Visszalépéses felsoroló

Gregorics Tibor

gt@inf.elte.hu
ELTE IK

Absztrakt. A visszalépéses keresésnek több változata is ismert. Ebben a cikkben azon feladatokat megoldó visszalépéses algoritmusokra fókuszálunk, amelyeknek problématere egy speciális irányított fával írható le. Elemezni fogjuk ennek a fának a bejárási stratégiáit, és rámutatunk arra, hogyan lehet ezek közül az egyik bejárást az úgynevezett visszalépéses felsoroló segítségével elvégezni. Ez vezet el a visszalépéses keresés algoritmusának egy olyan változatához, amely egy visszalépéses felsorolóval működő lineáris keresés. Az így kapott algoritmus objektum-orientált megvalósítását könnyen át lehet állítani egyik feladatról egy másikra, vagy lecserélhető benne a lineáris keresés más programozási tételre, ha például egy feladatot visszalépéses számlálással vagy éppen visszalépéses maximumkereséssel lehet csak megoldani.

Kulcsszavak: útkeresési probléma, visszalépéses keresés, felsoroló, programozás elmélet.

1. Bevezetés

A visszalépéses kereséssel azok a feladatok oldhatóak meg, amelyeket a modellezésük során útkereső problémaként tudunk értelmezni. Ilyenkor a feladat megoldását egy olyan irányított gráfban keressük, amely akár végtelen sok csúcsot is tartalmazhat, de minden csúcs kifoka (a csúcsból kivezető irányított élek száma) véges. A feladat problématerét e gráf startcsúcsból induló útjai alkotják. A megoldáshoz egy startcsúcsból induló célcsúcsba vezető úgynevezett megoldási utat kell megtalálni.

A visszalépéses keresésnek több változatát is ismerjük [10], amelyek közül annak megfelelően kell választani, hogy milyen speciális tulajdonságai vannak a modellt leíró gráfnak [8]. A legáltalánosabb változat [6,8,11] tetszőleges gráfokra alkalmazható, de szüksége van egy úgynevezett mélységi korlátra ahhoz, hogy a keresés biztosan terminál̈jon. Ekkor, ha van olyan megoldási út a gráfban, amelynek hossza nem nagyobb a mélységi korlátnál, akkor a keresés egy ilyen megoldást fog találni. A megfelelő mélységi korlát kiválasztása azonban nem egyszerű. Ha túl kicsi, nem találunk megoldást; ha túl nagy, megnő a futási idő. Ismerünk kizárólag véges és körmentes gráfokban kereső visszalépéses keresést is [6,8,11], amelynek nincs szüksége mélységi korlátra ahhoz, hogy mindig terminál̈jon és találjon megoldást, ha van. Még ismertebbek azok a változatok, amelyek egy speciális (véges, startcsúcs gyökerű, egy-egy szinten ugyanakkora kifokú csúcsokat tartalmazó) irányított fában működnek; a fa ágai alkotják a problématerét és ezek között kell a megoldási utat megtalálni.

Ez a cikk a visszalépéses keresés ezen utóbbi változatával foglalkozik. A 2. fejezet definiálja azt a feladatosztályt, amelyet ezzel a változattal meg lehet oldani, és megmutatja, hogyan lehet a probléma teret irányított fával szimbolizálni. A 3. fejezet két olyan módszert ismertet a feladatot modellező irányított fa bejárására, amely bejárásokat a visszalépéses keresés alkalmazza. Ezek a bejárások a szakirodalomból ismertek, de újszerű gondolat ezeknek a bejárásoknak a kereséstől való elválasztása. A 4. fejezet azt a bejárást – amelyik alapötlete Fóthi Ákostól származik [7,10] – önálló felsorolóként valósítja meg. Ezt hívjuk majd visszalépéses felsorolónak. Ennek

felhasználásával a vizsgált visszalépéses keresés változatot egy ismert programozási tételből, a felsorolóval működő lineáris keresésből [4,5] származtatjuk. Ennek a változatnak előnyeit az 5. fejezet tárgyalja, amely a javasolt változat objektum-orientált megvalósítását is bemutatja.

2. Visszalépéses kereséssel megoldható speciális tulajdonságú feladatok modellezése

Az állapottér-reprezentáció egy jól ismert általános modellezési technika, amellyel a feladatokat útkeresési problémaként lehet megfogalmazni. [6] Ennél meg kell konstruálnunk a feladat állapotterét (a feladat szempontjából lényeges adatok lehetséges érték-együtteseinek halmazát), ki kell jelölni a kezdő- és végállapotokat, definiálni kell az állapot-átmeneteket előidéző műveleteket úgy, hogy a feladat megoldása egy kezdőállapotból végállapotba vezető művelet-sorozat legyen. Az állapotér-reprezentációnak nyilvánvalóan megfeleltethetünk egy irányított gráfot, amelyben a csúcsok az állapotokat, az irányított élek az állapotok közötti átmeneteket szimbolizálják. Ekkor a kezdőállapotnak megfeleltetett startcsúcsból egy végállapotnak megfeleltetett célcsúcsba vezető irányított útnak, azaz egy megoldási útnak a megtalálása a feladat.

