Calendarização de Tarefas - Algoritmos em Grafos

Ana Paula Tomás

Desenho e Análise de Algoritmos
Universidade do Porto

Abril 2024

Tópicos a abordar

1. Revisão – Implementação: BFS_Visit (em Java)
2. Ordenação topológica de DAGs e aplicações (CPM)
3. CPM: Caminhos máximos em DAGs

Implementação de BFS_Visit (em Java)

```
import java.util.LinkedList;
import java.util.Queue;

public static void bfs_visit(int s,Graph g, boolean [] visto, int [] pai){
    // boolean [] visto = new boolean[g.num_vertices() + 1];
    // int [] pai = new int[g.num_vertices() + 1];
    visto[s] = true;
    Queue<Integer> q = new LinkedList<>();
    q.add(s);
    do {
        int v = q.poll();
        for (Edge e: g.adjs_no(v)) {
            int w = e.endnode();
            if (!visto[w]) {
                q.add(w);
                visto[w] = true;
                pai[w] = v;
            }
         }
     } while(!q.isEmpty());
}
```

Planeamento de tarefas

Exemplo 3: Scheduling

Um projeto é constituído por um conjunto de tarefas, sendo conhecida a **duração** de cada tarefa e as **restrições de precedência** entre tarefas. Não se pode dar início a uma tarefa sem que as que a precedem estejam concluídas. Pretendemos agendar as tarefas de modo a concluir o projeto o mais cedo possível. Cada tarefa requer um certo número de **trabalhadores**. Cada trabalhador só pode estar a realizar uma tarefa em cada instante. Assuma que:

- Caso 1: não há restrições quanto ao número de trabalhadores a contratar.
- Caso 2: há restrições quanto ao número de trabalhadores a contratar.

Admita que as habilitações necessárias são idênticas para todas as tarefas.

Caso 1: sem partilha de recursos

Caso 2: com partilha de recursos

Exemplo: Tarefas A, B, C, D e E; A precede C; B precede C e D; C precede E.

	A	B	C	D	E
duração	1	3	4	5	2
# trabalhadores	2	3	1	1	2

Escalonamento sem partilha de recursos

Exemplo 3 - Caso 1

Dada a descrição das tarefas do projeto, as suas durações d_i com $i \in$ *Tarefas*, e a relação de precedência $\mathcal{R}$, determinar a data de conclusão mais próxima para o projeto e uma data para início de cada tarefa.

Variáveis de decisão:

- z: data de conclusão do projeto
- x_i: data de início da tarefa i, para $i \in$ Tarefas

Modelo de otimização linear:

$$
\begin{array}{l}
\textbf{minimizar} \;\; z \\
\textbf{sujeito a} \\
\left\{
\begin{array}{ll}
x_i + d_i \leq x_j, & \text{para todo } (i,j) \in \mathcal{R} \\[2ex]
x_i + d_i \leq z, & \text{para todo } i \in \text{Tarefas} \\[2ex]
z \in \mathbb{R}_0^+, \;\; x_i \in \mathbb{R}_0^+, & \text{para todo } i \in \text{Tarefas}
\end{array}
\right.
\end{array}
$$

Planeamento de tarefas

Problema de escalonamento de tarefas

Um projeto é constituído por um conjunto de tarefas, sendo conhecida a duração de cada tarefa e as restrições de precedência entre tarefas. Não se pode dar início a uma tarefa sem concluir as que a precedem. Pretende-se agendar as tarefas de modo a concluir o projeto o mais cedo possível.

- Cenário 1: Há apenas uma pessoa para realizar o projeto e não pode realizar várias tarefas ao mesmo tempo.

  Problema: Determinar um plano, i.e., uma **ordenação das tarefas compatível com as precedências** definidas.

- Cenário 2: As tarefas não partilham recursos. Podem ser realizadas várias tarefas simultaneamente, devendo satisfazer as precedências definidas.

  Problema: Determinar quando é que o projeto pode ficar concluído:

  (i) se todas as tarefas tiverem **duração unitária**;
  (ii) se as tarefas puderem ter **durações distintas**.

Redes Nó-Atividade e Arco-Atividade

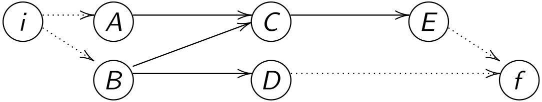
Exemplo: Tarefas A, B, C, D e E; A precede C, B precede C e D, e C precede E.

- **Nó-atividade:** os nós representam atividades e os ramos precedências.

Redes Nó-Atividade e Arco-Atividade

Exemplo: Tarefas A, B, C, D e E; A precede C, B precede C e D, e C precede E.

