21 - Adatszerkezetek

Egyszerű adattípusok – tömb, verem, sor, elsőbbségi sor, lista, bináris fa, gráf – ábrázolásuk és műveleteik.

1. Adatszerkezetek leírása

Az adattípusok leírásának több, egymásra épülő absztrakciós szintje van, ezek közül a legfelső három tartozik a tétel témakörébe:

1. Absztrakt adattípus (ADT)
2. Absztrakt adatszerkezet (ADS)
3. Reprezentáció (ábrázolás)

Az **ADT** esetén az adattípust úgy specifikáljuk, hogy szerkezetére, reprezentálására, implementálására semmilyen megfontolást, előírást, utalást nem teszünk. A leírásban kizárólag matematikai fogalmak használhatóak. Az ADT szinten az adattípus belső struktúráját nem látjuk, hanem műveleteik által definiáljuk őket, de a műveletek működése szintén nem megadott.
A műveletek definiálására két mód van. Az egyik az **algebrai (axiomatikus) specifikáció**, amikor axiómák segítségével alkotunk egy teljes, ellentmondásos rendszert. A másik lehetőség a **funkcionális specifikáció**, ekkor a műveleteknek az elő- és utófeltételét adjuk meg.

Az **ADS** ezt azzal egészíti ki, hogy megmondjuk azt is, hogy alapvetően milyen struktúrája van a szóban forgó adattípusnak. Közelebbről ez azt jelenti, hogy megadjuk az adatelem közötti legfontosabb rákövetkezési kapcsolatot, és ezt egy irányított gráf formájában le is rajzoljuk. Az ADS szintet tehát egy szerkezeti gráf és az ADT szinten bevezetett műveletek alkotják együttesen.

A **reprezentáció**s szinten arról hozunk döntést, hogy az ADS szinten megjelent gráf rákövetkezési relációit milyen módon ábrázoljuk. Az ábrázolás még mindig absztrakt, noha már a számítógépes megvalósítást modellezi. Két tiszta reprezentálási mód van, az aritmetikai (tömbös) és a láncolt (pointeres), illetve ezeket lehet vegyesen is alkalmazni.
Az **aritmetikai ábrázolás** során a memóriát úgy képzeljük el, mint egy hosszú, egydimenziós tömböt, vagyis vektort, amelynek elemei adatszerkezet alaptípusának példányai.
A **láncolt ábrázolás**hoz bevezetjük az absztrakt mutató (pointer) fogalmát, amely egy másik adatelemre hivatkozik. (A sehová sem mutató pointer értéke NIL.) Az adatelemek tartalmazhatnak adat és mutató komponenseket egyaránt. Utóbbiak segítségével láncoljuk össze az adatszerkezetet.

Bővebben >>

2. Tömb

2.1. ADT

Legyen T egy E alaptípus feletti $k(\geq 1)$ dimenziós tömb típus. Legyen $I = \times_{j=1}^{k} I_j$ indexhalmaz, ahol $\forall j \in [1,k]: I_j = [1, n_j]$. Az $A \in T$ tömbnek így $N = \times_{j=1}^{k} n_j$ eleme van, ez a $\{a_1, \dots, a_N\}$ halmaz.

Mindig létezik $f: I \longrightarrow [1, N]$ kölcsönösen egyértelmű leképezés, amely a tömb indexfüggvénye és használatával teljesül: $A[i_1, \dots, i_k] = a_{f(i_1,\dots,i_k)}$.

A tömb műveletei egy **tömbelem lekérdezése**, illetve egy **tömbelem módosítása** az f indexfüggvény által. A tömb mérete nem változik, elem beszúrása vagy törlése nem lehetséges.

A tömböt $k = 1$esetén vektornak, $k = 2$ esetén mátrixnak nevezzük.

2.2. ADS

A tömbelemekre definiáljuk a j. indexhalmaz szerinti **rákövetkezés**t, azaz:
$next_j A[i_1, \dots, i_j, \dots, i_k] = A[i_1, \dots, i_{j+1}, \dots, i_k]$, ahol $i_j < n_j$.

A rákövetkezést a dimenzió határán lévő elemekre nem definiáljuk. Minden belső elemeknek így k darab rákövetkezője van.

2.3. Reprezentáció

Aritmetikai ábrázolás esetén az elemeket általában **sorfolytonosan**, illetve **oszlopfolytonosan** helyezzük egy vektorba. Csak akkor használjuk ezt a módot, ha a hatékony ábrázolás úgy kívánja.

