Algorithmen und Programmierung 3, WS 2019/2020 — 10. Übungsblatt

Abgabe bis Freitag, 10. Januar 2020, 12:00 Uhr, in die Fächer der Tutoren

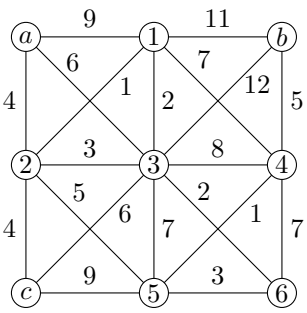
64. Addition einer Konstante zu den Kantengewichten, 7 Punkte

(a) Kann sich der kürzeste *Weg* von s nach t in einem gerichteten Graphen G ändern, wenn man eine Konstante $C > 0$ zu allen Kantengewichten des Graphen addiert? Begründen Sie Ihre Aussage.

(b) Kann sich der kürzeste *Spannbaum* in einem ungerichteten Graphen G ändern, wenn man eine Konstante $C > 0$ zu allen Kantengewichten des Graphen addiert? Begründen Sie Ihre Aussage.

(c) Wir suchen den kürzesten *zusammenhängenden Teilgraphen*, der alle Knoten eines ungerichteten Graphen G enthält. In der Vorlesung wurde argumentiert, dass der kürzeste Spannbaum dieses Problem löst, wenn alle Kantengewichte positiv sind. Wie löst man dieses Problem, wenn der Graph G auch Kanten mit negativem Gewicht enthält?

65. Kürzestes Verbindungsnetzwerk, 0 Punkte



Bestimmen Sie im nebenstehenden ungerichteten Netzwerk mit Kantengewichten den kürzesten Teilgraphen,

(a) der die Knoten a und b verbindet,

(b) der die Knoten a, b und c verbindet.

66. Der gierige Algorithmus, 0 Punkte

Die Jedi-Ritter wollen ihre Rebellenbasis auf dem Planeten *Orbis Pictus* gegen das Imperium verteidigen. Die Basis wird durch eine gerade Mauer mit einer Länge von ℓ Meilen geschützt, die zwei mächtige Felsen verbindet. Die Jedi müssen die Tore in der Mauer sichern. Die Positionen x_i der n Tore sind gegeben: $0 \leq x_1 < x_2 < \cdots < x_n \leq \ell$. Ein Ritter kann einen Umkreis von m Meilen überblicken, das heißt, ein Ritter, der an Position y steht, kann alle Tore i im Bereich $y - m \leq x_i \leq y + m$ überwachen und gegen Angriffe sichern. Können Sie den Jedi-Rittern helfen, die kleinste Zahl von Wächtern zu berechnen, die notwendig sind, um alle Tore zu überwachen? Geben Sie einen effizienten Algorithmus für dieses Problem an, und erklären Sie, warum er korrekt ist.

67. Volladdierer, Vorübung, 0 Punkte

Ein Volladdierer berechnet aus drei Eingabebits x, y, c zwei Ausgabebits r und c' nach folgender Formel, wobei $\oplus$ das exklusive Oder (XOR) ist:

$$\big(r \equiv (x \oplus y \oplus c)\big) \wedge \big(c' \equiv ((x \wedge y) \vee (c \wedge (x \vee y)))\big)$$

Schreiben Sie eine dazu äquivalente Formel in konjunktiver Normalform.

68. Hashtabellen, 6 Punkte

(a) Fügen Sie nacheinander die Schlüssel 5, 28, 19, 15, 20, 33, 12, 17, 10 in eine verkettete Hashtabelle der Größe 9 ein. Die Hashfunktion ist $h(x) = x \bmod 9$.

(b) Fügen Sie nacheinander die Schlüssel 10, 22, 31, 4, 15, 28, 17, 88, 59 in eine Hashtabelle der Größe 11 ein. Die Hashfunktion ist $h(x) = x \bmod 11$. Die Konflikte werden durch offene Adressierung mit linearem Sondieren gelöst.

