UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Nina Slivnik

Uporaba polkolobarjev z zaprtjem

Delo diplomskega seminarja

Mentor: doc. dr. Matija Pretnar
Somentorica: izr. prof. dr. Polona Oblak

Ljubljana, 2019

Kazalo

1. Uvod in definicije 4
2. Polkolobar matrik 6
3. Algoritmi za računanje zaprtja 9
3.1. Jacobijev algoritem 9
3.2. Algoritem Warshall-Floyd-Kleene 11
3.3. Gaussova metoda 14
4. Primeri 16
4.1. Dosegljivost v grafu 17
4.2. Poti v grafu 18
4.3. Polinomi in potenčne vrste 22
4.4. 0-1 nahrbtnik s ponavljanjem 24
Slovar strokovnih izrazov 25
Literatura 25

Uporaba polkolobarjev z zaprtjem

Povzetek

Polkolobar je algebrska struktura, ki podpira seštevanje in množenje, med katerima veljajo določene relacije. Ko dodamo še unarno operacijo zaprtje, dobimo polkolobar z zaprtjem. V diplomskem delu si bomo ogledali, kako definiramo zaprtje na polkolobarju matrik, tri algoritme za izračun zaprtja matrike ter nekaj primerov uporabe polkolobarjev z zaprtjem v programskem jeziku Haskell: dosegljivost v grafu, poti v grafu, polinome in potenčne vrste ter 0-1 nahrbtnik s ponavljanjem.

Application of closed semirings

Abstract

Semiring is an algebraic structure where we have two binary operations ('addition' and 'multiplication') and relations between them. If we add a unary operation called closure we obtain a closed semiring. In this work we define a closed semiring of matrices, describe three algorithms for computing the closure and some applications of closed semirings in Haskell: reachability in graphs, paths in graphs, polynomials and power series and 0-1 knapsack with repetition.

Math. Subj. Class. (2010): 68W01, 06A15, 16Y60, 05C12

Ključne besede: Polkolobar, zaprtje, polkolobar matrik, Jacobijev algoritem, algoritem Warshall-Floyd-Kleene, Gaussova metoda, Haskell

Keywords: Semiring, closure, matrix semiring, Jacobi's algorithm, Warshall-Floyd-Kleene's algorithm, Gauss method, Haskell

Polkolobar je algebrska struktura z operacijama 'seštevanje' in 'množenje', katerih lastnosti so malce šibkejše od tistih v kolobarju (nima nasprotnih elementov). Če pa ji dodamo še unarno operacijo zaprtje, dobimo polkolobar z zaprtjem in tako lahko enostavno rešimo veliko problemov, ki jih je mogoče opisati s sistemom linearnih enačb.

Najprej si bomo ogledali definiciji polkolobarja in zaprtja ter nekaj preprostih primerov. V razdelku 2 bomo pokazali, da kvadratne matrike nad polkolobarjem z zaprtjem tudi same tvorijo polkolobar z zaprtjem. V razdelku 3 bomo videli tri algoritme za izračun zaprtja matrike, to so Jacobijev, Warshall-Floyd-Kleene in Gaussov. V zadnjem razdelku, torej razdelku 4, pa bomo videli nekaj bolj zanimivih primerov polkolobarjev z zaprtjem: dosegljivost v grafu, poti v grafu, polinome in potenčne vrste ter 0-1 nahrbtnik s ponavljanjem.

Nekaj primerov si bomo ogledali tudi v programskem jeziku Haskell.

Definicija 1.1. *Polkolobar* S je algebrska struktura z operacijama $\oplus$ in $\odot$ ter nevtralnima elementoma 0 in 1, za katero veljajo naslednje lastnosti:

$$\begin{aligned} a \oplus b &= b \oplus a \\ a \oplus (b \oplus c) &= (a \oplus b) \oplus c \\ a \oplus 0 &= a \\ a \odot (b \odot c) &= (a \odot b) \odot c \\ a \odot 0 = 0 \odot a &= 0 \\ a \odot 1 = 1 \odot a &= a \\ a \odot (b \oplus c) &= a \odot b \oplus a \odot c \\ (a \oplus b) \odot c &= a \odot c \oplus b \odot c \end{aligned}$$

za vse $a, b, c \in S$.

Opomba 1.2. Lehmann [3] lastnosti $a \odot 0 = 0 \odot a = 0$ ne privzema v svoji definiciji polkolobarja.

Najprej si oglejmo primer polkolobarja:

Primer 1.3. *Tropski polkolobar* $\mathbb{T}$ ali polkolobar *min-plus* je definiran na množici $\mathbb{R}_{\geq 0} \cup \{\infty\}$. Operacijo $\oplus$ definiramo kot min, $\odot$ pa kot $+$. Enota za min je ∞, 0 pa za $+$. Če to upoštevamo, ko prepišemo enakosti iz definicije 1.1, vidimo, da vse veljajo, torej smo res definirali polkolobar. ♢

V programskem jeziku Haskell implementiramo razred Semiring takole:

```
infixl 9 @.
infixl 8 @+
class Semiring r where
        zero, one :: r
        closure :: r -> r
        (@+), (@.) :: r -> r -> r
```

Definicija 1.4. *Zaprtje* a^* elementa a je enočlena operacija, ki zadošča enakostim

$$a^* = 1 \oplus a \odot a^* = 1 \oplus a^* \odot a.$$

Opomba 1.5. Kjer ne bo dvoumnosti, bomo v nadaljevanju namesto $\oplus$ pisali kar $+$, namesto $\odot$ pa $\cdot$. Če ne bo šlo za navadno seštevanje in množenje, bomo to poudarili.

Poglejmo si še dva enostavna primera polkolobarja z zaprtjem:

Primer 1.6. Imejmo razširjena naravna števila $\mathbb{N} \cup \{\infty\}$ z običajnim razširjenim seštevanjem in množenjem, torej za $a \neq 0$ dodamo še $\infty + a = \infty$, $\infty \cdot 0 = 0$ in $\infty \cdot a = \infty$ (to sta razširjeno seštevanje in množenje), tako dobimo polkolobar. Zaprtje tu definiramo kot $0^* = 1$ in $a^* = \infty$ za $a \geq 1$. Ponovno se lahko o pravilnosti prepričamo z uporabo enakosti iz definicij 1.1 in 1.4. ♢

Primer 1.7. Vzemimo množico logičnih vrednosti $\{\top, \bot\}$. Za $\oplus$ vzamemo disjunkcijo $\vee$, za $\odot$ konjunkcijo $\wedge$, nato pa definiramo še enoti $0 = \bot$ in $1 = \top$. Z lahkoto se prepričamo, da operaciji zadostita vsem enakostim iz 1.1. Opazimo še, da lahko definiramo tudi operacijo zaprtja s predpisom $x^* = \top$, ki ustreza 1.4. Torej logične vrednosti $\top$ in $\bot$ z operacijama $\vee$ in $\wedge$ tvorijo polkolobar z zaprtjem.

To napišimo v Haskellu:

```
instance Semiring Bool where
        zero = False
        one = True
        closure x = True
        (@+) = (||)
        (@.) = (&&)
```

♢

Primer 1.8. V primeru 1.3 smo si ogledali tropski polkolobar $\mathbb{T}$, ki je definiran na $\mathbb{R}_{\geq 0} \cup \{\infty\}$, zdaj pa to množico razširimo na $\mathbb{R} \cup \{\infty\}$. Operaciji ostaneta min in $+$. V tem polkolobarju niso vsi elementi zaprti; zaprti so samo tisti, ki so bili zaprti že v primeru 1.3, za negativna števila pa v tem tropskem polkolobarju zaprtja ni. ♢

Omenimo še dva primera zaprtja:

- $a^* = 1 + a + a^2 + a^3 + \ldots$ in
- $a^* = (1 - a)^{-1}$, kjer je $1^* = \infty$,
- $a^* = \begin{cases} (1-a)^{-1} & , a \neq 1 \\ \infty & , a = 1 \end{cases}$

ki sta seveda definirana v polkolobarjih, kjer ima to smisel (v drugem primeru potrebujemo komutativnost množenja).

