MC202 - Estrutura de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Grafos

Sabemos que

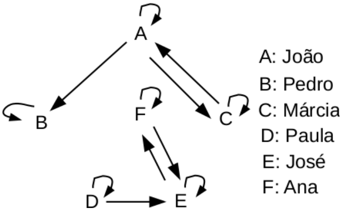
- um conjunto é uma coleção de objetos (pessoas, imagens, cidades, números, figuras geométricas),
- as informações sobre esses objetos podem ser armazenadas em *structs* (denominados nós),
- esses nós armazenados em diferentes tipos de estruturas de dados (vetores, listas, árvores), e que
- pares de objetos de um conjunto podem satisfazer a diferentes tipos de relação binária (maior que, irmão de, contido em).

Sabemos que

- um conjunto é uma coleção de objetos (pessoas, imagens, cidades, números, figuras geométricas),
- as informações sobre esses objetos podem ser armazenadas em *structs* (denominados nós),
- esses nós armazenados em diferentes tipos de estruturas de dados (vetores, listas, árvores), e que
- pares de objetos de um conjunto podem satisfazer a diferentes tipos de relação binária (maior que, irmão de, contido em).

Podemos então representar graficamente um conjunto de objetos e uma relação binária da seguinte forma.

Grafos



- $\mathcal{N} = \{A, B, C, D, E, F\}$ é o conjunto de objetos.
- $\mathcal{A} = \{(A, B),(A, C),(C, A),(F, E),(E, F),(D, E),(A, A),$ $(B, B),(C, C),(D, D),(E, E),(F, F)\}$ é o conjunto de pares ordenados de objetos que satisfazem uma dada relação binária.
- Por exemplo, $(A, B) \in \mathcal{A}$ e $(B, A) \notin \mathcal{A}$ pode significar que João sabe quem é Pedro, mas o contrário não é verdadeiro.

Definição canônica de Grafo

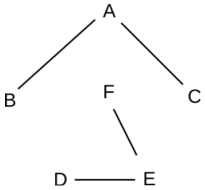
- Um grafo G, portanto, é um par $(\mathcal{N}, \mathcal{A})$, onde $\mathcal{N}$ é o conjunto de nós (vértices) e $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ é o conjunto de arcos (arestas).
- Se $(u, v) \in \mathcal{A}$, então o arco incide em $v \in \mathcal{N}$.
- O conjunto $\mathcal{A}$ descreve uma relação de adjacência, pois se $(u, v) \in \mathcal{A}$, para $u, v \in \mathcal{N}$, então o nó v é dito ser adjacente ao nó u no grafo.
- O conjunto $\mathcal{A}(u)$ contém os nós adjacentes ao nó u no grafo. Então se $(u, v) \in \mathcal{A}$, é porque $v \in \mathcal{A}(u)$.

Agenda

- Tipos de grafo e conceitos relacionados.
- Formas de representação.
- Percusos em grafo.
- Algoritmos de busca em grafo.

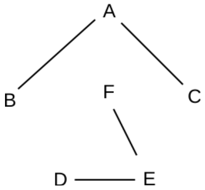
Grafo orientado e ponderado

- Se $\forall (u, v) \in \mathcal{A}$, $(u, v) = (v, u)$, então as setas são omitidas na representação gráfica e o grafo é dito não orientado.



Grafo orientado e ponderado

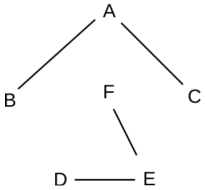
- Se $\forall (u, v) \in \mathcal{A}$, $(u, v) = (v, u)$, então as setas são omitidas na representação gráfica e o grafo é dito não orientado.



- Um grafo $G = (\mathcal{N}, \mathcal{A}, w)$ é dito ponderado quando exite uma função $w : \mathcal{A} \rightarrow \Re$ que associa um peso $w(u, v)$ a cada arco $(u, v) \in \mathcal{A}$ do grafo.

Grafo orientado e ponderado

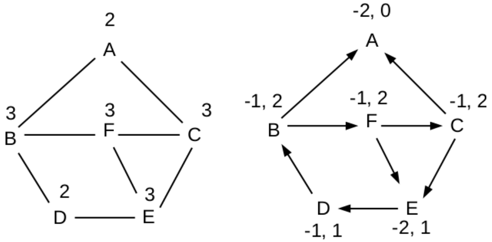
- Se $\forall (u, v) \in \mathcal{A}$, $(u, v) = (v, u)$, então as setas são omitidas na representação gráfica e o grafo é dito não orientado.



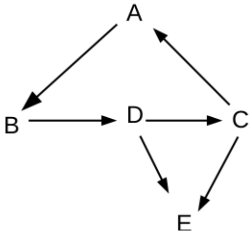
- Um grafo $G = (\mathcal{N}, \mathcal{A}, w)$ é dito ponderado quando exite uma função $w : \mathcal{A} \rightarrow \Re$ que associa um peso $w(u, v)$ a cada arco $(u, v) \in \mathcal{A}$ do grafo.
- Um grafo G não orientado só pode ser ponderado se $w(u, v) = w(v, u)$. Caso contrário, G é dito orientado (*digraph*/*directed graph*) e ponderado.

