Grundlagen Algorithmen und Datenstrukturen

Kapitel 13

Christian Scheideler + Helmut Seidl

SS 2009

Speicherverwaltung

Drei Ansätze:

- Allokiere neue Objekte auf einem Keller.
  Gib nie Speicherplatz wieder frei.
- Versuche, nicht mehr benötigten Platz wieder zu verwenden (C / C++)
- Automatische Speicherbereinigung (Java)

Speicherverwaltung

Drei Ansätze:

- Allokiere neue Objekte auf einem Keller.
  Gib nie Speicherplatz wieder frei.
- Versuche, nicht mehr benötigten Platz wieder zu verwenden (C / C++)
- Automatische Speicherbereinigung (Java)

Übersicht

- DS zur Speicherallokation
- Speicherallokation mit Verschiebung
- DS zur Bandbreitenallokation

Das Buddy System

Problem: Verwaltung freier Blöcke in einem gegebenen Adressraum {0,…,m-1} zur effizienten Allokation und Deallokation.

Vereinfachung:

- m ist eine Zweierpotenz
- nur Zweierpotenzen für allokierte Blockgrößen erlaubt

Das Buddy System

$M \subseteq \{0,\ldots,m-1\}$: freier Adressraum

Operationen:

- Allocate(i): allokiert Block der Größe 2^i, d.h.

$M = M \setminus B$; für einen Block $B \subseteq M$ der Größe 2^i

- Deallocate(B): gibt Block B wieder frei, d.h.

$M = M \cup B$;

Die Buddy Datenstruktur

Physikalischer Adressraum:

- F: Feld von log m +1 Blocklisten
  F[i], i>0, speichert Anfangsadressen freier Blöcke der Größe 2^i
- Verwende das erste Byte von jedem allokierten Block der Größe 2^i zur Speicherung von i+1 (das reicht, da damit Blockgrößen bis zu $2^{2^8-2} = 2^{254}$ möglich).
  Freie Blöcke haben dieses Byte auf 0 gesetzt.

