Resumo Grafos (prova 1)

Sophia Carrazza

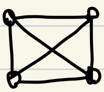
Resumo de Grafos:

Grafo: Coleção de vértices e arestas

grafo regular: todos os vértices têm o mesmo grau.
↳ se o grafo tiver componentes, a qntd de vertices deve ser divisível pela qntd de componentes (?).

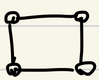
grafo conexo: existe um caminho entre qualquer par de vértices. ↳ nº mínimo de arestas = $n - 1$

grafo completo: grafo simples em que todo vértice é adjacente a todos os outros vértices.



$|V| = n$ $\quad |E| = \frac{n(n-1)}{2}$

grafo ciclo: consiste de um único ciclo (nenhuma aresta repetida)

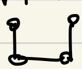


$|E| = n$

pode repetir vertices e arestas

grafo passeio: (walk) sequência de vértices e arestas, em que cada aresta conecta dois vértices consecutivos em sequência

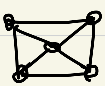
$|E| = n-1$
$|V| = n$



grafo caminho: (path) é um passeio sem arcos repetidos (os arcos são todos diferentes entre si. → nenhum vertice ou aresta é repetido

grafo trail: nenhuma aresta é repetida

grafo roda: formado por um único vertice ligado a todos os vertices de um grafo ciclo.



$|V| = n + 1$ $\quad |E| = n + n$

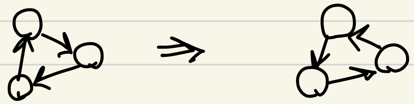
grafo bipartido: grafo cujos vértices podem ser divididos em dois conjuntos disjuntos U e V e cada aresta conecta um vértice de U a um vértice de V.

↳ A quantidade de vertices deve ser dividida igualmente.
↳ A formula p/ o máx de arestas num bipartido:

$$\boxed{\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}}$$

grafo complementar: é o grafo completo menos as ligações que contém no grafo G.

grafo transposto: ordem das direções invertidos



★ BFS → busca em largura

↳ inicializa todas as distâncias c/ −1
começa em um vértice v.
a distância de v para si mesmo é 0
fila += v
enquanto a fila não estiver vazia:
    remover início da fila
    p/ cada vizinho de v:
        se o vizinho ñ foi visitado,
        adiciona o vizinho ao fim da fila
        adiciona à distância atual.
retorna as distâncias

★ DFS → busca em profundidade → p/ achar o fecho transitivo direto

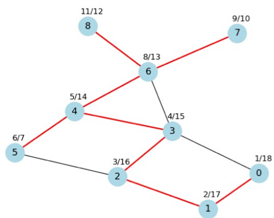
↳ dfs(v, grafo, visitados){
lista de visitados.
visitados.add(v) ← vértice
para cada vizinho de v no grafo:
se vizinho não_visitado:
dfs(v, grafo, visitados)
}

fecho_transitivo(v, grafo){
visitado ⇒ seta o visitado
dfs(v, grafo, visitados)
return visitados.
}

Busca em Profundidade

→ usa pilha

Considere G = (V, E) um grafo **conexo**. A ideia central do algoritmo é explorar o máximo possível ao longo de cada ramificação antes de retroceder. A busca começa em um vértice *s* pertencente à V e se aprofunda no grafo até que todos os vértices de V tenham sido atingidos. Logo, considere S um conjunto que se inicia com *s* e enquanto S for diferente de V:



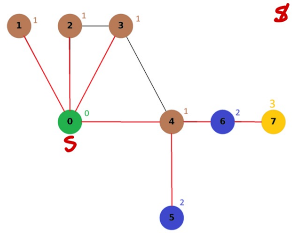
1 - A partir de *s* siga para um vértice vizinho *u* que não esteja em S, adicione *u* em S e repita o processo até alcançar um vértice sem vizinhos não visitados.
2 - Neste ponto, você deve retroceder (backtracking) até o último vértice que ainda possui vizinhos não contidos em S e continuar realizando os passos descritos no tópico anterior.

Busca em Largura

→ usa fila

fila: 0
~~0 1 2 3 4~~
~~5 6 7~~

Considere G = (V, E) um grafo **conexo**. A ideia central do algoritmo de busca em largura é explorar todos os vértices de um mesmo nível de proximidade antes de passar para os vértices do próximo nível. A busca começa em um vértice *s* pertencente à V e segue para todos os vértices adjacentes antes de mover-se para os vértices do próximo nível. Dessa forma, considere S um conjunto e Q fila que inicialmente contêm apenas *s*. Enquanto S for diferente de V:



1 - Desenfileire o próximo vértice da fila. Considere esse vértice como *v*.
2 - Para cada vizinho *u* de *v*, caso *u* não esteja contido em S, adicione-o em S e enfileire-o em Q.

$\Gamma^{+}(a) = \{a, b \dots\}$ /

* P/ achar o fecho transitivo inverso → encontra o grafo transposto e faz a busca em profundidade.

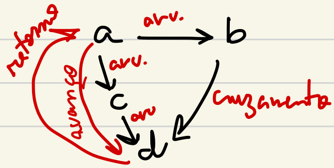
Excentricidade: Maior distância dentre as menores distâncias entre o vértice v e os outros vértices.

Raio: Menor das excentricidades do grafo

Diâmetro: Maior das excentricidades do grafo

Centro: Conjunto de vértices com a menor excentricidade

Arestas de:
- árvore: vai p/ um vértice que ainda não foi visitado
- retorno: acessando um elemento ancestral de outro elemento que é descendente dele
- avanço: indo p/ um vértice descendente pulando outros que estão no meio
- cruzamento: acessa um descendente que já foi acessado por outro ancestral.



Fórmulas:

* nº de subgrafos de um grafo completo $= \boxed{\sum_{i=1}^{n} 2^{\frac{i(i-1)}{2}} \cdot \binom{n}{i}}$ → $\frac{n!}{i!(n-i)!}$

★ nº máx de arestas de um grafo regular $= \boxed{\dfrac{n(n-1)}{2}}$

★ nº máx de arestas p/ um grafo com K componentes $= \boxed{\dfrac{(n-K)(n-K-1)}{2}}$

★ nº mín de arestas (p/ K componentes) $= \boxed{n - K}$

★ $$\sum_{v \in V} d(v) = \boxed{2 \cdot |E|}$$
(soma dos graus)

★ Porque o nº de vértices de grau ímpar deve ser par:

O número de vértices de grau ímpar sempre deve ser par pois a soma dos graus de um grafo é $2 \cdot |E|$, ou seja, a soma de graus ímpares também deve ser par para ter este resultado:

Logo, x é par se $x = 2 \cdot K$
x é ímpar se $x = 2 \cdot K + 1$ $\Big\}$ $\sum (2 \cdot K_v + 1)$

★ só é possível que a soma dos graus ímpares seja par se houver um número par de vértices c/ grau ímpar.

$$\underbrace{2 \cdot |E|}_{\text{PAR}} = \underbrace{\overset{\text{par}}{\sum d(v)}}_{\text{PAR}} + \underbrace{\overset{\text{ímpar}}{\sum d(v)}}_{\text{PAR}} \text{ ou } \underbrace{\sum 2K_v}_{\text{PAR}} + \underbrace{\sum 1}_{\text{PAR}}$$

SOMA dos vértices de grau par + vértices de grau ímpar

logo, a soma dos graus ímpares tem que ser par.

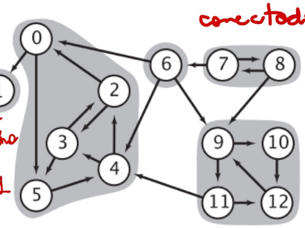
★ Porque um grafo G deve conter pelo menos dois vértices de mesmo grau?

Em um grafo simples, o grau de um vértice pode variar de 1 a $n-1$ ou de 0 a $n-2$, mas 0 e $n-1$ nunca podem coexistir. Assim, se existem mais vértices do que graus possíveis, pelo menos um deverá repetir seu grau / ter o mesmo grau que outro.

→ pois se há um vértice que está conectado a todos os outros ($n-1$), não pode existir um isolado (0)

SCC - Kosaraju

→ algoritmo para encontrar os componentes fortemente conectados (SCC)



1 - Realizar uma **busca em profundidade** (DFS) em G = (V, E), granvando os tempos de início e fim da visitação

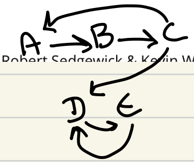
↳ ao final disso, temos os vértices empilhados em ordem decrescente de tempo de término

2 - Na ordem inversa dos tempos de término (**ordem topológica**), realizar uma busca em profundidade no **grafo transposto**, T de G

→ desempilhando os vértices da pilha criada no passo 1

3 - Para cada conjunto de vértices visitados em T, marque esse conjunto como um **componente fortemente conexo (scc)** em G.

Fonte: Robert Sedgewick & Kevin Wayne, Algorithms, ed. 4; p. 584; 2011



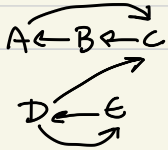
1- busca em prof. começando por A → A, B, C, D, E

$\lfloor A \rfloor$ $\lfloor B \rfloor$ $\lfloor C \rfloor$ $\lfloor D \rfloor$ $\lfloor E \rfloor$ (termina em D)

volta p/ A

↓ transposto

2- busca em prof. na ordem decrescente da pilha no transposto → $\boxed{\{D, E\} \text{ e } \{A, B, C\}}$



$\lfloor D \rfloor$ $\lfloor E \rfloor$ até voltar p/ D

$\lfloor A \rfloor$ $\lfloor C \rfloor$ $\lfloor B \rfloor$ (voltou p/ A)