Az algoritmustervezési stratégiák bemutatkoznak

Moderátor: Meghívtunk öt algoritmustervezési stratégiát egy képzeletbeli *talkshow*-ra, hadd mutatkozzanak be ők maguk. Vitaindító feladatként az alábbit választottuk:

Háromszög: Egy *n* soros négyzetes mátrix főátlóján és főátló alatti háromszögében természetes számok találhatók. Feltételezzük, hogy a mátrix egy *a* nevű kétdimenziós tömbben van tárolva. Határozzuk meg azt a „legrövidebb" utat, amely az *a*[1][1] elemtől indul és az *n*-edik sorig vezet, figyelembe véve a következőket:

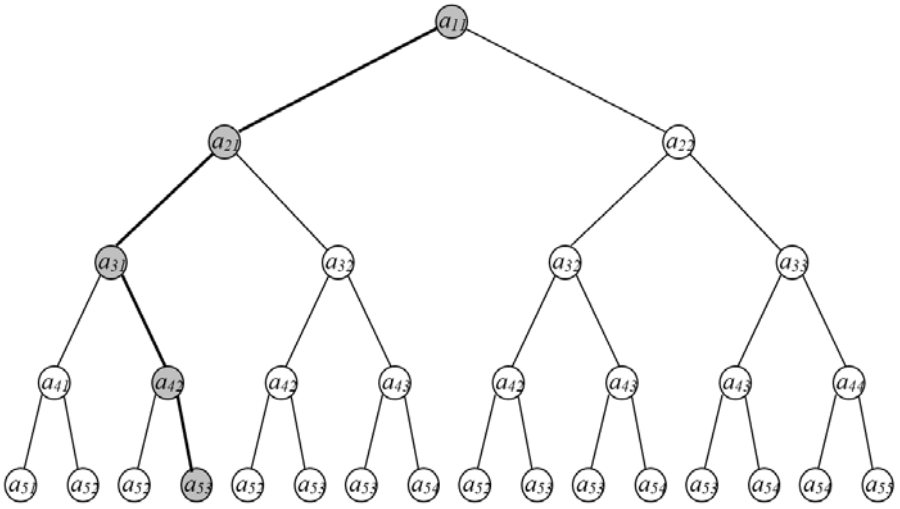
- egy úton az *a*[*i*][*j*] elemet az *a*[*i*+1][*j*] elem (függőlegesen le) vagy az *a*[*i*+1][*j*+1] elem (átlósan jobbra le) követheti, ahol $1 \leq i < n$ és $1 \leq j < n$.
- egy út „hossza" alatt az út mentén található elemek összegét értjük.

Az 1.1 ábrán látható mátrixban a csúcsból alapra vezető „legrövidebb" utat besatíroztuk.

7				
9	5			
1	99	4		
21	7	33	17	
2	15	8	3	1

1.1 ábra. *Példa-mátrix a háromszög feladathoz. A besatírozott legjobb út hossza 32.*

Elemezve a feladatot, észrevehetjük, hogy egy olyan optimalizálási problémáról van szó, amelyben az optimális megoldáshoz n-1 döntés nyomán juthatunk el, és mindenik döntésnél 2 választásunk van (melyik irányba lépjünk tovább, függőlegesen le vagy átlósan jobbra). Minden döntéssel a feladat egy hasonló, de egyszerűbb feladatra redukálódik. Tehát a feladat megoldásterét az 1.2 ábrán látható bináris fa képezi.



1.2 ábra

A példa-mátrix mögött megoldástérként meghúzódó bináris fa.

Az optimális megoldás meghatározása az optimális döntéssorozat megtalálását jelenti. Úgy is mondhatnánk, hogy meg kell találnunk a feladat megoldásterét képező fastruktúra 2^{n-1} darab – gyökértől levélhez vezető útja közül a „legjobbat". Más szóval, meg kell keresnünk a fa „legjobb levelét", azt, amelyikhez a „legjobb út" vezet.

