Corso di laurea in Informatica
Algoritmi 1
A.A. 2025/2026

Dizionari: B-Alberi

Tiziana Calamoneri

SAPIENZA
UNIVERSITÀ DI ROMA

Sommario

Oltre gli AVL

B-alberi:

- definizione e proprietà
- dimostrazione del bilanciamento
- algoritmo di ricerca
- algoritmo di inserimento
- algoritmo di cancellazione

Oltre gli alberi AVL (1)

Gli alberi AVL sono ideali per garantire accessi efficienti ai dati grazie al loro bilanciamento, che assicura un costo di O(logn) per le operazioni di ricerca, inserimento e cancellazione.

Tuttavia, presentano alcune limitazioni che emergono in presenza di dati di grandi dimensioni, in cui i dati non risiedono interamente in memoria principale, ma sono invece memorizzati su dispositivi di archiviazione secondaria:

Oltre gli alberi AVL (2)

Elevato numero di accessi a memoria secondaria: Ogni nodo di un albero AVL contiene un singolo valore e due puntatori. Quando la quantità di dati è elevata, non si riesce a sfruttare appieno il blocco di memoria letto con un accesso a memoria esterna (che di solito è più grande di un nodo).

rallentamento delle operazioni a causa del numero elevato di accessi a memoria secondaria

Oltre gli alberi AVL (3)

Altezza dell'albero:

Quando si esegue un algoritmo su alberi AVL, il numero di accessi alla memoria secondaria è proporzionale all'altezza.

Nonostante questa sia logaritmica, il numero di accessi risultante potrebbe risultare troppo elevato.

I **B-alberi** sono stati progettati per mitigare queste problematiche, ottimizzando le operazioni su dispositivi di archiviazione secondaria.

B-Alberi (1)

Nei B-alberi:

- i nodi contengono più chiavi, riducendo il numero dei nodi e quindi l'altezza
- è mantenuto il bilanciamento dopo ogni operazione di modifica, che garantisce altezza logaritmica nel numero di nodi
- un singolo accesso a memoria secondaria carica un intero nodo in memoria principale, permettendo di esaminare più chiavi in una sola volta.

B-Alberi (2)

Def. Un B-albero di grado minimo $t \geq 2$ ha le seguenti proprietà:

- La radice contiene s chiavi memorizzate in ordine crescente e $0 \leq s \leq 2t-1$;
- Ogni altro nodo contiene s chiavi memorizzate in ordine crescente e $t-1 \leq s \leq 2t-1$;
- Un nodo interno con s chiavi, $k_1 < k_2 < \ldots < k_s$ ha s+1 figli. Inoltre:

…

B-Alberi (3)

(segue def. di B-albero)

…

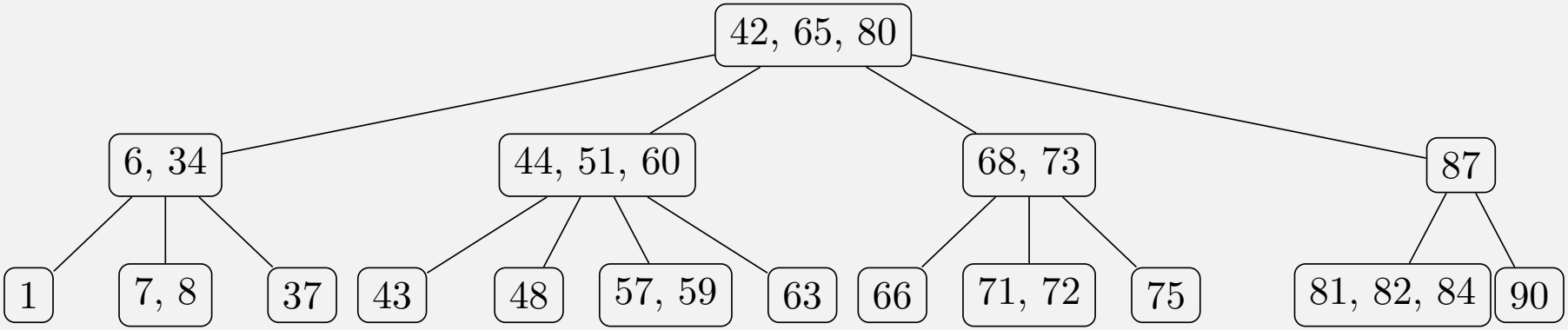
per le chiavi negli s+1 sottoalberi T_i si verifica che:

- le chiavi in T_1 sono minori di k_1;
- le chiavi in T_{s+1} sono maggiori di k_s;
- le chiavi in T_i, con $1< i <s+1$, sono comprese tra k_{i-1} e k_i.

