# Altklausuren Antworten

Landmesser Zusammenfassung

## Aufgabe 1

### 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) → Einfacher Zugriff auf ein Feld
UNION: O(log $n$) → x kann maximal log(n) mal seinen Namen ändern, da es sich nach jeder Namensänderung in einer doppelt so großen Menge befindet.

## 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 |

```
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] ≤ 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 $n$) → Tiefe von x (max Höhe des entstehende Baums, n-1 möglich)

UNION: O(1) $\rightarrow$ Nur Pointer ändern

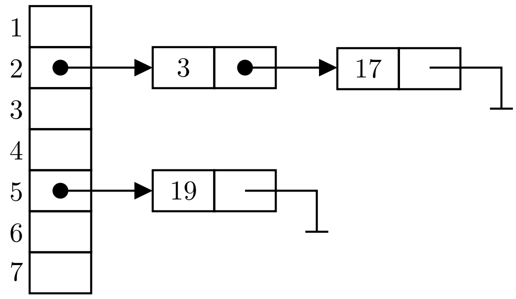
# Aufgabe 2

**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 $\rightarrow$ 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 $\rightarrow$ 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 $\beta$: Tabelle kann kalbiert werden $\rightarrow$ Ein sehr teures Delte, im Schnitt aber weiter O(1)

## Zusatzaufgabe: Perfektes Hashing

# Aufgabe 3

# Aufgabe 4

Sei G ein planarer Graph mit $n \geq 3$ Knoten und $m$ Kanten, dann gilt $m = 3n-6$. dh $m = O(n)$, also linear viele Kanten.

*H.* Ein maximal planarer Graph ist ein planarer Graph, der durch Hinzufügen einer Kante $(v, w) \notin E$ nicht-planar wird. Beobachtung: Alle Faces in jeder planaren Einbettung von G sind Dreiecke (Triangulierung). Jedes Face in einer Triangulierung hat 3 Rand-Kanten und jeder Kante liegt am Rand von 2 Faces.

$$\Rightarrow 3f = 2m$$

Einsetzen in Euler-Formel

$$n - m + \frac{2}{3}m = 2$$
$$m = 3n - 6$$

$m \leq 3n - 6$ für beliebige planare Graphen □

## Zusatzaufgabe

Sei $G$ ein **bipartiter** planarer Graph Dann gilt $m \leq 2n - 4$.

**Beweis:** Keine Kreise ungerader Länge in bipartiten Graphen. Kleinstmögliche Fläche in einem bipartiten Graphen ist ein Viereck. □

# Aufgabe 5

**Split-Find-Problem** Idee - Unkehrung von Union-Find Feld name[1,...,n]

**Initialisierung:**
**begin**
| **for** $\forall i, 1 \leq i \leq n$ **do**
| | name[i] = 1
| **end**
**end**

**Find(i) begin**
| return name[i] $\rightarrow \mathcal{O}(1)$
**end**
**Split(i)**

- Neuer Name des neuen Intervalls das durch Split entsteht ++count
- Relabel the smaller half
- Laufe parallel d.h. abwechselnd nach links und rechts von $i$ aus, bis Intervallgrenze erreicht d.h. $name[i] \neq name[betrachtetesElement]$
- Nenne den Teil um, der kleiner ist $[a, i]$ oder $[i + 1, b]$ indem nochmals über diesen Teil gelaufen wird. name[betrachtetes Element] = count

| a | | | i | | | b |
|---|---|---|---|---|---|---|
| $\cancel{1}$ | $\cancel{1}$ | $\cancel{1}$ | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | | | | |

$\Rightarrow$ Kosten für 1 Split $\mathcal{O}(2 * LaengedeskuerzerenIntervalls)$
**Analyse** $\mathcal{O}(\#Namensaenderungen) : max\frac{\text{Laenge des umzubennenendes Intervall}}{2} \Rightarrow \mathcal{O}(\log n)$

## Aufgabe 6

**Algorithmus von Dijkstra:**

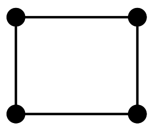
```
begin
    foreach v ∈ V do
        DIST[v]← ∞
        PRED[v]←NULL
    end
    DIST[s]←0
    PQ.insert(v,0)
    while not PQ.empty() do
        u ← PQ.delmin() //liefertInfo
        foreach v ∈ V mit (u, v) ∈ E do
            d←DIST[u]+c(u,v)
            if d¡DIST[v] then
                if DIST[v]==∞ then
                    PQ.insert(v,d)
                end
                else
                    PQ.decrease(v,d)
                end
                DIST[v]←d
                PRED[u]←u
            end
        end
    end
end
```

**Laufzeitanalyse:** $\mathcal{O}(\Sigma_{v\in V}(1 + outdeg(v)) + PQ_Operationen)n * (T_{insert} + T_{delmin} + T_{empty}) + m * T_{decrease}$

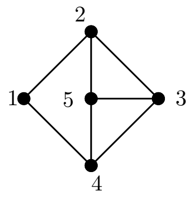
- Bei Binärem Heap: $\mathcal{O}(n * log(n) + m * log(n) = \mathcal{O}((n + m) * log(n)$
- Fibonacci-Heap:Amortisierte Analyse ist ok, da Gesamtlaufzeit betrachtet. $\mathcal{O}(n*log(n)+m)$, insert+empty = $\mathcal{O}(1)$, delmin=$\mathcal{O}(log(n))$, decrease = $\mathcal{O}(1)$

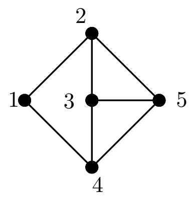
## Aufgabe 7

Eine Planare Einbettung ist genau dann eindeutig wenn diese 3-fach zusammenhängend ist:



Gleicher Graph verschiedene Einbettungen:






## Aufgabe 8