Mohamad Raafat Baki

Übung 5 – Listen:

5.1.1. Einfach Verkettete Liste (EVL)

a) Skizzieren Sie eine einfach verkettete Liste mit vier Elementen. Benutzen Sie Pfeile, um Zeiger darzustellen. Die Schlüssel der Elemente seien, in dieser Reihenfolge, die ganzen Zahlen 3, 5, 7 und 9.

b) Skizzieren Sie, wie die Liste aussieht, nachdem als Nachfolger des Elementes mit dem Schlüssel 7 ein neues Element mit dem Schlüssel 8 eingefügt wurde. Aus der Skizze soll erkenntlich sein, welche Zeiger gleichgeblieben sind und welche wie verändert wurden.

c) Skizzieren Sie, wie die Liste aussieht, nachdem nunmehr der Nachfolger des Elementes mit dem Schlüssel 5 aus der Liste gelöscht wurde. Aus der Skizze soll erkenntlich sein, welche Zeiger gleichgeblieben sind und welche wie verändert wurden.

d) Schreiben Sie eine C Typdefinition für einen Datentyp der für eine Implementierung der obigen Liste geeignet ist.

a)

|  |  |
| --- | --- |
| 7 |  |

|  |  |
| --- | --- |
| 9 |  |

L

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 5 |  |

Null

b)

|  |  |
| --- | --- |
| 7 |  |

L

|  |  |
| --- | --- |
| 3 | * v |

|  |  |
| --- | --- |
| 5 |  |

|  |  |
| --- | --- |
| 8 |  |

|  |  |
| --- | --- |
| 9 |  |

Null

|  |  |
| --- | --- |
| 7 |  |

|  |  |
| --- | --- |
| 8 |  |

|  |  |
| --- | --- |
| 9 |  |

c) L

|  |  |
| --- | --- |
| 3 |  |

5.1.2. Doppelt Verkettete Liste (DVL)

a) Skizzieren Sie eine doppelt verkettete Liste mit vier Elementen. Die Schlüssel der Elemente seien, in dieser Reihenfolge, die ganzen Zahlen 2, 4, 6 und 8.

b) Skizzieren Sie, wie die Liste aussieht, nachdem als Nachfolger des Elementes mit dem Schlüssel 6 ein neues Element mit dem Schlüssel 7 eingefügt wurde. Aus der Skizze soll erkenntlich sein, welche Zeiger gleichgeblieben sind und welche wie verändert wurden.

c) Skizzieren Sie, wie die Liste aussieht, nachdem nunmehr der Nachfolger des Elementes mit dem Schlüssel 4 aus der Liste gelöscht wurde. Aus der Skizze soll erkenntlich sein, welche Zeiger gleichgeblieben sind und welche wie verändert wurden.

d) Schreiben Sie eine C Typdefinition für einen Datentyp der für eine Implementierung der obigen Liste geeignet ist.

a)

|  |
| --- |
| 6 |
|  |
|  |

|  |
| --- |
| 8 |
|  |
|  |

* head tali

|  |
| --- |
| 2 |
|  |
|  |

|  |
| --- |
| 4 |
|  |
|  |

b)

|  |
| --- |
| 6 |
|  |
|  |

|  |
| --- |
| 7 |
|  |
|  |

|  |
| --- |
| 8 |
|  |
|  |

* head .tail

|  |
| --- |
| 2 |
|  |
|  |

|  |
| --- |
| 4 |
|  |
|  |

c)

|  |
| --- |
| 6 |
|  |
|  |

|  |
| --- |
| 7 |
|  |
|  |

* head . tail

|  |
| --- |
| 2 |
|  |
|  |

|  |
| --- |
| 8 |
|  |
|  |

5.2. Listen und Pointer

5.2.1.

Betrachten Sie die schematische Darstellung einer doppelt verketteten Liste unten.

a) was ist der Wert von p->next->key

b) was ist der Wert L->next->prev->next->key

c) Fügen Sie in der Skizze die fehlenden Null-Pointer hinzu.

a) next.key : 17

b) prev : 0 next : 11

|  |
| --- |
| p |

c)

|  |
| --- |
| L |

|  |
| --- |
| key: 9  next:  prev: . |

|  |
| --- |
| key: 11  next:  prev: |

|  |
| --- |
| key: 17  next: .  prev: |

|  |
| --- |
| Null-  Pointer |

|  |
| --- |
| Null-Pointer |

5.3. Pseudocode mit Listen

5.3.1. Dezimalzahl aus Binärrepräsentation

Eingabe ist eine einfach verkettete Liste L. Die Listenelemente sind binäre Ziffern Null (0) und Eins (1). Schreiben Sie einen Algorithmus in Pseudocode, der die Liste als Binärzahl interpretiert und die durch die Liste repräsentierte Dezimalzahl berechnet und diese als Ergebnis zurückgibt. Das erste Listenelement ist dabei das höchstwertige Bit usw. bis zum letzten Listenelement (niederwertigstes Bit). Es darf nur 𝒪(1) zusätzlicher Speicher verwendet werden. Jedes Listenelement darf nur einmal besucht werden. Beispiel: Eingabe L = 1►0►1►0►0►Ausgabe: 20.

