

ESERCIZIO 11:  
TRONCA (PAG 93)

### CAP 3

DISTANZA TRA MOLECOLE, ROTTINO AL CONTO APPROXIMATO (APPROX (RETURN (UNR)):

ALGORITMO:

$$M' = M$$

$$C = \{\}$$

WHILE  $M' \neq \emptyset$  DO

  seleziona una qualsiasi molecola  $M_i \in M'$

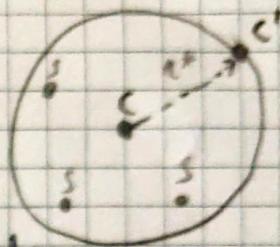
$$C = C \cup \{M_i\}$$

  per tutte le molecole  $M_j$  che risulta  $d(M_i, M_j) \leq \delta$

  RETURN C

## APPROXENTR(CNR CON VARIANTE)

Assumiamo che esista un insieme  $C$  con raggio di copertura  $r(C) \leq r^*$ , ovvero c'è un centro  $c \in C$  e una distanza dalla citta' di  $S$ . Per definizione esiste un centro  $c' \in C'$  a distanza al massimo  $r^*$  da tale qualche altro centro  $c \in C$ , cioè per ogni  $c \in C$  esiste un  $c'$  tale che  $\text{dist}(c, c') \leq r^*$ .



Non tutti vengono cancellati dall'insieme  $S'$  solo quando viene

coperto da un centro che dista al massimo  $r^*$ , ma solo se si ha che  $\text{dist}(r, c) \leq r^*$  e quindi  $r(C) \leq r^*$ .

Pertanto l'algoritmo APPROXENTR(CNR, con questa variante, restituisce un insieme  $C$ , tale insieme ha raggio di copertura  $r(C) \leq r^*$ .

OPPURE

Per il lemma 3.4.2., un insieme con quantità massima di punti da  $k$  centri, secondo l'algoritmo originale ha che  $|C| \leq r^*$  soggetto all'algoritmo con la restrizione che ha  $|C'| \geq |C|$ , se ha che  $|C'| > k$  viene rimosso di più centri. Pertanto, se si aumenta il valore di  $k$  allora l'algoritmo con le restrizioni qui fornite non con una approssimazione superiore soggetto all'originale.

# ESERCIZIO 1 (CAP 4)

TRONI (PAG 106)

CUSTO 1€

NEL GIORNO 3 CANDE, PERCHÉ LE GINNASTI TUNTE E ??:

VINCITA 2€

Per minimizzare la perdita o maximizzare la vincita, infatti:

• ALGORITMO che gira solo 1 carte:

in un partite  $\Rightarrow$  costo:  $m \in$

WIN PROB. MAX

$$\text{rebutti: } 2 \cdot \frac{1}{3} \cdot m = \frac{2}{3}m \in$$

$\Rightarrow$  la rittorba ottima è  $m - \text{costo}$ :  
ottima:  $\frac{2}{3}m - m = -\frac{1}{3}m \in$

• ALGORITMO che gira 2 carte:

in un partite  $\Rightarrow$  costo:  $\left(\frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 2\right) \cdot m = \frac{5}{3}m \in \Rightarrow$  ANTESA:  $-\frac{1}{3}m \in$

$$\text{rittora: } \frac{2}{3} \cdot 2m = \frac{4}{3}m \in$$

• ALGORITMO che gira 3 carte:

in un partite  $\Rightarrow$  costo:  $\left(\frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 2 + \frac{1}{3} \cdot 3\right)m = 2m \in \Rightarrow$  ottima:  $0 \in$

$$\text{rittora: } \left(\frac{1}{3} + \frac{1}{3} + \frac{1}{3}\right)2m = 2m \in$$

Dunque, girare 3 carte è la soluzione migliore siccome andiamo da più

ESERCIZIO 2:  
TRONCO (PERIODO):

CAP 4

|                                     |                                     |                                     |                                     |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|

6000 + CNE, 1€ A CNTA, VNCITA 2€, STRATEGIA MIGLIORE:

• ALGORITMO che gira 1 volta continua f:

$$m \text{ m partite} \Rightarrow \text{costo: } 1 \cdot m = m \text{ €} \Rightarrow \text{utile: } \left(\frac{1}{2} - 1\right)m = -\frac{1}{2}m \text{ €}$$
$$\text{ritorno: } \frac{1}{4} \cdot 2m = \frac{1}{2}m \text{ €}$$

• ALGORITMO che gira 2 volte m f:

$$m \text{ m partite} \Rightarrow \text{costo: } \left(\frac{1}{4} \cdot 1 + \frac{2}{4} \cdot 2\right)m = \frac{5}{4}m \text{ €} \Rightarrow \text{utile: } \left(1 - \frac{5}{4}\right)m = -\frac{1}{4}m \text{ €}$$
$$\text{ritorno: } \left(\frac{1}{4} + \frac{1}{2}\right) \cdot 2 \cdot m = m \text{ €}$$

• ALGORITMO che gira 3 volte m f:

$$m \text{ m partite} \Rightarrow \text{costo: } \left(\frac{1}{4} \cdot 1 + \frac{2}{4} \cdot 2 + \frac{3}{4} \cdot 3\right)m = \frac{14}{4}m \text{ €} \Rightarrow \text{utile: } \left(\frac{6}{4} - \frac{14}{4}\right)m = -\frac{8}{4}m = -2m \text{ €}$$
$$\text{ritorno: } \left(\frac{1}{4} + \frac{1}{2} + \frac{1}{4}\right) \cdot 2 \cdot m = \frac{6}{4}m \text{ €}$$

• ALGORITMO che gira tutte e 4:

$$m \text{ m partite} \Rightarrow \text{costo: } \left(\frac{1}{4} + \frac{2}{4} + \frac{3}{4} + \frac{1}{4}\right)m = \frac{10}{4}m \text{ €} \Rightarrow \text{utile: } \left(2 - \frac{10}{4}\right)m = -\frac{2}{4}m = -\frac{1}{2}m \text{ €}$$
$$\text{ritorno: } \left(\frac{1}{4} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4}\right) \cdot 2 \cdot m = 2m \text{ €}$$

La strategia migliore è non girare niente ma al massimo.

