

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

```
#define MAXPAROLA 30
#define MAXRIGA 80
```

```
int main(int argc, char *argv[])
```

```
{
    int freq[MAXPAROLA]; /* vettore di contatori
    delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;
```

```
for(i=0; i<MAXPAROLA; i++)
    freq[i]=0;
```

```
if(argc != 2)
```

```
{
    printf(stderr, "ERRORE, serve un parametro con il nome del file\n");
    exit(1);
}
```

```
f = fopen(argv[1], "r");
if(f==NULL)
```

```
{
    printf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
    exit(1);
}
```

```
while( fgets( riga, MAXRIGA, f ) != NULL )
```



Stallo di processi

Tecniche per evitare uno stallo

Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

Evitare il deadlock

- ❖ Le tecniche per **evitare** condizioni di stallo
 - Forzano i P a fornire (a priori) informazioni aggiuntive sulle richieste che effettueranno nel corso della loro esistenza
 - Ogni P deve indicare quali e quante risorse di ogni tipo saranno necessarie per terminare il suo compito
 - Tali informazioni permetteranno di schedare i processi in modo che non si verifichino deadlock
 - Se l'esecuzione di un processo può causare deadlock essa viene ritardata opportunamente

Evitare il deadlock

❖ I principali algoritmi

- Si differenziano per la quantità e il tipo di informazioni richieste
 - Il modello più semplice si basa sul forzare tutti i processi a dichiarare il massimo numero di risorse di ciascun tipo di cui il processo avrà bisogno
- In genere provocano una riduzione nell'utilizzo delle risorse e una minore efficienza del sistema
- Si basano sul concetto di **stato sicuro** e di **sequenza sicura**

Stato sicuro

Uno stato si dice **non sicuro** in caso contrario.

Stato sicuro

Il sistema è in grado di

- Allocare le risorse richieste a tutti i processi in esecuzione
- Impedire il verificarsi di uno stallo
- Trovare una sequenza sicura

