Algoritmusok és bonyolultságuk MSc hallgatók számára

Algoritmuselméleti alapok

2019. Előadó: Hajnal Péter

1. Az algoritmus naív fogalma

Egy feladat algoritmikus megoldása egy eljárás, ami az adatok megkapása után egy jól definiált lépéssort elvégezve kiszámolja a feladatra adandó választ. Minden lépésben egy mindenki számára könnyen elvégezhető elemi utasítást kell végrehajtani.

Az algoritmus fogalmával együtt kialakult egy az algoritmusokkal kapcsolatos nyelvezet is. Az adatokat *inputnak*, az eredményt *outputnak* nevezzük. Ha adott inputon az algoritmus utasításait követve végezzük az előírt lépéseket, akkor az *algoritmus futásáról* beszélünk.

Már az általános iskolában tanulunk algoritmusokat. Az alapműveletek elvégezésének szokásos módja is egy-egy algoritmus, ahol az elemi lépések a számjegyműveletek. Ezért is kezdik a számtan oktatást a szorzótábla memorizálásával. Az alapszerkesztések, a prímtényezőkre bontás megtanított módja, az Euklideszi-algoritmus mind jól ismertnek kell lenni egy érettségizett számára.

Legyen $\mathcal{I}$ az inputok, $\mathcal{O}$ az outputok halmaza. Tehát egy algoritmikus/számítási probléma egy $f : \mathcal{I} \to \mathcal{O}$ függvény.

Az algoritmus fenti leírása nem matematikai definíció. Ezen a ponton nem is adunk matematikai definíciót az algoritmusra. Megjegyezzük, hogy a matematikus közösség, hosszú vajúdás után az 1930-as években fogadott el egy mind a mai napig használt algoritmus fogalmat. Ez jól leírja azt ami a számítógépek használata közben történik. Azt is megjegyezzük, ha megjelennének a kvantum-számítógépek, akkor ezt a fogalmat újra kellene értékelnünk.

Az alábbiakban egy kissé pontosítjuk, hogy egy algoritmuselméleti probléma mikor van jól definiálva, mikor tudjuk matematikailag alapozottan vizsgálni.

1.1. Számítási problémák, elemi lépések

Legtöbbször azonban nem vagyunk ennyire formálisak. A FAKTORIZÁCIÓ például egy probléma. A szóhasználat jelentheti a következők bármelyikét.

Példa (FAKTORIZÁCIÓ I). Input: egy pozitív egész szám. Output: prím osztóinak listája a megfelelő multiplicitásokkal.

Példa (FAKTORIZÁCIÓ II). Input: egy pozitív egész szám. Output: egy prím osztója.

Példa (FAKTORIZÁCIÓ III). Input: egy pozitív egész szám. Output: legkisebb prím osztója.

Példa (FAKTORIZÁCIÓ IV). Input: egy pozitív egész szám és egy t érték. Döntsük el van-e 2 és t közötti osztó.

Bármelyik megoldása átalakítható (alap programozási technikák, például iteráció/rekurzió, bináris keresés ismeretével) a többi megoldásává.

Az inputok és outputok leírásában/leírtak értelmezésben is meg kell egyezniük a számítást követőknek. (Gyakorlatban egy géppel kell „közölnünk" az adatokat/inputot, az output kiszámolása után pedig a gép által adott végeredményt kell „értelmeznünk".)

Hogy az adatok, hogyan kódoltak az kérdése, és néha nem is olyan fontos.

Példa. Adott n valós szám, határozzuk meg rendezett sorrendjüket.

A valós számok halmaza kontinuum számosságú. Egy valódi számítógép véges 0-1 sorozatokat képes tárolni. A kódok egy megszámlálhatóan végtelen halmazt alkotnak. Ez „túl kicsi" az összes valós szám kezelésére. Ennek ellenére ez egy fontos feladat. Az elemi lépést az „két szám összehasonlításának" vehetjük. Egy futás hossza az lesz, hogy a rendezett sorrend hány összehasonlítás után lesz ismert. A futás hossza függ a számok számától. Hosszabb számsorozat esetén több összehasonlítást várunk. Az n paraméter az input mérete.

