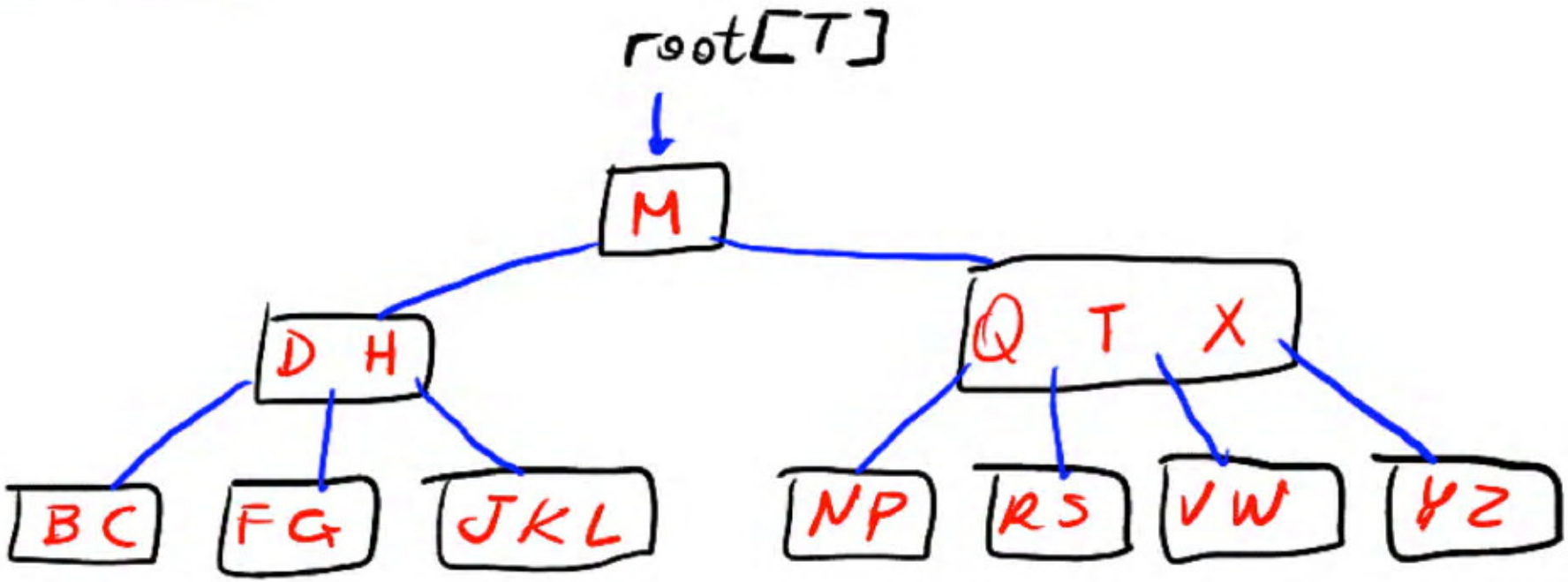
B-TREES

- I B-TREE SONO ALBERI BILANCIATI DI RICERCA PARTICOLARMENTE ADATTI PER ESSERE UTILIZZATI SU MEMORIA SECONDARIA (E QUINDI PER MANTENERE GRANDI QUANTITA' DI DATI)
- A DIFFERENZA DI ALTRI TIPI DI ALBERI BILANCIATI, I NODI DEI B-TREE POSSONO AVERE UN FATTORE DI RAMIFICAZIONE MOLTO ALTO.
- CIO' CONSENTE AI B-TREE DI AVERE UN'ALTEZZA PIU' BASSA RISPETTO AD ALTRI TIPI DI ALBERI BILANCIATI, A PARITA' DI NUMERO DI RECORD

ESEMPIO

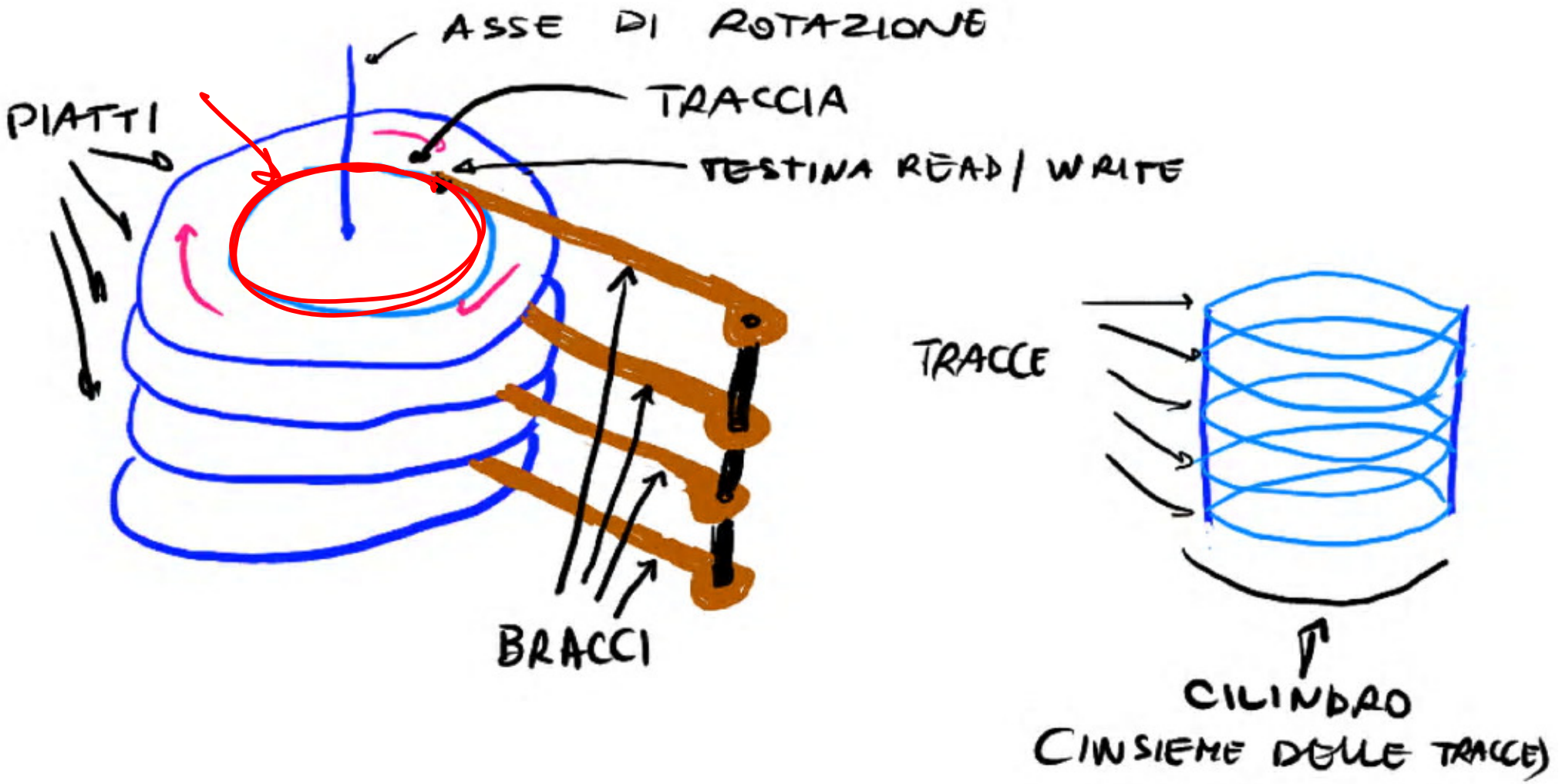


Digressione sulla memoria secondaria

- MEMORIA PRIMARIA O PRINCIPALE
  BASATA SU CHIP DI MEMORIA AL SILICIO
- MEMORIA SECONDARIA
  BASATA SU DISCHI MAGNETICI (O, PEGGIO ANCORA, SU NASTRI MAGNETICI)
- IL COSTO DELLA MEMORIA SECONDARIA E' TIPICAMENTE CIRCA 100 VOLTE INFERIORE A QUELLA PRIMARIA (PER BIT)

- I TEMPI DI ACCESSO ALLA MEMORIA SECONDARIA SOLITAMENTE SONO DI CIRCA 10^5 VOLTE SUPERIORI A QUELLI DELLA MEMORIA PRIMARIA
- PERTANTO GLI ALGORITMI CHE UTILIZZANO STRUTTURE DATI RESIDENTI SU MEMORIA SECONDARIA VANNO ANALIZZATI ANCHE IN BASE AL NUMERO DI ACCESSI READ/WRITE ALLA MEMORIA SECONDARIA