Trditev 1.9. *Če imamo linearno preslikavo $x \mapsto ax + b$ v nekem zaprtem polkolobarju, potem je $x = a^*b$ negibna točka te preslikave.*

Dokaz. Po definiciji je x negibna točka, če zanjo velja $f(x) = x$. Zato za x vstavimo kar a^*b in poračunamo. Ker je

$$f(a^*b) = a(a^*b) + b = (aa^*)b + b = (aa^* + 1)b = (1 + aa^*)b = a^*b,$$

je a^*b negibna točka preslikave $x \mapsto ax + b$. □

2. Polkolobar matrik

Z uporabo matrik si lahko pogosto poenostavimo računanje – to bomo videli v razdelku 4, ko si bomo ogledali primere uporabe polkolobarjev z zaprtjem. Da pa jih bomo lahko uporabili, moramo najprej pokazati, da kvadratne matrike velikosti $n \times n$ nad polkolobarjem z zaprtjem S tudi same tvorijo polkolobar z zaprtjem.

Naj bosta A in B matriki nad polkolobarjem z zaprtjem S:

$$A = [a_{ij}]_{i,j\in[1:n]}, B = [b_{ij}]_{i,j\in[1:n]}$$

Opomba 2.1. Tu smo vpeljali novo oznako: $i \in [1:n]$ je krajše za $i \in \{1, 2, ..., n\}$, $i, j \in [1:n]$ pa krajše za $(i, j) \in [1:n] \times [1:n]$.

Seštevanje in množenje matrik definiramo običajno:

$$A \oplus B = [a_{ij} \oplus b_{ij}]_{i,j\in[1:n]}$$
$$A \odot B = \left[\bigoplus_{k\in[1:n]} a_{ik} \odot b_{kj}\right]_{i,j\in[1:n]}$$

kjer je oznaka $\bigoplus$ zamenjava za $\sum$:

$$\bigoplus_{k\in[1:n]} a_k = a_1 \oplus ... \oplus a_n$$

Operaciji seštevanja in množenja imamo sedaj definirani, potrebujemo še enoti

- za seštevanje: $0_n = [c_{ij}]_{i,j\in[1:n]}$, kjer je $c_{ij} = 0$ za $i, j \in [1:n]$ in
- za množenje: $I_n = [\delta_{ij}]_{i,j\in[1:n]}$, kjer je $\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$.

Preprosto je preveriti, da vse lastnosti iz definicije 1.1 držijo.

Če želimo ta problem predstaviti v programskem jeziku Haskell, najprej definiramo seštevanje in množenje:

```
data Matrix a = Scalar a | Matrix [[a]]
type BlockMatrix a = (Matrix a, Matrix a, Matrix a, Matrix a)

mjoin :: BlockMatrix a -> Matrix a
mjoin (Matrix a, Matrix b, Matrix c, Matrix d) =
        Matrix ((a `hcat` b) ++ (c `hcat` d))
        where hcat = zipWith (++)

msplit :: Matrix a -> BlockMatrix a
msplit (Matrix (row:rows)) =
        (Matrix [[first]], Matrix[top], Matrix left, Matrix rest)
        where
                (first:top) = row
                (left, rest) = unzip (map (\(x:xs) -> ([x],xs))rows)

instance Semiring a => Semiring (Matrix a)
        where
                zero = Scalar zero
                one = Scalar one

        --addition
```

```
Scalar a @+ Scalar b = Scalar (a @+ b)

Matrix a @+ Matrix b = Matrix (zipWith (zipWith (@+)) a b)

Scalar s @+ m = m @+ Scalar s
Matrix [[a]] @+ Scalar b = Matrix [[a @+ b]]

m @+ s = mjoin (first @+ s, top, left, rest @+ s)
        where (first, top, left, rest) = msplit m

--multiplication
Scalar a @. Scalar b = Scalar (a @. b)
Scalar a @. Matrix b = Matrix (map (map (a @.)) b)
Matrix a @. Scalar b = Matrix (map (map (@. b)) a)
Matrix a @. Matrix b =
        Matrix [[foldl1 (@+) (zipWith (@.) row col)
        | col <- cols] | row <- a]
        where cols = transpose b
```

Definirajmo sedaj še zaprtje:

Definicija 2.2. Zaprtje matrike definiramo induktivno glede na njeno velikost z bločno delitvijo matrike na štiri podmatrike.

Definicija zaprtja za matrike velikosti $n \times n$:

- $n = 1 : [a]^* = [a^*]$
- $n > 1$: Matriko A predstavimo kot bločno matriko

$$A = \begin{bmatrix} C & D \\ E & F \end{bmatrix},$$

kjer velja, da so bloki naslednjih velikosti: $C^{k\times k}$, $D^{k\times(n-k)}$, $E^{(n-k)\times k}$ in $F^{(n-k)\times(n-k)}$, kjer je $0 < k < n$. Tedaj zaprtje matrike A predpišemo kot

$$A^* = \begin{bmatrix} C^* + C^*D\Delta^*EC^* & C^*D\Delta^* \\ \Delta^*EC^* & \Delta^* \end{bmatrix}$$

pri čemer je $\Delta = F + EC^*D$.

Opomba 2.3. Pri razbitju matrike na bločne matrike velikosti blokov niso pomembne, saj lahko dano matriko razbijemo na dva različna načina, rezultat pa je isti (Lehmann [3]).

Pokazati pa moramo še, da velja naslednji izrek:

Izrek 2.4. *Če je A matrika velikosti $n \times n$, potem velja:*

$$A^* = I_n + A \cdot A^* = I_n + A^* \cdot A,$$

kjer je I_n identiteta velikosti $n \times n$.

Dokaz. Dokazali bomo samo $A^* = I_n + A \cdot A^*$, saj je druga enakost simetrična, uporabili pa bomo indukcijo na velikost matrike.

- $n = 1 : a^* = 1 + a \cdot a^*$ po definiciji.

- $n > 1$: Naj bo

$$A = \begin{bmatrix} C & D \\ E & F \end{bmatrix}, \text{ kjer je } C^{k\times k},$$

indukcijska predpostavka pa

$$C^* = I_k + CC^* \text{ in } \Delta^* = I_{n-k} + \Delta\Delta^*,$$

kjer je $\Delta = F + EC^*D$, kot v definiciji zaprtja za matrike. Nato naprej po definiciji velja

$$A^* = \begin{bmatrix} C^* + C^*D\Delta^*EC^* & C^*D\Delta^* \\ \Delta^*EC^* & \Delta^* \end{bmatrix}$$

iz česar sledi

$$A \cdot A^* = \begin{bmatrix} CC^* + CC^*D\Delta^*EC^* + D\Delta^*EC^* & CC^*D\Delta^* + D\Delta^* \\ EC^* + EC^*D\Delta^*EC^* + F\Delta^*EC^* & EC^*D\Delta^* + F\Delta^* \end{bmatrix}$$

Bloke v tej matriki lahko poenostavimo z uporabo indukcijske predpostavke:

(1) $CC^* + CC^*D\Delta^*EC^* + D\Delta^*EC^* = CC^* + D\Delta^*EC^* + CC^*D\Delta^*EC^* = CC^* + (I_k + CC^*)D\Delta^*EC^* = CC^* + C^*D\Delta^*EC^*$

(2) $CC^*D\Delta^* + D\Delta^* = D\Delta^* + CC^*D\Delta^* = (I_k + CC^*)D\Delta^* = C^*D\Delta^*$

(3) $EC^* + EC^*D\Delta^*EC^* + F\Delta^*EC^* = EC^* + F\Delta^*EC^* + EC^*D\Delta^*EC^* = EC^* + (F + EC^*D)\Delta^*EC^* = EC^* + \Delta\Delta^*EC^* = (I_{n-k} + \Delta\Delta^*)EC^* = \Delta^*EC^*$

(4) $EC^*D\Delta^* + F\Delta^* = F\Delta^* + EC^*D\Delta^* = (F + EC^*D)\Delta^* = \Delta\Delta^*$

Sledi

$$I_n + A \cdot A^* = \begin{bmatrix} I_k + CC^* + C^*D\Delta^*EC^* & C^*D\Delta^* \\ \Delta^*EC^* & I_{n-k} + \Delta\Delta^* \end{bmatrix}$$

kar pa je po indukcijski predpostavki enako A^*, ko poenostavimo diagonalna člena:

(1) $(I_k + CC^*) + C^*D\Delta^*EC^* = C^* + C^*D\Delta^*EC^*$

(2) $I_{n-k} + \Delta\Delta^* = \Delta^*$

□

Iz izreka direktno sledi:

Posledica 2.5. *Za vsaki matriki A in B velikosti $n \times n$ velja:*

(1) $AA^* = A + AA^*A = A^*A$,

(2) $A^* = I_n + A + AA^*A$,

(3) $B + AA^*B = A^*B$ *in* $B + BA^*A = BA^*$.