Az, hogy milyen szám ábrázolással dolgozunk, hogy ebben az összehasonlítás hogyan valósítható meg nem érdekel minket. Ahogy a gyakorló programozó számára se mindig ismert/fontos, hogy a gépen belül mi történik egy programja futása során.

Egy teljesen más probléma a következő:

Példa. Adott n darab k bites szám, határozzuk meg rendezett sorrendjüket.

Itt a szövegezésből nyilvánvaló, hogy az input mérete $n \cdot k$ bit és bit műveletekben kell gondolkoznunk mint elemi műveletek.

Példa. Adott két egész szám, számoljuk ki összegüket/különbségüket/szorzatukat.

Az elemi lépés a számjegyek közötti elemi műveletek, a szorzótábla kétjegyű végeredményeinek továbbviteli jegyre/utolsó számjegyre való szétszedése.

Példa. Adott két természetes szám, számoljuk ki legnagyobb közös osztójukat.

Az elemi lépések az egészek közötti összehasonlítás/összeg/különbség képzés esetleg szorzás, maradékos osztás. Azaz most két szám összege/különbsége egyetlen elemi lépésnek számít.

Ha valakit ez zavar, akkor az input legyen bitsorozat és a korábbi $x \leftarrow x - y$ egyetlen elemi lépést helyettesítse az előző példa algoritmusával. Az elemi lépések számolása — amit az *algoritmus analízisének* nevezünk — a két analízisből (euklidészi

algoritmus valós számok szintjén történő analízise és az alapműveletek bit szintű analízise) „összerakható”.

Érdekes módon az input méretében viszont fontos, hogy mekkorák a számok. Azaz nem 2, hanem mondjuk a nagyobb szám szájegyszáma lehet egy jó mérték. A kérdés, hogy hány elemi lépés kell két n jegyű természetes számon az euklideszi algoritmus futtatásánál.

Példa. Adott $A, B \in \mathbb{R}^{n\times n}$ két mátrix. Számítsuk ki szorzatukat.

Ismét „pontos valós aritmetikát sugallunk”. Ez a való életben nem lehetséges, habár írhatunk programot, amiben két valósnak deklarált számot összeszorozhatunk. A program futása során lehet, hogy egy memória területen egy kerekített érték tárolódik és a szorzás után a szorzat a kerekített értékek szorzatának kerekített értéke lesz. Azonban elképezelhetünk egy olyan számítógépet, amiben egy bit tárolására alkalmas memória egység helyett egy doboz szerepel képzeletünkben. Ezekbe a dobozokba valós számokat rakhatunk és a szorzás művelet után két valós szám pontos szorzata kerül a szorzatot tároló dobozba. Ennek a képzeletbeli gépnek a futtatása és analízise nagyon természetes. Nézhetjük, hogy a mátrix szorzás definíciója szerint hány szorzás, összegzés történik. Ez az analízis nem a valóságtól elrugaszkodott, habár egy elméleti géppel dolgozik. Nagyon fontos a mátrix szorzás fenti típusú analízise.

A fenti példában az input mérete is fontos. Két mátrix adott, de az input méretét 2-nek venni csalás lenne. A természetes megállapodás, hogy valós számok „seregének” gondoljuk az inputot, $2n^2$ számra gondolunk A, B helyett. Az sem nagy csalás, ha az n paraméterrel (sorok/oszlopok közös száma, az input négyzetes mátrixok mérete) jelöljük mekkora feladatról van szó.

Példa. Adott egy egyszerű gráf határozzuk meg hány háromszög (három hosszú kör) van benne.

Hogyan adott egy n csúcsú egyszerű gráf? Több (mondhatjuk sok) lehetőség van. Csak kettőt emelünk ki.

