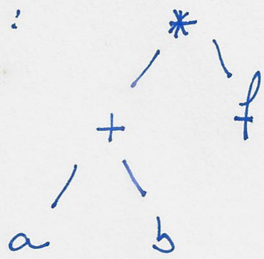
Gabarito da 1ª V.A. Realizada em 23/02/2023

Disciplina de Fundamentos de Problemas Computacionais II

Prof. Tiago A. G. Ferreira

1) Árvore apresentada:



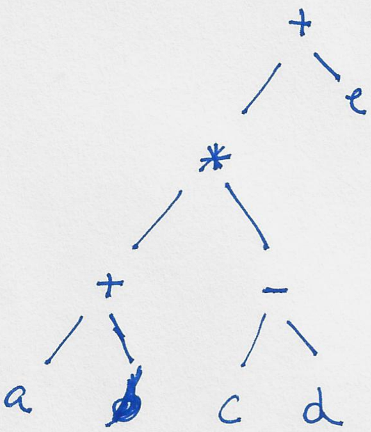
"Em ordem" $\longrightarrow$ $(a+b)*f$

Árvore (operando) nodo

Percurso em ordem: Inorder-tree-walk $(T, x)$

```
if x ≠ NIL
    Inorder-tree-walk(x.left)
    print x.key
    Inorder-tree-walk(x.right)
```

A árvore:



quando percorrida "Em ordem" segundo o algoritmo Inorder-tree-walk gera a expressão

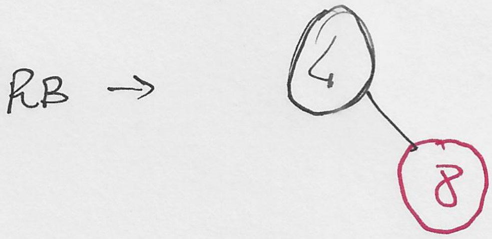
$$((a+b)*(c-d))+e$$

2) Sequência {4, 8, 7, 3}

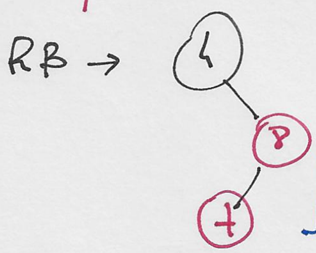
• inicialmente a árvore vermelho-preto RB é vazia e recebe o primeiro elemento da sequência.

O nodo 4 é criado vermelho, mas ao ser inserido na árvore, da regra que a raiz é preta, fica preto.


• O nodo 8 é criado vermelho e inserido na árvore.

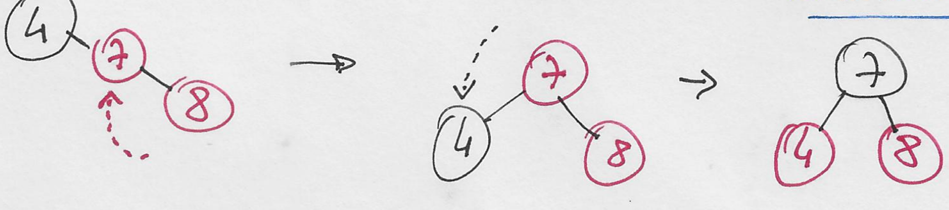



• O nodo 7 é criado vermelho e inserido na árvore



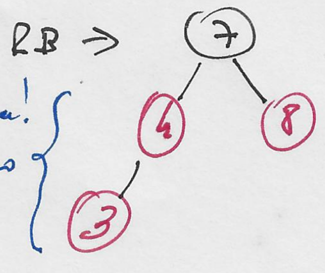

} Nodo vermelho com filho vermelho! → Rotação dupla a esquerda.

