BUDDY ALLOCATOR CON BITMAP

STUDENTE: Riccardo Pagliuca 1762734

Il progetto consiste nella realizzazione di un buddy allocator utilizzando come struttura ausiliaria una bitmap.

Il concetto generale è analogo a quello di un classico buddy allocator, ma in questo caso, per memorizzare i blocchi liberi e quelli occupati, viene utilizzata unicamente una bitmap. Tale bitmap serve per rappresentare l'albero della memoria nel quale ogni bit corrisponde a un nodo e ogni livello dell'albero corrisponde a un livello del buddy allocator.

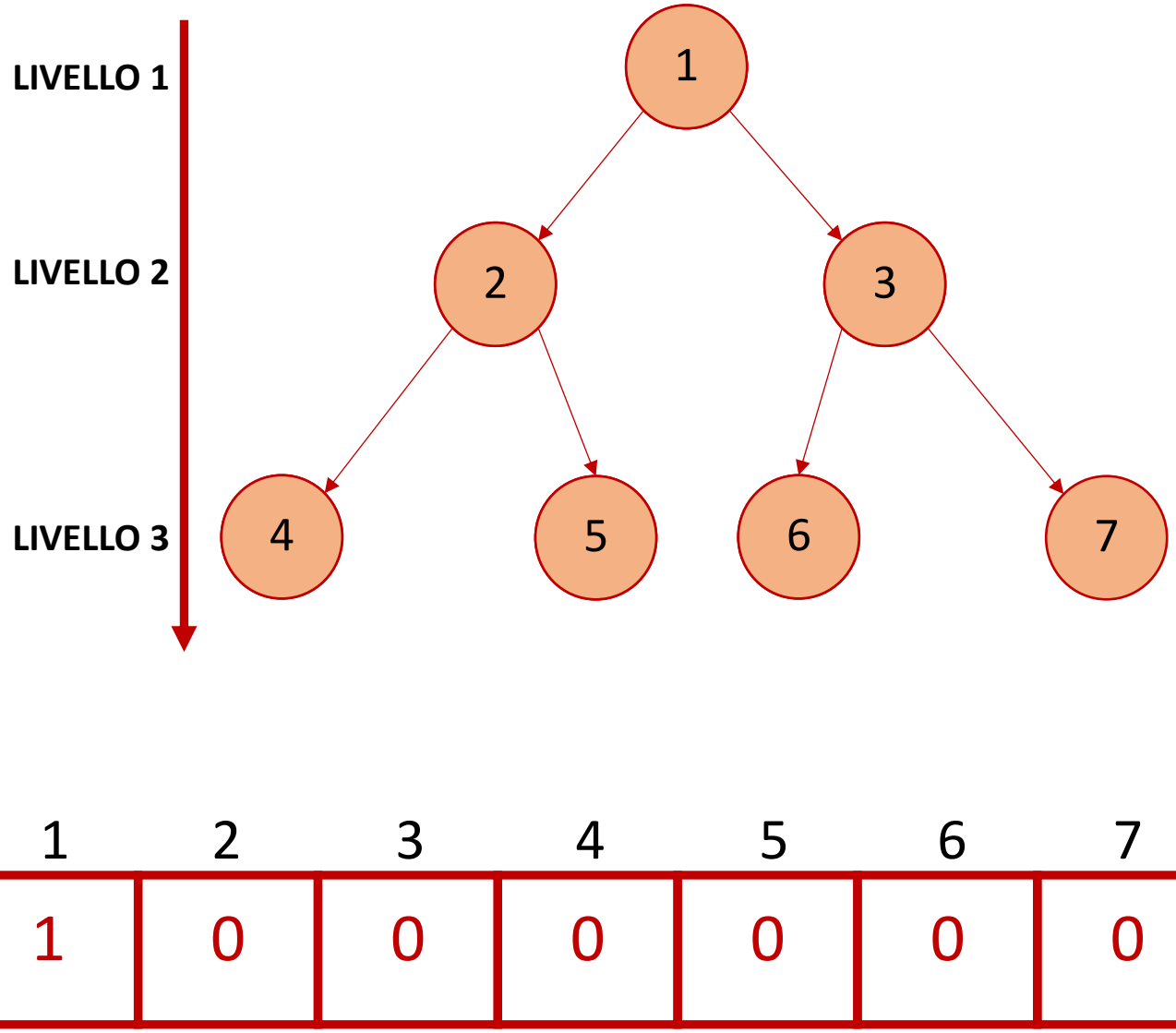


Figura 1. Bitmap rappresentante lo stato iniziale di un allocatore a 3 livelli. Inizialmente solo il primo nodo è libero perciò è l'unico con il relativo bit settato a 1

-OSS

È importante osservare che ogni nodo può assumere al massimo 2 valori:

- 1. Significa che la zona di memoria associata è disponibile
- 0. La zona di memoria non è disponibile, ma ciò potrebbe accadere per diversi motivi:
    - Il nodo corrispondente è già stato rilasciato in seguito a una richiesta precedente
    - Il nodo è stato diviso per creare due buddy a un livello inferiore
    - Il nodo genitore deve essere diviso per creare il buddy al livello richiesto

Inoltre l'albero è memorizzato all'interno della bitmap come un heap. Per questo motivo, dato l'indice di un nodo, si possono ottenere gli indici del nodo genitore, dei figli e del buddy in tempo costante.

