

Esercizi

Esercizio 1

Exercise (25/06/2018):

Si consideri la seguente sequenza di riferimenti a pagine in memoria: **577517111434123**.

Si utilizzi un algoritmo di sostituzione pagine di tipo **working-set** (versione esatta) con finestra $\Delta = 3$, assumendo che siano disponibili al **massimo 3 frame**. Determinare **quali e quanti page fault** (accessi a pagine non presenti nel resident set) e **page out** (rimozioni di pagine dal resident set) si verificheranno. Si richiede la visualizzazione (dopo ogni accesso) del resident set.

Riferimenti	5	7	7	5	1	7	1	1	1	4	3	4	1	2	3
Resident Set															
Page Fault															
Page Out															

Riferimenti	5	7	7	5	1	7	1	1	1	4	3	4	1	2	3
-	5	5	5	5											
-	7	7	7	7											
Resident Set															
Page Fault	X	X													
Page Out															

Vediamo i primi passi: all'inizio abbiamo i 3 frames liberi.

Riferimento a page 5: non è presente, la carichiamo in seguito ad un page fault.

Riferimento a page 7: non è presente, la carichiamo in seguito ad un page fault.

Riferimento a page 7: è presente, non c'è page fault.

Notiamo che allo stato attuale il working-set è

577 (le ultime 3 pagine referenziate).

Riferimenti	5	7	7	5	1	7	1	1	1	4	3	4	1	2	3
Resident Set	5	5	5	5	5	5									
	7	7	7	7	7	7	7								
					1	1	1								
Page Fault	X	X			X										
Page Out						X									

Vediamo che qui però, essendo il working-set diventato **171**, dobbiamo fare un **page-out** su 5, che appunto non fa più parte del working-set e quindi eliminiamo dalla memoria.

E così via fino alla fine:

Riferimenti	5	7	7	5	1	7	1	1	1	4	3	4	1	2	3
Resident Set	5	5	5	5	5	5				4	4	4	4	4	3
	7	7	7	7	7	7	7	7		3	3	3	3	2	2
					1	1	1	1	1	1	1	1	1	1	1
Page Fault	X	X			X					X	X		X	X	X
Page Out						X		X			X		X	X	X

$$\text{PF} = 8$$

$$\text{PO} = 5$$

In totale abbiamo 8 PF e 5 PO.

Esercizio 2

Exercise (25/06/2018):

Si vuole inoltre definire una misura di località del programma svolto, basato sulla "Reuse Distance". La Reuse Distance al tempo T_i (RD_i), in cui si accede alla pagina P_i , viene definita come il numero di pagine (distinte e diverse da P_i) a cui si è fatto accesso a partire dal precedente accesso a P_i (assumendo, convenzionalmente, per il primo accesso a una pagina, il numero totale di pagine cui si è fatto accesso sino a quell'istante). Ad esempio, al tempo 3, in cui si fa il secondo accesso alla pagina 5, $RD_3 = 1$, in quanto tra i due accessi alla pagina 5 si è fatto accesso, due volte, a una sola pagina (7). Dati i vari RD_i , se ne calcoli il valor medio RD_{avg} . La località del programma svolto viene definita come $L = 1 / (1 + RD_{avg})$. Si calcolino i valori RD_i , RD_{avg} e L .

Riferimenti	5	7	7	5	1	7	1	1	1	4	3	4	1	2	3
Resident Set	5	5	5	5	5	5				4	4	4	4	4	3
		7	7	7	7	7	7	7			3	3	3	2	2
					1	1	1	1	1				1	1	1
Page Fault	X	X			X					X	X		X	x	x
Page Out						X			X			X		X	X
RD															

Basta leggere la domanda per capire cosa bisogna fare, il risultato è:

Riferimenti	5	7	(7)	5	1	7	1.	1	1	4	3	4	1	2	3
Resident Set	5	5	5	5	5	5				4	4	4	4	4	3
		7	7	7	7	7	7	7			3	3	3	2	2
					1	1	1	1	1				1	1	1
Page Fault	X	X			X					X	X		X	x	x
Page Out						X			X			X		X	X
RD	0	1	0	1	2	2	1	0	0	3	4	1	2	5	3

Esercizio 3

Exercise (24/07/2019):

Sia data la stringa di riferimenti a pagine $3, 4, 1, (3, 1, 4, 4, 3, 1, 1)*10$, in cui la sintassi (...) n indica che la stringa tra parentesi viene ripetuta/iterata n volte (la stringa può ad esempio, derivare da un costrutto iterativo).

Si utilizzi un algoritmo di sostituzione pagine di **tipo working-set** (versione esatta) con finestra di durata $\Delta = 3$. Si supponga di denominare page-out la rimozione di una pagina dal resident set (in quanto esce dal working set). Si visualizzino nello schema che segue i riferimenti e il resident set dopo ogni riferimento, indicando i page-fault (accessi a pagine non presenti nel resident set) e i page-out. Si richiede la visualizzazione solo fino alle prime due iterazioni della sotto-stringa ripetuta 10 volte. ATTENZIONE: ogni riga del resident set rappresenta un frame, quindi una pagina presente in un frame non può cambiare riga quando rimane nel resident set. Nel caso di page-fault senza page-out, si utilizzi il primo frame libero dall'alto. Nel caso di page-out e (contemporaneo) page-fault, viene riutilizzato il frame appena liberato (da page-out). Qualora page-out e page-fault siano relativi alla stessa pagina, l'algoritmo di sostituzione ne tiene conto, evitando sia page-out che page-fault.

Exercise (24/07/2019):

Sia data la stringa di riferimenti a pagine $3, 4, 1, (3, 1, 4, 4, 3, 1, 1)*10$, in cui la sintassi (...) n indica che la stringa tra parentesi viene ripetuta/iterata n volte. Si utilizzi un algoritmo di sostituzione pagine di tipo working-set (versione esatta) con **finestra di durata $\Delta = 3$** .

Tempo	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Riferimenti	3	4	1	3	1	4	4	3	1	1	3	1	4	4	3	1	1
	3	3	3	3	3	3		3	3	3	3	3	3		3	3	3
Resident Set		4	4	4		4	4	4	4				4	4	4	4	
		1	1	1	1	1			1	1	1	1	1	1		1	1
Page Fault	X	X	X			X		X	X				X		X	X	
Page Out					X		X	X		X				X	X		X

Esercizio 4

Altro esercizio, parte A)

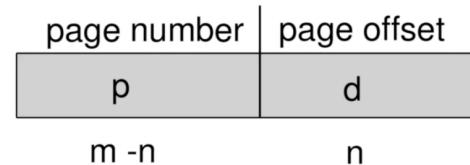
Exercise (05/07/2019):

A) Si consideri la seguente sequenza di riferimenti in memoria nel caso di un programma in cui, per ogni accesso (indirizzi in esadecimale, si indirizza il Byte), si indica se si tratta di lettura (R) o scrittura (W): **R 3F5, R 364, W 4D3, W 47E, R 4C8, W 2D1, R 465, W 2A0, R 3BA, W 4E6, R 480, R 294, R 0B8, R 14E**.

Supponendo che **sia indirizzi fisici** che logici siano su **12 bit**, che si usi paginazione con pagine di dimensione **128 Byte** e che il massimo indirizzo utilizzabile dal programma sia **C10**, si dica **quante pagine sono presenti nello spazio di indirizzamento** del programma e se ne **calcoli la frammentazione interna**.

