Teoria de Grafos

Kaway Henrique da Rocha Marinho

11 Setembro 2024

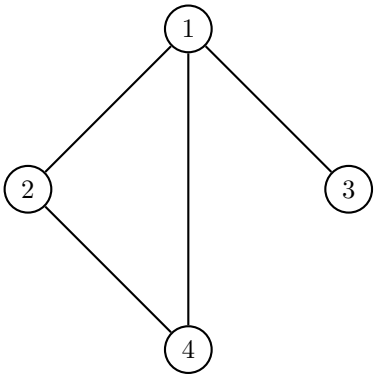
1 Questão 1

O grau de um vértice é o número de arestas ligado àquele grafo, isso permite verificar que m em função de d_v será dado por algo no formato de um somatório dos graus dos vértices do grafo em questão. Porém como o grafo é bidirecionado (está pedindo arestas em vez de arcos), sabemos que toda aresta seria contabilizada duas vezes. Chegamos então a seguinte fórmula:

$$|E| = \frac{1}{2} \sum_{v=1}^{|v|} d_v$$

Onde $|E|$ é o número de arestas do grafo, $|v|$ é o número de vértices daquele grafo e d_v é a função grau aplicada no vértice v.

Podemos testar a função no seguinte grafo:



Esse grafo pode ser representado como:
$V = \{1, 2, 3, 4\}$
$|E| = \{(1, 2), (1, 3), (1, 4), (2, 4)\}$
E nosso conjunto de graus será: $D_v = \{3, 2, 1, 2\}$. Logo seguindo a fórmula encontrada temos:

$$|E| = \frac{(3 + 2 + 1 + 2)}{2} = 4$$

O que condiz com o resultado encontrado por inspeção visual. É possível perceber também que essa fórmula nos permite visualizar o porquê de o número de arestas de um grafo regular (ou seja, todos os vértices possuem o mesmo grau) ser igual a $\frac{nr}{2}$. Caso o grafo fosse não bidirecionado, bastaria somar toodos os graus, ou multiplicar essa função por dois, que teríamos a função do número de arestas pelo grau.

2 Questão 2

2.1 Memória de uma matriz de adjacência

A matriz de adjacência de um grafo será uma matriz de tamanho $n * n$, onde cada elemento daquela matriz representa uma aresta entre os vértices linha e coluna. Portanto a função que representa a memória em bytes será:

$$M_{bytes} = n^2 \text{ bytes}$$

De maneira análoga:

$$M_{bits} = n^2 \text{ bits} = \frac{1}{8} M_{bytes}$$

Esse é um custo de memória relativamente alto, um grafo de 10 milhões de vértices(o que é factível principalmente para redes sociais ou grafos grandes) por exemplo consumiria um total de 100 terabytes. Existem no entanto algumas formas de mitigar isso, por exemplo para um grafo bidirecionado a matriz é simétrica, portanto bastaria guardar metades dos bytes que seriam normalmente necessários.

2.2 Memória de uma lista de adjacência

Uma lista de adjacência irá consumir em memória, para cada aresta, o equivalente a 12 bytes seguindo os parâmetros dados, portanto a função de custo de memória terá 12 como parte de sua composição. Também é possível perceber que cada aresta estará representada duas vezes, uma para cada vértice. Portanto o custo de memória será:

$$M_{bytes} = 2 * 12 * m_{arestas} \, bytes$$

Esse é um custo linear e consideravalmente menor que o custo para matriz de adjacência. Utilizando o mesmo grafo anterior e considerando-o regular e de grau 2, por exemplo, teríamos um consumo de memória de 240 megabytes, consideravelmente menor.

2.3 Relação entre memória consumida por uma matriz e uma lista de adjacência

A partir dos custos derivados das questões anteriores e colocando m em função da densidade do grafo, podemos chegar à seguinte inequação:

$$\frac{2 * 12 * d * n * (n-1)}{2} < n^2$$

Onde o termo da esquerda se refere ao custo de memória da lista e o da direita ao custo de memória da matriz. Partindo disso, chegamos à seguinte inequação:

$$d < \frac{n}{12 * (n-1)}$$

Para grafos muito grandes ($n > 10^6$) podemos desconsiderar o 1 no denominador, chegando a:

$$d < \frac{1}{12}$$

Ou seja, a densidade do grafo deve obedecer à primeira inequação para que a lista tenha efiência de memória melhor que a matriz. Adicionalmente, para grafos grandes, a densidade deve ser de cerca 8,3% para que seja eficiente.

2.4 Eficiência de memória do ator Kevin Bacon

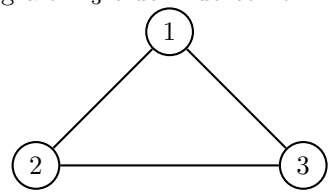
$$M_{matriz} = 62.500.000.000\, bytes \approx 59.605\, megabytes$$

$$M_{lista} = 182.400.000\, bytes \approx 174\, megabytes$$

Portanto a estrutura de lista é mais eficiente em termos de memória para esse caso. Poderíamos também verificar isso pela densidade do grafo (0.002) que é menor que a condição encontrada em 2.3

3 Questão 3

O grafo K_3 é definido como:



E suas árvores geradoras são:

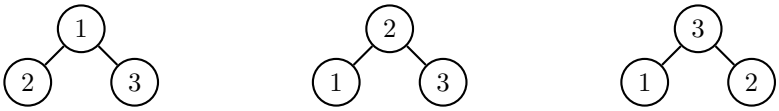


Figura 1: Árvores Geradoras induzidas por BFS

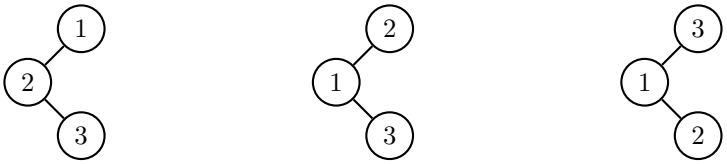


Figura 2: Árvores Geradoras induzidas por DFS, ordenando por vértice de menor valor

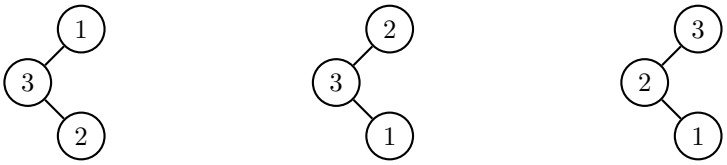


Figura 3: Árvores Geradoras induzidas por DFS, ordenando por vértice de maior valor

Essas árvores foram geradas levando em conta o conceito de raíz. Isso quer dizer que são tratadas como distintas para fins de demonstração mas são exatamente o mesmo grafo muitas das vezes. Na verdade, analisando as árvores por inspeção visual é possível perceber que só existem 3 árvores geradoras distintas para K_3, os grafos mais à esquerda de cada uma das figuras acima.

Para K_4 optei por não desenhar os grafos aqui, para manter a brevidade mas encontrei um total de 16 árvores distintas seguindo o critério acima.

Tentando deduzir, imaginei que o número de árvores distintas teria um formato polinomial, ou seja n^x já que K_3 gera 3 (ou 3^1) e K_4 gera 16 (ou 4^2). Não

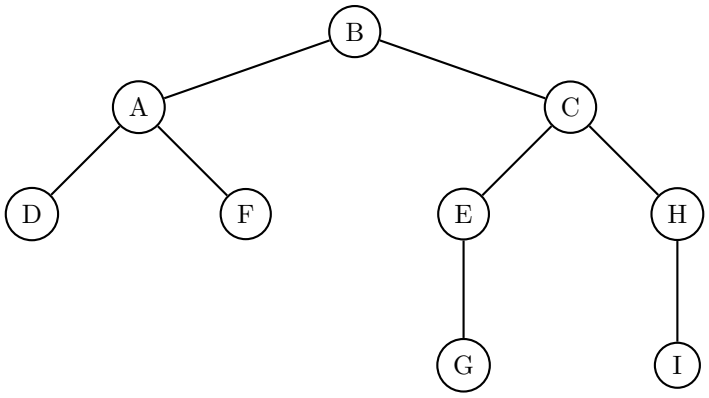
consegui prosseguir além disso pois K_5 demandaria muito tempo para fazer o cálculo de todas as árvores distintas. Pesquisando sobre o problema, encontrei a respeito da fórmula de Cayley que define que todo grafo completo possui n^{n-2} árvores geradoras distintas, resultado condizente com meu raciocínio.