Le foglie si trovano tutte alla stessa profondità.

B-Alberi (4)

Esempio con t=2



Nota 1: tutti i nodi (esclusa la radice) contengono s chiavi con $t-1 \leq s \leq 2t-1$.

Nota 2: con t=100, tre livelli possono rappresentare fino a circa 1000000 di chiavi.

B-Alberi (5)

I B-alberi sono usualmente memorizzati tramite record e puntatori. Ogni record contiene:

- un array *chiavi* di lunghezza massima 2t
- un array *figli* di puntatori di lunghezza massima 2t+1
- un campo *t*, che memorizza questo parametro
- una variabile booleana *is_leaf* che specifica se il nodo sia una foglia

B-Alberi (6)

I B-alberi:

- riducono la necessità di passare da un nodo all'altro
- gestiscono la crescita dinamica dei dati senza richiedere riorganizzazioni complete della struttura: inserimenti e cancellazioni sono progettate per mantenere la struttura bilanciata con un costo estremamente esiguo, non solo in termini asintotici, ma anche di numero di accessi a memoria esterna.

I B-Alberi sono bilanciati (1)

Per dimostrare che i B-alberi sono bilanciati, ovvero che l'altezza è $h=\Theta(\log_t n)$, dove n è il numero di chiavi (non di nodi!) studiamo i casi estremi, cioè quelli in cui ciascun nodo contiene rispettivamente il minimo ed il massimo numero di chiavi possibili.

I B-Alberi sono bilanciati (2)

Caso 1. Minimo numero di chiavi per nodo

- La radice contiene una chiave ed ha due figli
- ogni altro nodo contiene t-1 chiavi ed ha t figli
- a parità di altezza, questo B-albero contiene il minimo numero di nodi possibile ed il suo numero di chiavi n_{min} vale:

$$n_{min} = 1 + 2(t-1)\sum_{j=0}^{h-1} t^j = 2(t-1)\frac{t^h - 1}{t-1} > 2(t^h - 1)$$

Poiché $n \geq n_{min}$: $h < \log_t(n/2+1)$.

I B-Alberi sono bilanciati (3)

Caso 2. Massimo numero di chiavi per nodo

- Ogni nodo (inclusa la radice) contiene 2t-1 chiavi ed ha 2t figli
- a parità di altezza, questo B-albero contiene il massimo numero di nodi possibile ed il suo numero di chiavi n_{max} vale:

$$n_{max} = \sum_{j=0}^{h-1} (2t-1)(2t)^j = (2t-1)\frac{(2t)^{h+1}-1}{2t-1} = (2t)^{h+1} - 1$$

Poiché $n \leq n_{max}$: $h \geq \log_{2t}(n+1)-1$.

I B-Alberi sono bilanciati (4)

Combinando i due risultati:

$$\log_{2t}(n+1)-1 \leq h < \log_t(n/2+1)$$

cioè $h=\Theta(log_t\, n)$.

Nota 1: in generale t non cresce con n

Nota 2: nel caso in cui $t= \Theta(1)$ si ha: $h= \Theta(log\, n)$.

Inoltre, se n=#chiavi ed N=#nodi, vale che:

$$(t-1)(N-1) \leq n \leq 2(t-1)N$$

da cui $n= \Theta(t\, N)$.

Ricerca (1)

Algoritmo di **ricerca** di k in un B-albero:

- partenza dalla radice r:
  - se r vuota, caso base: return None
- ricerca nel generico nodo, che è foglia:
  - se k è nel nodo: return puntatore e indice di k
  - altrimenti: return None
- ricerca nel generico nodo, che è interno:
  - se k è nel nodo: return puntatore e indice di k
  - altrimenti: si prosegue ricorsivamente…

Ricerca (2)

… La ricerca prosegue ricorsivamente:

- se k è minore della chiave più piccola, prosegue nel figlio più a sinistra
- se k è maggiore della chiave più grande, prosegue nel figlio più a destra
- se k è compreso tra due chiavi consecutive c_i e c_{i+1}, prosegue nel sottoalbero tra le due chiavi.

Ricerca (3)

Costo computazionale:

Intuitivamente:

- O(t) per scorrere ciascun nodo
- il numero di nodi da visitare è $O(\log_t n)$

quindi il costo totale è $O(t \log_t n)$.

