Percursos e Conectividade em Grafos
Depth-First Search

Fernando Lobo

Algoritmos e Estrutura de Dados

DFS: Depth-First Search (pesquisa em profundidade)

- Assim que se descobre um nó, começa-se logo a explorar os descendentes desse nó. Quando não é possível explorar mais, volta-se para trás (backtracking).
- Ao contrário de BFS, a pesquisa pode recomeçar de novo. Isto é, todos os nós do grafo são explorados, mesmo que o grafo seja desconexo.
- Em vez de uma BFS Tree, produz uma DFS Forest (várias árvores, uma para cada componente conexa).

Implementação de DFS

- Usa-se um stack em vez de uma fila (e até podemos nem usar o stack, se implementarmos o algoritmo de forma recursiva).
- Mantemos dois *timestamps* em cada nó:
  - $v.d \rightarrow$ discovery time, instante em que v é descoberto.
  - $v.f \rightarrow$ finish time, instante em que todos os descendentes de v já foram descobertos.
- Estes timestamps são inteiros distintos entre 1 e $2|V|$
- $v.d < v.f \ , \ \forall_v$
- (Estes timestamps não são necessários, mas ajudam-nos a compreender o algoritmo.)

Implementação de DFS

- Tal como em BFS, cada nó tem um atributo booleano *visited*, que indica se o nó já foi visitado.
- (Uma vez mais, esta implementação é ligeiramente diferente da que está no livro CLRS. Em vez do atributo *visited*, o livro usa um atributo *color* que pode assumir os valores WHITE, GRAY, BLACK.)

Pseudocódigo (implementação recursiva)

```
DFS(G)
    for each u ∈ G.V
        u.visited = FALSE
        u.π = NIL
    time = 0
    for each u ∈ G.V
        if not u.visited
            DFS-VISIT(u)
```


Pseudocódigo

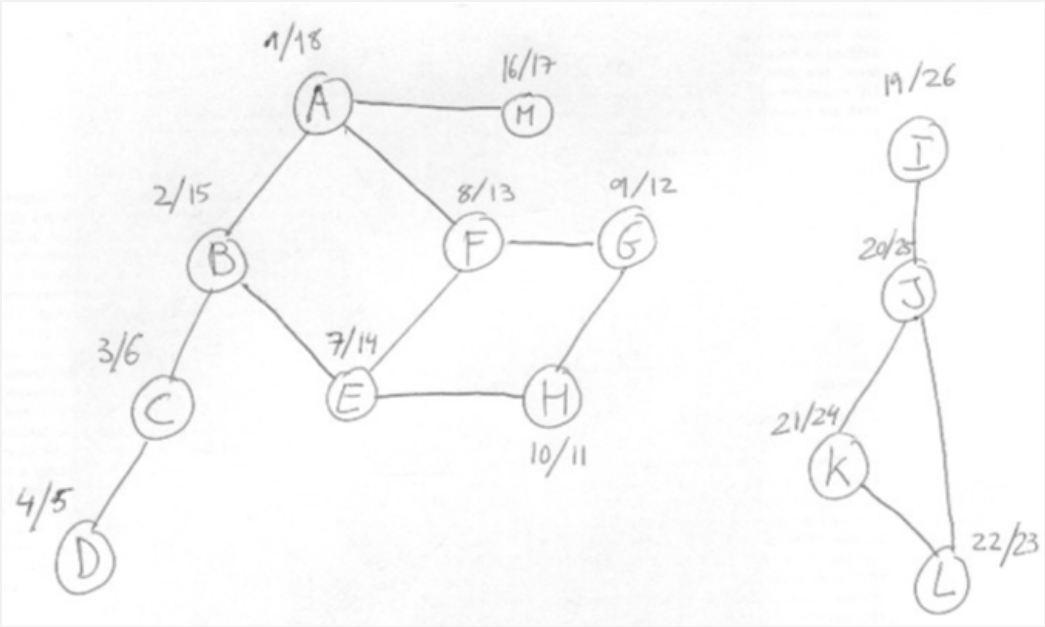
```
DFS-VISIT(u)
    u.visited = TRUE        // u acaba de ser descoberto
    time = time + 1
    u.d = time
    for each v ∈ G.Adj[u]        // explora o arco (u, v)
        if not v.visited
            v.π = u
            DFS-VISIT(v)
    // terminou de explorar os descendentes de u
    time = time + 1
    u.f = time
```

Complexidade de DFS

- Complexidade de DFS $= \Theta(V + E)$.
- Θ e não apenas O como em BFS, porque aqui todos os nós e todos os arcos são examinados.
- $\Theta(V)$ da inicialização.
- $\Theta(E)$ porque para cada nó u, são examinados $|Adj[u]|$ arcos.

