Relatório sobre trabalho de numero de conquista de castelos

Deivid Santos*
Faculdade de Informática — PUCRS

29 de maio de 2019

Resumo

A ideia principal do artigo é ajudar a explicar e entendermos a alternativa de solução escolhida para resolver o problema proposto no segundo trabalho da disciplina de Algoritmos e estruturas de dados II do segundo semestre, que trata do desenvolvimento de algoritmo que consiga descobrir a quantidade máxima de castelos que um exercito consegue conquistar, é apresentado a solução juntamente com os algoritmos, casos de teste e a eficiência dos mesmos.

Introdução

O exercicio proposto pela disciplina é o de resolver um problema que envolve conquista de castelos, onde o conde siberio, que o conquistador, deseja saber qual o numero limite de castelos que ele consegue conquistar com seu exército disponível. A entrada consiste em um documento de texto com informações sobre os castelos e os tamanhos dos exércitos, no caso a primeira linha mostra o tamanho inicial do exército siberio, seguido do numero total de castelos da vizinhança e o numero total de estradas na região, nas linhas seguintes é mostrado o tamanho do exército em cada um dos castelos e depois é mostrado todos os caminhos possíveis para o conde sibério chegar em todos os castelos. Neste caso o desafio principal é o de podermos demonstrar a partir dessa entrada, qual o maior numero de castelos que o conde Siberio pode conquistar com seu exército seguindo os caminhos disponíveis.

Solução proposta

O primeiro passo para obter a solução do problema proposto é a implementação de um grafo com os caminhos entre os castelos, nesse caso foi escolhida a linguagem de programação Java. Seguindo a entrada de texto, todos os castelos são representados como numeros, para representar isso foi criado um Grafo de Inteiros onde cada vértice representa um castelo, e as arestas são os caminhos entre os castelos e também para ser possível realizar os cálculos com os exércitos, foi criada a representação dos castelos, que utilizaremos para memorizar os exercitos restantes durante as simulações de batalha, isso foi divido em duas funções, uma responsável pela criação dos castelos e a outra pela criação do grafo como segue no algoritmo a seguir:

```
1 castle{
2       integer number , integer armySize
```

*deivid.santos@inf.pucrs.br

```
}

procedimento CRIARCASTELOS (List< String> linhasDoArquivo , integer tamanhoDoExercitoSiberio , int
    List< Castelo> castelos
    castles.add(Castelo(0 , tamanhoDoExercitoSiberio)) ;
    para 1 até numeroDeCastelosVizinhos + 1
        String[] linhas = linhasDoArquivo.get(i).split(" ") ;
        Castelo castelo = Castelo(linhas[IndiceDoNumeroDoCastelo]) ,linhas[IndiceDoExercitoDoCastelo
        castelos.add(castelo) ;
    fim
fim


procedimento CRIARGRAFO(List< String> linhasDoArquivo , int numeroDeCastelosVizinhos) {
    Grafo grafoDeCastelos = Grafo(numeroDeCastelosVizinhos + 1) ;
    List< String> arestas = linhasDoArquivo.subList(numeroDeCastelosVizinhos + 1 , linhasDoArquivo.siz
    para o até arestas.tamanho
        String[] arestaDeCastelos = arestas.get(i).split(" ") ;
        int castelo1 = arestaDeCastelos[IndiceDoNumeroDoCastelo] ;
        int castelo2 = arestaDeCastelos[IndiceDoExercitoDoCastelo] ;
        grafoDeCastelos.adicionarAresta(castelo2 , castelo2) ;
    fim
fim
```

Esse dois algoritmos possuem baixa complexidade, ambos são eficientes sendo apenas O(n) cada um.

A partir da execução desses dois algoritmos, temos um grafo com todos os castelos e estradas montados, e também temos todos os castelos prontos, faltando apenas descobrir os caminhos.

