Univerdade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados II
2º semestre de 2017
Prof. Fernando, Luiz, Raquel

TP2: Mapa Estelar

1 Introdução

Aeris é uma astrônoma sonhadora e seu trabalho é observar e estudar os astros no céu. Todos os dias Aeris recebe informações de um satélite chamado Seph. Seph é um poderoso satélite que emite, ao final da tarde, uma lista de estrelas que foram observadas no céu naquele dia. Para fins de simplificação, ela mapeia as estrelas observadas pelo satélite Seph em um plano cartesiano. Seu trabalho consiste em **duas partes**. Na primeira parte, ela basicamente divide o mapa estelar em quadrantes de forma que **cada quadrante contenha apenas uma estrela**.

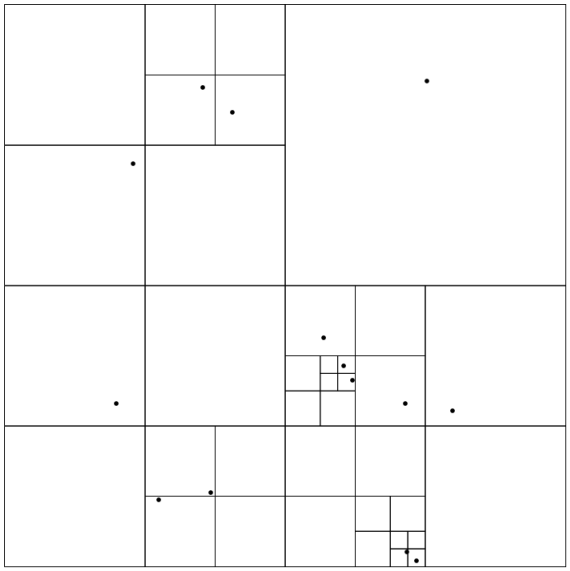

Figura 1: Mapa estelar simples: cada quadrante possui apenas uma estrela.

Para poucas estrelas, o trabalho é relativamente fácil. Basta que Aeris particione o plano em quadrantes e subquadrantes até que cada quadrante abrigue uma ou zero estrelas. Porém, para centenas de estrelas, esse trabalho é exaustivo. Nesse ritmo, o satélite Seph vai acabar matando Aeris.

Aeris conta com a sua ajuda para resolver esse problema e contornar esse cansativo trabalho. Você, estudante de Algoritmos e Estrutura de Dados, observou que cada estrela nada mais é que um **par de coordenadas cartesianas**, e o trabalho de Aeris consiste basicamente em dividir recursivamente o espaço cartesiano de forma a delinear quadrantes que cerquem cada estrela.

2 Detalhes do problema

Para te ajudar a descobrir qual Tipo Abstrato de Dados e quais algoritmos você deve utilizar para resolver este problema, Aeris te deu um exemplo de lista contendo pontos cartesianos que representam estrelas, observados pelo satélite Seph.

```
L = 16
N = 06

(12, 03)
(09, 07)
(15, 15)
(06, 05)
(01, 10)
(13, 13)
```

Aeris explicou que as coordenadas são sempre **inteiros positivos** e que toda instância do problema tem uma largura de quadrante máxima L, que é informada pelo satélite Seph logo no início da Lista, bem como o número de estrelas N. **Por exemplo, para o exemplo acima,** $L = 16$ **e** $N = 06$. Ou seja, a largura máxima é 16 e estão listadas 6 pares de coordenadas cartesianas que representam estrelas. Isso significa que para toda coordenada (x_i, y_i) informada pelo satélite, $0 \leq x_i < L$ e $0 \leq y_i < L$

Com a lista de coordenadas cartesianas em mãos, Aeris desenha os pontos em um plano cartesiano como ilustrado na figura 2.

