1 Algoritmo del Banchiere

Il banker's algorithm è la generalizzazione della tecnica di deadlock-avoidance al caso di tipi di risorse con istanze multiple

- anche in questo caso i processi devono dichiarare inizialmente quali/quante risorse al massimo richiederanno
- all'atto della richiesta si può verificare una attesa prima di ottenere la risorsa
- una volta ottenute le risorse, queste devono essere rilasciate entro un tempo limitato

È stato proposto da E.Dijkstra e il nome deriva dal fatto che idealmente simula l'operato di un banchiere che deve mantenere una riserva di denaro per soddisfare tutte le future richieste

Tre punti da analizzare:

- Definizione delle strutture dati
- Fase di verifica della safeness
- Protocollo per la richiesta/concessione delle risorse

1.1 Algoritmo del Banchiere: Strutture Dati

Sia n il numero di processi e m il numero di tipi di risorse. Si definiscono quattro vettori/matrici:

- $available[m]$: l'elemento $available[j]$ indica il numero di istanze della risorsa di tipo j disponibili (istante per istante)
- $max[n, m]$: è la matrice di reclamo (claim matrix). L'elemento $max[i, j]$ indica il massimo numero di risorse di tipo j che il processo i può richiedere
- $allocation[n, m]$: l'elemento $allocation[i, j]$ indica il numero di risorse di tipo j che il processo i detiene (istante per istante)
- $need[n, m]$: l'elemento $need[i, j]$ indica il massimo numero di risorse di tipo j che il processo i potrebbe ancora necessitare (e quindi richiedere) per poter completare l'esecuzione

Si noti che *need* è ottenibile come differenza tra *max* e *allocation*.

1.2 Algoritmo del Banchiere: Verifica di Safeness

Questo è un test della safeness di uno stato: determina, se esiste, una sequenza safe.

Si definiscono $work[m]$ e $finish[n]$ come due vettori così inizializzati:

$$work := available \quad \text{e} \quad finish[i] := false \quad \text{per ogni } i$$

1. trova i (con qualsiasi algoritmo) tale che
   - $finish[i] == false$, e
   - $need_i \leqslant work$ (dove $need_i$ è la riga i-esima di $need$)

   se tale i non esiste vai al passo 3
2. Poni $work := work + allocation_i$ e $finish[i] := true$
   vai al passo 1
3. Se $finish[i] == true$ per ogni i allora lo stato è safe

La successione degli indici i trovati al passo 1 è la sequenza safe

1.3 Algoritmo del Banchiere: Richiesta Risorse

Per ogni processo P_i si impiega un vettore $request_i[m]$.
$request_i[j] = k$ indica che il P_i sta richiedendo k istanze di R_j
Se P_i opera delle richieste allora:

1. se $request_i \leqslant need_i$ vai al passo 2. Altrimenti **error**: il processo richiede più risorse del massimo previsto
2. se $request_i \leqslant available$ vai al passo 3. Altrimenti P_i deve attendere che si liberino delle risorse
3. simula l'assegnamento le risorse a P_i modificando lo stato:
$$\begin{aligned} available &:= available - request_i \\ allocation_i &:= allocation_i + request_i \\ need_i &:= need_i - request_i \end{aligned}$$
Ora determina se lo stato ottenuto è safe
   - se è safe concedi effettivamente le risorse a P_i
   - se non è safe P_i deve attendere; ripristina lo stato precedente

1.4 Esempio

Supponiamo vi siano 5 processi $(P_0, \ldots, P_4)$, 3 tipi di risorse: A con 10 instanze, B con 5 instanze, e C con 7 instanze. Supponiamo che al tempo T_0 si abbia:

Processo	allocation			max			available			need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2	7	4	3
P_1	2	0	0	3	2	2				1	2	2
P_2	3	0	2	9	0	2				6	0	0
P_3	2	1	1	2	2	2				0	1	1
P_4	0	0	2	4	3	3				4	3	1

Usando l'algoritmo di test della safeness si determina la sequenza safe $\langle P_1, P_3, P_4, P_2, P_0 \rangle$, quindi lo stato è safe

