Datenstrukturen und Algorithmen: Blatt 8

Bernhard Dietrich (6256800)
Lars Fernhomberg (6256030)
Sebastian Kniesburges (6257120)
Marcus Köthenbürger (6258550)

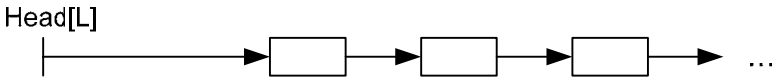
Übungsgruppe 11
Mittwoch, 11:00-13:00 Uhr in D1.303
Matthias Ernst

Aufgabe	1	2	3	4	Σ	Korrektor
Punkte						
von	6	4	4	6	20	

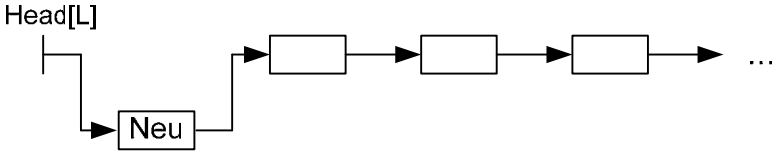
Aufgabe 1

Zeigen Sie wie ein Stack mit Hilfe einer einfach verketteten Liste implementiert werden kann. Sowohl die PUSH- als auch die POP-Operation soll dabei Zeit $O(1)$ benötigen.

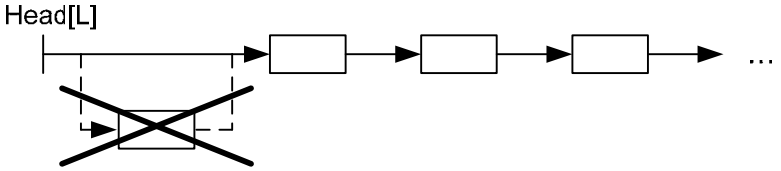
Ein Stack lässt sich mit einer einfach verketteten Liste implementieren, indem immer nur am Anfang der Liste gearbeitet wird.
Die PUSH-Operation fügt somit das neue Element an die erste Stelle ein, während die POP-Operation das Element an der ersten Stelle herausnimmt.
Zur Verdeutlichung:



Einfügen:



Löschen:



Pseudo-Code

```
Push(S,x)
1.    next[x] ← head[S]
2.    head[S] ← x

Pop(S)
1.    if Stack-Empty
2.          then  error „Stack underflow!"
3.          else  x ← head[S]
4.                head[S] ← next[x]
5.                return x
```

Wie man sieht, haben beide Algorithmen eine konstante Laufzeit, da alle Zeilen der Algorithmen jeweils konstant sind. Es ergibt sich somit für beide Algorithmen $O(1)$.

Aufgabe 2

Sie haben ein Wörterbuch durch direkte Adressierung implementiert. Nehmen Sie an, dass alle Schlüssel natürliche Zahlen sind. Wie schnell kann das Objekt mit dem größten Schlüssel gefunden werden?

Bei der direkten Adressierung werden n Elemente mit jeweils einem eindeutigen Schlüssel aus dem Universum U in ein Array $A[1...\#U]$ eingefügt, wobei gilt: $n \leq \#U$.

Das Element mit dem größten Schlüssel ist somit das letzte Element, welches einen konkreten Wert im Array stehen hat (der Wert, welcher an dieser Arrayposition steht, unterscheidet sich somit von einem „Platzhalter“ wie z.B. NIL).

Der Algorithmus hat somit folgende Form:

`getMaxKey(`$A[1...\#U]$`)`

```
1.    i ← length[A]
2.    while A[i] != Nil AND i > 0
3.          do i ← i - 1
4.    if i = 0
5.          then return Nil
6.          else return i
```

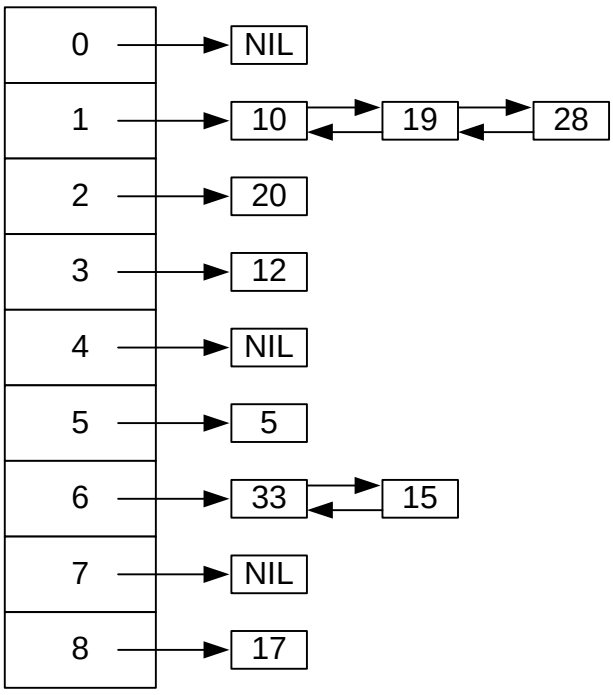
Die Zeilen $1,4,5,6$ haben konstante Laufzeit $O(1)$. Die Schleife in den Zeilen 2 und 3 wird im günstigsten Fall niemals – das letzte Element im Array enthält Daten und somit ist der Index des letzten Elements der größte Schlüssel – im ungünstigsten Fall allerdings $\#U-1$-mal durchlaufen. Dieses ist der Fall, wenn nur ein Element in das entsprechende Array eingefügt wird, das Array aber mehr Elemente enthält.

Somit ergibt sich für die Laufzeit $O(\#U)$

Aufgabe 3

Sie benutzen eine Hashtabelle mit verketteten Listen zur Kollisionsverwaltung. Ihre Hashfunktion h_1 ist gegeben durch $h_1(k) = k \bmod 9$. Ihre Hashtabelle hat damit die Größe 9.

Wie sieht die Hashtabelle nach Einfügen von Objekten mit den Schlüsseln 5, 28, 19, 15, 20, 33, 12, 17 und 10 aus?



Aufgabe 4

Wir benutzen die Hashfunktion h_2 um n Objekte in einer Hashtabelle mit m Positionen zu speichern. Wie groß ist dann unter Annahme des einfachen uniformen Hashings die erwartete Anzahl der Kollisionen, d.h., was ist die erwartete Anzahl von Paaren (k,l) von eingefügten Schlüsseln mit $h_2(k) = h_2(l)$?

Beim einfachen uniformen Hashing streut die Hashfunktion h_2 die n Elemente gleichmäßig, so dass die Wahrscheinlichkeit, dass ein Element an eine bestimmte Position gelangt $\frac{1}{m}$ beträgt.

Da wir n Elemente verteilen müssen, ergibt sich somit $n \cdot \frac{1}{m} = \frac{n}{m}$ als Anzahl der Elemente pro Position.

Der Erwartungswert für die Anzahl der Kollisionen pro Position ergibt sich somit durch $E(\text{Anzahl Kollisionen pro Position}) = \sum_{i=2}^{n} \frac{1}{m^i} \cdot \binom{i}{2}$.

Da der Erwartungswert für alle Positionen gefragt ist, ergibt sich folglich $E(\text{Anzahl Kollisionen}) = n \cdot \sum_{i=2}^{n} \frac{1}{m^i} \cdot \binom{i}{2}$.