

Relatório do Trabalho 1 da Disciplina “Você e as Minas”

GABRIEL FERREIRA KURTZ
gabriel.kurtz@acad.pucrs.br

18 de setembro de 2017

Abstract

This article is an attempt to solve a challenge of locating the maximum free rectangle in a hypothetical two-dimensional array, representing a minefield where some of the coordinates have been populated with landmines. The article must present an algorithm that reads such minefield from a set of files and locates the area and position of the largest rectangle free of mines.

It was developed as the first evaluated task for the Algorithms II class.

1. INTRODUÇÃO

Este trabalho procura solucionar o desafio de localizar o maior retângulo livre em um campo minado. Foram disponibilizados no enunciado dez arquivos contendo os casos de teste que serão utilizados, contendo o tamanho de cada terreno, o número de minas terrestres e a localização de cada mina no terreno.

Supõe-se que a ONU está planejando adquirir um terreno em um país onde houve uma guerra na qual foram plantadas diversas minas no solo do país. As localizações das minas são conhecidas, porém o projeto não prevê a retirada de minas e sim localizar dentro do terreno total, uma área retangular livre de minas onde será instalado o complexo.

Os casos de teste iniciam em um terreno total de 10.000 x 10.000 coordenadas,

com 100 minas, e aumentam progressivamente até o décimo caso, que é o maior e cujo terreno mede 100.000 x 100.000 coordenadas e tem 1.000 minas terrestres.

Ao longo do trabalho, tentaremos encontrar uma solução para o problema, e apresentar também casos de teste para cada conjunto de dados que foi fornecido, considerando a eficiência do algoritmo produzido, o tempo gasto em cada situação e perspectivas para casos maiores.

O algoritmo foi desenvolvido na linguagem Java utilizando o Eclipse.

2. LEITURA DOS DADOS

Os dados foram fornecidos em arquivos de texto, contendo na primeira linha três números, sendo que o primeiro é a largura do terreno (X), o segundo é sua altura (Y) e o terceiro é o número total de minas no terreno.

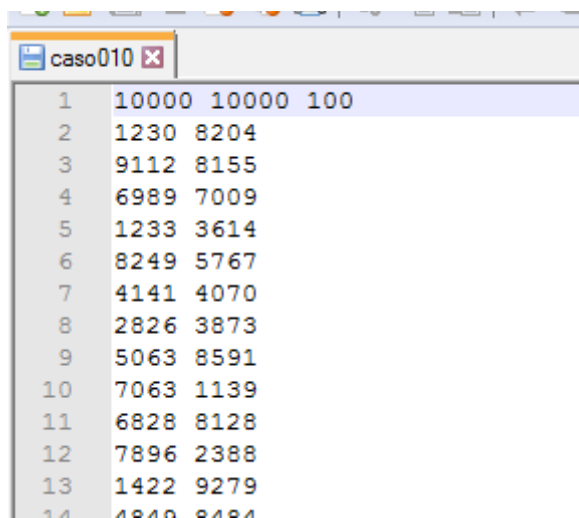
A partir da segunda linha, e até a última, são apresentadas as coordenadas de cada mina no terreno, de modo que cada linha de texto contém dois números, o primeiro sendo a coordenada horizontal da mina (eixo X) e o segundo sua coordenada vertical (eixo Y) e as coordenadas vão de 1 a X e de 1 a Y.

Optamos por realizar a leitura das linhas através de um `BufferedReader`, e cada linha é posteriormente analisada por um `Scanner` que separa cada informação, conforme o código a seguir:

```
String arquivo = in.nextLine();
Path input = Paths.get(arquivo);

try (BufferedReader br = Files.newBufferedReader(input,
Charset.forName("utf8"))) {
    String linha = br.readLine();
    Scanner sc = new Scanner(linha).useDelimiter(" ");

    . . .
}
```



1	10000	10000	100
2	1230	8204	
3	9112	8155	
4	6989	7009	
5	1233	3614	
6	8249	5767	
7	4141	4070	
8	2826	3873	
9	5063	8591	
10	7063	1139	
11	6828	8128	
12	7896	2388	
13	1422	9279	
14	4840	8484	

Figura 1: exemplos de entradas de dados

3. MONTANDO A PRIMEIRA MATRIZ

A primeira ideia aplicada foi montar uma matriz de duas dimensões, onde cada coordenada x,y representa uma posição do terreno, sendo que um conteúdo FALSE indica que não há minas na posição e um conteúdo TRUE indica que há mina.

Cada boolean representa uma informação de 1 bit(V ou F), porém seu tamanho real não é precisamente definido, podendo ocupar mais memória do que isso segundo dados oficiais do Java¹. Para efeitos desta matriz em particular, vamos considerar portanto que cada boolean ocupa 1 byte e não 1 bit, pois além do próprio boolean há as informações da própria matriz.

Tentamos representar o campo minado com a leitura dos dados através do código:

```
int larguraTerreno = sc.nextInt();
int alturaTerreno = sc.nextInt();
boolean[][] campo = new boolean[larguraTerreno][alturaTerreno];

while((linha = br.readLine()) != null) {
    sc = new Scanner(linha).useDelimiter(" ");

    minaX = sc.nextInt() - 1;
    minaY = sc.nextInt() - 1;
    campo[minaX][minaY] = true;
}
```

Embora o caso mais simples (contido no arquivo caso010) tenha inicializado a matriz com sucesso, o caso mais complexo (caso100) não foi possível. Não é difícil compreender a razão: o caso100 representa uma matriz de dimensão 100.000 x 100.000, portanto possui um total de 10.000.000.000 bits a representar. Supondo que cada bit de boolean ocupe um byte na matriz do Java, seriam necessários 9.3GB de memória para comportar esta matriz.

Em termos de memória RAM livre que possa ser acessada pelo eclipse, é um requisito praticamente impossível para um computador de uso pessoal.

Mesmo que fosse feita a opção de trabalhar com a matriz em arquivos, ou salvando seus dados em alguma espécie de banco, ainda assim 9.3GB é elevado demais para o propósito deste estudo, além de ser uma complexidade que não condiz com os objetivos propostos. Certamente o restante do cálculo exigiria um dispêndio de memória ainda maior.

Será necessário, portanto, alterar a abordagem para produzir um algoritmo que solucione corretamente o problema.

As figuras 2 e 3 demonstram o resultado do algoritmo criando a matriz em ambos estes casos de exemplos. O Consumo de Memória foi estimado utilizando os preceitos expostos anteriormente.