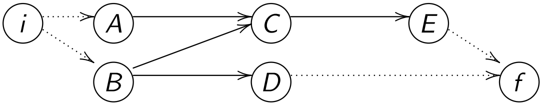
- **Nó-atividade:** os nós representam atividades e os ramos precedências.



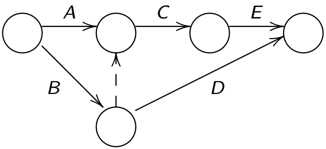
Redes Nó-Atividade e Arco-Atividade

Exemplo: Tarefas A, B, C, D e E; A precede C, B precede C e D, e C precede E.

- **Nó-atividade:** os nós representam atividades e os ramos precedências.



- **Arco-atividade:** os ramos representam atividades e os nós acontecimentos.



Para não criar precedências novas, foi necessário introduzir uma **atividade fictícia** (a tracejado no grafo). Define-se com **duração zero**

Planeamento de tarefas – Modelo Nó-Atividade

A **relação de precedência** é dada por um DAG (grafo dirigido acíclico). Os nós do grafo definem as tarefas e um ramo (x, y) representa o facto de a tarefa x preceder a tarefa y. Se tem os ramos (x, y) e (y, z), também x precede z mesmo que não tenha o ramo (x, z).

- Cenário 1: **Nenhum par de tarefas decorre simultaneamente.**
  Problema: corresponde à ordenação topológica dos nós de um DAG

- Cenário 2: **Algumas tarefas podem decorrer em simultâneo.**
  Calcular a duração mínima do projeto se:
  - (i) todas as tarefas têm **duração unitária**;
    Problema: Encontrar o caminho mais longo num DAG, sendo o comprimento é dado pelo *número de ramos* no caminho.
  - (ii) as tarefas podem ter **durações distintas**.
    Problema: Encontrar o caminho mais longo num DAG, sendo o comprimento dado pela *soma dos valores nos nós* do caminho.

Ordenação topológica dos nós de um DAG

Dado um DAG $G = (V, A)$, construir uma fila com os nós por ordem topológica.

TopSortDAG(G)

Para todo $v \in G.V$ fazer $GrauE[v] \leftarrow 0$;
Para todo $(v, w) \in G.A$ fazer $GrauE[w] \leftarrow GrauE[w] + 1$;
$S \leftarrow \{v \in G.V \mid GrauE[v] = 0\}$; /* S pode ser suportado por uma fila ou uma pilha. */
$Q \leftarrow \textsc{MkEmptyQueue}()$;
Enquanto $(S \neq \emptyset)$ fazer
    $v \leftarrow$ Retirar um elemento qualquer de S; /* se S for uma fila, retira o primeiro */
    $\textsc{Enqueue}(v, Q)$;
    Para todo $w \in G.Adjs[v]$ fazer
        $GrauE[w] \leftarrow GrauE[w] - 1$; /* implicitamente, está a retirar o ramo (v, w) */
        Se $GrauE[w] = 0$ então $S \leftarrow S \cup \{w\}$; /* todos os precedentes de w já tratados */
retorna Q;

Ordenação topológica dos nós de um DAG

Dado um DAG $G = (V, A)$, construir uma fila com os nós por ordem topológica.

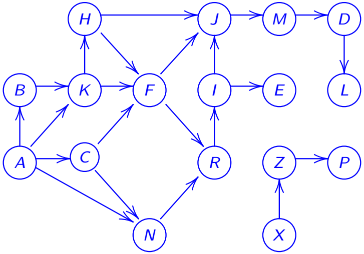
TopSortDAG(G)

Para todo $v \in G.V$ fazer $GrauE[v] \leftarrow 0$;
Para todo $(v, w) \in G.A$ fazer $GrauE[w] \leftarrow GrauE[w] + 1$;
$S \leftarrow \{v \in G.V \mid GrauE[v] = 0\}$; /* S pode ser suportado por uma fila ou uma pilha. */
$Q \leftarrow \textsc{MkEmptyQueue}()$;
Enquanto $(S \neq \emptyset)$ fazer
    $v \leftarrow$ Retirar um elemento qualquer de S; /* se S for uma fila, retira o primeiro */
    $\textsc{Enqueue}(v, Q)$;
    Para todo $w \in G.Adjs[v]$ fazer
        $GrauE[w] \leftarrow GrauE[w] - 1$; /* implicitamente, está a retirar o ramo (v, w) */
        Se $GrauE[w] = 0$ então $S \leftarrow S \cup \{w\}$; /* todos os precedentes de w já tratados */
retorna Q;

Que propriedade dos DAGs suporta a correção do algoritmo?

Qualquer DAG tem algum nó com **grau de entrada zero**. Se se retirar um ramo de um DAG, obtém-se um DAG. Se G tiver ciclos (não for DAG), termina mas $|Q| < |V|$.

