Separate chaining

```
proc create(capacity)
  return array with size capacity, with default value of empty list

proc rehash(set)
  newSet ← create(size of set * 2)
  for bucket in set do
    for value in bucket do
      insert(newSet, value)
  return newSet

proc ensure_capacity(set)
  if elements stored in set > (size of set / 2) then
    return rehash(set)
  return set

proc insert(set, value):
  set ← ensure_capacity(set)
  i ← hash(value) % size of set
  bucket ← set[i]
  if value not in bucket then
    elements stored in set += 1
    bucket.append(value)

proc contains(set, value)
  i ← hash(value) % size of set
  bucket ← set[i]
  if value in bucket then
    return true
  return false

proc delete(set, value)
  i ← hash(value) % size of set
  bucket ← set[i]
  if value in bucket then
    delete value from bucket
    elements stored in set -= 1
```

Pseudokoden er basert på kode fra notatet om hashing: https://pages.github.uio.no/IN2010/notater/hashing/

Linear probing

```
proc create(capacity)
  return array with size capacity

proc rehash(set)
  newSet ← create(size of set * 2)
  for every non-null value in set do
    insert(newSet, value)
  return newSet

proc ensure_capacity(set)
  if elements stored in set > (size of set / 2) then
    return rehash(set)
  return set

proc insert(set, value):
  set ← ensure_capacity(set)
  i ← hash(value) % size of set
  while set[i] ≠ null do
    if set[i] = value then
      return
    i = (i + 1) % size of set
  elements stored in set += 1
  set[i] ← value


proc contains(set, value)
  i ← hash(value) % size of set
  while set[i] ≠ null do
    if set[i] = value then
      return true
    i = (i + 1) % size of set
  return false

proc delete(set, value)
  i ← hash(value) % size of set
  while set[i] ≠ null do
    if set[i] = value then
      elements stored in set -= 1
      set[i] ← null
      fill_hole(set, i)
      return
    i = (i + 1) % size of set

proc fill_hole(set, hole)
  distance = 1
  while set[(hole + distance) % size of set] ≠ null do
    value ← set[(hole + distance) % size of set]
    value_belongs = hash(value) % size of set
    if !(0 < (value_belongs - hole) % size of set ≤ distance) then
      set[hole] ← value
      set[(hole + distance) % size of set] = null
      fill_hole(set, (hole + distance) % size of set)
      return
    distance += 1
```