SCHEMA DI DISCO MAGNETICO



- LA VELOCITA' DI ROTAZIONE DI UN DISCO MAGNETICO SOLITAMENTE VARIA DA 5400 A 15000 ROTAZIONI/MIN

- UNA ROTAZIONE COMPLETA RICHIEDE 8,33 mSEC (CIRCA 10^5 PIU' DEI 100 nanoSEC RICHIESTI DA UN ACCESSO ALLA MEMORIA PRINCIPALE)

- IL MOVIMENTO DELLE TESTINE RICHIEDE UN TEMPO VARIABILE DA 3 A 9 mSEC

- PER AMMORTIZZARE GLI ELEVATI TEMPI DI POSIZIONAMENTO DELLE TESTINE SULLE TRACCE DA LEGGERE/SCRIVERE, L'INFORMAZIONE E' ORGANIZZATA SUI CILINDRI MEDIANTE BLOCCHI DELLA STESSA DIMENSIONE, DETTI PAGINE

- LA DIMENSIONE TIPICA DI UNA PAGINA VA DAI 2^{11} AI 2^{14} BIT
- RISULTA QUINDI CONVENIENTE ORGANIZZARE LE STRUTTURE DATI RESIDENTI SU MEMORIA SECONDARIA IN MODO DA SFRUTTARE LA SUDDIVISIONE IN PAGINE IN QUANTO GLI ACCESSI IN LETTURA E/O SCRITTURA AVVENGONO SU PAGINE INTERE
- ANALIZZEREMO GLI ALGORITMI PER LA GESTIONE DEI B-TREE IN BASE A:
  - NUMERO DI ACCESSI AL DISCO
  - TEMPO DI CPU

- UN OGGETTO RESIDENTE IN MEMORIA SECONDARIA, CUI SI ACCEDE MEDIANTE PUNTATORE x, SARA' MANIPOLATO COME SEGUE:

  DISK-READ(x)

  OPERAZIONI CHE LEGGONO/SCRIVONO I CAMPI DI x

  DISK-WRITE(x)

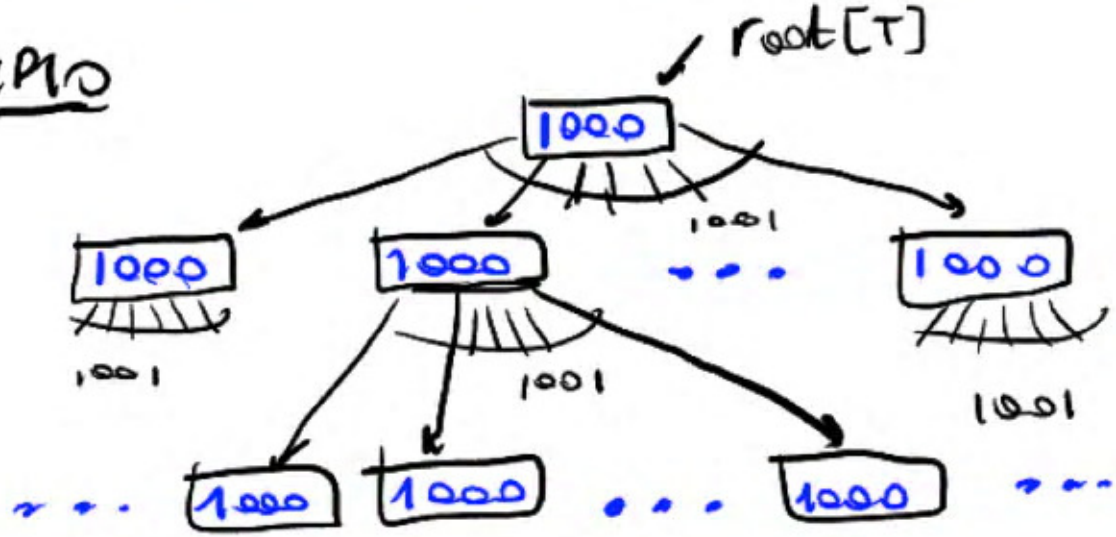
  ALTRE OPERAZIONI CHE ACCEDONO AI CAMPI DI x SENZA MODIFICARLI

- I B-TREE SONO UTILIZZATI PER GESTIRE GRANDI QUANTITA' DI DATI CHE NON POSSONO ESSERE MANTENUTI IN MEMORIA PRIMARIA

- TIPICAMENTE LA DIMENSIONE DI UN NODO IN UN B-TREE COINCIDE CON QUELLA DI UNA PAGINA

- TIPICI VALORI PER I FATTORI DI RAMIFICAZIONE DI UN B-TREE PER LA GESTIONE DI GRANDI QUANTITÀ DI DATI VANNO DA 50 A 2000, IN BASE ANCHE ALLA DIMENSIONE DI UNA PAGINA

ESEMPIO



1 NODO – 1000 CHIAVI

10

1001 NODI

1001.000 CHIAVI

1002 001 NODI

1002 001 000 CHIAVI

B-TREE DI ALTEZZA 2 E FATTORE DI RAMIFICAZIONE 1001

- PER SEMPLICITÀ SUPPORREMO CHE OGNI INFORMAZIONE COLLATERALE RIGUARDANTE UN CERTO OGGETTO SIA MEMORIZZATA NELLO STESSO NODO DEL B-TREE CONTENENTE LA CHIAVE DELL'OGGETTO STESSO.
- LA VARIANTE DEI B^+-TREE MANTIENE GLI OGGETTI SULLE FOGLIE E LE CHIAVI E PUNTATORI A FIGLI SUI NODI INTERNI

Definizione di B-tree

- UN B-TREE T E' UN ALBERO DI RICERCA CON RADICE $root[T]$, CHE GODE DELLE SEGUENTI PROPRIETA':

1. CIASCUN NODO x HA I SEGUENTI CAMPI:
   - $n[x]$, NUMERO DELLE CHIAVI IN x
   - LE $n[x]$ CHIAVI IN ORDINE NON DECRESCENTE:

     $$key_1[x] \leq key_2[x] \leq \ldots \leq key_{n[x]}[x]$$

   - $leaf[x]$, CAMPO BOOLEANO TALE CHE

     $leaf[x] = VERO \longrightarrow x$ E' UNA FOGLIA

     $leaf[x] = FALSO \longrightarrow x$ E' UN NODO INTERNO

./.

2. CIASCUN NODO INTERNO x CONTIENE $n[x]+1$ PUNTATORI AI FIGLI $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$

3. SE k_i DENOTA UNA CHIAVE NEL SOTTOALBERO DI RADICE $c_i[x]$, PER $i = 1, 2, \ldots, n[x]+1$, VALE:

$$k_1 \leq key_1[x] \leq k_2 \leq key_2[x] \leq \ldots \leq key_{n[x]}[x] \leq k_{n[x]+1}$$

(DA CUI SEGUE CHE IL B-TREE È UN ALBERO DI RICERCA)

4. TUTTE LE FOGLIE HANNO LA STESSA PROFONDITÀ UGUALE QUINDI ALL'ALTEZZA h DEL B-TREE

./.

5. IL B-TREE E' CARATTERIZZATO DA UNA QUANTITA' $t \geq 2$, DETTA GRADO MINIMO, TALE CHE

- TUTTI I NODI, AD ECCEZIONE DELLA RADICE, DEBBONO CONTENERE ALMENO $t-1$ CHIAVI.

  SE IL B-TREE E' NON-VUOTO, ALLORA
  - LA RADICE DEVE CONTENERE ALMENO UNA CHIAVE
- TUTTI I NODI DEBBONO CONTENERE AL PIÙ $2t-1$ CHIAVI
  - UN NODO CHE CONTIENE $2t-1$ CHIAVI SI DICE PIENO

ESEMPIO: $t=2 \longrightarrow$ 2-3-4 TREE

TEOREMA (ALTEZZA DI UN B-TREE)

L'ALTEZZA h DI UN B-TREE T DI GRADO MINIMO t CONTENENTE n CHIAVI, CON $n \geq 1$, SODDISFA

$$h \leq \log_t \frac{n+1}{2}$$

DIM SIA T COME NELL'ENUNCIATO E SIA T' IL B-TREE DI ALTEZZA h CONTENENTE IL MINOR NUMERO POSSIBILE DI CHIAVI.