Grau de um nó

- O grau de saída (positivo) de um nó $u \in \mathcal{N}$ é o número de arcos que partem dele para os nós adjacentes (i.e., $|\mathcal{A}(u)|$).
- O grau de entrada (negativo) de um nó é o número de arcos que chegam nele.
- Em um grafo não orientado, o grau de um nó é o número de arcos com extremidade nele.
- No grafo orientado, a soma dos graus de entrada e saída dos nós é zero.



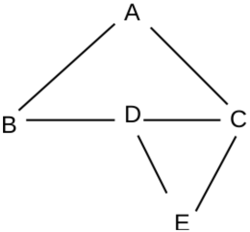
Podemos representar um grafo por uma matriz $|\mathcal{N}| \times |\mathcal{N}|$ de adjacência, na qual indicamos os arcos, incidentes e emergentes, e pesos, dependendo do caso.



	A	B	C	D	E
A		+1	-1		
B	-1			+1	
C	+1			-1	+1
D		-1	+1		+1
E			-1	-1	

A matriz pode ficar esparsa, com $O(|\mathcal{N}|^2)$ de gasto de memória, mas a adição/remoção de arcos é $O(1)$.

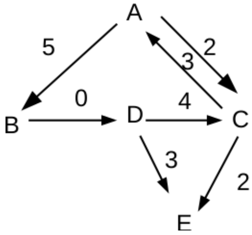
Podemos representar um grafo por uma matriz $|\mathcal{N}| \times |\mathcal{N}|$ de adjacência, na qual indicamos os arcos, incidentes e emergentes, e pesos, dependendo do caso.



	A	B	C	D	E
A		1	1		
B	1			1	
C	1			1	1
D		1	1		1
E			1	1	

A matriz pode ficar esparsa, com $O(|\mathcal{N}|^2)$ de gasto de memória, mas a adição/remoção de arcos é $O(1)$.

Podemos representar um grafo por uma matriz $|\mathcal{N}| \times |\mathcal{N}|$ de adjacência, na qual indicamos os arcos, incidentes e emergentes, e pesos, dependendo do caso.

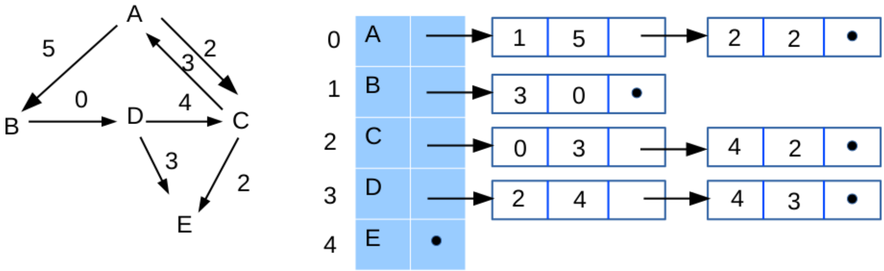


	A	B	C	D	E
A		5	2		
B				0	
C	3				2
D			4		3
E					

A matriz pode ficar esparsa, com $O(|\mathcal{N}|^2)$ de gasto de memória, mas a adição/remoção de arcos é $O(1)$.

Representações de grafo

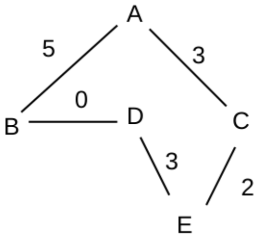
Outra opção de representação é por listas de adjacência, na qual os arcos são armazenados para cada nó, usando um vetor de nós.



O gasto de memória é $O(|\mathcal{N}| + |\mathcal{A}|)$ e o custo de acesso a um arco vai depender do grau do nó. Apesar de $(u, v) = (v, u)$ no grafo não orientado, ambos arcos são armazenados. Então, na prática $(u, v), (v, u) \in \mathcal{A}$.

Representações de grafo

Outra opção de representação é por listas de adjacência, na qual os arcos são armazenados para cada nó, usando um vetor de nós.



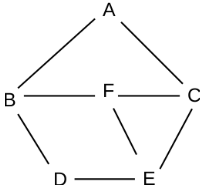
0	A	→	1	5	→	2	3	•
1	B	→	0	5	→	3	0	•
2	C	→	0	3	→	4	2	•
3	D	→	1	0	→	4	3	•
4	E	→	3	3	→	2	2	•

O gasto de memória é $O(|\mathcal{N}| + |\mathcal{A}|)$ e o custo de acesso a um arco vai depender do grau do nó. Apesar de $(u, v) = (v, u)$ no grafo não orientado, ambos arcos são armazenados. Então, na prática $(u, v), (v, u) \in \mathcal{A}$.

Caminhos em grafo

- Um caminho simples $\pi_{u_1 \rightarrow u_n}$ de um nó $u_1 \in \mathcal{N}$ a um nó $u_n \in \mathcal{N}$ é uma sequência $\langle u_1, u_2, \ldots, u_n \rangle$ de nós distintos, onde $(u_i, u_{i+1}) \in \mathcal{A}$, $i = 1, 2, \ldots, n-1$. Note que caminhos arbitrários podem repetir nós.

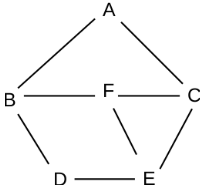
- Um caminho simples $\pi_{u_1 \rightarrow u_n}$ de um nó $u_1 \in \mathcal{N}$ a um nó $u_n \in \mathcal{N}$ é uma sequência $\langle u_1, u_2, \ldots, u_n \rangle$ de nós distintos, onde $(u_i, u_{i+1}) \in \mathcal{A}$, $i = 1, 2, \ldots, n-1$. Note que caminhos arbitrários podem repetir nós.