Recoloração



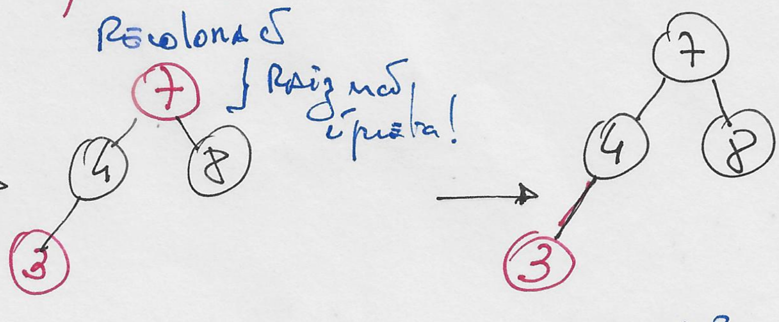
• O nodo 3 é criado vermelho e inserido na árvore.

Problema! vermelho com filho vermelho!



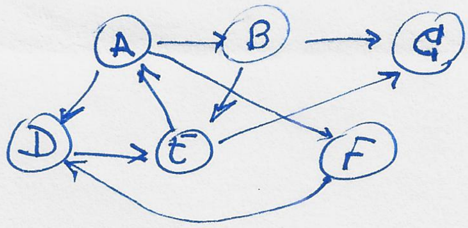

Recoloração

} Raiz não é preta!



Árvore RB final!

3) Dado o grafo:




a) Representações:

– lista adjacente

A → B → D → F /
B → C → E /
C → /
D → E /
E → A → C /
F → D /

– Matriz de Adjacência: (sentido da linha p/ coluna)

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 1 | 0 | 1 |
| B | 0 | 0 | 1 | 0 | 1 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | 0 | 0 | 1 | 0 |
| E | 1 | 0 | 1 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 1 | 0 | 0 |

b) Custo em memória:

– lista adjacente: $\Theta(V+E)$

– Matriz de Adjacência: $\Theta(V^2)$

Em termos práticos, é possível ter um grafo esparso ou denso. Se esparso $E \sim O(V)$ se denso $E \sim O(V^2)$. Logo, se o grafo é esparso, a representação de lista adjacente é menos custosa. Contudo se o grafo é denso, as duas representações tem custo em memória equivalentes. Porem, o manuseio da matriz é menos complexo, logo se o grafo é denso a representação por matriz de adjacência é mais eficiente.

e) Este grafo não pode ser considerado uma árvore ~~[illegible]~~ visto que:

1) G é cíclico; ou

2) $|E| > |V| - 1$; ou

3) O centro deste grafo possui mais de dois vértices.

4)

A) O algoritmo de Dijkstra, visto que este é ótimo e completo para grafos que não contêm arestas com pesos negativos.

B) Dado que a fonte s = "A", e w são os pesos das arestas, o algoritmo de Dijkstra é

```
Dijkstra(G, w, s)
    Initialize-Single-Source(G, s)
    S = ∅
    Q = G.V
    While Q ≠ ∅
        u = Extract-Min(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.adj[u]
            Relax(u, v, w)
```

onde: • Initialize-Single-Source(G, s)

```
    for each vertex v ∈ G.V
        v.d = ∞
        v.π = Nil
    s.d = 0
```

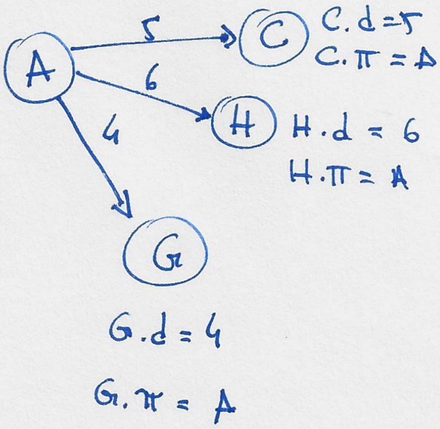
• Relax(u, v, w)

```
    if v.d > u.d + w(u, v)
        v.d = u.d + w(u, v)
        v.π = u
```