Dokaz.

(1) Samo z uporabo izreka: $AA^* = A(I_n + A^*A) = A + AA^*A = (I_n + AA^*)A = A^*A$.

(2) Najprej po izreku, nato iz točke (1): $A^* = I_n + AA^* = I_n + A + AA^*A$.

(3) $B + AA^*B = (I_n + AA^*)B = A^*B$ ter simetrično tudi $B + BA^*A = BA^*$.

□

Posledica 2.6. *Za kvadratno matriko A nad polkolobarjem z zaprtjem S velja:*

$$A^* = I_n + A + A^2 + \dots$$

Dokaz. Uporabimo točki (1) in (2) iz posledice 2.5:

$$A^* = I_n + A + AA^*A = I_n + A + A(A + AA^*A) = I_n + A + A^2 + A^2A^*A = \dots$$

□

Zaprtje po definiciji 2.2 v Haskellu:

```
--closure
closure (Matrix [[x]]) = Matrix [[closure x]]
closure m =
        mjoin (first' @+ top' @. rest' @. left',
        top' @. rest', rest' @. left', rest')
        where
                (first, top, left, rest) = msplit m
                first' = closure first
                top' = first' @. top
                left' = left @. first'
                rest' = closure (rest @+ left' @. top)
```

Časovno zahtevnost tega algoritma za zaprtje izračunamo preprosto: Vemo, da množenje dveh matrik velikosti $p \times q$ in $q \times r$ porabi $\mathcal{O}(pqr)$ operacij polkolobarja S. Naša funkcija zaprtja pri kvadratnih $n \times n$ matrikah porabi $\mathcal{O}(n^2)$ operacij matričnega množenja (množenje $1 \times n$ in $n \times n$), zopet $\mathcal{O}(n^2)$ za matrično seštevanje in `msplit` ter en rekurziven klic na matrikah velikosti $(n-1) \times (n-1)$.

3. Algoritmi za računanje zaprtja

V nadaljevanju si bomo pogledali tri algoritme za iskanje zaprtja matrik. Algoritme uporabimo, ker delujejo hitreje kot izračun po definiciji. Prvi je Jacobijev (razdelek 3.1), ki je najenostavnejši, a ni zelo hiter, drugi uporabi Floyd-Warshallov algoritem za računanje najkrajših poti v grafu (to bo algoritem Warshall-Floyd-Kleene v razdelku 3.2), tretji pa polkolobarju prirejeno Gaussovo eliminacijo (Gaussov algoritem v razdelku 3.3).

3.1. **Jacobijev algoritem.**

Jacobijev algoritem je iterativen algoritem za iskanje zaprtja matrike nad polkolobarjem z zaprtjem S, ki deluje tako, da izračunamo vsak stolpec (ali vrstico) matrike zaprtja posebej. Je najenostavnejši, vendar pa tudi najpočasnejši od tukaj naštetih, računa pa kar po definiciji 1.4 in z uporabo izreka 2.4.

Oglejmo si ga na primeru tropskega polkolobarja $\mathbb{T}$:

Naj bo A matrika, katere zaprtje iščemo, A^* matrika zaprtja, a_i pa i-ti stolpec matrike A. Izračunajmo i-ti stolpec A^*. Za začetni približek vzamemo vektor enot polkolobarja, kjer je na i-tem mestu enota za $\odot$, v našem primeru je to 0, na vseh

ostalih mestih pa enota za $\oplus$, to je ∞:

$$a_i^{(0)} = \begin{bmatrix} 0_S \\ \vdots \\ 0_S \\ 1_S \\ 0_S \\ \vdots \\ 0_S \end{bmatrix} \overset{\mathbb{T}}{=} \begin{bmatrix} \infty \\ \vdots \\ \infty \\ 0 \\ \infty \\ \vdots \\ \infty \end{bmatrix}$$

Za izračun $(k+1)$-tega približka i-tega stoplca uporabimo naslednjo formulo:

$$a_i^{(k+1)} = a_i^{(0)} \oplus A \odot a_i^{(k)} = \min\{a_i^{(0)}, A \odot a_i^{(k)}\},$$

kjer je min dveh vektorjev definiran po komponentah.

Na koncu vsakega koraka imamo tri možnosti: če je nov približek enak prejšnjemu, je to kar rešitev, če nista enaka in še nismo prišli do n-tega, iteracijo nadaljujemo, v zadnjem primeru pa iteracijo končamo in ugotovimo, da zaporedje približkov ne bo konvergiralo:

- $a_i^{(k+1)} = a_i^{(k)}$: $a_i^{(k)}$ je stolpec v A^*
- $a_i^{(k+1)} \neq a_i^{(k)}$ in $k \leq n$: nadaljujemo iteracijo
- $a_i^{(k+1)} \neq a_i^{(k)}$ in $k = n$: $A^* \neq A^{(n)}$, končamo iteracijo

Opomba 3.1. Vpeljimo novo oznako: $A^{(k)}$ naj bo delna vsota:

$$A^{(k)} = I + A + A^2 + \cdots + A^k$$

Algoritem 1 Jacobijev algoritem

Vhod: $A = [a_i]_{i\in[1:n]}$, $a_i = [\tilde{a}_{ij}]_{j\in[1:n]}$, $\tilde{a}_{ij} \in S$, S polkolobar z zaprtjem
Izhod: $R = [r_i]_{i\in[1:n]}$, $r_i = [\tilde{r}_{ij}]_{j\in[1:n]}$

for $i \leftarrow 1$ **to** n **do**
  $a_i^{(0)} = [\underbrace{0, ..., 0}_{i-1}, 1, 0, ..., 0]^T$
  $k = 1$
  while $k < n$ **or** $(k > 1$ **and** $a^{(k-1)} \neq a^{(k-2)})$ **do**
    $a_i^{(k)} = a_i^{(0)} + A \cdot a_i^{(k-1)}$
    $k = k + 1$
  end while
  if $a_i^{(k)} = a_i^{(k-1)}$ **then**
    $r_i = a_i^{(k)}$
  else
    error: ne konvergira
  end if
end for

Časovna zahtevnost algoritma je $\mathcal{O}(n^3)$ časa za vsak stolpec, torej skupaj $\mathcal{O}(n^4)$ za izračun zaprtja $n \times n$ matrike.

Izrek 3.2. *Naj bo A kvadratna matrika nad polkolobarjem z zaprtjem S:*

(1) Če obstaja število k, za katero je $A^* = A^{(k)}$, potem Jacobijev algoritem v največ k iteracijah izračuna iskani stolpec v matriki A. V nasprotnem primeru, torej če tak k ne obstaja, algoritem v največ k iteracijah pokaže, da velja $A^* \neq A^{(k)}$.
(2) Če obstaja zaprtje A^* in je S polkolobar z zaprtjem, potem Jacobijev algoritem generira konvergentno zaporedje proti iskanemu stolpcu matrike A.

Dokaz. Izrek bomo dokazali za stolpec a_i matrike A.

(1) Predpostavljamo, da obstaja končna konvergenca. Imamo vektor $a_i^{(0)}$ in posledično tudi $a_i^{(1)} = (I \oplus A) \odot a_i^{(0)}$, $a_i^{(2)} = (I \oplus A \oplus A^2) \odot a_i^{(0)}$ in naprej z indukcijo na $k \in \mathbb{N}$:

$$a_i^{(k)} = (I \oplus A \oplus A^2 \oplus \cdots \oplus A^k) \odot a_i^{(0)} = A^{(k)} \odot a_i^{(0)},$$

kjer v zadnjem koraku uporabimo posledico 2.6.

Če obstaja $k \in \mathbb{N}$, za katerega velja $A^{(k)} = A^*$, potem dobimo

$$a_i^{(k)} = A^* \odot a_i^{(0)} = a_i^{(k+1)} = \cdots$$

Če sta torej $a_i^{(k)}$ in $a_i^{(k+1)}$ različna, je to zato, ker je $A^* \neq A^{(k)}$.

(2) Pokažimo sedaj še konvergenco v polkolobarju z zaprtjem.

Za vsak k torej velja $a_i^{(k)} = A^{(k)} \odot a_i^{(0)}$. Ker A^* obstaja, zaporedje $A^{(k)}$ konvergira proti A^*. Ker je zaradi lastnosti $\oplus$ in $\odot$ konvergenca mogoča, lahko sklepamo, da $a_i^{(k)}$ konvergira k $A^* \odot a_i^{(0)}$, torej k i-temu stolpcu matrike A^*.