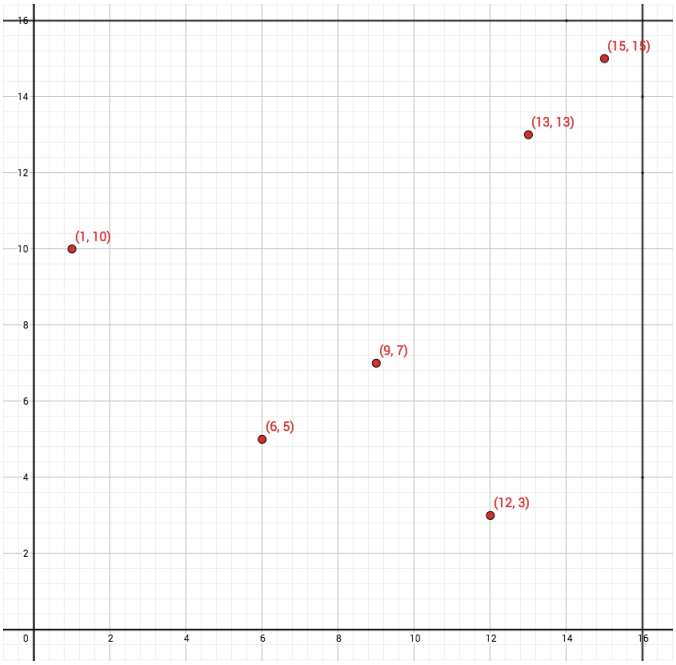

Figura 2: Estado inicial do mapa estelar: um Quadrante maior de lado $l = 16$ engloba todos os pontos da lista.

Como você já observou, o estado inicial dos pontos no mapa estelar não respeita a regra de um ponto por quadrante. No estado inicial há apenas um quadrante de lado $l = L$ e todos os pontos estão contidos nele. Aeris mostrou pra você qual deve ser o **estado final** do mapa na figura 3.

A primeira parte do trabalho de Aeris consiste em partir do estado inicial (Figura 2) e chegar ao estado final (Figura 3). Você concluiu facilmente que esse problema pode ser modelado em uma árvore quádrupla, em que cada nó é um quadrante, e cada nó tem 4 filhos, que são seus subquadrantes.

Aeris, porém, ao ver sua astúcia e sagacidade, resolveu te passar **mais uma tarefa**. Ela lhe disse que o seu trabalho não termina na construção do mapa estelar. Afinal, por quê Aeris precisa de um mapa estelar? Aeris te contou que o mapa estelar é utilizado para o cálculo de forças gravitacionais que as estrelas exercem umas sobre as outras. Mas não se desespere! Aeris não vai exigir que você saiba física. Aeris explica que

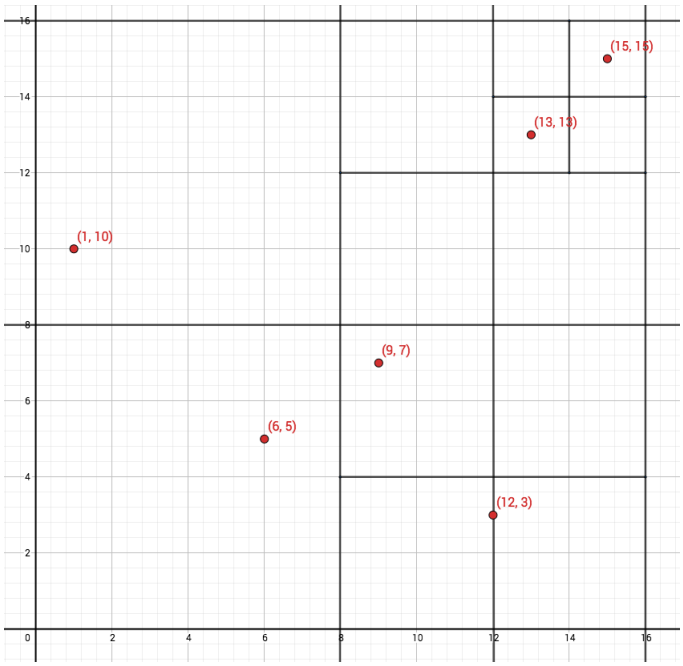

Figura 3: Estado final do mapa estelar.

cada estrela possui um peso associado, e dessa forma, ela deve calcular o peso total de cada quadrante no mapa estelar final.