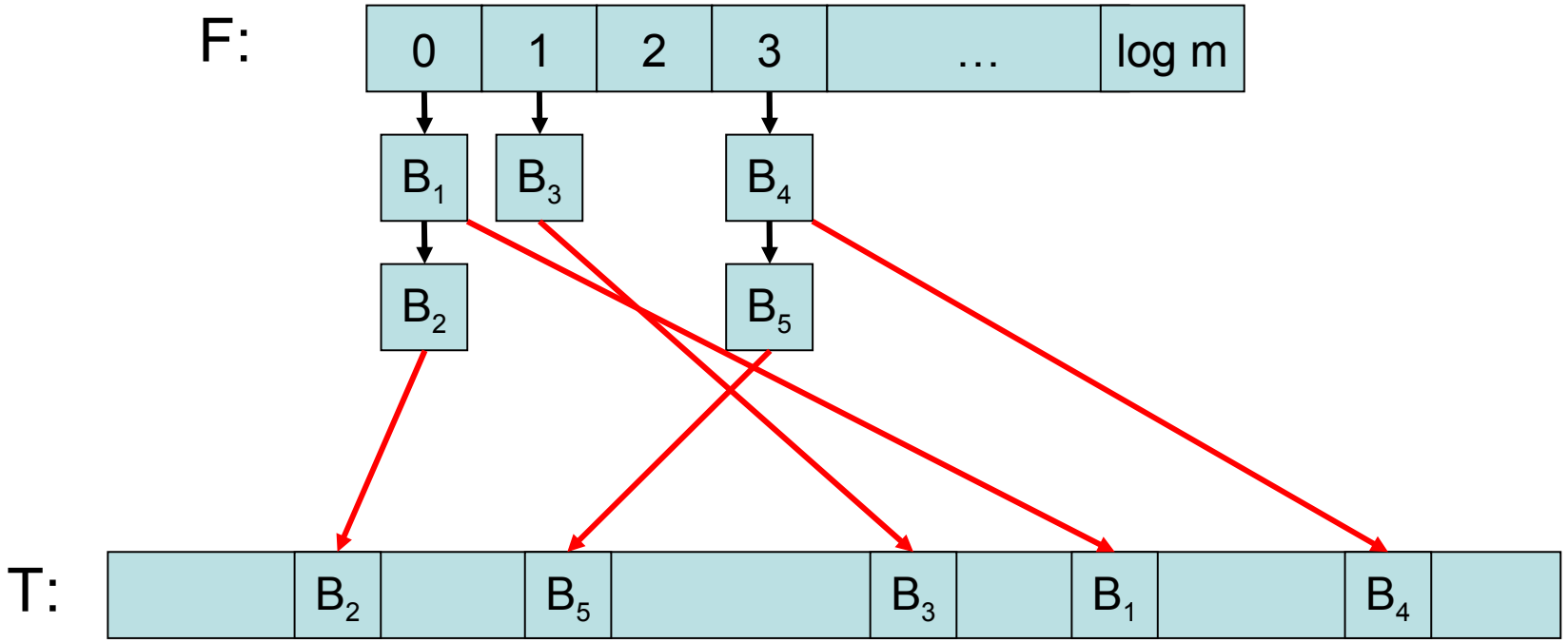
Die Buddy Datenstruktur

Virtueller Adressraum:

- F: Feld von log m +1 Blocklisten F[0],…,F[log m]
- T: Hashtabelle mit Einträgen zu freien Blöcken. Jeder Eintrag enthält:
  - Startadresse addr(B) des Blocks B
  - Größe von Block B ($|B|=2^i$: speichere i)

