Exercise 1.7 (page 11)

- For the static list accessing problem with l items,
- Instead of using a bound on the average (over all initial configurations) static optimal to derive $2l/(l+1)$ lower bound, show that we can use the frequency count of items in the request sequence.

Solution

- List size = l
- Size if request sequence $|\sigma| = n$
- Online algorithm
  - Adversary always asks for the last element in the list.
  - So, the cost for online alg = ln
- Offline algorithm looks at the request sequence, and rearranges the list according to the frequency count of items in the request sequence.
- Cost for arranging the list according to FC is less than $\frac{l^2}{2}$

Solution

- Cost for offline (FC based) algorithm
- Let the access probabilities of each of the elements be $p_1 \geq p_2 \geq ..... \geq p_l \geq 0$
- $\sum_{i=1}^{l} p_i = 1$, then $\sum_{i=1}^{l} i.p_i \leq \frac{l+1}{2}$, and if $\forall i, p_i = \frac{1}{l}$,the summation would be 1.
- Let f_i *be the frequency of item i's access,*

$$f_1 \geq f_2 \geq ... \geq f_l \quad , \quad f_i = np_i$$

- Cost of the offline algorithm = $\frac{l^2}{2} + \sum_{i=1}^{l} i * f_i$

$$= \frac{l^2}{2} + n\left(\sum_{i=1}^{l} i.p_i\right) \quad \leq \frac{l^2}{2} + n\frac{l+1}{2}$$

solution

- $C.R = \dfrac{ALG(\sigma)}{OPT(\sigma)}$
- $C.R = \dfrac{nl}{\dfrac{l^2}{2} + n\left(\dfrac{l+1}{2}\right)}$
- $= \dfrac{2nl}{l^2 + nl\left(\dfrac{l+1}{l}\right)} = \dfrac{2nl^2}{l^3 + nl * (l+1)}$
- As n grows beyond *l*, $\boxed{CR = \dfrac{2l}{l+1}}$

Exercise 3.5 (page 38) Prove that FWF is a marking algorithm

Marking algorithm

- Divide request sequence 'σ' into *'K'* phases.
  - *Each phase contains K distinct page requests.*
  - *Result: K-phase partition.*
- *A marking algorithm never evicts a marked page from its fast memory.*
  - *During each phase (with k distinct page requests), there are at most K page faults.*

FWF
(Flush When Full)

- When the fast memory is full, a request for a page not in fast memory causes the cache to be cleared.
- Inherent marking (Marking not explicitly maintained).
  - Can be seen as
  - Pages being marked as soon as they are brought into the cache
  - Once cache is full, all pages are unmarked.
  - All unmarked pages are evicted (flush), when the $k+1^{th}$ distinct page request arrives.

Proof

- If FWF is not a marking ALG
- FWF evicts a *‘marked’* page *‘x’* during some k-phase.
- When *‘x’* was first requested, it will be brought into the cache and ‘marked’.
- Similarly during the same *k-phase*, k-1 other distinct pages are brought into the cache and marked.
- For *‘x’* to be evicted, there needs to be a new ($k+1^{th}$) distinct request.
  - During the same k-phase, there can be a maximum of k distinct requests.
  - Contradiction!
- Therefore, FWF is a marking algorithm

Example

- K = 3
- σ = a,e,a,g,h,f,a,h,b,g,h,a,k,a,b
- Start of each phase, cache is empty
- Phase marking

a,e,a,g	h,f,a,h,b	g,h,a	k,a,b

- So during each k-phase, there are at most k page faults, and none of the *'marked'* pages are flushed.
- Therefore, FWF is a marking algorithm.

Exercise 3.8 (page 39)

- Prove that LRU, FIFO and CLOCK are *conservative* algorithms.
- A conservative algorithm, *"on any consecutive input subsequence containing k or fewer distinct page references, will incur k or fewer page faults"*.

Input sequence

- Consider k=3
- σ = g,h,a,b,b,c,a,c,e,h
- Blue → subsequence with k (=3) or fewer distinct page references.
- Need to show that there are at most 3 page faults for the subsequence.
- Assume initial cache configuration

g	j	m

Observation

- After the subsequence (in blue) starts, before caching all *k* pages, if there are *k or fewer* page faults, then condition for conservative ALG holds good
  - Once cached, k pages are not evicted unless the subsequence ends (i.e. the $k+1^{th}$ distinct request is made).