Speciális tulajdonságú feladatok esetében azonban használhatunk más modellezési technikát is. Mi a továbbiakban azzal a feladatosztállyal foglalkozunk majd, amelyek az alábbi, úgynevezett speciális modellel írhatóak le. [1,7,9,10]

1. Definíció. Adott n darab ($n \in \mathbb{N}$) véges halmaz: D_1 , ... , D_n. Képezzük ezek $D = D_1 \times ... \times D_n$ direkt szorzatát. A cél a D halmaz egy olyan elemének megtalálása, amely kielégít egy $\rho: D \to \mathbb{L}$ ($\mathbb{L}=\{igaz, hamis\}$) állítást, amelyhez megadható egy $\rho_0, \rho_1, ... , \rho_n: D \to \mathbb{L}$ állítás-sorozat az alábbi feltételekkel:

1. $\rho_0 \equiv igaz$
2. $\rho_n \equiv \rho$
3. $\rho_0, \rho_1, ... , \rho_n$ monoton, azaz $\forall i \in \{1, ... ,n\}$-re és $\forall u \in D$-re ha $\rho_i(u)$, akkor $\rho_{i-1}(u)$
4. $\rho_i(u)$ csak az $u \in D$ első $i \in \{1, ... ,n\}$ komponensétől függ, azaz

   $\forall i \in \{1, ... ,n\}$-re és $\forall u,v \in D$-re ha $\forall j, 1 \le j \le i$-re $u_j = v_j$, akkor $\rho_i(u) = \rho_i(v)$

Sok feladatnál a $\rho_0, \rho_1, ... , \rho_n$ állítás-sorozatot úgy adjuk meg, hogy $\forall i \in \{1, ... ,n\}$-re teljesül a $\rho_i \equiv \rho_{i-1} \wedge \beta_i$, ahol a $\beta_i : D \to \mathbb{L}$ egy olyan alkalmas állítás, amelyre $\beta_i(u)$ csak az $u \in D$ első $i \in \{1, ... ,n\}$ komponensétől függ. Ekkor ugyanis a fenti definíció 3. és 4. feltétele automatikusan teljesül.

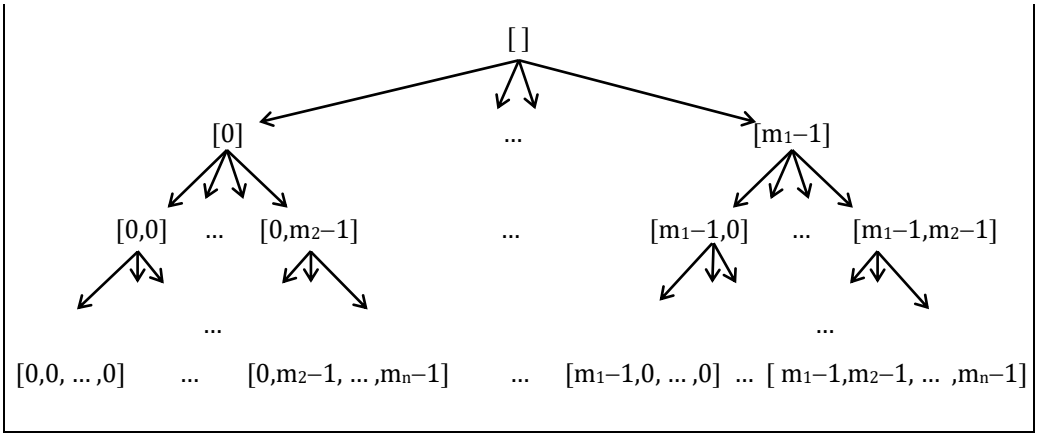
Klasszikus példát ad egy ilyen modellre az n-királynő probléma, ahol egy n×n-es sakktáblára kell felhelyezni n darab királynőt úgy, hogy egyik se üsse a másikat. A sakktábla i-edik sorában elhelyezendő i-edik királynő lehetséges oszlop pozícióit a $D_i=\{0, ... , n-1\}$ halmaz tartalmazza. (Mint látjuk, ebben az oszlopokat később megindokolt technikai okból 0-tól kezdve sorszámozzuk.) Legyen továbbá $\rho_0 \equiv \rho_1 \equiv igaz$ és

$$\forall i \in \{2, ... ,n\}\text{-re és } \forall u \in D\text{-re } \rho_i(u) = \rho_{i-1}(u) \wedge \forall j \in \{1, ... , i-1\}: (u_i \neq u_j \wedge |u_i - u_j| \neq |i - j|).$$

A definícióban szereplő D halmaz elemeit irányított fa segítségével ábrázolhatjuk. A fa gyökere egy speciális „üres” elem, amelynek nincsenek komponensei. A fa első szintjén a D_1 elemei, a második szintjén a $D_1 \times D_2$ elemei, általában az i-edik szinten a $D_1 \times ... \times D_i$ elemei

helyezkednek el. A fa i−1-edik szintjén levő csúcsok kifoka a D_i halmaz számossága. Jelölje a továbbiakban ezt a számosságot az m_i. Az i-edik szint bármely csúcsának első i−1 komponense éppen ezen csúcs szülőcsúcsa; másképpen fogalmazva a szülőcsúcs mindig prefixe a gyermekeinek. A fa levélcsúcsai a $D_1 \times ... \times D_n$ halmaz, azaz a D elemei. Mindezek alapján az 1. definícióban jellemzett feladatok úgy is felfoghatóak, hogy egy speciális fának azt a levelét keressük, amelyik kielégíti a ρ állítást. Másképpen fogalmazva a gyökércsúcsból (ami itt a startcsúcs) a ρ állításnak megfelelő levélcsúcsba (ez a célcsúcs) vezető utat keressük.