Por exemplo: $\pi_{A \rightarrow E} = \langle A, B, F, E \rangle$.

Caminhos em grafo

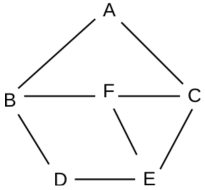
- Um caminho simples $\pi_{u_1 \rightarrow u_n}$ de um nó $u_1 \in \mathcal{N}$ a um nó $u_n \in \mathcal{N}$ é uma sequência $\langle u_1, u_2, \ldots, u_n \rangle$ de nós distintos, onde $(u_i, u_{i+1}) \in \mathcal{A}$, $i = 1, 2, \ldots, n-1$. Note que caminhos arbitrários podem repetir nós.



Por exemplo: $\pi_{A \rightarrow E} = \langle A, B, F, E \rangle$.

- Um caminho simples $\pi_{u_1 \rightarrow u_n}$ pode ser representado por π_u quando nos interessa apenas o seu nó terminal $u = u_n$.

- Um caminho simples $\pi_{u_1 \rightarrow u_n}$ de um nó $u_1 \in \mathcal{N}$ a um nó $u_n \in \mathcal{N}$ é uma sequência $\langle u_1, u_2, \ldots, u_n \rangle$ de nós distintos, onde $(u_i, u_{i+1}) \in \mathcal{A}$, $i = 1, 2, \ldots, n-1$. Note que caminhos arbitrários podem repetir nós.

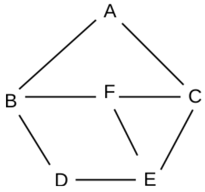


Por exemplo: $\pi_{A \rightarrow E} = \langle A, B, F, E \rangle$.

- Um caminho simples $\pi_{u_1 \rightarrow u_n}$ pode ser representado por π_u quando nos interessa apenas o seu nó terminal $u = u_n$.

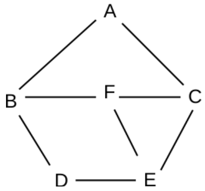
- Um caminho é dito trivial quando $\pi_u = \langle u \rangle$. Por exemplo: $\pi_A = \langle A \rangle$.

Caminhos em grafo



- O comprimento de um caminho é o número de arcos dele. Por exemplo: $c(\langle A, B, F, E\rangle) = 3$ e $c(\langle A\rangle) = 0$.

Caminhos em grafo



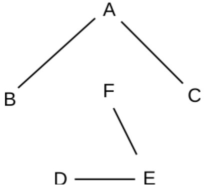
- O comprimento de um caminho é o número de arcos dele. Por exemplo: $c(\langle A, B, F, E\rangle) = 3$ e $c(\langle A\rangle) = 0$.

- A concatenação $\pi_{u_1 \rightarrow u_n} \cdot \langle u_n, u_1\rangle$, eliminando uma instância de u_n, forma um ciclo. Por exemplo: $\langle A, B, F, E, C\rangle \cdot \langle C, A\rangle = \langle A, B, F, E, C, A\rangle$.

Relação de conexidade e componentes conexos

- Um nó v é dito conexo a um nó u quando existe um caminho $\pi_{u \rightarrow v}$.

- Um nó v é dito conexo a um nó u quando existe um caminho $\pi_{u \to v}$.
- Em um grafo não orientado, um componente conexo é um conjunto maximal $\mathcal{C} \subseteq \mathcal{N}$ tal que existe um caminho $\pi_{u \to v}$, $\forall u, v \in \mathcal{C}$. O exemplo abaixo tem dois componentes conexos.



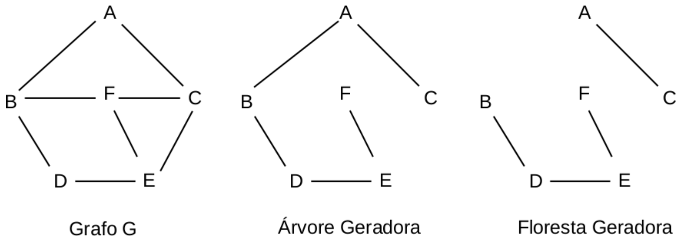
Um grafo é dito conexo quando tem um único componente.

Subgrafo, árvore e floresta

- Um grafo $G' = (\mathcal{N}', \mathcal{A}')$ é subgrafo de um grafo $G = (\mathcal{N}, \mathcal{A})$ se $\mathcal{N}' \subseteq \mathcal{N}$ e $\mathcal{A}' \subseteq \mathcal{A}$.
- Uma árvore G' de G é um subgrafo acíclico.

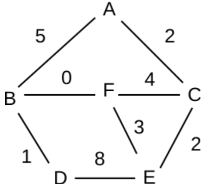
Subgrafo, árvore e floresta

- Um grafo $G' = (\mathcal{N}', \mathcal{A}')$ é subgrafo de um grafo $G = (\mathcal{N}, \mathcal{A})$ se $\mathcal{N}' \subseteq \mathcal{N}$ e $\mathcal{A}' \subseteq \mathcal{A}$.
- Uma árvore G' de G é um subgrafo acíclico.
- Uma árvore G' de G é geradora quando $\mathcal{N}' = \mathcal{N}$.
- Uma floresta G' de G é uma coleção de árvores de G e a floresta é geradora quando $\mathcal{N}' = \mathcal{N}$.



