Simulado 3 - CIC110

Algoritmo 1(G, w, r)

▷ Entrada: Um grafo G, uma função de distância w e um vértice raiz$r$; Encontra o caminho mínimo entre r e todos os vértices

$\overline{S}$ é o conjunto de vértices com caminho mais curto ainda a descobrir

`dist`$[x]$: comprimento do menor caminho de r até x, dado S

1. `dist`$[r] \leftarrow 0$; $\overline{S} \leftarrow V$;
2. **para** todo $v \in V - \{r\}$ **faça** `dist`$[v] \leftarrow \infty$;
3. `pred`$[r] \leftarrow$ nulo;
4. $A \leftarrow \{\}$; ▷ inicializa o conjunto de arestas da ACM(r)
5. **enquanto** ($\overline{S}$ não for vazio) **faça**
6. Remover de $\overline{S}$ o vértice u com o menor valor em `dist`;
7. **se** `pred`$[u] \neq$ `nulo`, $A \leftarrow A \cup \{($`pred`$[u], u)\}$;
   ▷ atualiza o valor de `dist`[.] para vértices adjacentes a u
8. **para** todo $v \in$ Adj$[u]$ **faça**
9. **se** $(v \in \overline{S})$ **e** (`dist`$[v] >$ `dist`$[u] + w[u, v])$ **então**
10. { `dist`$[v] \leftarrow$ `dist`$[u] + w[u, v]$; `pred`$[v] \leftarrow u$; }
11. **retorne** (A,`dist`).

Algoritmo 2(G, d, r)

▷ Entrada: Um grafo G, uma função de distância d e um vértice raiz$r$; Encontra o caminho mínimo entre r e todos os vértices

1. **para** todo $v \in V - \{r\}$ **faça** $a[v] \leftarrow$ `dist`$[v] \leftarrow \infty$;
2. $a[r] \leftarrow$ `dist`$[r] \leftarrow 0$; `continua` $\leftarrow$ `verdadeiro`; $k \leftarrow 1$;
3. **enquanto** $(k \leq |V|)$ **e** (`continua`) **faça**
4. **para** todo $u \in V$ **faça**
5. **para** todo $v \in$ Adj$[u]$ **faça**
6. **se** $(a[v] >$ `dist`$[u] + d[u, v])$ **então**
7. { $a[v] \leftarrow$ `dist`$[u] + d[u, v]$; `pred`$[v] \leftarrow u$; }
8. `continua` $\leftarrow$ `falso`;
9. **para** todo $v \in V$ **faça**
10. **se** $(a[v] \neq$ `dist`$[v])$ **então** `continua` $\leftarrow$ `verdadeiro`;
11. `dist`$[v] \leftarrow a[v]$;
12. $k \leftarrow k + 1$;
13. `CICLO` $\leftarrow$ ((`continua`) **e** $(k = |V| + 1))$;
14. **retorne**(`CICLO`, `dist`, `pred`).

Algoritmo 3(G, d)

▷ Entrada: Um grafo G, uma função de distância d, n é o número de vértices

1. inicializar `dist` e `pred`: (veja mais abaixo)
2. **para** k **de** 1 **até** n **faça**
3. **para** i **de** 1 **até** n **faça**
4. **para** j **de** 1 **até** n **faça**
5. **se** `dist`$[i, j] >$ `dist`$[i, k] +$ `dist`$[k, j]$ **então**
6. `dist`$[i, j] \leftarrow$ `dist`$[i, k] +$ `dist`$[k, j]$
7. `pred`$[i, j] \leftarrow$ `pred`$[k, j]$
8. **retorne** (`dist`, `pred`).

Algoritmo 4(G, w, r)

▷ r é um vértice escolhido para ser *raiz* da AGM

Q é o conjunto de vértices a incluir na árvore, `dist`$[x]$: peso da aresta mais leve ligando x a componente de r

1. `dist`$[r] \leftarrow 0$; $Q \leftarrow V$;
2. **para** todo $v \in V - \{r\}$ **faça** `dist`$[v] \leftarrow \infty$;
3. `pred`$[r] \leftarrow$ nulo;
4. $A \leftarrow \{\}$; ▷ inicializa o conjunto de arestas da AGM
5. $W \leftarrow 0$; ▷ inicializa o peso da AGM
6. **enquanto** (Q não for vazio) **faça**
7. Remover de Q o vértice u com o menor valor em `dist`;
8. $W \leftarrow W +$ `dist`$[u]$;
9. **se** `pred`$[u] \neq$ `nulo`, $A \leftarrow A \cup \{($`pred`$[u], u)\}$;
   ▷ atualiza o valor de `dist`[.] para vértices adjacentes a u
10. **para** todo $v \in$ Adj$[u]$ **faça**
11. **se** $(v \in Q)$ **e** (`dist`$[v] > w[u, v]$) **então**
12. { `dist`$[v] \leftarrow w[u, v]$; `pred`$[v] \leftarrow u$; }
13. **retorne** (A,W).

