→ Grafo dirigido aciclico - DAG

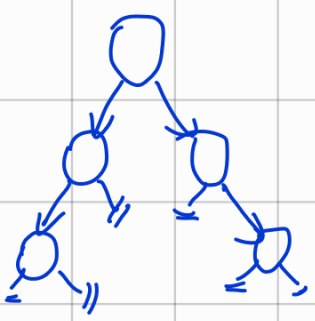
→ Podemos modelar problemas onde queremos realizar várias tarefas, mas existem dependências

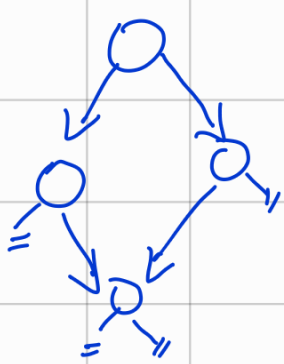
ex: Makefile make -j

Para realizar uma tarefa, precisamos primeiro realizar todas as tarefas das quais eles dependem

Definição: Um DAG binário é um grafo dirigido acíclico com duas arestas saindo de cada nó, identificados como arestas da esquerda e da direita, uma delas, ou ambas, podem ser NULL.

-> Então qual a diferença de DAG binário e Árvore binária?





A distinção entre DAG binário e árvore binária é que o DAG binário pode ter mais de um link apontando para um nó.

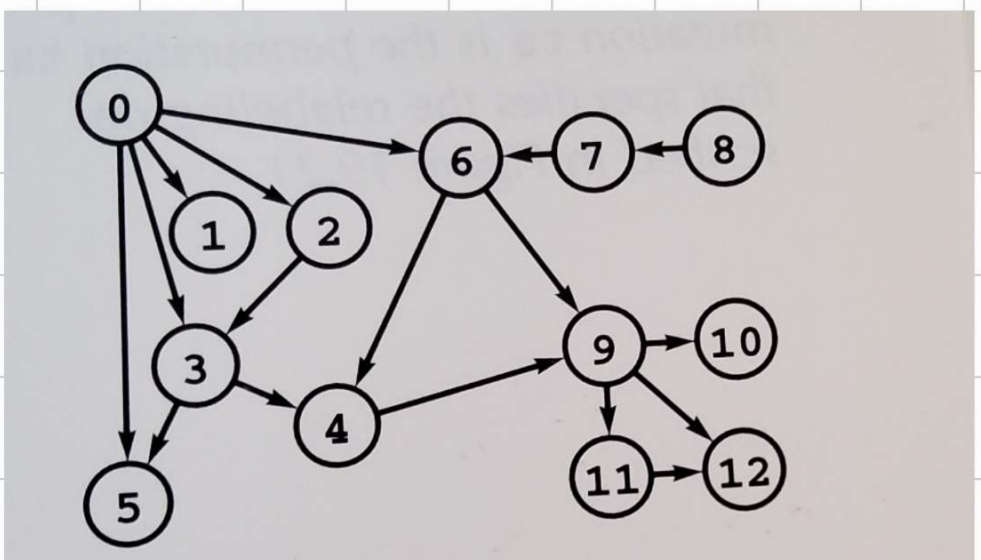
DAGs binários fornecem uma maneira compacta de representar árvores binárias em algumas aplicações

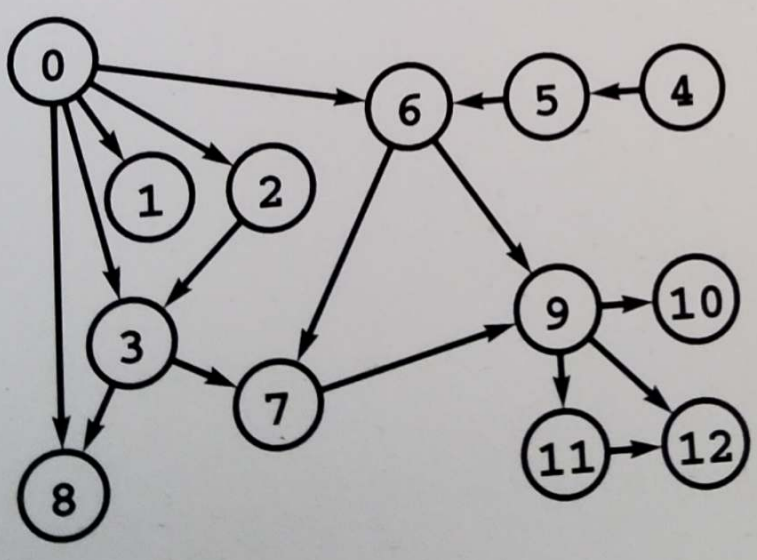
→ Ordenação topológica

→ O objetivo é processar os vértices de um DAG de tal forma que cada vértice é processado antes de todos os vértices para o qual aponta