□

3.2. **Algoritem Warshall-Floyd-Kleene.** Ta algoritem je direkten prevod Kleenejevega dokaza, da lahko vsak regularni jezik predstavimo z regularnim izrazom. Floydov algoritem za iskanje najcenejših poti v usmerjenem grafu je poseben primer zgornjega algoritma, če za vsak $a \in S$ vzamemo $a^* = 1$, Warshallov algoritem za zaprtje Boolovih matrik (matrik logičnih vrednosti) pa se omeji na $S = \{\top, \bot\}$. Algoritem računa po definiciji 2.2.

Opomba 3.3. V prejšnjem algoritmu smo stolpce, ki smo jih izračunali kot vmesne rezultate na poti do končnega zaprtja, označili kot $a^{(k)}$. Zdaj bomo z $A^{(k)}$ označevali matrike, ki so vmesni rezultati.

Algoritem Warshall-Floyd-Kleene (v nadaljevanju algoritem WFK), s katerim izračunamo zaprtje za matrike, je sledeč:

Algoritem 2 Algoritem WFK

Vhod: $A = [a_{ij}]_{i,j\in[1:n]}$, $a_{ij} \in S$, S polkolobar z zaprtjem
Izhod: R_{ij} za $i, j \in [1:n]$

1: **for each** i, j **in** $[1:n]$ **do**
2: $\quad A_{ij}^{(0)} \leftarrow A_{ij}$
3: **end for**
4: **for** $k \leftarrow 1$ **to** n **do**
5: $\quad$ **for each** i, j **in** $[1:n]$ **do**
6: $\quad\quad A_{ij}^{(k)} \leftarrow A_{ij}^{(k-1)} + A_{ik}^{(k-1)}(A_{kk}^{(k-1)})^* A_{kj}^{(k-1)}$
7: $\quad$ **end for**
8: **end for**
9: **for each** i, j **in** $[1:n]$ **do**
10: $\quad R_{ij} \leftarrow \delta_{ij} + A_{ij}^{(n)}$
11: **end for**

Opomba 3.4. Vpeljimo še nekaj novih oznak: za matriko C velikosti $n \times n$ definiramo $C_{[i,k][j,l]}$ kot podmatriko, ki vsebuje vrstice od vključno i do vključno k ter stolpce od vključno j do vključno l, pri čemer velja $1 \le i \le k \le n$ in $1 \le j \le l \le n$. Zaradi enostavnosti bomo uporabili tudi . namesto $[1:n]$ ter i namesto $[i, i]$. Tako je denimo $A_{.i}$ i-ti stolpec matrike A, A_{ij} pa element $A_{[i,j]}$.

Če ga predstavimo z matrikami, algoritem WFK izračuna zaporedje $n+1$ matrik velikosti $n \times n$: to so matrike $A^{(k)}$ za $0 \le k \le n$, ki so definirane sledeče:

$$A^{(0)} = A, \tag{1}$$

$$A^{(k)} = A^{(k-1)} + A_{.k}^{(k-1)} \cdot A_{kk}^{(k-1)*} \cdot A_{k.}^{(k-1)} \text{ za } 1 \le k \le n, \tag{2}$$

rezultat R pa je

$$R = I_n + A^{(n)} \tag{3}$$

Algoritem ima časovno zahtevnost $\mathcal{O}(n^3)$, saj v vrsticah 4-7 vidimo tri gnezdene zanke **for** (oz. eno **for** ter eno **for each** na kartezičnem produktu), vrstica 6 pa porabi konstanten čas zaradi enega seštevanja in dveh množenj.

Zanki **for**, ki sta uporabljeni v algoritmu, sta dveh tipov: običajna zanka **for**, kjer se iteracije izvršijo ena za drugo z naraščajočo vrednostjo indeksa in zanka **for each**, pri kateri vrstni red iteriranja ni pomemben, ker so posamezne iteracije med seboj neodvisne; tako je **for each** $i, j \in [1:n]$ okrajšava za **for each** $(i, j) \in [1:n]\times[1:n]$. Algoritem uporabi $n+1$ različnih matrik $A^{(k)} (0 \le k \le n)$ zaradi enostavnosti. Lahko bi napisali tudi algoritem, ki bi uporabil le eno tako matriko (potrebno je paziti le, da vhodi v matriki niso spremenjeni pred uporabo).

Dokazali bomo, da z algoritmom WFK lahko izračunamo zaprtje R matrike A, torej $R = A^*$, najprej pa pokažimo, da velja $A^{(n)} = A + AA^*A$.

Izrek 3.5. *Za vsak* $k \in [0:n]$ *velja*

$$A^{(k)} = A + A_{.[1:k]}(A_{[1:k][1:k]})^* A_{[1:k].}$$

kjer izpustimo $A_{.[1:0]}, A_{[1:0][1:0]}$ *in* $A_{[1:0].}$.

Opomba 3.6. Izrek seveda implicira tudi $A^{(n)} = A + AA^*A$, kar v resnici želimo pokazati.

Dokaz. Uporabimo indukcijo na k:

- $k = 0 : A^{(0)} = A$
- $k = l + 1$: Indukcijsko predpostavimo

$$A^{(l)} = A + A_{.[1:l]}(A_{[1:l][1:l]})^* A_{[1:l].} \tag{4}$$

kjer je $0 \leq l \leq n-1$, iz (2) pa dobimo matrično obliko algoritma WFK, pri čemer je $l = k - 1$:

$$A^{(k)} = A^{(l)} + A^{(l)}_{.k}(A^{(l)}_{kk})^* A^{(l)}_{k.} \tag{5}$$

Definiramo bločne matrike $B = A_{[1:l][1:l]}$, $P = A_{k[1:l]}$ in $Q = A_{[1:l]k}$, s shemo torej

$$A = \left[\begin{array}{c|c|} B & Q \\ \hline P & \\ \hline \end{array}\right]$$

Z uporabo indukcijske predpostavke (4) razpišemo naslednje člene:

$$A^{(l)}_{.k} = A_{.k} + A_{.[1:l]}B^*Q \tag{6}$$

$$A^{(l)}_{k.} = A_{k.} + PB^*A_{[1:l].} \tag{7}$$

$$A^{(l)}_{kk} = A_{kk} + PB^*Q \tag{8}$$

Definiramo še $\Delta = A^{(l)}_{kk} = A_{kk} + PB^*Q$.
Ko prepišemo (5) z uporabo enakosti (6), (7) in (8), dobimo

$$A^{(k)} = A^{(l)} + (A_{.k} + A_{.[1:l]}B^*Q)\Delta^*(A_{k.} + PB^*A_{[1:l].}),$$

nato pa uporabimo še indukcijsko predpostavko (4)

$$A^{(k)} = A + A_{.[1:l]}B^*A_{[1:l].} + (A_{.k} + A_{.[1:l]}B^*Q)\Delta^*(A_{k.} + PB^*A_{[1:l].}) \tag{9}$$

Pokažimo sedaj, da je

$$A_{.[1:k]}(A_{[1:k][1:k]})^* A_{[1:k].} = A_{.[1:l]}B^*A_{[1:l].} + \tag{10}$$

$$(A_{.k} + A_{.[1:l]}B^*Q)\Delta^*(A_{k.} + PB^*A_{[1:l].}) \tag{11}$$

Po definiciji zaprtja velja

$$(A_{[1:k][1:k]})^* = \begin{bmatrix} B & Q \\ P & A_{kk} \end{bmatrix}^* = \begin{bmatrix} B^* + B^*Q\Delta^*PB^* & B^*Q\Delta^* \\ \Delta^*PB^* & \Delta^* \end{bmatrix},$$

kar vstavimo naprej v

$$\begin{aligned} A_{.[1:k]}(A_{[1:k][1:k]})^* A_{[1:k].} = {} & A_{.[1:l]}(B^* + B^*Q\Delta^*PB^*)A_{[1:l].} + \\ & A_{.k}\Delta^*PB^*A_{[1:l].} + A_{.[1:l]}B^*Q\Delta^*A_{k.} + \\ & A_{.k}\Delta^*A_{k.} \end{aligned}$$

to pa enačimo z (10) in tako na koncu dobimo

$$A^{(k)} = A + A_{.[1:k]}(A_{[1:k][1:k]})^* A_{[1:k].}$$

kar smo želeli dokazati.

