Grafos

Problemas e algoritmos clássicos

Profs. André, Salles

Departamento de Informática
Universidade Federal de Viçosa

INF 333 - 2024/1

Árvore

Definição

Uma árvore é um grafo conectado acíclico

- obs.: uma árvore de n vértices sempre tem $n - 1$ arestas

Exercicio

Exercicio

Como descobrir facilmente (e rapidamente) se um grafo possui ciclos?

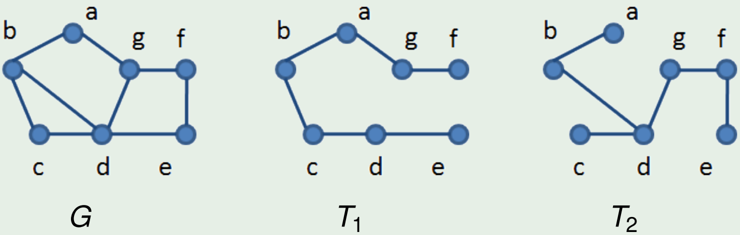
Árvore Geradora

Definição

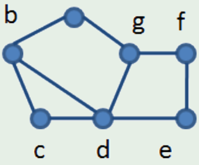
Uma árvore geradora de um grafo G é um subgrafo que contém todos os vértices e que é uma árvore

- ou seja, é um subgrafo conectado acíclico que contém todos os vértices

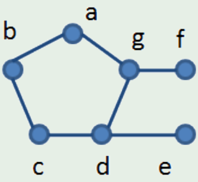
Exemplo



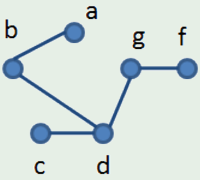
Exemplos que NÃO são árvores geradoras



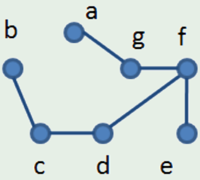
G



T_1



T_2



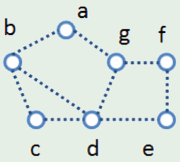
T_3

Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

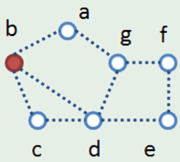


Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

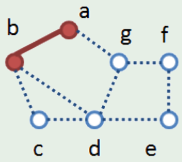


Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

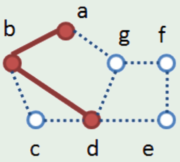


Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

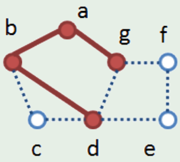
Exemplo



Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$
    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

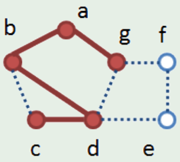


Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1 ... (|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

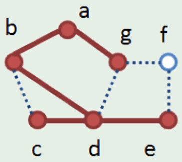


Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

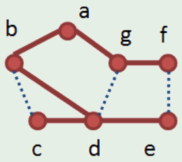


Algoritmo para construção de uma árvore geradora de um grafo

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo



Problema da Árvore Geradora Minima

Definição

O peso de um grafo valorado é a soma dos pesos das arestas do grafo

Definição

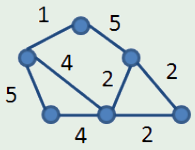
A árvore geradora mínima é a árvore geradora de peso mínimo

Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

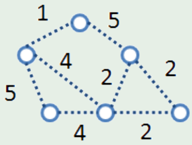


Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

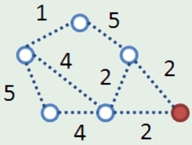
Exemplo



Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$
    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

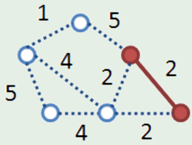
Exemplo



Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$
    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

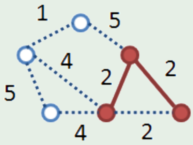


Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

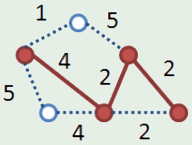
Exemplo



Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$
    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo

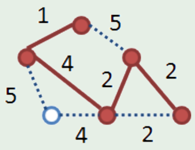


Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$

    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

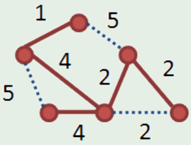
Exemplo



Algoritmo de PRIM para árvore geradora mínima

- $v \leftarrow$ um vértice qualquer de G
- $T_1 \leftarrow \{v\}$
- Para $k = 1...(|V| - 1)$
  - $e_k \leftarrow$ uma aresta* (u, v) de peso mínimo com $u \in T_k$ e $v \notin T_k$
    * se houver mais de uma escolha qualquer uma delas
  - $T_{k+1} \leftarrow T_k \cup e_k + v$
- Retornar $T_{|V|}$

Exemplo



Métodos

- Kruskal
  - Ordena as arestas por peso
  - Seleciona arestas nessa ordem, descartando as que formam ciclo com as já escolhidas
  - Para verificar ciclos de forma eficiente pode-se usar a estrutura de dados *union-find*
- Prim
  - É o algoritmo mostrado no slide anterior
  - Começa a árvore com um vértice qualquer e vai acrescentando um por um à árvore
  - O vértice acrescentado é o de menor distância até algum da árvore

Obs.: o método de Prim pode ser implementado como o de Dijkstra a seguir; a única diferença é no “if” final, que não soma D_v, usa apenas o d_{vu}

Árvore Geradora Mínima

Veja exemplos de execução do método Prim e Kruskal:
`https://visualgo.net/en/mst`

Tutorial e códigos de Kruskal e Prim: (clique aqui)

Digrafo

Definição

Um grafo direcionado é um par ordenado $D = (V, A)$, sendo V um conjunto de vértices, e A um conjunto de pares ordenados de vértices $\in V$

- Note que uma aresta nesse caso é um par ordenado (u, v), não um conjunto $\{u, v\}$

Exemplo

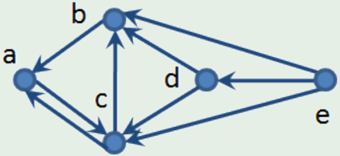
$D = (V, A)$
$V = \{a, b, c, d, e\}$
$A = \{(a, c), (b, a), (c, a), (c, b), (e, b), (e, d), (d, b), (e, c), (d, c)\}$

Digrafo

- Grafos direcionados são também chamados digrafos ou grafos dirigidos
- As arestas são chamadas arcos, e representadas por setas
- Note que a existência de um arco (u, v) não implica a existência de (v, u)

Exemplo

$$D = (V, A)$$
$$V = \{a, b, c, d, e\}$$
$$A = \{(a, c), (b, a), (c, a), (c, b), (e, b), (e, d), (d, b), (e, c), (d, c)\}$$

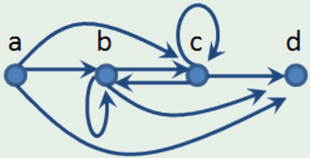


Definição

O fecho transitivo de um digrafo D é um digrafo D^* contendo o arco (u, v) se existe caminho direcionado de u a v em D

Exemplo

D



D^*

Fecho transitivo

Algoritmo - Warshall

$D^* \leftarrow D$
para $k = 1 \dots n$
  para $i = 1 \dots n$
    se (i, k) é arco em D^*
      para $j = 1 \dots n$
        se (k, j) é arco em D^*
          acrescente arco (i, j) em D^* (se ele não existia)

Fecho transitivo

Algoritmo - Warshall

$D[i][j] \leftarrow$ TRUE (se $(i,j) \in D$) ou FALSE (se $(i,j) \notin D$)

para $k = 1 \ldots n$
  para $i = 1 \ldots n$
    para $j = 1 \ldots n$
      $D[i][j] \leftarrow D[i][j] \vee (D[i][k] \wedge D[k][j])$

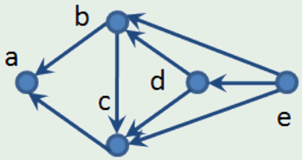
Ordenação Topológica

Definição

Um DAG (directed acyclic graph) é um grafo dirigido sem ciclos direcionados

- Cuidado! Não é necessariamente árvore nem conectado!

Exemplo

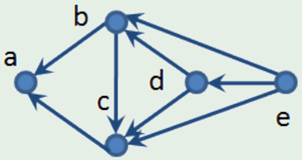


Ordenação Topológica

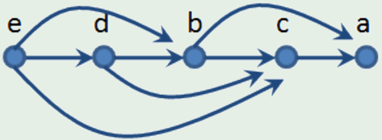
Definição

Uma ordenação topológica de um DAG é uma sequência de vértices tal que não exista arco de um vértice a algum anterior na sequência

Exemplo



- e, d, b, c, a



Ordenação Topológica

- Como construir uma ordenação topológica $s_1 s_2 \ldots s_n$ do DAG D?
- Um algoritmo simples consiste em repetidamente selecionar e remover vértice com grau de entrada zero (sem arcos chegando)

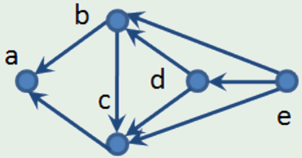
Algoritmo

para $i = 1 \ldots n$

$s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$

$D \leftarrow D - v$

Exemplo



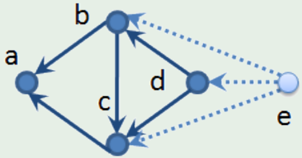
Ordenação:

Ordenação Topológica

Algoritmo

para $i = 1 \ldots n$
  $s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
  $D \leftarrow D - v$

Exemplo



Ordenação: *e*

Ordenação Topológica

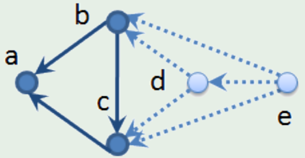
Algoritmo

para $i = 1 \dots n$

$s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$

$D \leftarrow D - v$

Exemplo



Ordenação: e, d

Ordenação Topológica

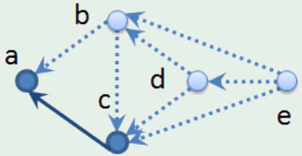
Algoritmo

para $i = 1 \ldots n$

$\quad s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$

$\quad D \leftarrow D - v$

Exemplo



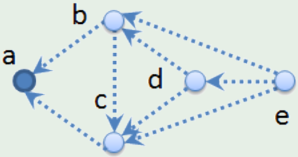
Ordenação: e, d, b

Ordenação Topológica

Algoritmo

para $i = 1 \ldots n$
$\quad s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
$\quad D \leftarrow D - v$

Exemplo



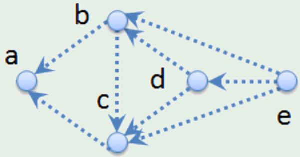
Ordenação: e, d, b, c

Ordenação Topológica

Algoritmo

para $i = 1 \ldots n$
  $s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
  $D \leftarrow D - v$

Exemplo



Ordenação: e, d, b, c, a

Ordenação Topológica

Algoritmo

para $i = 1 \dots n$
    $s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
    $D \leftarrow D - v$

Exemplo

- Note que só funciona em DAG
- Se houver ciclo direcionado, faltará v com $\delta^-(v) = 0$

- Construir a ordenação topológica $s_1 s_2 \dots s_n$ do digrafo D
- Como implementar o algoritmo abaixo de forma eficiente?

Algoritmo

para $i = 1 \dots n$
  se não há vértice $v \in D$ com $\delta^-(v) = 0$
    Erro, não é DAG!; break
  $s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
  $D \leftarrow D - v$

Problema do Caminho Mínimo

Problema do caminho mínimo

Dado um grafo com peso nas arestas, e dois vértices s e t, determinar o caminho-(s, t) de peso mínimo

Exemplo

...

Para encontrar o caminho mínimo de s para t, os algoritmos que resolvem o problema encontram também outros caminhos mínimos.

Problema do Caminho Mínimo

Single-source shortest paths

Dado um grafo com peso nas arestas e um vértice s, determinar o caminho-(s, v) de peso mínimo para cada vértice $v \neq s$.

ou seja, determinar o caminho mínimo de s até cada um dos outros

All-pairs shortest paths

Determinar o caminho-(v, u) de peso mínimo para cada par de vértices v e u.

ou seja, determinar o caminho mínimo de todos para todos

Método de Dijkstra

d_{uv}: dado de entrada, distância direta entre u e v (∞ se não tem aresta)
D_v: distância minima já conhecida de s a v
T: conjunto de vértices fechados (cuja distância mínima já foi estabelecida)
S: conjunto de vértices abertos (cuja dist. mínima ainda pode ser atualizada)

`// Single-source shortest paths`
Dijkstra (grafo G, vértice s)
    foreach $v \in V(G)$ `//Inicialização`
        $D_v \leftarrow \infty$
    $D_s \leftarrow 0$
    $T \leftarrow \emptyset$
    $S = V(G)$
    while $S \neq \emptyset$ `//Relaxamento`
        $v \leftarrow$ EXTRACT-MIN (S) `//Retira de S o` v `com menor` D_v
        $T \leftarrow T \cup v$
        foreach u adjacente a v
            if $u \notin T$ and $D_u > D_v + d_{vu}$ then
                $D_u = D_v + d_{vu}$

Veja exemplos de execução do método de Dijkstra:
`https://visualgo.net/en/sssp`

Tutorial e códigos de Dijkstra e Floyd-Warshall: (clique aqui)

Método de Floyd-Warshall

D_{ij}: inicialmente a distância direta de i para j (matriz de adjacência)
$= 0$ se $i = j$
$= d_{ij}$ se existe aresta (i, j)
$= \infty$ se não existe aresta (i, j)
No final D_{ij} será a distância mínima entre i e j

`// All-pairs shortest paths`
Floyd-Warshall (grafo G)
    foreach $k \in V$
        foreach $i \in V$
            foreach $j \in V$
                if $D_{ij} > D_{ik} + D_{kj}$ then
                    $D_{ij} = D_{ik} + D_{kj}$

Método de Floyd-Warshall

Na k-ésima iteração do laço mais externo: caminho mínimo que passa apenas pelos vértices 1...k como intermediários.

`// All-pairs shortest paths`

Floyd-Warshall (grafo G)

    foreach $k \in V$
        foreach $i \in V$
            foreach $j \in V$
                if $D_{ij} > D_{ik} + D_{kj}$ then
                    $D_{ij} = D_{ik} + D_{kj}$

D_{ij}: valor de i para j (matriz de adjacência)

$= 0$ se $i = j$

$= d_{ij}$ se existe aresta (i, j)

$= \infty$ se não existe aresta (i, j)

Além de poder ser utilizado para minimizar a soma das arestas ao longo de um caminho, também pode ser adaptado para minimizar a aresta mais cara (min-max). Adicionalmente, também resolve o max-min.

Floyd-Warshall (grafo G)
    foreach $k \in V$
        foreach $i \in V$
            foreach $j \in V$
                $D_{ij} = min(D_{ij}, max(D_{ik}, D_{kj}))$

Métodos de solução

- Algoritmo de Dijkstra
  - descobre o menor caminho de 1 para todos
  - não funciona se houver peso negativo
  - Guloso
  - $O(V^2)$ ou $O(ElogV)$ (dependendo da implementacao)
- Algoritmo de Floyd
  - descobre o menor caminho de todos para todos
  - funciona inclusive com peso negativo
    (se tiver ciclo negativo –> um valor da diagonal principal será neg.)
  - PD
  - Cada passo do for externo: menor caminho usando vértices $1..k$ como intermediários.
  - $O(V^3)$
- Algoritmo de Bellman-Ford
  - descobre o menor caminho de 1 para todos
  - funciona com peso negativo (e detecta ciclo negativo)
  - PD
  - Cada passo do for externo: menor caminho usando até k arestas.
  - $O(VE)$ (bom quando?)