A láncolt ábrázolás tipikusan a hézagosan kitöltött mátrixok (más néven **ritka mátrixok**) esetében használatos.

Bővebben >>

3. Verem

3.1. ADT

A verem olyan adattároló típus, amelyben mindig csak a legutoljára behelyezett – és még ki nem vett – adatelemet érjük el. A veremből az elemeket pontosan behelyezésével ellentétes sorrendben lehet kivenni.

Műveletek:

- $Emtpy: \longrightarrow V$, üres verem konstans, üres verem létrehozása;
- $IsEmpty: V \longrightarrow \mathbb{L}$, a verem üres voltának lekérdezése;
- $Push: V \times E \longrightarrow V$, elem betétele a verembe;
- $Pop: V \longrightarrow V \times E$, elem kivétele a veremből ($V \neq \emptyset$);
- $Top: V \longrightarrow E$, a legfelső elem lekérdezése ($V \neq \emptyset$).

A V egy E alaptípus feletti verem típus.

Axiómák:

$IsEmpty(Empty)$
$IsEmpty(v) \Rightarrow Empty = v$
$\neg IsEmpty(Push(v, e))$
$Pop\big(Push(v, e)\big) = (v, e)$

$Push(Pop(v)) = v$
$Top(Push(v,e)) = e$

3.2. ADS

A verem lineáris adatszerkezet, az alapvető szerkezetét egyirányú gráf ábrázolja.

3.3. Reprezentáció

Az **aritmetikai ábrázolás** esetén a vermet egy max elemszámú vektorral reprezentáljuk, ez a v. Bevezetjük a top változót, amely mindig a veremben lévő legfelső elem indexét adja meg ($top \in [0, max]$). A top értéke akkor 0, ha a verem üres.
Mivel a tömb betelhet, a szokásos műveletek mellett használni kell egy új, $IsFull: V \longrightarrow \mathbb{L}$ műveletet is, és egyes műveletek során ezt figyelni kell.

Láncolt ábrázolás esetén az adatelemek egyik komponense az elem értékét tartalmazza, a másik pedig egy pointer, ami a veremben alatta lévő elemre mutat. A v is egy pointer, ami a verem legfelső elemére hivatkozik.

Bővebben >>

4. Sor

4.1. ADT

A sor olyan adattároló típus, amelybe új adatelemet csak a végére tudunk beszúrni, a bennlévő elemek közül pedig csak a legelső – azaz a legrégebben betett és még ki nem vett – érthető el. A sorból az elemeket pontosan behelyezéssel azonos sorrendben lehet kivenni.

Műveletek:

- $Emtpy: \longrightarrow S$, üres sor konstans, üres sor létrehozása;
- $IsEmpty: S \longrightarrow \mathbb{L}$, a sor üres voltának lekérdezése;
- $In: S \times E \longrightarrow S$, elem betétele a sorba;
- $Out: S \longrightarrow S \times E$, elem kivétele a sorból ($S \neq \emptyset$);
- $First: S \longrightarrow E$, a legelső elem lekérdezése ($S \neq \emptyset$).

Az S egy E alaptípus feletti sor típus.

Axiómák:

$IsEmpty(Empty)$
$IsEmpty(s) \Rightarrow Empty = s$
$\neg IsEmpty(In(s,e))$
$Out(In(Empty,e)) = (Empty,e)$
$\neg IsEmpty(s) \Rightarrow Out(In(s,e))_2 = Out(s)_2$
$\neg IsEmpty(s) \Rightarrow In(Out(s)_1,e) = Out(In(s,e))_1$
$First(s) = Out(s)_2$

4.2. ADS

A sor is lineáris adatszerkezet, az alapvető szerkezetét egyirányú gráf ábrázolja.

4.3. Reprezentáció

Az **aritmetikai ábrázolás** esetén a sort egy max elemszámú vektorral reprezentáljuk, ez az s. Bevezetjük az $first$ változót, amely mindig a sorban lévő legelső elem indexét adja meg ($first \in [1, max]$), és a db változót, amely a sor elemszámát tartja nyilván ($db \in [0, max]$).
A sor úgy működik, hogy a sor végére rakjuk, illetve az elejéről veszünk ki az elemeket, ezért a tömb elejére „beengedjük" a tömb végén „kilógó" elemeket, különben betelne úgy a tömb, hogy egyes indexei felhasználatlanok. Egy tipikus sor így lassan „körbemegy" a tömbön.
Mivel a tömb betelhet, a szokásos műveletek mellett használni kell egy új, $IsFull: S \longrightarrow \mathbb{L}$ műveletet is, és egyes műveletek során ezt figyelni kell.

