Algorithmik

Sebastian Iwanowski
FH Wedel

7. Vorlesungswoche

Algorithmik 7

Basisalgorithmen für Graphenprobleme

Union-Find-Struktur

Effizientes Finden und Vereinigen von Zusammenhangskomponenten

O(log n)	**`Find (v)`** gibt eindeutigen Referenzknoten zurück aus der Zusammenhangskomponente, in der sich v befindet.
O(1)	**`Union (v,w)`** vereinigt die Zusammenhangskomponenten, in denen sich v bzw. w befinden.

mit Pfadkomprimierung:

Find hat erwartete Laufzeit von O(log*n)

Repräsentierung:

Array aus den Knoten: Inhalt ist der Index der Wurzel sowie die Höhe des Teilbaums

Referenzen zum Nacharbeiten und Vertiefen:

Skript Alt, Kap. 3.2 (S. 56 ff.)

Algorithmik 7

Basisalgorithmen für Graphenprobleme

Heap

Effizientes Verwalten einer Prioritätswarteschlange

Invarianten:
1) Ein Heap ist ein vollständiger Binärbaum (nur in letzter Ebene dürfen Elemente fehlen).
2) Die Kinder jedes Knotens sind kleiner gleich dem Knoten selbst.

O(log n) **`DeleteMin()`** löscht minimales Element.
O(log n) **`Insert (v)`** fügt beliebiges neues Element ein.
O(1) **`SearchMin()`** findet minimales Element.

Repräsentierung:

Array aus den Baumknoten:
Die Inhalte sind die Inhalte der Baumknoten.
Kinder (i) sind die Arrayelemente 2i und 2i+1 (falls das Array bei 1 beginnt)

Referenzen zum Nacharbeiten und Vertiefen:

Cormen, Kap. 6

Algorithmik 7

SSSP: Single Source Shortest Path

Finde die kürzesten Wege von einer ausgewählten Quelle s zu allen anderen Knoten

Anm.: Für das Problem, den kürzesten Weg zwischen zwei ausgewählten Knoten zu finden, ist kein besserer Algorithmus als für SSSP bekannt.

Algorithmus von Djkstra

funktioniert nur für nichtnegative Kantenkosten

Kantenkosten sollten in Heap organisiert werden.

$S := \{s\}$ $D[s] := 0$;
für alle Knoten $v \neq s$: $D[v] := C(s, v)$;
while $V \setminus S \neq \emptyset$ **do**
  wähle den Knoten $w \in V \setminus S$ mit minimalem $D[w]$;
  $S := S \cup \{w\}$;
  for *each* $u \in V \setminus S$, *u adjazent zu w* **do**
    $D[u] := min(D[u], D[w] + C(w, u))$

Algorithm 6: Dijkstra

Korrektheitsbeweis durch vollständige Induktion über die Anzahl der Schleifendurchläufe.

Laufzeit O((m+n)log n)

für allgemeine Graphen: $O(n^2 \log n)$
für Graphen mit konstanter Nachbarzahl: O(n log n)

Referenzen zum Nacharbeiten und Vertiefen:

Skript Alt 4.4.1 (S. 79-81),
Cormen, Kap. 24.3

APSP: All Pairs Shortest Path

Finde die kürzesten Wege zwischen allen Knotenpaaren

Triviale Variante:

Wende Dijkstra für alle Knoten als Quellen an

Laufzeit $O(n(m+n)\log n)$

für allgemeine Graphen: $O(n^3 \log n)$
für Graphen mit konstanter Nachbarzahl: $O(n^2 \log n)$

Algorithmus von Floyd-Warshall:

Sei $V = \{1,...n\}$.
$d_{ij}^{(k)}$ ist die Länge des kürzesten Wegs zwischen i und j, der als echte Zwischenknoten nur Knoten aus $\{1,...k\}$ benutzt.

Laufzeit $O(n^3)$

for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, n$ **do**
    $d_{ij}^{(0)} = \begin{cases} c(i,j): & \text{falls } (i,j) \in E \\ \infty: & \text{sonst} \end{cases}$
  end for
end for
for $k = 1, \ldots, n$ **do**
  for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, n$ **do**
      $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
    end for
  end for
end for

Referenzen zum Nacharbeiten und Vertiefen:

Skript Alt 4.4.2, 4.4.3 (S. 81-83),
Cormen, Kap. 25.2

Algorithmik 7

APSP: All Pairs Shortest Path

Finde die kürzesten Wege zwischen allen Knotenpaaren

Zusammenhang zur Matrixmultiplikation:

Sei $V = \{1,\ldots n\}$.
$d_{ij}^{(k)}$ ist die Länge des kürzesten Wegs zwischen i und j, der maximal k Kanten benutzt
Achtung: Diese Definition ist anders als bei Floyd-Warshall!

Satz: Sei A die Adjazenzmatrix.
Definiere die Operation min als Addition, die Operation + als Multiplikation und behandle das Element ∞ als Null bei der Operation +

Dann enthält das Matrixprodukt A^k an Position (i,j) das Element $d_{ij}^{(k)}$.
Insbesondere steht in A^{n-1} an Position (i,j) die Länge des kürzesten Weges von i nach j.

Quadratisches Potenzieren: Man kann A^{n-1} mit $O(\log n)$ Matrixmultiplikationen berechnen.

Algorithmus von Strassen: Man kann zwei nxn-Matrizen in Laufzeit $O(n^{\log 7})$ multiplizieren.

Folgerung für APSP: Laufzeit $O(n^{\log 7} \log n)$ <u>Anm.:</u> $\log 7 \approx 2{,}81$

Referenzen zum Nacharbeiten und Vertiefen:

Cormen, Kap. 25.1 (Zusammenhang zur Matrixmultiplikation), Kap. 28.2 (Strassen)