Formalmente:

- $T(h)=T(h-1)+O(t)$
- $T(0)= O(t)$

la cui soluzione è $O(t\ h)= O(t \log_t n)$.

Ricerca (4)

Costo computazionale:

Possiamo fare meglio?

Si! nei nodi: ricerca binaria! Allora:

- $T(h)=T(h-1)+O(\log_2 t)$
- $T(0)= O(\log_2 t)$

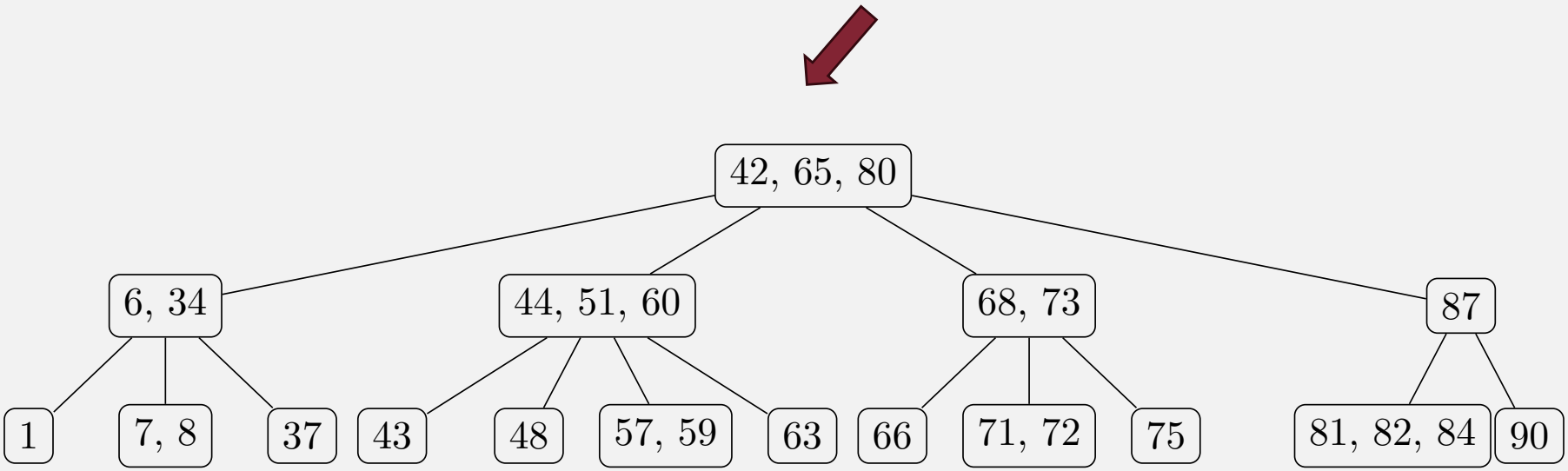
la cui soluzione è $O(\log_2 t\ h)= O(\log_2 t\ \log_t n)$.

Applicando la formula per il cambio base ($\log_t n=\log_2 n/\log_2 t$):

$T(n)=O(\log_2 t\ \log_2 n/\log_2 t)=O(\log_2 n)$.

Ricerca (5)

