Trabalho Final Teoria dos Grafos

Rangel Leônidas Silva

1. Determine os dois alunos que, provavelmente, possuem o melhor desempenho e defina o tipo de relação acadêmica existente entre eles na turma.

   De acordo com a análise do grafo invertido e a aplicação de busca em largura (BFS), os alunos mais alcançados foram o 19 e 10. Muitos alunos academicamente dependem deles. Tem relação direta pois existe uma aresta 10 → 19, e indireta porque o aluno 19 também influencia outros que chegam até ele via o aluno 10.

2. Explique, em suas próprias palavras, se neste caso podemos considerar tanto as arestas de entrada direta quanto os caminhos indiretos na definição de alcance.

   Sim, neste caso, podemos considerar tanto as arestas de entrada direta quanto os caminhos indiretos na definição de alcance.
   Ao avaliar a influência acadêmica de um aluno, é crucial considerar não apenas as conexões diretas, mas também as indiretas. Imagine, por exemplo, que o aluno 3 ajuda o aluno 2, e este, por sua vez, ajuda o aluno 1. Nesse cenário, o aluno 3 está influenciando indiretamente o aluno 1 por meio do aluno 2.
   No código, essa lógica foi implementada usando um **grafo invertido** e a busca em largura (**BFS**). As setas no grafo original indicam a direção da dependência, ou seja, de quem depende para quem é influente. Ao inverter essas setas, é possível rastrear a influência. A partir de um determinado aluno, o algoritmo percorre todas as conexões para encontrar todos os outros alunos que ele influencia, seja de forma direta ou indireta.
   Isso é o oposto do **fecho transitivo**, que, neste caso, seria usado para determinar de quem um aluno depende, mostrando sua "base de conhecimento", e não sua capacidade de influência.

3. O que é grafo invertido? Descreva como construí-lo a partir da lista de arestas.
   Um **grafo invertido** (ou grafo transposto) é uma representação de um grafo direcionado onde a orientação de todas as suas arestas é revertida. Isso significa que, se no grafo original existia uma aresta de um vértice A para um vértice B `(A→B)`, no grafo invertido essa aresta passará a ser de B para A `(B→A)`.
   A função `grafoInvertido` constrói essa estrutura usando um `Map<Integer, List<Integer>>`.

4. Quais mudanças conceituais ocorrem na interpretação dos vizinhos ao usar o grafo invertido?

   Ao utilizar um **grafo invertido**, a forma como interpretamos a relação de vizinhança muda completamente.
   No **grafo original**, a aresta `A → B` indica que o aluno A **depende** do aluno B. Portanto, os vizinhos de um vértice `A` são os alunos dos quais ele busca ajuda. Eles representam a **fonte de conhecimento** para `A`.
   Já no **grafo invertido**, as setas são revertidas, tornando-se `A ← B`. Isso significa que os vizinhos de um vértice `A` agora são os alunos `B` que dependiam dele no grafo original. A vizinhança de `A` passa a representar aqueles que **são influenciados** por ele.
   Essa inversão é crucial. A vizinhança de um aluno, que antes mostrava sua base de apoio, agora revela seu **alcance de influência direta**. Essa nova perspectiva é fundamental para

analisar quem exerce mais impacto na rede, permitindo o uso de algoritmos como o BFS para simular a propagação de conhecimento a partir de um único aluno.

---

5. Implemente o BFS que, dado um vértice v no grafo invertido, conte quantos nós são visitados.

   O método `bfsInvertido` funciona da seguinte forma:

   1. **Inicialização:** Ele começa com um vértice de partida (`inicio`) e cria uma fila (`fila`) para a busca e um conjunto (`visitados`) para rastrear os nós já processados.
   2. **BFS:** A busca se inicia a partir do nó `inicio`. Enquanto a fila não estiver vazia, o código faz o seguinte em cada passo:
      - **Pega o próximo nó:** Remove o nó da frente da fila (`atual`).
      - **Verifica vizinhos:** Para cada vizinho (`viz`) do nó atual no grafo invertido, ele verifica se o vizinho já foi visitado.
      - **Adiciona novos nós:** Se o vizinho ainda não foi visitado, ele é adicionado ao conjunto de visitados e à fila para ser processado mais tarde.
   3. **Contagem e Resultado:** A busca continua até que todos os nós alcançáveis a partir do `inicio` tenham sido visitados. No final, o método retorna o tamanho do conjunto `visitados` menos 1, pois o nó de início (`inicio`) é contado, mas não deve ser incluído no resultado final (a contagem é sobre os outros nós que o alcançam).

---

6. Defina o fecho transitivo de um grafo e apresente o algoritmo de Warshall (versão booleana).

   O fecho transitivo de um grafo é uma estrutura que revela a conectividade completa entre todos os seus vértices. Ele define todos os pares de vértices `(u,v)` para os quais existe um caminho, seja ele direto ou indireto, do vértice u para o vértice v. No contexto da rede de alunos, o fecho transitivo mostra todos os alunos dos quais um outro aluno depende, independentemente de a dependência ser direta (uma aresta) ou indireta (um caminho mais longo).
   Para calcular o fecho transitivo, o código usa a versão booleana do **Algoritmo de Warshall**. Esse algoritmo opera sobre uma matriz de adjacência, onde:

   - A matriz é inicialmente preenchida com `1` para cada aresta direta (`u, v`) e `0` para as que não existem.
   - O algoritmo itera sobre todos os vértices (`k`) e atualiza a matriz. Para cada par de vértices (`i, j`), ele verifica se existe um caminho de `i` para `j` passando por `k`.
   - A lógica é a seguinte: se existe um caminho de `i` para `k` E um caminho de `k` para `j`, então deve haver um caminho de `i` para `j`.