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

Exemplo

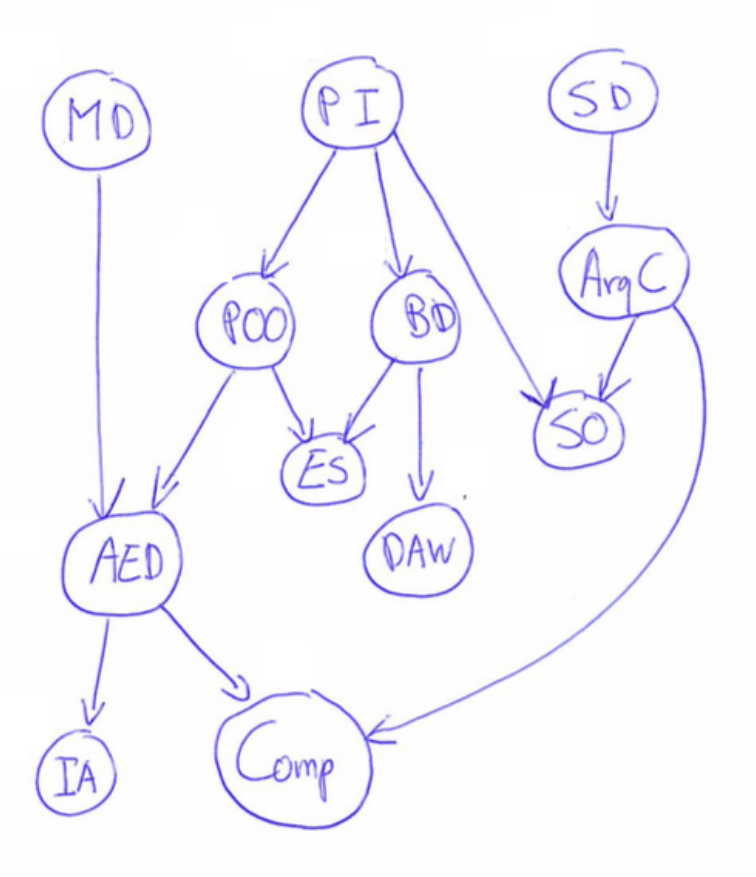


Notação na figura: $d/f \rightarrow$ discovery time / finish time.

Grafos orientados sem ciclos

- Em Inglês, *Directed Acyclic Graphs* (DAGs).
- São normalmente usados para expressar dependências.

Exemplo: precedências de disciplinas



DAGs - Ordenação topológica

- Aplica-se a DAGs
- É uma ordem linear dos nós de um DAG de tal forma que se $(u, v) \in E$, então u aparece antes de v.

TOPOLOGICAL-SORT(G)

