
6.0 2.0
1
1.0 2.0 4.0 6.0
```

Saída:

```
7
1.0 5.0, 2.0 2.0, 2.0 5.0, 3.0 5.0
```

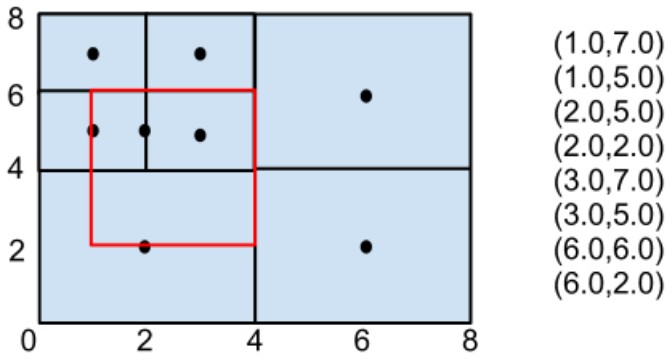

Figura 5: Estrutura do exemplo

Entrega

- A data de entrega desse trabalho é **25 de Março**.
- A penalização por atraso obedece à seguinte fórmula $2^{d-1}/0.32\%$, onde d são os dias úteis de atraso.
- Submeta apenas um arquivo chamado <numero_matricula>_<nome>.zip. Não utilize espaços no nome do arquivo. Ao invés disso utilize o caractere '_'.
- Não inclua arquivos compilados ou gerados por IDEs. **Apenas** os arquivos abaixos devem estar presentes no arquivo zip.
  - Makefile
  - Arquivos fonte (*.c e *.h)
  - Documentacao.pdf
- Não inclua **nenhuma pasta**. Coloque todos os arquivos na raiz do zip.
- Siga rigorosamente o formato do arquivo de saida descrito na especificação. Tome cuidado com whitespaces e formatação dos dados de saída
- NÃO SERÁ NECESSÁRIO ENTREGAR DOCUMENTAÇÃO IMPRESSA!
- Será adotada **média harmônica** entre as notas da **documentação e da execução**, o que implica que a nota final será 0 se uma das partes não for apresentada.

Documentação

A documentação não deve exceder 10 páginas e deve conter pelo menos os seguintes itens:

- Uma **introdução** do problema em questão.
- **Modelagem e solução proposta** para o problema. O algoritmo deve ser explicado de forma clara, possivelmente através de pseudo-código e esquemas ilustrativos.
- **Análise de complexidade** de tempo e espaço da solução implementada.

- **Experimentos** variando-se o tamanho da entrada e quaisquer outros parâmetros que afetem significavelmente a execução.
  - Espera-se ao menos um gráfico de *Número de Pontos X Tempo de Execução* para diversas instâncias juntamente com sua respectiva análise.
- Especificação da(s) **máquina(s) utilizada(s)** nos experimentos realizados.
- Uma breve **conclusão** do trabalho implementado.

Código

- O código deve ser obrigatoriamente escrito na **linguagem C**. Ele deve compilar e executar corretamente nas máquinas Linux dos laboratórios de graduação.
- O utilitário ***make*** deve ser utilizado para auxiliar a compilação, um arquivo *Makefile* deve portanto ser incluído no código submetido.
- As estruturas de dados devem ser **alocadas dinamicamente** e o código deve ser **modularizado** (divisão em múltiplos arquivos fonte e uso de arquivos cabeçalho .h)
- **Variáveis globais** devem ser evitadas.
- Parte da correção poderá ser feita de forma automatizada, portanto **siga rigorosamente os padrões de saída especificados**, caso contrário sua nota pode ser prejudicada.
- O arquivo executável deve ser chamado tp0.
- **Legibilidade e boas práticas** de programação serão avaliadas.