□

Posledica 3.7. *Matrika R iz (3) je enaka $R = A^*$.*

Dokaz. Z uporabo posledice 2.5 ter izreka 3.5 iz $R = I_n + A^{(n)} = I_n + A + AA^*A$ dobimo $R = A^*$. □

3.3. **Gaussova metoda.** Tretji algoritem za računanje zaprtja matrike v polkolobarju z zaprtjem S je Gaussov algoritem za računanje inverza za realne matrike (brez pivotiranja), torej direkten prevod Gaussove metode za iskanje inverza matrike.

Gaussov algoritem je tudi edini od tu naštetih, ki ni iterativen.

Algoritem 3 Gaussov algoritem

Vhod: $A = [a_{ij}]i, j \in [1:n]a_{ij} \in S$, S polkolobar z zaprtjem
Izhod: R_{ij} za $i, j \in [1:n]$

1: **for each** i, j **in** $[1:n]$ **do**
2: $\quad G_{ij}^{(0)} \leftarrow A_{ij}$
3: **end for**
4: **for** $k \leftarrow 1$ **to** n **do**
5: $\quad$ **for each** i, j **in** $[k:n] \times [1:n]$ **do**
6: $\quad\quad G_{ij}^{(k)} \leftarrow G_{ij}^{(k-1)} + G_{ik}^{(k-1)} \cdot (G_{kk}^{(k-1)})^* G_{kj}^{(k-1)}$
7: $\quad$ **end for**
8: **end for**
9: **for each** i, j **in** $[1:n]$ **do**
10: $\quad P_{ij} \leftarrow G_{ij}^{(i)}$
11: **end for**
12: **for** $i \leftarrow n-1$ **to** 1 **step** -1 **do**
13: $\quad$ **for each** j, k **in** $[1:n] \times [i+1:n]$ **do**
14: $\quad\quad P_{ij} \leftarrow P_{ij} + G_{ik}^{(i)} P_{kj}$
15: $\quad$ **end for**
16: **end for**
17: **for each** i, j **in** $[1:n]$ **do**
18: $\quad R_{ij} \leftarrow \delta_{ij} + P_{ij}$
19: **end for**

Opomba 3.8. Zgornja oblika algoritma je zelo prostorsko potratna, ampak jo lahko, kot v algoritmu WFK, spremenimo tako, da uporabimo le eno $n \times n$ matriko.

Glavna razlika med Gaussovo metodo in algoritmom WFK je v 5. vrstici, kjer i teče od k do n namesto od 1 do n ter da koraki nazaj (torej koraki za -1) pridejo na vrsto v 12. vrstici.

Prednost Gaussove metode je pri računanju na roke najbolj opazna, ko ima vhodna matrika veliko število elementov enakih 0, saj ničelni elementi v Gaussovi metodi ostanejo dlje kot v algoritmu WFK.

Pokažimo sedaj, da je izračun matrike R na koncu algoritma kar zaprtje vhodne matrike A, torej $R = A^*$. Oznake bodo enake kot v dokazu za algoritem WFK.

V prvih 8 vrsticah algoritem izračuna zaporedje $n+1$ matrik $G^{(0)}$, $G^{(1)}$, ..., $G^{(n)}$, da velja:

$$G^{(0)} = A^{(0)} = A, G^{(1)} = A^{(1)}$$
$$G^{(k)} = A^{(k)}_{[k:n].}; 1 \leq k \leq n$$

V vrsticah 9 do 11 nato izračunamo matriko $P^{(0)}$, za katero velja

$$P^{(0)}_{k.} = A^{(k)}_{k.}; 1 \leq k \leq n$$

oz.

$$P^{(0)} = \begin{bmatrix} A^{(1)}_{1.} \\ A^{(2)}_{2.} \\ \vdots \\ A^{(n)}_{n.} \end{bmatrix}$$

V vrsticah 12 do 16 algoritem izračuna zaporedje vrstičnih vektorjev $P^{(n)}$, ..., $P^{(1)}$, da velja:

$$P^{(n)} = P^{(0)}_{n.} = A^{(n)}_{n.}$$

in

$$P^{(k)} = P^{(0)}_{k.} + P^{(0)}_{k[k+1:n]} \begin{bmatrix} P^{(k+1)} \\ P^{(k+2)} \\ \vdots \\ P^{(n)} \end{bmatrix}$$

Na koncu pa izračunamo še $R = I_n + P$.

Časovna zahtevnost algoritma je $\mathcal{O}(n^3)$ (v vrsticah 4-8 in ponovno v 12-16 vidimo 3 gnezdene zanke **for** (oz. eno **for** ter eno **for each** na kartezičnem produktu), vrstici 6 in 14 pa porabita konstanten čas zaradi elementarnih računskih operacij).

Izrek 3.9. *Za vsak k, $1 \leq k \leq n$, velja $P^{(k)} = A^{(n)}_{k.}$.*

Dokaz. Uporabili bomo obratno indukcijo na k:

- Za $k = n$: $P^{(n)} = A^{(n)}_{n.}$
- Za $k = l - 1$: Indukcijska predpostavka je $P^{(l)} = A^{(n)}_{l.}$.
  Računamo

$$\begin{aligned} P^{(k)} &= P^{(0)}_{k.} + P^{(0)}_{k[k+1:n]} \begin{bmatrix} P^{(k+1)} \\ P^{(k+2)} \\ \vdots \\ P^{(n)} \end{bmatrix} \\ &= A^{(k)}_{k.} + A^{(k)}_{k[k+1:n]} \cdot A^{(n)}_{[k+1:n].} \end{aligned}$$

  Razbijmo sedaj matriko A na podmatrike, kjer je B podmatrika velikosti $k \times k$:

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$$

  oziroma bolj natančno $B = A_{[1:k][1:k]}$, $C = A_{[1:k][k+1:n]}$, $D = A_{[k+1:n][1:k]}$ in $E = A_{[k+1:n][k+1:n]}$.
  Iz izreka 3.5 vemo $A^{(k)} = A + A_{.[1:k]}(A_{[1:k][1:k]})^* A_{[1:k].}$ in $A^{(n)} = A + AA^*A$. Lahko pa uporabimo razbitje na podmatrike:

$$\begin{aligned} A^{(k)} &= \begin{bmatrix} B & C \\ D & E \end{bmatrix} + \begin{bmatrix} B \\ D \end{bmatrix} B^* \begin{bmatrix} B & C \end{bmatrix} = \\ &= \begin{bmatrix} B + BB^*B & C + BB^*C \\ D + DB^*B & \Delta \end{bmatrix} \end{aligned}$$

kjer je $\Delta = E + DB^*C$.
Iz posledice 2.5 ter izreka 3.5 vemo naslednje: $B + BB^*B = B^*B$, $A + AA^*A = A^*A$, $C + BB^*C = B^*C$ ter $D + DB^*B = DB^*$.
Nadaljujemo račun in dobimo

$$A^{(k)} = \begin{bmatrix} B + BB^*B & C + BB^*C \\ D + DB^*B & \Delta \end{bmatrix} = \begin{bmatrix} B^*B & B^*C \\ DB^* & \Delta \end{bmatrix}$$

in

$$A^{(n)} = A^*A = \begin{bmatrix} B^* + B^*C\Delta^*DB^* & B^*C\Delta^* \\ \Delta^*DB^* & \Delta^* \end{bmatrix} \begin{bmatrix} B & C \\ D & E \end{bmatrix},$$

ker je, kot smo videli v opombi 2.3, definicija zaprtja neodvisna od velikosti podmatrik.

Če člene prvega koraka zapišemo malo drugače, dobimo

$$\begin{aligned} A^{(k)}_{k.} &= \begin{bmatrix} P^*_{k.}B & P^*_{k.}C \end{bmatrix} \\ A^{(k)}_{k[k+1:n]} &= P^*_{k.}C \\ A^{(n)}_{[k+1:n].} &= \begin{bmatrix} \Delta^*DB^*B + \Delta^*D & \Delta^*DB^*C + \Delta^*E \end{bmatrix} \\ &= \begin{bmatrix} \Delta^*DB^* & \Delta^*\Delta \end{bmatrix} \end{aligned}$$

Nato jih spet združimo

$$\begin{aligned} P^{(k)} &= A^{(k)}_{k.} + A^{(k)}_{k[k+1:n]}A^{(n)}_{[k+1:n].} = \\ &= \begin{bmatrix} P^*_{k.}B + P^*_{k.}C\Delta^*DB^* & P^*_{k.}C + P^*_{k.}C\Delta^*\Delta \end{bmatrix} = \\ &= \begin{bmatrix} P^*_{k.}B + P^*_{k.}C\Delta^*DB^* & P^*_{k.}C\Delta^* \end{bmatrix} \end{aligned}$$

Na koncu pa vstavimo v $A^{(n)}_{k.}$:

$$A^{(n)}_{k.} = P^{(k)}$$

kar smo želeli dokazati.