LRU

σ = g,h,a,b,b,c,a,c,e,h

Data	g	j	m
LRU Time stamp	3	2	1

g

g	j	m
1	3	2

h

g	h	m
2	1	3

a

g	h	a
3	2	1

Page fault = 1

b

b	h	a
1	3	2

Page fault = 2

b

b	**h**	**a**
1	**3**	**2**

c

b	c	a
2	1	3

Page fault = 3

a

b	c	a
3	2	1

c

b	c	a
3	1	2

e

e	c	a
1	2	3

h

e	c	h
2	3	1

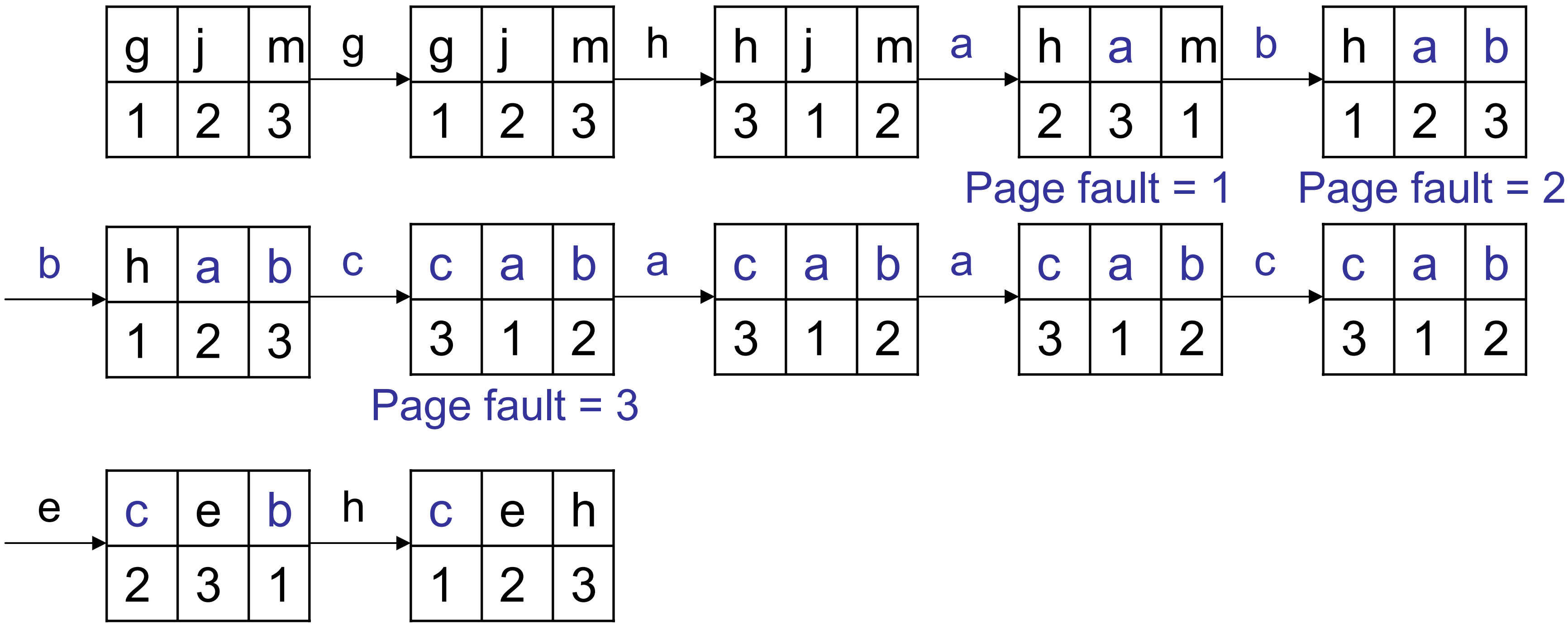
Number of page faults to handle the subsequence = 3 = k.