Seu trabalho é simples: leia os dados das estrelas do satélite Seph (posição no plano cartesiano e peso de cada estrela), crie o mapa estelar e calcule o peso total de cada quadrante.

3 Árvores Quádruplas

Para resolver o problema você chegou à conclusão de que a melhor estrutura para mapear as estrelas é uma **árvore quádrupla**. A interface que implementa a estrutura de árvore quádrupla pode ser observada a seguir:

Bloco de código 1: Árvore Quádrupla

```
typedef struct Ponto Ponto;
typedef struct Estrela Estrela;
typedef struct Quadrante Quadrante;

struct Ponto {
 int x;
 int y;
};

struct Estrela {
 Ponto p;
 int peso;
};

struct Quadrante {
```

```
 Quadrante* NE;
 Quadrante* NW;
 Quadrante* SE;
 Quadrante* SW;

 Estrela* star;

 Ponto centro;
};
```

Uma árvore quádrupla nada mais é que uma árvore em que cada nó possui quatro filhos. A lógica em percorrer uma árvore é a mesma independente do número de filhos. **Você deve respeitar essa interface, pois Aeris a utilizará para leitura dos seus dados**. Porém você pode adicionar mais informação às estruturas, respeitando os nomes das variáveis já existentes.

4 Entrada, saída e exemplos

O que Aeris quer de você?

Aeris vai te passar a lista do satélite Seph e espera que você leia a lista da **entrada padrão**, isto é, com a função *scanf()*. A primeira linha da entrada contem dois números L e N, que representam a largura do quadrante raiz e o número de estrelas, de forma que $8 < L \leq 16000$ e $0 < N < 5000$. Após isso, seguir-se-ão N linhas que representam, cada uma, a informação de uma estrela. Cada linha terá 3 números: X_i, Y_i e P_i, que representam o par de coordenadas da estrela i (X_i, X_i) e seu peso P_i, todos números inteiros. X_i, Y_i respeitam os limites da largura L e P_i é no máximo 2^{20}. Abaixo você vê a entrada relativa ao exemplo da seção 2.

```
16 6
12 3 54
9 7 94
15 15 23
6 5 76
1 10 83
13 13 10
```

Em seguida, Aeris espera que você escreva o mapa estelar na **saída padrão**, isto é, com a função *printf()*. Você deve imprimir o mapa estelar da seguinte forma: para cada quadrante Q_j, imprima o centro do quadrante (Cx_j, Cy_j) e o peso total do quadrante P_j. Como você é esperto, você sabe muito bem que existem diversas formas de se imprimir uma árvore quádrupla. Aeris quer que você imprima de uma forma bem específica. Deve-se imprimir os quadrantes na seguinte ordem: Nordeste (NE), Noroeste (NW), Sudoeste (SW), Sudeste(SE). Ela também disse que a impressão deve ser feita em pós-ordem, ou seja, primeiro imprime-se os filhos na ordem descrita anteriormente, e depois imprime-se os dados do pai. Veja a saída para o exemplo acima:

```
(15, 15) 23
(13, 15) 0
(13, 13) 10
(15, 13) 0
(14, 14) 33
(10, 14) 0
(10, 10) 0
(14, 10) 0
(12, 12) 33
(4, 12) 83
```

```
(4, 4) 76
(14, 6) 0
(10, 6) 94
(10, 2) 0
(14, 2) 54
(12, 4) 148
(8, 8) 340
```

Observe que, devido à impressão em pós-ordem, o centro do Quadrante raiz foi o último a ser impresso, $(8, 8)$ com peso 340. Um pseudocódigo do método de impressão pode ser visto a seguir:

Bloco de código 2: Impressão em pós-ordem

```
void imprimeQuadrante(Quadrante* Q) {
        ...
        imprimeQuadrante(Q->NE);
        imprimeQuadrante(Q->NW);
        imprimeQuadrante(Q->SW);
        imprimeQuadrante(Q->SE);
        imprimeCentroEPesoDoQuadrante(Q);
        ...
}
```

Obviamente, você pode implementar o seu próprio método de impressão, contanto que seja em pós-ordem e respeite a ordem relativa dos quadrantes estabelecido por Aeris.