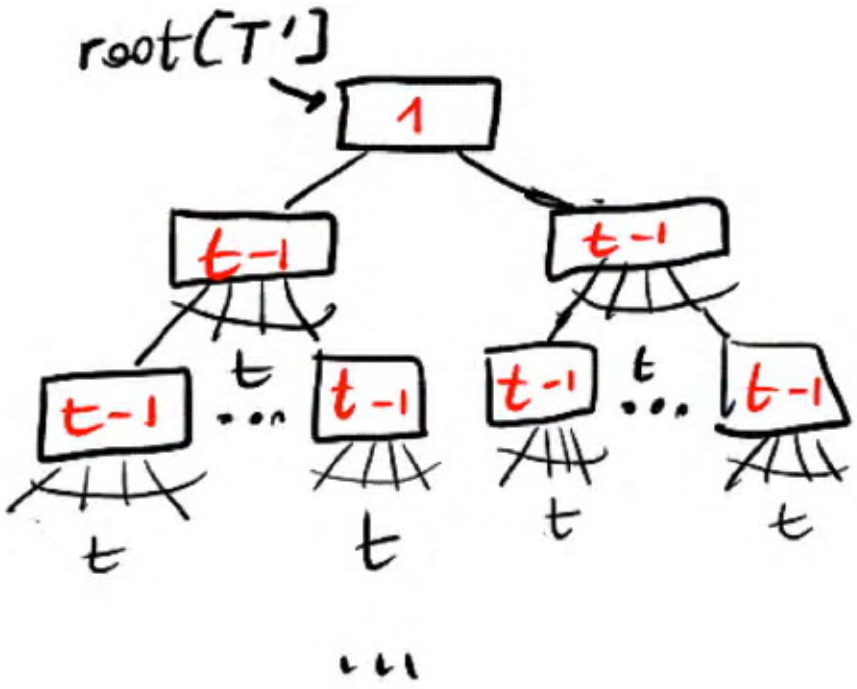
SIA n' IL NUMERO DI CHIAVI CONTENUTE IN T'.

OVVIAMENTE: $n' \leq n$.

DIMOSTREREMO CHE

$$h = \log_t \frac{n'+1}{2} \leq \log_t \frac{n+1}{2},$$

DA CUI LA TESI.



PROFONDITÀ	# NODI	# CHIAVI
0	1	1
1	2	$2(t-1)$
2	$2t$	$2t(t-1)$
3	$2t^2$	$2t^2(t-1)$
⋮	⋮	⋮
h	$2t^{h-1}$	$2t^{h-1}(t-1)$

$$n' = 1 + 2(t-1)\sum_{i=1}^{h-1} t^{i-1} = 1 + 2(t-1)\frac{t^h-1}{t-1} = 2t^h - 1 =$$

$$\Rightarrow 2t^h = n'+1 \Rightarrow t^h = \frac{n'+1}{2} \Rightarrow h = \log_t \frac{n'+1}{2} \quad \blacksquare$$

OPERAZIONI DI BASE SUI B-TREE

- B-TREE-SEARCH
- B-TREE-CREATE
- B-TREE-INSERT
- B-TREE-DELETE

- SUPPORREMO CHE:
  - LA RADICE DEL B-TREE SIA SEMPRE IN MEMORIA PRINCIPALE
  - PER TUTTI I NODI PASSATI COME PARAMETRO DEVE ESSERE STATA CHIAMATA LA PROCEDURA DISK-READ
- PRESENTEREMO ALGORITMI A SINGOLA PASSATA (TRANNE IN ALCUNI CASI DI B-TREE-DELETE)

B-TREE-SEARCH (x, k)

(x PUNTA ALLA RADICE, k E' LA CHIAVE CERCATA)

```
i := 1
while i ≤ n[x] and k > key_i[x] do
    i := i + 1
if i ≤ n[x] and k = key_i[x] then
    return (x, i)
if leaf[x] then
    return NIL
else
    DISK-READ( c_i[x])
    return B-TREE-SEARCH (c_i[x], k)
```

COMPLESSITA'

$O(h) = O(\log_t n)$ ACCESSI AL DISCO

$O(th) = O(t \log_t n)$ TEMPO DI CPU

B-TREE-CREATE(T)

(CREA UN B-TREE VUOTO T)

x := ALLOCATE-NODE()

leaf[x] := TRUE

n[x] := 0

DISK-WRITE(x)

root[T] := x

COMPLESSITA'

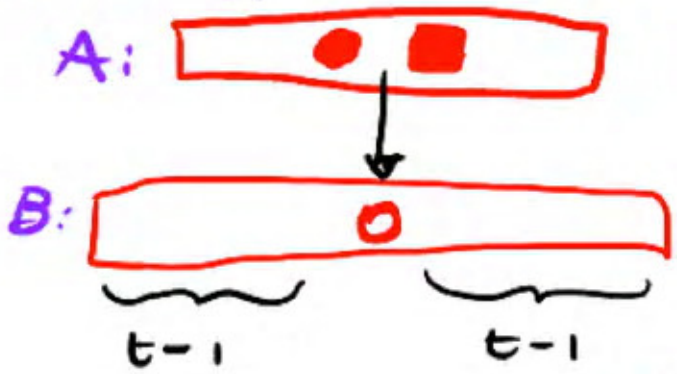
$O(1)$ ACCESSI AL DISCO

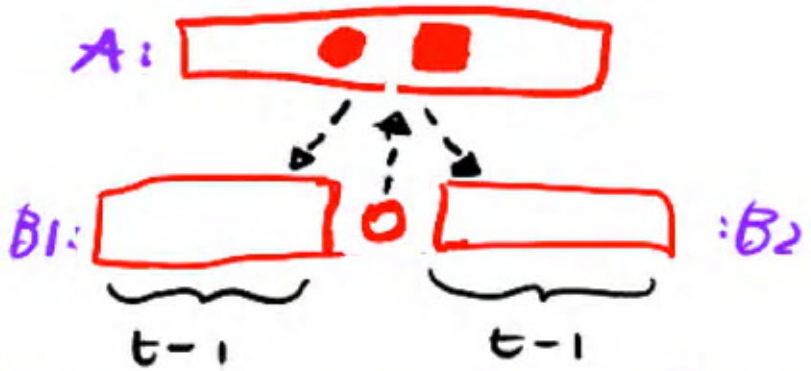
$O(1)$ TEMPO DI CPU

INSERIMENTO DI UNA CHIAVE IN UN B-TREE

PROCEDURA NATURALE (DUE POSSIBILI PASSATE)

- SI CHIAMI B-TREE-SEARCH (root[T], k) PER LOCALIZZARE LA FOGLIA IN CUI INSERIRE k
- SE TALE FOGLIA NON E' PIENA, SI INSERISCA k NEL POSTO OPPORTUNO
- ALTRIMENTI





- INSERIRE k OVE OPPORTUNO IN B1 O B2
- INSERIRE CHIAVE O IN A RICORSIVAMENTE

- PER EVITARE LA POSSIBILE RISALITA SINO ALLA RADICE, SI FARA' IN MODO DI SUDDIVIDERE I NODI PIENI NON APPENA VENGONO INCONTRATI DURANTE LA FASE DI LOCALIZZAZIONE DELLA FOGLIA IN CUI INSERIRE LA NUOVA CHIAVE k.