Giunge una richiesta di P_1: $(1, 0, 2)$. Verifichiamo se $request_1 \leqslant available$ (ovvero se $(1, 0, 2) \leqslant (3, 3, 2)$). Sì.
Il potenziale nuovo stato sarebbe:

Processo	allocation			max			available			need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	2	3	0	7	4	3
P_1	3	0	2	3	2	2				0	2	0
P_2	3	0	2	9	0	2				6	0	0
P_3	2	1	1	2	2	2				0	1	1
P_4	0	0	2	4	3	3				4	3	1

Usando l'algoritmo di test della safeness si determina la sequenza safe $\langle P_1, P_3, P_4, P_0, P_2 \rangle$, quindi lo stato è safe.
Le risorse possono essere concesse

Esercizio:
verificare se nello stato così raggiunto può essere soddisfatta

- una richiesta $(3, 3, 0)$ da parte di P_4, o
- una richiesta $(0, 2, 0)$ da parte di P_0

2 Algoritmo di Rilevamento del Deadlock

Similmente all'algoritmo del banchiere si utilizzano due vettori: $work[m]$ e $finish[n]$ come due vettori così inizializzati:

- $work := available$ e
- per ogni i, se $allocation[i] \neq 0$ allora $finish[i] := false$ altrimenti $finish[i] := true$

1. Trova un processo i tale che $finish[i] == false$ e $request[i] \leqslant work$.[1] Se tale processo non esiste vai al passo 3
2. Poni $work := work + allocation[i]$ e $finish[i] := true$.[2] Vai al passo 1
3. Se $finish[i] == false$, per qualche i, allora c'è un deadlock e il deadlock coinvolge il processo i.

N.B.: L'algoritmo necessita di ordine di $m \times n^2$ operazioni

2.1 Esempio

Supponiamo vi siano 5 processi $(P_0, \ldots, P_4)$, 3 tipi di risorse: A con 7 instanze, B con 2 instanze, e C con 6 instanze. Supponiamo che al tempo T_0 si abbia:

Processo	allocation			request			available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Il sistema non presenta un deadlock, infatti eseguendo l'algoritmo secondo la sequenza $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ si determina che $finish[i] == true$, per ogni i

Supponiamo ora che il processo P_2 richieda un'altra istanza di tipo C. La situazione cambia così:

Processo	allocation			request			available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Ora il sistema presenta un deadlock composto dai processi P_1, P_2, P_3 e P_4 (le risorse detenute da P_0 non basterebbero a bloccare la situazione)

3 Esercizi

3.1 Esercizio 1 (grafo di allocazione)

Si supponga che vi siano 7 processi, $P_1, \ldots, P_7$, e 6 tipi di risorse $R_1, \ldots, R_6$ tutte ad istanze singole, da utilizzare in mutua esclusione. Si imponga inoltre che non sia permessa la preemption di alcuna risorsa. La situazione del sistema può essere descritta come segue:

- P_1 detiene R_1 e richiede R_2

[1] Confronto tra vettori di m elementi

[2] Si assume il processo terminerà senza chiedere ulteriori risorse

- P_2 non detiene alcuna risorsa e richiede R_3
- P_3 non detiene alcuna risorsa e richiede R_2
- P_4 detiene R_4 e richiede sia R_2 sia R_3
- P_5 detiene R_3 e richiede R_5
- P_6 detiene R_6 e richiede R_2
- P_7 detiene R_5 e richiede R_4

Si determini, utilizzando il grafo di allocazione delle risorse, se il sistema è in deadlock e, in caso affermativo, quali sono i processi e le risorse coinvolti

3.2 Esercizio 2 (banker algorithm)

Supponiamo vi siano 4 processi $(P_1, \ldots, P_4)$, 3 tipi di risorse: A con 5 instanze, B con 8 instanze, e C con 16 instanze. Supponiamo che al tempo T_0 si abbia:

Processo	allocation			max		
	A	B	C	A	B	C
P_1	0	1	4	4	1	4
P_2	2	0	1	3	1	4
P_3	1	2	1	5	7	13
P_4	1	0	3	1	1	6