Láncolt ábrázolás esetén az adatelemek egyik komponense az elem értékét tartalmazza, a másik pedig egy pointer, ami a sorban utána következő elemre mutat. Az s is egy pointer, ami a sor legelső elemére hivatkozik. Annak érdekében, hogy a sor utolsó elemét konstans hatékonysággal elérjük – azaz könnyedén a sor végére szúrhassunk elemeket –, bevezetünk egy $last$ pointert is, ami mindig a sor utolsó elemére mutat.

Bővebben >>

5. Elsőbbségi sor

Köznapi fogalma, megjelenése: pl. sürgősségi osztályon a páciensek nem a beérkezési időnek megfelelően, hanem a sürgősség mértéke szerint kerülnek ellátásra. Az operációs rendszerekben a prioritásos jobok feldolgozása is hasonló elv alapján történik.

5.1. ADT

A prioritásos sor olyan adattároló típus, amelybe új elemeket beszúrhatunk, illetve a már benn lévő elemek közül a – valamilyen függvény által – legprioritásosabbnak ítéltet kivehetjük.

Műveletek:

- $Emtpy: \longrightarrow P$, üres elsőbbségi sor konstans, üres elsőbbségi sor létrehozása;
- $IsEmpty: P \longrightarrow \mathbb{L}$, az elsőbbségi sor üres voltának lekérdezése;
- $Insert: P \times \mathbb{N} \longrightarrow S$, elem beszúrása az elsőbbségi sorba;
- $DelMax: P \longrightarrow P \times \mathbb{N}$, legmagasabb prioritású elem kivétele az elsőbbségi sorból ($P \neq \emptyset$);
- $Max: P \longrightarrow \mathbb{N}$, a legmagasabb prioritású elem lekérdezése ($P \neq \emptyset$).

Az egyszerűség kedvéért a P prioritásos sor típusba csak az egyes elemek prioritását tesszük be, az elemet magát nem. Így a sorban csak $\mathbb{N}$-beli elemek vannak. Azaz a P az $E := \mathbb{N}$ alaptípus feletti elsőbbségi sor típus.

Axiómák:

$IsEmpty(Empty)$
$IsEmpty(p) \Rightarrow Empty = p$

$\neg IsEmpty(Insert(p,n))$
$DelMax(p)_2 = Max(p)$
$Insert\big(DelMax(p)\big) = p$
$\neg IsEmpty(DelMax(p)_1) \Rightarrow Max(p) \geq Max(DelMax(p)_1)$
$n \geq Max(p) \Rightarrow DelMax(Insert(p,n))_1 = p \wedge Max\big(Insert(p,n)\big) = n$
$n < Max(p) \Rightarrow\ Max\big(Insert(p,n)\big) = Max(p)$
$DelMax\big(Insert(Empy,n)\big) = (Empty,n)$
$Max\big(Insert(Empty,n)\big) = n$

5.2. ADS és reprezentáció

A két pontot együtt tárgyaljuk, ugyanis meg kell vizsgálni, hogy milyen **reprezentáció**val tudjuk leghatékonyabbá tenni a típus műveleteit. A különböző megvalósítási módok pedig különböző szerkezeti tulajdonságokat eredményeznek, melyekkel az **ADS**-szint foglalkozik. A következőkben látni fogjuk, hogy bármely megvalósítást is választjuk, az $Empty$ és az $IsEmpty$ műveletek konstans hatékonyságúak, ezért mindig csak a többi művelet hatékonyságát vizsgáljuk.

5.2.1. Megvalósítás rendezetlen tömbbel

A prioritásos sor elemei egy **rendezetlen tömb**be kerülnek, például a beírás sorrendjében. Egy segédváltozó használatával nyilvántarthatjuk, hogy melyik a tömb utolsó, feltöltött eleme. Ügyelni kell arra az esetre is, hogy a tömb betelhet.

A maximális prioritású elem **keresés**éhez a maximum keresés tétele alkalmazható. Törléskor a kitörölt elem helyére a tömb legutolsó eleme áthelyezhető.