- A TAL FINE SI UTILIZZERA' LA PROCEDURA AUSILIARIA B-TREE-SPLIT-CHILD (x, i) CON
  - x : PUNTATORE AD UN NODO NON PIENO
  - i : INDICE TALE CHE $c_i[x]$ SIA PIENO

- L'EFFETTO SARA' DI SPEZZARE IL NODO $c_i[x]$ IN DUE NODI $c_i[x]$ E z

```
B-TREE-SPLIT-CHILD(x, i)
y := c_i[x]
z := ALLOCATE-NODE()
leaf[z] := leaf[y]
n[z] := t-1
for j := 1 to t-1 do
    key_j[z] := key_{j+t}[y]
if not leaf[y] then
    for j := 1 to t do
        c_j[z] := c_{j+t}[y]
n[y] := t-1
for j := n[x]+1 downto i+1 do
    c_{j+1}[x] := c_j[x]
c_{i+1}[x] := z
for j := n[x] downto i do
    key_{j+1}[x] := key_j[x]
key_i[x] := key_t[y]
n[x] := n[x]+1
DISK-WRITE(y);
DISK-WRITE(z);
DISK-WRITE(x)
```

COMPLESSITÀ DI B-TREE-SPLIT-CHILD(x, i)

$\Theta(1)$ ACCESSI AL DISCO

$\Theta(t)$ TEMPO DI CPU

```
B-TREE-INSERT-NONFULL (x, k)
i := n[x]
if leaf[x] then
    while i ≥ 1 and k < key_i[x] do
        key_{i+1}[x] := key_i[x] ; i := i - 1 ;
    key_{i+1}[x] := k
    n[x] := n[x] + 1
    DISK-WRITE (x)
else
    while i ≥ 1 and k < key_i[x] do
        i := i - 1
    i := i + 1
    DISK-READ (c_i[x])
    if n[c_i[x]] = 2t - 1 then
        B-TREE-SPLIT-CHILD (x, i)
        if k > key_i[x] then
            i := i + 1
    B-TREE-INSERT-NONFULL (c_i[x], k)
```

COMPLESSITÀ DI B-TREE-INSERT-NONFULL (x, k)

$\Theta(h) = \Theta(\log_t n)$ ACCESSI AL DISCO

$O(th) = O(t \log_t n)$ TEMPO DI CPU

- SI OSSERVI CHE NONOSTANTE LA PROCEDURA SIA RICORSIVA (MA CON SEMPLICE RICORSIVITÀ DI CODA) IL NUMERO DI PAGINE CHE AD OGNI ISTANTE DEBBONO RISIEDERE IN MEMORIA PRINCIPALE È $O(1)$

```
B-TREE-INSERT (T, k)
  r := root[T]
  if n[r] = 2t-1 then
     s := ALLOCATE-NODE ()
     root[T] := s
     leaf[s] := FALSE
     n[s] := 0
     c1[s] := r
     B-TREE-SPLIT-CHILD ( s, 1)
     B-TREE-INSERT-NONFULL (s, k)
  else
     B-TREE-INSERT-NONFULL (r, k)
```

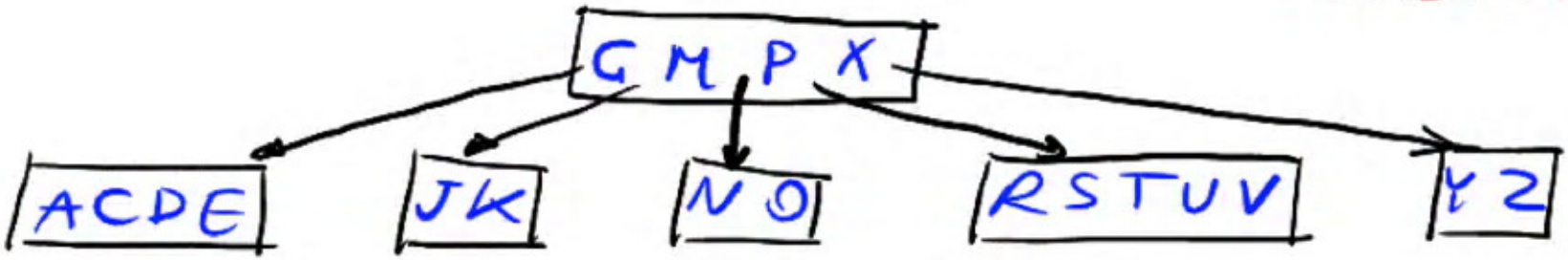
COMPLESSITÀ DI B-TREE-INSERT (T, k)