(c) (0 Punkte) Fügen Sie nacheinander die Schlüssel 10, 22, 31, 4, 15, 29, 17, 88, 59 in eine Hashtabelle der Größe 11 ein. Die Konflikte werden durch Kuckuckshashing mit $h_1(x) = x \bmod 11$ und $h_2(x) = (x \bmod 13) \bmod 11$ behandelt.

Illustrieren Sie jeweils die einzelnen Schritte, insbesondere beim Kuckuckshashing.

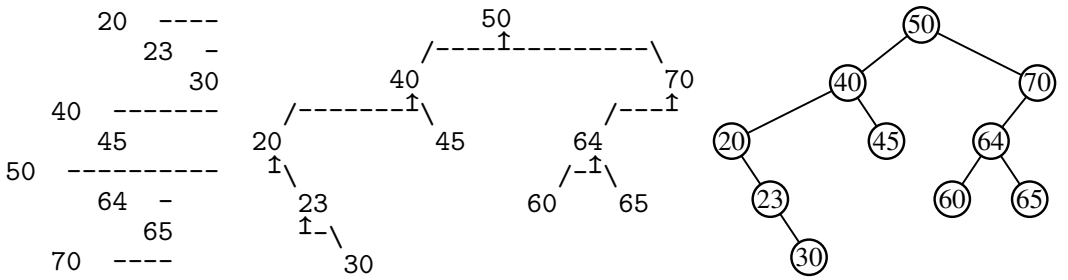
69. Löschen mit linearem Sondieren, 7 Punkte

(a) In der Vorlesung wurde gezeigt, dass es bei offener Adressierung nicht reicht, für ein entferntes Element bloß den Platz freizumachen, den es belegt. Zeigen Sie an einem Beispiel, dass auch das folgende Verfahren zum Entfernen des Schlüssels x im Zusammenhang mit linearem Sondieren nicht funktioniert, weil spätere Suchoperationen einen vorhandenen Schlüssel möglicherweise nicht finden.

$p := h(x)$
while $T[p] \neq x$:
    if $T[p] = \mathit{null}$: *return* "x nicht vorhanden"
    $p := (p + 1) \bmod M$
$p_0 := p$ // Position von x
while $T[(p + 1) \bmod M] \neq \mathit{null}$:
    $p := (p + 1) \bmod M$
$T[p_0] := T[p]$; $T[p] := \mathit{null}$

(b) Schreiben Sie in Pseudocode ein korrektes Verfahren zum Entfernen, das wirklich einen Platz in der Tabelle freimacht, und begründen Sie, warum es funktioniert.

70. (0 Punkte) Schreiben Sie ein Programm, das einen (nicht zu großen) binären Baum zweidimensional in übersichtlicher und schön lesbarer Form darstellt. Drei Vorschläge[1]:



71. Bestimmen des doppelten Elementes, 0 Punkte

In einem Feld $a_0, a_1, \ldots, a_n$ sind ganzzahlige Werte zwischen 1 und n gespeichert, und zwar kommt jede Zahl mindestens einmal vor. Daraus folgt, dass es genau eine Zahl geben muss, die doppelt vorkommt. Schreiben Sie ein Programm, das diese Zahl findet. Das Programm soll lineare Laufzeit haben, auf das Feld a nur *lesend* zugreifen, und nur konstanten zusätzlichen Speicher benötigen.[2]

72. Rettung der Geschenke, 0 Punkte

Der *Weihnachtsmann* ist mit seinem Rentierschlitten in die Mitte eines kreisförmigen Sees mit Umfang 100 m gefallen. Am Rande des Sees befindes sich ein *Grintsch*, der nicht schwimmen, aber 4,5-mal so schnell laufen kann wie die Rentiere den Schlitten im Wasser ziehen können. Auf dem Land und in der Luft ist der Rentierschlitten jedoch schneller als der Grintsch. Kann der Weihnachtsmann die Rentiere so lenken, dass die Weihnachtsgeschenke nicht dem Grintsch in die Hände fallen und er die Pakete noch rechtzeitig abliefern kann? Wenn ja, welchen Mindestabstand vom Grintsch kann der Weihnachtsmann erzielen, wenn er das rettende Ufer erreicht und von dort abheben kann? Der Einfachheit halber stelle man sich Grintsch und Rentierschlitten jeweils punktförmig vor.[3]

[1] Das dritte Beispiel ist in PostScript erstellt, siehe die Datei `Baum.eps`, `https://mycampus.imp.fu-berlin.de/x/Bxw2tg`. Ihr Programm kann sich an dieser Datei als Beispiel orientieren.