¹ <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

```
Problems  Javadoc  Declaration  Console X
<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe

--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---

Insira o nome do arquivo de dados a ser processado:
> caso010
|
--- Inicializando Matriz de Minas para arquivo caso010 ---
10000 10000 100
Largura do Terreno: 10000
Altura do Terreno: 10000
Número de Minas: 100
Total de Coordenadas no Terreno: 100000000
Consumo de Memória: 0.09GB

--- Matriz do Campo inicializada com sucesso ---
```

Figura 2: Criando a matriz no caso010.

```
Problems  Javadoc  Declaration  Console X
<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (1

--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---

Insira o nome do arquivo de dados a ser processado:
> caso100

--- Inicializando Matriz de Minas para arquivo caso100 ---
100000 100000 1000
Largura do Terreno: 100000
Altura do Terreno: 100000
Número de Minas: 1000|
Total de Coordenadas no Terreno: 10000000000
Consumo de Memória: 9.31GB
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at VoceEAsMinas.main(VoceEAsMinas.java:54)
```

Figura 3: Criando a matriz no caso100.

4. REPENSANDO A ABORDAGEM

Claramente, representar o campo inteiro em uma matriz não é a melhor alternativa para solucionar o problema proposto.

Tem-se a sensação que deve haver alguma solução utilizando estruturas de árvores, possivelmente Quadtrees, utilizando os conceitos que foram apresentados em aula pelo Prof. João Batista, porém está difícil encontrar uma aplicação clara.

Infelizmente, apesar de ser um método um tanto quanto bruto, talvez a saída mais pragmática seja reinterpretar o problema mantendo a visão de linhas e colunas, através de coordenadas bidimensionais, porém excluindo os dados desnecessários e utilizando menos memória.

Não estamos mantendo o método de matrizes por considerar uma solução ótima, mas simplesmente por falta de uma ideia melhor.

Uma conclusão importante a ser considerada é que as linhas onde não há minas são todas iguais, ou seja, todas suas coordenadas apresentam a informação FALSE para a presença de minas.

Além disso, podemos afirmar que o maior retângulo livre sempre será adjacente a minas em seus quatro lados, como será demonstrado na próxima figura.

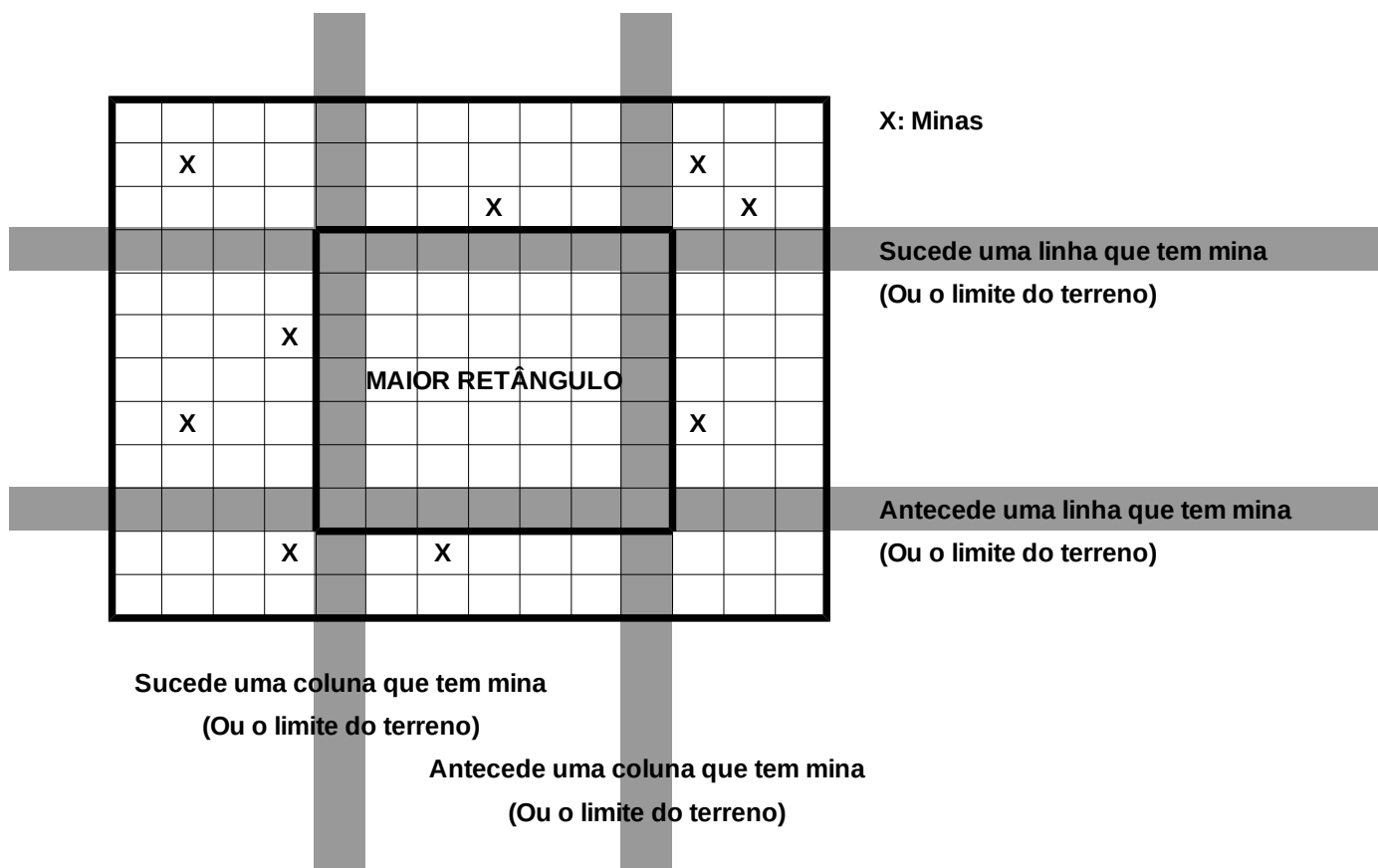


Figura 4: Maior retângulo hipotético.

Por estas razões, talvez seja possível representar o problema sem inicializar a matriz inteira, mas apenas as linhas relevantes para analisar a área fazendo uma

varredura por linhas, iniciando na linha mais de cima, e varrendo da esquerda para a direita em cada linha. Consideramos que as linhas relevantes sejam:

- 1) As linhas que têm minas, pois sem considerá-las é impossível saber qual a área livre em cada linha, bem como determinar as outras linhas relevantes
- 2) As linhas que sucedem as que têm minas, pois a primeira linha de um maior retângulo sempre estará adjacente a uma linha que tem mina, ou a algum dos limite fronteiros do campo minado.
- 3) As linhas que antecedem as que têm minas, pois a última linha de um maior retângulo sempre terá esta característica ou estará na fronteira do campo.