Első megadási módszer lehet a szomszédsági mátrix (egy $n \times n$ bit-mátrix) megadása. Egy kicsit pazarló módszer, hiszen a mátrix főátlóján nullák szerepelnek és szimmetrikus. Azaz igazából $\binom{n}{2}$ független bit a kódolás. Ennek ellenére a mátrix struktúra, algebra gyakran nagyon hasznos.

Egy másik lehetőség, hogy a csúcsok egy teljes listájának minden v eleméhez hozzátartozik a v csúcs szomszédjainak egy listája (egy s szomszéd igazából a vs élt reprezentálja). A szomszédok listáiban minden szomszéd tartalmazza a `következő-szomszéd` infot, ami erre a szomszédra mutat (amennyiben ez a szomszéd létezik, amennyiben az utolsó szomszédról beszélünk mondjuk egy speciális NIL értéket vesz fel). A teljes csúcs-listájában minden csúcshoz tartozik egy `első-szomszéd` info, ami lehet NIL is ha a csúcs izolált. Persze ott van a `következő-csúcs` info is, ami a teljes csúcs-lista utolsó csúcsánál NIL értékű. Persze lehetnek további információk is a struktúrában.

Például egy élsúlyozott gráf esetén a v csúcs s szomszédjánál szerepelhet a vs él súlya (`súly`), ami mondjuk egy pozitív egész. Így a gráf összetett adatok bonyolultan kapcsolt összessége.

A két ábrázolás más és más elemi műveleteket sugall és ugyanannak a magas szinten leírt algoritmusnak a megvalósítása, analízise teljesen más kérdéseket vet fel.

Lehet, hogy a hallgatót a fenti példák összezavarják. Nem ez volt a cél. A cél, hogy egy algoritmikus problémát jól át kell gondolni. Ha többen tárgyaljuk, akkor sokat kell kérdezni, tisztázni, hogy egy matematikailag megalapozott tárgyalás alakuljon ki.

2. Algoritmusok analízise

Látni fogjuk, hogy gyakran nagyon sokféle algoritmus adható ugyanarra a problémára. Melyik a jobb, mi melyiket használjuk adott problémára, van-e legjobb algoritmus? Van-e egy elméleti határ, amit semelyik algoritmus nem tud átlépni? Ezek nagyon természetes, központi kérdések.

Korábban már használtuk az algoritmus analízisének kifejezését: adott ω inputon futtatva az algoritmust meg kell számolnunk (néha megbecsülnünk) hány elemi lépés vezet el az outputhoz. Legyen $\mathcal{A}$ egy algoritmus, ω egy input. $t_{\mathcal{A}}(\omega)$ az elemi lépések száma ami szükséges, hogy ω inputon futtatva $\mathcal{A}$-t megkapjuk az outputot. Használtuk az input méretének fogalmát is. Azaz $\mathcal{I} = \cup_{s=0}^{\infty}\mathcal{I}_s$, ahol $\mathcal{I}_s$ az s méretű inputok halmaza.

Definíció.

$$t_{\mathcal{A}}(n) = \max\{t_{\mathcal{A}}(\omega) : \omega \in \mathcal{I}_n\}.$$

A fenti definíció nagyon fontos. Amikor az ebben definiált függvényt vizsgáljuk — például egy jó felső becslést adunk rá — akkor azt mondjuk a *legrosszabb eset analízist* végezzük el. Valóban, ha az $\mathcal{I}_n$ inputokat vizsgáljuk, akkor a maximum vétele a legtöbb elemi lépés számot veszi amit ezen inputok között végre kell hajtanunk az output meghatározásához. Azaz a felső becslés egy olyan „garancia”, ami az input hosszának függvénye.