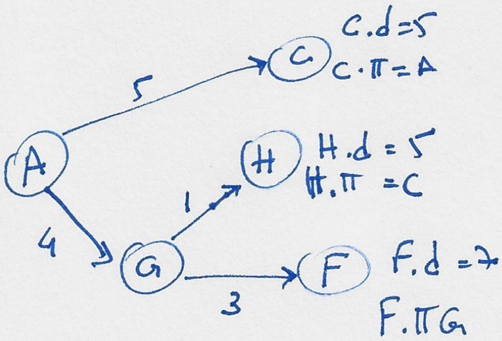
# Aplicando o algoritmo de Dijkstra

$s = A$ (fonte)

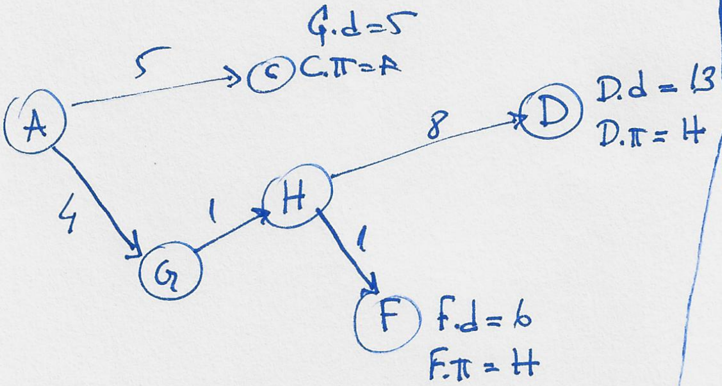
$u = A$



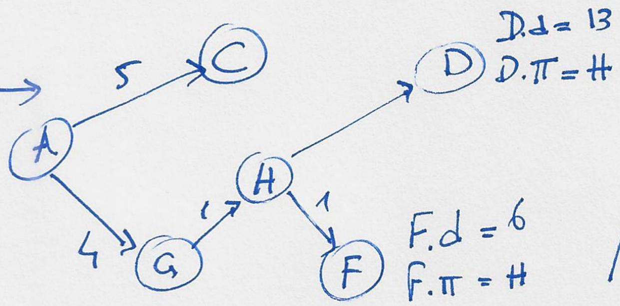

$u = G$



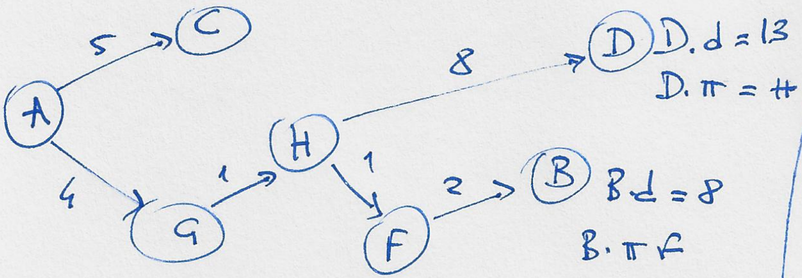

$u = H$




$u = C$

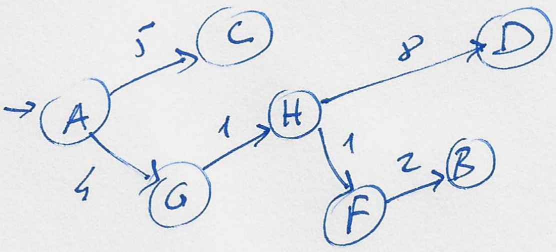



$u = F$




não há mais possibilidades de relaxar

$u = B, u = D$




→ Distância A → B: 8km

Percurso: A → G → H → F → B

c) Sim! Uma árvore geradora mínima contém os menores somatória de pesos que unem todos os nodos em um grafo conexo. Assim, um percurso simples mais curto em um grafo conexo, como é o caso, irá estar contido em alguma árvore geradora mínima.