

# GRAFOS

## 06/10

1. Seja  $G = (V, E)$  um grafo não-direcionado e conexo. **Seja  $(G, w)$  um grafo ponderado em que  $w: E \rightarrow R^+$ .** Projete uma solução para encontrar um sub-grafo conexo de forma que a soma dos pesos das arestas seja o maior possível.

R: Escolhemos qualquer vértice, pois qualquer um deles será um sub-grafo com o menor peso possível na soma de suas arestas.

2. Seja  $G = (V, E)$  um grafo não-direcionado e conexo. **Seja  $(G, w)$  um grafo ponderado em que  $w: E \rightarrow R^+$ .** Projete uma solução para encontrar um sub-grafo  $T = (V, E') \leq G$  conexo de forma que a soma dos pesos das arestas seja o maior possível.

R: PRIM DJKISTRA | KRUSKAL | REMOÇÃO REVERSA

## 08/10

1. Seja  $G = (V, E)$  um grafo não-direcionado e conexo, e  $(G, w)$  um grafo ponderado em que  $w: E \rightarrow R$ . Projete uma solução para encontrar uma árvore geradora mínima a partir de  $(G, w)$ .

R:

2. Seja  $G = (V, E)$  um grafo não-direcionado e conexo, e  $(G, w)$  um grafo ponderado em que  $w: E \rightarrow R$ . Projete uma solução para encontrar uma árvore geradora máxima a partir de  $(G, w)$ .

R:

3. É possível DIJKSTRA e KRUSKAL gerarem a "mesma árvore"?

Union-Find

## 15/10

1. Seja  $G = (V, E)$  um grafo não-direcionado e simples. Seja  $(G, w)$  um grafo ponderado em que  $w: E \rightarrow R^+$ 
  - a. Como calcular uma árvore geradora mínima em  $(G, w)$ ?

R: PRIM, KRUSKAL, REMOÇÃO REVERSA.

- b. Como aplicar Dijkstra em  $(G, w)$  a partir de um vértice dado?

R: Pode funcionar, mas nem sempre.

- c. Projete um algoritmo entre uma origem e um destino que tenha números positivos e negativos.

R:

Quando meu grafo é uma árvore, o Djikstra encontra a mesma árvore geradora.

Árvore  $T = (V, E)$

$u, v$  pertencem a  $V$

- Quantos caminhos simples existem entre  $u$  e  $v$ ?

R: Só existe um caminho, pois em árvores só possuem um caminho.

Conceitos:

- Árvore geradora de um grafo conexo → É um **SUBGRAFO** conexo desse grafo com todos os vértices do primeiro.
- Árvore geradora **MÍNIMA** de um grafo conexo → É um **SUBGRAFO** conexo desse grafo com todos os vértices do primeiro, de forma que tenha a menor quantidade de arestas.
  - Se for um grafo ponderado, tratamos o **SUBGRAFO** conexo observando a menor soma de pesos possível.

Como garantimos que elementos terminais se mantenham conexos.

Árvore de Steiner é um subgrafo, que minimize a soma de algum peso que a gente queira da minha árvore, mantendo os terminais conexos.

- Ela n precisa conter todos os elementos.
- Os pontos de Steiner são os pontos que não são terminais, mas que fazem parte da árvore de Steiner.

Quando a árvore de Steiner é resolvida de forma fácil?

- Quando o número de vértices é igual ao número de terminais.
- Pq é uma árvore geradora mínima para esse caso.

Árvore degenerada → Uma linha.

Estudar o que é uma **HEAP** mínima e máxima, principalmente por vetor.

PRIM, KRUSKAL E REMOÇÃO REVERSA geram a **MESMA** árvore se tiverem todos os pesos diferentes.

Árvore geradora máxima só funciona para pesos positivos para os algoritmos acima.

### Shortest Path

1. Uma origem e um destino → Djkistra funciona aqui.
2. Uma origem e vários destinos → Djkistra funciona aqui.
3. Várias origens e vários destinos → Djkistra não funciona de forma adequada aqui.

## 22/10

1.  $G = (V, E)$  em que  $V = \{1, 2, 3, 4\} \mid E = \{\{1, 2\}, \{3, 4\}\}$

$H = (V', E')$  em que  $V' = \{a, b, c, d\} \mid E = \{\{a, d\}, \{c, b\}\}$

$G = H$ ? Justifique sua resposta.