→ Ordenação topológica (renomear)

→ Dado um DAG, renomeie os vértices de forma que toda aresta dirigida aponta de um vértice de número menor para um de número maior



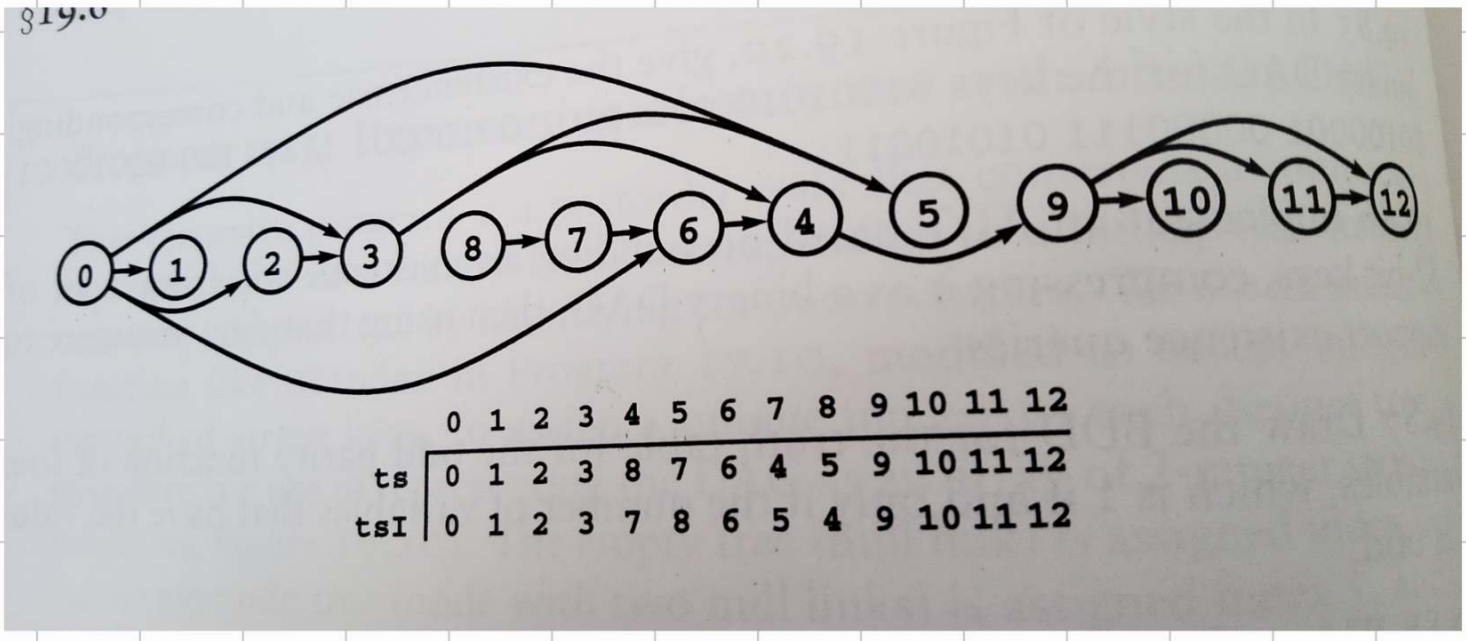


	0	1	2	3	4	5	6	7	8	9	10	11	12
tsI	0	1	2	3	7	8	6	5	4	9	10	11	12

Figure 19.21

→ Ordenação topológica (reorganizar)

→ Dado um DAG, reorganize os vértices em uma linha horizontal de forma que as arestas dirigidos apontem da esquerda para direita



	0	1	2	3	4	5	6	7	8	9	10	11	12
ts	0	1	2	3	8	7	6	4	5	9	10	11	12
tsI	0	1	2	3	7	8	6	5	4	9	10	11	12

Dada uma reorganização, podemos obter a renomeação apenas definindo o rótulo 0 para o primeiro vértice, 1 para o segundo e assim por diante.

