9. Tutorium - Algorithmen I

Nina Zimbel

17.06.2015

Heute:

- Anmerkungen zum letzten Übungsblatt
- Dynamisiertes Adjazenzarray
- Besprechung Übungsklausur

Anmerkungen zum letzten Übungsblatt

- Augabenstellung lesen! Bei Aufgabe 2 waren Trennstriche zwischen den Buckets gefordert. Hat fast niemand gemacht...
- Aufgabe 3 *kCore*: Mit welcher Datenstruktur werden Kanten bzw benachbarte Knoten gespeichert? Muss komplette Liste von Kanten durchsucht werden um inzidente Kanten zu einem Knoten zu finden?

Anmerkungen zum letzten Übungsblatt

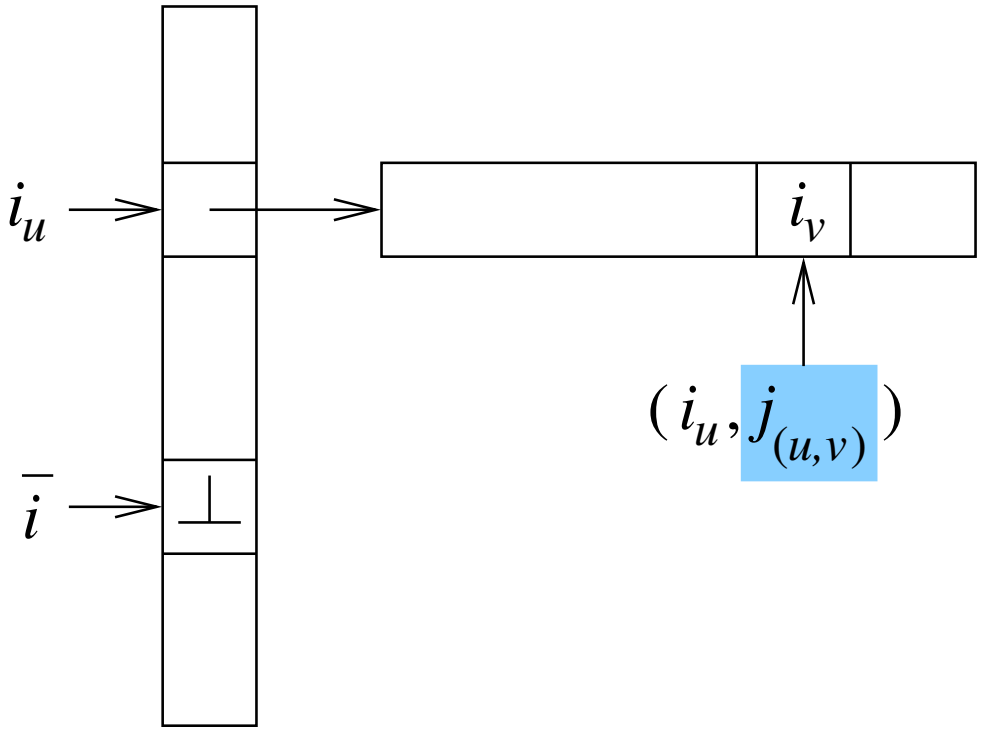
- Augabenstellung lesen! Bei Aufgabe 2 waren Trennstriche zwischen den Buckets gefordert. Hat fast niemand gemacht...
- Aufgabe 3 *kCore*: Mit welcher Datenstruktur werden Kanten bzw benachbarte Knoten gespeichert? Muss komplette Liste von Kanten durchsucht werden um inzidente Kanten zu einem Knoten zu finden?
- $\rightarrow$ es gibt folgende Graphrepräsentationen: Adjazenzlisten, Adjazenzfeld, Adjazenzmatrix, Kantenfolgenrepräsentation