Ha D_i számossága m_i, akkor a D_i-beli elemeket 0-tól m_i-1-ig megsorszámozhatjuk, ezáltal az elemekre a sorszámukkal hivatkozhatunk. Ennek megfelelően az előbb jellemzett fát az alábbi módon rajzolhatjuk le. Figyeljünk fel a problémateret ábrázoló fa (1. ábra) szabályosságára: levelei azonos szinten (n. szint) helyezkednek el, ugyanazon szinten levő csúcsok kifoka azonos (az i. szinten m_i).



1. ábra: Speciális útkeresési feladatok problématere

Érdemes megfigyelni, hogy a D_i elemeinek sorszámozása egy bijekciót definiál a D halmaz és a $\{0, ... , m_1 \cdot m_2 \cdot ... \cdot m_n - 1\}$ halmaz között. A D-nek egy eleme ugyanis kölcsönösen egyértelmű módon leképezhető egy vegyes alapú számrendszerbeli számra. Itt a helyiértékek alapja balról jobbra haladva rendre $m_1, m_2, ... , m_n$, és az i-edik számjegy értéke 0 és m_i-1 közé esik. Nyilvánvaló, hogy ennek a vegyes alapú számrendszerbeli természetes számnak az értéke 0 és $m_1 \cdot m_2 \cdot ... \cdot m_n - 1$ közé fog esni.

Formálisan, ha a $\nu_i: D_i \rightarrow \{0, ... , m_i-1\}$ egy bijekció, akkor a $\nu: D \rightarrow \{0, ... , m_1 \cdot m_2 \cdot ... \cdot m_n - 1\}$ leképezés, amely $\forall u \in D$-re $\nu(u) = \Sigma_{i=1...n} \nu_i(u_i) \cdot (\Pi_{j=1...n-i} m_{n-j})$ is egy bijekció lesz. Jelölje a ν leképezés inverzét a továbbiakban ϕ.

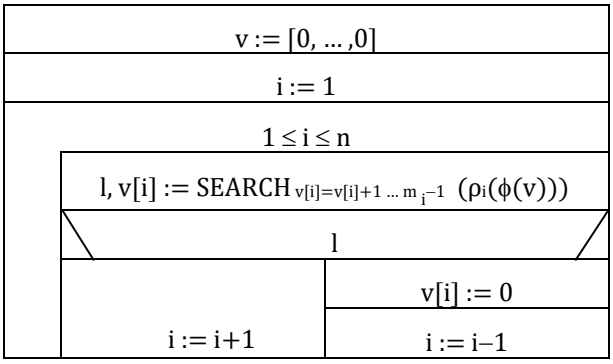
3. A problématér bejárása

A visszalépéses keresés változatainak többsége a mélységi bejárási stratégiát követve vizsgálják át a problémateret. Meg kell azonban említeni, hogy ez nem teljesen egyezik meg a gráfok közismert mélységi bejárásával [2]. Ez utóbbi ugyanis a csúcsok bejárását végzi, míg a visszalépéses keresés a startcsúcsból kivezető utak között keresi meg az első adott tulajdonságú utat. A gráfok mélységi bejárása explicit módon ismeri azt a gráfot, amelyet be kell járnia, a

visszalépéses keresés azonban ennél jóval kisebb memóriát használ: csak az aktuális utat tartja nyilván. Ez a tulajdonság különösen hasznos akkor, amikor a problémateret leíró irányított gráf olyan nagy (esetleg végtelen), hogy nem is lehet explicit módon azt eltárolni. Ennél fogva viszont a visszalépéses keresés nem képes nyilvántartani, hogy a gráf csúcsai közül melyeket érintett már korábban, így a visszalépések következtében elhagyott csúcsokat, ha azokhoz egy újabb utat talál, újra bejárja. Ez a kellemetlen jelenség nem következhet be, ha a bejárni kívánt gráf egy fa, mint például az általunk vizsgált speciális feladatok esetében, ahol problémateret az 1. ábrán bemutatott fa gyökércsúcsból levélcsúcsokba vezető útjai alkotják.

Egy irányított fában a visszalépéses keresés mélységi bejárást követő stratégiája azt jelenti, hogy a keresés szisztematikusan vizsgálja meg „balról jobbra haladva" a fa gyökércsúcsából kivezető ágait. Ehhez egy adott pillanatban egyetlen ágnak a nulladik szinttől az i-edik szintig tartó részét tárolja a keresés. Ha az i-edik szinten levő csúcsra nem teljesül a ρ_i állítás, akkor – mintha levágták volna az ez alatti részeket – nem lép lejjebb a fában, hanem visszalép egy szinttel feljebb és az onnan kivezető soron következő ágon indul lefelé. Ha nincs már ilyen soron következő ág, akkor addig lép vissza felfelé, amíg ilyet nem talál. Ennek megvalósításához nyilván kell tartani az aktuális ágon kívül a még ki nem próbált leágazásokat is. A keresés kétféleképpen érhet véget: vagy talál egy olyan ágat, amelyen egészen az n-edik szintig végig fut, és az ott levő levélcsúcsra teljesül a ρ állítás, vagy a gyökércsúcsból akar visszalépni, ami lehetetlen. Ez utóbbi jelzi, hogy nincs megoldás.

Az általunk vizsgált speciális modellnél a fent vázolt bejárási stratégia még egyszerűbb lesz. Egy-egy ág tárolása helyett ugyanis elég az ág végén levő csúcsot, az i jegyű vegyes alapú számrendszerbeli számot eltárolni. Ebből ugyanis nemcsak az olvasható ki, hogy melyek az idevezető ág csúcsai (ezek a szám különböző prefixei), hanem az is, hogy melyek az ág még ki nem próbált leágazásai. Ha ugyanis az ág i-edik szintjén szereplő szám i-edik helyiértéke x, akkor még nem vizsgáltuk az ág i–1-edik szintjén levő csúcs azon gyerekeit, amelyek i-edik helyiértéke az x+1 és m_i–1 közé esik.



