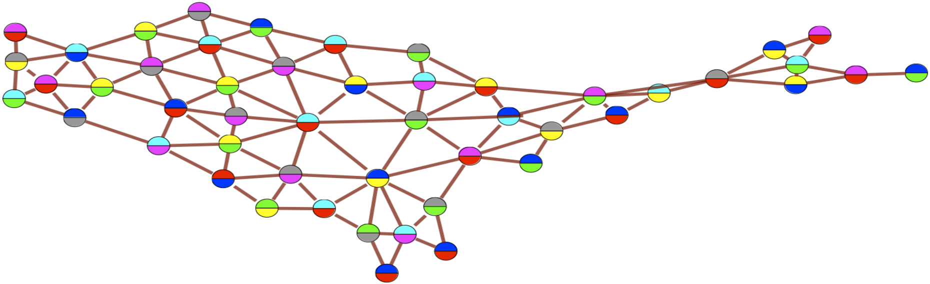
Karlsruher Institut für Technologie

Tutorium 9: Übungsklausur

Holger Ebhart | 17. Juni 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15



Gliederung

7. Übungsblatt

```
deg,active,core:Array [1...|V|] of Digit // active = 1
Q:AddressablePriorityQueue
foreach {u, v} ∈ E do
    deg[u]++
    deg[v]++
Q.buildHeap(V,deg)

while !Q.empty()
    v := Q.deleteMin() :Node
    core[v] := deg[v]
    active[v] := 0
    foreach {u, v} ∈ E and active[v] = 1 do
        if deg[u] > deg[v] then
            deg[u]–
            Q.decreaseKey(u, deg[u])
return  core
```

Übungsklausur

Aufgabe

Es soll ein *dynamisiertes Adjazenzarray* entwickelt werden. Gesucht ist eine Datenstruktur für gerichtete Graphen $G = (V, E)$ mit folgenden Eigenschaften:

- *Stabile und eindeutige Knoten-IDs.* Knoten sollen durch IDs eindeutig identifiziert werden. Diese IDs sollen Zahlen aus $\mathbb{N}_{\geq 0}$ sein. Dabei seien die KnotenIDs *stabil*, d. h. die ID eines Knotens ändere sich nie solange dieser Knoten exisiert (nach Entfernen eines Knotens darf dessen ID jedoch neu vergeben werden).
- *Eindeutige Kanten-IDs.* Die Kanten sollen ebenfalls durch IDs eindeutig identifiziert werden. Allerdings müssen diese nicht unbedingt Zahlen aus $\mathbb{N}_{\geq 0}$ sein und sie müssen auch nicht stabil sein.

Aufgabe

Es soll ein *dynamisiertes Adjazenzarray* entwickelt werden. Gesucht ist eine Datenstruktur für gerichtete Graphen $G = (V, E)$ mit folgenden Eigenschaften:

- *Stabile und eindeutige Knoten-IDs.* Knoten sollen durch IDs eindeutig identifiziert werden. Diese IDs sollen Zahlen aus $\mathbb{N}_{\geq 0}$ sein. Dabei seien die KnotenIDs *stabil*, d. h. die ID eines Knotens ändere sich nie solange dieser Knoten exisiert (nach Entfernen eines Knotens darf dessen ID jedoch neu vergeben werden).
- *Eindeutige Kanten-IDs.* Die Kanten sollen ebenfalls durch IDs eindeutig identifiziert werden. Allerdings müssen diese nicht unbedingt Zahlen aus $\mathbb{N}_{\geq 0}$ sein und sie müssen auch nicht stabil sein.

Aufgabe

Es soll ein *dynamisiertes Adjazenzarray* entwickelt werden. Gesucht ist eine Datenstruktur für gerichtete Graphen $G = (V, E)$ mit folgenden Eigenschaften:

- *Stabile und eindeutige Knoten-IDs.* Knoten sollen durch IDs eindeutig identifiziert werden. Diese IDs sollen Zahlen aus $\mathbb{N}_{\geq 0}$ sein. Dabei seien die KnotenIDs *stabil*, d. h. die ID eines Knotens ändere sich nie solange dieser Knoten exisiert (nach Entfernen eines Knotens darf dessen ID jedoch neu vergeben werden).
- *Eindeutige Kanten-IDs.* Die Kanten sollen ebenfalls durch IDs eindeutig identifiziert werden. Allerdings müssen diese nicht unbedingt Zahlen aus $\mathbb{N}_{\geq 0}$ sein und sie müssen auch nicht stabil sein.

Aufgabe

- *Effizienter wahlfreier Zugriff auf Knoten und Kanten.* Es gibt die Operationen
  - $\mathit{node}(u : \mathit{NodeID}) : \mathit{Handle}$ **of** Node und
  - $\mathit{edge}(e : \mathit{EdgeID}) : \mathit{Handle}$ **of** Edge,

  die in $\mathcal{O}(1)$ Zeit einen Handle auf das Knoten- bzw. Kantenobjekt zu einer Knoten- bzw. Kanten-ID liefern.