Si determini:

1. se il sistema è in uno stato safe
2. se l'assegnazione di una istanza della risorsa A al processo P_1 garantisca o meno il mantenimento dello stato safe
3. se l'assegnazione di 6 istanze della risorsa C al processo P_3 garantisca o meno il mantenimento dello stato safe

3.3 Esercizio 3 (deadlock detection)

Supponiamo vi siano 4 processi $(P_1, \ldots, P_4)$, 5 tipi di risorse $R_1, \ldots, R_5$ a istanza unica Supponiamo che al tempo T_0 si abbia:

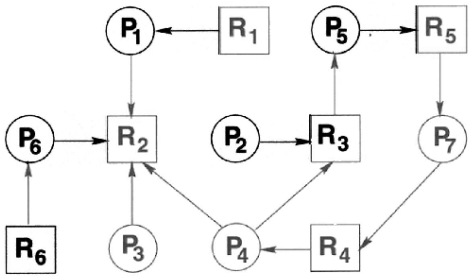
	allocation					request					available				
P_i	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5
P_1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1
P_2	1	1	0	0	0	0	0	1	0	1					
P_3	0	0	0	1	0	0	0	0	0	1					
P_4	0	0	0	0	0	1	0	1	0	1					

Determinare se sussiste deadlock e in caso quali processi coinvolge

4 Soluzioni

4.1 Soluzione Esercizio 1

Il grafo di allocazione è:



C'è un ciclo: $P_4 \rightarrow R_3 \rightarrow P_5 \rightarrow R_5 \rightarrow P_7 \rightarrow R_4 \rightarrow P_4$, quindi P_4, P_5 e P_7 sono in deadlock

4.2 Soluzione Esercizio 2–punto (1)

Processo	need			available		
	A	B	C	A	B	C
P_1	4	0	0	1	5	7
P_2	1	1	3			
P_3	4	5	12			
P_4	0	1	3			

Il sistema è in uno stato safe: una sequenza safe è: $\langle P_2, P_4, P_1, P_3 \rangle$

4.3 Soluzione Esercizio 2–punto (2)

Il nuovo stato, concedendo una istanza A al processo P_1, si ottiene aggiornando i vettori:

$$\begin{aligned} available &:= available - request_1 \\ allocation_1 &:= allocation_1 + request_1 \\ need_1 &:= need_1 - request_1 \end{aligned}$$

Si ottiene:

Processo	need			available		
	A	B	C	A	B	C
P_1	3	0	0	0	5	7
P_2	1	1	3			
P_3	4	5	12			
P_4	0	1	3			

Il sistema è in uno stato safe: una sequenza safe è: $\langle P_4, P_2, P_1, P_3 \rangle$

4.4 Soluzione Esercizio 2–punto (3)

Il nuovo stato, concedendo 6 istanze C al processo P_3, si ottiene aggiornando i vettori:

$$\begin{aligned} available &:= available - request_3 \\ allocation_3 &:= allocation_3 + request_3 \\ need_3 &:= need_3 - request_3 \end{aligned}$$

Si ottiene:

Processo	need			available		
	A	B	C	A	B	C
P_1	4	0	0	1	5	1
P_2	1	1	3			
P_3	4	5	6			
P_4	0	1	3			

Il sistema NON è in uno stato safe: nessun processo ha necessità residue che possano essere soddisfatte

4.5 Soluzione Esercizio 3

Supponiamo vi siano 4 processi $(P_1, \ldots, P_4)$, 5 tipi di risorse $R_1, \ldots, R_5$ a istanza unica Supponiamo che al tempo T_0 si abbia:

	allocation					request					available				
P_i	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5
P_1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1
P_2	1	1	0	0	0	0	0	1	0	1					
P_3	0	0	0	1	0	0	0	0	0	1					
P_4	0	0	0	0	0	1	0	1	0	1					

Determinare se sussiste deadlock e in caso quali processi coinvolge

RISPOSTA:

c'è una situazione di deadlock che coinvolge i processi P_1 e P_2