Exemplo



Se S for suportada por uma fila FIFO e se usar **ordem lexicográfica** para desempate então, no teste de $S \neq \emptyset$, pela i-ésima vez, S teria:

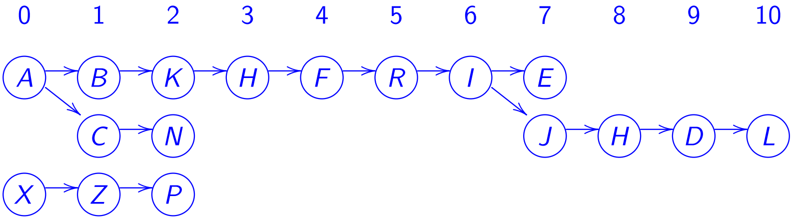
1	2	3	4	5	6	7	8	9	10	11
A	X	B	C	Z	K	N	P	H	F	R
X	B	C	Z	K	N	P	H			
	C	Z	K	N	P	H				

12	13	14	15	16	17	18
I	E	J	M	D	L	
	J					

$$Q = [A, X, B, C, Z, K, N, P, H, F, R, I, E, J, M, D, L]$$

Exemplo (cont)

Se os nós representarem tarefas unitárias (i.e., com duração 1) e o grafo precedências entre tarefas, como seria se se pudesse executar tarefas independentes em paralelo?



A e X podem começar no instante 0, B, C e Z em 1, K, N e P em 2, ...
L estaria concluída no instante 11 e não seria possível terminar mais cedo.

Ordenação topológica de um DAG por DFS

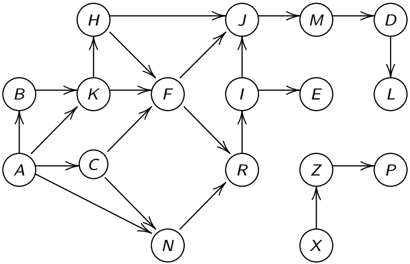
Os nós ficam na **stack construída por DFS** por ordem topológica (o primeiro no topo)

TopSort_DFS(G)
  $S \leftarrow \{\}$;   // definir stack vazia
  Para cada $v \in G.V$ fazer $visitado[v] \leftarrow$ **false**;
  Para cada $v \in G.V$ fazer
    Se $visitado[v] =$ **false** então TopSort_DFSVisit($v, G, visitado, S$);
  retorna S;   // ordem topológica se efetuar POPs sucessivos de S

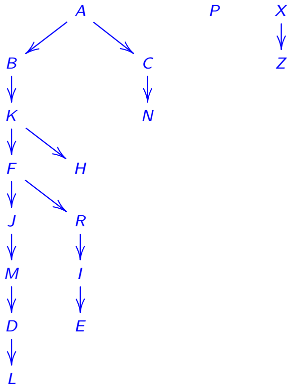
TopSort_DFSVisit($v, G, visitado, S$)
  $visitado[v] \leftarrow$ **true**;
  Para cada $w \in G.Adjs[v]$ fazer
    Se $visitado[w] =$ **false** então
      TopSort_DFSVisit($w, G, visitado, S$);
  Push(v, S);

Visita em profundidade para determinar ordem topológica

Assumir que, se for necessário escolher adjacente, segue **ordem lexicográfica**.



Floresta (pesquisa DFS)



Ordem de descoberta:

$A, B, K, F, J, M, D, L, R, I, E, H, C, N, P, X, Z$

Ordem na Stack (topo é X):

$L, D, M, J, E, I, R, F, H, K, B, N, C, A, P, Z, X$

1 Revisão – Implementação: BFS_Visit (em Java)

2 Ordenação topológica de DAGs e aplicações (CPM)

3 CPM: Caminhos máximos em DAGs

Caminho máximo num DAG (distâncias unitárias)

Problema: obter um **caminho máximo** num grafo dirigido acíclico $G = (V, A)$, sendo o comprimento dado pelo número de ramos do caminho.

MaxPathDAG(G)

Para todo $v \in G.V$ fazer $ES[v] \leftarrow 0$; $Pai[v] \leftarrow$ NULL; $GrauE[v] \leftarrow 0$;
Para todo $(v, w) \in G.A$ fazer $GrauE[w] \leftarrow GrauE[w] + 1$;
$S \leftarrow \{v \in G.V \mid GrauE[v] = 0\}$; /* S deve ser suportado por uma fila ou uma pilha. */
$Max \leftarrow -1$; $v_f \leftarrow$ NULL; // v_f **indica o vértice final no caminho mais longo obtido**
Enquanto $(S \neq \emptyset)$ fazer
    $v \leftarrow$ um qualquer elemento de S;
    $S \leftarrow S \setminus \{v\}$;
    Se $Max < ES[v]$ então $Max \leftarrow ES[v]$; $v_f \leftarrow v$; /* $ES[v]$ o número de ramos do caminho */
    Para todo $w \in G.Adjs[v]$ fazer
        Se $ES[w] < ES[v] + 1$ então
            $ES[w] \leftarrow ES[v] + 1$; $Pai[w] \leftarrow v$;
        $GrauE[w] \leftarrow GrauE[w] - 1$;
        Se $GrauE[w] = 0$ então $S \leftarrow S \cup \{w\}$;
EscreveCaminho(v_f, Pai); escrever(Max);