-OPERAZIONI

Le operazioni che servono all'utente sono 2: la malloc e la free.

-MALLOC

Quando l'utente richiede memoria di una certa dimensione M, l'allocatore esegue alcuni passi:

1. Stabilisce il livello a cui si trova la memoria da allocare, arrotondando la richiesta alla dimensione del livello più grande che può contenere tale dimensione (non è possibile allocare una zona di memoria la cui dimensione sia una via di mezzo tra le dimensioni di due livelli successivi). Inoltre controlla se la memoria richiesta non sia effettivamente più grande della dimensione massima disponibile.
2. Controlla se nella bitmap, al livello trovato precedentemente, ci sia un bit diverso da 0.

   A partire dal numero del livello è immediato trovare sia il primo nodo di tale livello sia il numero di nodi a tale livello. È sufficiente applicare le semplici equazioni degli alberi binari:

   $$\text{idx 1° nodo lvl} = \#\text{ nodi lvl} = 2^{(\text{lvl} - 1)}$$

3. Se trovo un bit uguale a 1 ritorno il nodo corrispondente
4. Se invece non c'è alcun buddy disponibile a questo livello provo con il livello superiore ritornando al punto 2.
5. Se in questo modo trovo un buddy disponibile divido ogni nodo a ciascun livello finchè non torno al livello originale
6. Se invece non è presente memoria disponibile ritorno NULL.

Alla fine la funzione non tornerà il nodo, ma il puntatore all'area di memoria associata a tale modo. Però, per poter effettuare successivamente la free, è opportuno "sacrificare" i primi byte della memoria per salvarvi l'indice del nodo corrispondente.

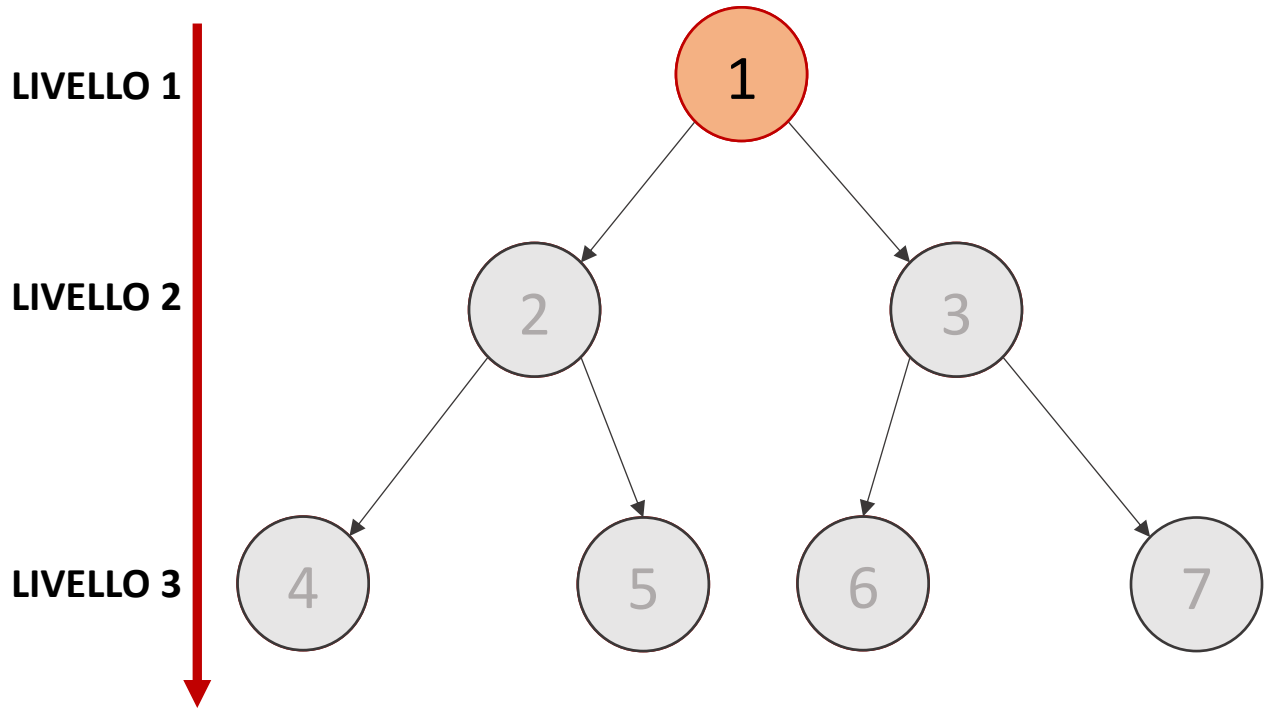
-FREE

Anche la free risulta abbastanza semplice:

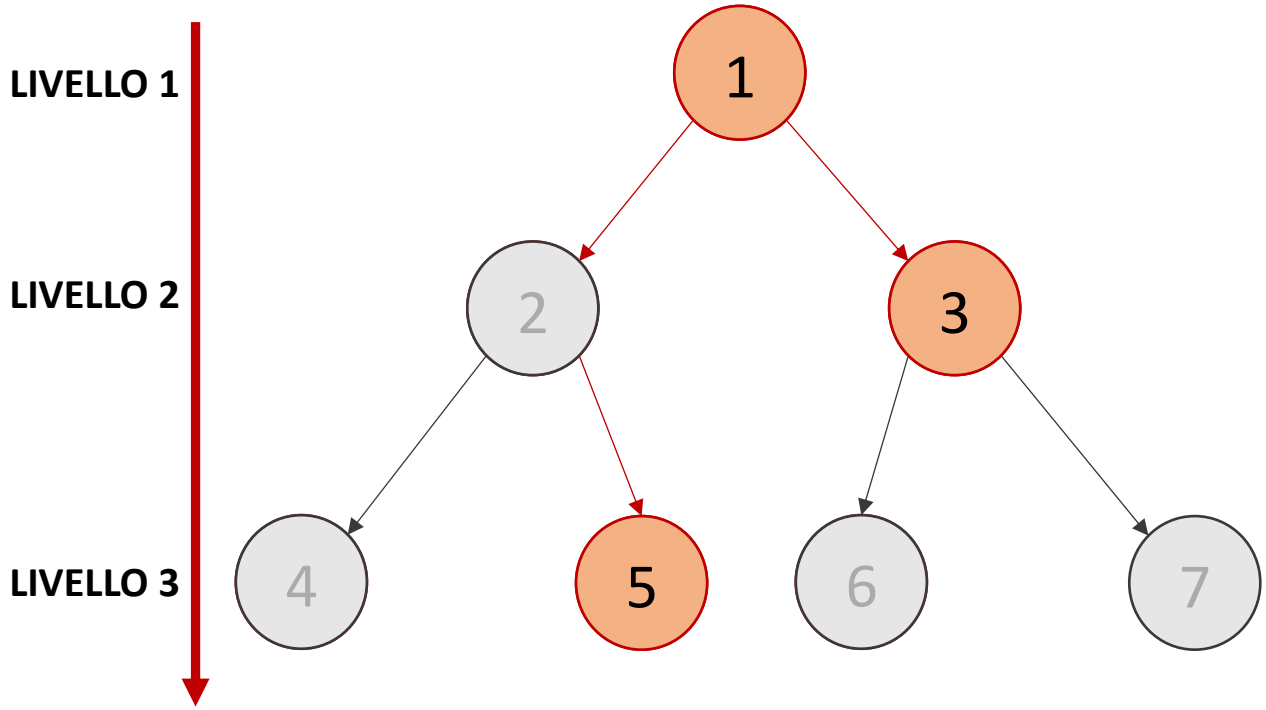
1. Per prima cosa ricavo l'indice del nodo che avevo precedentemente salvato dopo la malloc.
2. Successivamente cambio il bit corrispondente al nodo nella bit-map da 0 a 1.
3. Se anche il buddy del nodo appena rilasciato è libero (cioè è uguale a 1) allora devo unirli, liberando il nodo genitore. Ripeto il passo 3. con il nodo appena liberato finchè non arrivo alla radice o trovo un nodo il cui buddy non è libero.

-ESEMPIO DI ALLOCAZIONE

Ipotizziamo che l'allocatore sia tale che abbia al massimo 3 livelli e che l'utente richieda inizialmente una quantità di memoria più piccola della dimensione minima che si può allocare. La malloc arrotonderà la richiesta a tale dimensione e perciò cercherà un buddy libero all'ultimo livello, in questo caso il terzo.



Poichè non c'è nessun nodo disponibile cercherà prima al secondo livello e, non trovando nulla neanche qua, creerà due buddy a partire dall'unico nodo del primo livello e successivamente dividerà anche il nodo 2 al secondo livello, creando i buddy 4 e 5 che saranno diventati disponibili. Il nodo 4 sarà quello 'ritornato' all'utente.



Alla fine la bitmap sarà diventata:

1	2	3	4	5	6	7
0	0	1	1	0	0	0

-VARIANTE

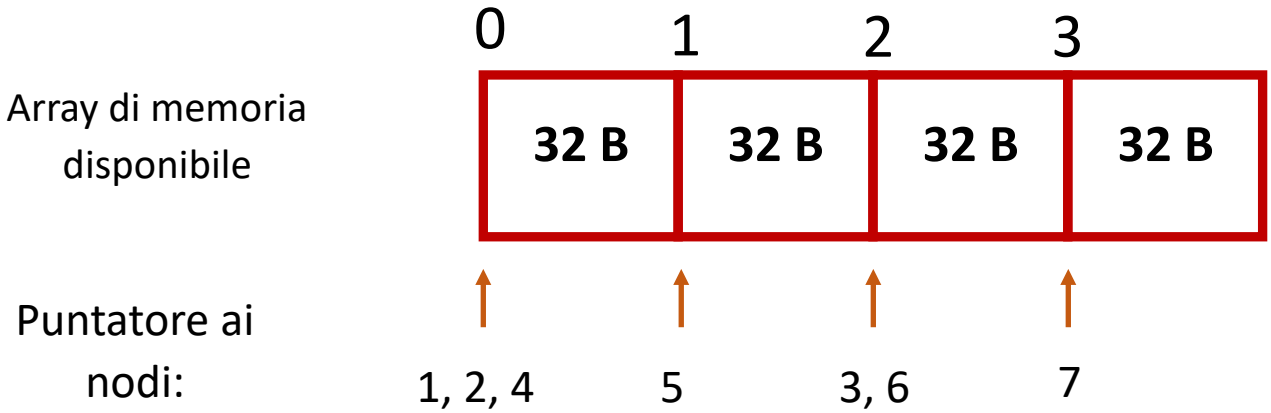
Nonostante il metodo precedente sia semplice da realizzare, non è totalmente efficiente in termini di spazio poiché 4 byte vengono sprecati in ogni zona allocata per mantenere l'indice.

Per questo motivo ho sviluppato un'altra versione che potesse risolvere questo problema. L'utilizzo dell'indice è necessario per la free: senza di esso si avrebbe un'ambiguità circa il corretto nodo da liberare. Di conseguenza è necessaria un'ulteriore informazione che permetta all'allocatore di capire su quale buddy deve eseguire la free.

Infatti, utilizzando questo allocatore, diversi buddy vengono associati allo stesso indirizzo (non contemporaneamente), di conseguenza il solo puntatore non è sufficiente per determinare il nodo da liberare.

Fortunatamente si può facilmente dimostrare che l'insieme di nodi che possono essere associati ad un indirizzo si possono ottenere moltiplicando per 2 un nodo appartenente all'insieme.

Ad esempio i nodi con indice 2, 4, 8, ecc. sono tutti associati al primo indirizzo della memoria disponibile.



L'idea è quella di utilizzare un'altra bitmap che sia grande quanto quella principale. Questa però conterrà per ogni nodo un bit con l'informazione se esso è occupato per un puntatore o meno:

- 1 => il nodo è 'puntato'
- 0 => il nodo non è 'puntato' (e quindi o è libero o non è disponibile perché è stato diviso in due o non ancora diviso)

Quando viene chiamata la free su un puntatore il processo è leggermente diverso:

1. Per prima cosa viene determinato il nodo al livello più basso che può essere associato a tale puntatore
2. Se il nodo è occupato da un puntatore (cioè se la seconda bitmap è settata a 1) allora tale nodo è quello da liberare
3. Altrimenti si passa al genitore e si torna al punto 2.

Una volta trovato il nodo da liberare si procede come nella prima versione.

Rispetto alla prima versione, la free esegue più passi ma anche in questo caso il costo rimane O(log n). Infatti nella ricerca del nodo da liberare si parte dall'ultimo livello e si sale di un livello alla volta, arrivando eventualmente al primo nel caso un cui si debbano unire tutti i buddy.

-HOW TO RUN

L'allocatore presenta in interfaccia con tre funzioni necessarie all'utente: init, malloc e free. Per utilizzarlo è sufficiente importare il modulo, inizializzare l'allocatore e infine utilizzare a seconda delle proprie necessità malloc e free.

All'interno delle repository sono presenti 4 file di prova: 2 per v1 e 2 per v2. Tali file sono compilati con un main di prova (oltre ai moduli necessari):

1. ./my_test
   Contiene una sequenza di prova creata da me

2. ./buddy_test

Contiene il main di prova fornito agli studenti per testare il buddy_allocator nella repository di sistemi operativi.