Address Translation Scheme

- Address generated by CPU is divided into:
 - **Page number (p)** – used as an index into a **page table** which contains base address of each page in physical memory
 - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit



- For given logical address space 2^m and page size 2^n
- This information is used by Page Table

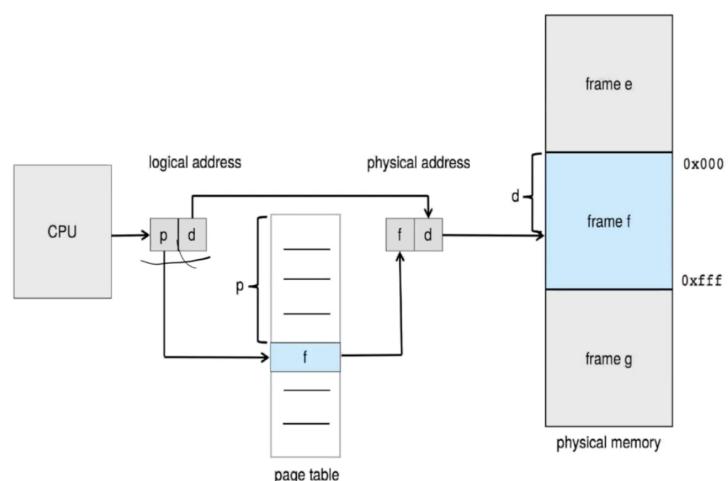
Abbiamo detto che un indirizzo, creato dalla CPU, viene separato in 2: page offset e page number.

Quanti bit sono dedicati all'una e l'altra parte dipende da dimensione massima della pagina e massimo numero di pagine che possiamo avere nel sistema.

Paging Hardware

- The steps taken by the MMU to translate a logical address generated by CPU to a physical address:

1. Extract the page number P and use it as an index into the page table.
2. Extract the corresponding frame number F from the page table.
3. Replace the page number P in the logical address with the frame number.



Ricordiamo qui il meccanismo per ricavare l'indirizzo fisico a cui troviamo la page referenziata dalla CPU.

Torniamo all'esercizio..

Abbiamo page size da 128B, dunque

$$128B = 2^7 \Rightarrow 7 \text{ bit di offset.}$$

L'ultimo accesso in memoria è

$$\mathbf{C10} = \boxed{1100 \ 0001 \ 0000}.$$

Avendo 7 bit per l'offset si ha quindi:

$\boxed{11000 \ 0010000}$, da cui possiamo estrarre il page number: $\boxed{11000} = 8 + 16 = 24$.

Sapendo quindi che 7 bit sono dedicati all'offset, i rimanenti 5 bit sono dunque dedicati al page number. Il che vuol dire che il numero totale di pagine indirizzabili è: $2^5 = 32$.

Ci potevamo arrivare anche sapendo che:

- page size = **128 Byte**
- indirizzi logici (e in questo caso anche fisici) a **12 bit**
- ogni indirizzo referenzia **1 Byte**

Si ha dunque che: $2^{12} = 4096$ (4096 Byte indirizzati) $\rightarrow 4096B / 128B = 32$ (4096B in totale raggruppati a 128B, che è il page size).

Però (*ho cappato io*), il testo dell'esercizio vuole sapere di quante pagine in totale dispone il programma, e sapendo che il massimo indirizzo utilizzabile dal programma è **C10**, che è uguale a 24, ciò vuol dire che il programma ha in totale **25 pagine** (da 0 a 24).

Abbiamo quindi calcolato, come richiesto,
quante pagine sono presenti nello spazio di indirizzamento del programma.

Vediamo ora di calcolarne *la frammentazione interna*.



Ricordiamo: ho frammentazione interna quando, come ad esempio in questo caso, ho pagine da 128B, ma magari il programma ha bisogno di allocare **meno** di 128B di memoria. Dunque gli viene assegnato uno spazio da 128B, ma non lo usa tutto.

Sappiamo che il programma ha usato fino alla pagina 24, ma di questa pagina sono stati usati tutti i 128B?

Prendiamo dunque l'indirizzo della page 24: **C10**.