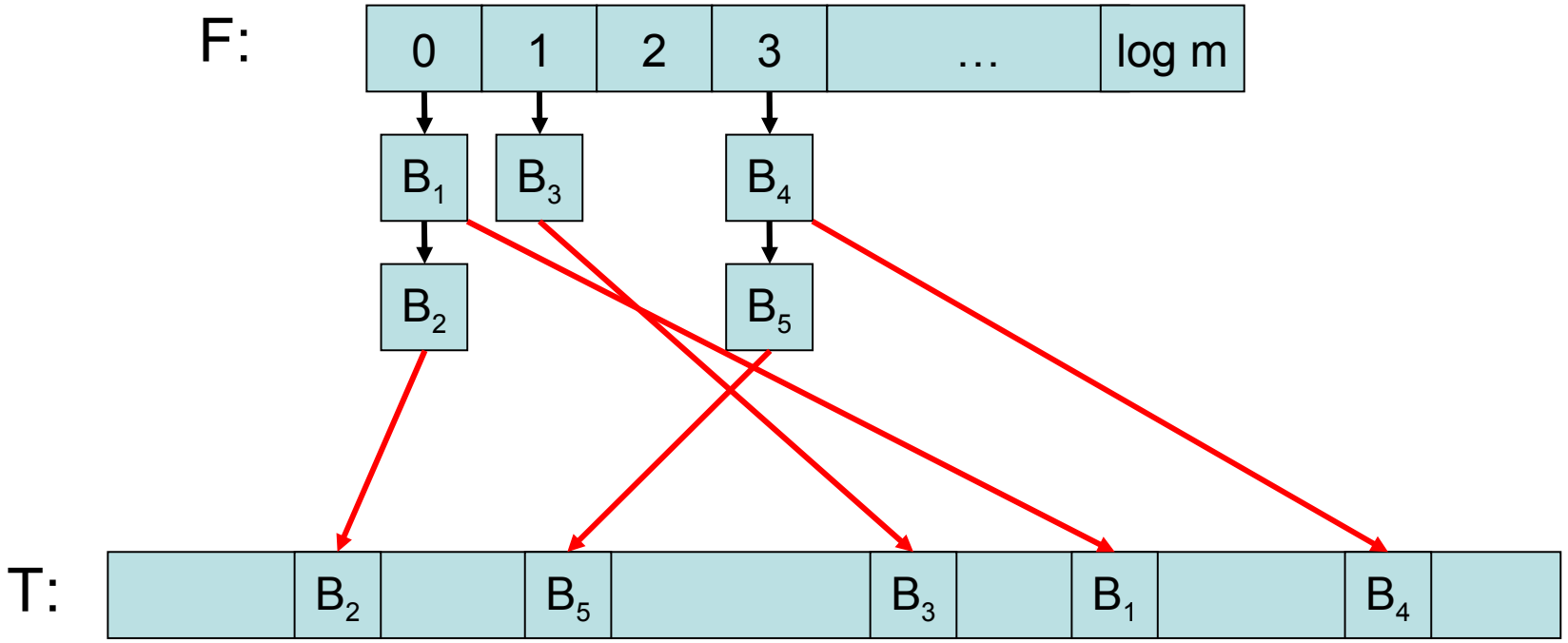
Das Buddy System

F[i]: Liste von Blockgrößen 2^i



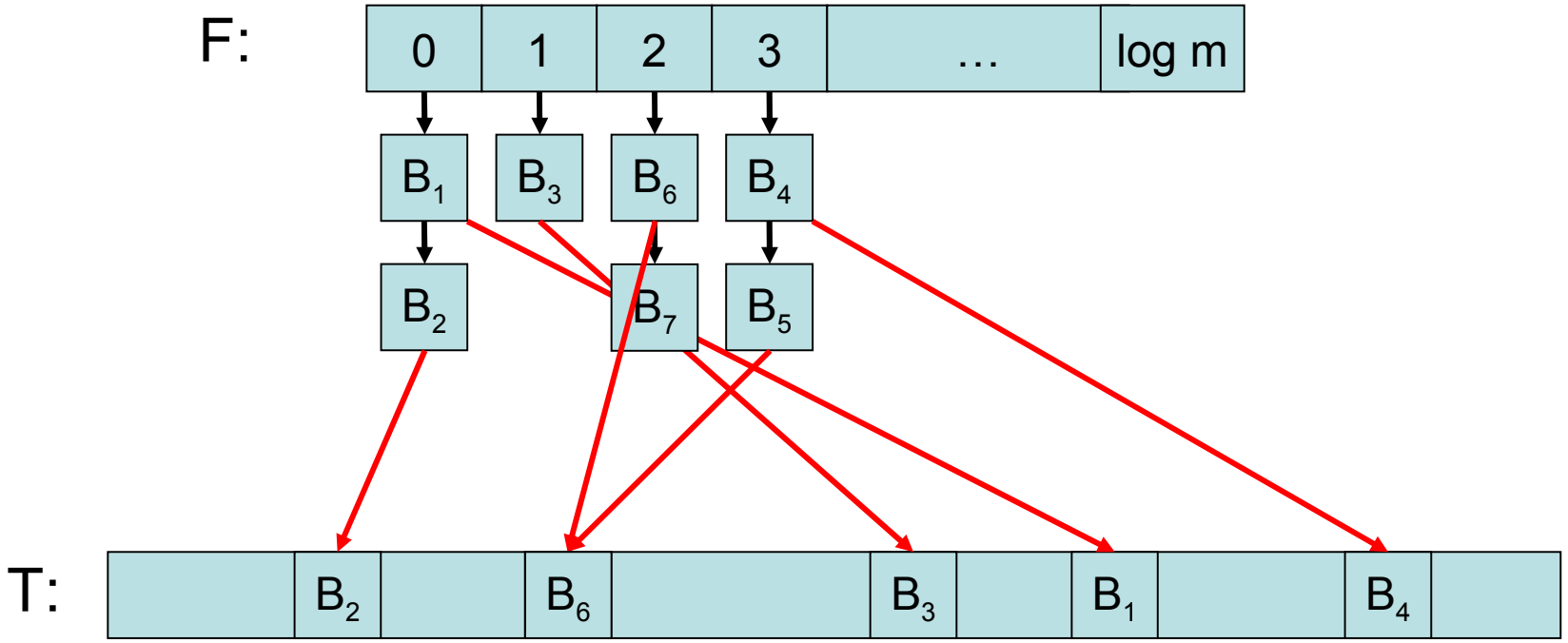
Das Buddy System

Allocate(3):