$\Theta(h) = \Theta(\log_t n)$ ACCESSI AL DISCO

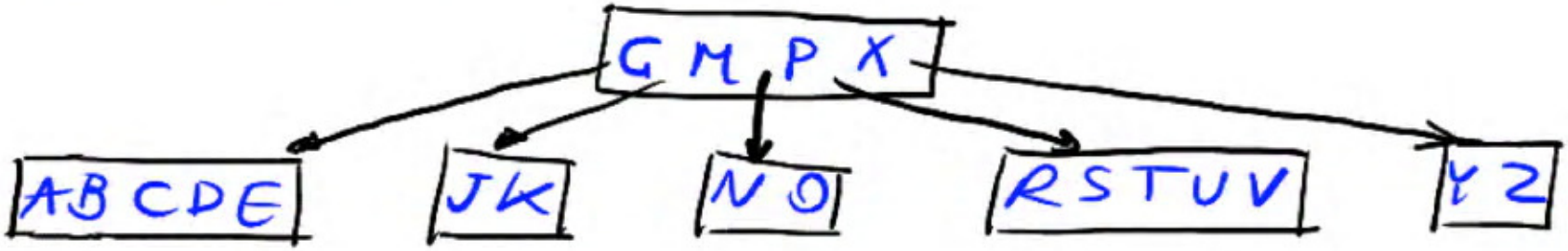
$O(th) = O(t \log_t n)$ TEMPO DI CPU

ESEMPIO

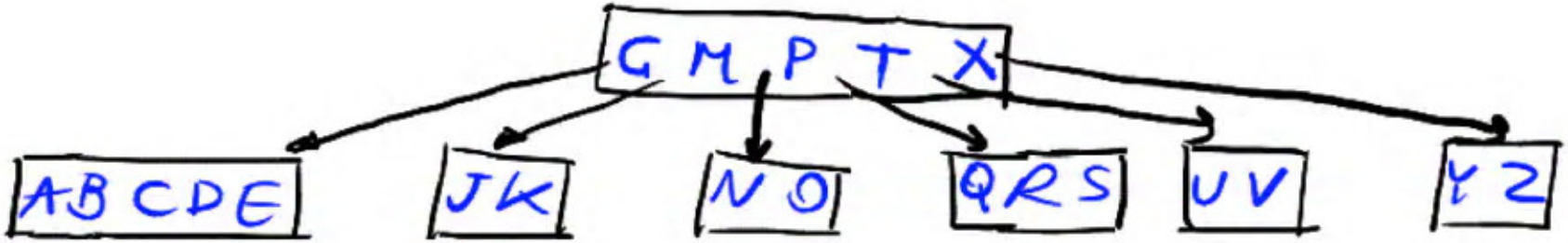
B-TREE DI GRADO MINIMO 3



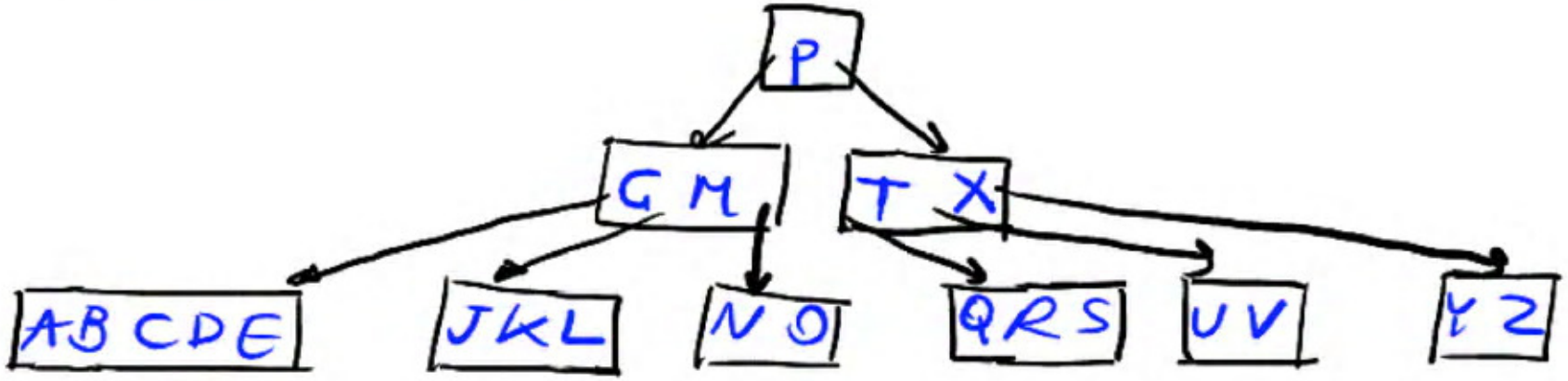
- INSERIMENTO DI B

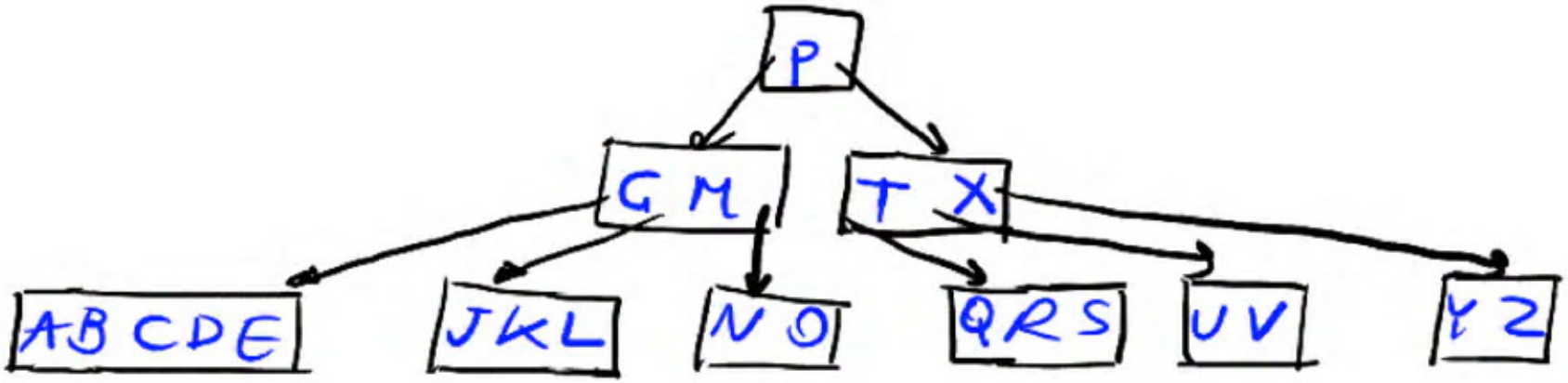


- INSERIMENTO DI Q

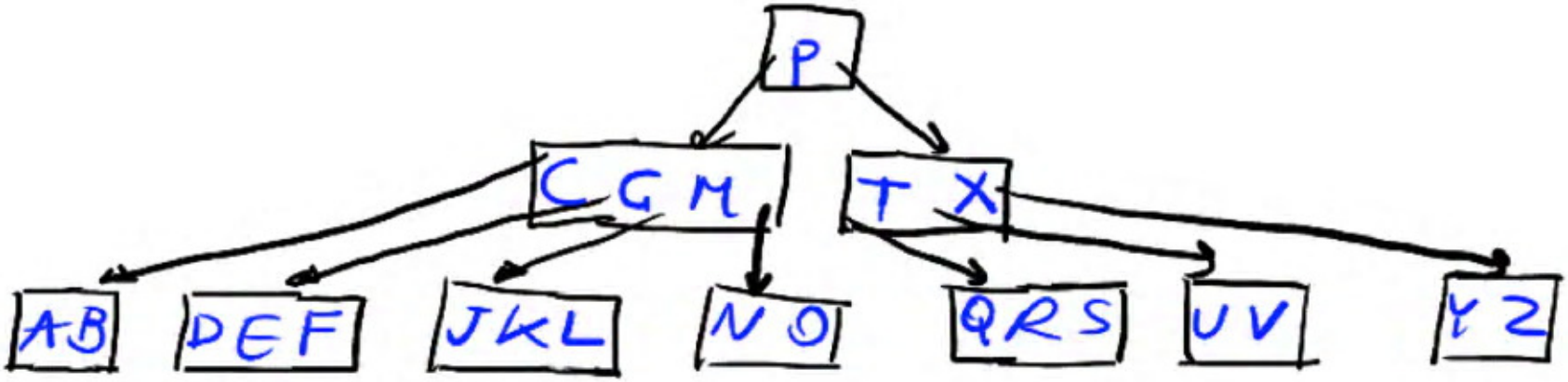


- INSERIMENTO DI L





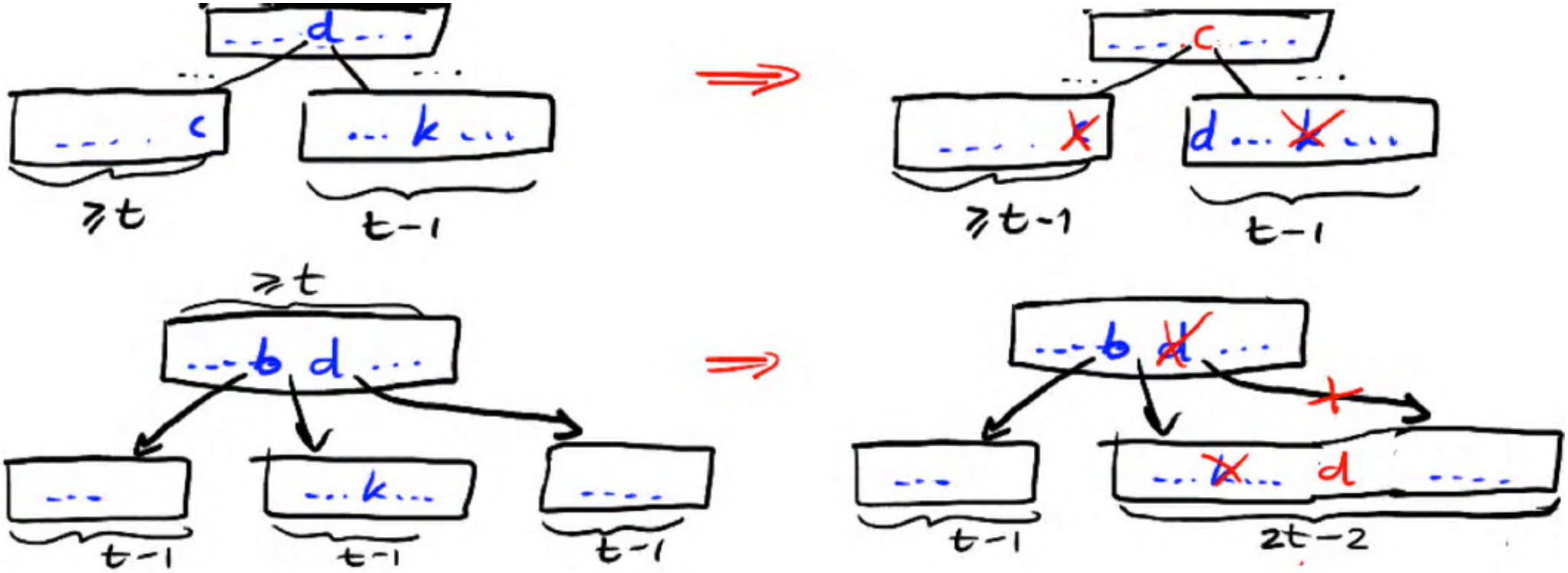
- INSERIMENTO DI F



CANCELLAZIONE DI UNA CHIAVE DA UN B-TREE

PROCEDURA NATURALE (DUE POSSIBILI PASSATE, ANCHE QUANDO LA CHIAVE DA CANCELLARE RISIEDE SU UNA FOGLIA)

- SI CHIAMI B-TREE-SEARCH (root[T], k) PER LOCALIZZARE IL NODO CHE CONTIENE LA CHIAVE k
- SE TALE NODO E' UNA FOGLIA CON ALMENO t CHIAVI (CIOE' SE TALE NODO NON E' MAGRO) SI CANCELLI LA CHIAVE k
- SE INVECE TALE NODO E' UNA FOGLIA "MAGRA" SI FARA' CEDERE UNA CHIAVE DA UNA FOGLIA SORELLA (SE POSSIBILE) OPPURE DAL NODO PADRE, CONTESTUALMENTE UNENDOSI AD UNA FOGLIA SORELLA. TALE PROCESSO POTREBBE PROPAGARSI SINO ALLA RADICE



- SE IL NODO CONTENENTE LA CHIAVE k E' INVECE UN NODO INTERNO, SI SOSTITUISCA k CON LA CHIAVE IMMEDIATAMENTE PIÙ PICCOLA (O PIÙ GRANDE) DI k E SI CANCELLI IL PREDECESSORE (O SUCCESSORE) DI k, CHE RISIEDERA' SICURAMENTE SU UNA FOGLIA

