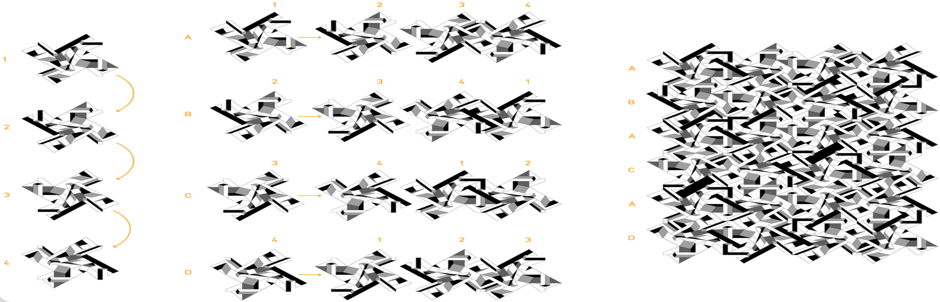
Karlsruher Institut für Technologie

Tutorium 13: Wiederholung

Holger Ebhart | 15. Juli 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15



Gliederung

11. Übungsblatt

Es sei ein Graph G=(V,E) gegeben. Für seine Kantengewichte gelte $\varphi : E \rightarrow \{a, b\}, \varphi(e) = a \vee \varphi(e) = b$ mit $a < b$ und $a, b \in \mathbb{N}$.

Finden Sie einen Algorithmus der in $\mathcal{O}(|E|)$ einen MST von G berechnet.

Einführung und Grundlagen

1. Wie multipliziert man 6 mit 123 mit dem *numberTimesDigit* Algorithmus?
   Führen Sie die Multiplikation beispielhaft aus.

Einführung und Grundlagen

1. Wie multipliziert man 6 mit 123 mit dem *numberTimesDigit* Algorithmus?
   Führen Sie die Multiplikation beispielhaft aus.

   low: |6|2|8|
   high: |0|1|1| – |
   carry:|0|0|0| – |
   result: |0|7|3|8|

Einführung und Grundlagen

1. Wie multipliziert man 6 mit 123 mit dem *numberTimesDigit* Algorithmus?
   Führen Sie die Multiplikation beispielhaft aus.

   low: |6|2|8|
   high: |0|1|1| – |
   carry:|0|0|0| – |
   result: |0|7|3|8|

2. Geben Sie die Definition von Θ an.

Einführung und Grundlagen

1. Wie multipliziert man 6 mit 123 mit dem *numberTimesDigit* Algorithmus?
   Führen Sie die Multiplikation beispielhaft aus.

   low: $|6|2|8|$
   high: $|0|1|1| - |$
   carry: $|0|0|0| - |$
   result: $|0|7|3|8|$

2. Geben Sie die Definition von Θ an.

   $\Theta(f(n)) = \{g(n) : \exists c_1, c_2 > 0 : \exists n_0 \in \mathbb{N}_+ : \forall n \geq n_0 : c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)\}$

Einführung und Grundlagen

3. Geben Sie das einfache Mastertheorem an und bestimmen Sie die Laufzeit von $T(n)$ mit
   $T(n) = 16 \cdot T(\frac{n}{4}) + 6n$

Einführung und Grundlagen

3. Geben Sie das einfache Mastertheorem an und bestimmen Sie die Laufzeit von $T(n)$ mit
   $T(n) = 16 \cdot T(\frac{n}{4}) + 6n$

$$T(n) = \begin{cases} \Theta(n) & \text{falls } d < b \\ \Theta(n \log(n)) & \text{falls } d = b \\ \Theta(n^{\log_b d}) & \text{falls } d > b \end{cases}$$

$$T(n) \in \Theta(n^{\log_4 16}) = \Theta(n^2)$$

Einführung und Grundlagen

3. Geben Sie das einfache Mastertheorem an und bestimmen Sie die Laufzeit von $T(n)$ mit
   $T(n) = 16 \cdot T(\frac{n}{4}) + 6n$
   $$T(n) = \begin{cases} \Theta(n) & \text{falls } d < b \\ \Theta(n \log(n)) & \text{falls } d = b \\ \Theta(n^{\log_b d}) & \text{falls } d > b \end{cases}$$
   $T(n) \in \Theta(n^{\log_4 16}) = \Theta(n^2)$
4. Ordnen Sie die Laufzeitabschätzungen in aufsteigender Weise an:
   $\mathcal{O}(\log(n!)), \mathcal{O}(\log(\log(n))), \mathcal{O}(\log(n)), \mathcal{O}(\frac{1}{2}^n), \mathcal{O}(\log(n^n)),$
   $\mathcal{O}(\sqrt[3]{n}), \mathcal{O}(n \log(n!))$

