Trabalho Prático 1 - Aritmofobia

Lucas Almeida Santos de Souza - 2021092563[1]

[1]Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

luscaxalmeidass@ufmg.br

1 Introdução

O trabalho consiste na resolução de um problema envolvendo grafos em que é preciso encontrar um caminho entre dois vértices de um grafo. O caminho deve 1-ter peso par, 2-passar apenas por arestas de peso par e 3-passar por um número par de arestas.

A entrada do programa desenvolvido consiste de um arquivo em que a primeira linha contém o número de vértices e o número de arestas do grafo separados por espaço e as linhas seguintes indicam as arestas do grafo, com o vértice de origem, o vértice de destino e o peso da aresta. A saída do programa é um arquivo contendo apenas um número que indica o peso do caminho encontrado, ou -1 caso não exista um caminho que satisfaça as condições.

2 Modelagem

Para a implementação do grafo, foi utilizada a representação de lista de adjacências utilizando um vetor de tamanho n representando os vértices do grafo, com um vetor em cada posição representando as adjacências de cada vértice. Esse vetor de adjacências armazena um par `(w, p)`, onde `w` é o vizinho do vértice e `p` é o peso da aresta que liga o vértice a esse vizinho. Podemos ver um exemplo dessa representação abaixo:

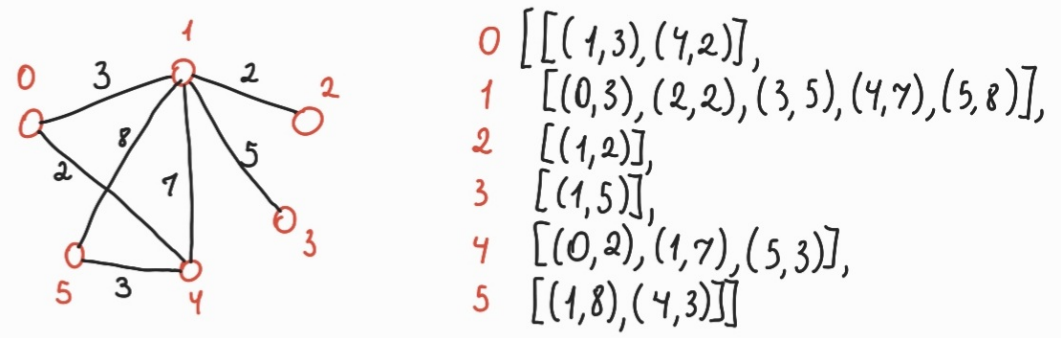

Figura 1: Representação da estrutura do grafo em lista de adjacências

Na Figura 1 os vizinhos de cada nó estão ordenados, mas isso não é necessário para o funcionamento

do programa. Um detalhe adicional a ser considerado é que o grafo recebido é iniciado em 1, porém, para facilitar a implementação, cada vértice é subtraído por 1 durante a leitura do arquivo, fazendo com que o grafo seja iniciado em 0.

Algoritmo

Para resolver o problema, foi utilizada uma variação do algoritmo de Dijkstra, além de uma restrição no momento de leitura do grafo. Ao ler uma aresta ímpar do arquivo de entrada, o programa main ignora e não a adiciona no grafo. Assim, a segunda condição já é satisfeita, pois o algoritmo nunca passará por arestas de peso ímpar.

Para evitar confusões, a expressão "caminho par"será utilizada para indicar um caminho com número par de arestas, pois como todas as arestas válidas são pares, todos os caminhos possíveis já terão peso par.

A variação do algoritmo de Dijkstra conta com uma estrutura chamada caminhos, que substitui o vetor original de distâncias. Ele consiste de um vetor de `pair` onde, para cada posição (vértice) o primeiro elemento do `pair` corresponde ao peso do caminho par mais curto e o segundo elemento corresponde ao peso do caminho ímpar mais curto. O vetor de visitados original do Dijkstra também sofreu uma alteração, agora contabilizando se o vértice foi visitado por um caminho par e/ou ímpar.

Antes de percorrer os vértices, o programa inicializa todos os caminhos como ∞ e todos os visitados como false, pois assim qualquer caminho até um vértice será menor que o armazenado. O algoritmo inicia no primeiro vértice, que já contém um caminho par até ele mesmo de peso 0 e 0 arestas, e adiciona em todos os seus vizinhos (que têm ∞ como menor distância ímpar) a distância ímpar até ele como menor distância par até eles. Isso é feito pois, como o caminho até o primeiro vértice é par, o caminho até seus vizinhos é ímpar.

Essa é a questão mais importante: para cada vértice, um caminho **par** até ele indica um caminho **ímpar** até seus vizinhos, passando por ele. Por isso, mesmo que estejamos buscando o menor caminho par até o destino, precisamos armazenar o menor caminho ímpar para cada vértice, pois ele pode fazer parte do menor caminho par até o destino. Por isso, o algoritmo armazena os dois caminhos para cada vértice.

Complexidade

Essa variação do aloritmo de Dijkstra percorre a lista de prioridade e, para cada vértice, percorre seus vizinhos em busca do menor caminho. Mesmo podendo percorrer os vértices mais de uma vez, esse valor não cresce proporcionalmente à quantidade de vértices e arestas, então podemos assumir a complexidade assintótica de $O(n + m)$, assim como o algoritmo de Dijkstra, adicionada às ordenações da lista de prioridade que ocorrem em $O(logn)$. Assim, temos que a complexidade geral do algoritmo é $O(m + nlogn)$

Pseudocódigo

O pseudocódigo do algoritmo é apresentado abaixo:

Jodds( ):

origem := 0, final := $n - 1$

caminhos := [(∞, ∞)] *// Vetor com os pares de caminhos (par, ímpar) inicializados em (∞, ∞)*
caminhos[origem].par := 0 *// O menor caminho par até a origem tem 0 arestas e peso 0*

visitados := [(false, false)] *// Vetor com os pares de visitados (par, ímpar) inicializados em (false, false)*
visitados[origem].par := true *// A origem é visitada no caminho par*

fila.push(caminhos[origem].menor, origem) *// Adiciona a origem na fila de prioridade*
enquanto fila não está vazia **faça**
    v := fila.top( )
    fila.pop( )

    se visitado[v] == (true, true) **faça**
        se v for final **faça**
            break *// Se o vértice é final e já foi visitado nos dois caminhos, encerra o loop*
        fim se
        continue *// Se o vértice já foi visitado nos dois caminhos, não analisa seus vizinhos*
    fim se

    para cada vizinho w **de** v **faça**
        se (caminhos[v].par + o peso da aresta vw) < caminhos[w].impar **faça**
            caminhos[w].impar := (caminhos[v].par + o peso da aresta vw)
            visitados[w].ímpar := true
            visitados[w].par := false
        fim se
        se (caminhos[v].impar + o peso da aresta vw) < caminhos[w].par **faça**
            caminhos[w].par := (caminhos[v].impar + o peso da aresta vw)
            visitados[w].par := true
            visitados[w].ímpar := false
        fim se
        se algum caminho até w foi atualizado **faça**
            fila.push(caminhos[w].menor, w)
        fim se
    fim para cada

fim enquanto
se caminhos[final].par != ∞ **faça**
    retorne caminhos[final].dp
senão
    retorne -1
fim se