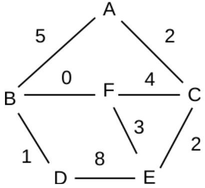
Árvore geradora mínima

- Seja $G = (\mathcal{N}, \mathcal{A}, w)$ um grafo ponderado e não orientado (veja Edmonds' algorithm para o caso orientado).

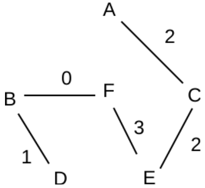


Árvore geradora mínima

- Seja $G = (\mathcal{N}, \mathcal{A}, w)$ um grafo ponderado e não orientado (veja Edmonds' algorithm para o caso orientado).



- Uma árvore geradora mínima (*minimum-spanning tree*) é uma árvore geradora $G' = (\mathcal{N}, \mathcal{A}')$ de G tal que $\sum_{\forall(u,v)\in\mathcal{A}'}\{w(u,v)\}$ é mínima (Algoritmo de Prim).



Percursos em grafo

Dois percursos básicos em grafo são:

- O percurso em largura (*breadth-first search*), que visita os nós conexos a um dado nó inicial, u_1, em ordem crescente de comprimento de caminho.

- O percurso em profundidade (*depth-first search*), que visita os nós conexos a um nó inicial, u_1, visitando em profundidade o maior número de nós possível a partir de cada adjacente de u_1.

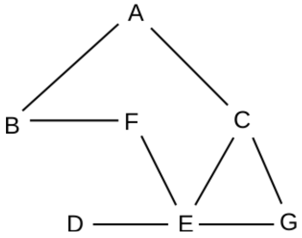
Percursos em grafo

Dois percursos básicos em grafo são:

- O percurso em largura (*breadth-first search*), que visita os nós conexos a um dado nó inicial, u_1, em ordem crescente de comprimento de caminho.

- O percurso em profundidade (*depth-first search*), que visita os nós conexos a um nó inicial, u_1, visitando em profundidade o maior número de nós possível a partir de cada adjacente de u_1.

Por exemplo: Iniciando no nó A.

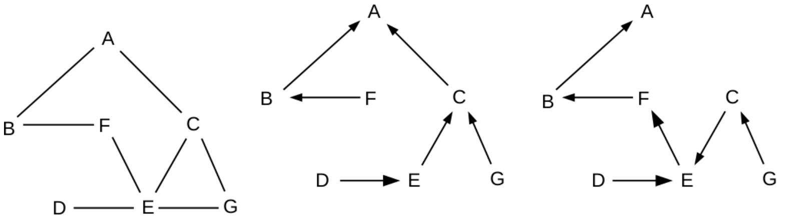


Largura: A, B, C, F, E, G, D

Profundidade: A, B, F, E, C, G, D

Percursos em grafo e mapa de predecessores

- Percursos em grafo atingem todos os nós v conexos ao nó u_1 por um caminho $\pi_{u_1 \rightarrow v}$, gerando uma árvore de caminhos com início em u_1 (nó raiz).
- Esta árvore é normalmente armazenada em um mapa de predecessores P: função acíclica que associa $P(v) = u$, quando $\pi_{u_1 \rightarrow v} = \pi_{u_1 \rightarrow u} \cdot \langle u, v \rangle$, e $P(v) = nil \notin \mathcal{N}$, quando $v = u_1$.
- Note que todos os caminhos com raiz u_1 são armazenados no mapa de predecessores do fim para o início.



Largura: A, B, C, F, E, G, D

Profundidade: A, B, F, E, C, G, D

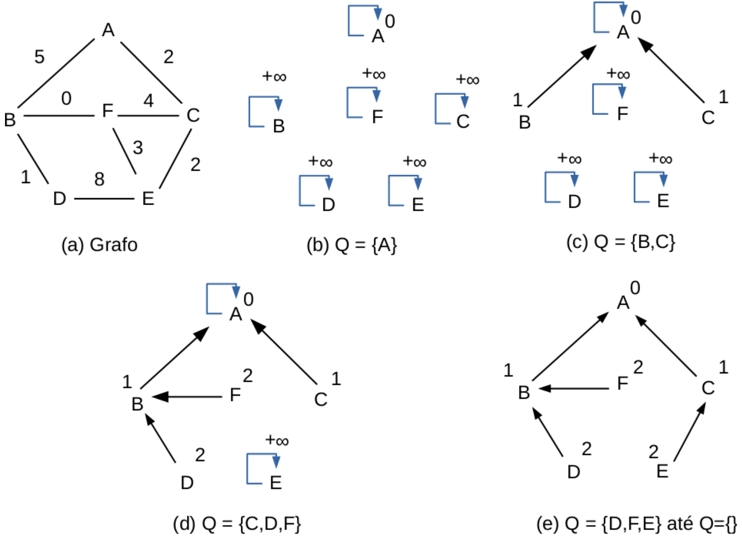
Algoritmo de busca em largura

- A busca em largura insere os nós visitados, iniciando pela raiz, em uma fila Q (*first-in-first-out*).
- Portanto, os nós são removidos em ordem crescente de comprimento de caminho partindo da raiz u_1.
- A entrada do algoritmo é $G = (\mathcal{N}, \mathcal{A}, w)$ e o nó inicial u_1.
- A saída é um mapa C de comprimento de caminho e o mapa de predecessores P.
- A ordem de visitação dos nós é a ordem de saída de Q.
- A variável auxiliar $cor(u)$ indica o estado de um nó em relação a Q: *branco* quando u nunca foi inserido em Q, *cinza* quando u está em Q, e *preto* quando u já foi removido de Q.