A maximális elem keresése konstans hatékonyságúra gyorsítható, ha egy újabb segédváltozóval nyilvántartjuk annak a tömbelemnek az indexét, amely a maximális prioritású elemet tárolja. A segédváltozó értékét beszúráskor és törléskor frissíteni kell(het), ez azonban az $Insert$ és $DelMax$ műveletek hatékonyságán nagyságrendileg nem változtat.

5.2.2. Megvalósítás rendezett tömbbel

Az elsőbbségi sor elemei egy **rendezett tömb**be kerülnek, a prioritás szerinti növekvő sorrendben. Ebben az esetben is ügyelni kell arra, hogy a tömb betelhet.

Beszúráskor az új elem helyét a logaritmikus keresés tétele alapján a lineáris keresésnél gyorsabban megtalálhatjuk. A beillesztendő adatelemnél nagyobb prioritással rendelkező tömbelemeket eggyel jobbra kell tolni.

5.2.3. Megvalósítás kupaccal

A harmadik megvalósítással egy **kupac** tulajdonságú fát töltünk fel az elemekkel. A kupac (*heap*) egy olyan majdnem teljes, balra tömörített bináris fa (tehát az utolsó szint jobb oldaláról esetleg hiányozhatnak elemek), amelyben minden belső elem nagyobb, vagy egyenlő, mint a gyermekei.
A számítástechnikai megvalósítás ebben az esetben sem lesz pointeres, ajánlott a fát szintfolytonosan egy **tömb**be rakni. Ha az eredetileg meglévő kapcsolatokat lánconként behúzzuk a vektor egyes elemei között, akkor láthatjuk, hogy a tömb sem nem rendezetlen, sem nem teljesen

rendezett, hanem egy bizonyos kompromisszumos részben rendezettség áll fenn: az elemek az egyes láncok mentén rendezettek, mivel csökkenő sorozatokat alkotnak.

Beszúráskor az új csúcs beillesztésével meg kell tartanunk a kupac tulajdonságot, ezért az új csúcs az utolsó sorba, balról az első üres helyre, vagy ha az utolsó sor teljes, akkor egy új sor bal szélére kerül. Azonban még rendbe kell hozni a csúcsok értékeinek viszonyát, tehát elemcseréket végzünk alulról a gyökér felé haladva a megfelelő élsorozaton, ameddig szükséges. (Ez a tömbös reprezentációban a megfelelő láncot jelenti.)

A legmagasabb prioritású elem **kiolvasás**ához elegendő a gyökérelemet, azaz a tömb első elemét visszaadni. Ha ezt az adatelemet **töröl**ni is kívánjuk, akkor a szintfolytonos bejárás során utolsóként bejárt (jobb alsó) értéket rakjuk a gyökérelem helyére, az utolsó csúcsot pedig töröljük. A kupac tulajdonság megtartása érdekében a gyökérbe került értéket elemcserékkel lesüllyesztjük – mindig a nagyobb gyerekkel cserélve – a megfelelő helyre.

5.2.4. Hatékonyság összevetése

Az elsőbbségi sor legyen n elemű.

	Rendezetlen tömb	Rendezett tömb	Kupac
Insert	$\Theta(1)$	$\Theta(n)$	$\Theta(log_2 n)$
DelMax	$\Theta(n)$	$\Theta(1)$	$\Theta(log_2 n)$
Max	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

Látható, hogy ha a behelyezés és kivétel körülbelül azonos gyakorisággal fordul elő az elsőbbségi sor használatakor, akkor a kupac-szerkezet kiegyensúlyozott, és a másik kettőnél hatékonyabb reprezentációt nyújt.

Bővebben >>

6. Lista

A listáknak számos változata van, lehetnek **egyirányú**ak vagy **kétirányú**ak, **ciklikus**ak vagy **aciklikus**ak, illetve **fejelemes**ek vagy **fejelem nélküli**ek.

6.1. ADT

A listákkal végezhető tevékenységek függnek a lista fajtájától. Általánosságban elmondható, hogy a listák segéd pointerek által nyilvántartják az első, – aciklikus lista esetén – az utolsó, illetve egy aktuálisan kiválasztott elemüket. Az aktuálisan kiválasztott adatelem megváltoztatható az első vagy – aciklikus lista esetén – az utolsó elemre, továbbá léptethető a következő és – kétirányú láncolás esetén – az előző elemre.