- PER EVITARE LA POSSIBILE RISALITA (ALMENO IN QUEI CASI IN CUI LA CHIAVE DA CANCELLARE RISIEDE SU UNA FOGLIA – CASO PIÙ PROBABILE) SI FARÀ IN MODO CHE I NODI SUL CAMMINO DALLA RADICE AL NODO CONTENENTE k NON SIANO MAGRI (AD ECCEZIONE DELLA RADICE)

- PERTANTO, IN QUANTO SEGUE SUPPORREMO CHE IL NODO x IN ESAME NON SIA MAGRO (A MENO CHE NON SI TRATTI DELLA RADICE)

- INOLTRE, SUPPORREMO CHE SE LA RADICE RIMANE SENZA ALCUNA CHIAVE, QUESTA VIENE DEALLOCATA

1. SE k E' NEL NODO x ED x E' UNA FOGLIA, SI CANCELLI k DA x

2. SE k E' NEL NODO x ED x E' UN NODO INTERNO

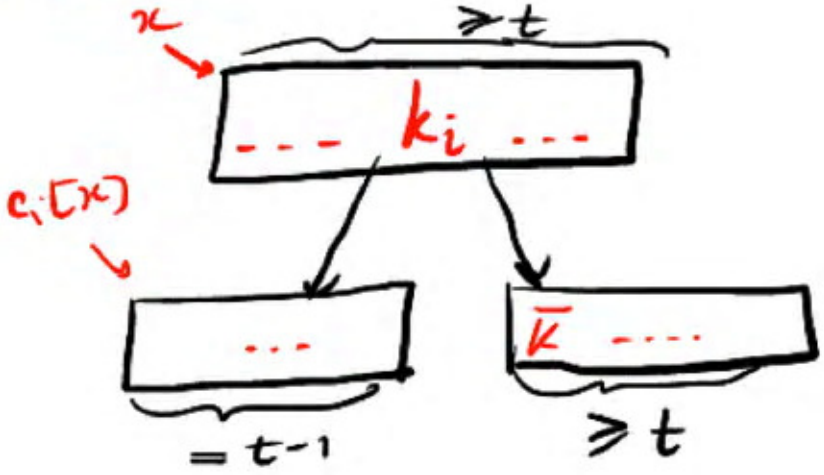
   a. SE IL FIGLIO y DI x CHE PRECEDE k HA ALMENO t CHIAVI, SI DETERMINI IL PREDECESSORE k' DI k NEL SOTTOALBERO RADICATO IN y.
   SI CANCELLI RICORSIVAMENTE k' E SI SOSTITUISCA k CON k'

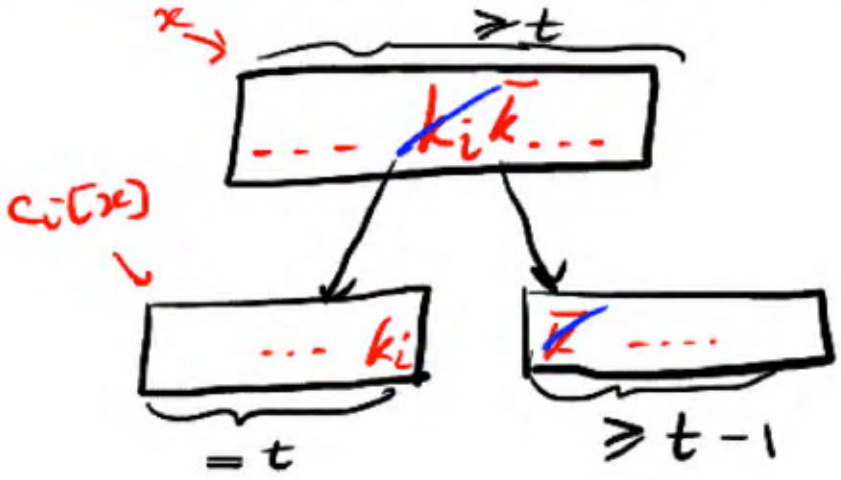
   b. SITUAZIONE SIMMETRICA DI a. PER IL FIGLIO z DI x CHE SEGUE k

   c. ALTRIMENTI, SE SIA y CHE z HANNO $t-1$ FIGLI, SI SI UNISCANO k E IL NODO z A y, SI CANCELLI k DA x, SI DEALLOCHI z, SI CANCELLI RICORSIVAMENTE k DA y

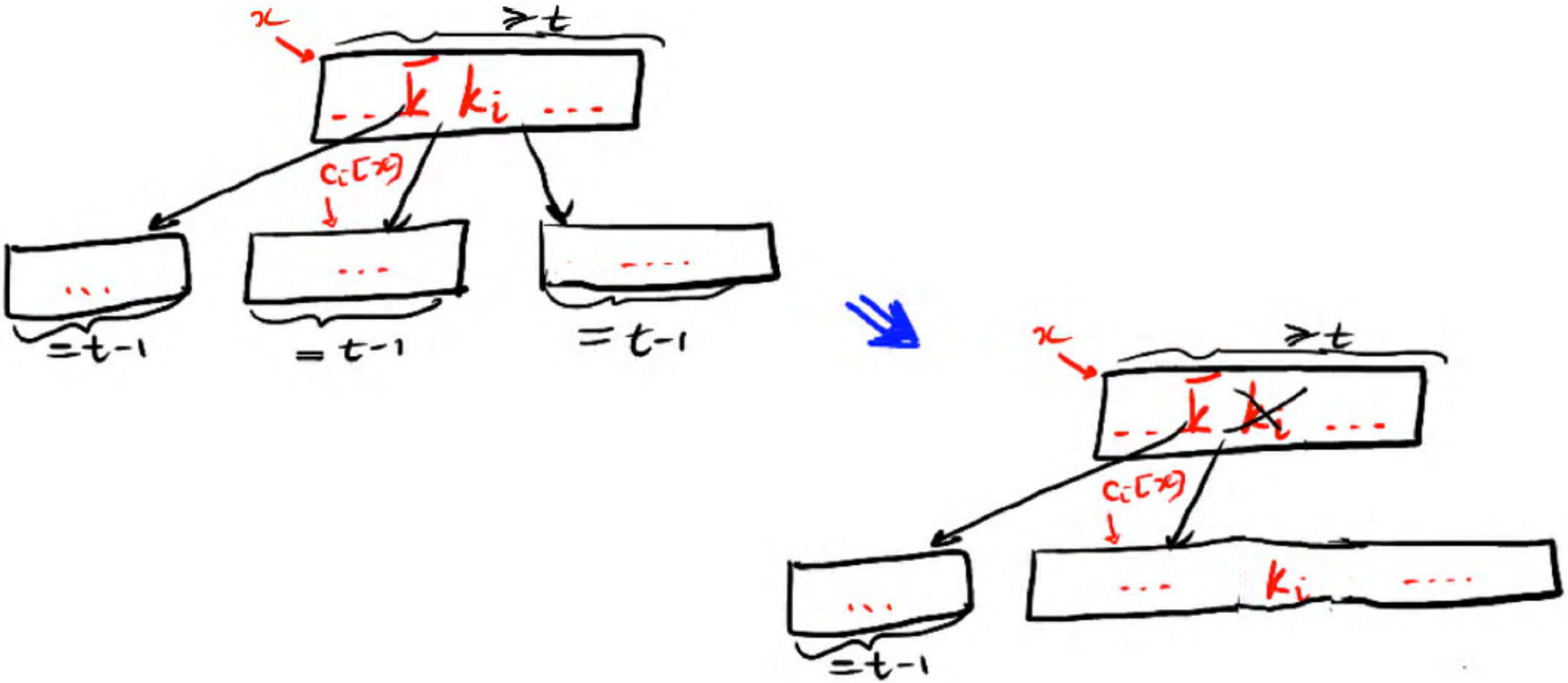
3. SE LA CHIAVE k NON E' PRESENTE NEL NODO INTERNO x, SI DETERMINI LA RADICE $c_i[x]$ DEL SOTTOALBERO DI x CHE CONTIENE k.
SE $c_i[x]$ E' MAGRO, SI ESEGUA IL PASSO 3.a O 3.b E QUINDI RICORSIVAMENTE SI RIMUOVA LA CHIAVE k DAL SOTTOALBERO CHE LA CONTIENE