Caminho máximo num DAG com pesos associados aos nós

Dado $G = (V, A, D)$ em que $D(v) \in \mathbb{R}_0^+$ é a duração da tarefa $v \in V$, determinar $ES[v]$, o instante mais próximo em que pode dar início a v **("earliest start")**.

MaxPathWeightedDAG(G)

Para todo $v \in G.V$ fazer $ES[v] \leftarrow 0$; $Pai[v] \leftarrow$ `Nenhum`; $GrauE[v] \leftarrow 0$;
Para todo $(v, w) \in G.A$ fazer $GrauE[w] \leftarrow GrauE[w] + 1$;
$S \leftarrow \{v \in G.V \mid GrauE[v] = 0\}$; /* S deve ser suportado por uma fila ou uma pilha. */
$Max \leftarrow -1$; $v_f \leftarrow$ `NULL`; /* $ES[v]$ o número de ramos do caminho */
Enquanto $(S \neq \emptyset)$ fazer
  $v \leftarrow$ um qualquer elemento de S; $S \leftarrow S \setminus \{v\}$;
  Se $Max < ES[v] + D[v]$ então $Max \leftarrow ES[v] + D[v]$; $v_f \leftarrow v$;
  Para todo $w \in G.Adjs[v]$ fazer
    Se $ES[w] < ES[v] + D[v]$ então
      $ES[w] \leftarrow ES[v] + D[v]$; $Pai[w] \leftarrow v$;
    $GrauE[w] \leftarrow GrauE[w] - 1$;
    Se $GrauE[w] = 0$ então $S \leftarrow S \cup \{w\}$;
EscreveCaminho(v_f, Pai); escrever(Max);

Modelo Nó-Atividade

Os nós do grafo denotam atividades e os arcos definem as precedências. As durações ficam associadas aos nós: d_i denota a duração da atividade i.

Objetivo: Determinar a duração mínima do projeto

Assumindo que se vai concluir o projeto o mais cedo possível, define-se:

ES_i *earliest start time*
data de início mais próxima para a tarefa i
$ES_i = \max_{(j,i)\in A}\{ EF_j \}$

LF_i *latest finish time*
data de conclusão mais afastada para a tarefa i
menor das datas de início mais afastadas para as tarefas que seguem i
$LF_i = \min_{(i,j)\in A}\{ LS_j \}$

EF_i *earliest finish time*
data de conclusão mais próxima para a tarefa i
$EF_i = ES_i + d_i$

LS_i *latest start time*
data de início mais afastada para a tarefa i
$LS_i = LF_i - d_i$

Planeamento de tarefas – **Modelo Arco-Atividade**

Os nós do grafo representam acontecimentos (início ou fim de um conjunto de atividades) e os arcos representam as atividades: a atividade (i,j) tem i como acontecimento inicial e j como acontecimento final. Seja d_{ij} a sua duração.

Objetivo: Determinar a duração mínima do projeto.

- Sejam 1 e n os acontecimentos início e fim do projeto. Então,
  - $ES_{1j} = 0$, para as atividades com início no nó 1;
  - $LF_{jn} =$ duração mínima do projeto, para as atividades com fim no nó n.

Planeamento de tarefas – Modelo Arco-Atividade

Os nós do grafo representam acontecimentos (início ou fim de um conjunto de atividades) e os arcos representam as atividades: a atividade (i,j) tem i como acontecimento inicial e j como acontecimento final. Seja d_{ij} a sua duração.

Objetivo: Determinar a duração mínima do projeto.