As mesmas propriedades ocorrem com as colunas do campo, embora no momento sua interpretação não seja tao importante, pois o plano é guardar as informações em tuplas de linhas e iniciar por aí a economia de memória.

5. ESTRUTURA DE EIXOS COM OBJETOS

Para a segunda tentativa, os dados do terreno foram organizados em dois vetores unidimensionais, cada um representando um eixo do terreno (largura e altura). Cada vetor contem objetos da classe Linha que foi criada, de modo que cada objeto Linha possui dois atributos:

```
public class Linha {  
    public boolean minada;  
    public HashSet<Integer> posicaoMinas;  
  
    public Linha() {  
        posicaoMinas = new HashSet<Integer>();  
    }  
    public void addMina(int pos) {  
        minada = true;  
        posicaoMinas.add(pos);  
    }  
}
```

boolean minada – Indica se existe mina naquela linha ou coluna. Booleans inicializam em FALSE na linguagem Java.

HashSet<Integer> posicaoMinas – Indica em qual posição do eixo oposto as minas se encontram.

```
eixoX = new Linha[larguraTerreno];  
eixoY = new Linha[alturaTerreno];  
while((linha = br.readLine()) != null) {  
    sc = new Scanner(linha).useDelimiter(" ");  
    int minaX = sc.nextInt() - 1;  
    int minaY = sc.nextInt() - 1;  
  
    eixoX[minaX].addMina(minaY);  
}
```

```
eixoY[minaY].addMina(minaX);
}
```

Perceba que foi utilizado o objeto tipo Linha para instanciar tanto as linhas quanto as colunas, pois o tratamento é o mesmo em ambos os casos. Assim, pode-se manter o tratamento de varredura por linhas e acessar facilmente a informação mais relevante da abordagem proposta, que é saber se uma determinada linha ou coluna possui minas, além de poder acessar facilmente a posição de cada mina fazendo o cruzamento da posição de um objeto no vetor do eixo com o conteúdo do atributo posicaoMinas.

Foram adicionados alguns métodos na classe Linha que permitem adicionar ou remover minas mantendo a sinergia com o atributo minada, bem como um método na classe main que permite copiar um eixo inteiro e todos os objetos Linha dentro dele.

Logramos sucesso em iniciar a matriz com todos os dados necessários em ambos os eixos mesmo no exemplo mais complexo (caso100), e ainda foi possível copiar ambos os eixos, de modo a possuir duas vezes toda a informação necessária na memória. Esta possibilidade de trabalhar com duas cópias das informações necessárias certamente será muito importante no momento de calcular as áreas para encontrar o maior retângulo, pois permite remover minas dinamicamente conforme são realizados os cálculos sem destruir a informação original, permitindo diversas iterações dinâmicas e simultâneas dos dados do terreno inteiro.

```
<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\java
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso100

--- Inicializando Matriz de Minas para arquivo caso100 ---
100000 100000 1000
Largura do Terreno: 100000
Altura do Terreno: 100000
Número de Minas: 1000
Total de Coordenadas no Terreno: 10000000000

--- Matriz do Campo inicializada com sucesso ---

--- Criando cópias dos eixos X e Y ---

--- Cópias criadas com sucesso ---
```

Figura 5: eixos populados e copiados com sucesso no exemplo mais dispendioso.

Embora a mesma informação pudesse ser representada de uma forma ainda mais simples utilizando vetores de HashSets em cada eixo em vez de vetores de objetos, na pesquisa realizada esta opção foi amplamente desencorajada². Ademais, não é possível garantir ainda que será possível ou factível calcular a área apenas com estes dados, e utilizando a classe Linha como uma espécie de *wrapper*, existe a flexibilidade de acrescentar mais atributos conforme necessário.

Com a mudança da perspectiva de representar a matriz inteira para a representação por eixos, foi possível reduzir o consumo de $O(n^2)$ para $O(2n)$ sem perda de dados. Foram realizados alguns testes para verificar a integridade dos dados tanto nos vetores originais quanto nas cópias, e aparentam estar íntegros e conter todos os dados.

6. CALCULANDO A ÁREA

Tendo estabelecido um método viável de uso de memória para guardar a estrutura de dados, a preocupação passa a ser o uso de processamento para calcular a área nos pontos que são candidatos a conter um maior retângulo livre.

A figura 6 representa o mesmo exemplo hipotético da figura 4, que de agora em diante será chamado de caso0, o qual incluímos nos testes do algoritmo pois seu tamanho torna bem mais tangível a interpretação dos dados do programa, visto que o menor caso de exemplo nos arquivos originais era grande demais para ser representado graficamente.

As coordenadas marcadas com a letra P demonstram os pontos que podem ser inícios de maiores retângulos pois são a intersecção de uma coluna que sucede outra coluna minada com uma linha que sucede uma linha minada (ou a linha/coluna inicial em qualquer um dos casos).