Algoritmo de Busca em Largura

1. Para todo $u \in \mathcal{N}$ faça
2. $\quad C(u) \leftarrow +\infty$, $cor(u) \leftarrow branco$, e $P(u) \leftarrow nil$.
3. $\quad$ Se $u = u_1$ então
4. $\quad\quad C(u) \leftarrow 0$, insere u em Q, e $cor(u) \leftarrow cinza$.
5. Enquanto $Q \neq \emptyset$ faça
6. $\quad$ Remove u de Q e $cor(u) \leftarrow preto$.
7. $\quad$ Para todo $v \in \mathcal{A}(u)$, tal que $cor(v) = branco$ faça
8. $\quad\quad P(v) \leftarrow u$ e $C(v) \leftarrow C(u) + 1$.
9. $\quad\quad$ Insere v em Q e $cor(v) \leftarrow cinza$.

Algoritmo de Busca em Largura



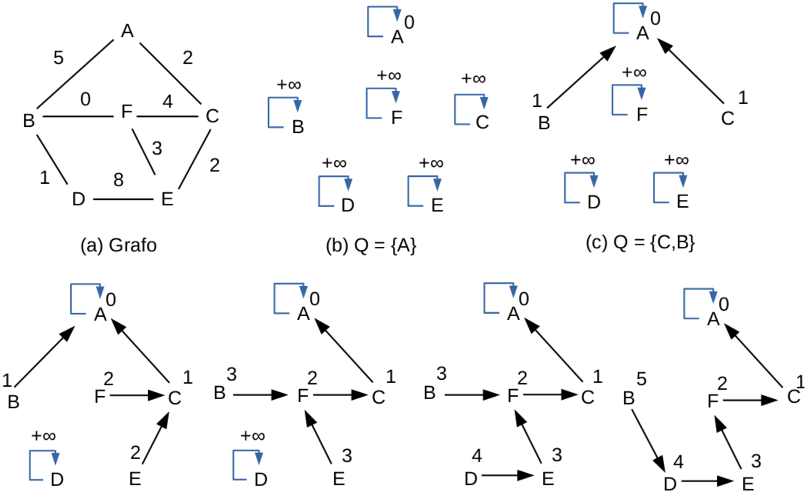
Algoritmo de busca em profundidade

- A busca em profundidade insere os nós visitados, iniciando pela raiz, em uma pilha Q (*last-in-first-out*).
- Os nós são visitados buscando sempre alcançar todos os nós conexos a cada adjacente da raiz u_1 por vez.
- A entrada do algoritmo é $G = (\mathcal{N}, \mathcal{A}, w)$ e o nó inicial u_1.
- A saída é um mapa C de comprimento de caminho e o mapa de predecessores P.
- A ordem de visitação dos nós é a ordem de saída de Q.
- A variável auxiliar $cor(u)$ indica o estado de um nó em relação a Q: *branco* quando u nunca foi inserido em Q, *cinza* quando u está em Q, e *preto* quando u já foi removido de Q.

Algoritmo de Busca em Profundidade

1. Para todo $u \in \mathcal{N}$ faça
2. $\quad C(u) \leftarrow +\infty$, $cor(u) \leftarrow branco$, e $P(u) \leftarrow nil$.
3. $\quad$ Se $u = u_1$ então
4. $\quad\quad C(u) \leftarrow 0$, empilha u em Q, e $cor(u) \leftarrow cinza$.
5. Enquanto $Q \neq \emptyset$ faça
6. $\quad$ Desempilha u de Q e $cor(u) \leftarrow preto$.
7. $\quad$ Para todo $v \in \mathcal{A}(u)$, tal que $cor(v) \neq preto$ faça
8. $\quad\quad P(v) \leftarrow u$ e $C(v) \leftarrow C(u) + 1$.
9. $\quad\quad$ Se $cor(u) = branco$, então empilha v em Q e $cor(v) \leftarrow cinza$.

Algoritmo de Busca em Profundidade



(a) Grafo (b) Q = {A} (c) Q = {C,B}

(c) Q = {F,E,B} (d) Q = {E,B} (e) Q = {D,B} (f) Q = {B} até Q={}

Função de custo de caminho

- Seja c uma função que atribui um valor de custo para qualquer caminho no grafo $G = (\mathcal{N}, \mathcal{A}, w)$ com relação a um dado nó inicial u_1.

Função de custo de caminho

- Seja c uma função que atribui um valor de custo para qualquer caminho no grafo $G = (\mathcal{N}, \mathcal{A}, w)$ com relação a um dado nó inicial u_1. Por exemplo,

$$\begin{aligned} c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v = u_1, \\ +\infty & \text{no caso contrário,} \end{cases} \\ c(\pi_u \cdot \langle u, v \rangle) &= c(\pi_u) + w(u, v), \end{aligned}$$

onde $w(u, v) \geq 0$. Para caminhos simples,

$$c(\langle u_1, u_2, \ldots, u_n = v \rangle) = \sum_{i=1}^{n} w(u_i, u_{i+1}).$$

Função de custo de caminho

- Seja c uma função que atribui um valor de custo para qualquer caminho no grafo $G = (\mathcal{N}, \mathcal{A}, w)$ com relação a um dado nó inicial u_1. Por exemplo,

$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v = u_1, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= c(\pi_u) + w(u, v),
\end{aligned}
$$

  onde $w(u, v) \geq 0$. Para caminhos simples,

$$
c(\langle u_1, u_2, \ldots, u_n = v \rangle) = \sum_{i=1}^{n} w(u_i, u_{i+1}).
$$

- Sendo $\Pi(u_1, v)$ o conjunto de todos os possíveis caminhos com início em u_1 e término em v, um caminho $\pi_{u_1 \rightarrow v}$ é de custo mínimo se $c(\pi_{u_1 \rightarrow v}) \leq c(\tau_{u_1 \rightarrow v})$ para qualquer $\tau_{u_1 \rightarrow v} \in \Pi(u_1, v)$.

Mapa de custo mínimo

- O percurso em largura pode ser visto como caso particular da minimização do mapa de custo $C(v)$, $\forall v \in \mathcal{N}$,

$$C(v) \quad = \quad \min_{\forall \pi_{u_1 \rightarrow v} \in \Pi(u_1, v)} \{c(\pi_{u_1 \rightarrow v})\}.$$

quando $w(u, v) = 1$, $\forall (u, v) \in \mathcal{A}$.

- No caso geral, os nós são visitados em ordem não-decrescente de custo de caminho (Algoritmo de Dijkstra).

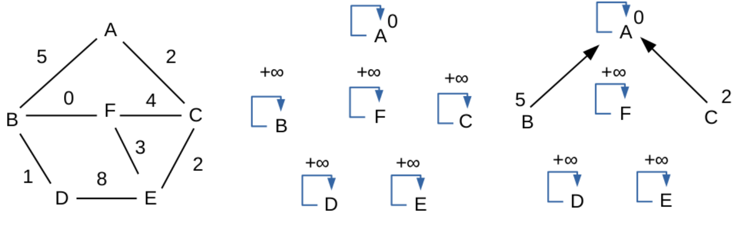
- O algoritmo de Dijkstra calcula no mapa de predecessores todos os caminhos de custo mínimo iniciados em u_1.

Algoritmo de Dijkstra

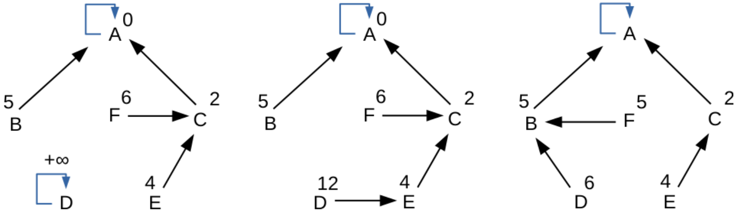
- O algoritmo de Dijkstra requer uma fila de prioridades Q (heap binário).
- Os nós são removidos na ordem não-decrescente de custo mínimo de caminho partindo da raiz u_1.
- A entrada do algoritmo é $G = (\mathcal{N}, \mathcal{A}, w)$ e o nó inicial u_1.
- A saída é o mapa C de custo mínimo de caminho e o mapa de predecessores P.
- A ordem de visitação dos nós é a ordem de saída de Q.
- Variáveis auxiliares: $cor(u)$, como antes, e custo tmp.

Algoritmo de Dijsktra

1. Para todo $u \in \mathcal{N}$ faça
2. $\quad C(u) \leftarrow +\infty$, $cor(u) \leftarrow branco$, e $P(u) \leftarrow nil$.
3. $\quad$ Se $u = u_1$ então
4. $\quad\quad C(u) \leftarrow 0$, insere u em Q, e $cor(u) \leftarrow cinza$.
5. Enquanto $Q \neq \emptyset$ faça
6. $\quad$ Remove u de Q, tal que $C(u)$ é mínimo, e $cor(u) \leftarrow preto$.
7. $\quad$ Para todo $v \in \mathcal{A}(u)$, tal que $cor(v) \neq preto$ faça
8. $\quad\quad tmp \leftarrow C(u) + w(u, v)$.
9. $\quad\quad$ Se $tmp < C(v)$ então
10. $\quad\quad\quad P(v) \leftarrow u$ e $C(v) \leftarrow tmp$.
11. $\quad\quad\quad$ Se $cor(v) = cinza$ então atualiza posição de v em Q e caso contrário, insere v em Q e $cor(v) \leftarrow cinza$.



(a) Grafo (b) Q = {A} (c) Q = {C,B}



(d) Q = {E,B,F} (e) Q = {B,F,D} (e) Q = {F,D} até Q={}

- As linhas de 1 a 4 essencialmente inicializam todos os caminhos como triviais, com custo 0 apenas para $\langle u_1 \rangle$.

Observações sobre o algoritmo de Dijkstra

- As linhas de 1 a 4 essencialmente inicializam todos os caminhos como triviais, com custo 0 apenas para $\langle u_1 \rangle$.
- Quanto u é removido de Q na linha 6, o caminho ótimo de u_1 até u está calculado em P.

Observações sobre o algoritmo de Dijkstra

- As linhas de 1 a 4 essencialmente inicializam todos os caminhos como triviais, com custo 0 apenas para $\langle u_1 \rangle$.
- Quanto u é removido de Q na linha 6, o caminho ótimo de u_1 até u está calculado em P.
- As linhas 8-10 essencialmente verificam se o custo do caminho $\pi_u \cdot \langle u, v \rangle$ é menor do que o custo do caminho atual π_v. Se for, então $\pi_v \leftarrow \pi_u \cdot \langle u, v \rangle$.

Observações sobre o algoritmo de Dijkstra

- As linhas de 1 a 4 essencialmente inicializam todos os caminhos como triviais, com custo 0 apenas para $\langle u_1 \rangle$.
- Quanto u é removido de Q na linha 6, o caminho ótimo de u_1 até u está calculado em P.
- As linhas 8-10 essencialmente verificam se o custo do caminho $\pi_u \cdot \langle u, v \rangle$ é menor do que o custo do caminho atual π_v. Se for, então $\pi_v \leftarrow \pi_u \cdot \langle u, v \rangle$.
- Cada nó entra e sai de Q uma única vez. A complexidade é $O(|\mathcal{N}| \log |\mathcal{N}| + |\mathcal{A}|)$. Se o grafo for esparso, $|\mathcal{A}| \ll |\mathcal{N}|^2$, a complexidade é $O(|\mathcal{N}| \log |\mathcal{N}|)$.

Observações sobre o algoritmo de Dijkstra

- As linhas de 1 a 4 essencialmente inicializam todos os caminhos como triviais, com custo 0 apenas para $\langle u_1 \rangle$.
- Quanto u é removido de Q na linha 6, o caminho ótimo de u_1 até u está calculado em P.
- As linhas 8-10 essencialmente verificam se o custo do caminho $\pi_u \cdot \langle u, v \rangle$ é menor do que o custo do caminho atual π_v. Se for, então $\pi_v \leftarrow \pi_u \cdot \langle u, v \rangle$.
- Cada nó entra e sai de Q uma única vez. A complexidade é $O(|\mathcal{N}| \log |\mathcal{N}| + |\mathcal{A}|)$. Se o grafo for esparso, $|\mathcal{A}| \ll |\mathcal{N}|^2$, a complexidade é $O(|\mathcal{N}| \log |\mathcal{N}|)$.
- É possível reduzir esta complexidade para $O(|\mathcal{N}|)$, quando os custos são inteiros e $0 \leq w(u, v) \leq K \in Z$.

- Podemos degenerar o algoritmo de Dijkstra no algoritmo de Prim, para cálculo de uma árvore geradora mínima em um grafo não-orientado.

- Podemos degenerar o algoritmo de Dijkstra no algoritmo de Prim, para cálculo de uma árvore geradora mínima em um grafo não-orientado.
- Basta executar o algoritmo de Dijkstra com função de custo c dada por

$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v = u_1, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= w(u, v).
\end{aligned}
$$

Árvore Geradora Mínima

- Podemos degenerar o algoritmo de Dijkstra no algoritmo de Prim, para cálculo de uma árvore geradora mínima em um grafo não-orientado.
- Basta executar o algoritmo de Dijkstra com função de custo c dada por

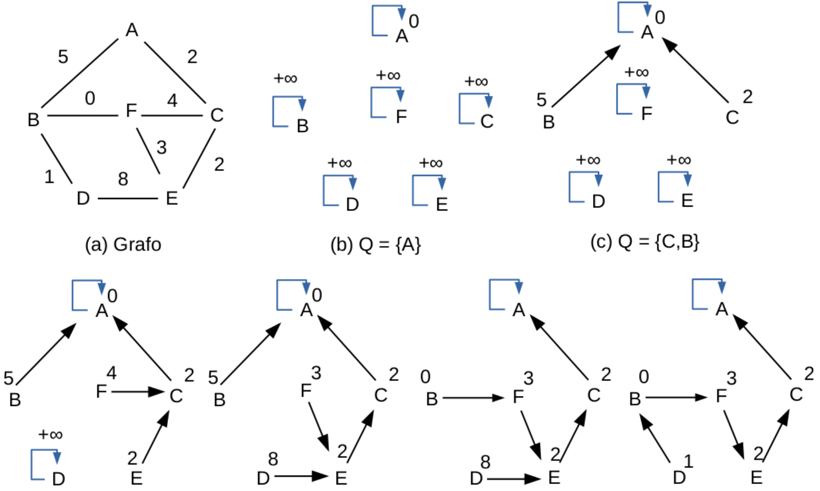
$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v = u_1, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= w(u, v).
\end{aligned}
$$

- Isso não gera uma árvore de caminhos ótimos de acordo com c, mas gera uma árvore geradora de peso mínimo se o grafo for conexo.

Algoritmo de Prim

1. Para todo $u \in \mathcal{N}$ faça
2. $\quad C(u) \leftarrow +\infty$, $cor(u) \leftarrow branco$, e $P(u) \leftarrow nil$.
3. $\quad$ Se $u = u_1$ então
4. $\qquad C(u) \leftarrow 0$, insere u em Q, e $cor(u) \leftarrow cinza$.
5. Enquanto $Q \neq \emptyset$ faça
6. $\quad$ Remove u de Q, tal que $C(u)$ é mínimo, e $cor(u) \leftarrow preto$.
7. $\quad$ Para todo $v \in \mathcal{A}(u)$, tal que $cor(v) \neq preto$ faça
8. $\qquad tmp \leftarrow w(u, v)$.
9. $\qquad$ Se $tmp < C(v)$ então
10. $\qquad\quad P(v) \leftarrow u$ e $C(v) \leftarrow tmp$.
11. $\qquad\quad$ Se $cor(v) = cinza$ então atualiza posição de v em Q e caso contrário, insere v em Q e $cor(v) \leftarrow cinza$.