EJOR(210) 3:  
TOM (M106)

CAP +

DIS 1€  
VINCITA 3€

(100 + 100) CON 3 E VINCITA;

ALGORTMO che gira 1 molo con 3 m:

$$\begin{aligned} \text{m m partiti} \Rightarrow \text{costi: } 1 \cdot m = m \text{ €} & \Rightarrow \text{altri: } \left(\frac{3}{4} - 1\right)m = -\frac{1}{4}m \text{ €} \\ \text{molti: } \frac{1}{4} \cdot 3m = \frac{3}{4}m \text{ €} \end{aligned}$$

ALGORTMO che gira 2 m + costi:

$$\begin{aligned} \text{m m partiti} \Rightarrow \text{costi: } \left(\frac{1}{4} + \frac{2}{4}\right)m = \frac{3}{4}m \text{ €} & \Rightarrow \text{altri: } \left(\frac{6}{4} - \frac{5}{4}\right)m = \frac{1}{4}m \text{ €} \\ \text{molti: } \left(\frac{1}{4} + \frac{1}{4}\right) \cdot 3m = \frac{6}{4}m \text{ €} \end{aligned}$$

ALGORTMO che gira 3 m + costi:

$$\begin{aligned} \text{m m partiti} \Rightarrow \text{costi: } \left(\frac{1}{4} + \frac{2}{4} \cdot 2 + \frac{3}{4} \cdot 3\right)m = \frac{14}{4}m \text{ €} & \Rightarrow \text{altri: } \left(\frac{9}{4} - \frac{14}{4}\right)m = -\frac{5}{4}m \text{ €} \\ \text{molti: } \left(\frac{1}{4} + \frac{1}{4} + \frac{1}{4}\right) \cdot 3m = \frac{9}{4}m \text{ €} \end{aligned}$$

ALGORTMO che gira tutto x 4:

$$\begin{aligned} \text{m m partiti} \Rightarrow \text{costi: } \left(\frac{1}{4} + \frac{2}{4} + \frac{3}{4} + \frac{4}{4}\right)m = \frac{10}{4}m \text{ €} & \Rightarrow \text{altri: } \left(3 - \frac{10}{4}\right)m = \frac{2}{4}m = \frac{1}{2}m \text{ €} \\ \text{molti: } \left(\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}\right) \cdot 3m = 3m \text{ €} \end{aligned}$$

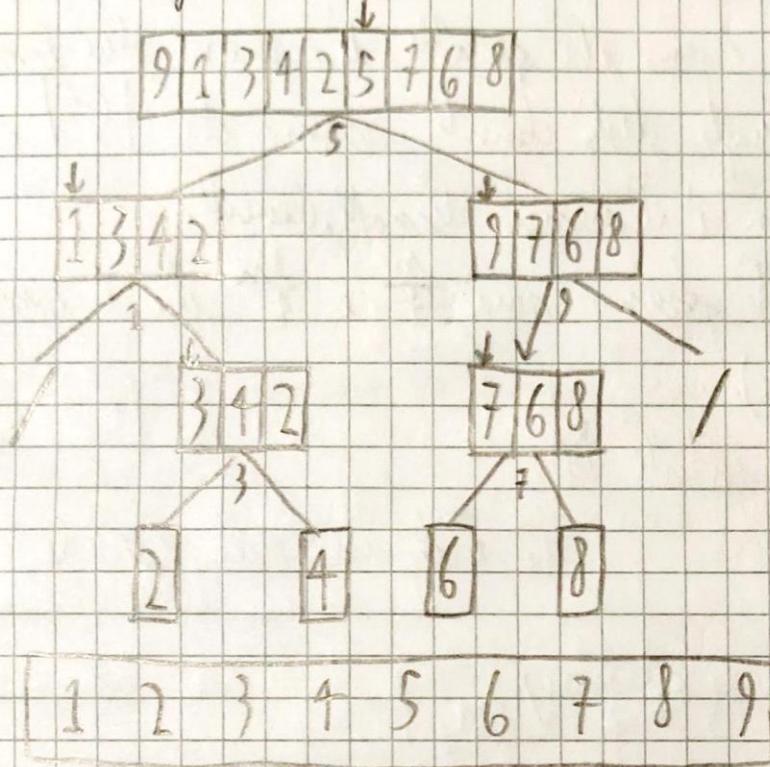
La strategia migliore è' gire tutto x 4 le molte, siccome m'incassa  $\frac{1}{2}m$  €.

ESERCIZIO  
TRAS (PAG: 110/2)

CAP 4

ESEMPIO QUICKSORT FORTUNATO/SPIRTOUMATO

Rappresentiamo di avere il seguente vettore da ordinare:



L'ordinamento ha alternato sortunato e sfortunato, siccome, nel 1° livello si è scelto il caso sfortunato siccome il geno nello è stato 5 che era  $\frac{1}{2}$  in quel momento. Nel 2° livello sono sortunati siccome nella parte sinistra c'è stato nello 1 che era il minimo della sequenza, quindi la partizione è stata 0 e  $n-1$ , siccome, la partizione non è stata una frattina di elementi.

## RANDOMSELECT BILL:

L'idea di RANDOMQUICKSORT BILL è quella di farcire la metà del perno negliendo un elemento "centrale", cioè elemento o perno che permette di generare due insiemini (destro sinistro) contenenti almeno tre quarti degli elementi, avremo che  $|S'| \geq |S''| \geq \frac{3}{4}n$

uguale ad almeno  $\frac{n}{4}$  sono in tutto il numero di elementi.

Con questo idea lo sforbiciamento generano sarà  $\frac{4}{3}$  e  $\frac{3}{4}$  che è comunque O(n log n).

ALGORITMO RANDOMSELECT BILL(S, k)

WHILE non troviamo un perno centrale DO:

$i \leftarrow \text{random}(n)$  //  $a_i$  è il perno nella casuamente

FOR ogni  $a_j \in S$  DO

IF  $a_j < a_i$  THEN  $S' = S' \cup \{a_j\}$

IF  $a_j > a_i$  THEN  $S'' = S'' \cup \{a_j\}$

IF  $|S'| \geq \frac{|S|}{4}$ ,  $|S''| \geq \frac{|S|}{4}$  THEN

-  $a_i$  è un perno centrale

IF  $|S'| = k-1$  THEN Il perno  $a_i$  è il valore cercato

IF  $|S'| \geq k-1$  THEN Ricordare RANDOMSELECT BILL( $S'$ , k)

IF  $|S'| < k-1$  THEN Ricordare RANDOMSELECT BILL( $S''$ ,  $k-1-|S'|$ )

## ANALISI TEMPO:

Il WHILE idea mette difficoltà che non si trova un perno centrale, quindi rimane il numero di tentativi per trovarlo. Il perno è nella uniformemente a caso e la sua probabilità è di  $\frac{1}{2}$  in quanto ci sono esattamente  $\frac{n}{2}$  elementi centrali.

Dal lemma 7.3.1 (pag. 109) si ha che il numero atteso delle iterazioni prima di trovare un elemento centrale è 2, significa che il tempo necessario al pernissimo raddoppia.

Tuttavia, anche se viene scelto un perno centrale (o quasi più vicino al centro) comporta comunque una relazione di ricorsività la cui soluzione è  $O(n)$ .

ESERCIZIO 7:

Tecnica (MAG: GRAPH COLORING)

(AP +

AVVISO CON DIPENDENTI, TRAVERE ALGORITMO RANDOM,  $\frac{2}{3}$  APPROSSIMATO.

$n$  dipendenti,  $m$  lavori:  $\{w_1, \dots, w_m\} \in \mathbb{N}^m$  |  $d_1(w_i) \leq d_2(w_i)$ .

Competenze  $a, b$  e  $c$ , si assegna una competenza a ogni dipendente in modo casuale.