Si ha che il dato referenziato si trova a 17 Bytes (da 0 a 16) dall'inizio della page (ricorda che C10 = `11000 0010000`, di cui 7 bit di offset, che in questo caso `0010000 = 16`).

Dunque abbiamo

$128B - 17B = 111B$ di **internal fragmentation**.

Exercise (05/07/2019):

3(a). Si consideri la seguente sequenza di riferimenti in memoria nel caso di un programma in cui, per ogni accesso (indirizzi in **esadecimale**, si indirizza il Byte), si indica se si tratta di lettura (R) o scrittura (W): **R 3F5, R 364, W 4D3, W 47E, R 4C8, W 2D1, R 465, W 2A0, R 3BA, W 4E6, R 480, R 294, R 0B8, R 14E**.

Supponendo che sia **indirizzi fisici** che logici siano su **12 bit**, che si usi paginazione con pagine di dimensione **128 Byte** e che il massimo indirizzo utilizzabile dal programma sia **C10**, si dica **quante pagine sono presenti nello spazio di indirizzamento** del programma e se ne **calcoli la frammentazione interna**.

C10 >> 1100 0001 0000
Page size= 128 byte (2^7) >> page offset = 7 bits

C10 >>

11000	0010000
-------	---------

Page Number Page Offset
Max Page number = $2^3+2^4= 24$

Fragmentation:

Last page is filled until $0010000 = 2^4 = 16$ bytes ($0 >> 16 = 17$ bytes)

Fragmentation >> page size (128) – used one (17 bytes) = 111 bytes

Stesso esercizio, parte **B**)

Exercise (05/07/2019):

B) Si determini la stringa dei riferimenti a pagine (Si consiglia di passare da esadecimale a binario, per determinare correttamente il numero di pagina e, se necessario, il displacement/offset). Si utilizzi un algoritmo di sostituzione pagine di tipo **LRU** (Least Recently Used). Si assuma che siano disponibili 3 frame, agli indirizzi fisici (espressi in esadecimale) **780, A00, B00**. Si richiede la visualizzazione (dopo ogni accesso) del resident set (i frame fisici contenenti pagine logiche).

Determinare quali e quanti page fault (accessi a pagine non presenti nel resident set) si verificheranno. Si dica infine a quali indirizzi fisici vengono effettuati gli accessi (tra quelli sopra elencati) R 3F5, W 4D3, R 3BA

Exercise (05/07/2019):

3(b). Si determini la stringa dei riferimenti a pagine (Si consiglia di passare da esadecimale a binario, per determinare correttamente il numero di pagina e, se necessario, il displacement/offset). **R 3F5, R 364, W 4D3, W 47E, R 4C8, W 2D1, R 465, W 2A0, R 3BA, W 4E6, R 480, R 294, R 0B8, R 14E**. Supponendo che sia **indirizzi fisici** che logici siano su **12 bit**, che si usi paginazione con pagine di dimensione **128 Byte**

3F5	00111110101	
364	001101100100	
4D3	010011010011	
47E	010001111110	
4C8	010011001000	
2D1	001011010001	
465	010001100101	
2A0	001010100000	
3BA	001110111010	
4E6	010011100110	
480	010010000000	
294	001010010100	
0B8	000010111000	
14E	000101001110	

Abbiamo quindi 7 bit di offset e 5 di page number, dunque facile trovare le pagine corrispondenti:

3F5 ↗	00111 110101	7
364	00110 1100100	6 .
4D3	01001 1010011	9
47E	01000 1111110	8
4C8	01001 1001000	9
2D1	00101 1010001	5
465	01000 1100101	8
2A0	00101 0100000	5
3BA	00111 0111010	7
4E6	01001 1100110	9
480	01001 0000000	9
294	00101 0010100	5
0B8	00001 0111000	1
14E	00010 1001110	2

Dobbiamo portare queste pagine in memoria, e nel caso in cui il programma faccia riferimento ad una page non presente in memoria, ho un page fault:

Exercise (05/07/2019):

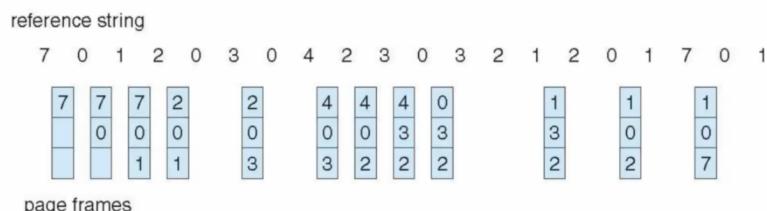
B) Si determini la stringa dei riferimenti a pagine (Si consiglia di passare da esadecimale a binario, per determinare correttamente il numero di pagina e, se necessario, il displacement/offset). Si utilizzi un algoritmo di sostituzione pagine di tipo **LRU** (Least Recently Used). Si assuma che siano disponibili 3 frame, agli indirizzi fisici (espressi in esadecimale) **780, A00, B00**. Si richiede la visualizzazione (dopo ogni accesso) del resident set (i frame fisici contenenti pagine logiche).

Determinare quali e quanti page fault (accessi a pagine non presenti nel resident set) si verificheranno. Si dica infine a quali indirizzi fisici vengono effettuati gli accessi (tra quelli sopra elencati) R 3F5, W 4D3, R 3BA

Riferimenti		7	6	9	8	9	5	8	5	7	9	9	5	1	2
Resident Set	780														
	A00														
	B00														
Page Fault															

Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
 - Replace page that has not been used in the most amount of time
 - Associate time of last use with each page
 - The main difference between the FIFO and OPT algorithms is that FIFO uses the time when a page was brought into memory, OPT uses the time when a page is to be used.



Ricordiamo come funziona LRU.

Alla fine si ha:

Riferimenti		7	6	9	8	9	5	8	5	7	9	9	5	1	2
Resident Set	780	7	7	7	8	8	8	8	8	9	9	9	9	9	2
	A00		6	6	6	6	5	5	5	5	5	5	5	5	5
	B00			9	9	9	9	9	9	7	7	7	7	1	1
Page Fault		X	X	X	X		X			X	X			X	X

In totale abbiamo 9 PF.

Exercise (05/07/2019):

3(d). Si dica infine a quali indirizzi fisici vengono effettuati gli accessi (tra quelli sopra elencati) R 3F5, W 4D3, R 3BA

Page Address	Page Address (binary)	Page Number	Frame Address	Frames Physical Address	Page Physical Address
3F5	00111 1110101	7			
4D3	01001 1010011	9			
3BA	00111 0111010	7			

Riferimenti		7 (3F5)	6 (364)	9 (4D3)	8 (47E)	9 (4C8)	5 (2D1)	8 (465)	5 (2A0)	7 (3BA)	9 (4E6)	9 (480)	5 (294)	1 (0B8)	2 (14E)
Resident Set	780	7	7	7	8		8			8	9			9	2
	A00		6	6	6		5			5	5			5	5
	B00			9	9		9			7	7			1	1
Page Fault		X	X	X	X		X			X	X			X	X

Dobbiamo ricavare il frame number dove viene posizionata la page 7, che ha indirizzo logico 3F5.

L'offset è uguale, ma cambia la parte dedicata al page number (ovviamente).