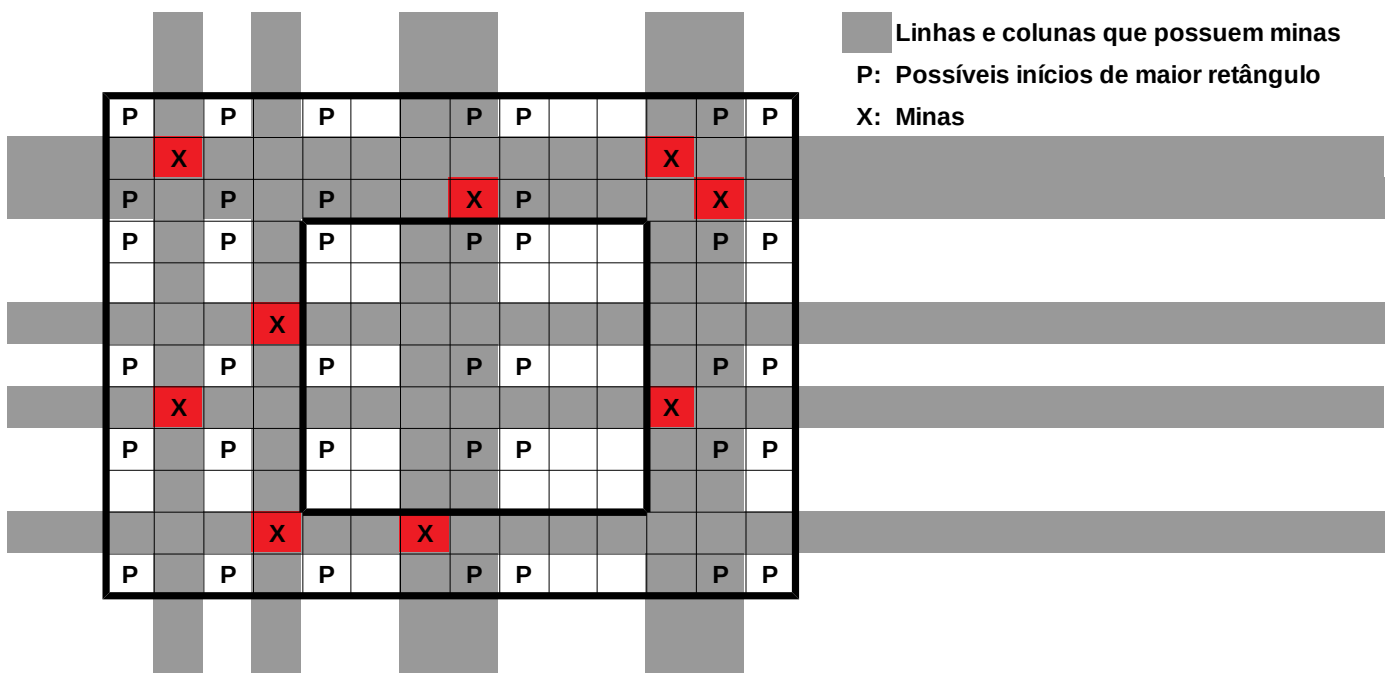


Figura 6: possíveis coordenadas iniciais de maior retângulo pelas propriedades apresentadas

² <https://stackoverflow.com/questions/2992615/how-to-create-an-array-of-a-collection>

Assim conseguimos filtrar bastante o cálculo do maior retângulo no terreno: tendo certeza que o maior retângulo inicia em uma coordenada com estas propriedades, basta calcular qual o maior retângulo que cabe em cada uma delas e compará-los, em vez de calcular o maior retângulo que cabe em todas as coordenadas do terreno.

Em casos onde a distribuição de minas no terreno é mais esparsa do que o caso0 (o que ocorre em todos os demais casos de teste que estamos trabalhando), esta distinção é ainda mais importante. Boa parte dos cálculos do algoritmo passam a ser atrelados à quantidade de bombas no problema, e não necessariamente ao tamanho do terreno (embora este ainda seja checado em diversas varreduras).

Como nesta altura o processamento tornou-se uma prioridade mais importante que a memória, e também para tornar o código mais simples, foram criados mais três vetores para auxiliar na solução do problema. Além disso, estes vetores solucionam também o problema de igualar as linhas iniciais às que sucedem minas, que necessitaria uma redundância de código e processamento para ser solucionado com condicionais:

```
boolean[] podeIniciarMaiorY = new boolean[eixoY.length];
for(int i = 0; i<eixoY.length;i++) {
    if(i==0) {
        podeIniciarMaiorY[i] = true;
    }
    else if(eixoY[i-1].temMina()) {
        podeIniciarMaiorY[i] = true;
    }
}

boolean[] podeIniciarMaiorX = new boolean[eixoX.length];
for(int i = 0; i<eixoX.length;i++) {
    if(i==0) {
        podeIniciarMaiorX[i] = true;
    }
    else if(eixoX[i-1].temMina()) {
        podeIniciarMaiorX[i] = true;
    }
}

boolean[] podeEncerrarMaiorY = new boolean[eixoY.length];
for(int i = 0; i<eixoY.length; i++) {
    if(i==eixoY.length-1) {
        podeEncerrarMaiorY[i] = true;
    }
    else if(eixoY[i+1].temMina()) {
        podeEncerrarMaiorY[i] = true;
    }
}
```

Agora com uma estrutura que contem todos os dados necessários para a solução do problema, o próximo passo torna-se efetuar a varredura para localizar os pontos P que podem conter o início do maior retângulo e compará-los para verificar qual oferece a maior área livre.

Foi utilizada uma estrutura de laços aninhados, onde o laço “de fora” executa a varredura linha a linha (coordenada Y), e dentro dele um laço que executa a varredura coluna a coluna (coordenada X), buscando pontos com a propriedade P.

```
for(int y = 0; y<eixoY.length; y++) {
    if(podeIniciarMaiorY[y]){
        for(int x = 0; x<eixoX.length; x++) {
            if(podeIniciarMaiorX[x]) {
                long maiorAreaDaCoord = 0;
                int areaLivreDireita = eixoX.length-x;
                int auxXF = 0;
                int auxYF = 0;
            }
        }
    }
}
```

Também foram inicializadas globalmente algumas variáveis para guardar o maior retângulo livre encontrado até o momento. Como demonstrado no código anterior, algumas variáveis também foram inicializadas neste ponto do laço para guardar localmente estas mesmas variáveis e poder compará-las com as globais, que estão demonstradas a seguir:

```
long maiorArea = 0;
int maiorX = 0;
int maiorY = 0;
int maiorXF = 0;
int maiorYF = 0;
```

Para cada ponto P localizado, é necessário calcular qual a área do maior retângulo livre de minas que aquela coordenada pode comportar. Para isto, foi executada uma nova varredura do eixo Y, desta vez buscando por linhas que:

- 1) Estão minadas. Caso a mina daquela linha tenha uma coordenada X maior do que a coordenada X do ponto P, ela reduz a área que pode ser comportada em todas as linhas subsequentes daquele ponto P. Esta condição foi definida no atributo int areaLivreDireita.
- 2) Podem ser a última linha do maior retângulo (antecedem uma linha minada ou é a última linha). Neste caso, o algoritmo calcula a área entre o ponto P e a linha Y1 atual, sendo a fronteira final do eixo X medida pela mina mais próxima que foi encontrada na checagem “1”, ou então pelo limite do terreno caso nenhuma mina tenha sido encontrada ainda neste laço que reduza o tamanho do retângulo que cabe ali.

Estes fatores foram considerados e o cálculo é realizado através do seguinte código:

```
for(int y1 = y; y1<eixoY.length; y1++) {
    //Atualiza area livre a direita para coordenada
    if(copiaEixoY[y1].temMina()) {
```

```

        for(int pos : copiaEixoY[y1].posicaoMinas) {
            if(pos>=x && pos-x<areaLivreDireita) {
                areaLivreDireita = pos-x;
            }
        }
    }

    //Calcula area caso possa ser a maior
    if(podeEncerrarMaiorY[y1]) {
        if(((y1-y+1)*(areaLivreDireita)+1) > maiorAreaDaCoord) {
            maiorAreaDaCoord = ((y1-y+1)*(areaLivreDireita));
            auxYF = y1;
            auxXF = x + areaLivreDireita -1;
        }
    }
}

```

Foi utilizada uma cópia do eixo Y, e não o original, sendo que o vetor copiaEixoY[] é exatamente igual ao vetor eixoY[], porém com elementos totalmente novos e não referências aos mesmos elementos. Será explicado melhor este processo posteriormente.