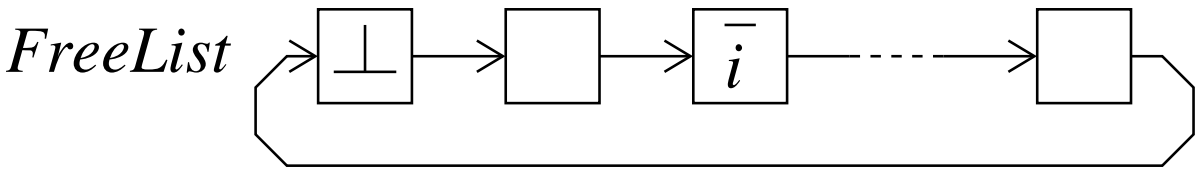
Dynamisiertes Adjazenzarray

Gesucht ist eine Datenstruktur für gerichtete Graphen G = (V, E) mit folgenden Eigenschaften:

- *Stabile und eindeutige Knoten-IDs*
- *Eindeutige Kanten-IDs*
- *Effizienter wahlfreier Zugriff auf Knoten und Kanten*
- *Effiziente Navigation*
- *Amortisiert konstantes Einfügen von Knoten und Kanten*
- *Amortisiert konstantes Entfernen von Knoten und Kanten*

a. Überlegen Sie sich, wie Sie diese Datenstruktur realisieren.


i_u
i_v
$(i_u, j_{(u,v)})$
$\bar{i}$
$\perp$


FreeList
$\perp$
$\bar{i}$

b. Begründen Sie, warum die beschriebenen Operationen in Ihrer Realisierung das geforderte Laufzeitverhalten aufweisen.

c. Wieviel Speicher kann ein Graph mit Ihrer Realisierung im schlimmsten Fall belegen (abhangig von aktuellen oder zwischenzeitlichen Werten von |V| und |E| und das nicht nur im O-Kalkül)?
Wieviel im besten Fall?
Vergleichen Sie mit dem Speicherverbrauch des statischen Adjazenzfeldes aus der Vorlesung.

Aufgabe 1 - Algorithm Engineering

Nennen Sie zwei Konzepte, die Algorithm Engineering im Gegensatz zu theoretischer Algorithmik ausmachen.

Aufgabe 1 - Algorithm Engineering

Nennen Sie zwei Konzepte, die Algorithm Engineering im Gegensatz zu theoretischer Algorithmik ausmachen.

- Implementierung
- Experimente
- reale Eingaben
- realistische Maschinenmodelle
- konstante Faktoren berücksichtigen

Aufgabe 2 - Listen

Nennen Sie zwei Operationen, die doppelt verkettete Listen in konstanter Worst-Case-Zeit unterstützen, einfach verkettete Listen jedoch nicht.

Nennen Sie zwei Nachteile von verketteten Listen gegenüber beschränkten Feldern.

Aufgabe 2 - Listen

Nennen Sie zwei Operationen, die doppelt verkettete Listen in konstanter Worst-Case-Zeit unterstützen, einfach verkettete Listen jedoch nicht.

- insert, remove, popBack

Nennen Sie zwei Nachteile von verketteten Listen gegenüber beschränkten Feldern.

Aufgabe 2 - Listen

Nennen Sie zwei Operationen, die doppelt verkettete Listen in konstanter Worst-Case-Zeit unterstützen, einfach verkettete Listen jedoch nicht.

- insert, remove, popBack

Nennen Sie zwei Nachteile von verketteten Listen gegenüber beschränkten Feldern.

- kein beliebiger Elementzugriff in konstanter Zeit
- Speicheraufwand für Zeiger
- kein cache-effizientes Iterieren

Aufgabe 3 - Hashing mit Kollisionen

Nennen Sie einen Vorteil und einen Nachteil von Hashing mit verketteten Listen gegenüber Hashing mit linearer Suche.

Vorteile

Nachteile

Aufgabe 3 - Hashing mit Kollisionen

Nennen Sie einen Vorteil und einen Nachteil von Hashing mit verketteten Listen gegenüber Hashing mit linearer Suche.