5 Visualizando suas árvores

Para visualizar sua árvore quádrupla, Aeris também disponibilizou uma biblioteca chamada *dot.h*, que lê um Quadrante raiz Q e imprime a árvore em um arquivo do tipo *dot*. Essa biblioteca já está implementada e está disponibilizada para você no *kit* de desenvolvimento. Ela possui uma função chamada *imprimeArvoreQuadruplaEmArquivoDot()* que recebe um *ponteiro* para um Quadrante raiz e um *nome de arquivo* no qual o arquivo *.dot* será escrito.

Bloco de código 3: Assinatura do método de impressão em arquivo dot

```
void imprimeArvoreQuadruplaEmArquivoDot(Quadrante* raiz, const char* nome_arquivo);
```

Arquivos do tipo *dot* são muito úteis para se visualizar grafos e árvores. Para poder visualizá-lo, você precisa instalar uma ferramenta capaz de ler arquivos *.dot* no seu sistema. Por exemplo, em ambiente *Linux*, basta você executar no terminal:

```
$ apt-get install graphviz
```

Para transformar seu arquivo *.dot* em um arquivo *.pdf*, basta você executar:

```
$ dot -Tpdf meuarquivo.dot -o meuarquivo.pdf
```

Por exemplo, a figura 4 é o arquivo *.dot* gerado pela entrada do exemplo na seção 4.

Aeris quer que você seja capaz de visualizar as suas árvores, então, na documentação do seu Trabalho você deve inserir a imagem de alguma árvore quádrupla gerada pelo seu programa.

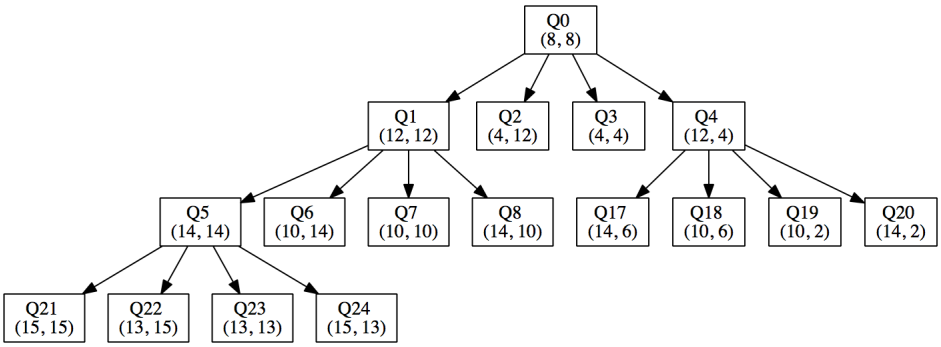

Figura 4: Árvore quádrupla para o exemplo acima.

6 Observações

Preocupada com o seu desenvolvimento, Aeris ponderou algumas observações:

- Para estrelas que estejam exatamente sobre a fronteira de algum quadrante, respeite a seguinte regra: uma estrela na posição (x, y) está localizada em um quadrante Q de limites $(x_{min}, y_{min}, x_{max}, y_{max})$ se, e somente se, $x_{min} \leq x < x_{max}$ e $y_{min} \leq y < y_{max}$.
- A menor largura l de um quadrante Q possível é 1. Isto significa que se houver mais de uma estrela num mesmo quadrante de tamanho mínimo ($l = 1$) ambas devem ser consideradas como uma só estrela com um peso total que equivale à soma dos pesos das duas estrelas. **Isto é, estrelas muito próximas podem ser consideradas como uma só com um peso que equivale à soma de seus pesos.**
- O centro e a divisão dos quadrantes deve ser feita com **divisão inteira**. Dessa forma, um Quadrante raiz de lado $l = 9$ terá seu centro no ponto $(4, 4)$, pois $9/2 = 4$.
- Uma forma fácil de verificar se parte de seu programa está correto é somar os pesos de todas as estrelas e verificar se bate com o peso do quadrante Raiz.