Einführung und Grundlagen

3. Geben Sie das einfache Mastertheorem an und bestimmen Sie die Laufzeit von $T(n)$ mit
   $T(n) = 16 \cdot T(\frac{n}{4}) + 6n$

   $$T(n) = \begin{cases} \Theta(n) & \text{falls } d < b \\ \Theta(n \log(n)) & \text{falls } d = b \\ \Theta(n^{\log_b d}) & \text{falls } d > b \end{cases}$$

   $T(n) \in \Theta(n^{\log_4 16}) = \Theta(n^2)$

4. Ordnen Sie die Laufzeitabschätzungen in aufsteigender Weise an:
   $\mathcal{O}(\log(n!)), \mathcal{O}(\log(\log(n))), \mathcal{O}(\log(n)), \mathcal{O}(\frac{1}{2}^n), \mathcal{O}(\log(n^n)),$
   $\mathcal{O}(\sqrt[3]{n}), \mathcal{O}(n \log(n!))$

   $\mathcal{O}(\frac{1}{2}^n) < \mathcal{O}(\mathcal{O}(\log(\log(n))) < \mathcal{O}(\log(n)) <$
   $\mathcal{O}(\sqrt[3]{n}) < \mathcal{O}(\log(n^n)) = \mathcal{O}(\log(n!)) < \mathcal{O}(n \log(n!))$

Listen, Felder, Arrays

5. Geben Sie die Invariante für doppelt und einfach verkettete Listen an.

Listen, Felder, Arrays

5. Geben Sie die Invariante für doppelt und einfach verkettete Listen an.

   doppelt verkettete Listen: next $\rightarrow$ prev = prev $\rightarrow$ next = this
   einfach verkette Listen: $\forall u \in$ Item: indegree(u)=1

Listen, Felder, Arrays

Karlsruher Institut für Technologie

5. Geben Sie die Invariante für doppelt und einfach verkettete Listen an.

   doppelt verkettete Listen: next $\rightarrow$ prev = prev $\rightarrow$ next = this
   einfach verkette Listen: $\forall u \in$ Item: indegree(u)=1

6. Erklären Sie die Funktion *splice* (Wofür braucht man sie und was macht sie wie).

Listen, Felder, Arrays

5. Geben Sie die Invariante für doppelt und einfach verkettete Listen an.

   doppelt verkettete Listen: next $\rightarrow$ prev = prev $\rightarrow$ next = this
   einfach verkette Listen: $\forall u \in$ Item: indegree(u)=1

6. Erklären Sie die Funktion *splice* (Wofür braucht man sie und was macht sie wie).

   splice(a,b,t) schneidet einen Teil einer doppelt verketteten Liste aus (beginnt mit Element a und endet mit Element b) und fügt ihn hinter Element t wieder ein

Listen, Felder, Arrays

5. Geben Sie die Invariante für doppelt und einfach verkettete Listen an.

   doppelt verkettete Listen: next $\rightarrow$ prev = prev $\rightarrow$ next = this
   einfach verkette Listen: $\forall u \in$ Item: indegree(u)=1

6. Erklären Sie die Funktion *splice* (Wofür braucht man sie und was macht sie wie).

   splice(a,b,t) schneidet einen Teil einer doppelt verketteten Liste aus (beginnt mit Element a und endet mit Element b) und fügt ihn hinter Element t wieder ein

7. Führen Sie eine amortisierte Analyse für die *pushBack* Operation bei dynamischen Arrays durch (Gehen Sie davon aus, dass sich die Arraygröße jeweils verdoppelt).

Listen, Felder, Arrays

5. Geben Sie die Invariante für doppelt und einfach verkettete Listen an.

   doppelt verkettete Listen: next $\rightarrow$ prev = prev $\rightarrow$ next = this
   einfach verkette Listen: $\forall u \in$ Item: indegree(u)=1

6. Erklären Sie die Funktion *splice* (Wofür braucht man sie und was macht sie wie).

   splice(a,b,t) schneidet einen Teil einer doppelt verketteten Liste aus (beginnt mit Element a und endet mit Element b) und fügt ihn hinter Element t wieder ein

7. Führen Sie eine amortisierte Analyse für die *pushBack* Operation bei dynamischen Arrays durch (Gehen Sie davon aus, dass sich die Arraygröße jeweils verdoppelt).

8. Erklären Sie die Funktionsweise eines *cyclic Arrays*.

Hashing

9. Erläutern Sie die drei möglichen Implementierungen von Hashtabellen mit ihren Vor- und Nachteilen.

Hashing

Karlsruher Institut für Technologie

9. Erläutern Sie die drei möglichen Implementierungen von Hashtabellen mit ihren Vor- und Nachteilen.

    - direktes Hashing (Array mit n Slots für n Elemente)
    - Hashing mit verketteten Listen
    - Hashing mit linearer Suche

Hashing

9. Erläutern Sie die drei möglichen Implementierungen von Hashtabellen mit ihren Vor- und Nachteilen.
   - direktes Hashing (Array mit n Slots für n Elemente)
   - Hashing mit verketteten Listen
   - Hashing mit linearer Suche
10. Geben Sie die Datenstrukturinvariante von Hashtabellen an und erklären Sie was eine universelle Hashfunktion ist.

9. Erläutern Sie die drei möglichen Implementierungen von Hashtabellen mit ihren Vor- und Nachteilen.
   - direktes Hashing (Array mit n Slots für n Elemente)
   - Hashing mit verketteten Listen
   - Hashing mit linearer Suche
10. Geben Sie die Datenstrukturinvariante von Hashtabellen an und erklären Sie was eine universelle Hashfunktion ist.

    $\forall e \in M : t[h(key(e))] = e$ und $\forall i \in \{0, ..., m-1\} : t[i] \in M \cup \{\perp\}$

Sortieren und Prioritätslisten

11. Geben Sie zu jedem Sortierverfahren die worst-case Laufzeit an und ob das Verfahren inplace und/oder stabil ist: Mergesort, Heapsort, Insertionsort, Radixsort, Quicksort

Sortieren und Prioritätslisten

11. Geben Sie zu jedem Sortierverfahren die worst-case Laufzeit an und ob das Verfahren inplace und/oder stabil ist: Mergesort, Heapsort, Insertionsort, Radixsort, Quicksort

Mergesort	$\mathcal{O}(n\log(n))$	stabil	**nicht** inplace
Heapsort	$\mathcal{O}(n\log(n))$	**nicht** stabil	inplace
Insertionsort	$\mathcal{O}(n^2)$	stabil	inplace
Quicksort	$\mathcal{O}(n^2)$	**nicht** stabil	(inplace)
Radixsort	$\mathcal{O}(d(n+K))$	stabil	**nicht** inplace

Sortieren und Prioritätslisten

11. Geben Sie zu jedem Sortierverfahren die worst-case Laufzeit an und ob das Verfahren inplace und/oder stabil ist: Mergesort, Heapsort, Insertionsort, Radixsort, Quicksort

Mergesort	$\mathcal{O}(n\log(n))$	stabil	**nicht** inplace
Heapsort	$\mathcal{O}(n\log(n))$	**nicht** stabil	inplace
Insertionsort	$\mathcal{O}(n^2)$	stabil	inplace
Quicksort	$\mathcal{O}(n^2)$	**nicht** stabil	(inplace)
Radixsort	$\mathcal{O}(d(n+K))$	stabil	**nicht** inplace

12. Wie lautet die Invariante eines Max-Heaps? Wie erreicht man *parent* und *childs* in einem Triären-Baum in Array-Darstellung?

Sortieren und Prioritätslisten

11. Geben Sie zu jedem Sortierverfahren die worst-case Laufzeit an und ob das Verfahren inplace und/oder stabil ist: Mergesort, Heapsort, Insertionsort, Radixsort, Quicksort

Mergesort	$\mathcal{O}(n\log(n))$	stabil	**nicht** inplace
Heapsort	$\mathcal{O}(n\log(n))$	**nicht** stabil	inplace
Insertionsort	$\mathcal{O}(n^2)$	stabil	inplace
Quicksort	$\mathcal{O}(n^2)$	**nicht** stabil	(inplace)
Radixsort	$\mathcal{O}(d(n+K))$	stabil	**nicht** inplace

12. Wie lautet die Invariante eines Max-Heaps? Wie erreicht man *parent* und *childs* in einem Triären-Baum in Array-Darstellung?

Invariante: $\forall v$: parent(v)$\geq v$

parent(j) = $\lfloor \frac{j}{3} \rfloor$

leftChild(j) = $3j$

middleChild(j) = $3j+1$

rightChild(j) = $3j+2$

Folgen und Suchbäume

13. Erklären Sie die Funktion *locate(e)* in einem binären-Suchbaum und geben Sie die dazugehörige Invariante an.

Folgen und Suchbäume

13. Erklären Sie die Funktion *locate(e)* in einem binären-Suchbaum und geben Sie die dazugehörige Invariante an.

    Invariante: Für jeden besuchten Knoten k gilt: *all* e' *left of* $k \leq e$ *und all* e' *right of* $k > e$

Folgen und Suchbäume

13. Erklären Sie die Funktion *locate(e)* in einem binären-Suchbaum und geben Sie die dazugehörige Invariante an.

    Invariante: Für jeden besuchten Knoten k gilt: *all* e' *left of* $k \leq e$ *und all* e' *right of* $k > e$

14. Bauen Sie aus $M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ einen (2,4)-Suchbaum und führen Sie *remove(3)* und *remove(2)* aus. Geben Sie außerdem die Einschränkungen an a und b eines (a,b)-Suchbaumes an.

Folgen und Suchbäume

Karlsruher Institut für Technologie

13. Erklären Sie die Funktion *locate(e)* in einem binären-Suchbaum und geben Sie die dazugehörige Invariante an.

    Invariante: Für jeden besuchten Knoten k gilt: *all e' left of $k \leq e$ und all e' right of $k > e$*

14. Bauen Sie aus $M = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ einen (2,4)-Suchbaum und führen Sie *remove(3)* und *remove(2)* aus. Geben Sie außerdem die Einschränkungen an a und b eines (a,b)-Suchbaumes an.

    Bedingungen: $b \geq 2a - 1$ und $a \geq 2$

Graphen

15. Vergleichen und erklären Sie die verschieden Methoden Graphen in Rechnern darzustellen.

Graphen

15. Vergleichen und erklären Sie die verschieden Methoden Graphen in Rechnern darzustellen.
    - Kantenliste
    - Adjazenzmatrix
    - Adjazenzfeld
    - Adjazenzarray

Graphen

15. Vergleichen und erklären Sie die verschieden Methoden Graphen in Rechnern darzustellen.
    - Kantenliste
    - Adjazenzmatrix
    - Adjazenzfeld
    - Adjazenzarray
16. Erklären Sie was ein DAG ist und geben Sie Pseudocode für die Funktion *isDAG* an.

Graphen

15. Vergleichen und erklären Sie die verschieden Methoden Graphen in Rechnern darzustellen.

    - Kantenliste
    - Adjazenzmatrix
    - Adjazenzfeld
    - Adjazenzarray

16. Erklären Sie was ein DAG ist und geben Sie Pseudocode für die Funktion *isDAG* an.

    Ein DAG ist ein gerichteter zyklenfreier Graph (directed acyclic graph).

procedure isDAG(G=(V,E))
    while $\exists v \in V$: outdegree(v)=0 **do**
        $V := V \setminus \{v\}$
        $E := E \setminus (\{v\} \times V \cup V \times \{v\})$
    return $|V| = 0$

Graphalgorithmen

17. Erläutern Sie wie die Breitensuche einen Graphen traversiert anhand von Pseudocode.

17. Erläutern Sie wie die Breitensuche einen Graphen traversiert anhand von Pseudocode.

```
function bfs(s)
    Q := ⟨ s ⟩
    while Q ≠ ⟨⟩ do
        explore nodes in Q
        note nodes for next layer in Q'
        Q := Q'
```

Graphalgorithmen

18. Erläutern Sie was es mit der DFS-Nummerierung und der Fertigstellungszeit auf sich hat.

Graphalgorithmen

18. Erläutern Sie was es mit der DFS-Nummerierung und der Fertigstellungszeit auf sich hat.

DFS-Nummerierung

init	dfsPos := 1 : $\{1, \ldots, n\}$
root(s)	dfsNum[s] := dfsPos++
traverseTreeEdge(v,w)	dfsNum[w] := dfsPos++

Fertigstellungszeit

init	fTime := 1 : $\{1, \ldots, n\}$
backtrack(u,v)	finishTime[v] := fTime++

Graphalgorithmen

19. Was ist eine starke Zusammenhangskomponente und was muss in ihr gelten bzw. was muss es mindestens geben?

Graphalgorithmen

19. Was ist eine starke Zusammenhangskomponente und was muss in ihr gelten bzw. was muss es mindestens geben?

    Eine starke Zusammenhangskomponente ist ein Teilgraph G' von G für den gilt: $\forall u, v \in V' : \exists Pfad(u, x), ..., (y, v)$ mit $x, y \in V'$
    Es muss also mindestens einen Zyklus in G' geben.
    Im Allgemeinen ist eine starke Zusammenhangskomponente eine Vereinigung von Zyklen.