---

7. Suponha que, após executar as buscas, os alunos A e B tenham contagens máximas de “quem os alcança”. Como você justifica que esses sejam os melhores desempenhos acadêmicos?

   No grafo original, esses alunos são fontes de conhecimento, ou seja, outros dependem deles. Isso sugere que A e B possuem alto desempenho, já que são frequentemente procurados como referência acadêmica.

---

8. Se A e B estiverem numa mesma componente fortemente conectada, o que isso diz sobre a relação de dependência entre eles?

Isso significa que existe um caminho de A para B e de B para A. Em termos práticos, eles não apenas ajudam outros, mas também aprendem entre si. Essa situação reflete uma relação de dependência mútua, onde o aprendizado é colaborativo e recíproco.

---

9. Além de contar predecessores, que outras métricas de centralidade poderiam indicar “influência” num grafo de dependências?

   Além de contar predecessores, que mede a popularidade, outras métricas de centralidade podem indicar diferentes tipos de influência em um grafo de dependências:

   - **Centralidade de Intermediação:** Identifica alunos que atuam como pontes entre diferentes grupos. Eles são cruciais para o fluxo de informação, pois muitos caminhos passam por eles.
   - **Centralidade de Proximidade:** Mede a rapidez com que um aluno pode espalhar informações por toda a rede. Alunos com alta proximidade são eficientes na comunicação.
   - **Centralidade de Vetor Próprio:** Avalia a influência com base na importância dos alunos conectados. Um aluno é considerado influente se estiver ligado a outros alunos influentes, como um "líder de opinião".

---

10. Em qual situação você preferiria usar fecho transitivo em vez de inverter + BFS? E vice-versa? (pensando em grafos esparsos vs. densos, em limites de memória e em facilidade de implementação.)

    Usar fecho transitivo (Warshall): O fecho transitivo é preferível quando o grafo é pequeno e denso, pois o algoritmo tem custo O(n3). Também é ideal quando se precisa responder a muitas perguntas sobre alcance entre todos os pares de vértices rapidamente.
    Usar grafo invertido + BFS: Esta abordagem é ideal para grafos esparsos e grandes. É mais eficiente quando se quer apenas responder sobre um vértice específico. No código, foi usado o algoritmo de Warshall para ver todos os alcances e o BFS quando o foco é saber quem alcança um único aluno.

---

11. Qual é a limitação de usar somente o grau de entrada direta (Grau de Entrada) para medir ‘influência’?

    O grau de entrada conta apenas relações diretas, ignorando toda a influência indireta que um aluno pode ter. Por exemplo, um aluno pode ser altamente influente por meio de intermediação (ajuda um que ajuda vários), mas isso não aparece no grau de entrada simples.

---

12. Como a existência de ciclos no grafo pode distorcer essa medida?

    Ciclos geram uma contagem inflada, pois os mesmos nós podem ser revisitados em diferentes caminhos. Se não tratados corretamente (por exemplo, não marcando o nó como visitado no BFS), isso pode causar overcounting ou loops infinitos.

---

13. Modifique o grafo para adicionar pesos às arestas (nível de dependência). Como você ajustaria o cálculo de alcance indireto?

    Enquanto uma busca simples (BFS) apenas conta o número de conexões, o Dijkstra leva em conta os pesos para encontrar o **caminho de menor custo** entre um aluno e todos os outros. O grafo ponderado (onde o peso representa o "nível de dependência"), isso permite medir o

alcance de um aluno de uma forma mais refinada, encontrando o caminho de "menor esforço" ou "maior probabilidade".

Como o Código Funciona

1. **Inicialização:**
   - Um mapa (`distancias`) é criado para armazenar o menor custo de cada nó a partir da origem. Todas as distâncias são inicializadas com infinito, exceto a do nó de origem, que é `0.0`.
   - Uma fila de prioridade (`heap`) é usada para armazenar os nós a serem visitados. A prioridade de um nó é baseada em sua distância atual, garantindo que o algoritmo sempre explore o nó mais próximo primeiro.
2. **Loop Principal:**
   - O algoritmo executa um loop enquanto houver nós na fila de prioridade.
   - Em cada iteração, ele remove o nó com a menor distância atual (`u`) da fila.
   - Uma verificação é feita (`if (custoAtual > distancias.get(u))`) para pular nós que já foram processados com um caminho mais curto, o que otimiza o algoritmo.
3. **Atualização de Distâncias:**
   - Para cada vizinho (`vizinho`) do nó atual (`u`), o algoritmo calcula um `novoCusto` somando a distância atual (`custoAtual`) ao peso da aresta (`aresta.peso`).
   - Se esse `novoCusto` for menor do que a distância registrada anteriormente para o `vizinho`, o mapa de distâncias é atualizado com o `novoCusto`.
   - O `vizinho` é então adicionado à fila de prioridade para ser processado futuramente.
4. **Retorno:**
   - Após o loop terminar, o mapa `distancias` contém o caminho de menor custo da origem para todos os nós alcançáveis. Se um nó for inalcançável, sua distância permanecerá como `Double.POSITIVE_INFINITY`.