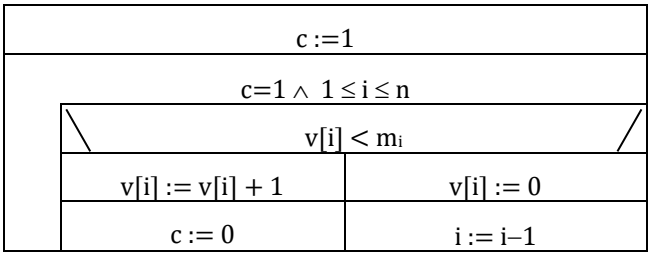
2. ábra: A problématér mélységi bejárását alkalmazó visszalépéses keresés

A 2. ábrán a speciális modellre adaptált visszalépéses keresésnek egy jól ismert változata látható. Ebben a $v:\mathbb{N}^n$ tömb első i–1 eleme ($i \in \{1, \dots, n+1\}$) az aktuális ág végén levő csúcsot jellemző helyiértékes számot mutatja. A v tömb és az i–1 index együtt egy aritmetikai ábrázolású vermet határoz meg. Az i=0 melletti leállás azt jelzi, hogy nincs megoldás; az i>n melletti terminálás esetén a megoldás a v-ben van. A ciklus mag első lépése az aktuális ág végén

levő csúcsnak azt a még nem vizsgált gyerekét keresi meg, amely kielégíti a ρ_i állítást. (A SEARCH az intervallumon értelmezett lineáris keresésre utal [4]: megkeresi a v[i] számára a jelenlegi értékénél nagyobb, de az m_i-nél kisebb első olyan értéket, amellyel teljesülni fog a $\rho_i(\phi(v))$ feltétel). Ha talál ilyet, akkor l:𝕃 igaz lesz, és ekkor lejjebb lép a bejárás az i-edik szintre a fában (i := i+1). Ha nem talál, akkor vágás történik és visszalép (i := i−1) a bejárás. (A v tömb a hozzátartozó i indexszel tulajdonképpen egy verem adatszerkezetet reprezentál.)

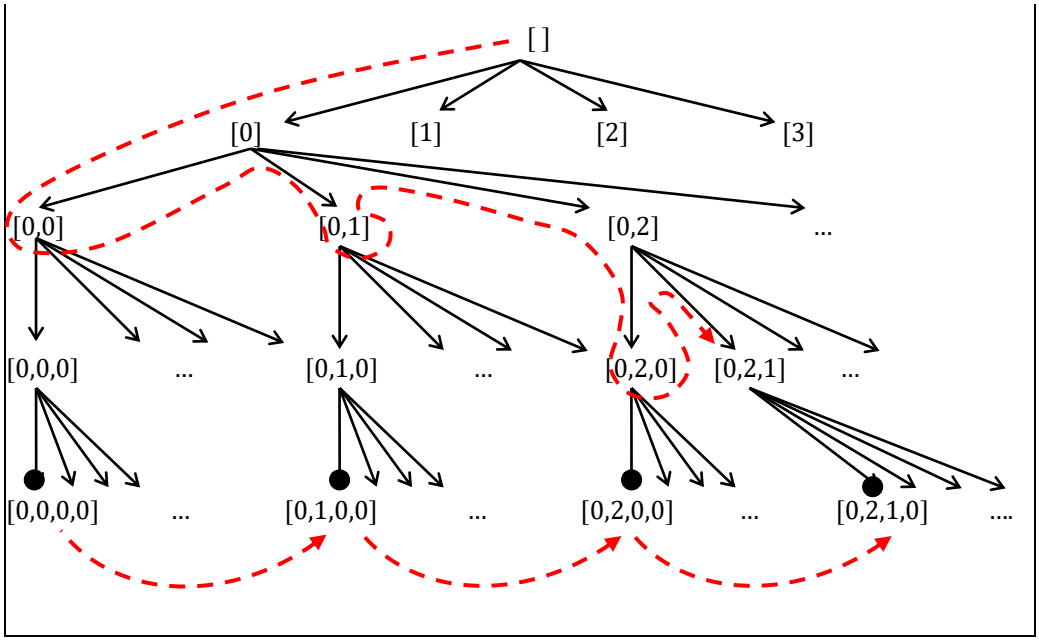
Az általunk vizsgált speciális modell problématere azonban egy egészen más stratégiával is bejárható. Elég ugyanis felsorolni az 1. ábrán látható fa utolsó szintjén levő levélcsúcsokat balról jobbra haladva. Mivel ezek a csúcsok helyiértékes számábrázolású természetes számok, a bejárásnak lényegében természetes számokat kell növekvő sorrendben felsorolnia. Nem érdemes azonban ezt a felsorolást egyesével végezni, hiszen azokat a csúcsokat átugorhatjuk, amelyek biztosan nem lehetnek célcsúcsok. De hogyan lehet ezt eldönteni anélkül, hogy megvizsgálnánk őket? Tegyük fel, hogy megvizsgáltuk a v számot, és $\phi(v)$-re fennáll a ρ_{i-1} állítás, de a ρ_i állítás már nem. Nyilvánvaló, hogy ezt a hibát a v szám helyiértékes számábrázolásának i-edik helyiértékén álló érték okozza. Hibába növelnénk meg ekkor a v-t eggyel, ez többnyire csak az utolsó helyiértéken okozna változást (a helyiértékes összeadás miatt persze keletkezhet átvitel is). Ezért a v-t ne eggyel, hanem annyival növeljük meg, hogy az i-edik helyiérték, ha minimálisan is, de biztosan megváltozzon.