Definiamo una variabile  $X_i = \begin{cases} 0 & \text{se competenze } d_1(w_i) = d_2(w_i) \\ 1 & \text{altrimenti} \end{cases}$

$\begin{array}{c} \rightarrow \text{aa} \\ \text{ab} \\ \text{ba} \\ \rightarrow \text{bb} \\ - \\ \rightarrow \text{ca} \end{array} \left. \begin{array}{c} 0 \\ 1 \\ 1 \\ 2 \\ 1 \\ 0 \end{array} \right\} m \right)$

Pertanto,  $P[X_i=0] = \frac{1}{3}$ , ricevo mi hanno 3 competenze e abbiano  $\frac{1}{3}$  possibilità di prendere la stessa competenza.

Mentre  $P[X_i=1] = \frac{2}{3}$ , ricevo mi hanno 3 competenze e abbiano  $\frac{2}{3}$  possibilità di avere competenze diverse.

Il valore atteso sarà  $E[X_i]$ , perché vogliamo minimizzare il numero di lavori per i quali le competenze dei 2 dipendenti sono diverse, quindi sare la media di  $X_i$  delle somma:

$$E[X] = E[X_1] + \dots + E[X_m] = \left(0 \cdot \frac{1}{3} + 1 \cdot \frac{2}{3}\right) + \dots + \left(0 \cdot \frac{1}{3} + 1 \cdot \frac{2}{3}\right) = \frac{2}{3}m$$

ESERCIZI 8  
esercizi (pag. 125)

CAP +

MAX SAT!

L'ipotesi è di avere una clausola MAXSAT:

$$\phi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_1)$$

Se  $Z$  una variabile casuale che denota il numero di clausole soddisfatte, decomposta nella somma di variabili  $Z_i = \begin{cases} Z_i = 1 & se C_i = \text{true} \\ 0 & \text{altrimenti} \end{cases}$

dove  $C_i$  è la disgiunzione di  $|C_i|$  letterali (supponendo che  $|C_i|$  sia il numero di letterali  $\in C_i$ ) ed è vera se almeno uno dei letterali ha valore vero.

L'auto con la cui la clausola è falsa è quella in cui tutti i letterali di  $C_i$  hanno valore falso che avviene con probabilità  $\frac{1}{|C_i|}$  mentre è vera con probabilità  $1 - \frac{1}{|C_i|}$ , quindi la media del valore atteso è  $E[Z_i] = \frac{|C_i|-1}{|C_i|}$ .

Mentre la linearità del valore atteso ci dice che:  $E[Z] = E[Z_1] + \dots + E[Z_k] = \frac{|C_1|-1}{|C_1|} \cdot k$

L'assegnamento ottimale per soddisfare al massimo  $k$  clausole:

• Se l'istruzione ha  $k \leq |C_i|-1$  clausole allora tutte le clausole sono vere.

• Se l'istruzione ha  $k > |C_i|-1$  possiamo ripetere l'assegnamento  $|C_i|/k$  volte per soddisfare tutte, per il lemma 3.4.1.

ESERCIZIO:  
TRAMO (CAP. 120)

CAP 4

EDGE CONTRACTION RIMUOVENDO ARCHI  $(s, t)$  IN OGNI CONTRAzione:

L'algoritmo di Contrazione procede collassando nodi su cui 'resta' caso fino a che il multigrato contiene solo due nodi. Con questa variazione, se si negli archi  $(s, t)$  punti non viene collassato e vengono rimossi tutti gli archi che connettono  $s$  a  $t$ , in questo modo i nodi  $s$  e  $t$  si trovano in seguenti diversi.

L'algoritmo Edge CONTRACTION puo' essere comunque usato, ma farra solo nei casi in cui  $s$  e  $t$  non vengono messi nello stesso multigrato.

Son le variazioni, l'escursione dovrebbe essere un po' più veloce perché salta alcuni dei nodi quelli da cui i nodi  $s$  e  $t$  vengono contattati.

Pertanto, l'algoritmo puo' essere molti riguardante, la differenza è che la variazione considera un multigrato delle possibili soluzioni dell'algoritmo originale, ovvero tutti le soluzioni da cui  $s$  e  $t$  si trovano in seguenti differenti.

# BSP (2/2) 1: (CAP 5)

TRONCA (PAU 133)

AFFRITTO SCI CON  $K < n \times K > n$ :  $\text{PITTARE} = 1€$ ,  $\text{COMPRA} = n€$ ,  $m = \text{VALORE CAR SI VA A SCIARE}$   
 $K = \text{PREZZO VOLTE PRIMA DI COMPRARE}$

- Ricaggettando con  $K = n$ :

- Paghiamo 1 fino alla  $K$ -esima volta, al

- costo riserva' fino a  $K = n$ .

- Una volta raggiunto  $K = n$ , n'acquistano gli "nuovi" pagamenti  $n$ .

- Se  $n < K$  (per l'attimo) n'affitta, mentre se  $n \geq K$  n'acquista subito.

Peraltro, il fattore di approssimazione è  $\frac{\text{COSTO ONLINE}}{\text{COSTO OTTIMO}} = \frac{2n-1}{n} = \underline{2 - \frac{1}{n}}$ .

- Con  $K > n$ :

- Paghiamo sempre 1 fino a  $K$  e a  $K$  n'acquista.

- La riserva ottima, invece, confronta  $n$  con  $n$ , acquista fino ad  $n-1$  affitti, mentre se  $n \geq n$  compra.

Fatto ad  $n$  n'ha l'ordine che l'ottimo non ottimale, da

$n$  da per l'online peggiore ed il suo peggiore è quando  $n \geq K$ , con  $K \geq n+1$ .

Il fattore di approssimazione è  $\frac{K-1+n}{n} \geq \frac{n+1-n+1}{n} = \frac{2n}{n} = \underline{2}$ .

- Con  $K < n$ :