- Sejam 1 e n os acontecimentos início e fim do projeto. Então,
  - $ES_{1j} = 0$, para as atividades com início no nó 1;
  - $LF_{jn} =$ duração mínima do projeto, para as atividades com fim no nó n.
- **Pode começar (i,j) logo que todas as atividades que a precedem estiverem concluídas.**

$$ES_{ij} = \max\{\, EF_{ki} \mid (k,i) \in A\} \qquad\qquad EF_{ij} = ES_{ij} + d_{ij}$$

- **Para não atrasar a realização do projeto, (i,j) tem de estar concluída quando alguma das atividades que a segue não puder ser mais adiada.**

$$LF_{ij} = \min\{\, LS_{jk} \mid (j,k) \in A\} \qquad\qquad LS_{ij} = LF_{ij} - d_{ij}$$

Atividades Críticas e Folgas

Dois tipos de folgas

- Folga Total FT_{ij}: diferença entre a data de início mais afastada e a data de início mais próxima para a atividade (i, j).

$$FT_{ij} = LS_{ij} - ES_{ij} = LF_{ij} - EF_{ij} = LF_{ij} - ES_{ij} - d_{ij}$$

Atividades Críticas e Folgas

Dois tipos de folgas

- Folga Total FT_{ij}: diferença entre a data de início mais afastada e a data de início mais próxima para a atividade (i, j).

$$\boxed{FT_{ij} = LS_{ij} - ES_{ij}} = LF_{ij} - EF_{ij} = LF_{ij} - ES_{ij} - d_{ij}$$

- Folga Livre FL_{ij}: folga que não impede que as atividades que seguem (i, j) possam começar na sua data de início mais próxima.

$$FL_{ij} = min\{ES_{jk} \mid (j, k) \in A\} - EF_{ij}$$

Atividades Críticas e Folgas

Dois tipos de folgas

- Folga Total FT_{ij}: diferença entre a data de início mais afastada e a data de início mais próxima para a atividade (i, j).

$$FT_{ij} = LS_{ij} - ES_{ij} = LF_{ij} - EF_{ij} = LF_{ij} - ES_{ij} - d_{ij}$$

- Folga Livre FL_{ij}: folga que não impede que as atividades que seguem (i, j) possam começar na sua data de início mais próxima.

$$FL_{ij} = min\{ES_{jk} \mid (j, k) \in A\} - EF_{ij}$$

Propriedade: A folga livre é sempre menor ou igual que a folga total.

Atividades Críticas e Folgas

Dois tipos de folgas

- Folga Total FT_{ij}: diferença entre a data de início mais afastada e a data de início mais próxima para a atividade (i, j).

$$\boxed{FT_{ij} = LS_{ij} - ES_{ij}} = LF_{ij} - EF_{ij} = LF_{ij} - ES_{ij} - d_{ij}$$

- Folga Livre FL_{ij}: folga que não impede que as atividades que seguem (i, j) possam começar na sua data de início mais próxima.

$$FL_{ij} = min\{ES_{jk} \mid (j, k) \in A\} - EF_{ij}$$

Propriedade: A folga livre é sempre menor ou igual que a folga total.

(i, j) é uma atividade crítica sse $ES_{ij} = LS_{ij}$, ou seja, tem folga total nula.

Observação: Noções idênticas para o modelo nó-atividade, com adaptações.

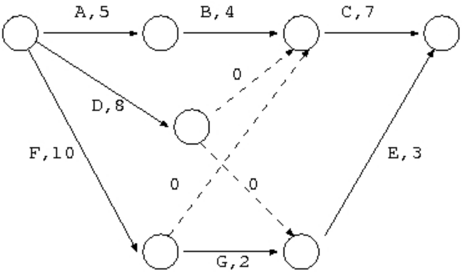
Exemplo 1

Qual é a duração mínima do projeto seguinte?

atividade	*duração (em dias)*	*precede*
A	5	B
B	4	C
C	7	-
D	8	C, E
E	3	-
F	10	C, G
G	2	E

Resposta: 17 dias

Modelo arco-atividade



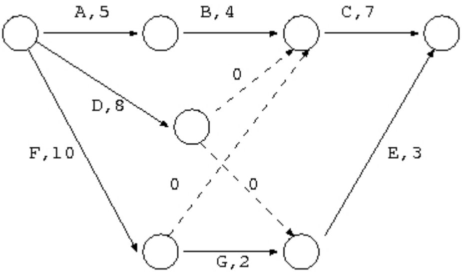
Exemplo 1

Qual é a duração mínima do projeto seguinte?

atividade	*duração (em dias)*	*precede*
A	5	B
B	4	C
C	7	-
D	8	C, E
E	3	-
F	10	C, G
G	2	E

Resposta: 17 dias

Modelo arco-atividade



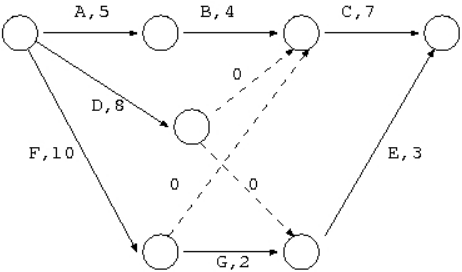
Exemplo 1

Qual é a duração mínima do projeto seguinte?

atividade	*duração (em dias)*	*precede*
A	5	B
B	4	C
C	7	-
D	8	C, E
E	3	-
F	10	C, G
G	2	E

Resposta: 17 dias

Modelo arco-atividade



A **duração mínima** do projeto é igual ao *comprimento do caminho máximo*.
No modelo arco-atividade, o *comprimento de um caminho* é a soma dos valores nos seus arcos. No modelo nó-atividade, é a soma dos valores nos seus nós. Em ambos os casos, esses valores representam as durações das atividades.