Egy n hosszú vegyes alapú számrendszerbeli szám i-edik helyiértékének növelését végzi el a 3.ábra algoritmusa az esetleges átvitelek kezelését is beleértve. [7,10] Ha az algoritmus c=1 mellett áll le, akkor ez azt jelzi, hogy túlcsordulás következett be: a megnövelt számot már nem lehet n helyiértéken ábrázolni, illetve ugyanez történik érvénytelen i megadásakor is. Ez a problématér úgynevezett növeléses bejárásának egy lépése.



3. ábra: A problématér növeléses bejárásának egy lépése

Ezt az algoritmust majd egy olyan környezetbe kell beágyazni, amelyik elemzi az aktuális v-t, és megkeresi az első olyan i-t, amelyre $\rho_i(\phi(v))$ hamis. (Ha nincs ilyen, akkor készen vagyunk, hiszen ekkor teljesül v-re a ρ_n is, tehát fennáll a ρ is.) Ezt az i-t kell odaadni a fenti eljárásnak, amely a kívánt módon megnöveli a v-t.



4. ábra: A problématér kétféle bejárása a 4-királynő probléma esetén

Érdemes összevetni a bemutatott kétféle bejárási technikát. A 4. ábra a 4-királynő probléma problématerét mutatja. Ennek fájában a mélységi bejárás vertikális mozgást végez, míg a növeléses bejárás a fa levelei mentén horizontálisan egy irányba mozog, pontosabban jó néhány levélcsúcsot kihagyva ugrál. A mélységi bejárás során jól megfigyelhető az, amit visszalépésnek hívunk. (A pontosság kedvéért meg kell jegyeznünk azt, hogy a 2. ábra mélységi bejárást alkalmazó visszalépéses keresése, amikor az aktuális ág végén levő csúcsnak egy még nem vizsgált gyerekét keresi, akkor azt nem pontosan a 4. ábrán látható módon teszi. Az i−1-edik szinten levő aktuális csúcsból nem lép ugyanis annak olyan gyerekcsúcsához, amelyik nem elégíti ki a ρ_i állítást, hogy utána visszalépjen. Ehelyett megkeresi a még nem vizsgált gyerekcsúcsok közül azt, amelyik kielégíti ρ_i állítást, ezáltal bizonyos visszalépéseket elrejt az algoritmus. Tényleges visszalépésre a ciklusmagbeli elágazás jobboldali ágában kerül sor, amikor nem talál megfelelő gyerekcsúcsot.) A növeléses bejárás esetében viszont teljesen rejtve maradnak a visszalépések. Amikor ez a bejárás egy újabb természetes számot választ, akkor több természetes számot is átugorhat. A 4. ábrán láthatjuk, hogy egy-egy ilyen ugrás megfelel a mélységi bejárást végző változat egy-egy olyan visszalépésekből álló lépéssorozatának, amikor a soron következő ág kiválasztására kerül sor. Ebből látszik, hogy a „növeléses” bejárás esetén akkor beszélhetünk visszalépésről, amikor csökken az a helyiérték, ahol az aktuális szám számjegyét növelni akarjuk. Ez a 3. ábra algoritmusában a ciklusmagbeli elágazás jobb oldala.

4. Visszalépéses felsoroló megvalósítása

Most elkészítjük az előző fejezetben másodikként kifejtett „növeléses” bejárást megvalósító, mostantól visszalépéses felsorolónak nevezett bejárást. Ehhez két adattípust vezetnünk be.

Először megadjuk a D-beli elemek típusát. (Az alábbi táblázat [4] felső sora a típus specifikációját: típusértékek halmazát és a típus műveleteket mutatja, alsó sora a típus megvalósítását: típusértékek reprezentációját és a típusműveletek implementációját tartalmazza.)

A típus lehetséges értékei a $D = D_1 \times ... \times D_n$ halmaz elemei. Ezekre egyetlen műveletet definiálunk (good()), amely el tudja dönteni, hogy egy elem kielégíti-e a ρ állítást.

D	l, ind := good(u) ahol u:D, l:𝕃, ind:ℕ
$m : (\mathbb{N}^+)^n$ $v : \mathbb{N}^n$	l, ind := $\forall SEARCH_{i=1..n}$ $(\rho_i(u))$

invariáns: $\forall i \in \{1, ... ,n\}: v[i] < m[i]$

5. ábra: A visszalépéses kereséssel megoldható feladatok típusának kezdetleges változata

Egy D-beli u elemet (típusértéket) egy n hosszú egydimenziós v tömbbel ábrázolhatjuk úgy, hogy $u=\phi(u.v)$, azaz a v tömbben az u elemnek megfeleltetett természetes szám megfelelő vegyes alapú számrendszerben felírt alakját tároljuk. A reprezentáció az m tömbben tárolja a D_i halmazok elemszámait. (Ez tehát egy konstans, objektum orientált megvalósítás esetén egy konstans osztályszintű adattag.) Mivel az i-edik helyiérték (i=1, … , n) alapszáma a D_i halmaz elemszáma (m[i]), ezért a v[i] értéke mindig egy ennél kisebb természetes szám kell legyen. Ezt rögzíti a típus invariánsa. Az l, ind := good() egy olyan eldöntéses feladat, amely azt vizsgálja, hogy teljesül-e mindegyik ρ_i állítás az u elemre. Ha nem, akkor megadja a legkisebb olyan indexet (ind), amelyre nem teljesül a ρ_{ind}. Az "l, ind := $\forall SEARCH_{i=1..n}$ $(\rho_i(u))$" jelölés ezen eldöntéses feladatot megoldó úgynevezett optimista lineáris keresésre utal. [4]