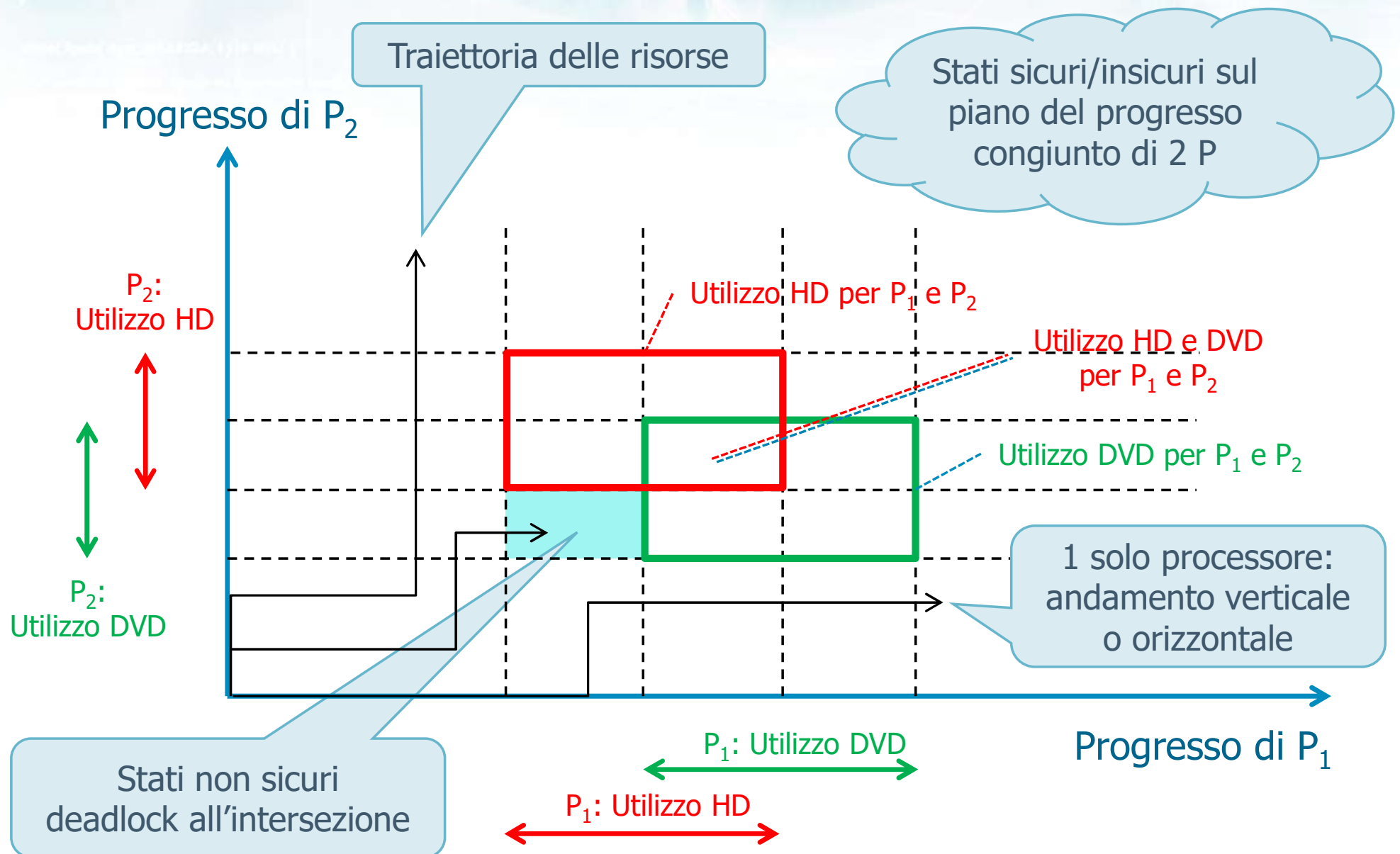
Sequenza sicura

Una sequenza di schedulazione dei processi $\{P_1, P_2, \dots, P_n\}$ tale che per ogni P_i le richieste che esso può ancora effettuare possono essere soddisfatte impiegando le risorse attualmente disponibili più le risorse liberate dai processi P_j con $j < i$



Non necessariamente uno stato non sicuro è uno stato di stallo: condurrà in uno stato di stallo in caso di comportamento standard. Inoltre non è detto le richieste "massime" vengano raggiunte.

Progresso congiunto di due processi



Strategie

- ❖ Per evitare uno stallo ci si assicura che il sistema rimanga sempre in uno stato sicuro
 - All'inizio il sistema è in uno stato sicuro
 - Ogni richiesta di nuova risorsa
 - Sarà soddisfatta se lascerà il sistema in uno stato sicuro
 - Sarà ritardata in caso contrario; il processo che ha effettuato la richiesta sarà posto in attesa
- ❖ Esistono due classi di strategie
 - Con risorse aventi istanze unitarie
 - Con risorse aventi istanze multiple

Algoritmo per istanze unitarie

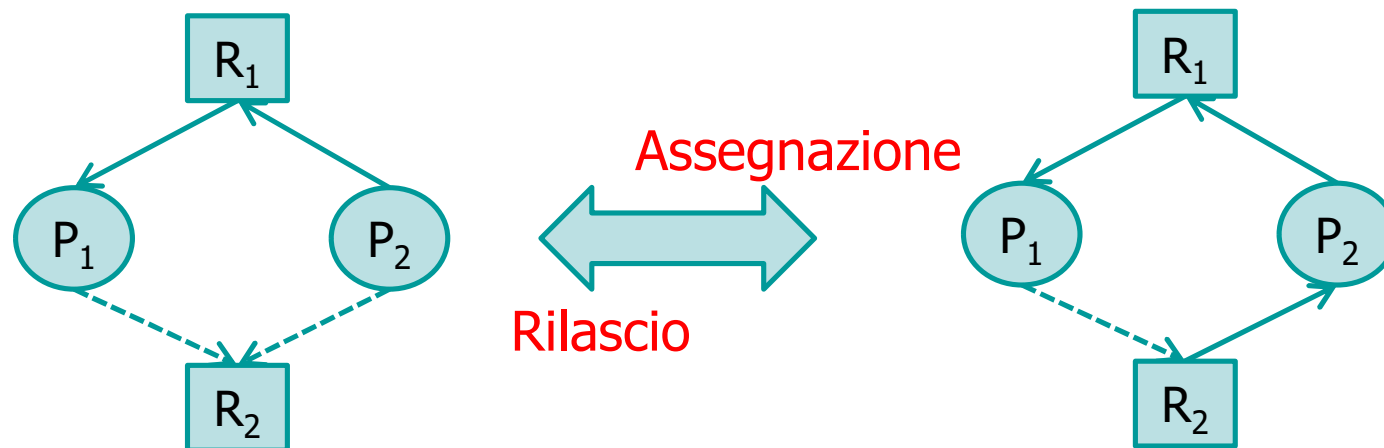
- ❖ Sono basate sulla determinazione di cicli, gestendo il grafo di rivendicazione
 - Tutte le richieste che saranno effettuate **devono** essere dichiarate a priori, ovvero all'inizio dell'esecuzione
 - Esse sono indicate come archi di reclamo sul grafo di rivendicazione
 - Nel momento in cui una richiesta viene effettuata
 - Il corrispondente arco di reclamo **dovrebbe** essere trasformato in un arco di assegnazione
 - Prima di effettuare tale trasformazione si verifica però se soddisfacendola si generano dei cicli