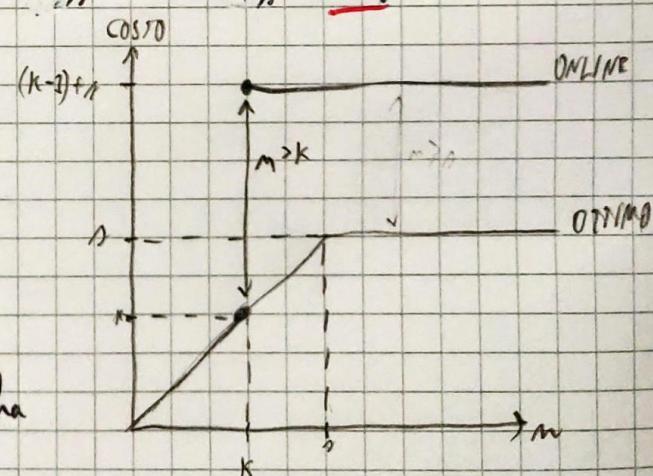
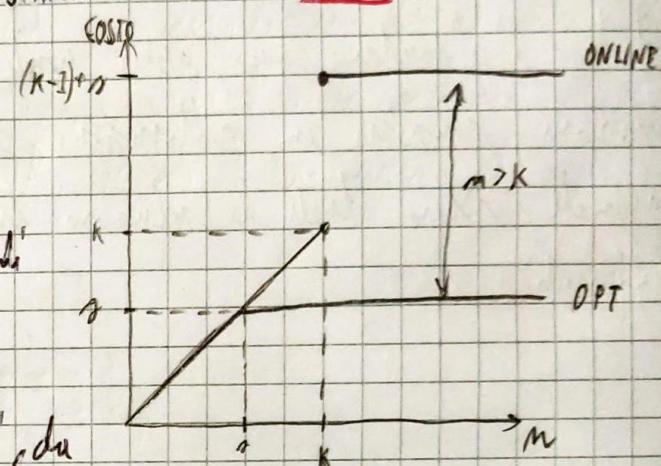
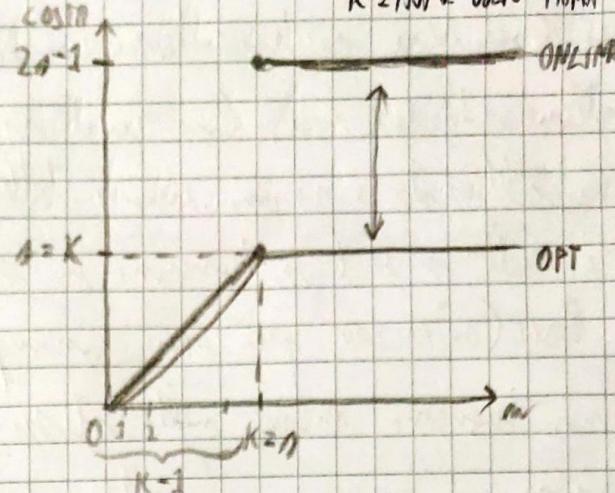
- Fatto a  $K$  n'affitta e poi n'acquista.

- Per l'ottimo n'acquista da  $n$ , comprando subito.

- In questo caso, il suo peggiore è all'infinito.

Fatto a  $K$  è ottimale da entrambi finché n'ha un pagamento, adesso è  $n \geq K+1$ .

Il fattore di approssimazione è  $\frac{n+(K-1)}{K} \geq \frac{K+1+K-1}{K} = \frac{2K}{K} = \underline{2}$ ,



### LIFO NON È C-COMPETITIVA

E' sempre possibile trovare una sequenza di input per le quali l'algoritmo LIFO genera un numero di PF più grande di  $\lambda$ .

Sia  $K$  la grandezza della memoria, fissiamo una costante  $\lambda$ , determiniamo una sequenza di input:  $1, 2, 3, \dots, K, \frac{K+1}{PF}, \frac{K}{PF}, \frac{K+1}{PF}, \dots$

Ritengono la memoria piena a  $K$ , successivamente richiede  $K+1$  non presente in memoria e si cerca un PF che espelle  $K$ , lo trodiché richiede di nuovo  $K$  dove si cerca un altro PF, poi richiede di nuovo  $K+1$  e così via ... Ma sequenza non "fatta", ogni nuova richiesta cerca un PF e qualunque sia il valore dell'attivo al termine della sua sequenza è definita.

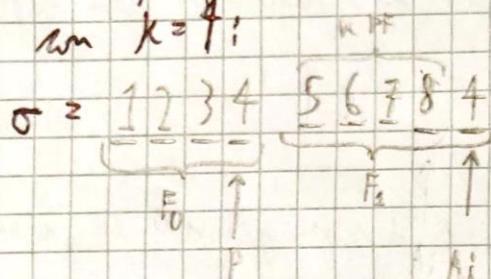
ESERCIZIO 3  
TEORIA (ARG 136)

CAP 5

ESEMPI DELLA PROVA K-COMP CON  $K=4$ :

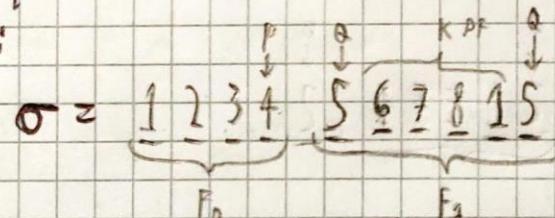
(a) Duchi  $P$  è la pagina più recente dell'histo della fase, per avere un PF con  $A_0 = P$  è quella di estrarre  $P$  dalla cache prima della richiesta  $A_i$ .  $P$  deve dunque la pagina più vecchia e per questo sono  $K-1$  richiesti diversi da  $P$  e poi un PF che togli  $P$ .

Esempio con  $K=4$ :



(b) Per avere una  $Q = A_i = t_j$ ,  $i < j$ , pagina duplicata è necessario che fra il primo PF  $A_i$  e il secondo PF  $A_j$  ci siano altri  $K$  PF e poiché  $P$  è un memoria, questi PF devono essere tutti per pagine diverse da  $P$ , mentre  $P$  deve dunque la pagina più vecchia per essere estraite dalla cache.

Esempio con  $K=4$ :



FIFO E K-COMPATITIVO:

Dobbiamo provare che per qualunque sequenza  $\sigma = \sigma_1 \dots \sigma_m$  di richieste si ha:

$$\text{cost}_{\text{FIFO}}(\sigma) \leq k \cdot \text{cost}_{\text{OPT}}(\sigma)$$

Partizioniamo la sequenza in pacchetti:  $\sigma = P_0 P_1 \dots P_n$

dove  $P_i$  è una sottosequenza di richieste su cui FIFO ha al più  $k$  PF e le  $P_i$  sono composte da sottosequenze esattamente  $k$  PF.

L'algoritmo ottimo ha almeno un PF:  $\text{cost}_{\text{OPT}}(\sigma) \geq n+1$ .

Dobbiamo dimostrare che in una germea  $F_i$ ,  $i > 0$ , c'è necessariamente almeno  $k$  richieste a pagine non presenti in precedenza. In più, dobbiamo dimostrare che ci sono sempre  $k$  richieste a pagine diverse da  $P$ , dove  $P$  è l'ultima pagina richiesta nella germea  $F_{i-1}$ :

- 1° caso, se le pagine richieste da  $F_i$  sono tutte diverse da  $P$ : <sup>caso</sup>  $P$  non borsale riceve ogni richiesta almeno un PF (almeno 1 PF).

- 2° caso, ci sono duplicati nelle richieste nella  $F_i$ , abbiamo 2 sotto casi:

- $P$  è duplicato, cioè  $P = A_i$  per un qualche  $i$ : (a) ESEMPIO 3

- $P$  non è duplicato, cioè  $P \neq Q = A_j = A_i$  con  $i \neq j$ : (b) ESEMPIO 3

In ogni caso, si ha che  $F_i$  contiene almeno  $k$  richieste a pagine distinte e diverse da  $P$ , pertanto le  $k$  richieste generano almeno un PF.