Com este pedaço do código, está concluído o trabalho de calcular a área dos pontos P: este laço, aninhado dentro dos dois laços anteriores, irá calcular o maior retângulo sem minas que inicia em cada ponto P do mapa, garantindo que um deles será o maior do terreno. De posse destes dados, agora basta comparar os valores obtidos em cada ponto do laço com as variáveis globais, onde fica guardado o maior retângulo encontrado até o momento e que, ao final do algoritmo, irá conter o maior retângulo absoluto para ser exibido ao usuário.

Dentro do laço onde foi inicializada a variável maiorArea, após todas as iterações de cada ponto P:

```

if(maiorAreaDaCoord > maiorArea) {
    maiorArea = maiorAreaDaCoord;
    maiorX = x;
    maiorY = y;
    maiorXF = auxXF;
    maiorYF = auxYF;
}

```

Finalmente, basta exibir as informações ao usuário no terminal, acrescentando 1 a cada coordenada pois os vetores estavam usando a notação de 0 a X para cada eixo, enquanto o enunciado solicita a notação de 1 a X. Uma transcrição do código inteiro do algoritmo encontra-se no Anexo II deste trabalho, bem como está disponível publicamente no GitHub³.

3 <https://github.com/guldratz/VoceEAsMinas>

```

System.out.println("--- Maior Área Livre de Minas ---"
+ "\nCoordenadas (X,Y inicial e final)"
+ "\nXi: " + (maiorX + 1) + "   Yi: " + (maiorY + 1)
+ "\nXf: " + (maiorXF + 1) + "   Yf: " + (maiorYF + 1)
+ "\nÁrea: " + maiorArea);

```

7. CASOS DE TESTE

Com o algoritmo concluído, agora podemos passar para a fase de testes. No enunciado do problema, foram solicitados os testes para 10 conjuntos de dados. Os resultados destes 10 casos podem ser encontrados no Anexo I do trabalho. Neste capítulo serão discutidos quatro dos casos de modo a ilustrar melhor o algoritmo.

Optamos por iniciar pelo teste do caso0, que foi criado por nós, primeiro por ser um conjunto de dados que pode ser facilmente representado visualmente neste artigo, e segundo porque possuímos as respostas para este conjunto.

Serão apresentados também três outros casos: um pequeno, um médio e um grande, de modo a comparar o desempenho do algoritmo em diferentes condições.

Em relação ao caso0, conforme as figuras 7 e 8, os dados encontrados nos testes estão condizentes com o que se espera. O maior retângulo livre de minas corresponde ao que já havíamos demonstrado, e suas coordenadas (5,4) e (11,10) condizem com os limites deste retângulo. A área calculada, 49, também corresponde ao esperado.

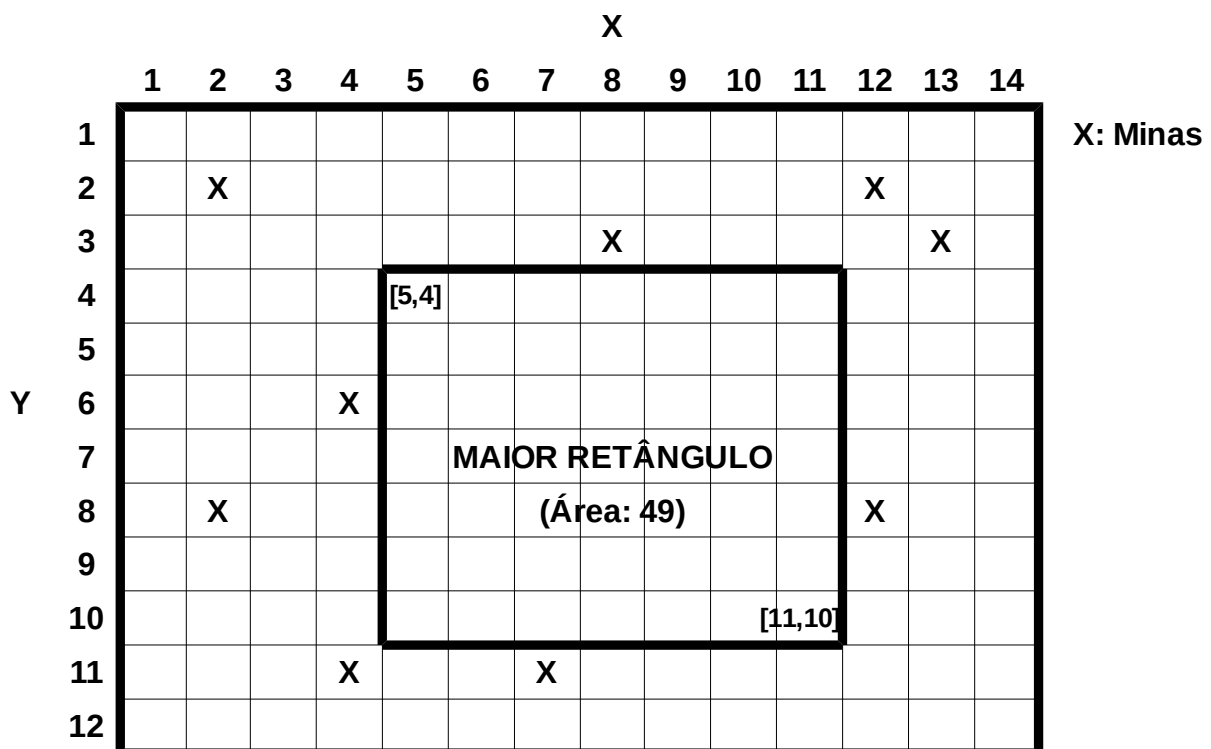


Figura 7: representação visual do caso0.

A execução foi extremamente rápida neste exemplo, também como já era esperado, afinal trata-se de um exemplo bastante pequeno.