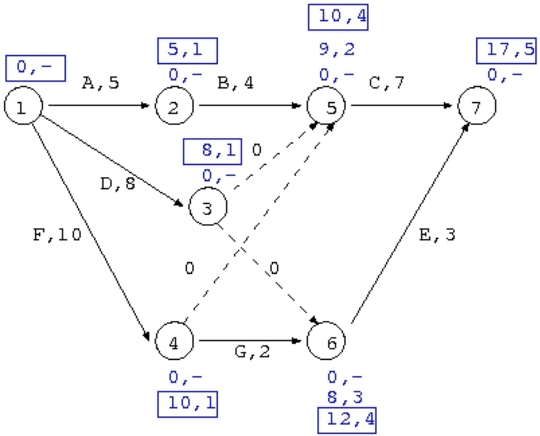
Método do Caminho Crítico (arco-atividade) – *earliest start*

Dado $G = (V, A, D)$, em que $D((i,j)) \in \mathbb{R}_0^+$ é a duração da tarefa $(i,j) \in A$, determinar a duração mínima do projeto e a data de início mais próxima para cada (i,j).

```
Para todo v ∈ G.V fazer ES[v] ← 0; Prec[v] ← Nenhum; GrauE[v] ← 0;
Para todo (v, w) ∈ G.A fazer GrauE[w] ← GrauE[w] + 1;
S ← {v ∈ G.V | GrauE[v] = 0};  /* S deve ser suportado por uma fila ou uma pilha. */
DurMin ← −1; v_f ← Nenhum;
Enquanto (S ≠ ∅) fazer
    v ← um qualquer elemento de S; S ← S \ {v};
    Se DurMin < ES[v] então DurMin ← ES[v]; v_f ← v;
    Para todo w ∈ G.Adjs[v] fazer
        Se ES[w] < ES[v] + D[(v, w)] então
            ES[w] ← ES[v] + D[(v, w)]; Prec[w] ← v;
        GrauE[w] ← GrauE[w] − 1;
        Se GrauE[w] = 0 então S ← S ∪ {w};
EscreveCaminho(v_f, Prec); escrever(DurMin);
```

O valor $ES[v]$ calculado pelo algoritmo é a data de início mais próxima **para as tarefas com início no nó** v, pelo que $ES_{ij} = ES[i]$, para cada $(i,j) \in A$.

Exemplo



$\boxed{\text{ES[v],Prec[v]}}$ indica o comprimento do caminho mais longo desde a origem até v e o nó que antecede v no caminho encontrado. As outras anotações nos nós são os valores em passos intermédios. $ES[v]$ é a data mais próxima para as tarefas com início em v.

Os nós estão numerados segundo a ordem de visita pelo algoritmo (que é uma ordem topológica).

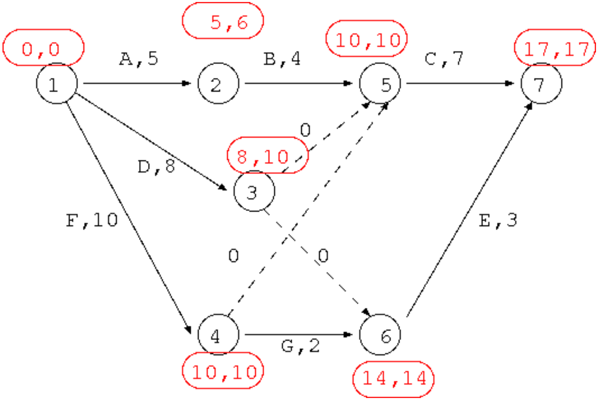
Método de Caminho Crítico - *Latest finish*

Obter a data de conclusão mais afastada para as tarefas com fim no nó v, para todo v, por **análise para trás** (*backward analysis*), fixando $LF[v]$ inicialmente como *DurMin* (a duração mínima do projeto).

```
Para todo v ∈ G.V fazer LF[v] ← DurMin; GrauS[v] ← 0;
Para todo (v, w) ∈ G.A fazer
    GrauS[v] ← GrauS[v] + 1;          // grau de saída
G^T ← grafo transposto de G;
S ← {v | GrauS[v] = 0};
Enquanto (S ≠ ∅) fazer
    v ← um qualquer elemento de S;
    S ← S \ {v};
    Para todo w ∈ G^T.Adjs[v] fazer
        Se LF[w] > LF[v] − D[(w, v)] então          // (w, v) em G.A
            LF[w] ← LF[v] − D[(w, v)];
        GrauS[w] ← GrauS[w] − 1;
        Se GrauS[w] = 0 então S ← S ∪ {w};
```

$LF_{ij} = LF[j]$, para todo (i, j).