A megoldástér egy alaposabb vizsgálata további észrevételekhez vezethet el:

1. A fa csomópontjainak száma, $1+2+2^2+...+2^{n-1}=2^n-1$. Ez azt jelenti, hogy bármely algoritmus, amely generálja/bejárja a teljes fát ahhoz, hogy megtalálja az optimális utat, exponenciális bonyolultságú lesz.
2. Amíg a fa a teljes feladatot képviseli, addig a részfái azokat a hasonló de egyszerűbb részfeladatokat (a levelek a triviális részfeladatokat), amelyre ez lebontható. Konkrétan: az a_{ij} gyökerű részfa, az a[i][j] elemtől az alapra vezető „legrövidebb út" meghatározásának feladatát ábrázolja.
3. A fenti ábra azt is kiemeli, hogy különböző döntéssorozatok azonos részfeladatokhoz vezethetnek, ami azt jelenti, hogy a fának vannak azonos részfái. Nem nehéz átlátni, hogy a különböző részfeladatok száma azonos a mátrix elemeinek számával, azaz $n(n+1)/2$. Tehát az algoritmus, amelynek sikerül elkerülni az azonos részfeladatok többszöri megoldását, négyzetes bonyolultságú lesz.

E rövid felvezető után engedjük szóhoz meghívottainkat. Hadd halljuk, ki hogyan közelíteni meg a felvetett feladatot.

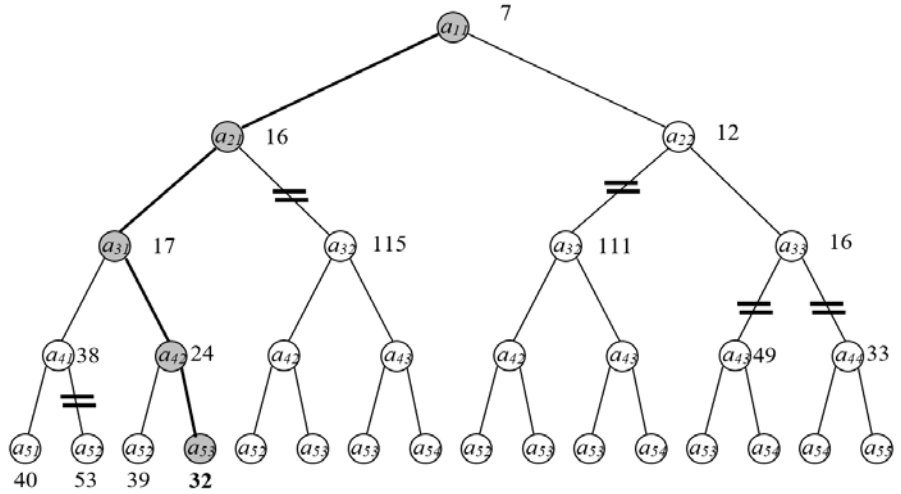
Greedy: Jelszavam, hogy *„élj a mának!”*. Ha minden napból sikerül a legtöbbet kihoznom, akkor remélhetőleg a lehető legtartalmasabb életem lesz. Jelen esetben ez azt jelenti, hogy indulva a csúcsból, minden lépésben a kisebbik elem irányába lépek tovább. Elvégre a legkisebb összegben vagyunk érdekeltek. (A példa-mátrix esetében a mohó-út 34 hosszú lesz, és ez a következő: (1,1),(2,2),(3,3),(4,4),(5,5)).

Backtracking: Greedy *„élj a mának!”* filozófiája nem eredményez optimális megoldást, ha a per-pillanat legígéretesebb választás elvág jövőbeni „nagy lehetőségektől”, vagy elkerülhetetlenné tesz későbbi buktatókat (az 5-ös érték mohó elválasztásával beleszaladt a 99, 33, 17 értékek képezte „gátvonalba”). Ezért az én elvem – biztos, ami biztos alapon – az, hogy *„legbizonyosabban úgy találod el a verebet, ha ágyúval lősz a fára”*. Más szóval, generálom sorra az összes csúcsból alapra vezető utat, és kiválasztom közülük a „legjobbat”. Elsőnek az (1,1),(2,1),(3,1),(4,1),(5,1) utat állítom elő, majd az (1,1),(2,1),(3,1),(4,1),(5,2) utat, és így tovább. Utolsóként generálom az (1,1),(2,2),(3,3),(4,4),(5,5) utat. Így nem csúszhat ki az ujjaim közt az optimális megoldás.