⋆

Egy fontos megjegyzés. Egy pontos analízis nagyon hosszú, összetett, áttekinthetetlen formulákhoz vezet. Másrészt az analízis az algoritmus futásához szükséges időt írja le. De milyen mértékegységben? Ha másodpercben mérnénk a futást, akkor az elemi lépések száma mellett egy szorzó szerepel, ami az elemi lépésekhez szükséges valódi idő. Ez a szorzó függ a gépünk hardware-étől/paramétereitől. Egy öt éves gép lecserélese egy maira jelentősen befolyásolja ezt a szorzót. Így az analízis első fázisában a konstans szorzók nem lényegesek. Általában elhanyagoljuk. Hogy ezt megtehesük/formalizáljuk néhány fontos matematikai jelölésre van szükség.

Definíció. Legyen $t, f : \mathbb{N} \to \mathbb{R}$. Ekkor $t = \mathcal{O}(f)$ azt jelenti, hogy alkalmas $c > 0$ konstansra és n_0 küszöbértékre fennáll, hogy $n > n_0$ esetén

$$|t(n)| \leq cf(n)$$

$t(n)$ az analízált idő, ez általában egy nemnegatív értékeket felvevő függvény. $f(n)$ egy „egyszerű" függvény, mint n, n^2, n^{10}, $n \log n$, 2^n, n^n, 2^{n^2}. (Álljunk meg. Az $f(n) = n \log n$ példában nem írtam oda a logaritmus alapját. Melyik alapra gondoltam? Számit a pontos érték?)

Példa.

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \mathcal{O}(n^6).$$

$n \log n$ tényleg $\mathbb{N}$-en értelmezett? A küszöbérték szereplése miatt nem nagyon zavar, hogy 0-ban nem értelmezett. Mondhatnánk, hogy kicsire nem adunk, de ez nem matematikusi hozzáállás. Az átláthatóság és pontosság között kell egyensúlyozni egy olyan területen, ahol nagyon sok NEM matematikus dolgozik. Ezért sokszor a matematikus szemüveg sok végiggondolnivalót/pontosító megjegyzést hagy az olvasó számára. Kérdezzünk, konzultáljunk ...

Természetesen $n^2 = \mathcal{O}(2^n)$. De egy négyzetes függvényt exponenciálissal becsülve nagyon hanyagok vagyunk. További jelölések segítik azt, hogy a pontos nagyságrendet kiemeljük.

Definíció. Legyen $t, f : \mathbb{N} \to \mathbb{R}$. Ekkor $t = \Omega(f)$ azt jelenti, hogy alkalmas $c > 0$ konstansra és n_0 küszöbértékre fennáll, hogy $n > n_0$ esetén

$$cf(n) \leq t(n).$$

$f(n)$ mindig egy egy idő után pozitív függvény lesz. Végül

Definíció. Legyen $t, f : \mathbb{N} \to \mathbb{R}$. Ekkor $t = \Theta(f)$ azt jelenti, hogy alkalmas $c, c' > 0$ konstansra és n_0 küszöbértékre fennáll, hogy $n > n_0$ esetén

$$cf(n) \leq t(n) \leq c'f(n)$$

Példa.

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \Theta(n^6).$$

Ha a nagyságrendet leíró függvény mellöl nem szeretnénk szönyeg alá söpörni a konstanst, akkor új jelölések szükségesek.

Definíció. $t(n) \sim f(n)$, azt jelenti, hogy $\lim_{n\to\infty} t(n)/f(n) = 1$.

Másképpen $t(n) \sim f(n)$, azt jelenti, hogy $t(n) = f(n) + o(f(n))$, ahol $o(f(n))$ azt mondja a jelölt függvényről, hogy $f(n)$-nel osztva 0-hoz tart, ha n tart a végtelenbe. Elég nagy n esetén a $o(f(n))$-nel jelölt maradéktag, $f(n)$-hez képest elhanyagolható.

Példa.

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 \sim \frac{18}{6!}n^6,$$

még pontosabban

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \frac{18}{6!}n^6 + \mathcal{O}(n^5).$$