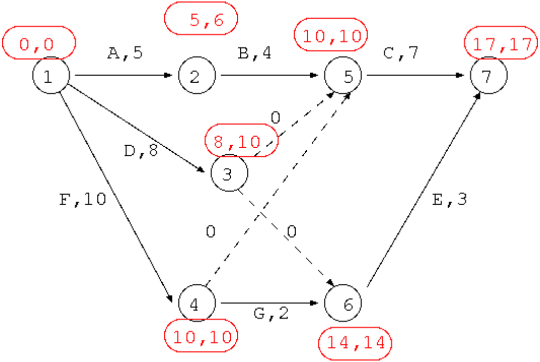
Exemplo (cont.)



As anotações representam $\boxed{ES[v], LF[v]}$, sendo $LF[v]$ a data de conclusão mais afastada para as tarefas que têm fim no nó v e $ES[v]$ a data de início mais próxima para as tarefas com início em v.

Exemplo (cont.)

As anotações representam $\boxed{ES[v], LF[v]}$.



	ES	LF	LS	FT	FL
A	0	6	1	1	0
B	5	10	6	1	1
C	10	17	10	0	0
D	0	10	2	2	2
E	12	17	14	2	2
F	0	10	0	0	0
G	10	14	12	2	0

$ES_{ij} = ES[i] \qquad LF_{ij} = LF[j]$

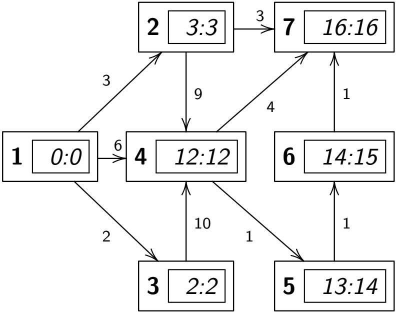
$LS_{ij} = LF_{ij} - d_{ij}$

$FT_{ij} = LS_{ij} - ES_{ij}$

$FL_{ij} = ES[j] - EF_{ij} = ES[j] - (ES_{ij} + d_{ij})$

Atividades críticas: C e F.

Exemplo 2



	ES	LS	FT	FL	EF	LF
$(1,2)$	0	0	0	0	3	3
$(1,3)$	0	0	0	0	2	2
$(1,4)$	0	6	6	6	6	12
$(2,4)$	3	3	0	0	12	12
$(2,7)$	3	13	10	10	6	16
$(3,4)$	2	2	0	0	12	12
$(4,5)$	12	13	1	0	13	14
$(4,7)$	12	12	0	0	16	16
$(5,6)$	13	14	1	0	14	15
$(6,7)$	14	15	1	1	15	16

Nós com anotação $\boxed{ES[v], LF[v]}$, sendo $ES[v]$ a data de início mais próxima para as tarefas que têm início no nó v e $LF[v]$ a data de conclusão mais afastada para as tarefas que têm fim em v.

Dois caminhos críticos: $((1,2),(2,4),(4,7))$ e $((1,3),(3,4),(4,7))$.

Propriedade: As atividades críticas são as que ocorrem em caminhos críticos.

Método do Caminho Crítico

Propriedades que suportam a correção dos métodos

- O grafo da relação de precedência é um DAG (grafo dirigido acíclico).
- A duração mínima do projeto é igual ao comprimento do caminho máximo no DAG.
- Qualquer DAG tem sempre algum vértice com grau de entrada zero.
- Se retirar alguma aresta a um DAG, obtém um DAG.
- O grafo transposto de um DAG é um DAG.

Nas referências sobre escalonamento sem partilha de recursos, o método descrito designa-se por **Método do Caminho Crítico** – *Critical path method* (CPM).

Escalonamento com partilha de recursos (extra aulas)

> "Constraint-based scheduling is one of the most successful application areas of CP. One of the key factors of this success lies in the fact that a combination was found of the best of two fields of research that pay attention to scheduling – namely, operations research (OR) and artificial intelligence (AI)." Cap. 2, Handbook of Constraint Programming, Elsevier, 2006
>
> https://doi.org/10.1016/S1574-6526(06)80026-X

Para saber mais...

- Course on Constraint Programming and Scheduling, by H.Rudová
  https://www.fi.muni.cz/~hanka/konstanz09/

Focaremos aqui o problema de minimização do número de trabalhadores.