[2] Diese Aufgabe ist schwierig; sie dient nur zur Anregung und hat mit dem Stoff der Vorlesung nichts zu tun. Wenn Sie das Rätsel aus einer anderen Quelle bereits kennen, dann würde mich interessieren, woher.

[3] Mit etwas Kreativität kann man diese Aufgabe auch mit Hilfe eines Computers lösen.

68. Hashtabellen, 6 Punkte

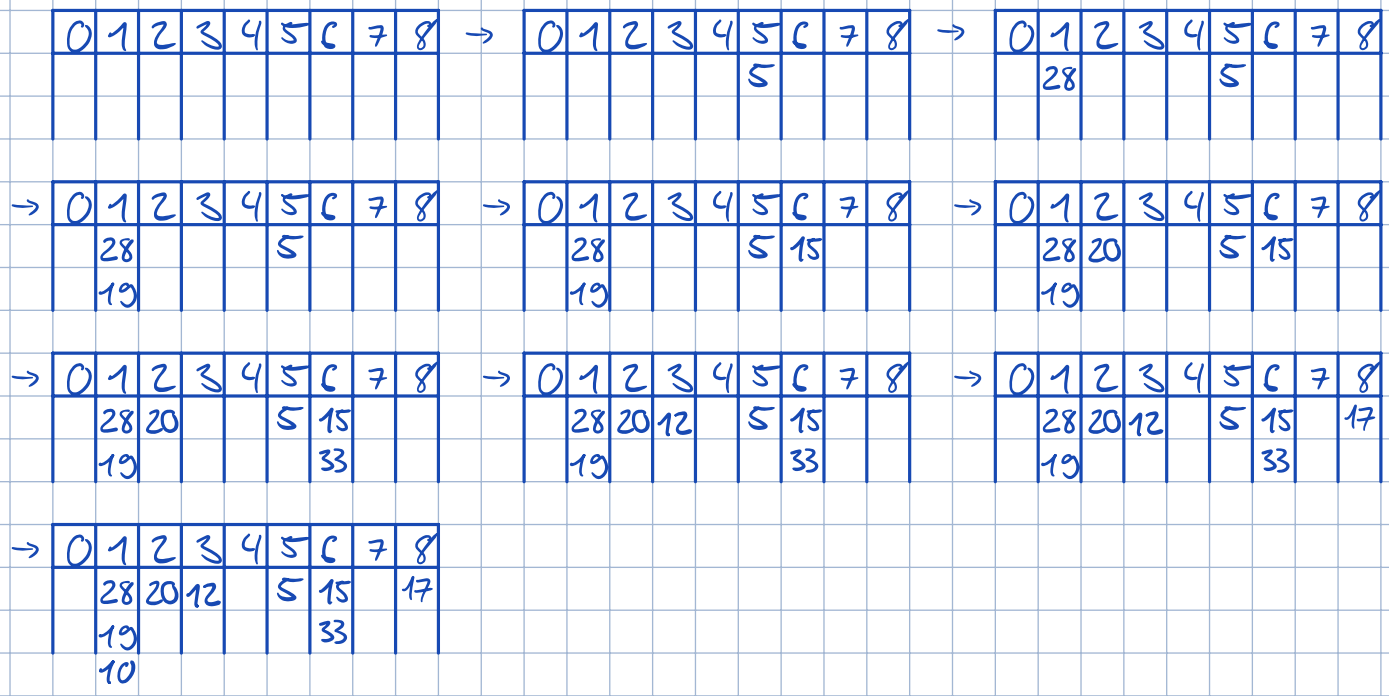
(a) Fügen Sie nacheinander die Schlüssel 5, 28, 19, 15, 20, 33, 12, 17, 10 in eine verkettete Hashtabelle der Größe 9 ein. Die Hashfunktion ist $h(x) = x \bmod 9$.