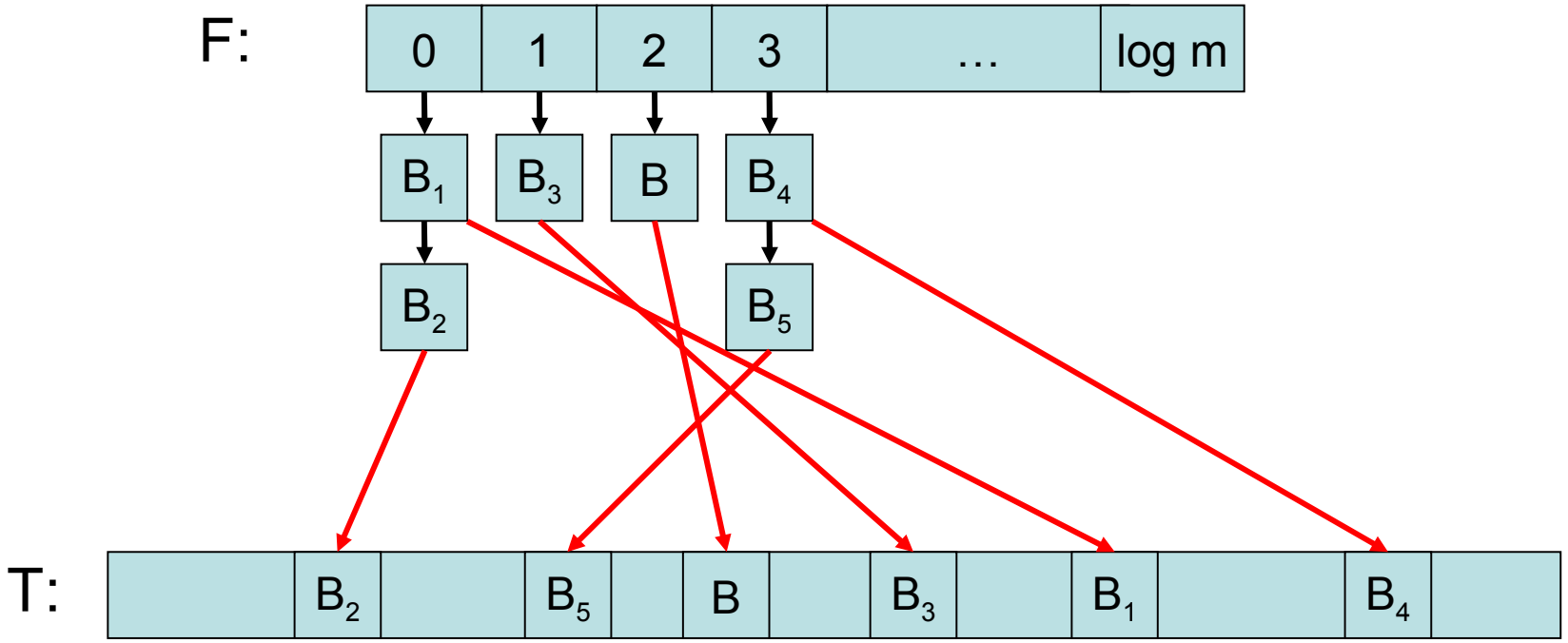
Das Buddy System

Allocate(2):



Das Buddy System

Deallocate(B): (Größe von B ist 2^2)



Das Buddy System

Problem: zunehmende Fragmentierung

Definition 6.3:

- Block B der Größe 2^i ist gültig: Startadresse von B ist 0 für die ersten i Bits
- Buddy von Block B der Größe 2^i: Block B', für den $B \cup B'$ einen gültigen Block der Größe 2^{i+1} ergibt

Invariante: Für jeden freien Block B in Feld F ist der Buddy belegt.

Das Buddy System

Bewahrung der Invariante bei Deallocate(B):

```
while (Buddy(B) frei)
    B = B ∪ Buddy(B);
speichere B in F und T ab;
```

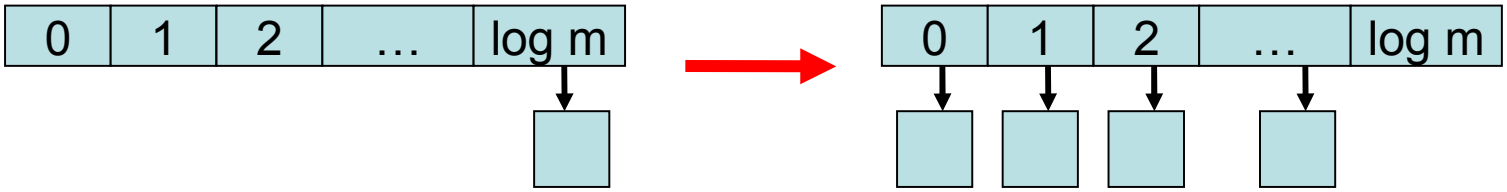
Schnelle Bestimmung von Buddy(B):

- berechne Startadresse von B' = Buddy(B) (folgt direkt aus Startadresse von B)
- prüfe mittels T, ob B' existiert (beim physikalischem Adressraum reicht es stattdessen, direkt auf das erste Byte von B' zuzugreifen)

Das Buddy System

Probleme:

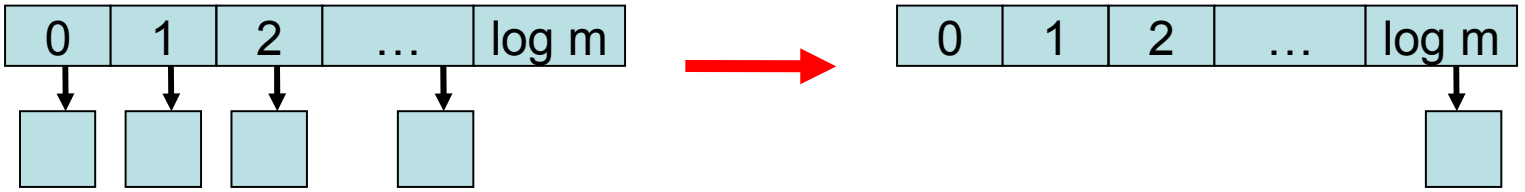
1. trotz Buddy-Ansatz keine garantiert niedrigen Obergrenzen für Fragmentierung
2. Allocate und Deallocate können Θ (log m) viele Schritte laufen (wegen split- oder merge-Operationen)
   Beispiel: Allocate(0)



Das Buddy System

Probleme:

1. trotz Buddy-Ansatz keine garantiert niedrigen Obergrenzen für Fragmentierung
2. Allocate und Deallocate können Θ (log m) viele Schritte laufen (wegen split- oder merge-Operationen)
   Beispiel: Deallocate(0)



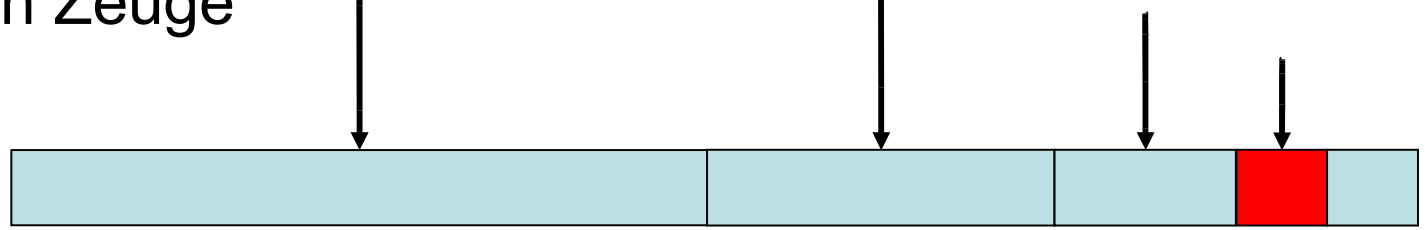
Das Buddy System

1. Problem: Fragmentierung

Theorem 6.4: Die Anzahl freier Blöcke ist höchstens $O(\log m \cdot$ Anzahl allokierter Blöcke$)$.

Beweis:

- Freier Block nicht kombinierbar: allokierter Block dafür ein Zeuge



- Allokierter Block wird von höchstens $\log m$ freien Blöcken als Zeuge verwendet.

Verbessertes Buddy System

2. Problem: Allocate und Deallocate brauchen Zeit $\Theta(\log m)$: kann effizient gelöst werden.

Idee: erlaube Blöcke, die freien Speicher der Form $2^k - 2^i$ ($i<k$) angeben.

Solche Blöcke werden in F[k-1] gespeichert.

Verbessertes Buddy System

Allocate(i): führe lazy splitting durch.

- Fall 1: 2^k-Block B vorhanden, k>i.
  Schneide aus B vorderen 2^i-Block raus, Rest von B wird 2^k-2^i-Block.
- Fall 2: nur 2^k-2^j-Block B vorhanden, k>i.
  Schneide ersten gültigen 2^i-Block aus B raus.
  Damit teilt sich B in 2^i-2^j-Block und 2^k-2^{i+1}-Block auf.

Laufzeit: O(1), falls Suche nach 2^k-Block bzw. 2^k-2^j-Block in O(1) Zeit machbar.

Verbessertes Buddy System

O(1) Suchzeit nach passendem Block:

- Strategie 1: Falls Worte der Größe $\log m + 1$ verfügbar, die mit Einheitskosten bearbeitet werden können, dann setze Bit i von Indexwort W auf 1 genau dann, wenn $F[i]$ ein Element enthält. Suche dann nach dem erstem gesetzten Bit $k>i$ bzw. $k>i$ (was mit x86-Prozessoren in $O(1)$ Zeit machbar ist).
- Strategie 2: Tabelle der Größe m^{ε}, die für jede Zahl $z \in \{0,\ldots,m^{\varepsilon}-1\}$ das niedrigste gesetzte Bit in z enthält. Damit maximal $1/\varepsilon$ Zeit, bis passen-der Index $k>i$ bzw. $k>i$ gefunden.