Il costo dell'algoritmo ottimo deve essere più ad almeno il minore di pacchetti  $n+1$ :

$$\text{cost}_{\text{FIFO}}(\sigma) \leq k(n+1) = k \cdot \text{cost}_{\text{OPT}}(\sigma)$$

BSP R(12/10 6  
TRONCA (PAG. 135)

CAP 5

ANOMALIA FIFO: PF CON CACHE PIU' GRANDE > PF CON CACHE PIU' PICCOLA: (BELARDI ANOMALY)

1 2 3, 4 1 2 5 1 2 3 4 5

k=3

1 2 3

4 2 3

4 1 3

4 1 2

5 1 2

5 3 2

5 3 4

k=4

1 2 3 4

5 2 3 4

5 1 3 4

5 1 2 4

5 1 2 3

4 1 2 3

4 5 2 3

PF = 9

PF = 10

ESERCIZIO 4

(FP S)

ANALIZZARE FWF, SE ('R) UN PF SI SVUOTA LA CACHE;

Considerando il fatto che FWF, ad unico PF della sequenza, non genererà k PF della cache, ci saranno altri  $k-1$  PF siccome la cache viene svuotata.

Pertanto, se consideriamo la sequenza di richieste  $\sigma$ , divisa in fasi  $F_0, \dots, F_n$ , abbiamo che VFP, ad unico PF siccome al primo PF della sequenza ne conseguono altri  $k-1$  PR, ad eccezione dell'ultima fase  $F_n$  dove ci sarà almeno un PF, pertanto abbiamo che  $\text{lost}_{FWF}(\sigma) \leq k \cdot (n+1) = k \cdot \text{lost}_{OPT}(\sigma)$ . In alcuni casi,

FWF crea un flush e  $k$  PF, LRU, ad esempio:

$k=3$  1 2 3 4 5 6

| LRU   | FWF   |
|-------|-------|
| 1 2 3 | 1 2 3 |
| 4 2 3 | 4 1 1 |
| 4 5 3 | 4 5 1 |
| 4 5 6 | 4 5 6 |

altro esempio:

$k=3$  1 2 3 4 2 3

| LRU   | FWF   |
|-------|-------|
| 1 2 3 | 1 2 3 |
| 4 2 3 | 4 2 3 |

FWF crea più PF del LRU, quindi dove LRU genera un PF ricarica la cache anche FWF ma in più ne genererà altri  $k-1$  PF siccome la cache si svuota.

All'inizio della fase 1 la cache è piena e siccome si sono  $k$  PF il primo svuota la cache. Per avere i restanti  $k-1$  flush dobbiamo avere per fases  $k-1$  richieste diverse dall'elemento che ha svuotato la cache. Quindi in ogni fase avremo che ha  $k$  richieste differenti, quindi è  $k$ -competitivo.

## ESERCIZIO 2 (CAP 5) PAG 198

TRAMA (PAGI 146)

PAGI

VERGILIA AZIONI

$N_{\text{azioni}}$

Supponiamo di avere  $N_{\text{azioni}}$  e che il loro valore cambi ogni giorno  $N_1, \dots, N_m$  con  $m$  che sono i giorni.

$N_{\text{azioni}}$   
 $i = 1 \dots m$  giorni

ALGORITMO:

Consideriamo un algoritmo parametrizzato dall'indice  $n$ , con  $1 \leq n \leq m$ , e vogliamo che renda per i primi  $n-1$  giorni, calcolando il massimo  $M_{n-1}$ , accettiamo il quinto valore,  $\geq M_{n-1}$  e se tutti i successivi valori sono minori rivediamo l'ultimo giorno.

ANALISI:

Poiché ci sono  $m$  valori, i possibili  $n$  sono  $m$ , siamo in possibili istanze dell'algoritmo online, ognuna del quali seleziona un elemento dall' $n$ -esimo all'ultimo da dare al valore di  $n$ .

Considerando l'esempio dati:  $2 \ 1 \ 5 \ \overbrace{7 \ 8}^{M_6=8} \ 3 \ 4 \ 1 \ 7 \ 9 \ 2$

Ripetiamo le prime 6 e calcoliamo il massimo che è  $M_6=8$ , scartiamo la successiva offerta giorno di 9 che è il nuovo massimo, pertanto rendiamo le Nazioni il giorno 10 che hanno  $N_{10}=9$  valore.

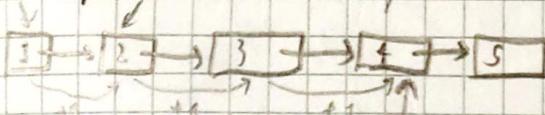
Indice considerando l'esempio:  $9 \ 7 \ 8 \ 2 \ 4 \ 7 \ 1 \ 2 \ 3 \ 2 \ 4$

Questo è un caso pessimo siccome rende l'ultimo giorno, ma il valore è minore del massimo.

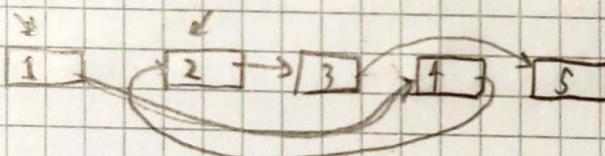
LISTA ON LINE, PERCHÉ NON SPOSTARE UN ELEMENTO VERSO LA CODA DELLA LISTA?

Per spostare un elemento verso la testa della lista, non devo spostare più di quello che già ho spostato, perché posso ricordarmi qual è il precedente, siccome la lista l'ho attraversata partendo dalla testa ed ogni volta fago grande passo all'elemento successivo.

Basta memorizzare il precedente e poi inserire un nuovo elemento per poter poi spostare l'elemento a cui avrò fatto quella sostituzione. Mentre se voglio spostare l'elemento più da avanti rispetto dove c'era:



Li punto dalla Testa della lista e proponiamo di scorrere pagando ad ogni passo, all'elemento successivo, al termine di un elemento (+). Se voglio spostare un elemento ad esempio tra 1 e 2, basta mantenere un puntatore nell'1 e 2, gli poter poi cambiare in tempi costanti:



«Soltanto il puntatore 1-2, mettere 1-3 e così via.

Se invece li spostiamo verso la coda, bisogna pagare tutti i nodi della lista fino all'ultimo elemento, perché il costo non è costante.

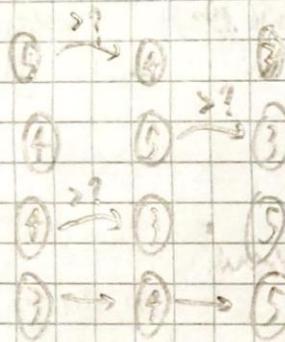
Mentre se trovi all'inizio in quale fare buon modo costante mantenimento dei puntatori.

B) D (RIO 7 - CAP 5 - PAG 14)

TEMA (PAG) 133

ORDINAMENTO DI LISTE CON SCambi SCAMBII A DIACRITICI.

Dati una lista di elementi, è possibile ordinare effettuando scambi adiacenti secondo la strategia bubblesort, dove si prende il primo elemento che lo si confronta col successivo e se l'elemento è maggiore del successivo elemento allora lo si scambia di posizione finché non si trova un elemento maggiore o si arriva allo fine.



ESERCIZIO 11  
TEORIA (PAG: 199)

CAP 5

BIN PACKING MA SENZA CHIUDERE I CONTENITORI!