Therefore, LRU is a conservative algorithm

FIFO

counter = order in which item was brought in

σ = g,h,a,b,b,c,a,c,e,h

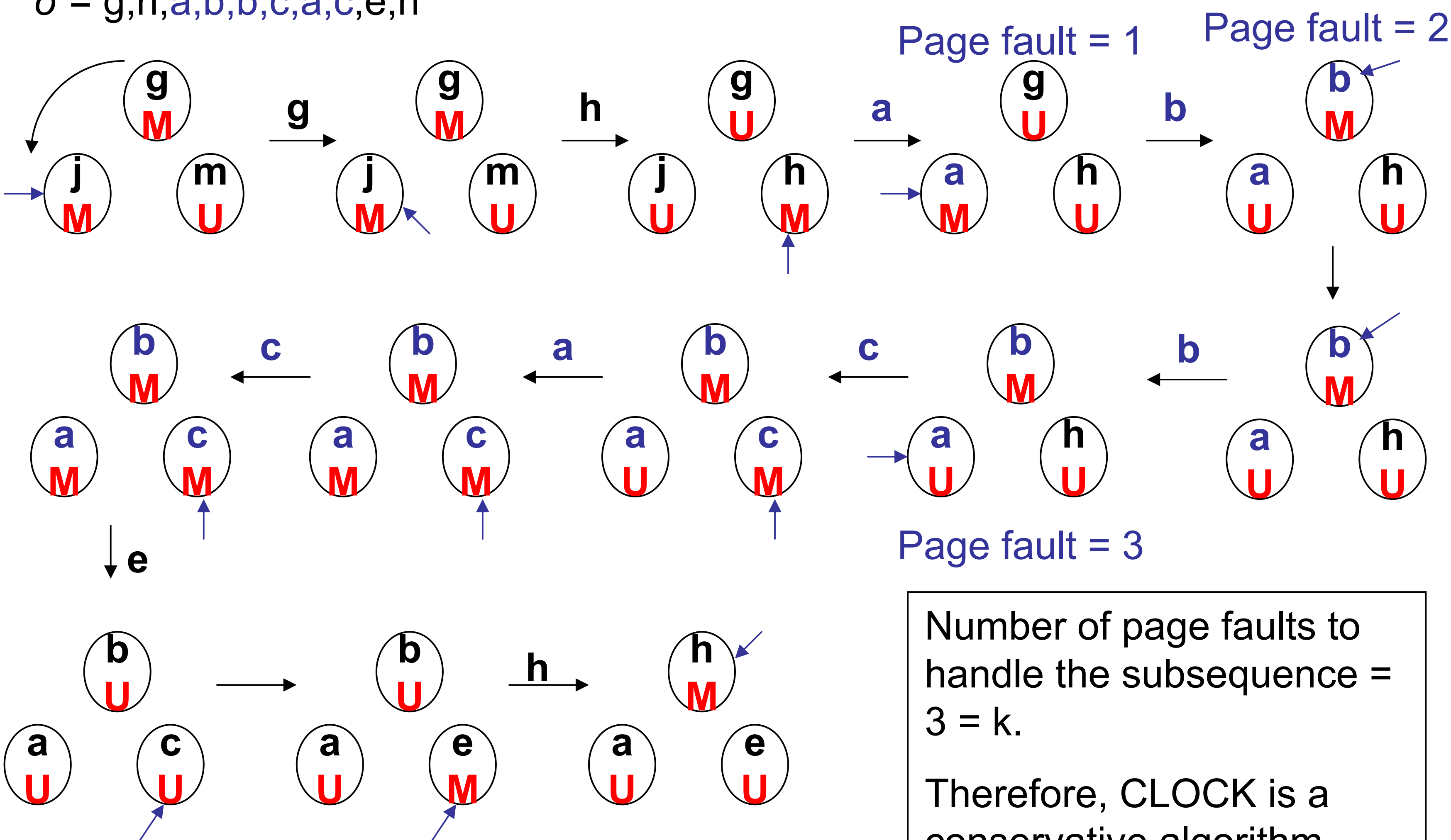


Number of page faults to handle the subsequence = 3 = k.

Therefore, FIFO is a conservative algorithm

CLOCK

σ = g,h,a,b,b,c,a,c,e,h



Number of page faults to handle the subsequence = 3 = k.

Therefore, CLOCK is a conservative algorithm