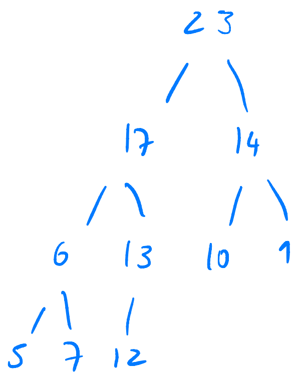
Prática 2

- Heaps
- DFS + Propriedades
- Representação de Grafos

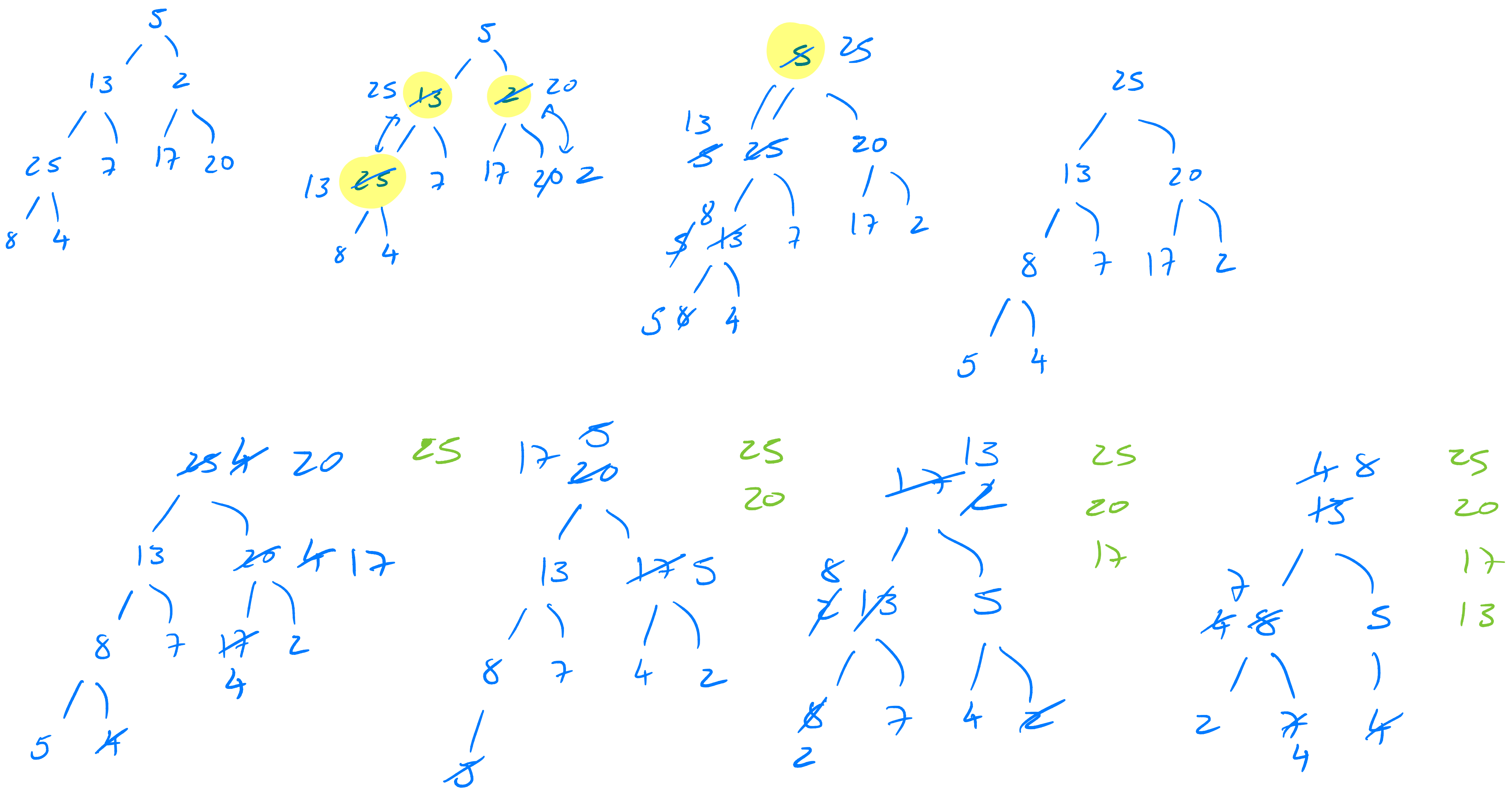
Ex 6.1-6

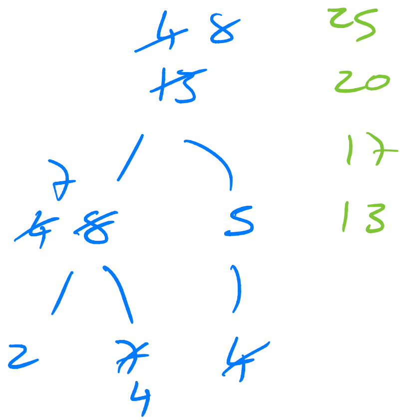
$\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$