Il bin packing non chiudere i contenitori è lo stesso problema del load balancing, pertanto puoi usare solo l'algoritmo APPROX LOAD IN ORDER quale:

MONTAG

$j \leftarrow 1$   
FOR  $i \leftarrow 1$  TO m DO

Numero contenitore  
// Per ogni oggetto

IF  $a_i$  non può essere inserito in nessun  $B_j$  THEN

$L_j \leftarrow j + 1$

Se  $B_j$  è il contenitore meno pieno fra quelli disponibili

$L_{max}(B_j) \leftarrow a_i$  in  $B_j$

MONTAG

Sia  $r = \frac{\sum_{i=1}^m a_i}{c}$  dove  $c$  è la capacità di ogni contenitore  $B_j$ , allora che  $\text{cost-}(NRXTFIT) \leq 2r$  (RATIO).

Con l'algoritmo descritto sicuramente  $\text{cost-}(A) \geq r$  e  $\text{cost-}(A) \leq 2r$ .

Nel caso peggiore fare gli oggetti avremo per  $\frac{r}{2} + 1$ , il numero di contenitori usati sarà uguale al numero di oggetti, ovvero  $2r$ , mentre negli altri casi sarà maggiore siccome potremo continuare a mettere oggetti nei contenitori.

# ESERCIZIO 1 CAP 6 PAG. 204

TRONCA (PAG. 258)

ANALISI COORDINATO CON  $n \geq 3$ . SI DIA UN ALGORITMO CHE LO RISOLVE O PROVA L'IMPOSSIBILITÀ?

Consideriamo un sistema distribuito con  $n^{\geq 3}$  processi, comuni da un canale che puo' perdere messaggi. PROPOSTA:

• Accordo: Tutti decidono 0 o tutti 1;

• Validità: Se c'è qualche processo che decide 1 allora ci devono essere almeno due processi che decidono 1;

• Garanzia: Tutti i processi decidono.

Non esiste un algoritmo che possa risolvere tale problema.

DIM:

Per contraddizione, assumiamo che esista un algoritmo A che risolve il problema, esso permette di spedire messaggi a tutti i processi in ogni round.

Se  $\alpha$  è l'esecuzione in cui tutti i processi hanno input 1 e in cui tutti i messaggi vengono consegnati. Dalle 3 proprietà, n'ha che la decisione finale per i processi è 1.

Se  $\alpha_0$  è l'esecuzione in cui tutti decidono 0 ma  $\alpha_0$  l'esecuzione identica ad  $\alpha$  fino al round  $r$  e diverse da  $\alpha$  in poi in quanto i messaggi spediti si perdono.

Poiché le esecuzioni  $\alpha$  e  $\alpha_0$  sono indistinguibili fino ad  $r$ , tutti i processi decidono 1 anche in  $\alpha_0$ .

Consideriamo  $\alpha_1$  identica ad  $\alpha_0$  ma gli ultimi messaggi da  $p_1$  a  $p_2, \dots, p_n$  vengono persi. Per  $p_1$   $\alpha_1$  è indistinguibile da  $\alpha_0$ , pertanto  $p_1$  che decide 1 in  $\alpha_0$  lo farà anche in  $\alpha_1$ .

Consideriamo  $\alpha_2$  identica ad  $\alpha_1$  ma gli ultimi messaggi da  $p_2$  a  $p_3, \dots, p_n$  vengono persi.

Per  $p_2$ ,  $\alpha_2$  è identica ad  $\alpha_1$ , pertanto  $p_2$  che decide 1 in  $\alpha_1$  lo farà anche in  $\alpha_2$ .

E così via per il successivo  $\alpha_3, \dots, \alpha_i$  definito in modo analogo.

L'arriverà ad  $\alpha_{i+1}$  in cui nessun messaggio viene consegnato tra i processi comunque decidono 1.

Consideriamo  $\alpha_{i+2}$ , identico a  $\alpha_{i+1}$  (senza messaggi consegnati) ma in cui  $p_m$  ha input 0.

Per  $p_2$  non c'è differenza con  $\alpha_{i+1}$ , quindi  $p_2$  decide 1. In modo analogo anche  $\alpha_{i+3}, \dots$

Sia  $\alpha_{i+3}$  identica da  $\alpha_{i+(j-1)}$  con tutti i processi con input 0, per tutti gli altri processi non c'è differenza col precedente, quindi  $p_2$  decide 1 in  $\alpha_{i+j}$ .

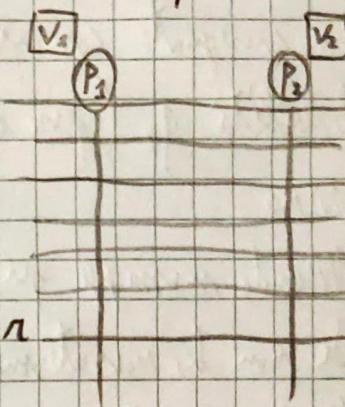
Questo è una violazione delle proprietà di accordo, siccome ha tutti input 0 ma decidono 1.

EERC(210 2  
TEORIA (PAU 160)

CAP 6

ANALISI DI UN ROLLO CON 2 PROCESSORI RANDOMIZZATO CON PROBABILITÀ P MESSAGGI CONSEGNAZIONI

Consideriamo un sistema con 2 processori  $P_1$  e  $P_2$ , ognuno invia un valore binario  $\{0, 1\}$ , l'algoritmo deve terminare l'esecuzione in un numero fisso  $n \geq 1$  di round. Ogni round verrà spedito un messaggio da ogni processo e i messaggi hanno probabilità  $\gamma$  di arrivare e  $1 - \gamma$  di non arrivare.



ALGORITMO:

Manda messaggi per  $n$  round ( $n$  fisso)

Se non ho ricevuto messaggi allora deido  $N_1$  (valore binario)

Se ho ricevuto, se  $N_1 = N_2$  allora deido  $N_1$  o  $N_2$

altrimenti deido 0 (valore di default)

VARIAZIONE 3 PROPRIETÀ:

1. Accordo: con probabilità  $(1 - \gamma)^2$  non mi ricevono messaggi, pertanto, abbiano con probabilità  $\frac{1}{2}$  i processori inviano valori diversi, quindi con probabilità  $(1 - \gamma)/2$  ci è disaccordo.

2. Validità: Sempre rispettato, anche se non mi ricevono messaggi ma hanno entrambi lo stesso input, deidano quell'input. Se mi ricevono messaggi mi verifica la condizione  $N_1 = N_2$  e mi deido lo stesso input.

3. Minimizzazione: Sempre, dopo  $n$  round.

ESERCIZIO 3

TRONI (pag. 160)

CAP 6

ATTACCO COORDINATO 2 PROCESSORI RANDOM, CON PROBABILITÀ  $q_1, q_2$  PER  $T_1$  E  $T_2$ :

Consideriamo un sistema con processori  $T_1$  e  $T_2$ , ognuno habile con un valore binario  $\{0,1\}$ . L'algoritmo termina in un numero finito  $n \geq 1$  di round.

