RELATÓRIO
-ASSALTO AO BANCO-

Lucas Guerbari Blacovicz
Engenharia de Software – PUCRS

Introdução

A fim de saber a quantidade deixada pelos ladroes em sua fuga, precisamos percorrer o caminho que eles fizeram e somar a quantidade de dinheiro que eles abandonaram. O problema que precisamos solucionar é como iremos percorrer o traçado e somar os valores encontrados até o ponto em que os bandidos foram capturados.

Modelagem do Problema

Para percorrer o mapa, precisamos achar o início do caminho à esquerda e seguir algumas regras:

- '-' – caminho é percorrido tanto para a esquerda quanto para a direita
- '|'– caminho é percorrido tanto para cima quanto para baixo
- '/' – mudança de direção para cima se o caminho estava sendo percorrido para a direita; mudança de direção para a esquerda se o caminho estava sendo percorrido para baixo. mudança de direção para baixo se o caminho estava sendo percorrido para a esquerda; mudança de direção para a direita se o caminho estava sendo percorrido para cima.
- '\' – mudança de direção para cima se o caminho estava sendo percorrido para a esquerda; mudança de direção para a direita se o caminho estava sendo percorrido para baixo. mudança de direção para baixo se o caminho estava sendo percorrido para a direita; mudança de direção para a esquerda se o caminho estava sendo percorrido para cima.
- '#' - onde os ladroes foram capturados; fim do percurso.

Solução

Quando a atividade foi proporcionada, tive dificuldade em pensar em como ela deveria ser resolvida por causa de minha inexperiência com o desafio. Resolvi dividir o problema em etapas e a primeira delas seria fazer o algoritmo ler os arquivos. Para isto, usei uma das minhas atividades de POO do semestre passado para me basear pois havia esquecido como ler arquivo em Java (apenas lembrava que era necessário ter tratamento *try/catch*). Simultaneamente, pensei em alguma maneira de fazer com que a lista fosse percorrida tanto para os lados quanto para cima ou para baixo. Felizmente, achei a solução em outro trabalho de POO, que seria carregar linha por linha em uma *List* ao mesmo tempo que o arquivo era lido. Assim, seria possível ir e voltar para cada linha e percorrer caractere por caractere usando *mapa.get(y).charAt(x)*, similar a um jogo RPG ou coordenadas no Minecraft.

```
private void carrega(String arq){
        try (BufferedReader reader = new BufferedReader(new FileReader(arq))) {
            String line;
            while((line=reader.readLine()) != null){
                mapa.add(line);
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Após o mapa ser carregado na *List*, o início do caminho a ser percorrido deve ser encontrado. Para isto, foi usado um *for* para caminhar "na borda" esquerda do arquivo de texto a fim de encontrar o caractere '-' (percebi que em todos os arquivos o caminho começava na esquerda portanto, seguindo caminho para a direita. Por isto, *dir* recebe 'r'.).

```
private void caminha(List<String> mapa){
        int x=0;
        int y=0;
        char dir= 'r'; //variável que guarda direção
        int sum= 0; //soma
        for(int i=0; i<mapa.size(); i++){
            if(mapa.get(i).charAt(0)=='-'){
                y=i;
                System.out.println(x+ " "+ y);
                break;
            }
```

Com as "coordenadas" do início caminho localizadas, já seria possível seguir o traçado. Sabendo que o programa seria encerrado assim que o caractere '#' fosse encontrado, usei um *while* até achar o caractere enquanto a lógica usada para percorrer e alterar as coordenadas x e y é repetida indefinitivamente. Para saber qual direção deveria ser percorrida, criei a variável *dir* que guarda a direção sendo elas:

- 'r'- right para direita.
- 'l'- left para esquerda.
- 'u'- up para cima.
- 'd'- down para baixo.

Cada caractere que foi citado anteriormente em "Modelagem do Problema" possui uma lógica composta de condições que, dependendo da variável que está armazenada em *dir* e da regra pré-estabelecida, pode mudar a direção.

'-' - para esquerda ou para direita:

```
if (currentChar == '-') {
                System.out.println("a");
                if (a == 'r'){
                    x++;
                }
                else if (a == 'l'){
                    x--;
                }
                else if(a == 'u'){
                    y--;
                }
                else if(a == 'd'){
                    y++;
                }
            }
```

‘|’ - para cima ou para baixo:

```
else if (currentChar == '|') {
                System.out.println("b");
                if (a == 'u'){
                    y--;
                } else if (a == 'd'){
                    y++;
                } else{
                    if(a == 'r'){
                        x++;
                    }else  if(a == 'l'){
                        x--;
                    }
                }
            }
```

‘/’ - mudança de direção:

```
else if (currentChar == '/') {
                System.out.println("c");
                if (a == 'r') {
                    y--;
                    a = 'u';
                } else if (a == 'l') {
                    y++;
                    a = 'd';
                } else if (a == 'u') {
                    x++;
                    a = 'r';
                } else if (a == 'd') {
                    x--;
                    a = 'l';
                }
            }
```

‘\’ - mudança de direção:

```
else if (currentChar == '\\') {
                System.out.println("d");
                if (a == 'r') {
                    y++;
                    a = 'd';
                } else if (a == 'l') {
                    y--;
                    a = 'u';
                } else if (a == 'u') {
                    x--;
                    a = 'l';
                } else if (a == 'd') {
                    x++;
                    a = 'r';
                }
            }
```

Após muitos testes e correções, o algoritmo era capaz de percorrer toda a lista até chegar ao ‘#’. Porém ainda faltava somar os números encontrados pelo caminho. Para isto, era necessário que fosse reconhecido todo o valor, não apenas o char que estava na posição atual. Ex: 53 não é 5+3, como seria se a lógica não houvesse sido implementada.

Se o número estivesse sendo lido da esquerda para direita:

```
if(a=='r'){
	while (x < mapa.get(y).length() && Character.isDigit(mapa.get(y).charAt(x))) {
		number = number * 10 + Character.getNumericValue(mapa.get(y).charAt(x));
		x++;
	}
}
```

Se o número estivesse sendo lido da direita para esquerda:

```
else if(a=='l'){
	while (x < mapa.get(y).length() && Character.isDigit(mapa.get(y).charAt(x))) {
		number = number +
Character.getNumericValue(mapa.get(y).charAt(x))*(Math.pow(10, times)); //inverso de
right
		x--;
		times++;
		}
}
```

Se a direção fosse para cima:

```
else if(a== 'u'){
	if((x+1 < mapa.get(y).length() || x-1 < 0) &&
((Character.isDigit(mapa.get(y).charAt(x-1))==true ||
Character.isDigit(mapa.get(y).charAt(x+1))==true) || ((mapa.get(y).charAt(x+1)=='-
')||(mapa.get(y).charAt(x-1)=='-')))){
		y--;
	}else{
		number = Character.getNumericValue(mapa.get(y).charAt(x));
		y--;
		}

}
```

Se a direção fosse para baixo:

```
else if(a== 'd'){
	if((x+1 < mapa.get(y).length() || x-1 < 0) &&
((Character.isDigit(mapa.get(y).charAt(x-1))==true ||
Character.isDigit(mapa.get(y).charAt(x+1))==true) || ((mapa.get(y).charAt(x+1)=='-
')||(mapa.get(y).charAt(x-1)=='-')))){
		y++;
	}else{
		number = Character.getNumericValue(mapa.get(y).charAt(x));
		y++;
	}
}
```

A fim de evitar que valores fossem somados duplamente em cruzamentos, se os caracteres aos lados forem dígito ou traçado, apenas será acrescentado ou descontado da variável y (desce ou sobe, respectivamente). Se o caminhamento for horizontal, números maiores que um dígito serão somados.

Por fim, a soma será armazenada na variável *dindin* e será printada juntamente com o tempo. Para se obter melhor entendimento do que foi percorrido, também é printado o caractere atual.

Casos de Teste

Caso: K

Tamanho: 50
--> Soma: 667
Tempo: 0.018

Tamanho: 100
--> Soma: 3895
Tempo: 0.022

Tamanho: 200
--> Soma: 33219
Tempo: 0.037

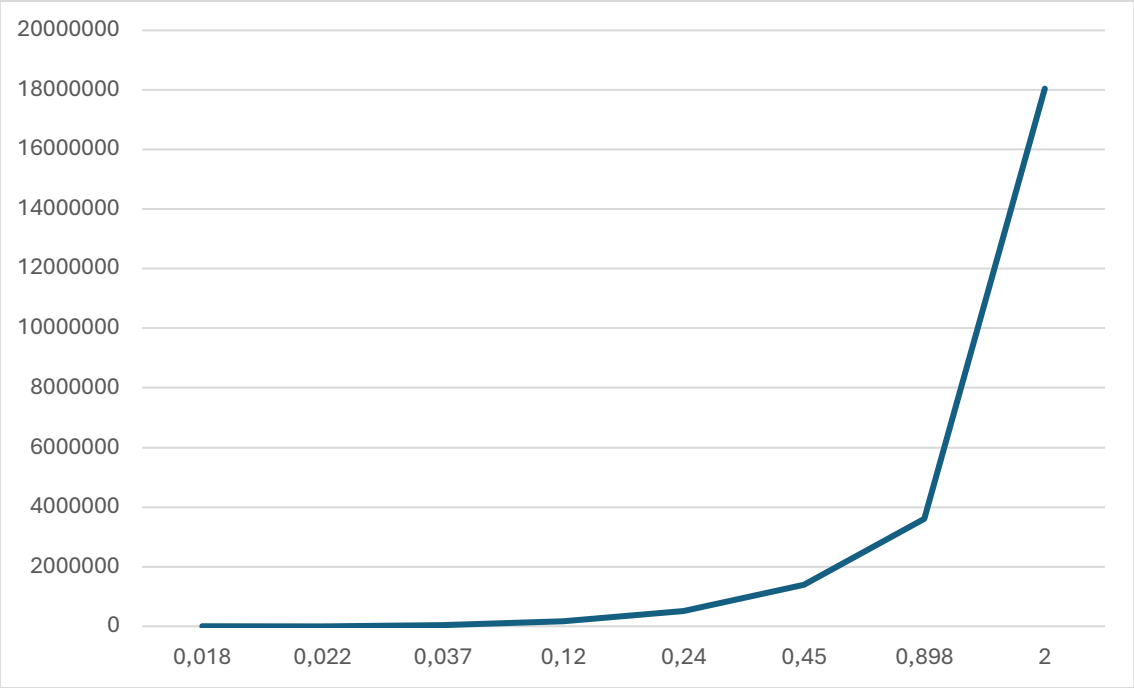
Tamanho: 500
--> Soma: 166901
Tempo: 0.12

Tamanho: 750
--> Soma: 508050
Tempo: 0.24

Tamanho: 1000
--> Soma: 1398755
Tempo: 0.45

Tamanho: 1500
--> Soma: 3600706
Tempo: 0.898

Tamanho: 2000
--> Soma: 18043854
Tempo: 1.575



Y= palavras; X= tempo

O(n^2)

Conclusões

Por fim, acredito que o problema e a forma como foi solucionado acrescentam conhecimento de estratégias e mecanismos que podem ser utilizados na solução de problemas como este. O algoritmo segue as regras impostas e leva em consideração o contexto em que está inserido (como não somar quando estiver subindo ou descendo em um cruzamento). É perceptível, porém, compreensível, que o algoritmo fica mais lento ao executar arquivos maiores, como visto no gráfico acima.

Referencias

- Trabalhos de POO – Professor Marcelo Veiga Neves

Obs.

- Na classe Detetive, está contido o algoritmo completo. A classe Policia serve para rodar o programa.