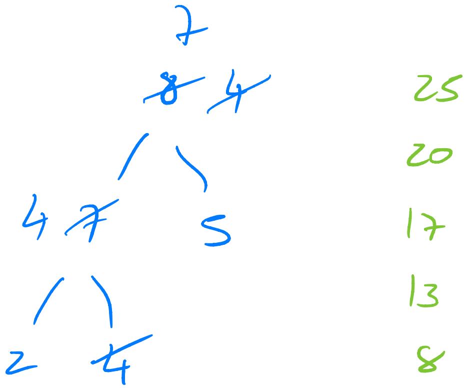


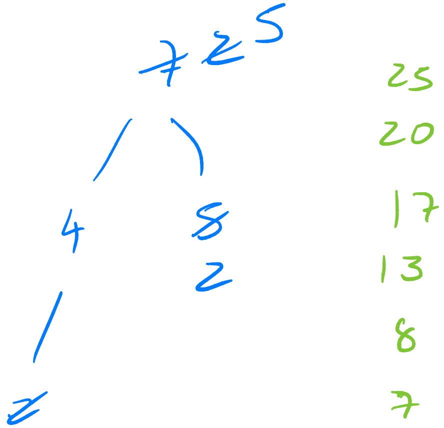
Ex 6.4-1

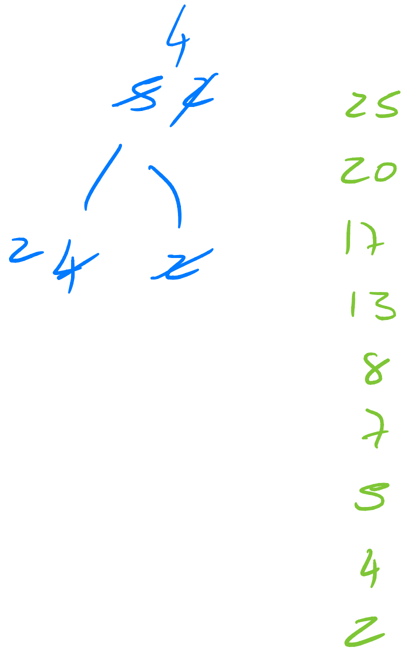
$\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$




25
20
17
13


25
20
17
13
8

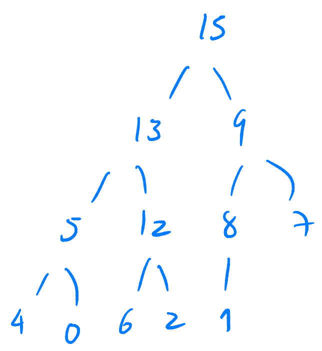

25
20
17
13
8
7

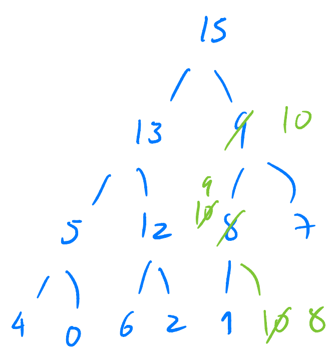

25
20
17
13
8
7
5
4
2

Ex 6.5-2

$\langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$

MaxHeapInsert(10) ...