Importante ressaltar que as coordenadas localizadas correspondem a pontos que estão dentro do retângulo, porém nos seus vértices. Na forma de representação visual que optamos, tendo o eixo X na vertical crescendo para a direita e o Y na horizontal crescendo para baixo, a coordenada inicial é o vértice superior esquerdo do retângulo, enquanto a coordenada final é o vértice inferior direito.

```
<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso0

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso0 ---
14 12 9
Largura do Terreno: 14 - Altura do Terreno: 12
Número de Minas: 9 - Total de Coordenadas no Terreno: 168
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 5 Yi: 4
Xf: 11 Yf: 10
Área: 49
--- Tempo Total de Execução: 0.01 segundos ---
```

Figura 8: resultado do teste do caso0.

```
<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso010
|
--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso010 ---
10000 10000 100
Largura do Terreno: 10000 - Altura do Terreno: 10000
Número de Minas: 100 - Total de Coordenadas no Terreno: 100000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 329 Yi: 6014
Xf: 5198 Yf: 7497
Área: 7227080
--- Tempo Total de Execução: 0.24 segundos ---
```

Figura 9: teste de problema pequeno (caso010)

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso050
|
--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso050 ---
50000 50000 500
Largura do Terreno: 50000 - Altura do Terreno: 50000
Número de Minas: 500 - Total de Coordenadas no Terreno: 2500000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 10467 Yi: 32959
Xf: 19886 Yf: 38528
Área: 52469400
--- Tempo Total de Execução: 22.15 segundos ---

```

Figura 10: teste de problema médio (caso050)

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso100

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso100 ---
100000 100000 1000
Largura do Terreno: 100000 - Altura do Terreno: 100000
Número de Minas: 1000 - Total de Coordenadas no Terreno: 10000000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 1 Yi: 51915
Xf: 62584 Yf: 53811
Área: 118721848
--- Tempo Total de Execução: 197.8 segundos ---

```

Figura11: teste de problema grande (caso100).

O algoritmo aparenta estar comportando-se como esperado também nos demais casos testados, e conseguiu concluir com sucesso mesmo no caso mais complexo proposto, levando pouco mais de três minutos.

8. TEMPO DE EXECUÇÃO

Para mensurar em segundos o tempo de execução do algoritmo, foi acrescentado um marcador de hora logo após a entrada do nome do arquivo com os dados, antes mesmo de começar a montar a estrutura, e outro após o término de todos os métodos, no momento em que o programa é finalizado. A diferença entre estas duas horas determina o tempo em segundos.

```
//Início do cronômetro
long startTime = System.nanoTime();
//Final do cronômetro
long endTime = System.nanoTime();
double duration = ((endTime*1.0 - startTime)/1000000000);
duration = (Math.round(duration*100)/100.0);
```

Os casos de teste demonstram que o dispêndio de recursos tende a crescer exponencialmente conforme aumenta o tamanho dos dados inseridos. Na realidade, dos três principais elementos que compõem o problema, dois são bidimensionais: o tamanho do terreno e a área a ser calculada. Já o terceiro elemento, o número de minas, é unidimensional.

Portanto, os trechos que forem atrelados ao tamanho do terreno e da área, como por exemplo, varreduras da estrutura inteira, certamente terão um tempo de execução que será, no mínimo, $O(n^2)$. Os trechos do algoritmo que forem possíveis atrelar ao número de minas, tendem a crescer de uma forma mais amena, podendo chegar a $O(n)$.

De modo geral, é aparentemente impossível desenvolver um algoritmo com aplicação prática para este problema que não utilize uma mistura destes dois aspectos. Assim, o fator importante a ser considerado é que se está desenvolvendo algo que irá interagir simultaneamente com elementos bidimensionais e unidimensionais.

Procuramos desenvolver um algoritmo que tivesse a maior relação possível com o número de minas e as posições delas, no entanto o código ainda usa diversas varreduras cujo dispêndio de recursos é proporcional ao tamanho do terreno e às áreas que estão sendo calculadas.

Os casos de teste também possuem características de ambas as situações: a cada aumento do input de dados, havia um aumento tanto do tamanho do terreno em ambas as dimensões quanto do número de minas.

Com base nisto, tentamos desenvolver o equilíbrio que proporcionasse a solução mais eficiente.

9. PERSPECTIVA PARA CASOS MAIORES

Para os casos maiores do que foram solicitados, fica claro que o algoritmo está próximo do quanto pode comportar. Aparentemente, o dispêndio de memória está sob controle, pois está utilizando somente vetores unidimensionais. No entanto, os requerimentos de processamento e, portanto, de tempo, estão crescendo exponencialmente em terrenos maiores, e já estão bastante próximos do que se pode esperar que um computador pessoal resolva em um tempo razoável.

Uma possível solução para casos que extrapolem o limite seria reduzir a precisão do algoritmo. Por exemplo, ao consultar um terreno 10 vezes maior do que o caso100, poderia-se truncar todos os valores de posição (dimensões do terreno e posição das minas), de modo a remover o último dígito.

Com esta abordagem, poderia-se solucionar um problema 10 vezes maior com exatamente o mesmo dispêndio de recursos, mantendo a certeza de que o terreno não teria nenhuma mina dentro dele. O custo seria uma imprecisão de no mínimo 0 e no máximo 9 posições que seriam desperdiçadas fora do terreno, que parece ser bastante razoável para o tamanho das áreas que estão sendo trabalhadas.

A mesma simplificação poderia ser feita para áreas de, virtualmente, qualquer tamanho, sempre mantendo a mesma proporção de imprecisão. O principal fator impeditivo neste caso é que, mantendo o mesmo volume de minas por área nos dados iniciais, certamente haveria um volume bem maior de minas por área na visão truncada, na mesma proporção em que fosse simplificado o problema.

Quanto a aprimorar o algoritmo existente, ainda há algum espaço para aprimoramento, porém nenhuma das opções mais aparentes alterariam o fato do problema crescer exponencialmente. Poder-se-ia, por exemplo, aproveitar melhor o fato dos laços de varredura estarem trabalhando com uma cópia dos dados para destruir as minas que estão em áreas que já foram varridas em cada degrau do laço, evitando assim seu uso redundante.

No entanto, analisando o feedback linha a linha do cálculo (que pode ser reproduzido descomentando as linhas apropriadas no código), fica claro que a análise de cada linha já está ficando muito mais rápida nas linhas finais do que nas iniciais, portanto os condicionais utilizados já estão fazendo um bom trabalho de filtrar a informação irrelevante.

Aparentemente, uma melhora que realmente viabilizasse casos significativamente maiores, com o mesmo volume de minas por área, exigiria repensar completamente a abordagem utilizada.

10. CONCLUSÃO

O algoritmo que foi desenvolvido, supondo que esteja funcionando corretamente (e tudo indica que está), é capaz de solucionar o problema proposto com a maior precisão possível e em tempo razoável.