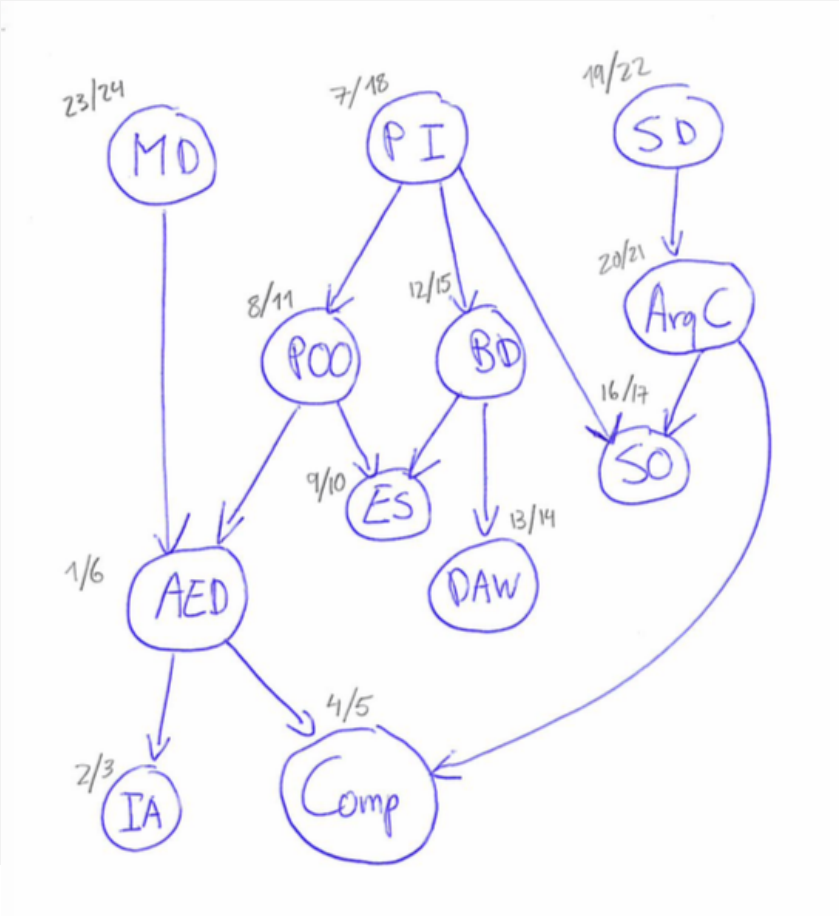
1 Chamar DFS(G) para obter $v.f$, $\forall_v$
2 Assim que cada nó termina, insere-o na cabeça de uma lista.
3 retorna a lista de nós.

Nota: Os passos 2 e 3 dão os nós do grafo por ordem decrescente de finish time.

Complexidade?

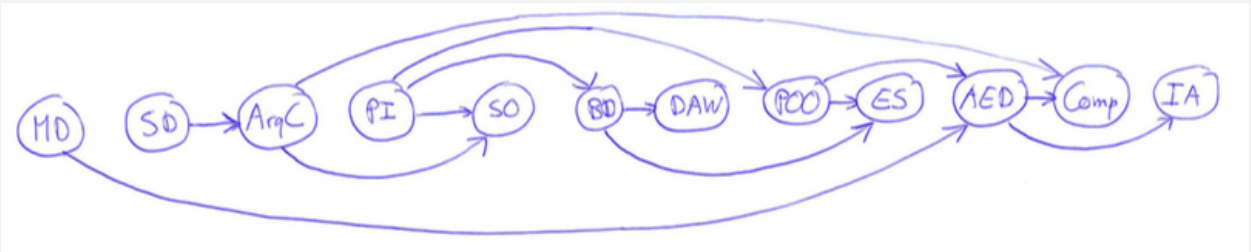
- $\Theta(V + E)$
- Idêntico à complexidade de DFS.
- Inserir na lista é $\Theta(1)$.

Exemplo com DFS: começa em AED e recomeça em PI, SD, MD



Redesenhando o DAG

O.T. = MD, SD, ArqC, PI, SO, BD, DAW, POO, ES, AED, Comp, IA.



Grafo pode ser redesenhado com os arcos a irem todos da esquerda para a direita.

Correção do algoritmo

- Seja $G = (V, E)$ um DAG. Teremos de mostrar que após correr DFS em G,

$$v.f < u.f \ , \ \forall_{(u,v) \in E}$$

- Quando (u, v) é explorado, $u.visited$ é garantidamente $\textsc{True}$.
  - Se $v.visited$ é $\textsc{False}$, então visita v, e garantidamente $v.f < u.f$
  - Se $v.visited$ é $\textsc{True}$, então:
    - ⋆ Se v ainda não terminou, então v é antecessor de u, o que implica que o grafo tem ciclo, i.e. não é um DAG.
    - ⋆ Se v já terminou, como estamos a explorar (u, v) então garantidamente $v.f < u.f$