(b) Fügen Sie nacheinander die Schlüssel 10, 22, 31, 4, 15, 28, 17, 88, 59 in eine Hashtabelle der Größe 11 ein. Die Hashfunktion ist $h(x) = x \bmod 11$. Die Konflikte werden durch offene Adressierung mit linearem Sondieren gelöst.

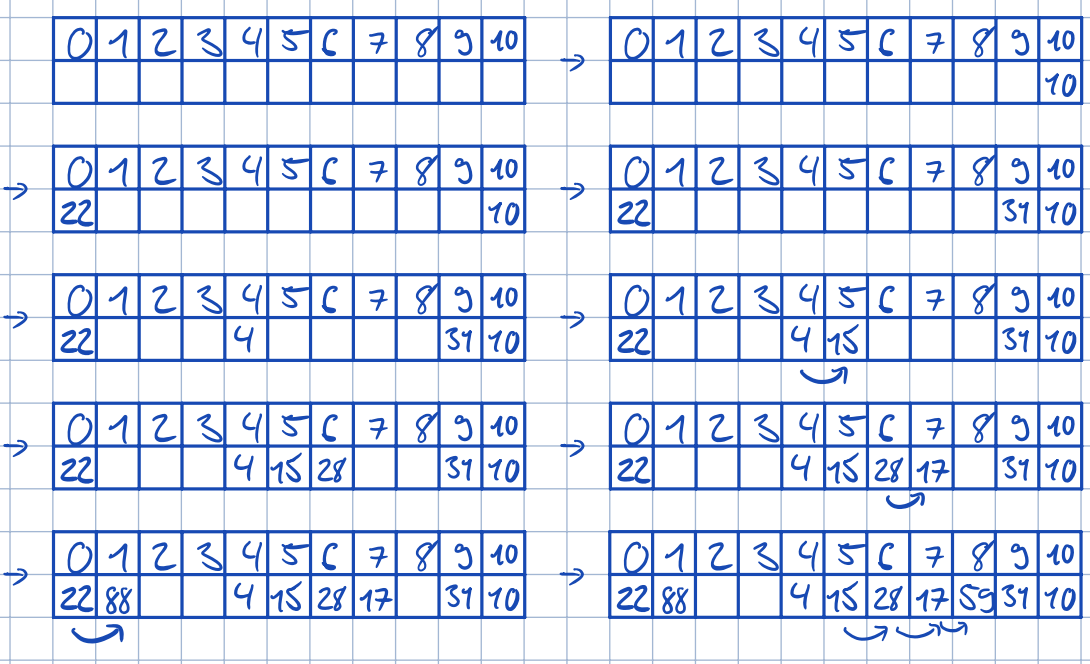
(c) (0 Punkte) Fügen Sie nacheinander die Schlüssel 10, 22, 31, 4, 15, 29, 17, 88, 59 in eine Hashtabelle der Größe 11 ein. Die Konflikte werden durch Kuckuckshashing mit $h_1(x) = x \bmod 11$ und $h_2(x) = (x \bmod 13) \bmod 11$ behandelt.

Illustrieren Sie jeweils die einzelnen Schritte, insbesondere beim Kuckuckshashing.

a)



b)



64. Addition einer Konstante zu den Kantengewichten, 7 Punkte

(a) Kann sich der kürzeste *Weg* von s nach t in einem gerichteten Graphen G ändern, wenn man eine Konstante $C > 0$ zu allen Kantengewichten des Graphen addiert? Begründen Sie Ihre Aussage.

(b) Kann sich der kürzeste *Spannbaum* in einem ungerichteten Graphen G ändern, wenn man eine Konstante $C > 0$ zu allen Kantengewichten des Graphen addiert? Begründen Sie Ihre Aussage.