□

Vidimo, da torej res drži

$$R = I_n + \begin{bmatrix} P^{(1)} \\ \vdots \\ P^{(n)} \end{bmatrix} = I_n + A^{(n)} = A^*$$

4. Primeri

Ena pomembnejših uporab matrik v linearni algebri je reševanje sistemov linearnih enačb. Ker lahko sami določimo polkolobar, lahko sami izberemo, kaj linearnost pomeni.

Recimo, da imamo sistem enačb v nekem polkolobarju z zaprtjem s spremenljivkami $\{x_1, ..., x_n\}$, kjer je vsak x_i definiran kot linearna kombinacija spremenljivk in konstante, torej

$$x_i = a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n + b_i$$

(a_{ij} in b_i so dani). Neznanke x_i shranimo v vektor x, koeficiente a_{ij} v kvadratno matriko M in konstante b_i v vektor b. Sistem enačb tako postane

$$x = Mx + b.$$

Tako je x fiksna točka preslikave $x \mapsto Mx + b$, rešitev pa je $x = M^*b$ (kot smo videli že v izreku 1.9, izračunamo pa z algoritmi zaprtja za matrike).

Opazimo, da smo prej računali s kvadratnimi $n \times n$ matrikami, zdaj pa sta x in b vektorja velikosti $n \times 1$. Ta problem lahko rešimo tako, da ju razširimo do kvadratnih matrik, sestavljenih iz samih istih stolpcev, torej

$$\widetilde{X} = [x, ..., x]$$

in enako za b:

$$\widetilde{B} = [b, ..., b].$$

Oglejmo si sedaj nekaj primerov.

4.1. **Dosegljivost v grafu.** Imamo enostaven usmerjen graf G z n vozlišči, $V = \{v_1, v_2, \dots v_n\}$. Tak graf lahko predstavimo z matriko sosednosti M, kjer velja $M_{ij} = 1 = \top$, če obstaja povezava od vozlišča j v vozlišče i, sicer $M_{ij} = 0 = \bot$, torej $S = \{\top, \bot\}$.

Zanima nas, katera vozlišča so med seboj dosegljiva. Kako bi ugotovili, ali je j-to vozlišče dosegljivo iz i-tega? Načinov je veliko, mi pa bomo do rešitve skušali priti z uporabo sistema linearnih enačb. Uvedimo naslednjo oznako:

$$x_{ij} \dots \text{obstaja pot med } v_i \text{ in } v_j.$$

Vrednost x_{ij} je torej lahko $\top$, če sta vozlišči povezani, in $\bot$, če nista.

Sedaj pa zgornje oznake uporabimo, da zapišemo sistem linearnih enačb:

$$\begin{aligned} x_{ij} &= (x_{i1} \wedge M_{1j}) \vee (x_{i2} \wedge M_{2j}) \vee \dots \vee (x_{in} \wedge M_{nj}) \vee (i = j) = \\ &= (M_{1j} \wedge x_{i1}) \vee (M_{2j} \wedge x_{i2}) \vee \dots \vee (M_{nj} \wedge x_{in}) \vee (i = j). \end{aligned}$$

To pomeni, da sta vozlišči v_i in v_j povezani, če obstaja pot od v_i do v_1 in nato povezava med v_1 in v_j, ali pa pot od v_i do v_2 in povezava med v_2 in v_j,... ali pa je v_i kar enako v_j.

Ta sistem lahko zapišemo kot

$$X = MX + I,$$

kjer je $X = [x_{ij}]$, rešitev sistema pa je $X = M^*I$.

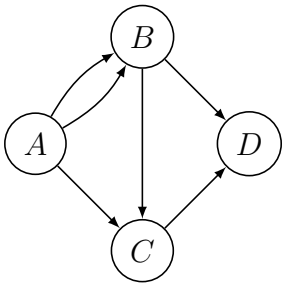
Velja

$$M^* = I + M + M^2 + M^3 + \dots, \tag{12}$$

saj pot poljubne dolžine obstaja, če upoštevamo vse poti dolžine 0 (torej opisane v matriki I), vse poti dolžine 1 (matrika M), dolžine 2 (matrika M^2)... Od n-te potence matrike M naprej so vse enake, torej $M^n = M^{n+1} = ...$, saj najdaljša pot poteka skozi največ n vozlišč. Formula (12) ustreza enemu od predpisov za zaprtje, ki smo jih videli v primeru 1.7.

Oglejmo si še zgled:

Zgled 4.1. Imamo enostaven usmerjen graf:



ki ga predstavimo s sledečo matriko sosednosti M:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

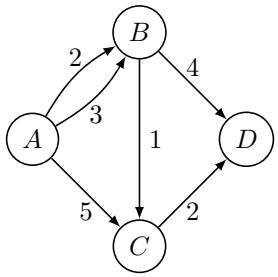
Rešitev našega sistema oz. matrika, ki pove, katera vozlišča so med seboj dosegljiva, je

$$M^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

♢

4.2. **Poti v grafu.** Začnimo kar z zgledom:

Zgled 4.2. Naj bodo A, B, C in D mesta, ki jih povezujejo različno drage ceste. Zanima nas, kako draga je najcenejša pot iz A v D. Upoštevati moramo, da je najcenejših poti lahko več, torej nas zanimajo vse poti, ali pa podamo dodatne pogoje. Mi bomo predpostavili, da so vozlišča urejena leksikografsko in bomo tako izbrali najcenejšo pot.



Vidimo, da najcenejša pot iz A v D stane 5, obiščemo pa vozlišča A, B, C in D (v tem vrstnem redu). ♢

Naša naloga je torej podobna tisti v prejšnjem razdelku (razdelku 4.1), le da je sedaj usmerjen graf tudi utežen.

V tem primeru uporabimo tropski polkolobar $\mathbb{T}$. Utežen graf predstavimo z matriko M, kjer je M_{ij} teža povezave iz vozlišča j v vozlišče i, oziroma ∞, če povezava ne obstaja. Za M_{ii} definiramo $M_{ii} = 0$.

Ponovno bomo za zaprtje uporabili neskončno vrsto: $M_{ij}^{(k)}$ je dolžina najcenejše poti s k povezavami iz vozlišča i v j, M^* pa je vsota (torej minimum) $M^{(k)}$ za vse k. M_{ij}^* je tako dolžina najcenejše poti iz i v j (število povezav je poljubno).

Zgled 4.3. Izračunajmo sedaj rešitev za zgornji zgled z uporabo Jacobijevega algoritma, torej z uporabo formule

$$a^{k+1} = a^{(0)} \oplus M \odot a^{(k)}$$

Še enkrat si oglejmo graf iz zgleda 4.2.

Predstavimo ga z matriko sosednosti M:

$$M = \begin{bmatrix} 0 & \infty & \infty & \infty \\ 2 & 0 & \infty & \infty \\ 5 & 1 & 0 & \infty \\ \infty & 4 & 2 & 0 \end{bmatrix}$$

ki jo beremo po stolpcih kot: iz vozlišča A pridemo v A po povezavi dolžine 0, v B po povezavi dolžine 2, povezava iz A v C je dolga 5, v D pa ne moremo, torej ∞.

Izračunali bomo stolpec za vozlišče A. Začnemo z $a^{(0)}$ in $a^{(1)}$, pri čemer slednjega prepišemo kar iz matrike M.

$$a_A^{(0)} = \begin{bmatrix} 0 \\ \infty \\ \infty \\ \infty \end{bmatrix}, \quad a_A^{(1)} = \begin{bmatrix} 0 \\ 2 \\ 5 \\ \infty \end{bmatrix}$$

Uporabimo sedaj formulo $a^{k+1} = a^{(0)} \oplus M \odot a^{(k)}$:

$$a_A^{(2)} = \min\{a_A^{(0)}, M \odot a_A^{(1)}\} = \min\left\{ \begin{bmatrix} 0 \\ \infty \\ \infty \\ \infty \end{bmatrix}, \begin{bmatrix} \min\{0, \infty, \infty, \infty\} \\ \min\{2, 2, \infty, \infty\} \\ \min\{5, 3, 5, \infty\} \\ \min\{\infty, 6, 7\infty\} \end{bmatrix} \right\} = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 6 \end{bmatrix}$$

Z največ dvema povezavama tako lahko iz vozlišča A v vozlišča A, B, C in D pridemo s potmi dolžine 0, 2, 3 in 6 (v tem vrstnem redu).