In ogni round vengono spediti messaggi a tutti i processori, ma il messaggio da  $T_2$  a  $T_1$  viene consegnato con probabilità  $q_1$  (pero con  $1-q_1$ ) e da  $T_1$  a  $T_2$  viene consegnato con probabilità  $q_2$  (pero con  $1-q_2$ ).

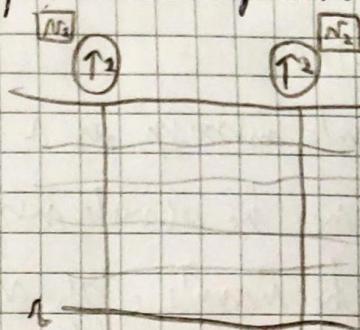
ALGORITMO

Mando messaggi per  $n$  round (n fissato)

Se non ho ricevuto messaggi allora decido  $N_1$  (valore binario)

Se ho ricevuto, se  $N_1 = N_2$  allora decido  $N_1$  o  $N_2$

altimenti decido  $Q$  (valore default)



VALUTAZIONE 3 PROPRIETÀ:

1. ACCORDO: Con probabilità  $[(1-q_1)+(1-q_2)]^n$  non si ricevono messaggi e abbiamo che i processori hanno probabilità  $\frac{1}{2}$  di avere valori diversi; quindi con probabilità  $\frac{[(1-q_1)+(1-q_2)]^n}{2}$  si ha un accordo;

2. VANDITA: Sempre rispettata, siccome se hanno tutti lo stesso input e non si ricevono messaggi si decide comunque lo stesso valore, mentre se non si ricevono solo soddisfatta la condizione  $N_1 = N_2$  e si sceglie lo stesso input.

3. TERMINAZIONE: Sempre, dopo  $n$  round.

ESERCIZIO 4 - CAP 6 - PAG: 205

TRONCA (PAG: 165)

FLOODSET PRESENZA ESECUZIONE  $\alpha$ :

Abbiamo 4 processi che iniziano con valori 1,0,0,0.

Supponiamo che  $f=2$ , così seguenti round:

• 1° round:  $p_1$  invia con valore 1 e spedisce info

•  $p_2$  fa poi quarantina\*. La conoscenza dei processi sarà:

$$\text{quar} = \{1, 0, 0, 0\} - \{?, 0, 0, 0\} = \{?, 0, 0, 0\}$$

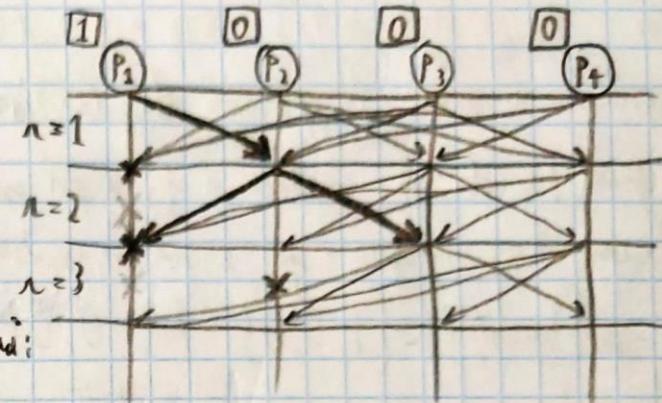
• 2° round:  $p_2$  spedisce a  $p_1$  e  $p_3$  e poi in quarantina\*. La conoscenza sarà:

$$\text{quar} = \text{quar} - \{1, 0, 0, 0\} = \{?, 0, 0, 0\}$$

• 3° round:  $p_3$  e  $p_4$  spediscono a tutti i processi, la conoscenza sarà:

$$\text{quar} = \text{quar} - \{1, 0, 0, 0\} - \{1, 0, 0, 0\}$$

A questo punto, si decide tramite una regola di decisione, ad esempio max W,



PROBLEMA PERCHÉ DECIDE A  $t+1$  & NON A  $t+0$   $t+2$ ?

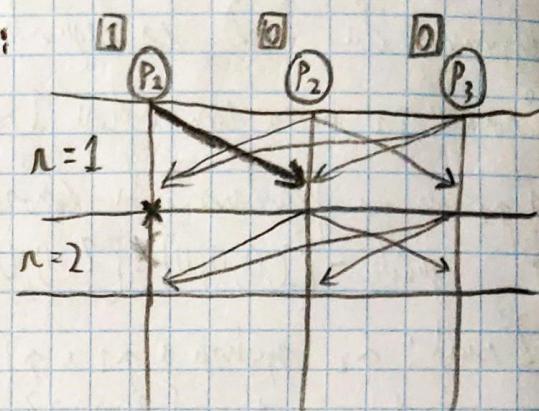
L'appuntamento di avere 3 processi che dividono con 1, 0, 0 valori rispettivamente con  $t=1$  numero di quanti e che  $p_1$  spedisce a  $p_2$  e  $p_3$  in questo, nel 1° round:

- 1° round:  $p_1$  spedisce a  $p_2$  e  $p_3$  in questo, ~~consenso~~:

$$\text{quanto} - \{1, 0, 0\} - \{?, 0, 0\}$$

- 2° round:  ~~$p_2$  e  $p_3$  spediscono a tutti, consenso!~~

$$\text{quanto} - \{1, 0, 0\} - \{1, 0, 0\}$$



Ottro esempio di esecuzione mostrano che i processi non quanti ottengono la stessa concordanza, ma lo stesso insieme di valori W, al round  $t+1$  (2° nell'esempio), pertanto dal round  $t+1$  da poi possono decidere. Il round  $t+2$  la concordanza dei processi non quanti continua ad essere semiforme, pertanto, si puo' decidere anche al 3° round  $t+2$ .

Invece, al round  $t$  la decisione non puo' essere presa siccome alcuni processi non quanti non hanno conoscenza di valori di alcuni processi, n' necessita di un altro round di scambio messaggi, nell'esempio al 1° round ( $t=1$ ).  $p_3$  non ha conoscenza di  $p_2$ .

BSPC(1210 6

POROSI MA SCEGLIE IL MIN!

CAP 6

Suggeriamo di avere 3 processi che arrivano con 0, 1, 1 e non

$f=1$  questi e chi  $p_2$  spedisce a  $p_2$  e poi si guardi nel 1° round:

- 1° round:  $p_2$  spedisce a  $p_2$  e i questi, mentre  $p_2$  e  $p_3$  spediscono