Algoritmo per istanze unitarie

- Se il grafo non presenterà comunque alcun ciclo, la conversione viene effettuata e la risorsa assegnata
- In caso negativo, l'assegnazione della risorsa richiesta porterebbe il sistema in uno stato non sicuro e quindi viene rimandata

➤ Quando una risorsa viene rilasciata

- L'arco di assegnazione ritorna a essere un arco di reclamo (per gestire eventuali richieste successive)



Algoritmo per istanze multiple

- ❖ Valutano lo stato del sistema per comprendere se le risorse disponibili sono sufficienti per portare a termine tutti i P
 - Si basano sul numero di risorse disponibili, assegnate e massimo richiesto
- ❖ Ogni processo
 - Deve dichiarare a priori il massimo uso di risorse
 - Quando richiede una risorsa può essere bloccato
 - Quando ottiene le risorse che gli servono deve garantire che le restituirà in un tempo finito

Algoritmo per istanze multiple

- ❖ Algoritmo del Banchiere (Dijkstra, [1965])
 - È costituito da **due** sezioni
 - La prima verifica che uno stato sia sicuro
 - La seconda verifica che una nuova richiesta possa essere soddisfatta lasciando il sistema in uno stato sicuro
- ❖ L'algoritmo utilizza le strutture dati elencate di seguito

Algoritmo per istanze multiple

Siano dati:

- Un insieme di processi P_r con cardinalità n
- Un insieme di risorse R_c con cardinalità m

Nome	Dim.	Contenuto e significato
Fine	$[n]$	$\text{Fine}[r]=\text{false}$ indica che P_r non ha terminato
Assegnate	$[n][m]$	$\text{Assegnate}[r][c]=k$ P_r possiede k istanze di R_c
Massimo	$[n][m]$	$\text{Massimo}[r][c]=k$ P_r può richiedere al massimo k istanze di R_c
Necessità	$[n][m]$	$\text{Necessità}[r][c]=k$ P_r ha bisogno di altre k istanze di R_c $\forall i \forall j \text{ Necessità}[i][j] = \text{Massimo}[i][j] - \text{Assegnate}[i][j]$
Disponibili	$[m]$	$\text{Disponibili}[c]=k$ disponibilità pari a k per R_c

Esempio: Step 1

- ❖ Applicando l'algoritmo del banchiere il sistema sottostante si trova in uno stato sicuro?

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		R ₀ R ₁ R ₂	R ₀ R ₁ R ₂	R ₀ R ₁ R ₂	R ₀ R ₁ R ₂
P ₀	F	0 1 0	7 5 3		3 3 2
P ₁	F	2 0 0	3 2 2		
P ₂	F	3 0 2	9 0 2		
P ₃	F	2 1 1	2 2 2		
P ₄	F	0 0 2	4 3 3		

Soluzione

❖ Applicando l'algoritmo del banchiere il sistema sottostante si trova in uno stato sicuro?

➤ Sì; sequenza sicura: P_1, P_3, P_0, P_2, P_4

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	T	0 1 0	7 5 3	7 4 3	3 3 2
P_1	T	2 0 0	3 2 2	1 2 2	5 3 2
P_2	T	3 0 2	9 0 2	6 0 0	7 4 3
P_3	T	2 1 1	2 2 2	0 1 1	7 5 3
P_4	T	0 0 2	4 3 3	4 3 1	10 5 5
					10 5 7

Esempio: Step 2

❖ La richiesta di P_1 (1, 0, 2) può essere soddisfatta?

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	F	0 1 0	7 5 3	7 4 3	3 3 2
P_1	F	2 0 0	3 2 2	1 2 2	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Soluzione: Check A

- ❖ La richiesta di P_1 (1, 0, 2) può essere soddisfatta?
- Sì; nuovo stato del sistema; il nuovo stato è sicuro?

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	F	0 1 0	7 5 3	7 4 3	3 3 2 2 3 0
P_1	F	2 0 0 3 0 2	3 2 2	1 2 2 0 2 0	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Soluzione: Nuovo stato

❖ Il nuovo stato è sicuro?

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		R ₀ R ₁ R ₂	R ₀ R ₁ R ₂	R ₀ R ₁ R ₂	R ₀ R ₁ R ₂
P ₀	F	0 1 0	7 5 3	7 4 3	2 3 0
P ₁	F	3 0 2	3 2 2	0 2 0	
P ₂	F	3 0 2	9 0 2	6 0 0	
P ₃	F	2 1 1	2 2 2	0 1 1	
P ₄	F	0 0 2	4 3 3	4 3 1	

Soluzione: Check B

❖ Il nuovo stato è sicuro?