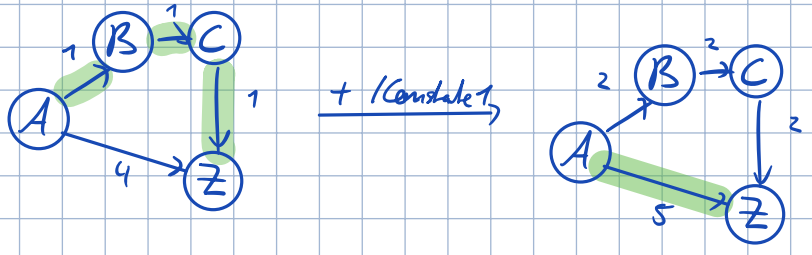
(c) Wir suchen den kürzesten *zusammenhängenden Teilgraphen*, der alle Knoten eines ungerichteten Graphen G enthält. In der Vorlesung wurde argumentiert, dass der kürzeste Spannbaum dieses Problem löst, wenn alle Kantengewichte positiv sind. Wie löst man dieses Problem, wenn der Graph G auch Kanten mit negativem Gewicht enthält?

9 11

a) Ja. Da ein kürzester Weg aus vielen Kanten schneller steigen könnte als ein "längerer" Weg aus weniger Kanten.

Bsp.

A = Anfang
Z = Ziel



Kürzester Weg über A, B, C, Z = 3

neuer Kürzester Weg über A, Z = 5

b) Nein, denn alle kürzesten Spannbäume besitzen die gleiche Anzahl an Kanten.

Das bedeutet, dass wenn eine Konstante C zu allen Kantengewichten dazu addiert wird, alle Spannbäume die gleiche Steigung besitzen.
⇒ Kürzester Spannbaum bleibt der gleiche.

c) Sollte trotztem mit der Alg.'s funktionieren (Kruskal, Prim, Boruvka)

Wenn die Algorithmen negative Kanten nicht vergleichen können, könnte man sie einfach erweitern.

Z.B.
Wähle aus allen Kanten die kürzeste Kante und entferne sie aus dem Graph
(falls mehrere gleich kurz sind, nehme irgendeinen)

- füge sie zum MST hinzu.
- falls mit der Kante ein Kreis entsteht, füge die Kante nicht zum MST hinzu (entferne sie wieder aus dem MST)

⇒ erst werden alle negativen Kantengewichte entfernt und dem MST hinzugefügt
↳ danach kann z.B. Boruvka verwendet werden, um die restlichen positiven Kanten zu finden
⇒ Spannbaum mit negativen Kantengewichten.

69. Löschen mit linearem Sondieren, 7 Punkte

(a) In der Vorlesung wurde gezeigt, dass es bei offener Adressierung nicht reicht, für ein entferntes Element bloß den Platz freizumachen, den es belegt. Zeigen Sie an einem Beispiel, dass auch das folgende Verfahren zum Entfernen des Schlüssels x im Zusammenhang mit linearem Sondieren nicht funktioniert, weil spätere Suchoperationen einen vorhandenen Schlüssel möglicherweise nicht finden.

```
p := h(x)
while T[p] ≠ x:
    if T[p] = null: return "x nicht vorhanden"
    p := (p + 1) mod M
p_0 := p        // Position von x
while T[(p + 1) mod M] ≠ null:
    p := (p + 1) mod M
T[p_0] := T[p]; T[p] := null    // mache die letzte Position des Blockes frei.
```

(b) Schreiben Sie in Pseudocode ein korrektes Verfahren zum Entfernen, das wirklich einen Platz in der Tabelle freimacht, und begründen Sie, warum es funktioniert.

a) Bsp.