Restrição Cumulative para Scheduling

Um projeto é constituído por tarefas. Cada uma requer um certo número de pessoas e tem uma duração. São dadas as precedências. As habilitações necessárias são idênticas para todas as tarefas. Determinar um calendário para as tarefas que **minimize o prazo de execução do projeto** e, **adicionalmente**, **minimize o número de trabalhadores a contratar**.

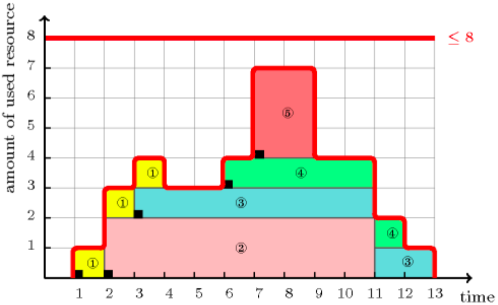


Ilustração de *Global Constraint Catalog* para a restrição global `cumulative`.

https://sofdem.github.io/gccat/gccat/Ccumulative.html

$$\forall t \quad \sum_{i:\ D_i \leq t \leq D_i + d_i} c_i \leq C$$

O número total de pessoas necessárias para as tarefas que estão a decorrer em cada instante não excede o número das existentes.

Restrição Cumulative para Scheduling

Por ter diversas aplicações, a restrição `cumulative` está disponível como **restrição global** nos sistemas de programação por restrições, e tem associados algoritmos específicos de propagação de consistência.

O módulo `library(ic_cumulative)` do sistema ECLiPSe CLP contém `cumulative(+StartTimes, +Durations, +Resources, ++ResourceLimit)`, sendo

- `StartTimes` as datas de início para as tarefas (variáveis `fd` ou inteiros)
- `Durations` as suas durações (variáveis `fd` ou inteiros)
- `Resources` os recursos que usam (variáveis `fd` ou inteiros)
- ResourceLimit limite máximo de recurso disponível (inteiro)

Esta restrição impõe que, em cada instante, a soma total dos recursos usados não excede ResourceLimit.

Aplicação à minimização de trabalhadores

Para um caso de teste, definido por predicado `tarefa(J,SegJ,DurJ,TrabJ)`

```
tarefa(1,[2,3],5,4).%a   tarefa(5,[],3,2). %f    tarefa(3,[],2,1).  %b
tarefa(4,[6],2,2). %c    tarefa(6,[],10,1). %e   tarefa(2,[],12,1). %d
```

os resultados foram

```
Ntrabs minimo para as tarefas criticas : 4
Tarefas criticas e suas datas de inicio
tarefa(1) : inicio(0)
tarefa(2) : inicio(5)
Ntrabs se as tarefas comecam na data mais proxima : 8
Found a solution with cost 4
Trabalhadores : 4
tarefa(1) : inicio(0)
tarefa(4) : inicio(5)
tarefa(5) : inicio(7)
tarefa(6) : inicio(7)
tarefa(3) : inicio(5)
tarefa(2) : inicio(5)
Datas = [0, 5, 7, 7, 5, 5]  Concl = 17  Ntrabs = 4
```

Incerteza: *"E se as durações exatas não são conhecidas?"*

Para informação... não será tratado na uc de Desenho de Algoritmos

Nem sempre as durações exatas são conhecidas. O método PERT (Program Evaluation and Review Technique) assume que as durações das tarefas são variáveis aleatórias, i.e., seguem distribuições de probabilidade.

- É necessário estimar as durações das atividades (análise estatística).
- Dadas as durações *otimista* (a, " tudo corre bem"), *pessimista* (b, " tudo corre mal") e *mais frequente* (m) para cada tarefa, assume que a duração d segue uma **distribuição β** (semelhante à distribuição normal, sendo nula a probabilidade da duração da atividade ser $> b$ e $< a$) com:

$$\begin{array}{lcl} \text{valor esperado } (P(d \leq \mu_d) = 0.5) & \mu_d = & \frac{a+4m+b}{6} \\ \text{desvio padrão} & \sigma_d = & \frac{b-a}{6} \end{array}$$

- Assume que as durações das tarefas são variáveis aleatórias independentes e que a **soma das durações das tarefas T num caminho crítico segue distribuição normal**, com média μ (dada pela soma das médias) e desvio padrão σ (soma dos desvios padrão): $\frac{T-\mu}{\sigma} \sim Normal(0,1)$.
- $\frac{T-\mu}{\sigma} \sim Normal(0,1)$ pode ser usado para: estimar a probabilidade de a duração do projeto ser menor ou igual ao valor calculado pelo CPM; indicar uma duração T que possa ser garantida com uma certa probabilidade, ...