Most pedig tervezzük meg a D-beli elemek felsorolását biztosító típust. A típus leírására az előzőhöz hasonló táblázatot használunk.

enor(D)	first()	next(i) i:ℕ	end()	current()
u : D c : {0, 1}	**if** n<1 **then** c:=1 **else** c:=0 u.v := [0, ... ,0]	c, u := inc(u, i)	c=1	u

6. ábra: A visszalépéses felsoroló típusának kezdetleges változata

Egy felsoroló típusértékei a felsorolások, azaz a felsorolni kívánt elemek véges sorozatai, műveletei pedig a first(), next(), end(), és current(). [4,5] A D-beli elemeket az azokat reprezentáló természetes számok növekvő sorrendjében soroljuk fel. Ehhez elég a felsorolás aktuális elemét tárolni (u), illetve egy speciális flag-et (c), amely a felsorolás végét jelzi majd. A felsorolás a [0, ... ,0] számmal ábrázolt elemmel kezdődik (first()), ilyenkor c=0, de csak n≥1 esetén; a felsorolás végét a c=1 feltétel jelzi (end());a felsorolás current() művelete az aktuális u elemet adja vissza; a felsorolás next() művelete előállítja az u-t követő elemet. Ez utóbbi rendhagyó a felsorolók világában, mivel van egy i:ℕ bemenő paramétere is, amely arra szolgál, hogy a soron következő elemhez a korábban már bemutatott "ugrással" juthassunk el. Nem

meglepő, hogy ezért a next() műveletként a 3. ábra eljárását használjuk azzal a különbséggel, hogy benne a v tömbre u.v-ként, az m_i-re u.m[i]-ként kell hivatkoznunk. Jelöljük ezt a c, u := inc(u, i) értékadással (ahol u egyszerre bemenő- és eredmény változó is), ahol az inc() a növeléses eljárásra utal. Ez így előállított új u elemet fogja majd a current() művelet segítségével a good() művelet megkapni. Az u elem good()-beli vizsgálata gyorsítható, ha az u elem mellett megadjuk az u elem előállítását végző inc() eljárás i változójának végső értékét. Jelöljük ezt is ind-del. Be lehet ugyanis látni, hogy az előállított u elemre fenn áll a $\rho_{ind-1}(u)$. Emiatt a good() műveletnek felesleges minden ρ_i állítást i=0 … n -ig ellenőriznie az u-ra, elég csak i=ind … n -ig megnézni azokat. Ez az észrevétel jelenik meg a D-beli elemek típusának alábbi javított változatában. Látható, hogy az ind egyszerre bemenő- és eredmény változója is a good() műveletnek.

D	l, ind := good(u, ind) ahol u:D, l:𝕃, ind:ℕ
m : $(\mathbb{N}^+)^n$ v : $\mathbb{N}^n$	l, ind := $\forall$SEARCH $_{i=ind..n}$ ($\rho_i(u)$)

invariáns: $\forall i \in \{1, \ldots, n\}: v[i] < m[i]$

7. ábra: A visszalépéses kereséssel megoldható feladatok típusa

A teljesség kedvéért megadjuk a good() művelet részletes algoritmusát is, amely tehát egy optimista lineáris keresés.

l, ind := good(u, ind

l, i := igaz, ind

l ∧ i ≤ n

l := $\rho_i(u)$

i := i+1

ind := i–1

8. ábra: A visszalépéses kereséssel megoldható feladat good() művelete

A fentieknek megfelelően módosítjuk a visszalépéses felsoroló típusának leírását is. A felsoroló reprezentációja kiegészül az ind:ℕ változóval, amely egyrészt azt jelzi, hogy fenn áll az aktuális u elemre a $\rho_{ind-1}(u)$, másrészt, hogy az u.v tömb ind-edik és azt követő elemei mind nullák. Ez a felsoroló típusának invariánsa. Ezt szem előtt tartva kell a first() műveletnek ind:=1-ként inicializálnia az ind változót, a next() műveletnek pedig az inc() eljárásban használt i változó utoljára felvett értékét kell az ind változónak átadni. A current() művelet az u elemmel együtt ennek az ind változónak az értékét is visszaadja.

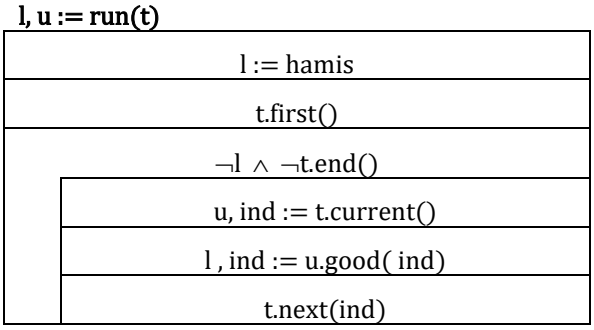
enor(D)	first()	next(i) i:ℕ	end()	current()
u : D c : {0, 1} ind : ℕ	**if** n<1 **then** c:=1 **else** c:=0 u.v := [0, ... ,0] ind:=1	c, u, i := inc(u, i) ind := i	c=1	u, ind

