

Siege of Leningrad

Relatório Técnico

Luiz Carlos B. Vieira, Sérgio Caetano Júnior

¹Instituto de Informática – Universidade Federal de Goiás (UFG)

luizcarlosbv@discente.ufg.br, sergio-caetano@discente.ufg.br

1. Descrição

Durante a Segunda Guerra Mundial, no ano de 1942, diversos snipers alemães tomaram pontos estratégicos da cidade de Leningrado, tornando o cerco de retomada da cidade uma missão quase impossível. O soldado soviético, durante o cerco a cidade, possuía um número limitado de munições para enfrentar esses diversos pontos estratégicos, sendo que ao errar um tiro ele seria morto por um dos snipers.

O exercício questiona qual a probabilidade de um soldado soviético chegar em um determinado ponto estratégico, partindo de outro ponto estratégico, eliminando todos os snipers em seu caminho (contando com o ponto de partida e o ponto almejado).

Para a solução, o algoritmo estipula como entrada os valores N , M , K e P , que representam, respectivamente, o número de pontos estratégicos, o número de ruas que interligam esses pontos, a quantidade de munição do soldado soviético e a probabilidade (variando entre 0 e 1) do soldado soviético eliminar um sniper alemão com um único disparo. Além disso, é inserido um número inteiro A que representa o total de soldados e um vetor informado suas distribuições. Os pontos estratégicos são numerados de $1 \dots N$, sendo colocados os pares de pontos, individualmente, em uma lista que representa todos os acessos a esses pontos.

2. Solução

O problema foi interpretado da seguinte forma: os pontos estratégicos serão considerados como vértices de um grafo, as ruas como arestas bidirecionais que interligam os vértices, e os pesos serão relativos a quantidades de tropas no vértice que direciona a aresta. Uma vez que o soldado tem que atingir todos os atiradores em seu caminho, além de ter apenas uma chance para cada, o caminho pelo qual ele tem maior chance de chegar no destino é aquele no qual ele encontrará o menor número de atiradores, de modo que o problema se reduz a achar o caminho de menor custo em um grafo.

Para isso, utilizaremos uma implementação do algoritmo de Dijkstra. O algoritmo representará um grafo a partir de uma matriz de adjacências, cujos pesos das arestas i - j correspondem aos valores das posições (i,j) nessa matriz. Toda a matriz será percorrida segundo a implementação original de Dijkstra, ou seja, a cada iteração, o vértice aberto de menor custo será selecionado, todas as suas adjacências serão relaxadas e esse vértice será fechado. O processo se repete até a completa resolução do menor caminho entre uma raiz e cada outro vértice do grafo. O retorno final é um vetor de menores custos para atingir cada vértice a partir da raiz, vetor construído ao longo de cada iteração. Após achar o caminho de menor custo C , ou seja, o caminho que passa pelo menor número de atiradores,

é simples achar a probabilidade do soldado sobreviver. Se $C > K$ a probabilidade é 0, já que não há munição suficiente para atingir todos. Caso contrário, note que os eventos de acertar um determinado atirador são independentes. Logo, a resposta é P^C , pois para cada atirador do caminho ele tem probabilidade P de acertar o primeiro tiro e não pode errar nenhum tiro.

3. Testes


	i1	i2	i3
i1	p1'	p1''	p1'''
i2	p2'	p2''	p2'''
i3	p3'	p3''	p3'''

Figura 1. Matriz de Adjacências e Pesos

A entrada do algoritmo solução recebe valores que são utilizados para construir uma representação do grafo. A primeira linha de cada instância contém três inteiros definidos anteriormente na Descrição e uma variável de ponto flutuante: N , M , K , e P . As M linhas seguintes recebem pares de inteiros i j que indicam as arestas orientadas entre i e j . Há uma linha contendo um inteiro A , correspondente ao número de atiradores na cidade, valor seguido por A inteiros que indicam a posição (vértice) de cada um desses atiradores. A última linha contém dois inteiros que indicam a raiz e o destino do percurso a ser estudado.

A matriz de adjacências acima é um modelo genérico da representação implementada pelo algoritmo solução. Para cada um dos seguintes conjuntos de entradas, obtemos uma representação matricial para os grafos apresentados subsequentemente.

Conjunto de entradas 1:

3 2 6 0.1 $\rightarrow N, M, K \text{ e } P$.

1 2

2 3

10 1 1 3 3 1 3 1 1 3 3

1 3

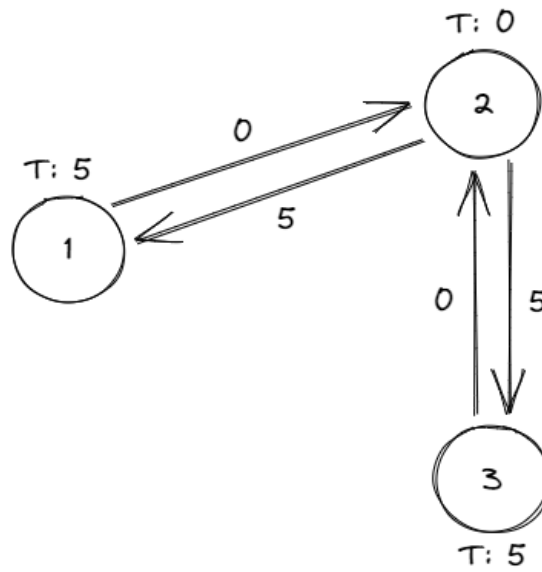


Figura 2. Grafo do conjunto de entradas 1

Conjunto de entradas 2:

5 5 10 0.3 -> N, M, K e P.