Por exemplo, se um vetor ts possui os vértices em ordem topológica, então o laço

```
for(i=0; i<V; i++) tsI[ts[i]] = i;
```

define a renomeação do vetor indexado por vértice tsI

Da mesma forma, se temos um vetor de renomeação tsI, podemos obter a reorganização com o loop

```
for(i=0; i<V; i++) ts[tsI[i]] = i;
```

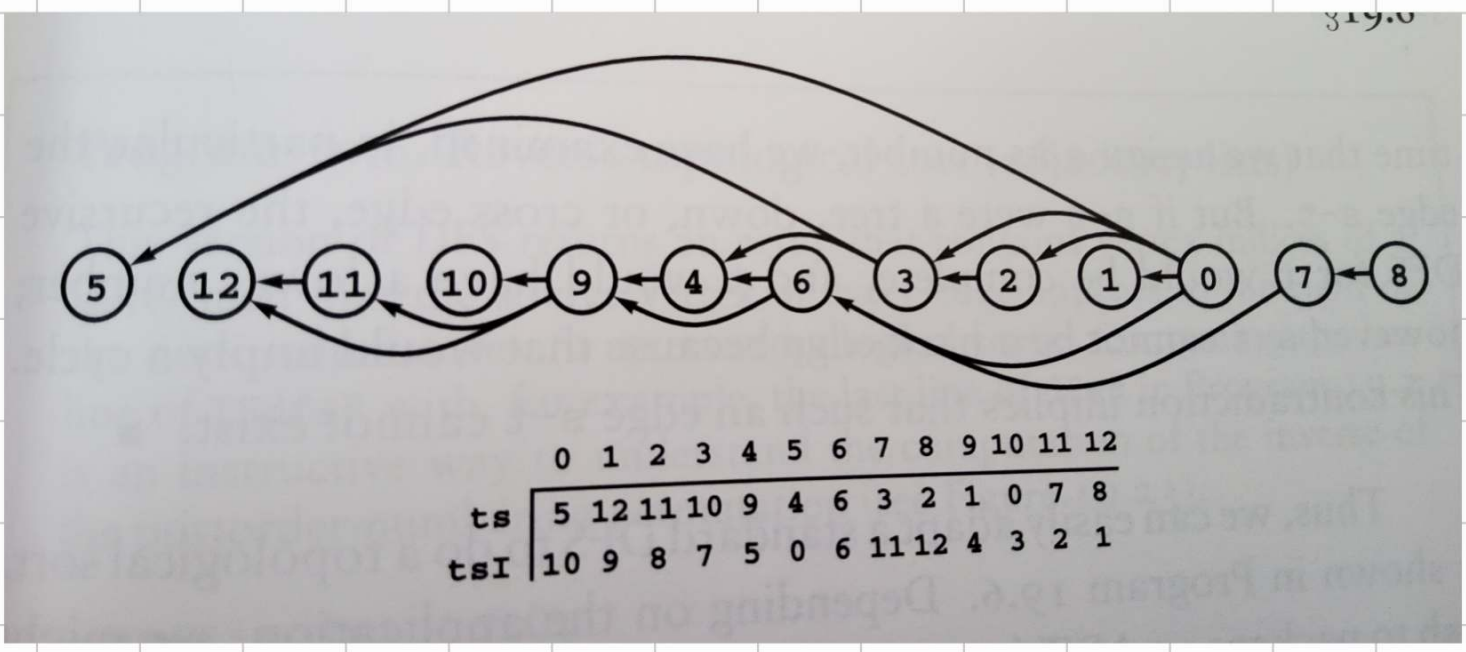
que coloca o vértice que teria o rótulo 0 na primeira posição da lista, o vértice que teria o rótulo 1 em segundo ....

Na maior parte das vezes usamos o termo Ordenação topológica para se referir ao problema de reorganização

Não existe somente uma ordenação topológica. Em problemas de realização de tarefas isto fica muito claro, especialmente quando uma tarefa não possui dependência direta, ou indireta, de outra tarefa e, então, podem ser realizados em qualquer ordem, ou até mesmo em paralelo.