Ko računamo še naprej (ker $k = 2$ ni enak $n = 4$), dobimo

$$a_A^{(3)} = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 5 \end{bmatrix}, \quad a_A^{(4)} = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 5 \end{bmatrix}$$

tu pa je $k = n$ in $a^{(k)} = a^{(k+1)}$, zato iteracijo končamo, $a_A^{(3)} = a_A^{(4)}$ pa je naša rešitev za prvi stolpec.

Ko iteracijo izvedemo še na preostalih treh stolpcih, dobimo končno rešitev, torej zaprtje matrike M, M^*, je enako:

$$M^* = \begin{bmatrix} 0 & \infty & \infty & \infty \\ 2 & 0 & \infty & \infty \\ 3 & 1 & 0 & \infty \\ 5 & 3 & 2 & 0 \end{bmatrix}$$

♢

Opomba 4.4. Algoritem bi deloval tudi za graf z negativnimi utežmi, dokler imajo vsi cikli utež, za katero obstaja zaprtje. Tako bi lahko namesto običajnega $\mathbb{T}$ uporabili tudi tropski polkolobar iz primera 1.8 (graf torej ne sme imeti negativnih ciklov).

Opomba 4.5. Za izračun zaprtja matrike bi lahko uporabili tudi algoritem WFK ali Haskell algoritem iz razdelka 2. Oba porabita $\mathcal{O}(n^3)$ operacij, a drugi vedno razišče celoten graf in vrne rezultat za vse pare vozlišč, zato ni najbolj optimalen za preverjanje povezanosti za samo en par vozlišč.

V nadaljevanju si bomo ogledali tri primere v Haskellu, to bodo:

- `ShortestDistance`, ki vrne samo ceno najcenejše poti,
- `ShortestPath` vrne ceno najcenejše poti in seznam vozlišč, ki jih na tej poti obiščemo ter še
- `LongestDistance`, ki vrne samo ceno najdražje poti.

V Haskellu definiramo najcenejšo pot kot `ShortestDistance`, ki je lahko bodisi neka številska razdalja bodisi nedosegljivo oz. `Unreachable`, če pot ne obstaja. Definiramo še enoti in lastnosti.

```
data ShortestDistance = Distance Int | Unreachable
instance Semiring ShortestDiatance where
        zero = Unreachable
        one = Distance 0

        --addition
        x @+ Unreachable = x
        Unreachable @+ x = x
        Distance a @+ Distance b = Distance (min a b)

        --multiplication
        x @. Unreachable = Unreachable
        Unreachable @. x = Unreachable
        Distance a @. Distance b = Distance (a+b)

        --closure
        closure x = one
```

Oglejmo si sedaj še zgled v Haskellu:

Zgled 4.6. Izračunali bomo zaprtje konkretne matrike.

```
data Matrix2 a = Matrix2 Int [[a]] deriving Show

next :: Semiring a => Matrix2 a -> Int -> Matrix2 a
next akm1 k =
        Matrix2 n [[((m!!i)!!j) @+ ((m!!i)!!k) @. (closure((m!!k)!!k)) @.
        ((m!!k)!!j) | j <- [0..(n-1)]] | i <- [0..(n-1)]]
    where Matrix2 n m = akm1

compute :: Semiring a => Matrix2 a -> Int -> Matrix2 a
compute a0 0 = a0
compute a0 k = next (compute a0 (k-1)) (k-1)

delta :: Semiring r => Int -> Int -> r
delta i j
```

```
    | i == j = one
    | otherwise = zero

solution :: Semiring a => Matrix2 a -> Int -> Matrix2 a
solution a0 k  =
        Matrix2 n [[(delta i j) @+ ((ac!!i)!!j)
        | j <- [0..(n-1)]] | i <- [0..(n-1)]]
    where Matrix2 n ac = compute a0 k
```

Opomba 4.7. Tu uporabimo razred `Matrika2` namesto `Matrika`, ki smo ga videli v razdelku 2.

Matriko m iz zgleda 4.3 zapišemo kot

```
m = Matrix2 4 [[Distance 0, Unreachable, Unreachable, Unreachable],
    [Distance 2, Distance 0, Unreachable, Unreachable],
    [Distance 5, Distance 1, Distance 0, Unreachable],
    [Unreachable, Distance 4, Distance 2, Distance 0]]
```

Če program poženemo z ukazom `solution m 4`, kar pomeni, da bo izračunal poti do vključno dolžine 4, nam za rešitev vrne

$$m^* = \begin{bmatrix} 0 & \infty & \infty & \infty \\ 2 & 0 & \infty & \infty \\ 3 & 1 & 0 & \infty \\ 5 & 3 & 2 & 0 \end{bmatrix}$$

kar je ista matrika, kot smo jo videli v zgledu 4.3. ♢

Podoben problem je problem najkrajše poti v grafu, kjer nas zanima le število povezav, ki jih prehodimo. Tako vsaki povezavi priredimo težo 1. Tako graf postane enostaven, matrika m pa 0/1 matrika. Tu sta operaciji spet min namesto $\oplus$ ter $+$ namesto $\odot$, kjer $a \odot b$ predstavlja dolžino poti, $a \oplus b$ pa izbere najkrajšo pot. Poti iste dolžine so spet urejene leksikografsko.

Primer `ShortestPath` uporabimo, ko nas poleg cene najcenejše poti zanima še, katera vozlišča prepotujemo na tej poti. Številski rezultat je tako enak kot pri `ShortestDistance`, le da dobimo še seznam vozlišč.

```
data ShortestPath n = Path Int [(n,n)] | NoPath
instance Ord n => Semiring (ShortestPath n) where
        zero = NoPath
        one = Path 0 []

        --addition
        x @+ NoPath = x
        NoPath @+ x = x
        Path a p @+ Path a' p'
                | a < a'                        = Path a p
                | a == a' && p < p'         = Path a p
                | otherwise                     = Path a' p'

        --multiplication
```

```
        x @. NoPath = NoPath
        NoPath @. x = NoPath
        Path a p @. Path a' p' = Path (a + a') (p ++ p')

        --closure
        closure x = one
```

Zadnji primer pa je primer najdaljše poti `LongestDistance`, ki vrne ceno najdražje poti, torej deluje ravno obratno kot `ShortestDistance`. Zato kot operaciji izberemo max kot $\oplus$ in seštevanje $+$ kot $\odot$. Pozorni moramo biti le na grafe s cikli, saj so tu poti lahko neskončno dolge. Zato v Haskellu definiramo še dve vrednosti: `LUnreachable`, če med dvema vozliščema ni poti, in `LInfinite`, če je pot neskončno dolga zaradi pozitivnega cikla.

```
data LongestDistance = LDistance Int | LUnreachable | LInfinite
instance Semiring LongestDistance where
        zero = LUnreachable
        one = LDistance 0

        --addition
        x @+ LUnreachable = x
        LUnreachable @+ x = x
        LInfinite @+ _ = LInfinite
        _ @+ LInfinite = LInfinite
        LDistance x @+ LDistance y = LDistance (max x y)

        --multiplication
        x @. LUnreachable = LUnreachable
        LUnreachable @. x = LUnreachable
        LInfinite @. _ = LInfinite
        _ @. LInfinite = LInfinite
        LDistance x @. LDistance y = LDistance (x+y)

        --closure
        closure LUnreachable = LDistance 0
        closure (LDistance 0) = LDistance 0
        closure _ = LInfinite
```

4.3. **Polinomi in potenčne vrste.** Za dan polkolobar z zaprtjem S lahko definiramo polkolobar polinomov $S[x]$ spremenljivke x, katerih koeficienti so elementi S. Polinome predstavimo kot sezname koeficientov, kjer i-ti element seznama predstavlja koeficient pri členu x^i. Tako polinom $7 + x^2$ predstavimo kot $[7, 0, 1]$.