Kürzeste Wege und MST

Karlsruher Institut für Technologie

20. Wie funktioniert Dijkstras Algorithmus? Geben Sie auch die allgemeine Laufzeit und die Invariante/n an.

Kürzeste Wege und MST

20. Wie funktioniert Dijkstras Algorithmus? Geben Sie auch die allgemeine Laufzeit und die Invariante/n an.

    Laufzeit:$\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$
    Invarianten: Es gilt stets $\forall v \in V : d[v] \geq \mu(v)$ und $parent[v]$ bezeugt $d[v]$

Kürzeste Wege und MST

20. Wie funktioniert Dijkstras Algorithmus? Geben Sie auch die allgemeine Laufzeit und die Invariante/n an.

    Laufzeit:$\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$
    Invarianten: Es gilt stets $\forall v \in V : d[v] \geq \mu(v)$ und $parent[v]$ bezeugt $d[v]$

21. Erläutern Sie die Schnitteigenschaft und geben Sie eine Beweisskizze dafür an.

Kürzeste Wege und MST

20. Wie funktioniert Dijkstras Algorithmus? Geben Sie auch die allgemeine Laufzeit und die Invariante/n an.

    Laufzeit:$\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$
    Invarianten: Es gilt stets $\forall v \in V : d[v] \geq \mu(v)$ und $parent[v]$ bezeugt $d[v]$

21. Erläutern Sie die Schnitteigenschaft und geben Sie eine Beweisskizze dafür an.

    Die leichteste Kante in einem Schnitt $C = \{\{u, v\} \in E : u \in S, v \in V \setminus S\} (S \subset V)$ kann in einem MST verwendet werden.

Kürzeste Wege und MST

20. Wie funktioniert Dijkstras Algorithmus? Geben Sie auch die allgemeine Laufzeit und die Invariante/n an.

    Laufzeit: $\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$
    Invarianten: Es gilt stets $\forall v \in V : d[v] \geq \mu(v)$ und $parent[v]$ bezeugt $d[v]$

21. Erläutern Sie die Schnitteigenschaft und geben Sie eine Beweisskizze dafür an.

    Die leichteste Kante in einem Schnitt $C = \{\{u, v\} \in E : u \in S, v \in V \setminus S\} (S \subset V)$ kann in einem MST verwendet werden.

22. Erläutern Sie die Kreiseigenschaft und geben Sie eine Beweisskizze dafür an.

Kürzeste Wege und MST

20. Wie funktioniert Dijkstras Algorithmus? Geben Sie auch die allgemeine Laufzeit und die Invariante/n an.

    Laufzeit: $\mathcal{O}(|V| \cdot (T_{deleteMin}(|V|) + T_{insert}(|V|)) + |E| \cdot T_{decreaseKey}(|E|))$
    Invarianten: Es gilt stets $\forall v \in V : d[v] \geq \mu(v)$ und $parent[v]$ bezeugt $d[v]$

21. Erläutern Sie die Schnitteigenschaft und geben Sie eine Beweisskizze dafür an.

    Die leichteste Kante in einem Schnitt $C = \{\{u, v\} \in E : u \in S, v \in V \setminus S\} (S \subset V)$ kann in einem MST verwendet werden.

22. Erläutern Sie die Kreiseigenschaft und geben Sie eine Beweisskizze dafür an.

    Die schwerste Kante eines Kreises wird nicht für den MST benötigt.

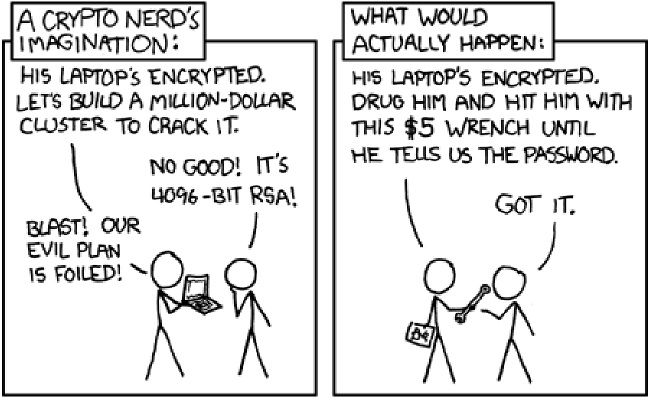
Fragen?

Feedback

Bitte schreibt auf was euch am Tutorium gut gefallen hat und was nicht und was ich verbessern sollte.

Vielen Dank für eure Aufmerksamkeit!

VIEL ERFOLG FÜR DIE PRÜFUNG!!!



stackoverflow.com