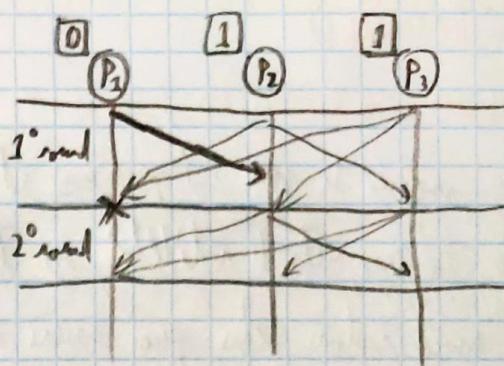
a tutti, la conoscenza sarà:

$$\text{gusta} = \min\{0, 1, 1\} = 0 - \min\{?, 1, 1\} = 1$$

- 2° round:  $p_2$  e  $p_3$  spediscono a tutti, la conoscenza sarà:

$$\text{gusta} = \min\{0, 1\} = 0 - \min\{1, 0\} = 0$$

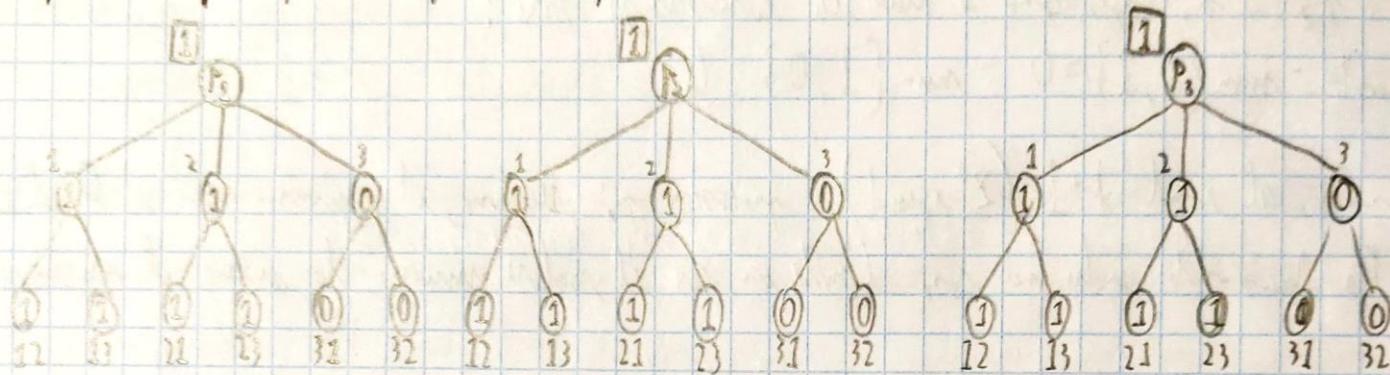
A questi punti, al round  $f+1$  (2° round) i processi  $p_i$  dovranno il minimo che è 0 ed è esatto. Ha funzionato anche nel caso lassisti in cui il valore minimo lo avesse il processo gusti.



BIGSTOP MA BIZANTINO (VIOLAZIONE PROPRIÀ 'VALIDITÀ')

Supponiamo di avere 3 processi ed 1 di essi bizantino.

Per la proprietà di validità, se tutti i processi hanno valore massimo pari a 1, allora 1 è l'unico valore che può essere inserito in  $\mathcal{W}$ . Perché ogni processo non guarda bisogna solo 1 in  $\mathcal{W}$ , al round  $j+1$  si deciderà 1, ma se uno di essi è bizantino non funziona. Esempio: se  $p_1$  e  $p_2$  non guardano 0,  $p_3$  è bizantino e mette 0 andrà 1:



3 valori massimi dei 3 processi è 1,1,1 pertanto si dovrebbe decidere 1 per la proprietà di validità, ma  $p_3$  è bizantino e mette 0 al posto di 1. Perché, BIGSTOP decide il valore di decisione in base alle seguenti regole:

Se  $|\mathcal{W}| = 1$  allora il processo decide sull'unico valore in  $\mathcal{W}$ , altrimenti decide un valore di default, no.

Supponiamo che  $\mathcal{W} = \{0, 1\}$  abbiamo che la decisione all'esempio è 0 siccome  $|\mathcal{W}| > 1$ , ovvero

### CONSENSO GS CON EARLY STOPPING ( $f' < f$ ):

È possibile usare l'algoritmo FloodSET, siccome non vogliamo che il numero di gessiti  $f$  sia  $0 \leq f \leq n$ , ma se  $f' < f$  continuerà a valutare la condizione e sarà  $0 \leq f' \leq f$ .

ALGORITMO:

$$round = round + 1$$

Sia  $X_j$  il messaggio inviato da  $j$ ,  $V_j$  da cui un messaggio arriva

$$W = W \cup U_j X_j$$

IF  $f' < f$  allora  $f' < f$  gessito, THEN  $f = f'$

IF  $round = f$  THEN  $f = f + 1$

IF  $|W| = 1$  THEN

decision =  $N$ , dove  $W = \{N\}$

ELSE decision =  $N_0$  (valore default)

EARLY STOPPING: È soddisfatta, ma come se si verificasse  $f' < f$  quanto le terminazioni non realizzate al round  $f' + 1$ .

ALTRIMENTI:

contiamo in FloodSET quanti gessiti  $f'$  ci sono e se arriva ad  $f'$ , si chiama a  $f' + 1$  round.

EERC1210 1.1

CAP 6

TEOREM (PAG. 171)

EERC1210 PRECEDENTE MA CON GUASTI BIENNINI:

Sappiamo che i nodi sabino  $n > 3$  s'oggetto ai nodi balsamici (ricorre per  $n \leq 3$  il problema del consensus ma è risolvibile da dei lemma studiati), può essere usato EIGByz:  
I processori propongono i valori disponibili per  $\delta^t + 1$  round, messaggi non conformi vengono ignorati. Alla fine del round  $\delta^t + 1$  il valore dell'elenco che sono nulli vengono trascurati dal valore  $No$  (di default).

Per decidere, i nodi utilizzano EIG puntando sulle foglie per arrivare alle radici, calcolando un nuovo valore per ogni nodo interno come la maggioranza stretta dei figli del nodo, se non esiste maggioranza stretta si pone il nuovo valore a  $No$ .

• PART STOPPING: //



MIRAVVA:

Contiamo in EIGByz quanti processori mandano messaggi non conformi e se arriva ad  $\delta^t$ , si riporta ad  $\delta^{t+1}$  round.

ESERCIZIO 1210 '14 (AP 6)

TRONCA (PAG. 115)

PAXOS 5 PROCESSORI CON 3 ROUND:

- (1, A): Il processore A invia il round (1, A) e A, B e D rispondono con LAST garantendo di non accettare i successivi round con numero inferiore.

A non è la maggioranza di nessun round precedente,

pertanto, è libero di scegliere il valore binario. I processori A e D accettano il valore proposto, ma non si raggiunge il consenso perché non c'è una maggioranza.

- (2, B): Il processore B invia il round (2, B), B, C ed E inviano LAST impegnandosi. Il valore proposto da B non è accettato solo da B, pertanto, non c'è decisione.

- (3, A): Il processore A invia il round (3, A) e A, C, D ed E mandano LAST, in più, ottengono una maggioranza di ACCEPT, pertanto, si decide il valore  $v_A$ .

I successivi round con numero inferiore non potranno avere una maggioranza, mentre round con numero superiore potranno proporre solo il valore  $v_A$ .

| N° round | value | A | B | C | D | E |
|----------|-------|---|---|---|---|---|
| (1, A)   | $v_A$ | ✓ |   |   |   | ✓ |
| (2, B)   | $v_B$ |   | ✓ |   |   |   |
| (3, A)   | $v_A$ | ✓ |   | ✓ | ✓ | ✓ |