7 Arquivos a serem entregues

- *arvorequad.h*: Interface que implementa as estruturas para o desenvolvimento do trabalho. Você deve declarar métodos neste arquivo. Você pode também adicionar informação às estruturas já existentes, se achar necessário.
- *arvorequad.c*: A implementação dos métodos declarados em *arvorequad.h*.
- *main.c*: Seu módulo principal, que deve ler da entrada padrão uma lista de pontos (estrelas) e seus pesos, e escrever na saída padrão uma lista de pontos (centro de cada quadrante) e seus pesos totais. Se você quiser, você pode usar esse método para escrever também o arquivo *.dot* e visualizar a sua árvore durante a fase de testes.
- *inputs/gen.py*: Um gerador de exemplos aleatórios. Este script gera uma lista com N estrelas de peso máximo P, em um quadrante de largura máxima L e em um arquivo de saída determinado pelo usuário. Para executá-lo:

```
python gen.py <L> <N> <P> <ARQUIVO_SAIDA>
```

Obs.: Precisa-se da linguagem **Python** para executá-lo.

- *Makefile*: Arquivo Makefile que já está implementado. Para compilar o código, basta digitar

  ```
  $ make
  ```

  Para executar para alguma entrada qualquer:

  ```
  $ ./TP2 < arquivo_de_entrada.txt
  ```

  Obs.: Você pode encontrar alguns arquivos de entrada exemplos na pasta *inputs*, junto ao gerador de entradas aleatório.

  Para verificar vazamentos de memória:

  ```
  $ valgrind --leak-check=full ./TP2 < arquivo_de_entrada.txt
  ```

  Em ambos você deve especificar o nome do arquivo de entrada.

- *Documentação*: Sua documentação deve contar os seguintes tópicos:

  1. **Introdução**: Descreva o problema.
  2. **Desenvolvimento**: Descreva como você resolveu o problema. Você precisou fazer uso de recursividade? Faça uso de figuras.
  3. **Exemplos**: Anexe um exemplo de entrada, saída e imagem da respectiva árvore (gerada pelo leitor de arquivos *.dot*) original gerado por você de no mínimo 25 nós.
  4. **Análise de complexidade** (extra / opcional): Qual a complexidade do seu algoritmo?
  5. **Análise experimental** (extra / opcional): Usando exemplos de tamanhos diferentes, teste o tempo de execução do seu programa e verifique se os resultados experimentais estão de acordo com a complexidade teórica do programa.
  6. **Conclusão**: Escreva suas conclusões sobre o projeto, suas dificuldades e os resultados obtidos.

8 Considerações finais

- O exercício deve ser implementado na linguagem *C* usando bibliotecas de padrões como *stdio*, *stdlib* e possivelmente *string*. Deve-se entregar todo o código desenvolvido para resolver o problema. Esse código deve ter como arquivo principal o arquivo *main.c*. O *Makefile* deverá ser entregue junto ao código, **principalmente** no caso de alterações do mesmo. O código deve ser organizado e bem comentado.
- Você deve fazer uso de alocação de memória dinâmica. Fique atento para liberar todo espaço de memória alocado antes do término do programa. Gerenciamento de memória é um dos critérios de avaliação.
- É necessário que o seu programa rode em ambiente *Linux*.
- Não se desespere, apesar da longa especificação, este trabalho não é complexo. Basta você fazer a leitura dos pontos, projetar uma forma de quebrar os quadrantes em subquadrantes **recursivamente** e calcular o peso total de cada quadrante criado.
- Comece a fazer o programa hoje! Boa sorte.