Branch and bound: Lenne egy javaslatom, hogy miként tudna Bracktracking gyorsítani algoritmusán. Ha a kurrens út hossza *már több*, mint az addig talált legjobb úté, akkor értelmetlen az illető irányba folytatni az útgenerálást. Az 1.3 ábrán bejelöltük, hogy mely részfák metszhetők le a generálandó fáról (a csomópontok mellett a csúcsból odavezető út hossza lett feltüntetve; 53>40, 115>32, 111>32, 49>32, 33>32). Ebben a gondolatmentben kívánatos találni minél előbb egy minél jobb megoldást, ha nem is garantáltan az optimumot. Ha az épülő fa koronájának mindig a legkisebb részösszegű csomópontjától ágazunk tovább, akkor – jó eséllyel – még jobban meg tudjuk metszeni a fát (1.4 ábra). A koronát képező csomópontok (részösszeg szerint csökkenő sorrendben), lépésről lépésre, a következők:

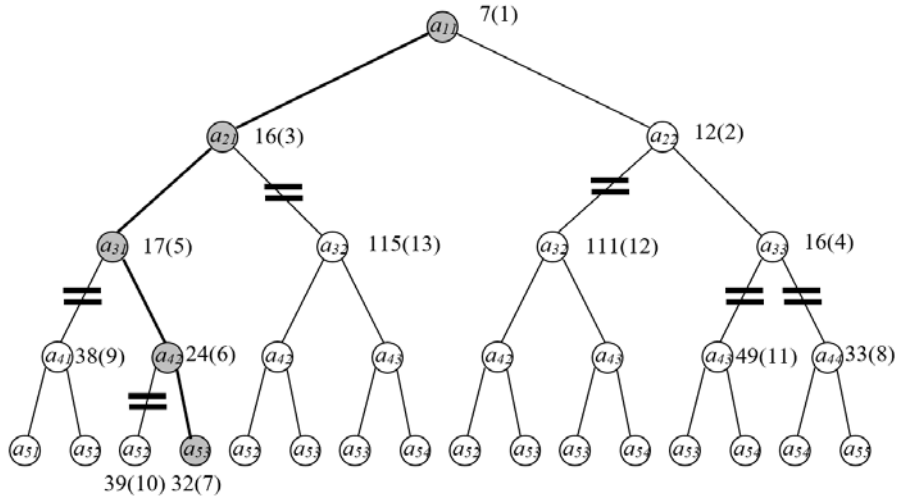
1. {}
2. {(1,1,**7**)}
3. {(2,2,**12**), (2,1,**16**)}
4. {(2,1,**16**), (3,3,**16**), (3,2,**111**)}
5. {(3,3,**16**), (3,1,**17**), (3,2,**111**), (3,2,**115**)}
6. {(3,1,**17**), (4,4,**33**), (4,3,**49**), (3,2,**111**), (3,2,**115**)}
7. {(4,2,**24**), (4,4,**33**), (4,1,**38**), (4,3,**49**), (3,2,**111**), (3,2,**115**)}
8. {(5,3,**32**), (4,4,**33**), (4,1,**38**), (5,2,**39**), (4,3,**49**), (3,2,**111**), (3,2,**115**)}
9. {(4,4,**33**), (4,1,**38**), (5,2,**39**), (4,3,**49**), (3,2,**111**), (3,2,**115**)}
10. {(4,1,**38**), (5,2,**39**), (4,3,**49**), (3,2,**111**), (3,2,**115**)}
11. {(5,2,**39**), (4,3,**49**), (3,2,**111**), (3,2,**115**)}
12. {(4,3,**49**), (3,2,**111**), (3,2,**115**)}
13. {(3,2,**111**), (3,2,**115**)}
14. {(3,2,**115**)}
15. {}

Minden lépésben a sor elsőt helyettesítettük a fiaival (feltéve, ha a megfelelő részösszeg nem már nagyobb, mint az addig talált legjobb megoldás).



1.3. ábra

Branch and bound ötlettel javított Backtracking algoritmus metszette fastruktúra. A csomópontok mellet feltüntettük a megfelelő részösszegeket (a csúcsból oda vezető út hosszát).



1.4. ábra

„Best first" ötletre épülő Branch and bound algoritmus metszette fastruktúra. Zárójelben feltüntettük, hogy milyen sorrendben hajtanak ágakat (vagy hajtanának ágakat, ha nem bizonyulnának száraz iránynak) a növekvő fa koronáját képező csomópontok.

Divide et impera: Az én megközelítésemet már a római császárok is használták: *„oszd meg és uralkodj”*. A kurrens részfeladatnak (kezdetben az eredeti feladatnak), mint apafeladatnak, a megoldását visszavezetem a fiúrészfeladatai megoldásaira (egy-egy rekurzív hívás által; „oszd meg” fázis), majd e fiú-megoldásokból felépítem az apafeladat megoldását „uralkodj” fázis). Ha a kurrens apafeladatnak az (i,j) pozícióból alapra vezető legjobb út problémáját tekintjük, akkor ennek fiú-részfeladatai az (i+1,j) es (i+1,j+1) pozíciókból alapra vezető legjobb utak problémái. A stratégiám alapjául szolgáló rekurzív képlet nyilvánvalóan a következő: legjobbút(i,j) = a[i][j] + min{legjobbút(i+1,j), legjobbút(i+1,j+1)}, ahol $1 \leq i < n$.

Dinamikus programozás: A Divide et impera megközelítéssel két gondom is van. Először is csak a legjobb út hosszát szolgáltatja és nem magát az utat is. Még súlyosabb gond, hogy mivel nem tart nyilvántartást a már megoldott részfeladatokról, azonos részfeladatokat többször is megold. Például, a „(3,2) részfeladatot” kétszer oldja meg: egyszer a „(2,1) részfeladat” jobbfiú-részfeladataként, majd a „(2,2) részfeladat” balfiú-részfeladataként is. Én egy másik tömbben (például c[1..n][1..n]) eltárolnám minden részfeladat első példányának megoldását (a megfelelő legjobb út hosszát), és ha újra találkoznék ugyanazzal a részfeladattal, akkor csak elővenném a megoldását.

A Branch and bound megközelítés is javítható az alapötletemmel: ha ugyanazon cella többszörösen kerülne a növekvő fa koronájára, csak attól a példánytól ágazzunk tovább, amelyhez tartozó részösszeg a legkisebb (a többit nyugodtan tekinthetjük száraz iránynak). Persze, ez megint csak azt feltételezi, hogy nyilvántartást vezetünk. Jelen feladat esetében, az elsőként koronára kerülő példány részösszege lesz a minimális.

A Divide et impera-t javító ötletem iteratív megközelítésben így foglalható össze: indulva a triviálisan egyszerű részfeladatok ((n,j) alakú részfeladatok, j=1,n) nyilvánvaló megoldásainak szintjéről, a már rendelkezésre álló (eltárolt) fiú-részmegoldásokból felépítjük (a rekurzív képlet alapján) az aparészfeladatok megoldásait (utolsóként az eredeti feladatét) (1.5 ábra)

c[n][j] = a[n][j], j=1..n

c[i][j] = a[i][j] + min{c[i+1][j], c[i+1][j+1]}, i=n-1, n-2, ..., 1; j=1..i

A c tömbből azonnal adódik a legjobb út is: a csúcsból alapra vezető mohó vonal.

Ugyanez Branch and bound irányból (a gyökértől a levelek irányába) is megfogalmazható (1.6 ábra):

c[1][1] = a[1][1]

c[i][j] = a[i][j] + min{c[i-1][j], c[i-1][j-1]}, i=2, 3, ..., n; j=2..i-1

c[i][1] = a[i][1] + c[i-1][1], i=2, 3, ..., n

c[i][i] = a[i][i] + c[i-1][i-1], i=2, 3, ..., n

32				
25	27			
16	114	22		
23	15	36	18	
2	15	8	3	1

1.5 ábra

Levelek-gyökér irányú dinamikus programozással feltöltött optimális részmegoldások-tömb.

7				
16	12			
17	111	16		
38	24	49	33	
40	39	32	36	34

1.6 ábra

Gyökér-levelek irányú dinamikus programozással feltöltött optimális részmegoldások-tömb.

A moderátor összegzése: A Greedy javasolta algoritmus a leggyorsabb (lineáris idő bonyolultságú), de nem garantál optimális megoldást. Backtracking és Divide et impera biztosítják ugyan a legjobb megoldást, de algoritmusaik exponenciális bonyolultságúak. „Szerencsétlen esetekben" még a „Branch and bound javításokkal" is marad az exponenciális bonyolultság. Ennél a feladatnál a pálmát a dinamikus programozás viszi el, hiszen képes polinomiális időben (négyzetes időbonyolultság) optimális megoldással szolgálni.

Kátai Zoltán