Vorteile

- insert in O(1)
- für dichte Hashtabellen besser
- referentielle Integrität
- Leistungsgarantien mit universellem Hashing

Nachteile

Aufgabe 3 - Hashing mit Kollisionen

Nennen Sie einen Vorteil und einen Nachteil von Hashing mit verketteten Listen gegenüber Hashing mit linearer Suche.

Vorteile

- insert in O(1)
- für dichte Hashtabellen besser
- referentielle Integrität
- Leistungsgarantien mit universellem Hashing

Nachteile

- nicht so platz-effizient
- nicht so cache-effizient
- nicht so einfach

Aufgabe 4 - Dynamische Arrays

Welche der folgenden Operationen eines unbeschränkten/dynamischen Arrays haben im schlimmsten Fall (ohne Amortisierung) $\Theta(n)$ Zeitkomplexität?
pushBack, Folge von n pushBack, Elementzugriff, Abfrage der Größe

Geben Sie die amortisierte Laufzeit der Operation pushBack an und zeigen Sie dies per amortisierter Analyse mit der Bankkontomethode. Nehmen Sie an, dass die Kapazität des Arrays verdoppelt wird, sobald sie nicht mehr ausreichend ist, und dass nur pushBack- Operationen ausgeführt werden.

Aufgabe 4 - Dynamische Arrays

Welche der folgenden Operationen eines unbeschränkten/dynamischen Arrays haben im schlimmsten Fall (ohne Amortisierung) $\Theta(n)$ Zeitkomplexität?
pushBack, Folge von n pushBack, Elementzugriff, Abfrage der Größe

- pushBack, Folge von n pushBack (wenn Arraygröße in O(n))

Geben Sie die amortisierte Laufzeit der Operation pushBack an und zeigen Sie dies per amortisierter Analyse mit der Bankkontomethode. Nehmen Sie an, dass die Kapazität des Arrays verdoppelt wird, sobald sie nicht mehr ausreichend ist, und dass nur pushBack- Operationen ausgeführt werden.

Aufgabe 4 - Dynamische Arrays

Welche der folgenden Operationen eines unbeschränkten/dynamischen Arrays haben im schlimmsten Fall (ohne Amortisierung) $\Theta(n)$ Zeitkomplexität?
pushBack, Folge von n pushBack, Elementzugriff, Abfrage der Größe

- pushBack, Folge von n pushBack (wenn Arraygröße in O(n))

Geben Sie die amortisierte Laufzeit der Operation pushBack an und zeigen Sie dies per amortisierter Analyse mit der Bankkontomethode. Nehmen Sie an, dass die Kapazität des Arrays verdoppelt wird, sobald sie nicht mehr ausreichend ist, und dass nur pushBack- Operationen ausgeführt werden.

- Die Operation pushBack hat amortisiert konstante Laufzeit ($O(1)$).
- Beweis mit Bankkontomethode (siehe Tafel)

Aufgabe 5 - Sortieren

Nennen Sie einen Vorteil von Radixsort gegenüber Mergesort (bei geeigneten Schlüsseln) und einen Vorteil von Mergesort gegenüber Quicksort.

Warum wird Quicksort in der Praxis trotzdem sehr häufig eingesetzt?

Aufgabe 5 - Sortieren

Nennen Sie einen Vorteil von Radixsort gegenüber Mergesort (bei geeigneten Schlüsseln) und einen Vorteil von Mergesort gegenüber Quicksort.

- *Vorteil Radixsort gegenüber Mergesort:* Zeit in $O(n)$ statt $\Theta(n \log n)$ bei ganzzahligen Schlüsseln
- *Vorteile Mergesort gegenüber Quicksort:* stabil, $O(nlogn)$ bzw. $nlogn + O(n)$ Vergleiche im worst-case

Warum wird Quicksort in der Praxis trotzdem sehr häufig eingesetzt?

Aufgabe 5 - Sortieren

Nennen Sie einen Vorteil von Radixsort gegenüber Mergesort (bei geeigneten Schlüsseln) und einen Vorteil von Mergesort gegenüber Quicksort.