invariáns: $\rho_{ind-1}(u)$ és $\forall i \in [ind+1..\ n]$: u.v[i]=0

9. ábra: A visszalépéses felsoroló típusa

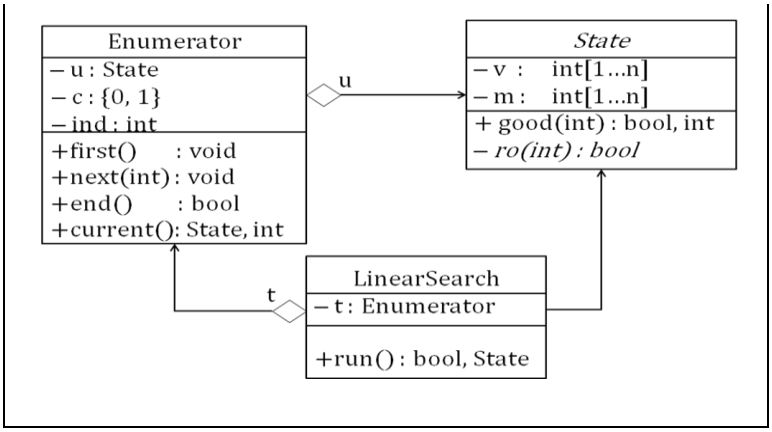
5. Értékelés

A visszalépéses felsoroló birtokában könnyen elkészíthető a visszalépéses keresésnek egy új változata (10. ábra). Ez a felsorolókra általánosított lineáris keresés programozási tételéből [4,5] származik úgy, hogy a t felsoroló műveleteire támaszkodik és gondoskodik az aktuális helyiérték (ind) átadásáról a current(), a good() és a next() műveletek között.



10. ábra: Visszalépéses felsorolójú lineáris keresés

Ezt a típus központú, de alapvetően procedurális megoldást objektum-orientált formában is implementálhatjuk. Ehhez a korábban bevezetett típusokat osztályokkal írjuk le, továbbá egy új osztályt is bevezetünk, amelynek egyetlen metódusa (run()) a 10. ábra lineáris keresése. A lineáris keresés használatához egy olyan objektumra van szükség, amelyre ezt a run() metódust meg lehet hívni. A visszalépéses keresés új változata tehát három objektum együttműködésén alapul: a feladat aktuális állapotát képviselő u objektumon, a t visszalépéses felsoroló objektumon, és a lineáris keresést végző objektumon. A három objektum osztálya az 11. ábra szerinti kapcsolódik egymáshoz.



11. ábra: Visszalépéses felsorolóra épített visszalépéses keresés osztálydiagramja

A State osztály (korábbi jelölésben a D típus) good() metódusában a $\rho_i()$, pontosabban a $\rho(i)$ metódus absztrakt. Amikor egy konkrét problémát kell megoldani, akkor ezt a $\rho(i)$ metódust kell felüldefiniálni. A State osztálynak az u objektumát tartalmazza az Enumerator osztály ($enor(D\times\mathbb{N})$ típus) t felsoroló objektuma, amely viszont a LinearSearch típusú lineáris keresés objektum tartozéka. A lineáris keresés közvetlenül is hivatkozik a State osztályra, hiszen run() metódusa meghívja a good() metódust.

Az, hogy a fenn definiált visszalépéses (felsorolóra épített visszalépéses) keresés három jól elkülöníthető részből (objektumból) áll, igen rugalmassá teszi a keresést a különféle változtatásokkal szemben, hiszen könnyű benne egy-egy modult (objektumot) lecserélni.

Egy konkrét probléma megoldásához csak a State osztály $\rho(i)$ metódusát kell újradefiniálni. Ezt úgy érdemes megtenni, hogy a State osztályból származtatjuk a konkrét probléma állapotainak osztályát, amelyben az eredetileg absztrakt $\rho(i)$ metódust felüldefiniáljuk. Az u objektum ennek a származtatott osztálynak lesz tehát az objektuma, miközben a másik két objektum (felsoroló és a lineáris keresés) változatlan marad, azok tehát újrahasznosíthatóak.

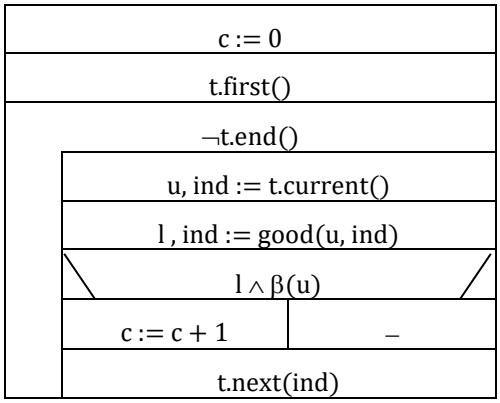
Ugyancsak a State osztályt kell módosítani az olyan feladatok megoldásánál, amelyek modellje némileg eltér az 1. definícióban leírtaktól. Sokszor ugyanis egy útkeresési feladat problématere egy olyan irányított fával ábrázolható, amelyben a fa egy szintjén különböző kifokú csúcsok helyezkednek el, vagy a célcsúcsok lehetnek belső csúcsok is.

Az első esetben a fa kiegészíthető úgy, hogy egy csúcsnak alibi leszármazottai is legyenek, és így egy szinten minden csúcsnak azonos lesz a kifoka. Ehhez a ρ_i értelmezését kell kiterjeszteni úgy, hogy az olyan D-beli elemekre, amelyeknek i-edik komponense ilyen „alibi” érték, hamis értéket adjon. Ez a módosítás tehát a $\rho(i)$ metódust érinti csak.