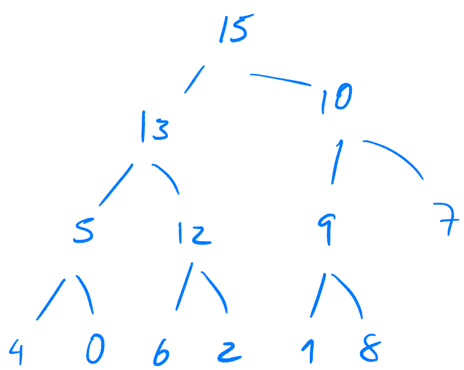
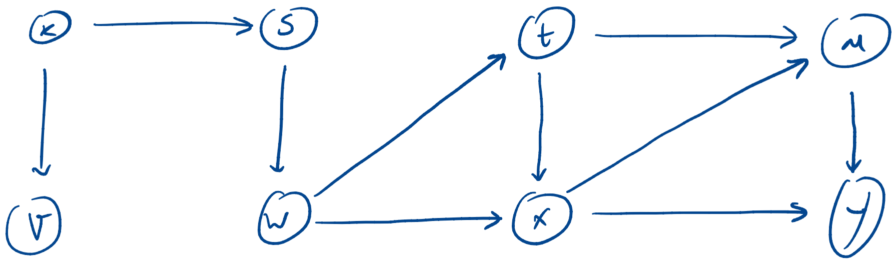
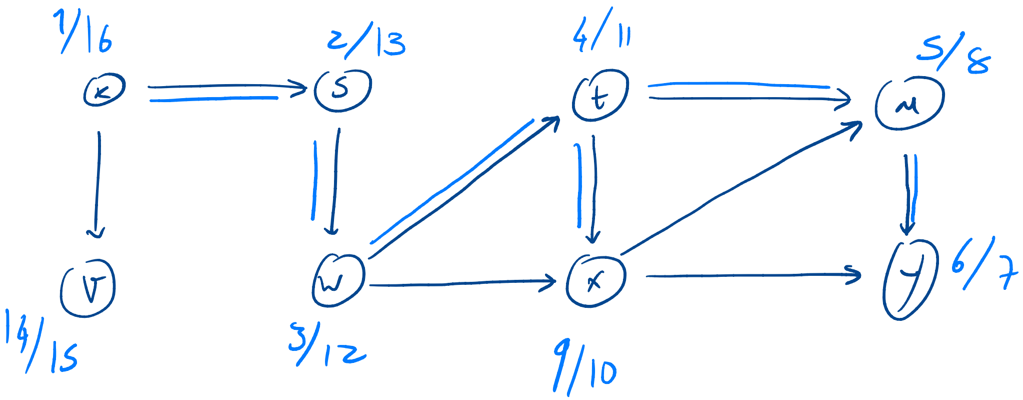


Fig 22.3

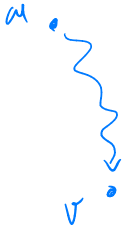


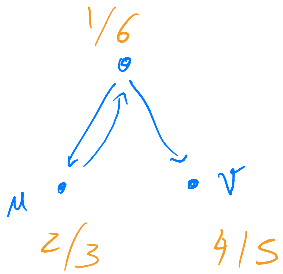


Ex 22.3-8

$\Rightarrow$ Provide a counter example for:

$u \rightsquigarrow v \land d_u < d_v \Rightarrow v$ é descendente de u na floresta DFS





$[_u \quad [_v$

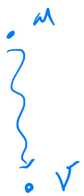
- $[_u \quad]_u \quad [_v \quad]_v \Rightarrow v$ não é descendente de u
- $[_u \quad [_v \quad]_v \quad]_u \Rightarrow v$ é descendente de u

• Queremos provar q̃ é possível termos: $f_u < d_v$

$$[_u \quad]_u \quad [_v \quad]_v$$

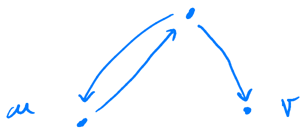
Ex. 22.3-9

Se um grafo dirigido G tem um caminho entre u e v,
então qualquer DFS resulta em $d_v < f_u$

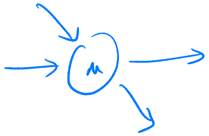


$[_v \quad]_u \Rightarrow$ negar isto

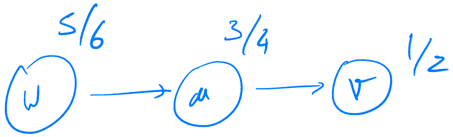
$[_u \quad]_u \quad [_v \quad]_v$



Ex 22.3-11



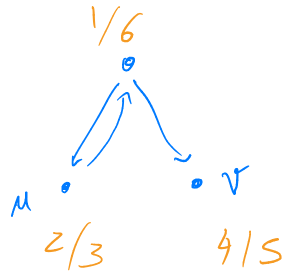
• how can u end up in a DFS tree by itself?



- Para qualquer DFS existe sempre um vértice c/ tempo de fim igual a $2|V|$

T

- Seja $u \in V$ um vértice atingível a partir de todos os vértices do grafo. u é necessariamente o primeiro vértice a ser fechado.