Operaciji seštevanja $\oplus$ in množenja $\odot$ definiramo sledeče:

- Pri seštevanju samo seštejemo ustrezne elemente obeh seznamov (i-ti člen vsote je vsota i-tih členov polinomov, ki ju seštevamo), če pa je en polinom "daljši" od drugega (ima več členov), ju naredimo enako dolga tako, da krajšemu seznamu dodamo ustrezno dolg seznam ničel (polinom $7 + x^2$ lahko zapišemo tako $[7, 0, 1]$ kot $[7, 0, 1, 0, 0, 0]$, saj velja $7 + x^2 = 7 + 0x + x^2 + 0x^3 + 0x^4 + 0x^5$).

- Množenje je malo bolj zapleteno. Vemo, da je vsak polinom sestavljen iz konstantnega člena (prvi element v seznamu) in elementov, ki so deljivi z x. V *Haskell*-u nam delitev seznama $a : p$ razdeli polinom p na prosti koeficient a ter polinom q, kjer torej velja $p = a + qx$. Če pomnožimo dva taka polinoma, dobimo

  $$(a + px)(b + qx) = ab + (aq + bp + pqx)x.$$

  Tako smo v resnici definirali konvolucijo, saj velja

  $$c_n = \sum_{i=0}^{n} a_i b_{n-i}$$

  kjer smo množili polinoma $\tilde{p}$ in $\tilde{q}$ s koeficienti a_i in b_i, dobili pa njun produkt, polinom $\tilde{r} = \tilde{p} \cdot \tilde{q}$ s koeficienti c_i.

Problem pa se pojavi, ko želimo definirati zaprtje, saj noben polinom p^* ne ustreza predpisu $p^* = 1 + p^* x$ (stopnji na levi in desni se ne ujemata). Zato bomo polinome posplošili na formalne potenčne vrste, torej dodamo še neskončne sezname koeficientov. Enota za seštevanje je polinom $p = 0$, za množenje pa $p = 1$. Tako dobimo polkolobar $S[[x]]$, saj ustreza vsem lastnostim iz definicije 1.1.

Potenčne vrste, ki jih bomo uporabili, bodo povsem formalne; ne bo nas zanimala vrednost za določen x, prav tako ne neskončne vsote, limite ter konvergenca. Tako bo "množenje z x" predstavljalo samo premik zaporedja za eno mesto. Celo za polkolobarje S, ki nimajo komutativnega množenja, ti premiki še vedno veljajo in lahko zapišemo $pxq = pqx$ (seveda pa ne $pqx = qpx$).

Dano imamo torej formalno potenčno vrsto $s = a + px$, iščemo pa njeno zaprtje $s^* = b + qx$, ki zadošča $s^* = 1 + s \cdot s^*$. Pri tem je $p, q \in S[[x]]$:

$$\begin{aligned} b + qx = 1 + (a + px)(b + qx) = 1 + ab + aqx + bpx + pqx^2 = \\ = 1 + ab + aqx + p(b + qx)x \end{aligned}$$

Enačimo koeficiente:

- $b = 1 + ab$, od prej vemo, da je fiksna točka afine preslikave kar $b = a^*$.
- $q = aq + p(b + qx) = aq + ps^*$, spet $q = a^* ps^*$.

Ko to vstavimo nazaj v $s^* = b + qx$, dobimo predpis za zaprtje $s^* = a^* + a^* ps^* x = a^*(1 + ps^*)$.

Spet na začetku definiramo enoti, nato podamo še predpise za seštevanje, množenje in zaprtje:

```
instance Semiring r => Semiring [r] where
        zero = []
        one = [one]

        --addition
        [] @+ y = y
        x @+ [] = x
        (x:xs) @+ (y:ys) = (x @+ y):(xs @+ ys)

        --multiplication
        [] @. _ = []
        _ @. [] = []
```

```
(a:p) @. (b:q) =
(a @. b):(map (a @.) q @+ map (@. b) p @+ (zero:(p @. q)))

-- closure
closure [] = one
closure (a:p) = r
        where r = [closure a] @. (one @+ (p @. r))
```

Sedaj lahko rešujemo vse enačbe oblike $x = bx + c$, kjer sta b in c potenčni vrsti nad poljubnim polkolobarjem. Take enačbe se pojavijo v primerih dinamičnega programiranja, kot je na primer 0-1 nahrbtnik s ponavljanjem.

4.4. **0-1 nahrbtnik s ponavljanjem.** Danih imamo n predmetov s težami w_1, ..., w_n, kjer je $w_i > 0$ za vsak i in cenami $v_1, ..., v_n$, $v_i \geq 0$ za vsak i. Pri maksimalni ceni želimo napolniti nahrbtnik, ki drži težo W, vzamemo pa lahko poljubno naravno število vsakega predmeta.

Algoritem izračuna tabelo t, kjer je $t(w)$ najvišja cena pri omejitvi teže w. Predpostavimo $t(0) = 0$, za vse nadaljnje vrednosti pa uporabimo formulo

$$t(w) = \max_{0 \leq w_i \leq w} (v_i + t(w - w_i))$$

Opomba 4.8. V nadaljevanju se bomo osredotočili na max-plus polkolobar z zaprtjem S, ki smo ga definirali v 4.2, torej polkolobar najdražjih poti oz. `LongestDistance`. Tu sta enoti ∞ za $\oplus$ (max) in 0 za $\odot$ (seštevanje). Zaradi preglednosti bomo uporabili splošni operaciji $+$ in $\cdot$.

Naj bodo v_i in $t(w)$ elementi max-plus polkolobarja z zaprtjem S. V notaciji tega polkolobarja velja

$$t(0) = 1_S$$

in

$$t(w) = \sum_{i=0}^{w} v_i \cdot t(w - w_i).$$

Parametra v_i in w_i lahko združimo v isti polinom $V = \sum_i v_i x^{w_i}$. Če na primer napolnimo nahrbtnik s štirimi tipi kovancev s cenami 1, 5, 7 in 10 ter težami 3, 6, 8 in 6, dobimo polinom V:

$$V = x^3 + 5x^6 + 7x^8 + 10x^6$$

Ker uporabljamo polkolobar max-plus, je to ekvivalentno

$$V = x^3 + 10x^6 + 7x^8$$

Če V predstavimo s seznamom, je w-ti element cena najbolj vrednega predmeta s težo w (ki je lahko 0, če ne obstaja noben predmet s težo w). Podobno predstavimo $t(w)$ kot potenčno vrsto $T = \sum_i t(i)x^i$. Če T predstavimo s seznamom, je w-ti element najvišja dosežena cena pri omejitvi teže w.

Vidimo, da je zgornja definicija $t(w)$ v resnici konvolucija polinoma T in potenčne vrste V. Če upoštevamo še $t(0)$ kot enoto polkolobarja, dobimo enostavnejšo definicijo $t(w)$:

$$T = 1 + V \cdot T$$

oziroma $T = V^*$, tako smo dobili elegantno rešitev za 0-1 nahrbtnik s ponavljanjem.

Na tem mestu opazimo, da lahko uporabimo prej definirano zaprtje

$$x^* = 1 + x + x^2 + ...$$

saj je rešitev problema nahrbtnika najvišja cena v primeru, če ne izberemo nobenega predmeta (člen 1), če izberemo enega (x), dva (x^2),...

Namesto uporabe polkolobarja z zaprtjem `LongestDistance` lahko definiramo novega `LongestPath` podobno kot smo `ShortestPath` v razdelku 4.2, le z operacijo max namesto min. Ko uporabljamo ta polkolobar, naša definicija nahrbtnika še vedno deluje ter vrne množico predmetov, ki smo jih izbrali, namesto le njihove končne vrednosti oz. cene.

Slovar strokovnih izrazov

semiring polkolobar
closure zaprtje
tropical semiring tropski polkolobar

Literatura

[1] S. Dolan, *Fun with Semirings - a functional pearl on the abuse of linear algebra*, ACM SIGPLAN Notices, ICFP 2013, volume 4, issue 9, pp. 101-110, 2013.

[2] M. Gondran, M. Minoux, *Graphs, dioids and semirings - new models and algorithms*, Springer Science+Business Media, New York, 2008.

[3] D. J. Lehmann, *Algebraic structures for transitive closure*, Theoretical Computer Science, volume 4, issue 1, pp. 59-76, 1997.

[4] S. Thompson, *Haskell: the craft of functional programming*, Addison-Wesley, 1999.