7.4 Kürzeste Pfade in einem Graphen

Ein weiteres klassisches Problem, das basierend auf dynamischer Programmierung effizient gelöst werden kann, ist die Suche nach einem kürzesten Pfad in einem gewichteten Graphen. Wir betrachten hierzu den von Edsger W. Dijkstra im Jahre 1959 veröffentlichten Algorithmus, der unter der Bezeichnung *Dijkstras Algorithmus* sehr bekannt ist.

Gegeben sei ein gerichteter Graph $G = (V, E)$ mit Bogenlängen $d_{u,v} > 0, \forall (u, v) \in E$, sowie ein Startknoten $s \in V$ (*source*) und ein Zielknoten $t \in V$ (*target*). Gesucht ist der kürzeste Pfad $P \subseteq E$ von Knoten s zu Knoten $t \in V$.

Die Länge eines Pfades P ist hierbei

$$d(P) = \sum_{(u,v)\in P} d_{u,v}.$$

Sei $dist[v]$ die Länge des bisher kürzesten gefundenen Pfades von s zu Knoten v, $\forall v \in V$. Dijkstras Algorithmus basiert auf folgendem rekursiven Zusammenhang:

$$dist[v] = \begin{cases} 0 & \text{für } v = s \\ \min_{\forall u|(u,v)\in E} \left(dist[u] + d_{u,v}\right) & \forall v \in V \setminus \{s\} \end{cases}$$

Die Grundidee ist, in einer Datenstruktur Q alle Knoten zu speichern, für die die kürzesten Pfade von s aus noch nicht bekannt sind. Anfangs sind das alle Knoten in V. In $pred[v]$ wird für jeden Knoten $v \in V$ ein Vorgänger (*predecessor*) gespeichert, um damit die kürzesten bisher gefundenen Pfade zu jedem Knoten zu repräsentieren. Nun wird iterativ jeweils ein Knoten v aus Q herausgenommen, der von s aus mit der kleinsten bisher bekannten Pfadlänge erreichbar ist. Anfangs ist $v = s$. Dieser Knoten v wird *besucht* und seine ausgehenden Kanten werden in Betracht gezogen um für die Nachbarknoten $N^+(v)$ gegebenenfalls kürzere als bisher bekannte Pfade (bzw. anfangs überhaupt gültige Pfade) zu finden. Wird irgendwann t besucht, dann terminiert das Verfahren. Algorithmus 49 zeigt Dijkstras Algorithmus in konkretem Pseudocode.

Abbildung 7.7 zeigt ein Beispiel, in dem der kürzeste Weg von H nach B in einem ungerichteten Graphen gesucht wird. Die Kantenbeschriftungen geben die Kantenlängen $d_{u,v}$ an, besuchte Knoten werden eingefärbt und die in Klammern stehenden Werte bei den Knoten repräsentieren die Längen $dist[v]$ der bisher bekannten kürzesten Pfade von s nach v.

Korrektheit

Die Korrektheit von Dijkstras Algorithmus ergibt sich aus der obigen Rekursionsgleichung und der Tatsache, dass immer ein noch nicht besuchter Knoten als nächster besucht wird, der auf kürzestem Weg erreicht werden kann. Der Pfad zu diesem Knoten muss daher bereits der tatsächlich kürzeste Pfad sein.

Algorithmus 49 Dijkstra$(G = (V, E), s, t)$

```
1: für alle Knoten v ∈ V {
2:    dist[v] = ∞; // Distanz des kürzesten Kantenzuges von s nach v
3:    pred[v] = undef; // Vorgänger am kürzesten Pfad von s nach v
4: }
5: dist[s] = 0;
6: Q = V;
7: solange Q nicht leer {
8:    Finde u ∈ Q mit minimalem dist[u];
9:    Q = Q \ {u};
10:   falls dist[u] = ∞ dann {
11:      Abbruch: Es existiert kein Pfad von s nach t;
12:   }
13:   falls u = t dann {
         // Kürzester Pfad gefunden; gib Distanz und Pfad (rückwärts) aus;
14:      Ausgabe: dist[t];
15:      solange pred[u] ≠ undef {
16:         Ausgabe: u;
17:         u = pred[u];
18:      }
19:      Abbruch;
20:   }
21:   für alle Nachbarknoten v ∈ N+(u) {
22:      distalt = dist[u] + d_{u,v};
23:      falls dist[v] > distalt dann {
24:         dist[v] = distalt;
25:         pred[v] = u;
26:      }
27:   }
28: }
```

Laufzeit

Die Laufzeitkomplexität des Algorithmus hängt maßgeblich von der verwendeten Datenstruktur Q ab, in der die noch nicht besuchten Knoten abgespeichert werden. Da jeweils ein Knoten mit minimaler Entfernung von s entnommen wird, bietet sich hierfür ein *Heap* an.

Die Initialisierung benötigt dann $\Theta(|V|)$ Zeit. Im Hauptteil sind dann bis zu $O(|V|)$ Schleifendurchläufe mit Laufzeit $O(|V|)$ erforderlich. Die Ausgabe des Ergebnisses erfordert abschließend ebenfalls $O(|V|)$ Zeit. Somit ist der Gesamtaufwand mit $O(|V|^2)$ beschränkt.

In dichten Graphen ist dieser Aufwand auch tatsächlich asymptotisch optimal, da im Worst-Case jede Kante zumindest einmal betrachtet werden muss.

