Blatt Nr:08

Gruppe : Nikolic, Hattinger, Nofal

Aufgabe 22

Antwort:

Eine Kollision tritt auf, wenn für zwei verschiedene Eingabewerte derselbe Slot in der Hashtabelle errechnet wird. Mathematisch ausgedrückt also $i \neq j \wedge h(i) = h(j)$. Um die erwartete Anzahl an Kollisionen zu ermitteln, betrachte man also eine Menge $\{(i,j) : i \neq j \wedge h(i) = h(j)\}$. Die Anzahl der in dieser Menge vorkommenden Elemente entspricht dann der erwarteten Anzahl an Kollisionen.

Sei X eine Indikator-Zufallsvariable $X_{ij} = I\{h(i) = h(j)\}$, dann ist der Erwartungswert $E[X_{ij}] = \frac{1}{m}$, da laut Angabe einfaches, uniformes (gleichverteiltes) Hashing verwendet wird.

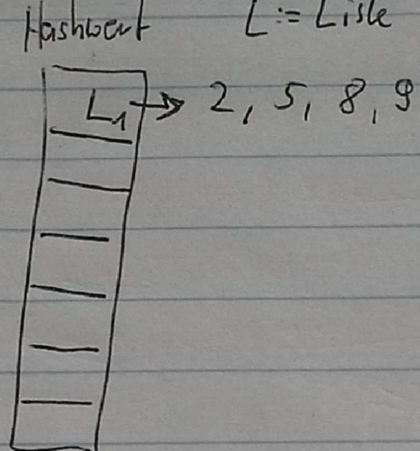
Der Erwartungswert der Summe aller X_{ij} ergibt dann die Anzahl der Kollisionen: $E\left[\sum_{i \neq j} X_{ij}\right] = \sum_{i \neq j} E[X_{ij}]$.

Durch die Eigenschaft des Binomialkoeffizienten $\sum_{j=1}^{n-1} j = \binom{n}{2}$ und Anwendung von $E[X_{ij}] = \frac{1}{m}$ ergibt sich daraus:

$$\binom{n}{2} \cdot \frac{1}{m} = \frac{n!}{2! \cdot (n-2)!} \cdot \frac{1}{m} = \frac{n \cdot (n-1) \cdot (n-2)!}{2 \cdot (n-2)!} \cdot \frac{1}{m} = \frac{n \cdot (n-1)}{2} \cdot \frac{1}{m}$$

Aufgabe 23

Ang.: Hashing-Verfahren mit Chaining



bei z.B. $h(x) = x \bmod 10$

- Wenn Liste unsortiert, so muss jedes Element der Liste überprüft werden, ob es das zu löschende/suchende ~~E~~ ist. $\overset{\text{worst case}}{\Longrightarrow} O(n)$

- Wenn Liste sortiert, so kann diese Eigenschaft z.B. durch eine Form von Binärer Suche ausgenutzt werden, wodurch das Suchintervall innerhalb der Liste immer halbiert wird.
$\Rightarrow$ worst-case $O(\log(n))$ bei Suche und Remove

- Allerdings bei insert ebenso $O(\log(n))$

Aufgabe 24

- Prinzip: Eingangs wird überprüft, ob die Knoten- und Kantenanzahl die Tree-Eigenschaft überhaupt zulassen, denn wären nicht genau $|V| - 1$ Knoten bei einer Kantenanzahl von $|E|$, würde sich nichtmal theoretisch eine Tree konstruieren lassen.
  Anschließend wird die Tree-Eigenschaft geprüft. Finde alle Knoten, die nicht Zielknoten eines anderen Knotens sind (Menge R). Wenn die Kardinalität dieser Menge R ungleich 1 ist, gibt es mehr als eine Wurzel, deshalb kann der Graph kein Tree sein. Ist die Kardinalität hingegen genau 1, wurde die einzig mögliche Wurzel r gefunden. Von dieser gefundenen Wurzel r werden nun alle Knoten rekursiv markiert, die von der Wurzel r zu erreichen sind. Wenn ein Knoten gefunden wird, der bereits markiert ist, wurde ein Kreis entdeckt, wodurch wiederum die Tree-Eigenschaft verletzt wird. Wurden keine Kreis gefunden, wird am Ende noch überpüft, ob auch wirklich alle Knoten des Graphen von der Wurzel r erreichbar sind. Ist dies nicht der Fall ist wiederum die Tree-Eigenschaft verletzt. Sind alle Knoten von der Wurzel r erreichbar und existieren keine Kreise im Graphen, dann ist die Tree-Eigenschaft gegeben.

```
check_tree(G)
if |E| = |V| − 1 return false
R ← V
for each vertex u∈V do
  if v∈Adj[u] then R ← R\{v}
if |R| ≠ 1 then return false
else for each vertex u∈V do color[u] = white
r ← r∈R
cycle_found ← visit_and_check_acyclicity(G, r)
if cycle_found = true then return false
for each vertex u∈V do
   if color[u] = white then return false
return true

visit_and_check_acyclicity(G, u)
color[u] ← black
for each v∈Adj[u] do
  if color[v] = white then visit_and_check_acyclicity(G, v)
  else return true
return false
```

- Durch die Speicherung des Graphen in Form von Adjazenzlisten wird die Laufzeit des anfänglichen Suchens der Wurzel r wesentlich beeinflusst, da von jedem Knoten die gesamten Adjazenzliste durchlaufen werden muss um alle

Verbindungen zu erkennen. Würde man den Graphen als Adjazenzmatrix repräsentieren, würde die Abfrage einer Verbindung in $O(1)$ geschehen, allerdings müssen auch hier alle Knoten durchlaufen werden um eine mögliche Wurzel r zu finden. Der Speicherbedarf für die Adjazenzmatrix ($\Theta(V^2)$ unabh. von der Kantenanzahl) ist jedoch sehr viel höher als der Speicherbedarf bei einer Repräsentation per Adjazenzlisten ($\Theta(V + E)$).