Az összes listafajtára igaz még, hogy az aktuális adatelemben tárolt érték módosítható, illetve a kijelölt elem törölhető. Lehetőség van új elem beszúrására az aktuálisan kiválasztott elem elé és – kétirányú láncolás esetén – után, továbbá a lista elejére és – aciklikus lista esetén – végére is.

Végül természetesen lehet új, üres listát létrehozni, valamint megvizsgálni, hogy egy lista üres-e.

6.2. ADS

A lista lehet lineáris adatszerkezet, az alapvető szerkezetét egyirányú vagy kétirányú gráf ábrázolja.

6.3. Reprezentáció

A lista adatszerkezetet **láncolt listás reprezentáció**val célszerű ábrázolni. Minden adatelem egy komponense az elem értékét tartalmazza, a másik komponens pedig a következő elemre mutató pointert tartalmazza. Az utolsó elem a NIL-re mutat.
Kétirányú listáknál egy harmadik komponens adja az előző elemre mutató pointert. A fejelemes listák mindig rendelkeznek egy fejelemmel, így a lista ténylegesen üres soha nem lesz. A fejelem nem tartalmaz valós adatot az érték komponensében, csak a pointere(i) fontos(ak).

Bővebben >>

7. Bináris fa

7.1. ADT

A bináris fák szemléletesen olyan irányított fák, melyeknek minden csúcsa legfeljebb kettő gyerekcsúccsal rendelkezik. A bináris keresőfák olyan bináris fák, melyeknél megkülönböztetjük a bal és jobb oldali gyereket, és a fa minden csúcsára igaz, hogy ha van gyerekük, akkor a bal gyerek értéke a szülő csúcs értékénél kisebb, a jobb gyereké pedig nagyobb.

Műveletek:

- Üres, illetve egy elemes bináris fa létrehozása.
- A bináris fa megváltoztatása nélkül annak bal, illetve jobb részfájának lekérdezése.
- Új elem beszúrása a bináris fába.
- Elem törlése a bináris fából.

7.2. ADS

A bináris fát láncolt adatszerkezetként, gráffal ábrázolva képzeljük el.

7.3. Reprezentáció

Láncolt ábrázolást használva, minden adatelem három komponensből fog állni: a tárolt értékből, és a bal, valamint a jobb gyerek elemre mutató pointerből. Ha egy csúcsnak nincsen bal és/vagy jobb gyereke, akkor az a pointer NIL-re mutat.

Aritmetikai ábrázolást alkalmazva a bináris fa elemeit különböző bejárási módoktól függő sorrendben helyezzük el egy vektorban. A négy leggyakrabban használt bejárási mód:

- szintfolytonos
- pre-order
- in-order
- post-order

Bővebben >>

8. Gráf

A gráfokat feladatok széles körének megoldásához használják. A feladattól függően a gráfok tulajdonságai szűkíthetőek, például irányítatlan vagy egyszerű gráfra. A műveletek halmaza is attól függ, hogy milyen feladatot szeretnénk megoldani. Általában alapvető műveletnek tekintjük a csúcsok és élek hozzáadását és törlését, a bemenő és kimenő élek fokszámának lekérdezését, a szomszédos csúcsok meghatározását.

Két elterjedt, és gyakori gráfábrázolás használatos, az első aritmetikai, a második vegyes:

Csúcsmárixos: az n elemű gráfot egy $n \times n$-es mátrixszal reprezentáljuk. Az i. sor és j. oszlop metszetében található cella az i. csúcsból a j. csúcsba mutató él súlyát adja meg, nullát, ha a két csúcs nincsen összekötve. Súlyozatlan gráfok esetén a cellák tartalmazhatnak logikai értékeket. Irányítatlan gráfok esetén elegendő lehet a mátrix főátlóját és a feletti elemeket kitölteni.
A csúcsmátrixos ábrázolás sűrű gráfok esetén hatékony.

Éllistás: az n elemű gráfhoz egy n elemű tömböt rendelünk, minden csúcshoz egy tömbelemet. A tömbelemek egy-egy (fejelem nélküli) lista kezdőelemére mutatnak. A listák az adott sorszámú csúcs szomszédjait tárolják – sorszámmal és az odavezető él súlyával.

Nevezetes gráfalgoritmusok >>