Algoritmo 5(G, w)

1. $W \leftarrow 0$; $A \leftarrow \varnothing$; ▷ inicializações
   ▷ Iniciar G_A com $|V|$ árvores com um vértice cada
2. **para** todo $v \in V$ **faça** $a[v] \leftarrow \{v\}$; ▷ $a[v]$ é identificado com v
   ▷ lista de arestas em ordem não decrescente de peso
3. $L \leftarrow$ **ordene**(E,w);
4. $k \leftarrow 0$; ▷ conta arestas **aceitas**
5. **enquanto** $k \neq |V| - 1$ **faça**
6. **remove**$(L, (u, v))$; ▷ tomar primeira aresta em L
   ▷ acha componentes de u e v
7. $a[u] \leftarrow$ **encontrar**(u); $a[v] \leftarrow$ **encontrar**(v);
8. **se** $a[u] \neq a[v]$ **então** ▷ aceita (u, v) se não forma ciclo com A
9. $A \leftarrow A \cup \{(u, v)\}$; $W \leftarrow W + w(u, v)$;
11. $k \leftarrow k + 1$;
12. **unir**$(a[u], a[v])$; ▷ unir componentes
13. **retorne** (A, W).

1. Considere o Algoritmo 1, suponha que é implementado como um vetor linear, de modo que a operação da linha 6 tenha complexidade $O(|V|)$ e a atualização da linha 11 é $O(1)$. Qual a complexidade desse algoritmo? Demonstre sua afirmação.

2. Considere o Algoritmo 1, suponha que *dist* é agora implementado como um heap, de modo que a operação da linha 6 tenha complexidade $O(1)$ e a atualização da linha 11 é $O(\lg |V|)$. Qual a complexidade desse algoritmo? Demonstre sua afirmação.

3. Qual a complexidade do algoritmo 2? Demonstre sua afirmação.

4. Qual a complexidade do algoritmo 3? Demonstre sua afirmação.

5. Suponha que você queira encontrar a distância entre um determinado par de vértices (u, v) e você sabe que no grafo de entrada $|E| = O(|V|)$, e que todos os pesos são positivos. Qual algoritmo com qual estrutura de dados seria mais eficiente? E por que?

6. Suponha que você queira encontrar a distância entre um determinado par de vértices (u, v) e você sabe que no grafo de entrada $|E| = \Omega(|V|^2)$, e que todos os pesos são positivos. Qual algoritmo com qual estrutura de dados seria mais eficiente? E por que?

7. Suponha que você queira encontrar a distância entre um determinado par de vértices (u, v) e você sabe que no grafo de entrada $|E| = O(|V|)$, porém existem arestas de pesos negativos. Qual algoritmo com qual estrutura de dados seria mais eficiente? E por que?

8. Suponha que você queira encontrar a distância entre todos os pares de vértices e você sabe que no grafo de entrada $|E| = \Omega(|V|^2)$, e que todos os pesos são positivos.

9. Suponha que você queira encontrar a distância entre todos os pares de vértices e você sabe que no grafo de entrada $|E| = O(|V|)$, e que todos os pesos são positivos. Qual algoritmo com qual estrutura de dados seria mais eficiente? E por que?

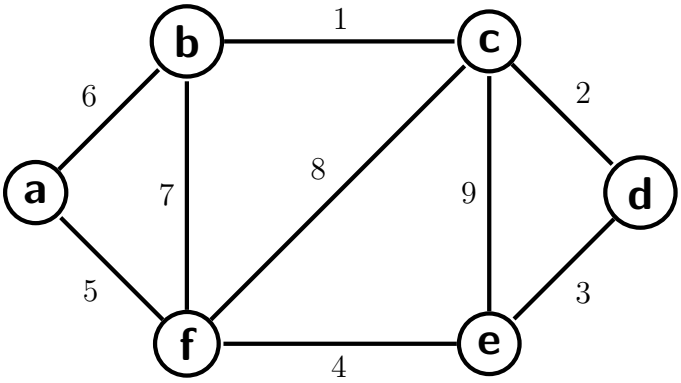
10. Qual a complexidade do algoritmo 4 se Q for implementado por um vetor simples? Justifique.

11. Qual a complexidade do algoritmo 4 se Q for implementado por um heap comum (e não o heap de Fibonacci)? Justifique.

12. Qual a complexidade na prática do algoritmo 5 usando a estrutura UNION-FIND, com balanceamento mas sem compressão de caminhos? Justifique.

13. Qual a complexidade na prática do algoritmo 5 usando a estrutura UNION-FIND, com balanceamento e com compressão de caminhos? Justifique.

Considere o seguinte grafo:



14. Quais arestas fazem parte de uma árvore geradora mínima?

15. Quais arestas fazem parte de uma árvore de caminhos mínimos com raiz em a?