Dunque, dato che facendo l'esercizio sappiamo che la page 7 viene messa all'indirizzo fisico 780, ricaviamo da questo indirizzo il frame number:

$$780 = 0111\ 1000\ 0000 \rightarrow \underline{01111}\ \underline{0000000} \rightarrow \text{Frame number} = 01111 = 15$$

Sostituiamo alla parte del page number nell'indirizzo logico della page 7: 00111 1110101 → 01111 1110101 ⇒ 7F5

Dunque in conclusione:

Page Address	Page Address (binary)	Page Number	Frame Address	Frames Physical Address	Page Physical Address
3F5	00111 1110101	7	780 ↙	01111 0000000	01111 1110101
4D3	01001 1010011	9	B00 ↙	10110 0000000	10110 1010011
3BA	00111 0111010	7	B00 ↙	10110 0000000	10110 0111010

Esercizio 5

Exercise (011/09/2018):

A) Si consideri la seguente sequenza di riferimenti in memoria nel caso di un programma di 4K parole in cui, per ogni accesso (indirizzi in esadecimale), si indica se si tratta di lettura (R) o scrittura (W): W 3A1, R 3F5, R A64, W BD3, W 57E, R A08, R B85, W 3A0, R A1A, W A36, R B20, R 734, R AB8, R C4E, W B64.

Si determini la stringa dei riferimenti a pagine, supponendo che la loro dimensione sia di 512 parole. Si utilizzi un algoritmo di sostituzione pagine di tipo **Enhanced Second-Chance**, per il quale, al **bit di riferimento** (da inizializzare a 0 in corrispondenza al primo accesso a una nuova pagina dopo il relativo page fault), si unisce il **bit di modifica** (modify bit). Si assuma che una pagina venga sempre modificata in corrispondenza a una scrittura (write), che siano disponibili 3 frame e che l'algoritmo operi con il criterio seguente: dato il puntatore alla pagina corrente (secondo la strategia FIFO) si fa un primo giro, senza modificare il reference bit, sulle pagine per localizzare la vittima (l'ordine di priorità è (reference,modify):

(0,0), (0,1), (1,0), (1,1)); una volta determinata la vittima, si fa un secondo giro per azzerare i reference bit delle pagine "salvate" (comprese tra la posizione di partenza e la vittima).

Determinare quali e quanti page fault (accessi a pagine non presenti nel resident set) si verificheranno. Si richiede la visualizzazione (dopo ogni accesso) del resident set, indicando per ogni frame i bit di riferimento e modifica. Si numerino le pagine a partire da 0.

Ricordiamo come funziona Enhanced Second-Chance:

Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit
- With these two bits, we have the following four possible classes:
 - (0, 0) neither recently used nor modified – best page to replace
 - (0, 1) not recently used but modified – not quite as good, must write out before replacement
 - (1, 0) recently used but clean – probably will be used again soon
 - (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement
- When page replacement is called for, use the clock scheme but use the four classes to replace the page in the lowest non-empty class
 - Might need to search circular queue several times

Exercise (011/09/2018):

4(a). un programma di **4K parole** in cui, per ogni accesso (indirizzi in esadecimale), si indica se si tratta di lettura (R) o scrittura (W): W 3A1, R 3F5, R A64, W BD3, W 57E, R A08, R B85, W 3A0, R A1A, W A36, R B20, R 734, R AB8, R C4E, W B64. Si determini la **stringa dei riferimenti a pagine**, supponendo che la loro dimensione sia di **512 parole (2⁹)**

3A1	001 110100001	1
3F5	001 111110101	1
A64	101 001100100	5
BD3	101 111010011	5
57E	010 101111110	2
A08	101 000001000	5
B85	101 110000101	5
3A0	001 110100000	1
A1A	101 000011010	5
A36	101 000110110	5
B20	101 100100000	5
734	011 100110100	3
AB8	101 010111000	5
C4E	110 001001110	6
B64	101 101100100	5

$$4\text{K parole} / 512 \text{ parole} = (4 * 1024) / 512 = 8 \text{ (} 2^3 \text{ - max page number)}$$

3 bits + 9 bits = 12 bits memory adresses

stringa dei riferimenti a pagine = 1,1,5,5,2,5,5,1,5,5,5,3,5,6,5

Dunque avendo 8 pages in totale, usiamo 3 bit per indirizzare le pages.