O proximo algoritmo seria o mais importante, mais complexo e também o mais demorado para executar, no caso ele é responsável pela busca dos melhores caminhos no grafo, ele faz a divisão dos exercitos para saber se o exercito pode seguir atacando por aquele caminho ou não, a busca de caminhos funciona da seguinte forma: Como o castelo Siberio foi definido como padrão com numero 0, é verificado todos os caminhos possíveis entre o castelo de numero 0 até todos os outros, como a seguir:

```
marcado[] , melhorCaminho , tamanhoDoCaminhoAtual

procedimento montarMelhorCaminho()
    para cada castelo
        BUSCARMAIORCAMINHO(castelo(0) , castelo.numero , grafo) ;
    fim

procedimento BUSCARMAIORCAMINHO(origem , destino , grafo)
    marcado[origem] = true ;
    se origem = destino
        se melhorCaminho < tamanhoDoCaminhoAtual
            melhorCaminho = tamanhoDoCaminhoAtual ;
        fim
        marcado[origem] = false ;
        parar
    fim

```

```
18      para cada int i em grafo.adj(origem)
19          se nao marcado[i] and temExercitoSuficienteParaAtacar(origem , castelos(i))
20              tamanhoOriginalExercito = calcularExercitoAposAtaque(source , i)
21              se todosExercitos[i] < tamanhoDoExercito or tamanhoDoCaminhoAtual > todosCaminhos[i]
22                  todosCaminhos[i] = tamanhoDoCaminhoAtual
23                  todosExercitos[i] = tamanhoOriginalExercito
24                  castelos(i).tamanhoOriginalExercito = calcularTamanhoDoExercitoAposAtaque(origem , i
25                  tamanhoDoCaminhoAtual++
26                  tamanhoOriginalExercito(i , destino , grafo)
27                  tamanhoDoCaminhoAtual−−
28                  castelos(i).tamanhoDoExercito = tamanhoOriginalExercito
29              fim
30          fim
31      fim
32      marcado[origem] = false ;
33  fim
34
35  procedimento calcularTamanhoDoExercitoAposAtaque(origem , destino)
36      castelos(origem).tamanhoDoExercito −(castelos(destino).tamanhoDoExercito ∗ 2) −50 ;
37  fim
38
39  procedimento temExercitoSuficienteParaAtacar(origem ,destino)
40      castelos(origem).tamanhoExercito ∗ 2 + 100 < castelos(destino).tamanhoExercito
41  fim
```

Nesse algoritmo foi utilizada recursão para mapear todos os caminhos possíveis entre todos os vértices, o algoritmo primeiro verifica se a origem e o destino são iguais, o que significa que ele chegou no final, e então pode verificar se o caminho final é maior que o que ele já possui, se for maior, substitui pelo atual, se não for maior, retorna na recursão e segue o loop de busca dos vertices adjacentes que faz com que o exercito vá reduzindo de acordo com os ataques e avanços do exercito, seguindo a formula de cálculo do exercito fornecida, caso o exercito chegue no final, a recursão volta e o exercito volta ao que era antes para ser possível fazer testes em outra direção, esse algoritmo possui uma complexidade alta e demora para ser executado, chegando a ser "O(n^3)", pois ele verifica todas as possibilidades de caminhos, em todos os castelos, para todas as arestas de cada um dos vertices até chegar no destino, porém ele sempre finaliza a recursão no momento em que chega em um castelo que já atacou por um caminho maior ou quando possui um exercito maior que aquele caminho. Todos os algoritmos possuem a complexidade valem para o pior e para o melhor caso, sendo todos Θ.

Esses algoritmos foram implementados utilizando alguns exemplos fornecidos em aula, seguindo o resultado:

	Tempo de execução (em milisegundos)		
Casos de teste	Inserção	Busca	Total (Incluindo leitura do arquivo)
Caso30	1	191	200
Caso32	2	223	250
Caso34	1	347	363
Caso36	1	2761	2780
Caso38	0	7420	7489
Caso40	1	6230	6243
Caso42	1	3727	3757
Caso44	1	22502	22612
Caso46	1	28634	28793

Conclusão

O algoritmo principal está funcionando de maneira um pouco lenta quando adicionamos muitas arestas nos grafos, pois ele faz a busca de todos os caminhos possíveis para todos os lados, então isso pode fazer com que demore um pouco na execução, mas no geral, funciona corretamente e sempre faz a busca conforme esperado.