Hashing

Hvordan fungerer hashing?

- Start med et array, og nøkkel-verdi par
- Hash nøkkelen vha. hashfunksjon for å få en indeks
- Sett inn verdien på denne indeksen
  - Håndter evt. kollisjoner
  - Utvid/minsk arrayet om nødvendig
- Håndter oppslag og sletting

Hashfunksjoner

- Input: En nøkkel og maks-indeks
- Returverdi: Indeksen som nøkkelen "hasher" til
- Alltid samme output for samme input
- Må gi få kollisjoner

Kollisjonshåndtering

- → Separate chaining
  - → "Bøtter" av kolliderende elementer
- → Linear probing
  - → "Kjeder" av kolliderende elementer

```
38
39  def settInn_SepChain(key, value):
40      i = hash(key)
41
42      bucket = A[i]
43      if bucket == null then:
44          newBucket = new LinkedList().add(Element(key, value))
45          A[i] = newBucket
46      else:
47          for Element(k,v) in bucket do:
48              if key == k then:
49                  element.v = value
50                  return
51          end
52          bucket.add(Element(key, value))
53
```

```
53
54  def oppslag_SepChain(key):
55      i = hash(key)
56
57      bucket = A[i]
58      if bucket == null then:
59          return null
60
61      for Element(k,v) in bucket do:
62          if key == k then:
63              return Element(k,v)
64      end
65      return null
66
67
68
```

```
69
70  def sletting_SepChain(key):
71      i = hash(key)
72
73      bucket = A[i]
74      if bucket == null then:
75          return null
76
77      for Element(k,v) in bucket do:
78          if key == k then:
79              Fjern Element(k,v)
80              return Element(k,v)
81      end
82      return null
83
84
```

Seperate chaining

```
def settInn_LinProb(i, key, value):
    if A[i] == null then:
        A[i] = Element(key, value)
    else if A[i].k == key then:
        A[i].v = value
    else:
        i = (i + 1) mod N
        settInn_LinProb(i, key, value)
```

```
111
112 def oppslag_LinProb(i, key):
113     if A[i] == null then:
114         return null
115     else if A[i].k == key then:
116         return A[i]
117     else:
118         i = (i + 1) mod N
119         oppslag_LinProb(i, key)
120
121
122
123
124
125
126
```

```
133
134 def sletting_LinProb(i, key):
135     if A[i] == null then:
136         return null
137
138     # Det ligger noe paa indeksen
139     if A[i].k != key then:
140         i = (i + 1) mod N
141         sletting_LinProb(i, key)
142
143     else if A[i].k == key then:
144         returverdi = A[i]
145         // Marker plassen som slettet, bruker et tomt element
146         A[i] = Element(null, null)
147         return returverdi
148
```

Linear probing

Tette hull ved linear probing

- → Marker plassen som "slettet" (brukt i forrige slide)
- → Tett hullet ved å flytte på elementer

Effektivitet

- Ikke bra med for mange eller for få elementer i arrayet
- Ideel bør load factor ligge mellom 0.5 og 0.75
  - (Altså 50%-75% fylt opp)
- "Så godt som" O(1) på alle operasjoner

Modulo

- (a mod b) = a - (round(a / b) * b)
- https://larstvei.github.io/linear-probing/
- Bruker i = i + 1 mod N
  - Hvorfor gjør vi det?

Implementasjon av hashing

→ https://larstvei.github.io/hashing/

Linear probing

2 poeng

Vi starter med et tomt array på størrelse 10.

0	1	2	3	4	5	6	7	8	9

Hashfunksjonen du skal bruke er $h(k, N) = k \mod N$, som for dette eksempelet blir det samme som $h(k) = k \mod 10$. Altså hasher et tall til sitt siste siffer.

Bruk *linear probing* til å sette inn disse tallene i den gitte rekkefølgen:

$$93, 48, 74, 99, 29, 13, 45$$

Fyll ut tabellen slik den ser ut etter alle tallene er satt inn med linear probing.

Eksamensoppgave H20