Foi possível solucionar todos os casos de teste propostos, sendo que o mais demorado levou menos de quatro minutos. Nenhum dado foi ignorado e não houve nenhum arredondamento ou truncamento dos valores. O maior retângulo livre encontrado pelo algoritmo é realmente o maior que existe na amostragem.

Foi um trabalho bastante interessante, que proporcionou muitas oportunidades de pesquisa e aprendizado. Certamente há soluções melhores, mas estamos satisfeitos com o resultado apresentado.

ANEXO I – LISTA COMPLETA DE CASOS DE TESTE

```
<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\jav
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso010
|
--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso010 ---
10000 10000 100
Largura do Terreno: 10000 - Altura do Terreno: 10000
Número de Minas: 100 - Total de Coordenadas no Terreno: 100000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 329 Yi: 6014
Xf: 5198 Yf: 7497
Área: 7227080
--- Tempo Total de Execução: 0.24 segundos ---
```

```
<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\jav
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso020

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso020 ---
20000 20000 200
Largura do Terreno: 20000 - Altura do Terreno: 20000
Número de Minas: 200 - Total de Coordenadas no Terreno: 400000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 764 Yi: 10141
Xf: 3924 Yf: 15192
Área: 15969372
--- Tempo Total de Execução: 2.1 segundos ---
```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\jav
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso030

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso030 ---
30000 30000 300
Largura do Terreno: 30000 - Altura do Terreno: 30000
Número de Minas: 300 - Total de Coordenadas no Terreno: 900000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 19021 Yi: 17617
Xf: 27020 Yf: 21891
Área: 34200000
--- Tempo Total de Execução: 4.92 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso040

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso040 ---
40000 40000 400
Largura do Terreno: 40000 - Altura do Terreno: 40000
Número de Minas: 400 - Total de Coordenadas no Terreno: 1600000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 12496 Yi: 1918
Xf: 13704 Yf: 36562
Área: 41885805
--- Tempo Total de Execução: 12.17 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso050
|
--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso050 ---
50000 50000 500
Largura do Terreno: 50000 - Altura do Terreno: 50000
Número de Minas: 500 - Total de Coordenadas no Terreno: 2500000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 10467 Yi: 32959
Xf: 19886 Yf: 38528
Área: 52469400
--- Tempo Total de Execução: 22.15 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.e
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso060

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso060 ---
60000 60000 600
Largura do Terreno: 60000 - Altura do Terreno: 60000
Número de Minas: 600 - Total de Coordenadas no Terreno: 3600000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 16948 Yi: 35357
Xf: 21079 Yf: 51391
Área: 66256620
--- Tempo Total de Execução: 40.14 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\java
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso070
|
--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso070 ---
70000 70000 700
Largura do Terreno: 70000 - Altura do Terreno: 70000
Número de Minas: 700 - Total de Coordenadas no Terreno: 4900000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 1 Yi: 16788
Xf: 65437 Yf: 18019
Área: 80618384
--- Tempo Total de Execução: 59.42 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso080
|
--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso080 ---
80000 80000 800
Largura do Terreno: 80000 - Altura do Terreno: 80000
Número de Minas: 800 - Total de Coordenadas no Terreno: 6400000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 60673 Yi: 27163
Xf: 62373 Yf: 80000
Área: 89877438
--- Tempo Total de Execução: 94.77 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.e
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso090
|
--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso090 ---
90000 90000 900
Largura do Terreno: 90000 - Altura do Terreno: 90000
Número de Minas: 900 - Total de Coordenadas no Terreno: 8100000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 54024 Yi: 26455
Xf: 56503 Yf: 68464
Área: 104184800
--- Tempo Total de Execução: 132.28 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.e
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso100

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso100 ---
100000 100000 1000
Largura do Terreno: 100000 - Altura do Terreno: 100000
Número de Minas: 1000 - Total de Coordenadas no Terreno: 10000000000
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 1 Yi: 51915
Xf: 62584 Yf: 53811
Área: 118721848
--- Tempo Total de Execução: 197.8 segundos ---

```

```

<terminated> VoceEAsMinas [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw
-----
--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---
-----

Insira o nome do arquivo de dados a ser processado:
> caso0

--- Iniciando Cronômetro ---
--- Inicializando Estrutura de Minas para arquivo caso0 ---
14 12 9
Largura do Terreno: 14 - Altura do Terreno: 12
Número de Minas: 9 - Total de Coordenadas no Terreno: 168
--- Estrutura do Campo inicializada com sucesso ---
--- Procurando Maior Retangulo ---
--- Maior Área Livre de Minas ---
Coordenadas (X,Y inicial e final)
Xi: 5  Yi: 4
Xf: 11  Yf: 10
Área: 49
--- Tempo Total de Execução: 0.01 segundos ---

```

ANEXO II – CÓDIGO DO ALGORITMO EM JAVA

```
//---Classe VoceEAsMinas
import java.io.BufferedReader;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashSet;
import java.util.Scanner;

public class VoceEAsMinas {

    public static void main(String args[]) {

        Scanner in = new Scanner(System.in);

        System.out.print("--- Pontificia Universidade Catolica do Rio Grande do Sul ---"
            + "\n--- Bacharelado em Engenharia de Software ---"
            + "\n--- Disciplina de Algoritmos e Estruturas de Dados II ---"
            + "\n--- Prof. Joao Batista Oliveira ---"
            + "\n--- Gabriel Ferreira Kurtz ---"
            + "\n\n-----"
            + "\n--- ALGORITMO DE DETECÇÃO DE MAIOR RETANGULO EM CAMPO MINADO ---"
            + "\n-----"
            + "\n\nInsira o nome do arquivo de dados a ser processado:"
            + "\n> ");

        String arquivo = in.nextLine();
        Path input = Paths.get(arquivo);

        long startTime = System.nanoTime();

        System.out.println("\n--- Iniciando Cronômetro ---"
            + "\n--- Inicializando Estrutura de Minas para arquivo " + arquivo + " ---");

        Linha[] eixoX = new Linha[1];
        Linha[] eixoY = new Linha[1];

        try (BufferedReader br = Files.newBufferedReader(input, Charset.forName("utf8")))
        {
            String linha = br.readLine();
            Scanner sc = new Scanner(linha).useDelimiter(" ");

            System.out.println(linha);

            int larguraTerreno = sc.nextInt();
            int alturaTerreno = sc.nextInt();
            System.out.println("Largura do Terreno: " + larguraTerreno + " - Altura do Terreno: " +
alturaTerreno);
```