A második esetben a speciális modell definíciója úgy módosul, hogy a $\rho\equiv\rho_n$ kritérium helyett a $\rho\equiv \exists i\in\{1, \dots ,n\}: \rho_i$, vagy általánosabban $\rho\equiv\exists i\in\{1, \dots ,n\}: \rho_i\wedge\gamma_i$ jelenik meg, ahol $\gamma_i:D\to\mathbb{L}$ olyan, hogy a $\gamma_i(u)$ csak az u első i komponensétől függ. Ennek kezeléséhez olyan keresést kell a good() metódusban implementálni, amely sikeresen terminál, ha talál olyan $i\in\{1, \dots ,n\}$-t, amelyre $\rho_i(u)\wedge\gamma_i(u)$ igaz; egyébként megadja az első olyan $ind\in\{1, \dots ,n\}$-t, amelyre $\rho_{ind}(u)$ hamis. Itt tehát a good() metódust kell felüldefiniálni.

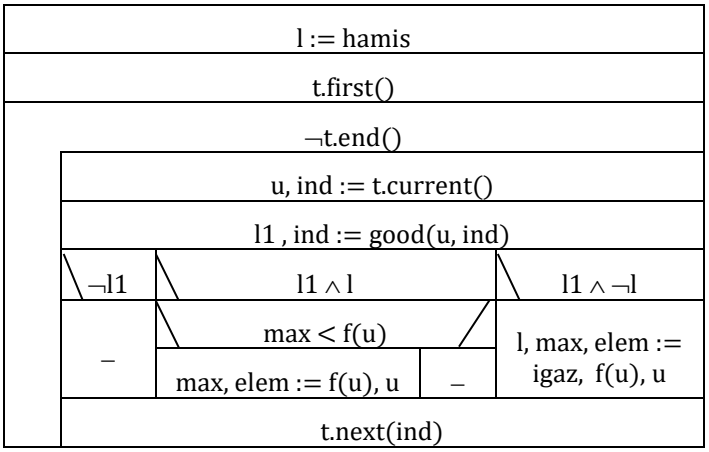
A lineáris keresés objektumát kell lecserélni akkor, amikor például olyan problémával találkozunk, amikor nem egy megoldást keresünk, hanem mondjuk meg kell számolnunk a megoldásokat, vagy meg kell keresnünk valamilyen szempontból a legjobb megoldást. Ilyenkor nem kell mást tennünk, mint a felsorolókra megfogalmazott lineáris keresés helyett egy felsorolókra megfogalmazott számlálást vagy egy maximumkeresést használni.

A "visszalépéses számlálást" a felsorolóra általánosított számlálás programozási tételéből nyerhetjük [4,5] a visszalépéses felsoroló használatával. A program tovább is általánosítható úgy, hogy csak bizonyos tulajdonságú megoldásokat számoljon meg. Ilyenkor megjelenik benne ezt a "bizonyos" tulajdonságot vizsgáló $\beta:D\rightarrow\mathbb{L}$ logikai függvény.



12. ábra: Visszalépéses számlálás

A "visszalépéses maximumkeresés" a felsorolóra általánosított feltételes maximumkeresés programozási tételéből [4,5], és a visszalépéses felsorolóból áll össze. Az $f:D\rightarrow H$ függvény olyan H halmazba képez, amelyik elemei jól rendezhetőek.



13. ábra: Visszalépéses maximumkeresés

Végezetül említsük meg azt a didaktikai előnyt, amely a visszalépéses felsorolóra épülő visszalépéses keresésnek az oktatásba történő bevezetésénél jelentkezik. Ekkor ugyanis csak a visszalépéses felsoroló bemutatása jelent újdonságot, hiszen a lineáris keresés a tanmenet e szakaszában már ismert algoritmusséma a hallgatóság előtt. Segítséget jelenthet, ha a hallgatók ekkor már találkoztak felsoroló fogalmával, sőt használtak már különféle felsorolókat (egy tömb elemeinek felsorolásán kívül). Természetesen a megoldandó problémák jellemzésére ki kell térni, de ez a visszalépéses keresés többi változatának tanítása esetén sem nélkülözhető.

Irodalom

1. Berman, K.A., Paul, J. L.: *Fundamentals of Sequential Algorithms*, PWS Publishing, 1996.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L.: *Introduction to Algorithms*, Massachusetts Institute of Technology, 1990.
3. Dijkstra, E.W.: *A Discipline of Programming*, Prentice-Hall, 1976.
4. Gregorics, T.: *Programozás 1.kötet Tervezés*, ELTE Eötvös Kiadó, 2013.
5. Gregorics, T.: *Programming theorems on enumerator*. Teaching Mathematics and Computer Science, Debrecen, 8/1 (2010), 89-108.
6. Fekete, I., Gregorics, T., Nagy, S.: *Bevezetés a mesterséges intelligenciába*, LSI, Budapest, 1990.
7. Fóthi, Á.: *Bevezetés a programozáshoz*. ELTE Eötvös Kiadó. 2005.
8. Futó, I. (szerk): *Mesterséges intelligencia*, Aula, Budapest, 1999.
9. Knuth, D. E.: *Estimating the Efficiency of Backtrack Programs*, Mathematics of Computation, Vol. 29 (1975), 121-136.
10. Lőrentey, K., Fekete, I., Fóthi, Á., Gregorics, T.: *On the Wide Variety of Backtracking Algorithms*, ICAI'05 Eger, Hungary, January 28-February 3. 2005, 165-174.
11. Nilsson, N. J.: *Principles of Artificial Intelligence*, Springer-Verlag, Berlin, 1982.