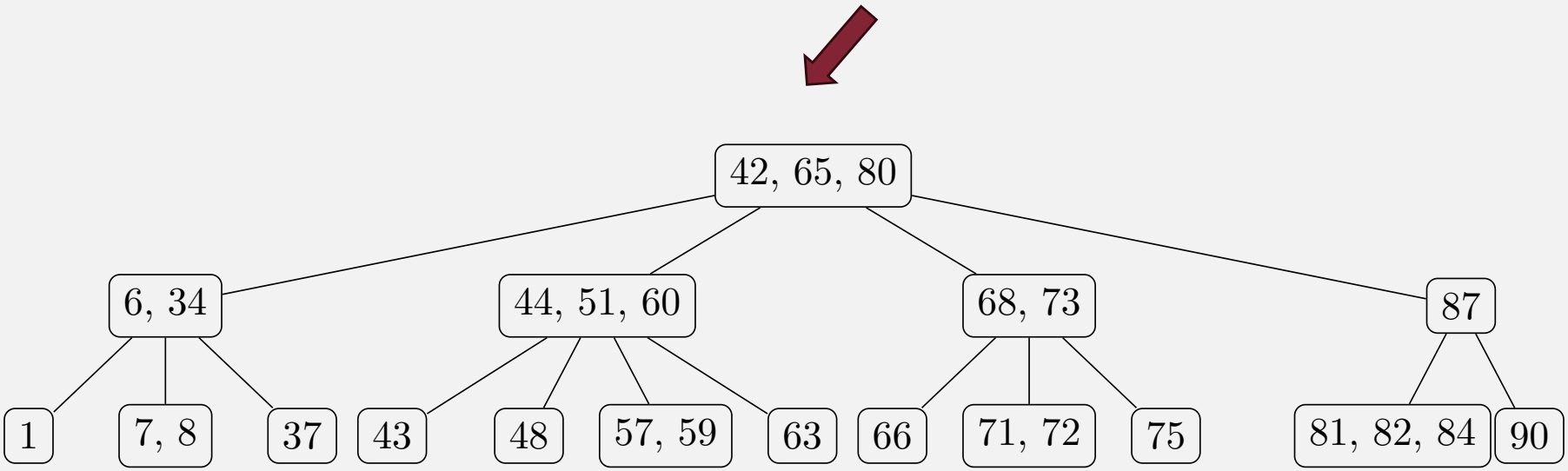
Esempio: ricerca della chiave 70



chiave 70 non trovata

Ricerca (6)

Esempio: ricerca della chiave 51



chiave 51 trovata

Inserimento (1)

Algoritmo di **inserimento** di k in un B-albero:

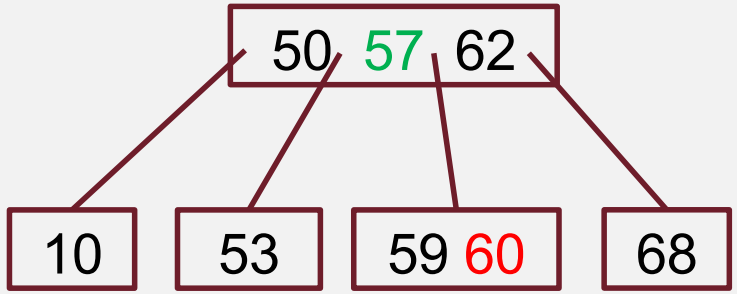
- ricerca della foglia in cui inserire k
  - si segue un algoritmo di ricerca di k fino ad arrivare ad una foglia F
  - inserimento di k in F seguendo l'ordinamento delle chiavi
  - se il numero delle chiavi è al più 2t-1 return
  - altrimenti operazione di **split…**

Inserimento (2)

… operazione di **split**:

se t=2 il massimo numero di chiavi per nodo è 2t-1=3.

Si spezza il nodo con t chiavi in due nodi con t-1 e t chiavi, rispettivamente; la chiave centrale si sposta nel padre:

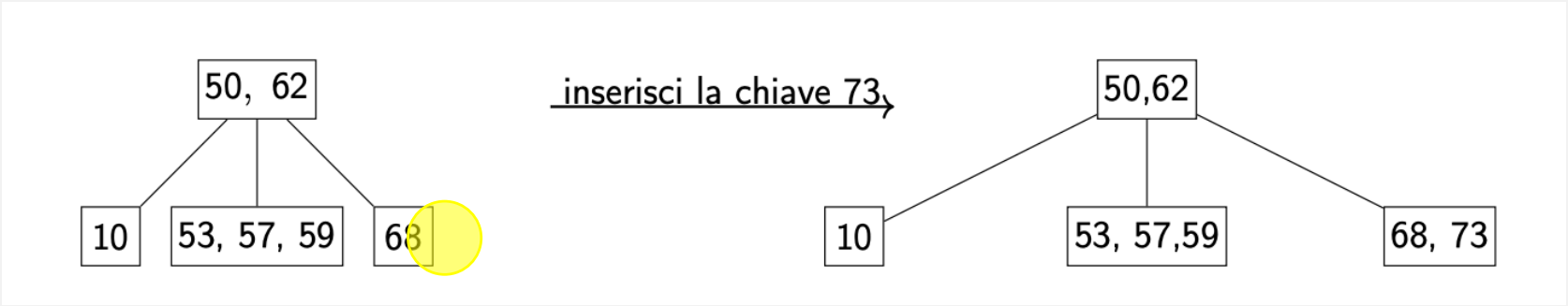


Inserimento (3)