As vezes podemos interpretar as arestas de outra maneira: Dizemos que uma aresta dirigida de s a t significa que o vértice s "depende" do vértice t

Por exemplo, os vértices podem representar os termos definidos em um livro, com uma aresta de s a t se a definição s use a t. Neste caso, seria útil encontrar uma ordenação com a propriedade que cada termo é definido antes de ser usado por outra definição. Usando esta ordenação corresponde a posicionar os vértices em uma linha tal que as arestas vão da direita para a esquerda - ordenação topológica reversa.



	0	1	2	3	4	5	6	7	8	9	10	11	12
ts	5	12	11	10	9	4	6	3	2	1	0	7	8
tsI	10	9	8	7	5	0	6	11	12	4	3	2	1

E como fazemos a ordenação topológica Reversa?

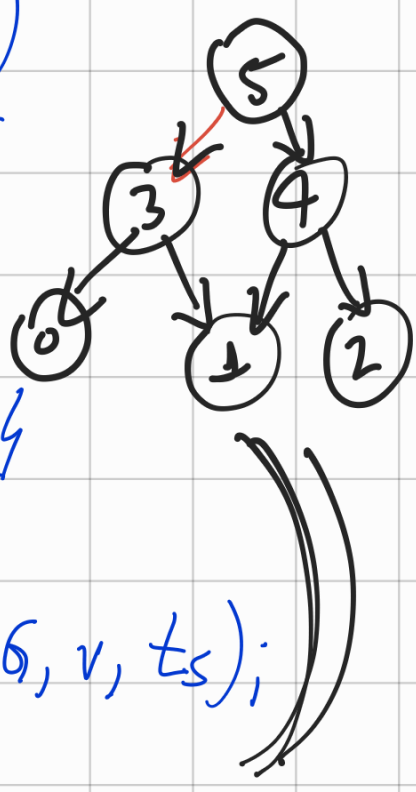
DFS

Quando a entrada é um DAG, a numeração posorden coloca os vértices em ordenação topológica reversa.

Ou seja, numeremos cada vértice como a ação final de função recursiva de DFS,

Propriedade: A numeração posorden de uma DFS produz uma ordenação topológica reversa.

```
Void DAGts(Graph G, int ts[])
{ int v;
  cnt0=0;
  [ for(v=0; v<G->V; v++)
      { ts[v]=-1; pre[v]=-1; }
  for(v=0; v<G->V; v++)
      if(pre[v]==-1) tSdfsR(G, v, ts);
}
```



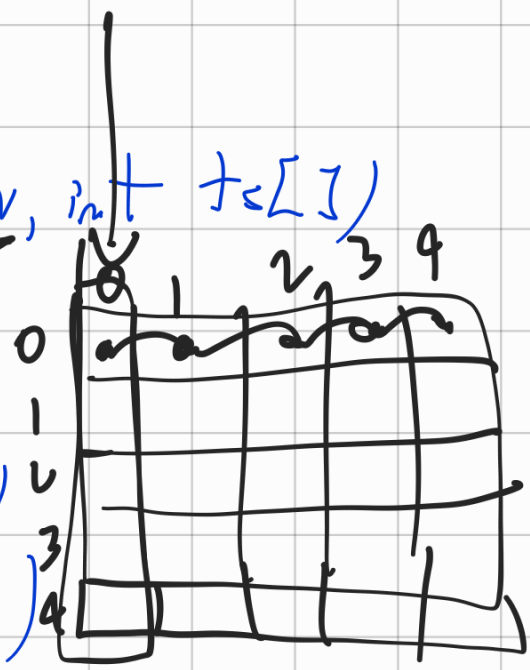
V+(V+E)

