Kürzeste Pfade

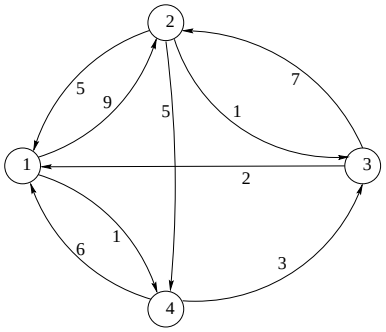
Wir beginnen mit einem klassischen Algorithmus zur kürzesten Pfadbestimmung. Sei $\mathcal{G}_n$ ein gerichteter Graph mit Knoten $1, 2, \ldots, n$ und bewerteten Kanten $d_{ij}, i, j \in \{1, 2, , \ldots, n\}$. Hierbei sei

$$0 \leq d_{ij} = \begin{cases} \infty, \text{ falls keine Kante von } i \text{ nach } j \text{ existiert,} \\ \text{Länge der Kante von } i \text{ nach } j. \end{cases}$$

Die Kantenlängen werden in der $n \times n$ Matrix $D = (d_{ij})$ zusammengefaßt, wobei wir für ∞ eine sehr große Zahl wählen.
Wir führen folgende Bezeichnungen ein:

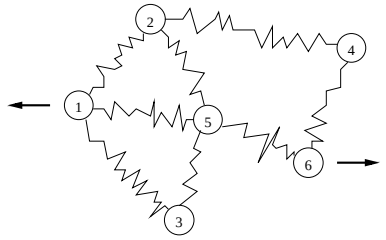
$\mathcal{W}^k(i,j)$	kürzester Weg von i nach j, mit Knoten aus $\{1, 2, \ldots, k\}$,
d_{ij}^k	Länge von $\mathcal{W}^k(i,j)$
r_{ij}^k	direkter Vorgänger von j in $\mathcal{W}^k(i,j)$.

Beispiel. Wir betrachten



$$D = \begin{pmatrix} \infty & 9 & \infty & 1 \\ 5 & \infty & 1 & 5 \\ 2 & 7 & \infty & \infty \\ 6 & \infty & 3 & \infty \end{pmatrix}$$

Analog-Methode (Vogel, 1980). Recht verblüffend ist folgende Überlegung, die zu einem Analog-Verfahren führt.

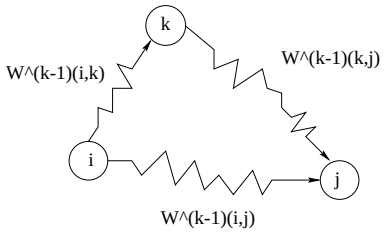


Man realisiert das Netzwerkdurch ein Fadenmodell, bei dem die Kantenlängen d_{ij} betragen. Um den kürzesten Weg von i nach j zu finden, ziehe man die beide Knoten soweit wie möglich auseinander.

Kaskadenmethode (S. Warshall) Kürzeste Wege zu jedem Knotenpaar i und j können mit folgenden Überlegungen[1] leicht berechnet werden.
Für $k = 0$ setzen wir

$$d_{ij}^0 = d_{ij}, \;\; r_{ij}^0 = i, \;\; i, j = 1, 2, \ldots, n.$$

$\mathcal{W}^0(i, j)$ ist somit der kürzeste Weg von i nach j, wenn nur die Knoten i und j berücksichtigt werden. In r_{ij}^k wird der Knoten angelegt, der auf dem Weg $\mathcal{W}^0(i, j)$ direkt vor dem Knoten j liegt.



Man kann jetzt induktiv den kürzesten Weg $\mathcal{W}^k(i, j)$ aus den Daten zu $\mathcal{W}^{k-1}(i, j)$ gewinnen. Die Länge aller kürzesten Wege d_{ij}^{k-1} ist bekannt. Wenn man die Länge von $\mathcal{W}^{k-1}(i, j)$ mit der Summe der Weglängen von $\mathcal{W}^{k-1}(i, k)$ und $\mathcal{W}^{k-1}(k, j)$ vergleicht, kann man d_{ij}^k und r_{ij}^k bestimmen.

Wir haben somit die Iteration:

$k = 1, 2, \ldots, n$

$$i, j = 1, 2, \ldots, k-1, k+1, \ldots, n,$$

$$d_{ij}^k = \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\},$$
$$r_{ij}^k = r_{kj}^{k-1}, \text{ falls } d_{ij}^k \neq d_{ij}^{k-1},$$

Aus der Matrix $R^n = (r_{ij}^N)_{ij}$ können wir die Knoten von $\mathcal{W}^n(i, j)$ für festes i und j rückwärts berechnen.

$$s_1 = j, \; s_2 = r_{i,s_1}^k, \; s_3 = r_{i,s_2}^k, \; \ldots, \; s_t = r_{i,s_{t-1}} = i.$$

Jetzt durchläuft $\mathcal{W}^k(i, j)$ die Knoten

$$s_\nu, \;\; \nu = t, t-1, \ldots 1.$$

Dieser Algorithmus wurde von R. W. Floyd[2] als 11-zeilige Algol-Programm angegeben, allerdings ohne die Berechnung der Knoten, die zu den kürzesten Pfaden gehören.

[1] S. Warshall: A theorem on Boolean matrices, J. ACM **9**, 1962, 11 – 12.
[2] R.W. Floyd: Algorithm 97, Comm. ACM **5**, 1962, 345.

Listing 1: Kürzeste Pfade (floyd.c)

```
#include <stdio.h>

void write_matrix(int);
void read_matrix(int);

int d[20][20];

int main(void)
{
int r[20][20];
int i, j, k, newlen, z, n;
int pfad[20];
(void) scanf("%i", &n);
read_matrix(n);
write_matrix(n);
for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++)
                r[i][j] = i;
}
for (k = 1; k <= n; k++) {
        for (i = 1; i <= n; i++) {
                if (i == k)
                        continue;
                for (j = 1; j <= n; j++) {
                        if (j == k)
                                continue;
                        newlen = d[i][k] + d[k][j];
                        if (d[i][j] >= newlen) {
                                d[i][j] = newlen;
                                r[i][j] = r[k][j];
                        }
                }
        }
}
```

```
printf("-------------------------------------\n");
printf(" von  nach   Laenge              Knoten\n");
printf("-------------------------------------\n");
for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
                printf("   %2i   %2i  ", i, j);
                printf("%7i        ", d[i][j]);
                pfad[1] = j;
                for (k = 1; k <= n; k++) {
                        pfad[k + 1] = r[i][pfad[k]];
                        if (pfad[k + 1] == i) break;
                }
                for (z = k + 1; z >= 1; z--)
                        printf("%5i", pfad[z]);
                printf("\n");
        }
}
return (0);
}

void write_matrix(int n) {
        int i, j;
        for (i = 1; i <= n; i++) {
                for (j = 1; j <= n; j++)
                        printf("%10i ", d[i][j]);
                printf("\n");
        }
}

void read_matrix(int n) {
        int i, j;
        for (i = 1; i <= n; i++) {
                for (j = 1; j <= n; j++)
                        (void) scanf("%i", &d[i][j]);
        }
}
```