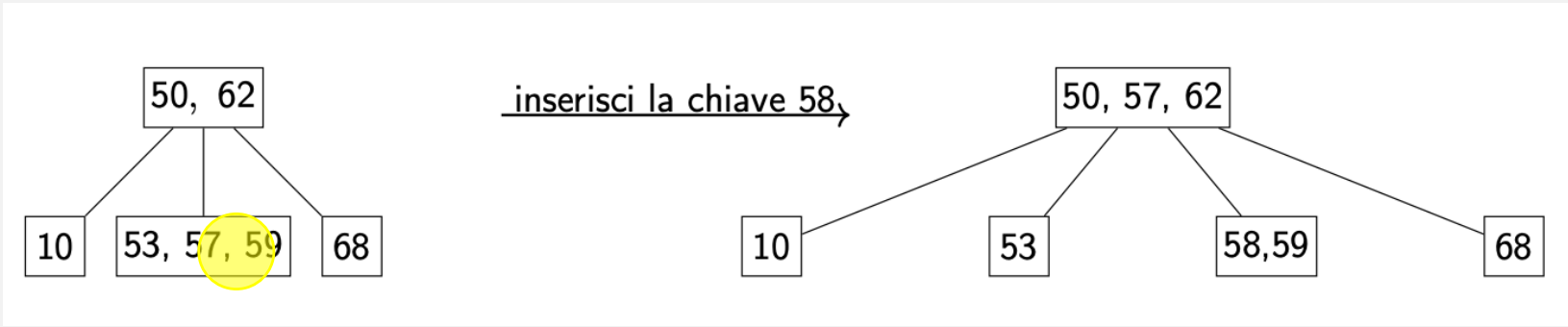
Potrebbe accadere che, a seguito dell'inserimento della chiave mediana nel padre, anche questo nodo sia troppo pieno e debba, a sua volta, subire uno split.
In questo caso, oltre alle chiavi, anche i figli vengono divisi tra due nodi, e la chiave mediana sale.
Questo processo **si propaga**, al più, fino alla radice.
Lo split della radice causa l'aumento dell'altezza.

Inserimento (4)

Esempio di **split con aumento dell'altezza**:



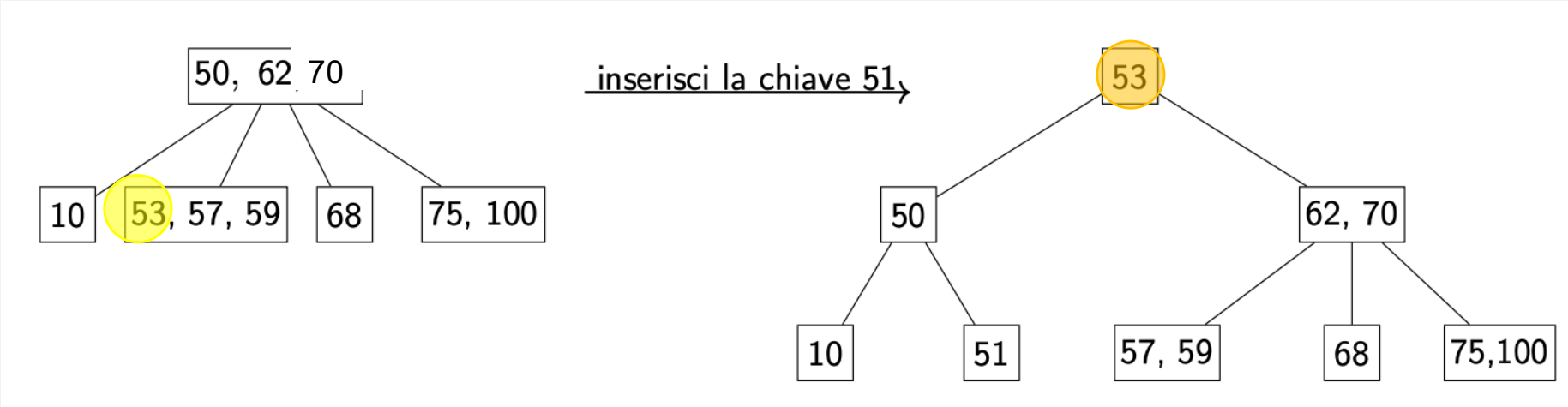
… la chiave viene inserita senza richiedere altre modifiche



… split della foglia…

Inserimento (5)

(segue esempio di **split con aumento dell'altezza**)



… split della foglia che si propaga fino alla radice…

Inserimento (6)

Costo computazionale dell'algoritmo di inserimento di k in un B-albero:

- ricerca della foglia in cui inserire k $O(t \log_t n)$
- inserimento di k nella foglia seguendo l'ordinamento delle chiavi $O(t)$
- 1 operazione di split… O(1) …ripetuta al più una volta per ogni nodo dalla foglia alla radice $O(\log_t n)$

TOT. $O(t \log_t n)$

Cancellazione (1)

Algoritmo di **cancellazione** di k in un B-albero:
ricerca della chiave k da cancellare

- si segue una ricerca di k fino ad arrivare al nodo P che la contiene;
- se P è un nodo interno:
  k viene sostituita con il suo predecessore ed il problema si riduce alla cancell. in una foglia
- se P è una foglia:
  La chiave da cancellare viene semplicemente rimossa dalla lista delle chiavi del nodo.

Cancellazione (2)

A seguito della cancellazione di k, P potrebbe non avere più il minimo delle chiavi t-1. Allora:

Spostamento chiavi:

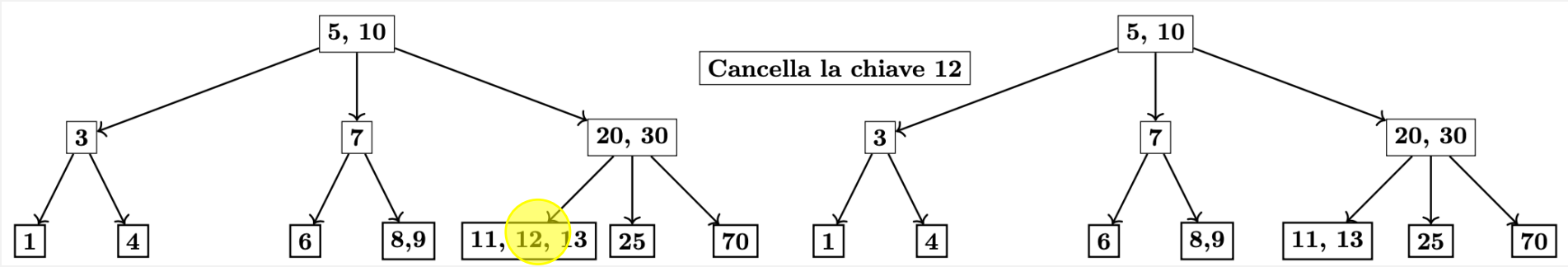
- si prende una chiave dal fratello sinistro o destro, se ne ha almeno t
- se ciò non è possibile:

Fusione:

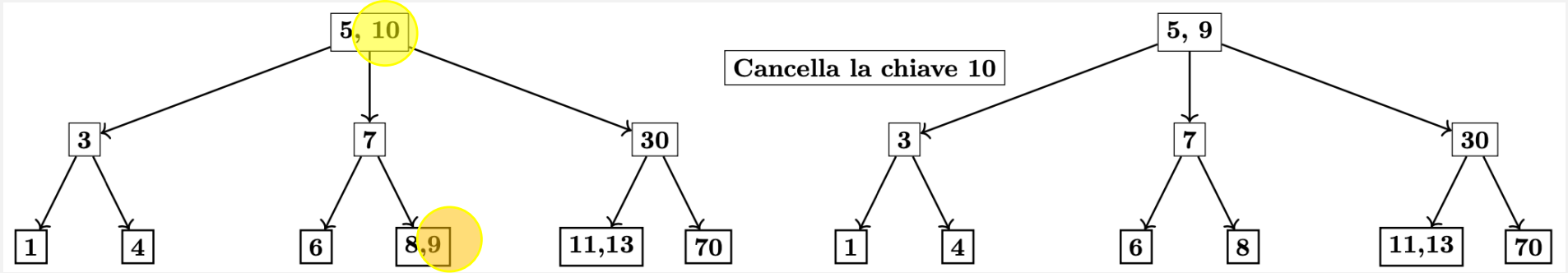
- P viene fuso con uno dei suoi fratelli, includendo una chiave del padre
- la fusione si può **propagare** verso l'alto fino a causare la diminuzione dell'altezza.