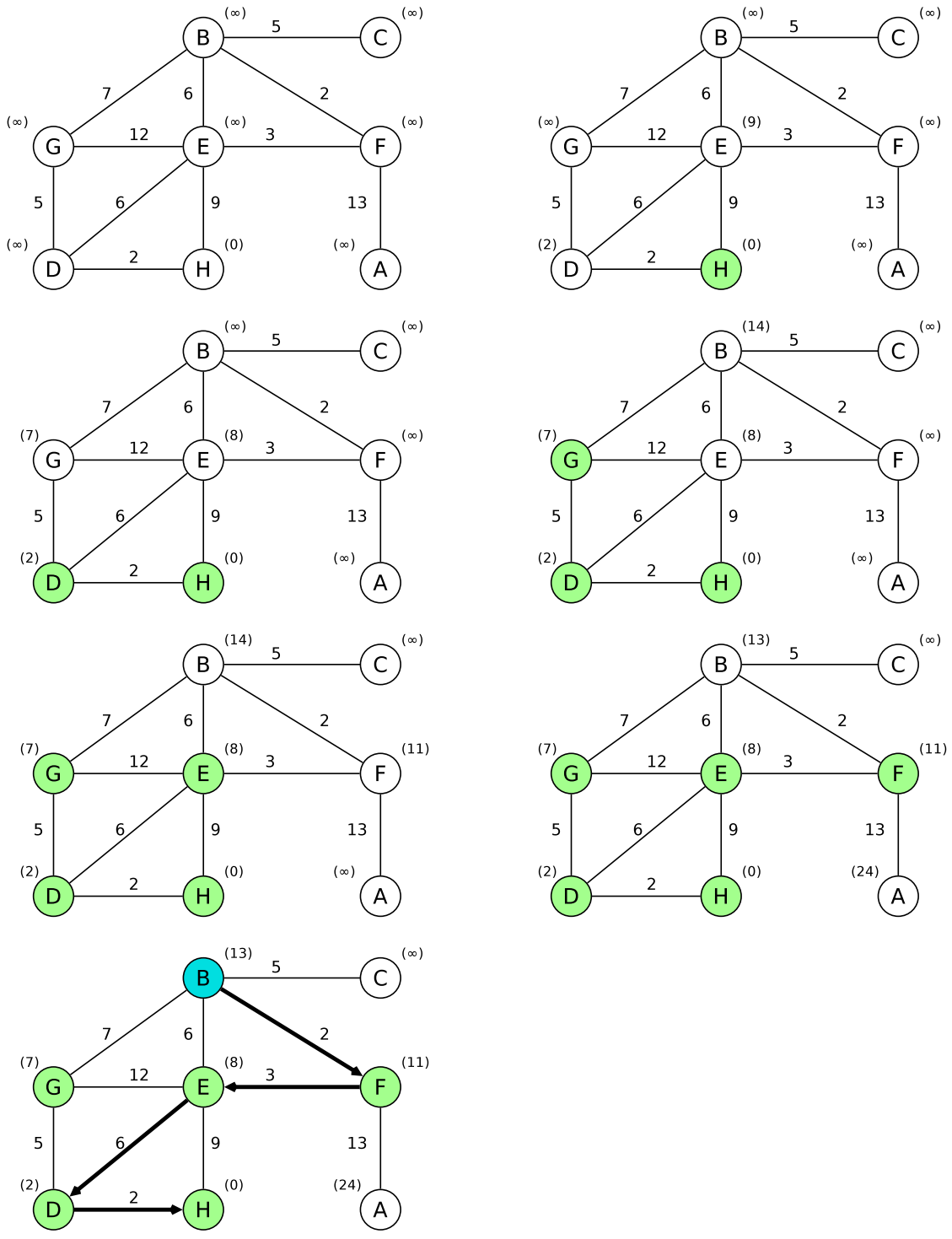

Abbildung 7.7: Beispiel zur Suche des kürzesten Weges von H nach B mit Dijkstras Algorithmus in einem ungerichteten Graphen.

Für dünne Graphen, wie sie im Speziellen in realen Routenplanungsproblemen auftreten, gibt es allerdings verschiedene Verbesserungsmöglichkeiten bzw. effizientere Alternativen. Beispielsweise macht es dann Sinn, in Q nur jene Knoten zu speichern, zu denen bereits irgendein Pfad bekannt ist (d.h. den „Rand“ des bereits „gelösten“ Teils des Graphen). Darüber hinaus bringen erweiterte Datenstrukturen wie der Fibonacci-Heap ebenfalls Vorteile. Das Finden eines kürzesten Pfades bzw. der kürzesten Pfade von einem zu allen anderen Knoten kann dann mit einer Laufzeitkomplexität von $O(|E| + |V| \log |V|)$ gelöst werden.

Abschließend sei nochmals betont, dass wir hier von nicht-negativen Distanzen ausgehen. Ist $d_{u,v} < 0$ möglich, dann würde Dijkstras Algorithmus i.A. keine korrekten Ergebnisse mehr liefern. Kürzeste Pfade können dann auch Schleifen enthalten, d.h., „Umwege“ können zu besseren Lösungen führen.

Weiterführende Literatur

- C.H. Papadimitriou und K. Steiglitz: „Combinatorial Optimization: Algorithms and Complexity“, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982
- W.J. Cook, W.H. Cunningham, W.R. Pulleyblank und A. Schrijver: „Combinatorial Optimization“, John Wiley & Sons, Chichester, 1998