Di conseguenza, avendo indirizzi logici a 12 bit, i restanti 9 bits sono di offset.

Exercise (011/09/2018):

4(b). Si utilizzi un algoritmo di sostituzione pagine di tipo **Enhanced Second-Chance**, per il quale, al **bit di riferimento** (da inizializzare a 0 in corrispondenza al primo accesso a una nuova pagina dopo il relativo page fault), si unisce il bit di modifica (modify bit). Si assuma che una **pagina venga sempre modificata** in corrispondenza a una scrittura (**write**), che siano disponibili 3 frame e che l'algoritmo operi con il criterio seguente: dato il puntatore alla pagina corrente (secondo la strategia FIFO) si fa un primo giro, senza modificare il reference bit, sulle pagine per localizzare la vittima (l'ordine di priorità è (reference,modify)). Una volta determinata la vittima, si fa un secondo giro per azzerare i reference bit delle pagine "salvate" (comprese tra la posizione di partenza e la vittima).

Riferimenti <i>Page#</i>	1	1	5	5	2	5	5	1	5	5	5	5	3	5	6	5
Read/Write \rightarrow	W	R	R	W	W	R	R	W	R	W	R	R	R	R	R	W
Resident Set																
Page Fault																

In **Enhanced Second-Chance**, per ogni pagina dobbiamo considerare il **reference bit** e il **modify bit**.

Riferimenti <i>Page#</i>	1 ✓	1	5	5	2	5	5	1	5	5	5	5	3	5	6	5
Read/Write \rightarrow	W	R	R	W	W	R	R	W	R	W	R	R	R	R	R	W
Resident Set	1,0,1	1,1														
Page Fault	X															

Ad esempio — primo accesso a page 1: abbiamo un PF, lo portiamo in memoria per operazione di Write, dunque il modify bit diventa 1, mentre il reference bit rimane 0 dato che abbiamo appena portato in memoria la page.

Al secondo accesso, in lettura, il modify bit rimane 1, e in questo momento diventa 1 anche il reference bit!

E così via..

Riferimenti	1	1	5	5	2	5	5	1	5	5	5	5	3	5	6	5
Read/Write →	W	R	R	W	W	R	R	W	R	W	R	R	R	R	R	W
Resident Set	1 _{0,1}	1 _{1,1}	1 _{0,1}													
			5 _{0,0}	5 _{1,1}	5 _{0,1}											
Page Fault	X		X		X								X		X	

Che succede quando dobbiamo sostituire una page?

Vediamo che arriva la page 3.

Iniziamo dalla prima page in memoria, la page 1, e vediamo che ha reference bit a 1, quindi lo portiamo a 0 e gli diamo una *second-chance*;

Andiamo avanti con la coda FIFO, c'è la pagina 5: anch'essa ha reference bit a 1, quindi portiamo a 0 e andiamo avanti.

Page 2: ha reference bit a 0, quindi è la prima candidata ad essere sostituita dalla page 3.

In conclusione:

Riferimenti	1	1	5	5	2	5	5	1	5	5	5	3	5	6	5	
Read/Write →	W	R	R	W	W	R	R	W	R	W	R	R	R	R	W	
Resident Set	1 _{0,1}	1 _{1,1}	1 _{0,1}	1 _{0,1}	1 _{0,1}											
			5 _{0,0}	5 _{1,1}	5 _{0,1}	5 _{0,1}	5 _{1,1}									
Page Fault	X		X		X							X		X		

Riferimenti	1	1	5	5	2	5	5	1	5	5	5	3	5	6	5	
Read/Write	W	R	R	W	W	R	R	W	R	W	R	R	R	R	W	
Resident Set	1 _{0,1}	1 _{1,1}	1 _{0,1}	1 _{0,1}	1 _{0,1}											
			5 _{0,0}	5 _{1,1}	5 _{0,1}	5 _{1,1}	5 _{1,1}									
Page Fault	X		X		X							X		X		