Cancellazione (3)

Esempio di **cancellazione in un nodo foglia**:

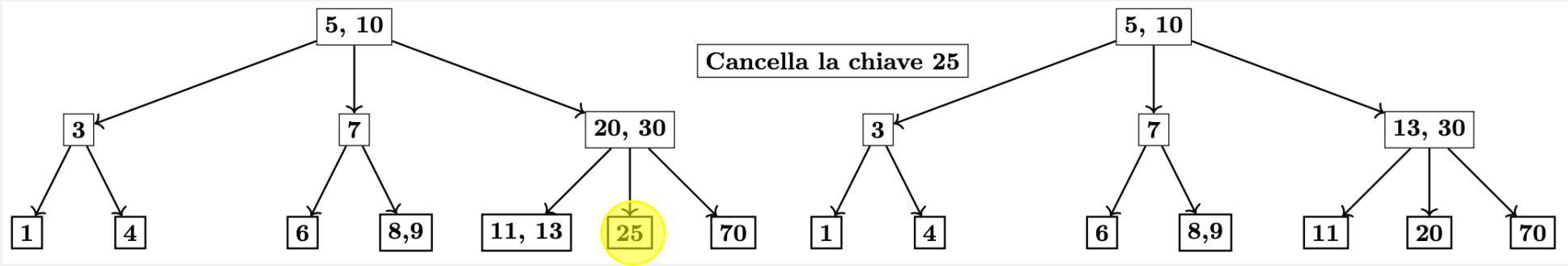


Esempio di **cancellazione in un nodo interno** (con scambio con il predecessore):

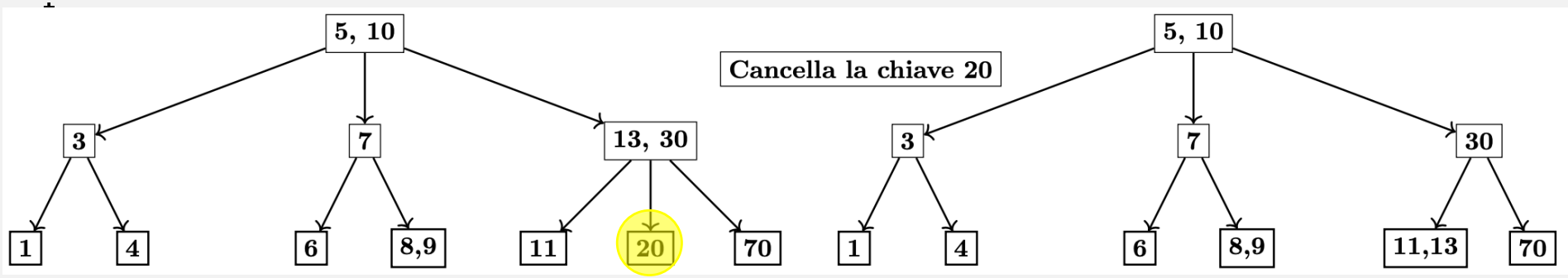


Cancellazione (4)

Esempio di **cancellazione con spostamento chiavi** dal fratello sinistro:

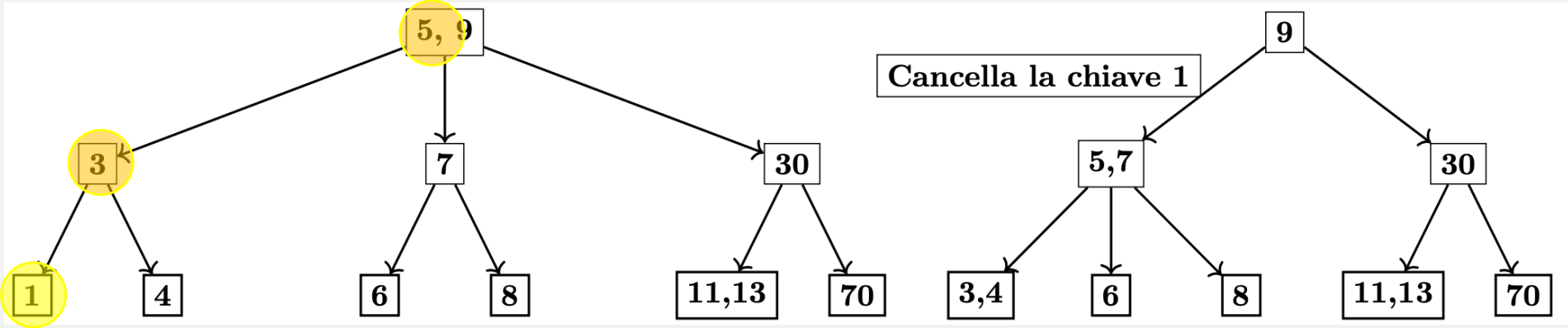


Esempio di **cancellazione con fusione**:

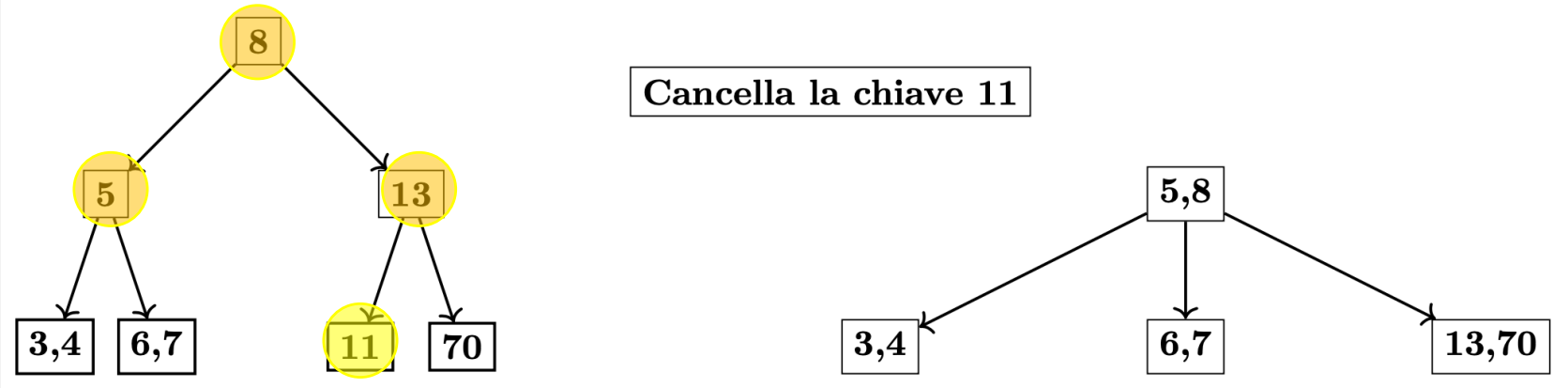


Cancellazione (5)

Esempio di **cancellazione con fusione propagata** verso l'alto:



Esempio di **cancellazione con riduz. dell'altezza**:



Cancellazione (6)

Costo computazionale dell'algoritmo di cancellazione in un B-albero:

- ricerca del nodo da cui cancellare k — $O(t \log_t n)$
- cancellazione di k — $O(t)$
- spostamento di chiavi — $O(1)$
- 1 operazione di fusione… — $O(t)$

  (ricordare che l'operazione di fusione di due sequenze ordinate si esegue in tempo lineare rispetto al numero di chiavi)
- …ripetuta al più una volta per ogni nodo dalla foglia alla radice — $O(\log_t n)$ volte

TOT. $O(t \log_t n)$

Corso di laurea in Informatica
Algoritmi 1
A.A. 2025/2026

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi (1)

- Costruire un B-albero con t=2 inserendo sequenzialmente le seguenti chiavi:

  6, 19, 7, 11, 3, 12, 8, 20, 22, 23, 13, 18, 14, 16, 1, 2, 24, 25, 4, 26, 5.

  Disegnare l'albero dopo ogni inserimento.

- Partendo dall'albero ottenuto nell'esercizio precedente:
  - Trovare la sequenza di chiavi visitate per la ricerca delle chiavi 1, 14, e 20.
  - Cancellare ordinatamente le chiavi da 1 a 8. Disegnare l'albero dopo ogni cancellazione.

Esercizi (2)

Costruire un B-albero con t=3 inserendo sequenzialmente le seguenti chiavi:

6, 19, 7, 11, 3, 12, 8, 20, 22, 23, 13, 18, 14, 16, 1, 2, 24, 25, 4, 26, 5.

Disegnare l'albero dopo ogni inserimento.

Esercizi (3)

Progettare un algoritmo che, dato in input un B-albero tramite il puntatore alla radice:

- ne calcoli il numero di chiavi;

  Il costo computazionale deve essere $O\left(\frac{n}{t}\right)$;
- ne calcoli l'altezza. Il costo computazionale deve essere $O(\log_t n)$;
- verifichi che ogni nodo dell'albero contenga un numero di chiavi compreso tra t-1 e 2t-1. Il costo computazionale deve essere *O(n)*;
- verifichi che tutti i suoi nodi foglia siano allo stesso livello. Il costo computazionale deve essere $O\left(\frac{n}{t}\right)$.