R: Dois grafos são iguais se e somente se  $V = V'$  e  $E = E' \rightarrow$  Mesmo conjunto de vértices e mesmo conjunto de arestas. Nesse caso, não são iguais.

$$V \subseteq V' \wedge V' \subseteq V$$

$$E \subseteq E' \wedge E' \subseteq E$$

Agora, caso seja possível mapear cada vértice de um grafo em um outro grafo, de forma que os dois terão vértices e arestas com as mesmas características, eles são **ISOMORFOS**.

- Mesma quantidade de vértices.
- Mesma quantidade de arestas.

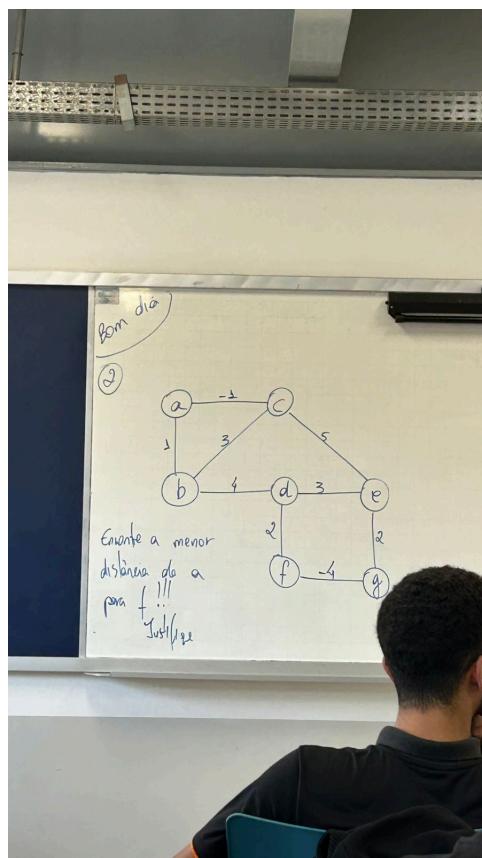
- Mesma sequência de graus.
- Vizinhos de um vértice com as mesmas características; Vizinhos dos vizinhos com as mesmas características (Mesmo mapeamento).
- Mesma quantidade de componentes.

Grafo associado → Grafo direcionado que tiramos a direção

- Adaptações ao processo de isomorfismo

Estrutura topológica → Dois grafos possuem a mesma estrutura, logo são **ISOMORFOS**.

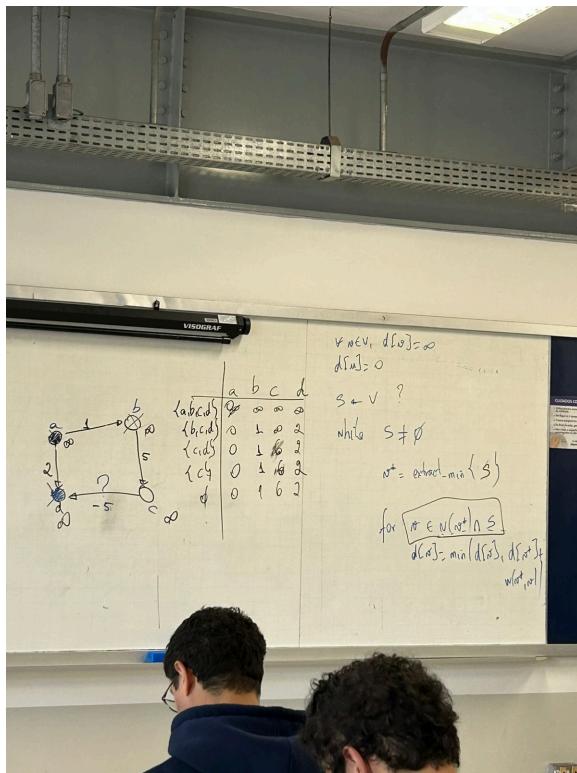
2. Questão 2:



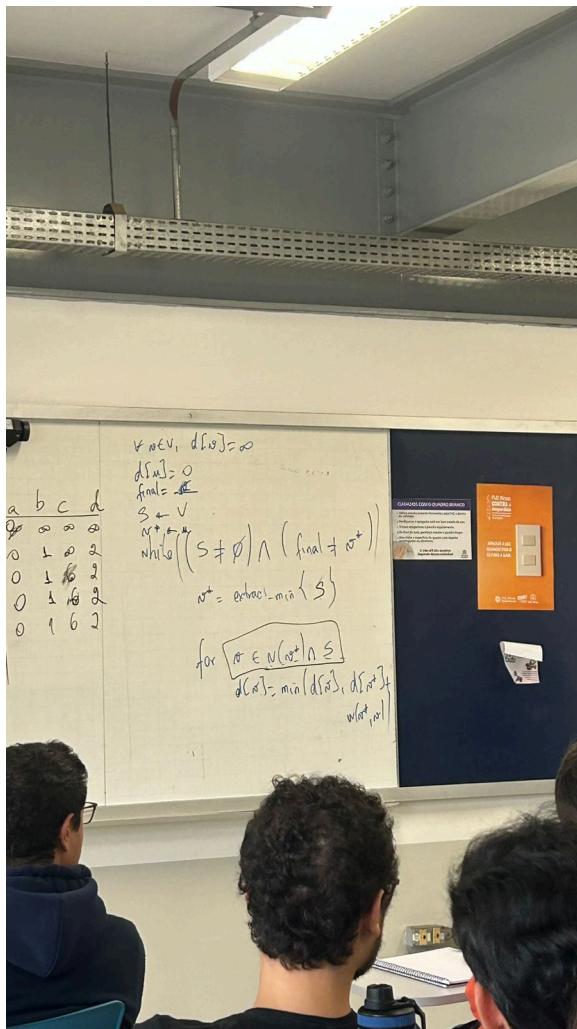
Bellman Ford não funciona para grafos não-direcionados.

## 23/10

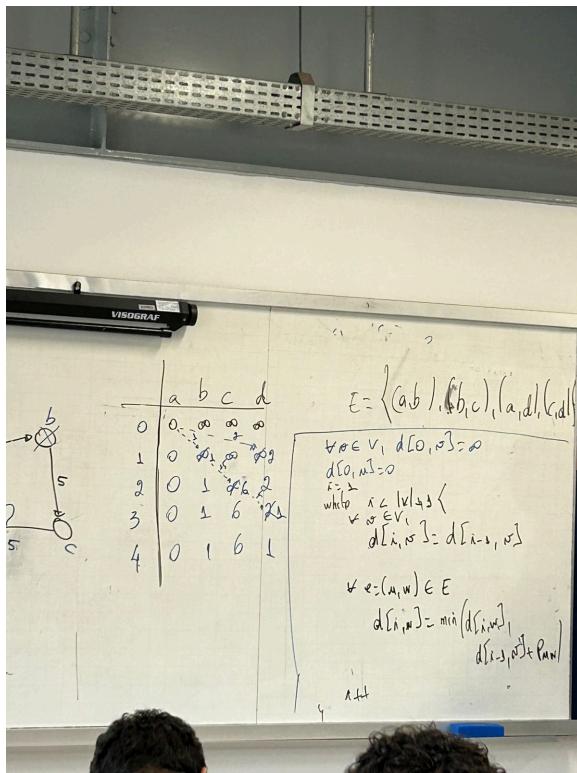
Dijkstra x Peso negativo → Um vértice já visitado pode não ser atualizado novamente, mesmo que seja necessário para encontrar o menor caminho, caso tenham números negativos e positivos.



Dijkstra com início e sem fim, com pesos positivos e negativos.



Dijkstra com início e fim, com pesos positivos e negativos.



Bellman Ford (O segundo "u" não se refere ao primeiro).

Grafos Eulerianos:

- Ciclo.
- Passar em todas as arestas uma única vez.
- Voltar para origem.

## 27/10 - Exercício 4, 12, 14

Exercício 4:

Em um rede de comunicação direcionada, uma mensagem é enviada de um nó  $a$  para outro nó  $b$ . Em cada arco  $(i, j)$  da rede, existe uma probabilidade  $p_{i,j}$  de que a mensagem se perca neste arco. Essas probabilidades são fixas e independentes. O gerente da rede deseja escolher um caminho de  $a$  até  $b$ , tal que a probabilidade de a mensagem se perder nesse caminho seja mínima. Traduza este problema em um problema de menor caminho. Este problema pode ser resolvido com o algoritmo de Dijkstra?

1. Produto das probabilidades que estão em cada aresta.
2.  $\log b(a.c) = \log b(a) + \log b(c) \rightarrow$  Temos que fazer essa transformação, pois o Dijkstra não funciona fazendo o produto do peso das arestas.

3. Aplicamos  $\text{logb}(w(e)) \rightarrow$  Aplicamos o log em cada probabilidade (peso) das arestas.
4.  $0 < w(e) < 1 \rightarrow$  resultado negativo.
5.  $w'(e) = -\text{logb}(w(e)) \rightarrow$  Negamos o resultado do log para que ele fique positivo.
6. Aplicamos o Dijkstra.

### Exercício 12:

Seja um grafo conexo  $G$  com pesos positivos e uma árvore geradora mínima de  $G$ . Assinale a(s) afirmativa(s) correta(s) para manter a mesma árvore geradora mínima.

- a) Altere o peso de cada aresta de  $w(e)$  para  $w(e) + 15$ .
- b) Altere o peso de cada aresta de  $w(e)$  para  $w(e) \times 15$
- c) Altere o peso de cada aresta de  $w(e)$  para  $w(e) \times w(e)$

### AGM → Usando Kruskal:

- Grafo conexo.
- Ordena “Não-Decrescente”.
- Análise de arestas que estão gerando ciclos  $\rightarrow$  A árvore geradora mínima contém a menor quantidade de arestas, desde que não tenha ciclos.
- $T = (V, E') \subseteq G$   
 $T' = (V, E'') \subseteq G$   
onde  $E' = E''$
- Para essa questão, devemos preservar a ordem da análise das arestas.
  - Isso nos leva a aplicar a mesma função tanto no grafo  $G$ , quanto na AGM.
  - $(T, w)$  por exemplo.
  - Como nessa questão o peso é **SEMPRE** positivo, está tudo bem e podemos afirmar **TODAS** as afirmativas.
  - A atenção está para quando existem pesos negativos, pois a ordem de cada aresta pode variar.
    - $w: E \rightarrow N \mid$  Funciona

- $w: E \rightarrow Z^+ \mid$  Funciona
- $w: E \rightarrow R^+ \mid$  Funciona
- Não funciona com certeza quando tivermos pesos negativos, ou quando a constante que multiplicarmos for negativa.

Exercício 14:

Projete um algoritmo para determinar a menor mudança no custo da aresta que produziria uma mudança na árvore geradora mínima.

$$G = (V, E)$$

$$T = (V, E') \text{ em que } E' \subseteq E$$

$$T' = (V, E'') \text{ em que } E'' \subseteq E$$

$T$  é uma AGM se  $\nexists T'$  tal que a soma de pesos das arestas de  $T'$  seja menor que a soma de pesos das arestas de  $T$ .

Passo a passo de para montar uma AGM com o Kruskal:

1. Ordena de forma “não decrescente” os pesos das arestas.
2. Começa a unir as arestas, analisando sempre se a inclusão de uma aresta irá criar um ciclo. Se criar, não é incluída.

Para essa questão, devemos fazer um algoritmo que acha ciclos no grafo original. No ciclo vamos verificar a menor ...

## 28/10

1. Seja  $G = (V, E)$  um grafo não-direcionado. Analise e responda:

a. Se  $G$  é euleriano, então  $G$  é hamiltoniano?

Podemos dizer que é possível ter um grafo euleriano e hamiltoniano, mas não afirmar que sendo euleriano ele é hamiltoniano sempre. O motivo disso é: para ser euleriano ele deve passar por todas as arestas, mas sem repetí-las. Passar por todas arestas implica visitar todos os vértices, mas não podemos garantir que os vértices não serão repetidos. Caso tenham vértices repetidos, não podemos dizer que ele é hamiltoniano. Um ciclo em forma de um triângulo, por exemplo, é euleriano e hamiltoniano, pois ele consegue passar por todas as arestas

sem repetição das mesmas e em todos os vértices sem repetição dos mesmos, exceto do inicial que também é o ponto de retorno. Agora, um pentágono com um vértice diagonal em seu interior não pode ser euleriano e hamiltoniano, apenas euleriano. Isso acontece justamente porque, nesse caso, é possível visitar todas as arestas sem repetí-las, mas impossível não repetir um vértice. Ou seja, quando temos dois ciclos que compartilham um vértice entre si, não é possível que o grafo tenha ambas classificações.

b. Se  $G$  é hamiltoniano, então  $G$  é euleriano?

A resposta é não, pois temos um quadrado, por exemplo, em que é possível passar em todos os vértices, mas não é possível passar em todas as arestas. (Quando a resposta for não, podemos apenas mostrar um contra exemplo)

2. Seja  $G = (V, E)$  um grafo direcionado, projete um algoritmo para encontrar o maior caminho de  $G$ , caso possível. Façam as considerações que julgarem necessárias.

Temos três opções:

Primeiramente devemos considerar que temos um grafo **ACÍCLICO**.

- Partimos do pressuposto que fizemos uma ordenação topológica (colocar em uma ordem de forma que todas as arestas saiam da esquerda para direita - utilizamos o algoritmo de remoção de bases para isso). Essa ordenação define a ordem de análise dos vértices, garantindo que todo o fecho transitivo inverso de um  $V$  já tenha sido analisado quando chega a vez dele.
- Utilizamos o algoritmo de remoção de bases atualizando a distância e os antecessores de cada vértice, sempre atualizando o antecessor pelo último encontrado.
- Utilizamos o Dijkstra nas bases do grafo e, antes disso, colocamos o peso das arestas como -1.