4 Questão 4

4.1 Ordem exploração BFS

A ordem de exploração utilizando o algoritmo de busca em largura é:
B A C D F E H G I

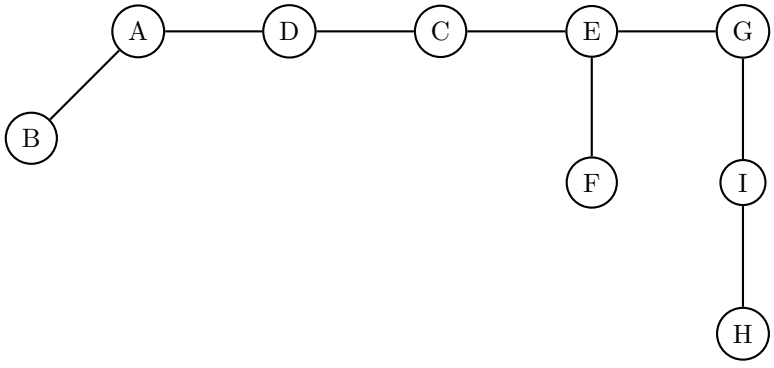
4.2 Árvore Geradora da BFS



4.3 Ordem exploração DFS

A ordem de exploração utilizando o algoritmo de busca em profundidade é:
B A D C E F G I H

4.4 Árvore Geradora da DFS



5 Questão 5

5.1 Exploração de vértices

Ambos os algoritmos seguem as etapas de um algoritmo genérico que consiste em classsificar os vértices entre desconhecidos, descobertos e explorados. Quando um vértice desconhecido torna-se descoberto, todos os vértices não explorados adjacentes a ele são descobertos em uma etapa seguinte, isso garante que o algoritmo nunca voltará a um vértice já explorado, portanto explorando cada um apenas uma vez.

5.2 Examinação de arestas

Seguindo ainda a ideia do algoritmo genéricos, toda vez que o algoritmo encontrar um vértice ele irá consultar todas as arestas incidentes a ele, para verificar se aquele vértice da aresta é exploradoo ou não, portanto acontecerá duas examinações por aresta.

5.3 Número de elementos fila BFS

Seguindo a lógica do BFS, de adicionar todos os vértices adjacentes ao vértice de origem, é possível notar que o maior tamanho de fila de uma busca desse tipo é de N, sendo N o número de vértices presentes no grafo em questão. Isso ocorre pois primeiro o vértice tido como início é adicionado a fila e então todos os vértices adjacentes são também adicionados, isso ocorre em grafos onde o vértice inicial seja conectado a todos os outros vértices, como é o caso de um grafo completo.

5.4 Número de elementos pilha DFS

A busca em profundidade percorrerá os vértices de maneira recursiva, indo cada vez mais fundo nos grafos e gerando uma árvore mais profunda que a BFS. Esse comportamento mostra que em um grafo onde cada vértice esteja conectado a apenas um outro vértice a pilha de uma DFS será de N, onde N é o número de vértices daquele grafo, pois a pilha só passará a ser esvaziada ao encontrar o vértice final.

6 Questão 6

```
main()
    Definir s como vértice qualquer do grafo
    Definir ciclo como uma fila vazia
    Definir existe_ciclo como falso
    Definir pai como vetor vazio
    Desmarcar todos os vértices
    Definir fila Q vazia
    Marcar s e inserir s em

    Enquanto Q não estiver vazio e existe_ciclo não \\
    for verdadeiro
        Retirar vértice v de Q
        Para todo vizinho w de v faça
            Se w não estiver marcado
                Colocar pai[w] = v
                Marcar w
                Inserir w em Q
            Se w estiver marcado
                Adicionar (w,v) a ciclo
                Concatenar Encontrar_caminho(w, pai) \\
                à ciclo
                Colocar existe_ciclo como verdadeiro
                Quebrar laço de repetição

    Se existe_ciclo for verdadeiro
        Imprimir("Grafo possui ciclo")
        Imprimir(ciclo)

    Se houver vértice não marcado
        Imprimir("Grafo desconexo")

    Se existe_ciclo for falso e não houver vértice \\
    não marcado
        Imprimir("Grafo é uma árvore")

Encontrar_caminho(vértice, pai)
    Definir ciclo como uma fila vazia
    Enquanto pai[vértice] for definido
        Adicionar (vértice, pai[vértice]) a ciclo
        Colocar vértice como pai[vértice]
    Retornar ciclo
```

7 Questão 7

```
main(s)
    Desmarcar todos os vértices
    Definir fila Q vazia
    Definir nivel_atual = 0
    Definir nivel como vetor vazio
    Definir pai como vetor vazio
    Marcar s e inserir s na fila Q
    Enquanto Q não estiver vazia
        Se pai[v] for definido
            nivel[v] = nivel[pai[v]] + 1
        Se não
            nivel[v] = 0
        Retirar v de Q
        Para todo vizinho w de v faça
            Colocar pai[w] = v
            Se w não estiver marcado
                Marcar w
                Inserir w em Q
```