F

- Se $d_v = d_u + 1$, então (u, v) é um arco de árvore

T

- Se $(u, v) \in E$ então temos necessariamente que $d_u < d_v$

F

- Se $f_v < d_u \wedge (u, v) \in E$, então (u, v) é um arco de cruzamento

$[_v \;]_v \; [_u \;]_u$

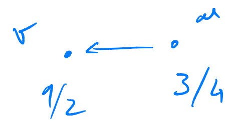
· Cross edge

T

- Se $d_v < d_u \wedge (u, v) \in E$, então (u, v) é um back edge

$[_v \;]_v \; [_u \;]_u$

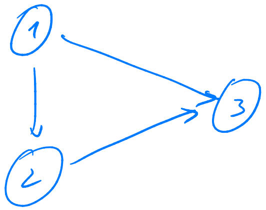
F



cross edge

22.1-6

- Descobrir se um grafo (guardado em memória como uma matriz de adjacências) tem um universal sink em tempo $\underline{O(V)}$



$$
\begin{array}{c c}
 & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}
\end{array}
$$

* i é universal sink sse
- a coluna i só contém 1's
- a linha i só contém 0's exceto coluna i

$$
\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}
$$

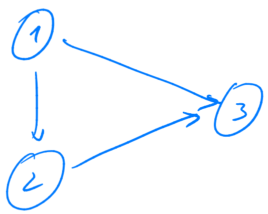
$$
\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}
$$

```
isSink (M, k)
    for j=1 to M.size
        if M_kj ≠ 0
            return false
    for i=1 to M.size
        if M_ik ≠ 1 ∧ i ≠ k
            return false
return true
```

Complexidade:
$\underline{O(n)}$

22.1-6

- Descobrir se um grafo tem um universal sink em tempo $\underline{O(V)}$
  (guardado em memória como uma matriz de adjacências)

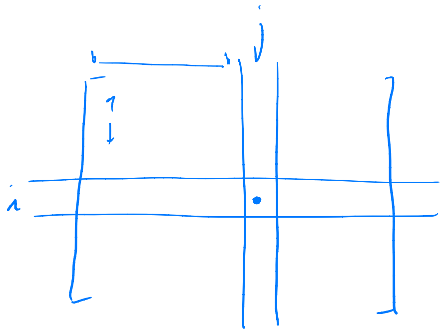


$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

* i é universal sink sse
- a coluna i só contém 1's
- a linha i só contém 0's exceto coluna i

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

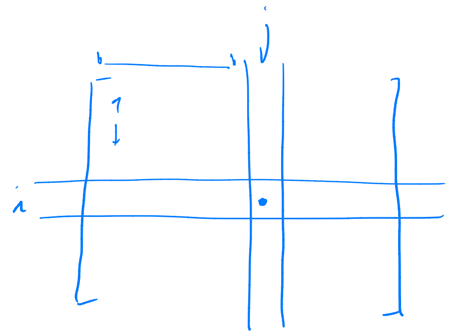
$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



- Vértices $\underbrace{1 \leq k \leq i-1}$ têm pelo menos um "1" nas suas linhas
  $\Rightarrow$ Não são universal sinks

22.1-6

• Descobrir se um grafo (guardado em memória como uma matriz de adjacências) tem um universal sink em tempo $\underline{O(V)}$

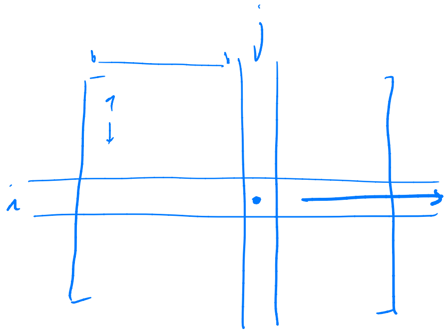


• vértices $\underbrace{1 \le k < i-1}$

têm pelo menos um "1" nas suas linhas

⇒ Não são universal sinks

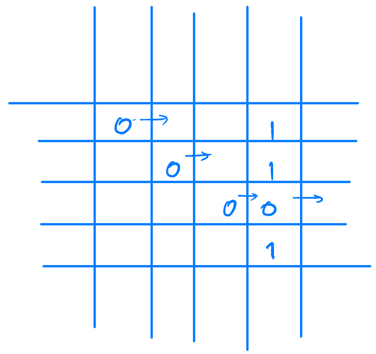
Observação 1



• vértices $\underbrace{i < k \le n}$

todos estes vértices têm um "0" numa posição não diagonal

⇒ Não são universal sinks



(1, 1) (1, 2)

(2, 2) (2, 3)

22.1-6

- Descobrir se um grafo (guardado em memória como uma matriz de adjacências) tem um universal sink em tempo $\underline{O(V)}$

```
FindSink(M)
    i = 1;  j = 1;
    while (i ≤ M.size && j ≤ M.size) {
        if (M_ij = 0) {
            j++
        } else {
            i++
        }
    }
    if (i ≤ M.size)
        return isSink(M, i)
    else return Nil
}
```