- *Vorteil Radixsort gegenüber Mergesort:* Zeit in $O(n)$ statt $\Theta(n \log n)$ bei ganzzahligen Schlüsseln
- *Vorteile Mergesort gegenüber Quicksort:* stabil, $O(nlogn)$ bzw. $nlogn + O(n)$ Vergleiche im worst-case

Warum wird Quicksort in der Praxis trotzdem sehr häufig eingesetzt?

- average-case-Laufzeit n log n bei Randomisierung
- praktisch meist der schnellste Algorithmus
- inplace

Aufgabe 6 - Lösen von Rekurrenzen

Bestimmen Sie die Lösungen der folgenden Rekurrenzen im Θ-Kalkül mit der einfachen Form des Master-Theorems:

$T(1) = 1,\ T(n) = 16\,T(n/4) + 4n,\ n = 4^k,\ k \in \mathbb{N}$

$S(1) = 55,\ S(n) = n + 3S(n/3),\ n = 3^k,\ k \in \mathbb{N}$

Aufgabe 7 - O-Kalkül

a. Zeigen Sie: $\log(n!) \in O(n \log n)$.

b. Zeigen Sie: $\log(n!) \in O(n \log n)$.

Aufgabe 8 - Entwurf einer Datenstruktur

Aufgabe 8. (*Entwurf einer Datenstruktur*) [8 Punkte]

Eine Datenstruktur D soll Paare der Form $(Schluessel, Wert)$ speichern (sowohl *Schluessel* und *Wert* seien Zahlen aus $\mathbb{Z}$). Paare (x,c) und (y,c') mit $x = y$ dürfen in D **nicht gleichzeitig** vorkommen. Weiter soll D folgende Operationen mit jeweils gegebenem Laufzeitverhalten unterstützen (n bezeichne dabei die Anzahl der in D enthaltenen Paare):

- $insert(x : Schluessel,\ c : Wert)$
  fügt (x,c) in D ein. Ist schon ein Paar (y,c') mit $y = x$ in D vorhanden, so wird D nicht verändert. Der Zeitbedarf sei erwartet $O(\log n)$.

- $removeMin : Schluessel \times Wert$
  entfernt aus D ein Paar (x,c) mit **minimalem Wert** c und liefert das entfernte Paar als Ergebnis zurück. Der Zeitbedarf sei erwartet $O(\log n)$.

- $contains(x : Schluessel) : boolean$
  stellt fest, ob D ein Paar (y,c) mit $y = x$ enthält. Der Zeitbedarf sei erwartet $O(1)$.

Anwendungsbedingt sei bekannt, dass in D zu jedem Zeitpunkt höchstens m Paare gleichzeitig vorhanden sind, d. h. es gilt stets $n \leq m$. Über die Beschaffenheit der auftretenden Paare wisse man im Voraus aber nichts.

a. Skizzieren Sie, wie Sie diese Datenstruktur und die drei beschriebenen Operationen realisieren würden. [6 Punkte]

Aufgabe 9 - Hashing

Betrachten Sie die folgende Hashtabelle:

0	1	2	3	4	5	6	7	8	9	10
05	01	27	37	44	34	19	71		99	

Zur Kollisionsauflösung wird Hashing mit linearer Suche (mit Puffer $m' = 1$) verwendet. Die Hashfunktion $h(x) = x$ DIV 10 bildet eine zweistellige Zahl $x \in \{01, \ldots, 99\}$ auf ihre höchstwertige Ziffer ab. Beispiel: $h(62) = 6$, $h(06) = 0$.

a. Sei die Tabelle in dem oben angegebenen Zustand. Geben Sie den Zustand der Tabelle nach dem Entfernen von 37 an.

b. Sei die Hashtabelle nun leer. Fügen Sie die Zahlen 12, 17, 88, 89, 05, 22, 48, 49, 50 in dieser Reihenfolge in die Hashtabelle ein.