Aufgaben und Lösungen

Aufgabe 1

8x vorgekommen
Beschreiben Sie jeweils eine Lösung für das Union-Find-Problem mit Laufzeit

1. $O(\log n)$ (amortisiert) für UNION und O(1) für FIND
2. O(1) für UNION und O(log$n$) für FIND

wobei n die Anzahl der Elemente ist. Begründen Sie in beiden Fällen die entsprechenden Laufzeiten.

Lösung

O(log n) (amortisiert) für UNION und O(1) für FIND

Feld name[1...n]:	name[x] = Name des Blocks der x enthält. $1 \leq x \leq n$
size[1..n]:	size[A] = Anzahl Elemente im Block A, initialisiert mit 1
L[1..n]:	L[A] = Liste aller Elemente in Block A, initialisiert L[i] = {i}

```
Initialisierung:
begin
  for i := 1 to n do
    name[i]=i
    size[i]=1
    L[i]={i}
  end
end

FIND(x):
begin
  return name[x]
end

UNION(A,B):
begin
  if size[A] ≤ size[B] then
    foreach i in L[A] do
      name[i] = B
    end
    size[B] += size[A]
    L[B] = L[B].concat(L[A])
  else
    foreach i in L[B] do
      name[i] = A
    end
    size[A] += size[B]
    L[A] = L[A].concat(L[B])
  end
end
```

Laufzeit: FIND(x): O(1) $\rightarrow$ Einfacher Zugriff auf ein Feld
UNION: O(log n) $\rightarrow$ x kann maximal log(n) mal seinen Namen ändern, da es sich nach jeder Namensänderung in einer doppelt so großen Menge befindet. (Die kleinere Menge wird umbenannt)

O(1) für UNION und O(log n) für FIND

Feld name[1...n]:	name[x] = Name des Blocks mit Wurzel x (hat nur Bedeutung, falls x Wurzel)
Feld vater[1...n]:	vater[x] = $\begin{cases} Vater\ von\ x\ in\ seinem\ Baum \\ 0,\ falls\ x\ Wurzel \end{cases}$
Feld wurzel[1...n]:	wurzel[x] = Wurzel des Blocks mit Namen x
Feld size[1..n]:	size[x] = Anzahl Knoten im Unterbaum mit Wurzel x

```
Initialisierung:
begin
    for i := 1 to n do
        vater[i]=0
        name[i]=i
        wurzel[i]=i
    end
end

FIND(x):
begin
    while vater[x] != 0 do
        x = vater[x]
    end
    return name[x]
end

UNION(A,B,C):
begin
    r1 = wurzel[A]
    r2 = wurzel[B]
    if size[r1] ≤ size[r2] then
        vater[r1] = r2
        name[r2] = C
        wurzel[C] = r2
        size[r2] += size[r1]
    else
        vater[r2] = r1
        name[r1] = C
        wurzel[C] = r1
        size[r1] += size[r2]
    end
end
```

Laufzeit:
FIND(x): O(log n) $\rightarrow$ Weighted UNION Rule
UNION: O(1) $\rightarrow$ Nur Pointer ändern

Warum hat der Baum logarithmische Höhe/Tiefe? Im Worst-Case wird ein UNION auf zwei gleich große und gleich tiefe Bäume ausgeführt. Dabei ist die Größe von C doppelt so groß wie die ursprünglichen Bäume, jedoch ist die Tiefe nur um 1 gewachsen ($\log(size(x)) \geq Hoehe(x)$)

Aufgabe 2

8x vorgekommen
Entwickeln Sie eine Datenstruktur zur Speicherung von n Schlüsseln aus dem Universum $\{1, ..., N\}$(wobei $n << N$), die eine Zugriffszeit von O(1) garantiert. Sie dürfen dabei $O(n^2)$ Speicherplatz verwenden.

5x vorgekommen
(Perfektes Hashing) Verbessern Sie die Datenstruktur aus Aufgabe , so dass nur noch Speicherplatz $O(n)$ benutzt wird.

Hashig durch Verkettung und mit offener Adressierung (Linear Probing:Wie funktioniert Delete())

Lösung

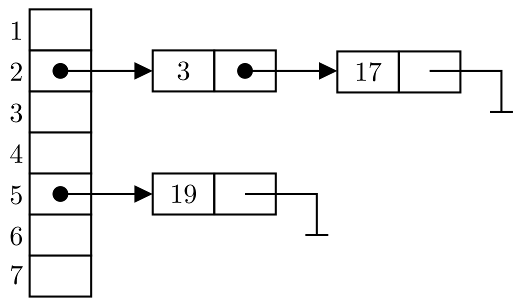
Hashing mit Verkettung

löse Kollisionen nicht auf, speichere mehrere Schlüssel an der gleichen Position

Speichere für jedes Ergebnis der Hashfunktion h eine Liste
Lookup(x): lineare Suche in Liste $T[h(x)]$

- Worst Case: alle Keys in derselben List → O(n)
- erwartete Zeit: O($\frac{n}{m}$)
- Belegungsfactor $\beta = \frac{n}{m} \leftarrow$ erw. Länge einer Liste T[x]
- wenn $m \geq n$, d.h. $\beta \leq 1$ dann → erw. Laufzeit O(1)

Insert(x): $x \notin S$. Füge x an erst freie Stelle in $T[h(x)]$ ein
Delete(x): Entferne x aus $T[h(x)]$



meist wird als Hashfunktion einfaches Modulo verwendet.

Verbesserung Verdopplungs-Strategie:

- Immer wenn $\beta > 2$, verdopple Tafelgröße $\rightarrow$ 1 sehr teures Insert (da alle Elemente mit neuer Hashfunktion umgespeichert werden), im Schnitt aber weiter O(1)

- Bei Delete und kleinem β: Tabelle kann kalbiert werden $\rightarrow$ Ein sehr teures Delte, im Schnitt aber weiter O(1)

Zusatzaufgabe: Perfektes Hashing