a. SE $c_i[x]$ HA UN FRATELLO IMMEDIATO NON MAGRO



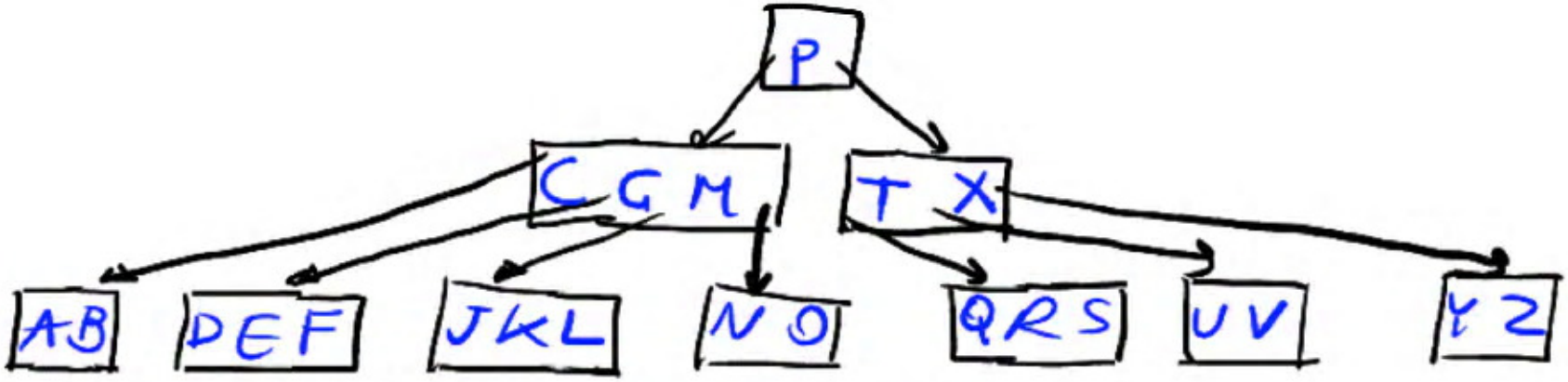


b. SE ENTRAMBI I FRATELLI IMMEDIATI DI $c_i[x]$ SONO MAGRI

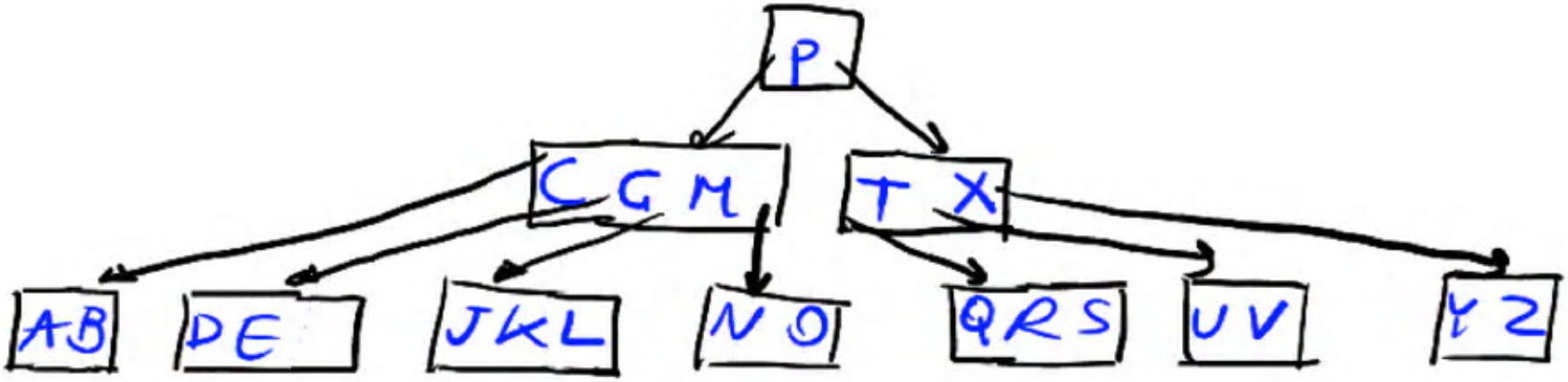


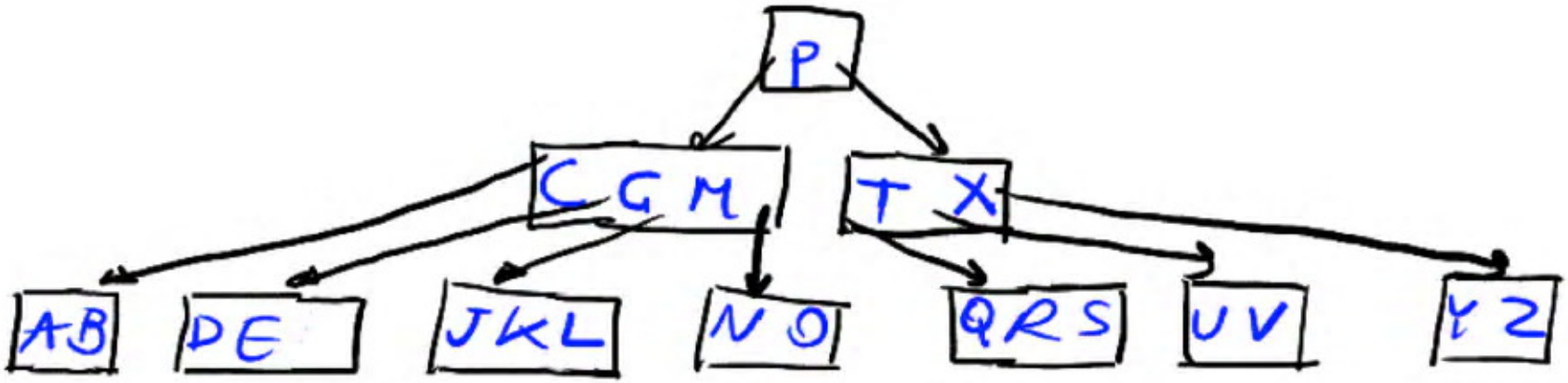
ALTRIMENTI, SE $c_i[x]$ NON È MAGRO, SI PROCEDA RICORSIVAMENTE A CANCELLARE LA CHIAVE k A PARTIRE DAL NODO $c_i[x]$.