0	4	2	
0	1	2	3

Inhalt

Index

mod 4

Löschen von x = 4

```
p := h(x)        p = 4 mod 4 = 0
while T[p] ≠ x:
    if T[p] = null: return "x nicht vorhanden"
    p := (p + 1) mod M
p_0 := p        // Position von x        p = 0+1
while T[(p + 1) mod M] ≠ null:
    p := (p + 1) mod M
T[p_0] := T[p]; T[p] := null
```

T[0] x≠0

0	4	2	
0	1	2	3
p			

```
p := h(x)
while T[p] ≠ x:
    if T[p] = null: return "x nicht vorhanden"
    p := (p + 1) mod M
p_0 := p        // Position von x        p_0 = 1
while T[(p + 1) mod M] ≠ null:
    p := (p + 1) mod M
T[p_0] := T[p]; T[p] := null
```

T[1] x=4 ✓

0	4	2	
0	1	2	3
p_0	p		

```
p := h(x)
while T[p] ≠ x:
    if T[p] = null: return "x nicht vorhanden"
    p := (p + 1) mod M
p_0 := p        // Position von x
while T[(p + 1) mod M] ≠ null:
    p := (p + 1) mod M        p = 1+1
T[p_0] := T[p]; T[p] := null
```

T[1+1] ≠null

0	4	2	
0	1	2	3
p_0	p		

```
p := h(x)
while T[p] ≠ x:
    if T[p] = null: return "x nicht vorhanden"
    p := (p + 1) mod M
p_0 := p        // Position von x
while T[(p + 1) mod M] ≠ null:
    p := (p + 1) mod M
T[p_0] := T[p]; T[p] := null
```

T[2+1] =null ✓

0	4	2	
0	1	2	3
p_0		p	

```
p := h(x)
while T[p] ≠ x:
    if T[p] = null: return "x nicht vorhanden"
    p := (p + 1) mod M
p_0 := p        // Position von x
while T[(p + 1) mod M] ≠ null:
    p := (p + 1) mod M
T[p_0] := T[p]; T[p] := null
```

2 wurde verschoben!

0	4	2	
0	1	2	3
p_0		p	

⇒

0	2	2	
0	1	2	3
p_0		p	

⇒

0	2		
0	1	2	3
p_0		p	

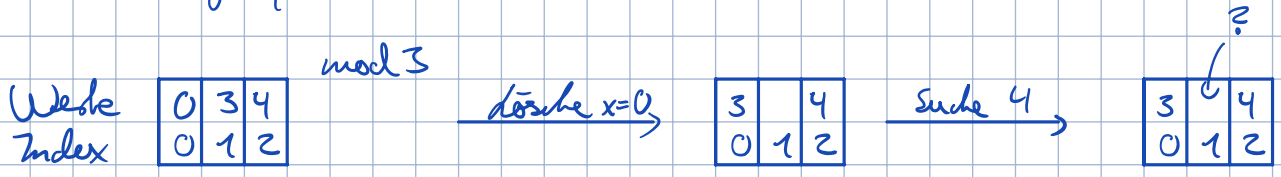
b)

$p := h(x)$
while $T[p] \neq x$:
    if $T[p] = null$: *return* "x nicht vorhanden"
    $p := (p+1) \bmod M$
$p_0 := p$ $p_x := p$
while $T[(p+1) \bmod M] \neq null$: and $(p+1) \bmod M \neq p_0$
    if $(x \bmod M == T[(p+1) \bmod M] \bmod M)$: $p_x := (p+1) \bmod M$
    $p := (p+1) \bmod M$
$T[p_0] := T[p_x]$; $T[p_x] := null$

while $T[(p_x+1) \bmod M] \neq Null$ and $T[(p_x+1) \bmod M] \bmod M \neq (p_x+1) \bmod M$
    $T[p_x] := T[(p_x+1) \bmod M]$; $T[p_x+1 \bmod M] := null$
    $p_x := (p_x+1) \bmod M$

Funktioniert nun, da das Problem aus a) nicht mehr vorkommt (an grüner Stelle behandelt). Es kein Endlosschleife kommt falls die Liste voll ist und etwas gelöscht werden soll (an oranger Stelle). Und durch das Einrücken auch der letzte mögliche Fehler behoben wurde. (an roter Stelle)

Bsp. für letzten Fehler ohne roten Code:



Mit roten Code: