Az 5.b ábrán látható ACB és ODA háromszögek hasonlóak, ezért $BC/AB = AD/AO$. Viszont $BC = \Delta_{vák.}$, $AO = R$ és használva az $AB = 2.AD \approx l = 3\,cm$ megközelítést: $\Delta_{vák.} \cong l^2/2.R$, ahonnan $\Delta_{vák.} = 2,3\,mm$.

A habba lépő **α**-részecske mozgási energiája fokozatosan nullára csökken. Ennek következtében a pálya görbületi sugara is rövidül, amiért Δ_{hab} valamivel nagyobb kell legyen mint a vákuumban kiszámított $\Delta_{vák.}$ értéke. Tehát elvárható, hogy a $\Delta_{hab} \geq \Delta_{vák.}$ teljesüljön.

• A mért- és a számított-érték összehasonlítása:

A 3 cm hosszú pálya végének, görbültsége okozta eltávolodása a függőlegestől, vákuumban $\Delta_{vák.} = 2,3\,mm$ lenne, viszont ennek a habban kísérletileg mért értéke $\Delta_{hab} \approx 3\,mm$. Mivel $3\,mm \geq 2,3\,mm$, ez *eleget tesz* elvárásunknak!

• Továbbá:

 Látványosabb – görbültebb – pályanyomokhoz jutnánk, ha a habkamrát még erősebb $B > 2\,T$ mágneses mezőbe helyeznénk.

 Érdekes lehetne a habok vizsgálata, vagyis az **α**-sugárzás által még hatékonyabban bontható habnak a keresése.

Irodalom:

[1]. Radioaktív sugárzások kimutatása „Kóbor Macska" módszerekkel /Kawakatsu Hiroshi, Kishizawa Shinichi/ Fizikai Szemle 1996/4.

[2]. A habkamra /Bíró Tibor / FIRKA 1997-98/2.

Bíró Tibor

Dinamikus programozás

III. rész

1. D*ivide et impera* vagy dinamikus programozás

Mindkét stratégia úgy fogja fel a feladatot, mint ami kisebb méretű hasonló részfeladatokra bontható, vagy hogy ezekből épül fel. Ez a szerkezet mindkét esetben fastruktúra. A fa csomópontjai, illetve a hozzájuk tartozó részfák ábrázolják az egyes részfeladatokat. Megtörténhet, hogy a feladat lebontásakor különböző ágakon azonos részfeladatokhoz jutunk, ami azt jelenti, hogy a fának lesznek identikus részfái.

Ezen emlékeztetők után lássuk a hasonlóságokat és a különbségeket.

1. A divide et impera akkor nem hatékony, ha a feladat lebontásakor identikus részfeladatok jelennek meg, hiszen ezeket többször is megoldja. A dinamikus programozás viszont annál hatékonyabb, minél több az azonos részfeladat, mivel képes elkerülni ezek ismételt megoldását. Ez a különbség a stratégiáikból adódik.
2. Egy divide et impera algoritmus először lebontja a feladatot (preorder mélységi bejárás szerint), majd a rekurzió visszaútján posztorder sorrendben megold minden részfeladatot, amellyel a lebontás alkalmával találkozott. Mivel minden részfeladat megoldását a közvetlen fiú-részfeladatai megoldásaiból

építi fel, ezért csak ezeket tárolja el, és ezeket is csak ideiglenesen (amíg az apacsomópont megoldása megépül). Más szóval, nem vezet nyilvántartást a már megoldott részfeladatokról és azok megoldásairól. Ez a magyarázata annak, hogy az azonos részfeladatokkal – anélkül, hogy tudomása lenne róla – többször is találkozik, újra és újra megoldva őket. Ezzel szemben a dinamikus programozás lentről (az egyszerűtől a bonyolult felé haladva) kezd neki a feladatnak, és minden részfeladatot csak egyszer old meg. Nyilvántartást vezet (általában egy tömbben) a már megoldott részfeladatok optimális megoldásairól, hogy amennyiben valamelyikre szükség lenne a későbbiekben, ne kelljen újra megoldania.

3. Optimalizálási feladatok esetében a dinamikus programozás a részfeladatok optimumértékeiről végzett nyilvántartásából utólag elő tudja állítani magát az optimális megoldást is. Ezzel szemben a divide et impera csak az optimális megoldáshoz tartozó optimumértékkel tud szolgálni.
4. Mi történne, ha a divide et impera kölcsönvenné a dinamikus programozástól a már megoldott részfeladatok nyilvántartásának ötletét? Úgy értjük ezt, hogy amikor először találkozik egy részfeladattal, a dinamikus programozáshoz hasonlóan, tárolja el a megoldását egy tömbbe, hogy valahányszor újra találkozik vele, egyszerűen csak elő kelljen vegye a megoldását. E feljavítás esetén a két technika ugyanolyan komplexitású algoritmust fog nyújtani. Ez esetben a részfeladatok megoldásának lentről felfelé iránya a postorder mélységi bejárási sorrendnek megfelelő fordított topologikus sorrend lesz. Ezt a stratégiát inkább a dinamikus programozás rekurzív változatának nevezhetnénk, mint divide et imperának.
5. Egy másik hasonlóság a divide et impera és a dinamikus programozás között, hogy mindkét esetben a gyökérben hirdetünk megoldást: a divide et impera a rekurzióból visszaérkezve ide, a dinamikus programozás iteratívan felérkezve ide. Tehát, míg az egyik algoritmus alapvetően rekurzív, a másik alapvetően iteratív.

2. Mohón vagy dinamikusan?

Ha tisztán látjuk a két stratégia közötti alapvető hasonlóságokat és különbségeket, akkor ez segíteni fog abban, hogy felismerjük, mikor célszerű alkalmazni őket, és el fogjuk kerülni az alábbi tévedéseket is:

- Dinamikus programozást alkalmazunk, bár a mohó megközelítés is kielégítő lenne.
- Mohó algoritmust használunk ott, ahol dinamikus programozásra lenne szükség.

A mohó és dinamikus programozási stratégiák váll váll mellett:

1. Általában mindkét technikát optimalizálási feladatok megoldására használjuk.
2. Mind a mohó, mind a dinamikus programozási stratégia esetében a megoldást egy optimális döntéssorozat jelenti.
3. Míg az első technika egyetlen döntéssorozatot állít elő (bízva abban, hogy ez lesz az optimális), addig a második több (optimális) részdöntéssorozatot is generál, amelyekből majd felépíti az eredeti feladatot megoldó (optimális) döntéssorozatot. Ez a különbség abból adódik, hogy a mohó algoritmus mohó

döntések sorozata által, a dinamikus programozás pedig az optimalitás alapel-alapelve szerint építkezve oldja meg a feladatot.

4. A fenti megállapítással összhangban, a mohó stratégia fentről lefelé, a dinamikus programozás pedig lentről felfelé oldja meg a feladatot. A mohó algoritmusok mindig a döntési fa gyökerétől a levelei felé haladnak, és minden mohó választással a feladatot kisebb méretű hasonló feladattá redukálják, míg triviálissá nem válik. A dinamikus programozás esetében a lentről felfelé jelenthet mind levelek-gyökér, mind gyökér-levelek irányt. Az első esetben a triviális részfeladatokat ábrázoló levelektől indulva, felépítjük az egyre bonyolultabb részfeladatok optimális megoldásait, végül pedig – felérkezve a gyökérbe – az eredeti feladatnak, mint legnagyobb feladatnak az optimális megoldását. A második esetben a gyökér képviselte kezdeti állapothoz tartozó triviális részfeladatból indulunk. Nem rendelkezvén kellő információval ahhoz, hogy mohó döntést hozzunk, regisztráljuk a terebélyesedő fa koronáján megjelenő összes – egymástól különböző – csomóponthoz (amelyek egymástól különböző állapotokat képviseltek) vezető optimális döntéssorozatot, mint az illető csomóponthoz tartozó részfeladat optimális megoldását. Egyre több és egyre bonyolultabb részfeladatot oldva meg, végül „felérkezünk" a döntési fa leveleibe. Miután kiválasztjuk az optimális levelet, az ide vezető gyökér-levél út képviseli az eredeti feladatnak, mint legnagyobb részfeladatnak az optimális megoldását.
5. A két technika alkalmazása más-más komplexitású algoritmust eredményez. Tegyük fel, hogy az illető feladathoz rendelhető fa magassága n. Döntési fáról lévén szó, a fa össz-csomópontjainak száma nyilván exponenciálisan függ n-től. A mohó-stratégia alkalmazása lineáris algoritmust (O(n)) eredményez, hiszen egyetlen gyökér-levél utat jár be. Bár a dinamikus programozás általában nem tudja elérni ezt a komplexitást, ha az egymástól különböző részfeladatok száma polinom függvény szerint függ n-től, akkor algoritmusa polinomiális lesz.
   Megjegyzés: Itt a stratégiából adódó komplexitást vizsgáltuk. Ez a komplexitás nőhet még, attól függően, hogy az egyes döntések meghozatala milyen komplexitású plusz feladattal jár.
6. Mindkét technika valamilyen mértékben az optimalitás alapelvére támaszkodik. A dinamikus programozás algoritmusok teljesen erre az alapelvre épülnek. A mohó technika esetében viszont csak szükséges feltétele annak, hogy a feladat megoldható legyen mohó döntéssorozat által. A mohó algoritmusok a mohó-választás alapelvére épülnek. Tehát, míg a dinamikus programozás teljesen kihasználja az optimalitás alapelvét, a mohó-technika csak részlegesen (a módszer helyességének bizonyításában).

Kátai Zoltán,
Sapientia-EMTE, Matematika-informatika Tanszék, Marosvásárhely

A fűszerekről