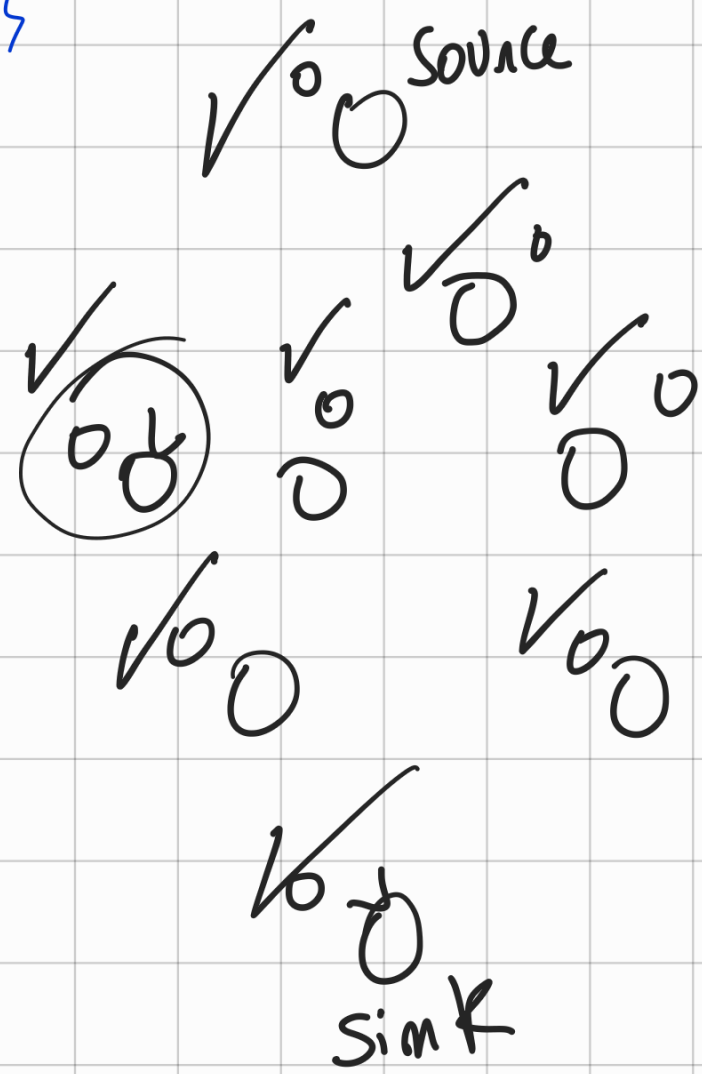
```
Void tSdfsR(Graph G, int v, int ts[])
{
  pre[v]=0;
  (for(int t=G->adj[v]; t!=NULL; t=t->next)
      if(pre[t->v]==-1) tSdfsR(G, t->v, ts);

  ts[cnt0++]=v;
}
```

- Como produzir uma ordenação topológica?

Geron TS com DFS

```
Void tsDfsR(Graph G, int v, int ts[])
{ int u;
  • pre[v]=0;
  • for (u=0; u < G->V; u++)
      ● if(G->adj[u][v] != 0)
          if(pre[u] == -1) tsDfsR(G, u, ts)

  ts[cnt0++] = v;
}
```

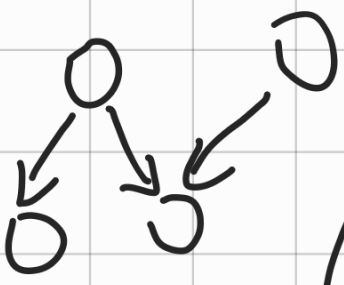




→ Propriedade: Todo DAG possui pelo menos 1 source e pelo menos 1 sink

Usando esta propriedade podemos fazer uma TS usando algo parecido com uma BFS:

- Mantendo um vetor vértice-indexado que mantém a contagem de entrada em cada vértice.
  - Vértices com grau de entrada 0 são sources, então inicializamos uma fila em uma passada no DAG
  - então realizamos as seguintes operações até que a fila fique vazia:
    - Remove uma source da fila (é notável)
    - Decrementa o grau de entrada dos vértices de destino
      - Se decrementar causa o grau de entrada ficar em 0, insira este vértice na fila



```
static int in[maxV];

void DAGts(Graph G, int ts[])
{
  for (int v=0; v<G->V; v++)
    { in[v]=0; ts[v]=-1; }

  for (int v=0; v<G->V; v++)
    for (t=G->adj[v]; t!=NULL; t=t->next)
      in[t->v]++;

  QUEUEinit(G->V);
  for (int v=0; v<G->V; v++)
    if (in[v]==0) QUEUEput(v);

  int v;
  for (int i=0; !QUEUEempty(); i++)
    {
      ts[i]=(v=QUEUEget());
      for (t=G->adj[v]; t!=NULL; t=t->next)
        if (--in[t->v]==0) QUEUEput(t->v);
    }
}
```

V+E