struct List

num key

List next

var p5 = new List

p5.key = 1

p5.next = null

var p4 = new List

p4.key = 0

p4.next = p5

var p3 = new List

p3.key = 1

p3.next = p4

var p2 = new List

p2.key = 0

p2.next = p3

var p1 = new List

p1.key = 0

p1.next = p2

var L = p1

var r = new List

r.next = L.next

func Dez(L)

var x = 0

var dez = 0

while L != null

r.key = L.key \* 2^(x)

L = L.next

x = x + 1

dez = dez + r.key

r = r.next

return dez

print Dez(L)

5.3.2. Mittleres Element in einer Liste finden

Eingabe ist eine einfach verkettete Liste L mit aufsteigend sortierten ganzen Zahlen als Schlüssel. Schreiben Sie einen Algorithmus in Pseudocode, der das Element in der Mitte der Liste als Ergebnis zurückgibt. Ist die Länge der Liste gerade, soll das zweite der beiden mittleren Elemente zurückgegeben werden. Dabei dürfen keine arithmetischen Operationen ausgeführt werden, insbesondere dürfen die Elemente in der Liste nicht gezählt werden. Die Zeitkomplexität Ihrer Lösung muss 𝒪(n) betragen und es darf nur 𝒪(1) zusätzlicher Speicher verwendet werden.

Beispiel: Eingabe L= 1►1►2►3►5►8►11 Ergebnis: 3. Tipp: Verwenden Sie mehrere Pointer.

struct List

num key

List next

var p7 = new List

p7.key = 11

p7.next = null

var p6 = new List

p6.key = 8

p6.next = p7

var p5 = new List

p5.key = 5

p5.next = p6

var p4 = new List

p4.key = 3

p4.next = p5

var p3 = new List

p3.key = 2

p3.next = p4

var p2 = new List

p2.key = 1

p2.next = p3

var p1 = new List

p1.key = 1

p1.next = p2

var L = p1

func Mitte(L)

var q = L

var q2 = L

while (q2 != null) && (q2.next != null)

q = q.next

q2 = q2.next.next

return q.key

print Mitte(L)

5.3.3. Finden von zwei Elementen mit einer vorgegebenen Summe

Eingabe ist eine doppelt verkettete Liste L mit aufsteigend sortierten ganzen Zahlen als Schlüssel sowie eine weitere ganze Zahl x. Schreiben Sie einen Algorithmus in Pseudocode, der ermittelt ob die Liste L zwei Elemente enthält deren Summe x ergibt. Falls ja, soll der Algorithmus 1 ausgeben und 0 sonst. Beispiel: H=-1 2 3 5 9 11=T und x=12 ergibt „ja“ weil 3+9=12 ist. Die Zeitkomplexität Ihrer Lösung muss 𝒪(𝑛) Zeitkomplexität haben und es darf nur 𝒪(1) zusätzlicher Speicher verwendet werden.

struct List

num key

List next

List prev

var p6 = new List

p6.key = 11

p6.next = null

var p5 = new List

p5.key = 9

p5.next = p6

p6.prev = p5

var p4 = new List

p4.key = 5

p4.next = p5

p5.prev = p4

var p3 = new List

p3.key = 3

p3.next = p4

p4.prev = p3

var p2 = new List

p2.key = 2

p2.next = p3

p3.prev = p2

var p1 = new List

func sum\_liste(h,t,x)

var p = new List

var p2 = new List

p = h

p7 = t

while h != t

if x == h.key + t.key

return 1

if x > h.key + t.key

h = h.next

if x < h.key + t.key

t = t.prev

return 0

print sum\_liste(p6,p1,8)

5.3.5. eine EVL L in annähernd gleichgroße Listen L1 und L2 zerlegen (split)

Eingabe ist eine EVL (einfach verkettete Liste) L mit ganzen Zahlen als Schlüssel. sowie zwei leere Listen L1 und L2. Schreiben Sie einen möglichst effizienten Algorithmus split(L,L1,L2) in Pseudocode, der die Elemente aus L entfernt und abwechselnd am Tail der Listen L1 und L2 hinzufügt. War die Anzahl der Elemente in L anfangs ungerade, soll L1 am Ende ein Element mehr enthalten als L2. Die Zeitkomplexität Ihrer Lösung muss 𝒪(n) Zeitkomplexität haben und es darf nur 𝒪(1) zusätzlicher Speicher verwendet werden.

Beispiel: Eingabe: L = 1►5►6►3►8►7►4►10 Ausgabe: L1 =1►6►8►4 ,L2 = 5►3►7►10.

struct List

num key

List next

var p8 = new List

p8.key = 10

p8.next = null

var p7 = new List

p7.key = 4

p7.next = p8

var p6 = new List

p6.key = 7

p6.next = p7

var p5 = new List

p5.key = 8

p5.next = p6

var p4 = new List

p4.key = 3

p4.next = p5

var p3 = new List

p3.key = 6

p3.next = p4

var p2 = new List

p2.key = 5

p2.next = p3

var p1 = new List

p1.key = 1

p1.next = p2

var L = p1

func split(L,L1,L2)

if L == null

throw "error"

L1 = L

if L.next != null

L = L.next

else

return

L2 = L

if L.next != null

L = L.next

else

return

var p = L1

var q = L2

while L != null

p.next = L

if L.next != null

L = L.next

p = p.next

if L.next != null

q.next = L

L = L.next

q = q.next

p.next = null

q.next = null

print split(L,L1,L2)