Algoritmo de Prim



(a) Grafo (b) Q = {A} (c) Q = {C,B}

(d) Q = {E,F,B} (e) Q = {F,B,D} (e) Q = {B,D} (f) Q = {D} até Q={}

Observações sobre o algoritmo de Prim

- A árvore geradora mínima é na verdade um subgrafo $G' = (\mathcal{N}', \mathcal{A}', w)$ não-orientado de $G = (\mathcal{N}, \mathcal{A}, w)$, onde $\mathcal{N}' \subseteq \mathcal{N}$ contém apenas os nós conexos a u_1.

Observações sobre o algoritmo de Prim

- A árvore geradora mínima é na verdade um subgrafo $G' = (\mathcal{N}', \mathcal{A}', w)$ não-orientado de $G = (\mathcal{N}, \mathcal{A}, w)$, onde $\mathcal{N}' \subseteq \mathcal{N}$ contém apenas os nós conexos a u_1.
- Os arcos em $\mathcal{A}'$ podem ser identificados entre as linhas 6 e 7, com o teste:
  Se $P(u) \neq nil$ então $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{(P(u), u), (u, P(u))\}$.

- A árvore geradora mínima é na verdade um subgrafo $G' = (\mathcal{N}', \mathcal{A}', w)$ não-orientado de $G = (\mathcal{N}, \mathcal{A}, w)$, onde $\mathcal{N}' \subseteq \mathcal{N}$ contém apenas os nós conexos a u_1.
- Os arcos em $\mathcal{A}'$ podem ser identificados entre as linhas 6 e 7, com o teste:
  Se $P(u) \neq nil$ então $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{(P(u), u), (u, P(u))\}$.
- Para gerar uma floresta geradora mínima $G' = (\mathcal{N}, \mathcal{A}', w)$, basta modificar a função de custo c para.

$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v \in \mathcal{S} \subset \mathcal{N}, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= w(u, v),
\end{aligned}
$$

  onde o conjunto $\mathcal{S}$ contém uma raiz arbitrária para cada componente conexo do grafo G e essas raízes são descobertas durante o algoritmo, quando o nó u removido de Q tem $P(u) = nil$.

Algoritmo de Floresta Geradora Mínima

1. Para todo $u \in \mathcal{N}$ faça
2. $\quad C(u) \leftarrow +\infty$, $P(u) \leftarrow nil$, insere u em Q e $cor(u) \leftarrow cinza$.
3. Enquanto $Q \neq \emptyset$ faça
4. $\quad$ Remove u de Q, tal que $C(u)$ é mínimo, e $cor(u) \leftarrow preto$.
5. $\quad$ Se $P(u) = nil$ então $C(u) \leftarrow 0$. Caso contrário, $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{(u, P(u)), (P(u), u)\}$.
6. $\quad$ Para todo $v \in \mathcal{A}(u)$, tal que $cor(v) \neq preto$ faça
7. $\quad\quad tmp \leftarrow w(u, v)$.
8. $\quad\quad$ Se $tmp < C(v)$ então
9. $\quad\quad\quad P(v) \leftarrow u$ e $C(v) \leftarrow tmp$.
10. $\quad\quad\quad$ Atualiza posição de v em Q.

Mais observações sobre o algoritmo de Dijkstra

- Para um dado conjunto $\mathcal{S} \subset \mathcal{N}$ de nós raízes, o algoritmo de Dijkstra pode ser executado com função c de custo de caminho

$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v \in \mathcal{S}, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= c(\pi_u) + w(u, v).
\end{aligned}
$$

Mais observações sobre o algoritmo de Dijkstra

- Para um dado conjunto $\mathcal{S} \subset \mathcal{N}$ de nós raízes, o algoritmo de Dijkstra pode ser executado com função c de custo de caminho

$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v \in \mathcal{S}, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= c(\pi_u) + w(u, v).
\end{aligned}
$$

- Neste caso, as raízes em $\mathcal{S}$ competem pelos nós mais fortemente conexos a elas, gerando uma árvore de caminhos de custo mínimo para cada raiz (floresta de caminhos ótimos).

Mais observações sobre o algoritmo de Dijkstra

- Para um dado conjunto $\mathcal{S} \subset \mathcal{N}$ de nós raízes, o algoritmo de Dijkstra pode ser executado com função c de custo de caminho

$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v \in \mathcal{S}, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= c(\pi_u) + w(u, v).
\end{aligned}
$$

- Neste caso, as raízes em $\mathcal{S}$ competem pelos nós mais fortemente conexos a elas, gerando uma árvore de caminhos de custo mínimo para cada raiz (floresta de caminhos ótimos).
- A ideia se aplica a outras funções de custo, tais como

$$
\begin{aligned}
c(\langle v \rangle) &= \begin{cases} 0 & \text{se } v \in \mathcal{S}, \\ +\infty & \text{no caso contrário,} \end{cases} \\
c(\pi_u \cdot \langle u, v \rangle) &= \max\{c(\pi_u), w(u, v)\},
\end{aligned}
$$

para condições mais gerais, em Falcão, IEEE TPAMI, 2004.