3. Seja  $G = (V, E)$  um grafo não-direcionado e  $(G, w)$  um grafo ponderado.

Seja  $T = (V, E')$  uma árvore obtida pelo Dijkstra para achar o maior caminho e  $T' = (V, E'')$  uma árvore geradora máxima. Podemos afirmar que a aresta de maior peso está em ambas as árvores?

Dijkstra só funciona para acíclicos e com pesos todos positivos ou todos negativos.

Em grafos acíclicos a resposta é verdadeiro.

Em grafos cíclicos a resposta é falsa.

## 02/11 - Estudos para a prova do dia 03/11

A capacidade mínima de um caminho em um grafo ponderado é definida como o peso da aresta de menor peso do caminho. O problema de *Widest path* é um definido como um problema de encontrar, entre dois vértices especificados em um grafo ponderado, um caminho que maximize o peso dentre todas as capacidades mínimas (dos caminhos) do grafo. Este problema também é conhecido como *problema de caminho de capacidade máxima*. Projete uma solução para resolver o problema de caminho de capacidade máxima entre estes vértices. Discorra sobre o custo computacional da sua solução.

- Para solucionar esse problema, é necessário encontrar o caminho em que o menor peso entre suas arestas seja o maior peso entre as demais menores arestas dos outros caminhos. Desse modo, para encontrar esse caminho, podemos utilizar um algoritmo de busca em profundidade para identificar todos os caminhos possíveis entre dois vértices e, em seguida, identificar os menores pesos das arestas desses caminhos. Por fim, bastaria comparar esses pesos e escolher, dentre eles, o maior. O custo dessa solução não é bom, pois tratando de grafos massivos, a busca por profundidade em todos os vértices seria de custo altíssimo. → **FORÇA BRUTA E INTRATÁVEL PARA GRAFOS MASSIVOS**
- Para solucionar esse problema, podemos utilizar o conceito de Árvore Geradora Máxima que, por definição, é um subgrafo que mantém todos os vértices do grafo original conexos por meio de um caminho que maximize a soma dos pesos das arestas. Essa árvore é útil para o nosso caso, pois, enquanto maximiza a soma dos pesos, garante que o caminho que percorre todos os vértices nela tenha o maior menor peso entre qualquer outro caminho que exista no grafo original entre os mesmos vértices. Desse modo, ao encontrar a árvore gerada máxima, conseguimos definir a capacidade máxima desse grafo, sendo ela igual ao menor peso presente no caminho entre os dois vértices foco.

Em um rede de comunicação direcionada, uma mensagem é enviada de um nó  $a$  para outro nó  $b$ . Em cada arco  $(i, j)$  da rede, existe uma probabilidade  $p_{i,j}$  de que a mensagem se perca neste arco. Essas probabilidades são fixas e independentes. O gerente da rede deseja escolher um caminho de  $a$  até  $b$ , tal que a probabilidade de a mensagem se perder nesse caminho seja mínima. Traduza este problema em um problema de menor caminho. Este problema pode ser resolvido com o algoritmo de Dijkstra?

- Do modo como nos é entregue o problema não é possível resolver com o Dijkstra, pois o menor caminho nesse caso está atrelado a probabilidade e uma multiplicação entre elas (pesos nas arestas) deve ser feita para encontrar a probabilidade total da mensagem se perder por um caminho. No entanto, ainda é possível utilizar o Dijkstra desde que façamos uma mudança em como as informações serão calculadas. Na matemática, temos que  $\log x(a.b) = \log x(a) + \log x(b)$ , então é possível fazer uma transformação no grafo, alterando os pesos das arestas para seus logarítmicos em uma base padrão para todos. Assim, quando o Dijkstra realizar a soma dos pesos das arestas para encontrar o menor caminho, a ideia de multiplicação de probabilidades ainda estará sendo respeitada.

Seja  $e$  uma aresta arbitrária em um grafo  $G$ . É sempre possível construir uma árvore geradora de  $G$  que contenha  $e$ ? E se tivermos um conjunto de arestas?

- A árvore geradora, por definição, é um subgrafo que mantém todos os vértices do grafo original conexos e sem existência de ciclos. Podemos dizer que é sempre possível construir uma árvore geradora de  $G$  que contenha uma aresta arbitrária  $e$  de  $G$ , pois ao forçar a inserção dessa aresta na árvore, podemos optar por retirar alguma outra que esteja criando um ciclo. Se essa aresta não acabar criando um ciclo e, na verdade, ser uma ponte, é garantido que ela estaria na árvore. A lógica para o conjunto de arestas é parecido: é necessário verificar a existência de algum ciclo após a inserção desse conjunto. Se não existe nenhum ciclo, podemos manter essas arestas na árvore.

Em alguns problemas queremos que certos pares de vértices estejam diretamente conectados entre si. Modifique o algoritmo de Prim para resolver o problema da árvore geradora mínima com esta restrição adicional.

- O algoritmo Prim gera a árvore geradora mínima a partir de um vértice passado. Como queremos que obrigatoriamente alguns vértices tenham conexões com outros vértices, podemos criar uma estrutura que armazene essas conexões e que elas tenham prioridade no início do Prim. Aconteceria da seguinte forma: Adicionaríamos as arestas que desejamos que existam nessa árvore geradora mínima nessa estrutura. Iniciaríamos o Prim, de forma que passamos um vértice e a estrutura. O Prim, por sua vez, antes de seguir o procedimento padrão, iria inserir a primeira aresta da estrutura, verificar se não tem ciclo e, caso não tenha, passar para a próxima. Desse modo, ao terminar esse processo, teríamos componentes

conexos distintos, alguns com mais de um vértice, mas bastaria seguir agora com o algoritmo padrão, onde ele analisa os vizinhos do vértice de início, conecta ele com outro vértice pela aresta de menor peso e repete o processo, sempre conferindo se não está criando ciclos. → **ACABEI**

### **MODIFICANDO O KRUSKAL, NÃO O PRIM**

- O algoritmo Prim gera a árvore geradora mínima a partir de um vértice, de forma que ele vai crescendo um único componente em relação aos outros a cada interação. Nesse problema, precisamos garantir que teremos algumas arestas e por isso temos um problema ao utilizar o Prim, afinal ele não é utilizado para quando temos vértices que já estão conectados fora do componente que ele está “cultivando”. Desse modo, proponho que um novo grafo seja criado com base no original para que nele seja possível aplicar o Prim. Devemos identificar os vértices que estarão conectados pelas arestas obrigatórias e, a partir deles, criar os super-vértices necessários. Fazendo isso, cada componente conexo estará sendo representado por um único ponto e, agora, será possível utilizar o Prim a partir de um vértice, que será cultivado até encontrar o resultado.

Seja  $G$  um grafo não-direcionado com pesos que podem ser positivos ou negativos. Podemos afirmar que os algoritmos de Prim e Kruskal produzirão árvores geradoras mínimas corretas para  $G$ ?

- Sim, podemos afirmar isso. O Prim e o Kruskal funcionam seguindo uma ordem de análise das arestas. No Kruskal, as arestas são ordenadas com base nos seus pesos de forma não decrescente e, a cada interação, são adicionadas na AGM caso não criarem um ciclo. No Prim, a ordem de análise das arestas acontece da menor para a maior a partir dos vértices que estão na fronteira dos vértices já analisados. Desse modo, independente se os pesos são todos positivos, todos negativos ou ambos, os dois algoritmos funcionam, pois um número negativo segue sendo menor que um positivo e a ordem de análise se mantém correta.

Seja  $G = (V, E)$  um grafo não-direcionado, e  $(G, W)$  um grafo não-direcionado ponderado. Considere que  $T = (V, E')$  seja uma árvore geradora de  $G$ . Uma árvore geradora máxima é uma árvore geradora em que

a soma dos pesos das arestas é máxima dentre todas as árvores geradoras. Altere os algoritmos de Prim e Kruskal para encontrar uma árvore geradora máxima.

- No Kruskal, devemos alterar a ordenação das arestas para análise. O que antes era em ordem não-decrescente agora será em ordem não-crescente.

Desse modo, o algoritmo vai analisar primeiro as arestas de maior peso e ir avançando para as de menores peso, verificando a inserção para que nenhum ciclo seja criado.

No Prim, devemos alterar a forma em que as arestas são retiradas da fila de prioridade e inseridas no resultado. O que antes era feito de modo que a de menor peso era retirada da fila, agora deve ser a que tem o maior peso.

Considere um algoritmo “reverso” do Kruskall para calcular uma árvore geradora mínima. Inicialize  $E'$  como seja o conjunto de toda as arestas do grafos. Em seguida, considere as arestas em ordem decrescente dos pesos das arestas. Para cada aresta, remova-a de  $E'$  se esta aresta pertence a um ciclo em T. Assuma que todas as arestas são distintas. Este algoritmos encontraria corretamente uma árvore geradora mínima?

- Sim.

Seja um grafo conexo G com pesos positivos e uma árvore geradora mínima de G. Assinale a(s) afirmativa(s) correta(s) para manter a mesma árvore geradora mínima.

- Altere o peso de cada aresta de  $w(e)$  para  $w(e) + 15$ .
- Altere o peso de cada aresta de  $w(e)$  para  $w(e) \times 15$
- Altere o peso de cada aresta de  $w(e)$  para  $w(e) \times w(e)$

Todas estão corretas, pois estamos mudando apenas a escala dos pesos.

Mantendo a escala igual para todos os pesos, nenhum deles estaria mudando alguma ordem de análise, gerando a mesma árvore geradora mínima.

**11/11**

Seja  $G = (V_1 \cup V_2, E)$  um  
grafo bipartido. Um matching  
é dado por um subgrafo  $H$   
em que duas arestas de  $H$   
não podem ser adjacentes.  
Projetar uma solução para maximizar  
o tamanho do conjunto de arestas de  $H$ .

- Podemos encontrar o número de subgrafos, depois encontrar esses subgrafos e, dentre eles, encontrar a maior quantidade de arestas → Esse não está tão correto, mas não está absolutamente errado para tudo.
- Para poder utilizar a lógica de fluxo, o grafo tem que ser:
  - Direcionado
  - Ter **Raiz** e **Anti-raiz**
  - Pesos positivos e inteiros
    - Para resolução, inserimos uma raiz e uma anti-raiz no grafo (S-T), de forma que uma se conecte a um dos vértices de um dos subconjuntos do grafo bipartido e a outra ao outro subconjunto, respectivamente.
    - Ao transformar um grafo não-direcionado em direcionado, teremos alguns ciclos e a questão de fluxo busca entender os caminhos disjuntos simples de arestas.
      - Para resolver isso, alteramos o grafo **direcionado** original de modo que mantemos as arestas que respeitem o fluxo do grafo com a raiz e anti-raiz.
- Método final:
  - Adaptamos como o nosso contexto atual do nosso grafo está para podermos implementar o algoritmo que encontra o fluxo máximo, que nesse caso é equivalente a maximizar o tamanho de conjunto de arestas de H no exercício proposto.
  - Problema: Matching em um grafo bipartido de cardinalidade máxima.
    - Matching é um subgrafo que tem conjunto de vértices e arestas em que o conjunto de arestas não pode ter arestas adjacentes e não pode ter vértices isolados.
    - Baseado em fluxo.

**12/11**

Seja  $G = (V, E)$  um grafo  
não-direcionado.

Projetete uma solução para encontrar um subgrafo nulo  $H$  de  $G$  de maior cardinalidade de vértices.

$$S_1 = \emptyset$$

$$V_1 = V$$

while  $V \neq \emptyset$

$v = \text{seleciona\_vértice\_menor\_grau}(G)$

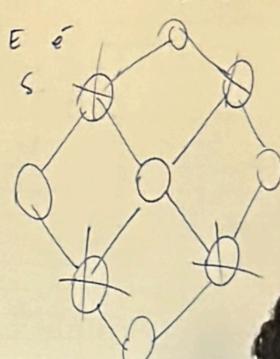
$S_1 = S_1 \cup \{v\}$

    remove  $v$  e vizinhos de  $G$

Esse algoritmo acima pode ser alterado para realizar o algoritmo abaixo.  
Para isso, selecionamos o vértice de **maior** grau.

Seja  $G = (V, E)$  um grafo não-direcionado.  
Encontre um conjunto  $S \subseteq V$  de menor cardinalidade

que forma que toda aresta  $e \in E$  é vizinha de algum vértice de  $S$ .



Não tem solução ótima para esse problema. Ele é chamado de "cobertura de vértices". No caso do exemplo, se usarmos um algoritmo para verificar o grau dos vértices, selecionar os de maior grau junto das arestas conectadas a ele e

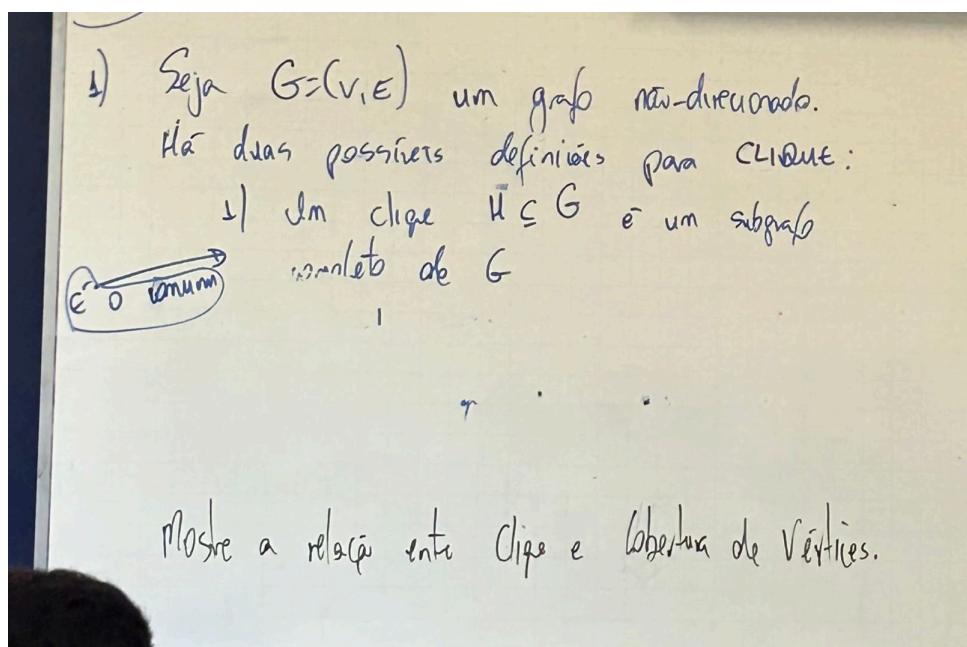
guarda-los, teremos encontrado uma resposta próxima da correta, pois em alguns casos não será de fato a **melhor** de todas. No desenhado pelo professor, essa solução não é a melhor, afinal poderíamos pegar apenas os vértices do meio de cada lado, totalizando 4, e o algoritmo proposto encontraria 5.

- **Como os dois problemas se relacionam?**

- Ao utilizar o algoritmo proposto acima, estamos minimizando o número de vértices para cobrir todas as arestas. Então, ao fazer isso, estamos identificando os vértices que não vão fazer parte desse conjunto e eles serão justamente o subgrafo induzido nulo do grafo original, com a maior quantidade de vértices.
- Dar errado não necessariamente é que o algoritmo não funciona, mas sim que ele não encontra a solução ótima.

Se  $H = (V', E')$  é um subgrafo induzido por  $V'$  é nulo em  $G = (V, E)$  então  $H_1 = (V', E'')$  é um subgrafo induzido por  $V'$  e será completo em  $G'$ , onde  $G'$  é o complemento de  $G$

**13/11**



<p><math>S</math> é uma cobertura de vértices de <math>G = (V, E)</math> se <math>\forall e \in E</math>, ou <math>u \in S</math> ou <math>v \in S</math>.</p>	<p><math>S</math> é um conjunto independente de <math>G = (V, E)</math> se <math>\forall e \in E</math> <math>\forall u, v \in S</math>.</p>
$S_1 = \{a, b\}$	$V = \{a, b, c, d, e\}$ $E = \{\{a, b\}, \{a, c\}, \{b, d\}, \{a, b, c\}\}$ $S = \{e, c, d\}$

Pode ser que seja necessário, em  $S$ , que todo vértice tenha uma aresta correspondente em  $G$ . Temos que se atentar à definição que estamos utilizando (Nunca utilizamos assim na vida real, essa possibilidade é para entendermos como alterar a definição pode alterar o problema).

$S = \{a, c\}$ $G' = (V', E') \subseteq G$ se $G'$ é completo entre	$S_2$ é uma cobertura de vértices de $G = (V, E)$ se $\forall e \in E$ , ou $u \in S_2$ ou $v \in S_2$ .	$S$ é um conjunto independente ( $c$ ) de $G = (V, E)$ se $\forall e \in E$ $\forall u, v \in S$ .
$\begin{cases} V \setminus V' \text{ é CR em } \bar{G} \\ \text{CI em } \bar{G} \end{cases}$	$\begin{cases} S_1 \text{ é CI em } G, \text{ se } V \setminus S_1 \text{ é CR em } \bar{G} \\ \text{Se } S_2 \text{ é CI em } G, \text{ se } S_2 \text{ não é CI em } \bar{G} \end{cases}$	$\begin{cases} S_1 \text{ é CI em } G, \text{ se } V \setminus S_1 \text{ é CR em } \bar{G} \\ \text{Se } S_2 \text{ é CI em } G, \text{ se } S_2 \text{ não é CI em } \bar{G} \end{cases}$

$V \setminus V' \rightarrow \setminus$  significa subtração.