1 2

2 4

2 5

4 5

5 3

6 3 3 3 3 3 3

1 3

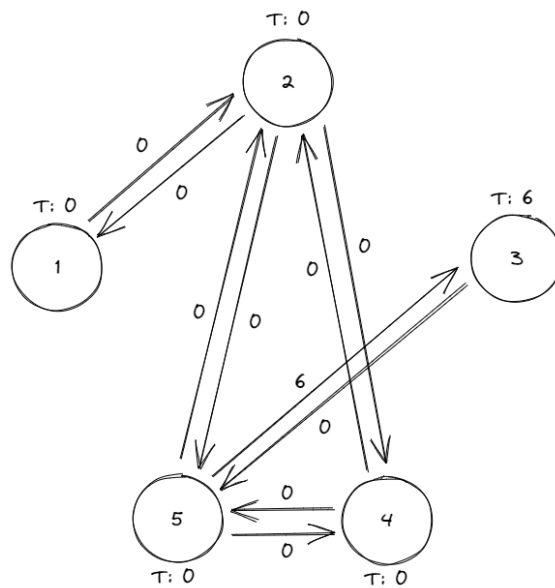


Figura 3. Grafo do conjunto de entradas 2

Ao instanciar os conjuntos de entradas 1 e 2, obtemos, como saída do algoritmo solução, o valor da probabilidade de um soldado partir da raiz e chegar ao destino apontados na instância. Para os conjuntos apresentados, essas probabilidades são:

$$P1 = 0.000$$

$$P2 = 0.001$$

4. Caso não trivial

Para o caso não trivial, iremos analisar uma situação hipotética que simula um mapa com vários pontos estratégicos. O conjunto de entradas do caso não trivial é:

```

20 23 20 0.5
1 2
1 7
2 3
3 4
3 6
4 5
6 5
7 8
8 9
8 10
8 16
9 10
10 11
11 12
12 13
12 15

```

13 14
 15 14
 16 17
 16 18
 18 19
 18 20
 19 20
 30 1 2 3 3 3 4 4 6 5 5 8 10 10 9 11 7
 13 14 15 16 16 17 18 19 20 18 8 7 9 11
 1 5

A seguir, há a representação do grafo gerado pelo conjunto de entradas. Foram definidos como pontos de partida e chegada, respectivamente, os vértices 1 e 5.

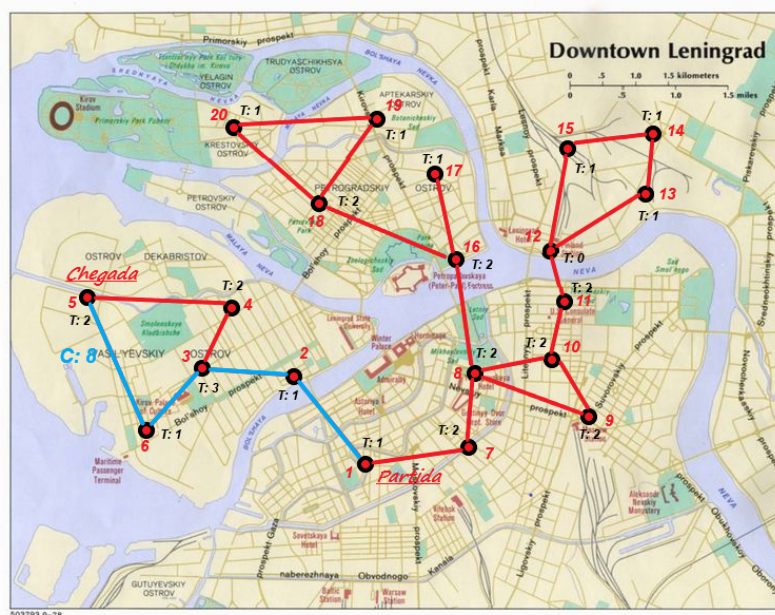


Figura 4. Matriz de Adjacências e Pesos

Concluindo, o algoritmo retornou o menor caminho $\langle 1, 2, 3, 6, 5 \rangle$ de custo 8. Como $8 > K$, então a probabilidade do evento pode ser definida como P^C , ou seja,

$$(1/2)^8 = 0.004$$

5. Considerações finais

Validamos o algoritmo solução para alguns casos que exemplificam bem a sua eficácia. Observa-se, pela comparação dos tempos de execução do algoritmo com outros algoritmos semelhantes, que a representação de um grafo como uma matriz de adjacências m com pesos $m[i][j]$ permite facilmente alcançar uma boa eficiência de tempo, além de simplificar o desenvolvimento de Dijkstra.

Segue o link da explicação em vídeo do problema do Cerco de Leningrado e o seu algoritmo solução:

<https://youtu.be/BcWpgd5E4t8>

6. Referencia bibliográfica

FEOFILOFF, Paulo. Algoritmo de Dijkstra. <https://ime.usp.br>, 2020. Disponível em: https://www.ime.usp.br/pf/algoritmos_para_grafos/aulas/dijkstra.html

FEOFILOFF, Paulo. Estruturas de dados para grafos. <https://ime.usp.br>, 2019. Disponível em: https://www.ime.usp.br/pf/algoritmos_para_grafos/aulas/graphdatastructs.html#GRAPHinit-matrix

FEOFILOFF, Paulo. Custos nos arcos e arestas. <https://ime.usp.br>, 2015. Disponível em: https://www.ime.usp.br/pf/algoritmos_para_grafos/aulas/weightedgraphs.html#sec:lists