Altklausuren Antworten

Landmesser Zusammenfassung

Aufgabe 1

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

O(1) für UNION und O(log$n$) für FIND

```
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]
  vater[r1] = r2
  name[r2] = C
  wurzel[C] = r2
end
```

Aufgabe 2

Zusatzaufgabe: Perfektes Hashing

Aufgabe3

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.

Beweis. 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 jede Kante liegt am Rand von 3 Faces.

$$\Rightarrow 3f = 2m$$

Einstetzen 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

Aufgabe 6

Algorithmus von Dijkstra:

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

Aufgabe 8

```
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
```