ESEMPIO

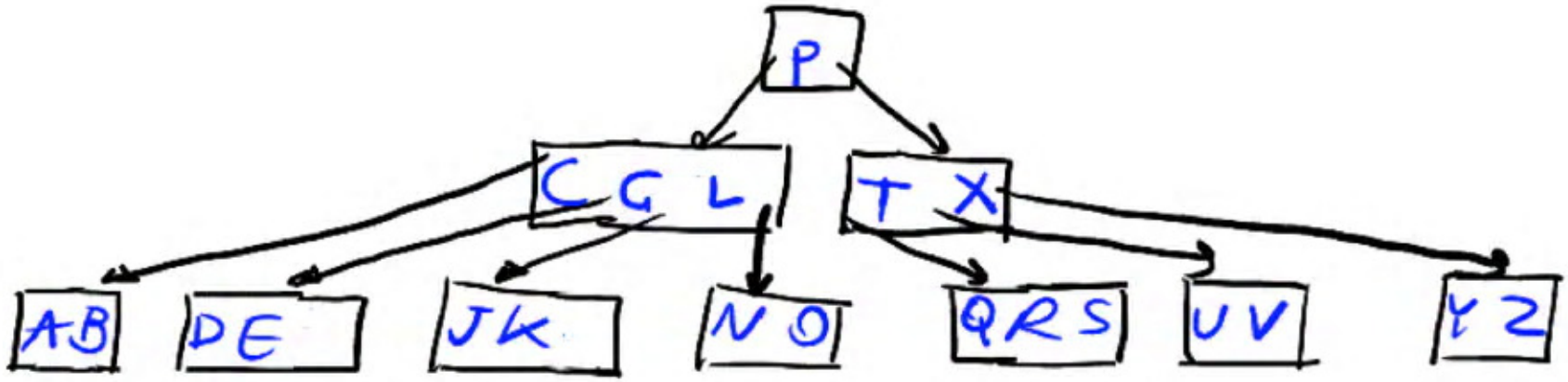


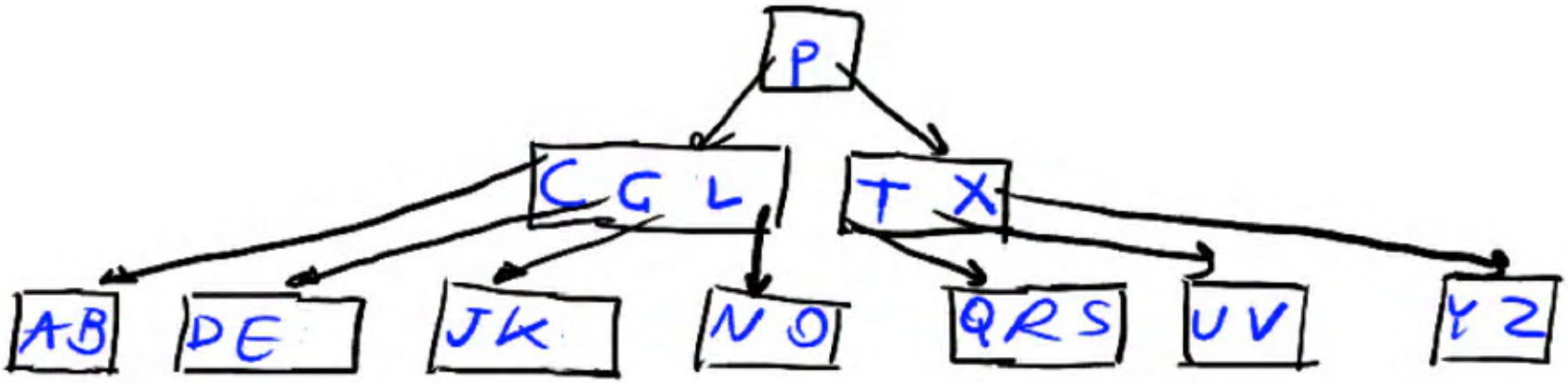
- CANCELLAZIONE DI F (CASO 1)



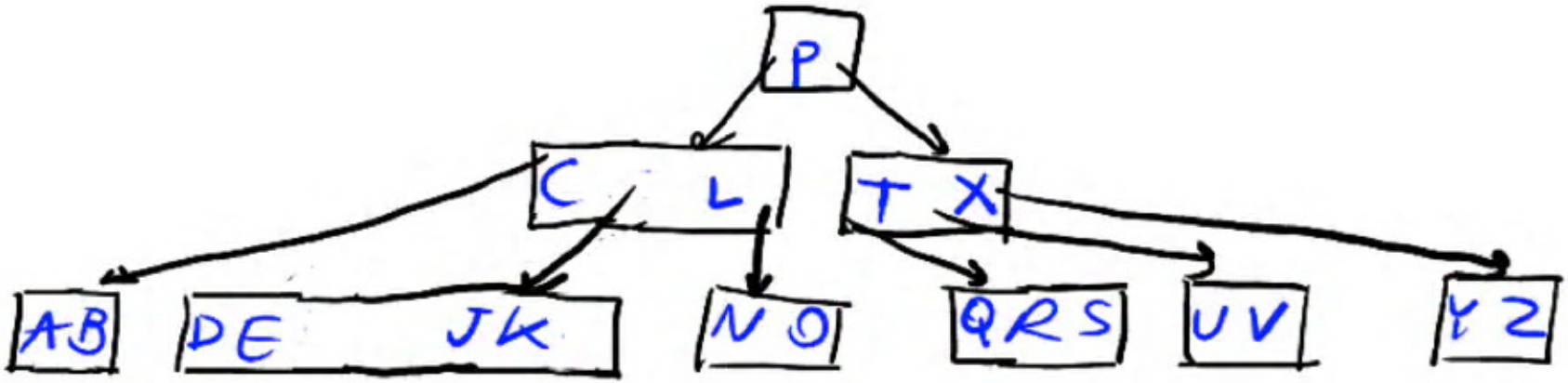


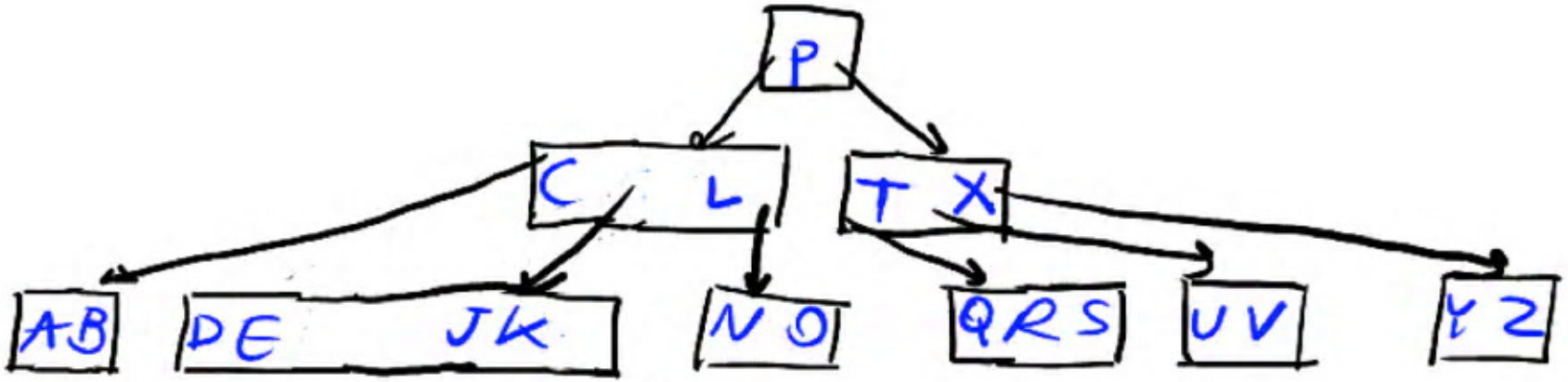
- CANCELLAZIONE DI M (CASO 2a)



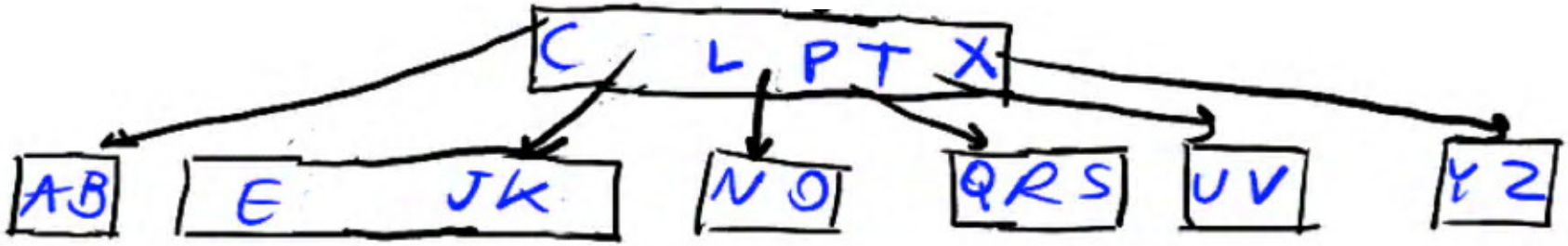


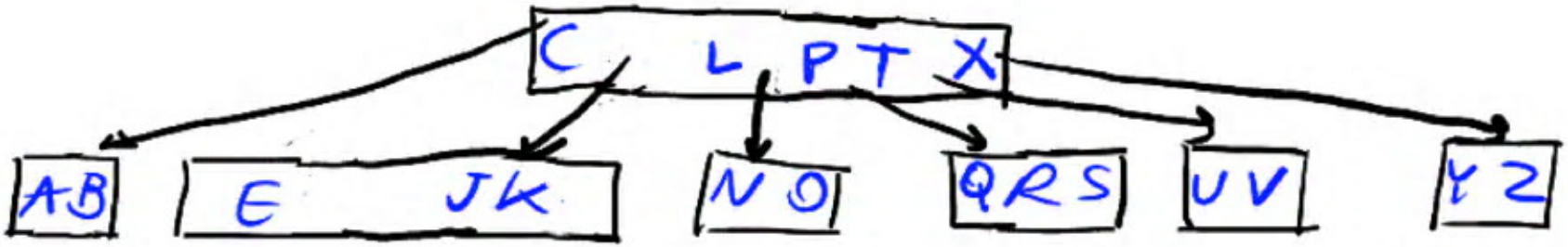
- CANCELLAZIONE DI G (CASO 2C)





- CANCELLAZIONE DI D (CASO 3b)





- CANCELLAZIONE DI B (CASO 3a)