Aufgabe

- *Effiziente Navigation.* Es gibt die Operationen
  - $\mathit{firstEdge}(v : \mathit{NodeID}) : \mathit{EdgeID} \cup \{\perp\}$ und
  - $\mathit{nextEdge}(e : \mathit{EdgeID}) : \mathit{EdgeID} \cup \{\perp\}$,

  mit deren Hilfe wie folgt in einem Graph G über alle ausgehenden Kanten eines Knoten v iteriert werden kann:

  for ($\mathit{EdgeID}\ e := \mathit{graph.firstEdge}(v)$; $e \neq \perp$; $e := \mathit{nextEdge}(e)$)
      $h_e := G.\mathit{edge}(e) : \mathit{Handle}$ **of** Edge
      / do something */*
  end for

  Sowohl *firstEdge* als auch *nextEdge* dürfen höchstens $\mathcal{O}(1)$ Zeit brauchen.

Aufgabe

- *Effiziente Navigation.* Es gibt die Operationen
  - $\mathit{firstEdge}(v : \mathit{NodeID}) : \mathit{EdgeID} \cup \{\perp\}$ und
  - $\mathit{nextEdge}(e : \mathit{EdgeID}) : \mathit{EdgeID} \cup \{\perp\}$,

  mit deren Hilfe wie folgt in einem Graph G über alle ausgehenden Kanten eines Knoten v iteriert werden kann:

  for ($\mathit{EdgeID}\ e := \mathit{graph.firstEdge}(v)$; $e \neq \perp$; $e := \mathit{nextEdge}(e)$)
  $\quad h_e := G.\mathit{edge}(e) : \mathit{Handle}$ **of** Edge
  $\quad$ */* do something */*
  end for

  Sowohl *firstEdge* als auch *nextEdge* dürfen höchstens $\mathcal{O}(1)$ Zeit brauchen.

Aufgabe

Karlsruher Institut für Technologie

- *Amortisiert konstantes Einfügen von Knoten und Kanten.* Es gibt Operationen
  - $insertNode : NodeID$ und
  - $insertEdge(u, v : NodeID) : EdgeID$,

  die in amortisiert konstanter Zeit einen neuen Knoten bzw. eine neue Kante von u nach v einfügen und jeweils die ID des neu erzeugten Elements zurückliefern. Beide Operationen dürfen höchstens *amortisiert* konstante Zeit brauchen.
- *Amortisiert konstantes Entfernen von Knoten und Kanten.* Es gibt Operationen
  - $deleteNode(v : NodeID)$ und
  - $deleteEdge(e : EdgeID)$,

  die einen Knoten bzw. eine Kante entfernen. Der Einfachheit halber darf ein Knoten dabei nur entfernt werden, wenn bereits alle seine Kanten entfernt worden sind. Beide Operationen dürfen höchstens amortisiert konstante Zeit brauchen.

Aufgabe

- *Amortisiert konstantes Einfügen von Knoten und Kanten.* Es gibt Operationen
  - $insertNode : NodeID$ und
  - $insertEdge(u, v : NodeID) : EdgeID$,

  die in amortisiert konstanter Zeit einen neuen Knoten bzw. eine neue Kante von u nach v einfügen und jeweils die ID des neu erzeugten Elements zurückliefern. Beide Operationen dürfen höchstens *amortisiert* konstante Zeit brauchen.
- *Amortisiert konstantes Entfernen von Knoten und Kanten.* Es gibt Operationen
  - $deleteNode(v : NodeID)$ und
  - $deleteEdge(e : EdgeID)$,

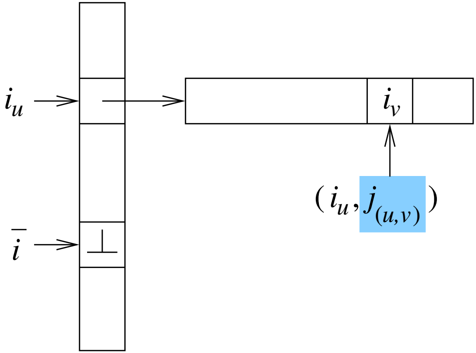
  die einen Knoten bzw. eine Kante entfernen. Der Einfachheit halber darf ein Knoten dabei nur entfernt werden, wenn bereits alle seine Kanten entfernt worden sind. Beide Operationen dürfen höchstens amortisiert konstante Zeit brauchen.

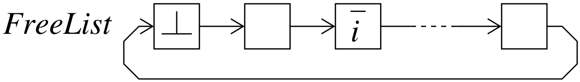
Aufgabe

Karlsruher Institut für Technologie

a) Überlegen Sie sich, wie Sie diese Datenstruktur realisieren.
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 (abhängig von aktuellen oder zwischenzeitlichen Werten von $|V|$ und $|E|$ und das nicht nur im $\mathcal{O}$-Kalkül)? Wieviel im besten Fall? Vergleichen Sie mit dem Speicherverbrauch des statischen Adjazenzfeldes aus der Vorlesung.

Lösungsskizze





Vielen Dank für eure Aufmerksamkeit!
Bis zum nächsten Mal.

stackoverflow.com