Verbessertes Buddy System

Deallocate(B): führe lazy merging durch.

- Wiederhole, bis kein Fall mehr eintritt:
  - Fall 1: B hat freien Buddy B'=Buddy(B) in F:
    $B = B \cup B'$;
  - Fall 2: B gehört zu freiem 2^k-2^i-Block B' in F, Größe von B ist 2^i:
    $B = B \cup B'$;
- Speichere B in F (und T) ab

Laufzeit: amortisiert O(1).

Verbessertes Buddy System

Lemma: Die amortisierte Laufzeit der Deallocate-Operation ist O(1).

Beweis:

- Ein Merge wird nur dann auf B und B' angewandt, wenn $B \cup B'$ vorher in einem Allocate in B und B' geteilt worden ist.
- Zahle für jeden Block, der Ergebnis eines Splittings ist, auf ein Konto. Verrechne mit der Einzahlung die Kosten des Mergens.

Übersicht

- Union-Find Datenstruktur
- DS zur Speicherallokation
- Allokation mit Verschiebung
- DS zur Bandbreitenallokation

Buddies mit Reallokation

Situation hier:

- Speicherreallokationen erlaubt
- Allocate(i) hat ohne Reallokation 2^i Zeit-aufwand (Speicher wird überschrieben)
- Deallocate(i) hat keinen extra Zeitaufwand (Speicher wird lediglich freigegeben)

Buddies mit Reallokation

Allocate(i):

- Annahme: zusammen mit neuem 2^i-Block sind $(1-\varepsilon)m$ der Speicherzellen belegt.
- Suche gültigen 2^i-Block B mit Belegung $<(1-\varepsilon)2^i$ (muss existieren!)
- Für alle Blöcke B' in B: weise B' Allocate(log|B'|) zu
- Gib B zurück

Deallocate(B): wie im original Buddy-System

Buddies mit Reallokation

Theorem 6.5: Allocate(i) hat einen Zeitaufwand von höchstens $O(2^i/\varepsilon)$.

Beweis:

- Allokation von Block B: Aufwand $O(2^i)$
- Reallokation der belegten Blöcke in B: Aufwand $O((1-\varepsilon)2^i)$
- Reallokation der dadurch verdrängten Blöcke $O((1-\varepsilon)^2 2^i)$
- usw.
- Aufwand insgesamt max. $O(2^i \sum_{j>0} (1-\varepsilon))^j = O(2^i/\varepsilon)$

Übersicht

- DS zur Speicherallokation
- Allokation mit Verschiebung
- DS zur Bandbreitenallokation

Bandbreitenallokation

Problem: Verwaltung freier Frequenzen in einem gegebenen Frequenzraum {0,…,m-1} zur effizienten Allokation und Deallokation.

Vereinfachung (wie bei Buddy-System):

- m ist eine Zweierpotenz
- nur Zweierpotenzen für allokierte Blockgrößen erlaubt

Bandbreitenallokation

Situation hier: Reallokation von zugewiesenen Bandbreiten ist einfach machbar (O(1) Zeit pro Block)

Problem beim Buddy System: Reallokation teuer bzw. nicht möglich, da Speicherbereiche umkopiert werden müssen.

Bandbreitenallokation

$M \subseteq \{0,\ldots,m-1\}$: freier Frequenzraum

Operationen:

- Allocate(i): allokiert Block der Größe 2^i, d.h.

$M = M \setminus B$ für einen Block $B \subseteq M$ der Größe 2^i

- Deallocate(B): gibt Block B wieder frei, d.h.

$M = M \cup B$;

Bandbreitenallokation

Naiver Ansatz: Halte die belegten Blöcke sortiert nach Größe.

Das kann sehr viel Umorganisation verursachen.

Eine Strategie mit maximal 5 Umplatzierungen pro Allocate und Deallocate ist in “A Constant-Competitive Algorithm for Online OVSF Code Assignment” von F.Y.L. Chin, H.F. Ting und Y. Zhang zu finden.

Nächstes Kapitel

Klausur!

Bandbreitenallokation

Datenstruktur: Binärbaum mit freien ( ) und belegten Blättern ( )