➤ Sì; sequenza sicura: P_1, P_3, P_0, P_4, P_2

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	T	0 1 0	7 5 3	7 4 3	2 3 0
P_1	T	3 0 2	3 2 2	0 2 0	5 3 2
P_2	T	3 0 2	9 0 2	6 0 0	7 4 3
P_3	T	2 1 1	2 2 2	0 1 1	7 5 3
P_4	T	0 0 2	4 3 3	4 3 1	10 5 5
					10 5 7

The diagram shows the sequence of processes that can complete safely: P_1, P_3, P_0, P_4, P_2 . Red dashed arrows point from the 'Disponibili' column of each process to the next process in the sequence, indicating that the resources released by one process are sufficient to start the next.

Esercizio

Stesso stato iniziale

❖ La richiesta $P_4 (3, 3, 0)$ può essere soddisfatta?

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	F	0 1 0	7 5 3	7 4 3	2 3 0
P_1	F	3 0 2	3 2 2	0 2 0	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Esercizio

Stesso stato iniziale

- ❖ La richiesta $P_4 (3, 3, 0)$ può essere soddisfatta?
- No; non c'è disponibilità

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	F	0 1 0	7 5 3	7 4 3	2 3 0 ? 0 0
P_1	F	3 0 2	3 2 2	0 2 0	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2 3 3 2	4 3 3	4 3 1 1 0 1	

Esercizio

Stesso stato iniziale

❖ La richiesta $P_0 (0, 3, 0)$ può essere soddisfatta?

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	F	0 1 0	7 5 3	7 4 3	2 3 0
P_1	F	3 0 2	3 2 2	0 2 0	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Esercizio

Stesso stato iniziale

- ❖ La richiesta P_0 (0, 3, 0) può essere soddisfatta?
 - No; lo stato risultante non è sicuro

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	F	0 1 0 0 4 0	7 5 3	7 4 3 7 1 3	2 3 0 2 0 0
P_1	F	3 0 2	3 2 2	0 2 0	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Algoritmo del banchiere

❖ Verifica che uno stato sia sicuro

verifica_stato ()

1. $\forall i \forall j$ $Necessita[i][j] = Massimo[i][j] - Assegnate[i][j]$
 $\forall i$ $Fine[i] = false$
2. Trova i per cui
 $Fine[i] = false$ AND $\forall j$ $Necessità[i][j] \leq Disponibili[j]$
Se tale i non esiste goto step 4
3. $\forall j$ $Disponibili[j] = Disponibili[j] + Assegnate[i][j]$
 $Fine[i] = true$
goto step 2
4. Se $(\forall i$ $Fine[i] = true)$ allora
 return (SICURO)
altrimenti
 return (NON SICURO)

Algoritmo del banchiere

❖ Verifica una richiesta sia soddisfacibile

verifica_richiesta ()

se

$\forall_j \text{Richieste}[i][j] \leq \text{Necessità}[i][j]$

AND

$\forall_j \text{Richieste}[i][j] \leq \text{Disponibili}[j]$

allora

$\forall_j \text{Disponibili}[j] = \text{Disponibili}[j] - \text{Richieste}[i][j]$

$\forall_j \text{Assegnate}[i][j] = \text{Assegnate}[i][j] + \text{Richieste}[i][j]$

$\forall_j \text{Necessità}[i][j] = \text{Necessità}[i][j] - \text{Richieste}[i][j]$

se (**verifica_stato()** == SICURO) allora

conferma tale assegnazione

altrimenti

ripristina lo stato precedente

Esercizio

- ❖ La richiesta $P_1 (1, 0, 1)$ può essere soddisfatta?
 - Si ...
- ❖ La richiesta $P_2 (1, 0, 1)$ può essere soddisfatta?
 - No ... lo stato risultante non è sicuro

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$	$R_0 \ R_1 \ R_2$
P_0	F	1 0 0	3 2 2		1 1 2
P_1	F	5 1 1	6 1 3		
P_2	F	2 1 1	3 1 4		
P_3	F	0 0 2	4 2 2		

Esercizio

❖ I seguenti due stati sono sicuri o non sicuri?
(problemi risorsa unica)

P	F	A	M	N	D
P ₀	F	3	9		3
P ₁	F	2	4		
P ₂	F	2	7		

... stato sicuro

P	F	A	M	N	D
P ₀	F	4	9		2
P ₁	F	2	4		
P ₂	F	2	7		

... stato non sicuro

Algoritmo del banchiere

- ❖ La complessità dell'algoritmo del banchiere è
 - $O(m \cdot n^2) = O(|R| \cdot |P|^2)$
- ❖ Si basa inoltre su ipotesi poco realistiche
 - I processi devono indicare le richieste in anticipo
 - Le risorse necessarie non sempre sono note
 - Inoltre non è noto quando saranno necessarie
 - Suppone le risorse siano in numero costante
 - Le risorse possono aumentare o ridursi a causa di guasti temporanei o duraturi
 - Richiede una popolazione fissa di processi
 - I processi attivi nel sistema aumentano e si riducono in maniera dinamica