```

        int numeroMinas = sc.nextInt();
        long totalCoord = Long.valueOf(larguraTerreno) * Long.valueOf(alturaTerreno);
        System.out.println("Número de Minas: " + numeroMinas + " - Total de Coordenadas no
Terreno: " + totalCoord);
//
//        double gbMemoria = totalCoord/1073741824.0;
//        System.out.println("Consumo de Memória: " + Math.round(gbMemoria*100)/100.0 + "GB");

        // Inicia vetores do tamanho do terreno com todos os pontos falsos
        eixoX = new Linha[larguraTerreno];
        for(int i=0; i<eixoX.length; i++) {
            eixoX[i] = new Linha();
        }

        eixoY = new Linha[alturaTerreno];
        for(int i=0; i<eixoY.length; i++) {
            eixoY[i] = new Linha();
        }

        //Populando campo com as minas do arquivo e informações relevantes
        int minaX, minaY;
        while((linha = br.readLine()) != null) {
            sc = new Scanner(linha).useDelimiter(" ");

            minaX = sc.nextInt() - 1;
            minaY = sc.nextInt() - 1;

            eixoX[minaX].addMina(minaY);
            eixoY[minaY].addMina(minaX);
        }
    }
    catch (IOException x) {
        System.err.format("Erro de E/S: %s%n", x);
    }

    System.out.println("--- Estrutura do Campo inicializada com sucesso ---"
        + "\n--- Procurando Maior Retangulo ---");

    procuraMaiorArea(eixoX, eixoY);

    long endTime = System.nanoTime();
    double duration = ((endTime*1.0 - startTime)/1000000000);
    duration = (Math.round(duration*100)/100.0);
    System.out.println("--- Tempo Total de Execução: " + duration + " segundos ---");
}

public static Linha[] copiarEixo(Linha[] eixo) {
    Linha[] copia = new Linha[eixo.length];
    for(int i = 0; i<eixo.length; i++) {
        copia[i] = eixo[i].copiar();
    }
    return copia;
}

```



```

    }

    public static void procuraMaiorArea(Linha[] eixoX, Linha[] eixoY) {
        long maiorArea = 0;
        int maiorX = 0;
        int maiorY = 0;
        int maiorXF = 0;
        int maiorYF = 0;

        //Procura linha que pode ser inicio de maior retangulo (Sucede linha com mina ou é a primeira
linha)
        boolean[] podeIniciarMaiorY = new boolean[eixoY.length];
        for(int i = 0; i<eixoY.length;i++) {
            if(i==0) {
                podeIniciarMaiorY[i] = true;
            }
            else if(eixoY[i-1].temMina()) {
                podeIniciarMaiorY[i] = true;
            }
        }

        //Procura coluna que pode ser inicio de maior retangulo (Sucede coluna com mina ou é a primeira
coluna)
        boolean[] podeIniciarMaiorX = new boolean[eixoX.length];
        for(int i = 0; i<eixoX.length;i++) {
            if(i==0) {
                podeIniciarMaiorX[i] = true;
            }
            else if(eixoX[i-1].temMina()) {
                podeIniciarMaiorX[i] = true;
            }
        }

        //Procura linha que pode ser final de maior retangulo (Antecede linha com mina ou é a última
linha)
        boolean[] podeEncerrarMaiorY = new boolean[eixoY.length];
        for(int i = 0; i<eixoY.length; i++) {
            if(i==eixoY.length-1) {
                podeEncerrarMaiorY[i] = true;
            }
            else if(eixoY[i+1].temMina()) {
                podeEncerrarMaiorY[i] = true;
            }
        }

        //Iterando possiveis linhas iniciais
        for(int y = 0; y<eixoY.length; y++) {

            if(podeIniciarMaiorY[y]){
                Linha[] copiaEixoX = copiarEixo(eixoX);
                Linha[] copiaEixoY = copiarEixo(eixoY);

                //
                Descomentar para ter o feedback linha a linha no terminal
            }
        }
    }
}

```

```

//
" - Y: " + maiorY);

System.out.println("Linha " + y + " - Maior: " + maiorArea + " - X: " + maiorX +

//Iterando possiveis colunas iniciais
for(int x = 0; x<eixoX.length; x++) {
    if(podeIniciarMaiorX[x]) {
        long maiorAreaDaCoord = 0;
        int areaLivreDireita = eixoX.length-x;
        int auxXF = 0;
        int auxYF = 0;

        //Varredura por linha;
        for(int y1 = y; y1<eixoY.length; y1++) {
            //Atualiza area livre a direita para coordenada
            if(copiaEixoY[y1].temMina()) {
                for(int pos : copiaEixoY[y1].posicaoMinas) {
                    if(pos>=x && pos-x<areaLivreDireita)

                        areaLivreDireita = pos-x;
                }
            }

            //Calcula area caso possa ser a maior
            if(podeEncerrarMaiorY[y1]) {
                if(((y1-y+1)*(areaLivreDireita)+1) >

maiorAreaDaCoord) {

                    maiorAreaDaCoord = ((y1-

y+1)*(areaLivreDireita));

                    auxYF = y1;
                    auxXF = x + areaLivreDireita -1;
                }
            }
        }

        if(maiorAreaDaCoord > maiorArea) {
            maiorArea = maiorAreaDaCoord;
            maiorX = x;
            maiorY = y;
            maiorXF = auxXF;
            maiorYF = auxYF;
        }
    }
}

System.out.println("--- Maior Área Livre de Minas ---"
    + "\nCoordenadas (X,Y inicial e final)"
    + "\nXi: " + (maiorX + 1) + " Yi: " + (maiorY + 1)
    + "\nXf: " + (maiorXF + 1) + " Yf: " + (maiorYF + 1)
    + "\nÁrea: " + maiorArea);

```

```

    }
}

//--- Classe Linha
import java.util.HashSet;

public class Linha {
    public boolean minada;
    public HashSet<Integer> posicaoMinas;

    public Linha() {
        posicaoMinas = new HashSet<Integer>();
    }

    public void addMina(int pos) {
        minada = true;
        posicaoMinas.add(pos);
    }

    public void removeMina(int pos) {
        for (Integer mina : posicaoMinas) {
            if(mina == pos) {
                posicaoMinas.remove(mina);
                if(posicaoMinas.isEmpty()){
                    minada = false;
                }
            }
        }
    }

    public boolean temMina() {
        return minada;
    }

    public Linha copiar() {
        Linha copia = new Linha();
        for (Integer mina : this.posicaoMinas) {
            copia.addMina(mina);
        }
        return copia;
    }

    public String toString() {
        String ret =" Minada: " + minada;
        for(Integer mina : this.posicaoMinas) {
            ret += " Mina: " + mina;
        }
        return ret;
    }
}

```