Algoritmusok lépésszáma

Egy fejlesztő életében nagyon fontos, hogy meg tudja becsülni egy algoritmus hatékonyságát, illetve (viszonylag) hatékony algoritmust tudjon írni, de legalábbis ne írjon a szükségesnél sokkal rosszabbat.

Processzorok sebessége

A számítógépek sebessége olyan gyorsan nő, mi szükség van erre?

Már nem nő annyira gyorsan.

A processzorok utasításvégrehajtó sebességét két dolog határozza: egy utasítást mennyi idő alatt hajt végre és hogy hány utasítás hajtható végre rajta egyszerre

- Az egy utasítás végrehajtási idejének csökkentése komoly gondolkodást igényel, és egy elméleti minimum alá egyszerűen nem vihető.
- Az időegység alatt végrehajtható utasítások számát
    - régebben a processzor órajelének emelésével növelték (az órajel olyasmi, mint egy metronóm: a processzorban minden ezen ütem szerint történik. Kétszer gyorsabb metronómsebességnél kétszer több hangjegyet lehet lejátszani a kottából. A helyzet az, hogy 4-5 gigahertz-nél (4-5 milliárd ütés másodpercenként) ezt nem sikerült eddig feljebbvinni. Ráadásul az energiafelhasználás ennek az ütemnek a négyzetével nő (kétszer gyorsabb órajel négyszerannyi energiába kerül), ami nem feltétlenül az energia miatt fontos, hanem amiatt, hogy az energiából végső soron keletkező hőt el kell valahogy vezetni. Ha nem sikerül, akkor elég a processzor.
    - manapság pedig azzal emelik, hogy több magot (kb. processzort) integrálnak egy chipbe. Hátulütője, hogy egy feladatot nem lehet teszőlegesen több, párhuzamosan végrehajtható részre bontani, így a végrehajtás sebességét ez elméleti szinten limitálja. A több mag egyébként a szerverprocesszoroknál nagyon jó, amikor a világból érkező több, párhuzamos kérést kb. párhuzamosan végre tudja hajtani... legalábbis a számítás részét.

Ha megfigyelted az elmúlt éveket, tapasztalhattad Te is, hogy kb. 10 éve nem nagyon változott a processzorok órajele, maximum a az integrált magok száma nőtt.

Szumma szummárum, fontos, hogy elég jó algoritmusokat írjuk.

Algoritmusok minősége (aszimptótikus komplexitás)

Vegyünk egy jó és egy rossz algoritmust. Ha két adatra futtatjuk le őket, akkor nagy valószínűséggel mindkét algoritmus szemvillanásnyi idő alatt lefut. Ezen a szinten nincs különbség. Ha viszont nekieresztjük 1.000.000 adatnak, a jó algoritmus lehet, hogy még mindig szemvillanásnyi idő alatt lefut, míg a rossz algoritmusnak egy-két napra is szükség van, vagy még rosszabb esetben hónapokra vagy évekre. Vegyük csak példaként a lineáris és a bináris keresést. A lineáris keresés 1.000.000 adatban való keresésre átlagosan 500.000 összehasonlítást végez, míg a bináris keresés 20 összehasonlítást végez el.

Az algoritmusokat is az alapján osztályozzák, hogy egyre nagyobb elemszámú bemenetre hogyan viselkedik, azaz milyen függvényhez hasonlóan nő a futásidő (a lépésszám). Ebben a vizsgálatban a konstans szorzótényezőket elhagyják - a lépésszámnövekmény matematikai számítása közben kiesik a képletből.

Az egyes csoportokat tudományosan úgy is jelölik, hogy ordó (írott O betű) és mögötte zárójelben a függvény, pl (`O(logn)` vagy `O(n)` vagy `O(n^2)` - ordó-log-n, ordó-n, ordó-n-négyzet). Magyarul ordó a neve, angolul big-o-nak hívják.

Algoritmusok csoportosítása

- **logaritmikus algoritmusok** (`O(logn)`): pl. a bináris keresés. Ha a bemenetet megduplázzuk, akkor a lépésszám/futásidő azonos gépen mindig egy megadott értékkel nő meg. (Középiskolai érettségi rettegett témájával, a logaritmus azonosságaival ez ellenőrizhető). Vagy marad a gondolatkísérlet: az 1.000.000 elemű rendezett elemsorozatban való bináris keresés 20 lépésből eredményt hozott. Ha 2.000.000 elemünk van, 1-gyel több lépés kell. Ha 4.000.000 elemünk, akkor 2-vel, és így tovább. A duplázásra egy pici lépésszámnövekménnyel reagál az algoritmus. Ez a **legkiválóbb** osztály
- **lineáris algoritmusok** (`O(n)`): pl. lineáris keresés, melyben minden elemet (esetleg azok felét, harmadát) megnézzük, megvizsgáljuk. Ha a bemenet méretét megduplázzuk, akkor a szükséges lépésszám is duplázódik: kétszerakkora tömbben kétszerannyi elemet kell végigvizsgálni lineáris keresés esetén. Ez egy **jó** osztály
- **ordó-n-szer-log-n**: csomó rendezési algoritmus ilyen, elég közel van a lineáris osztályhoz, így hasonlóan **jó** osztály.
- **négyzetes algoritmusok** (`O(n^2)` - ordó-n-négyzet): két tömbből párokat alkotunk, buborékrendezést végzünk (minden elemet összehasonlítunk az elemek kb. felével, de a konstansszorzó nem számít), vagy pl. szorzótáblát készítünk. A bemenet méretének duplázása esetén

(10 helyett 20-ig készítjük a szorzótáblát) a lépésszám 4-szeresére változik. Ez még **elég jó** osztály.

- **nagyobb kitevőjű algoritmusok** (`O(n^x)` - ordó-n-valahanyadikon): ahogy haladunk az `x`-szel felfelé, 3,4,5..., egyre rosszabb.
- **exponenciális algoritmusok** (`O(2^n)` - ordó-2-az-n-ediken): A bemenet méretét jelentő `n` most a kitevőben van, ami azt jelenti, hogy n=1-re ha 2 időegység alatt végzünk, akkor n=2-re 4, n=3-ra 8, ... n=10-re már 1024, n=20-ra több, mint 1 millió... a logaritmikusnak pont a fordítottja.
  Jut eszembe az a legenda, mely szerint a sakk nagyon megtetszett egy indiai rádzsának, így meg akarta jutalmazni a játék kitalálóját, aki azt kérte, hogy a sakktábla első mezőjére rakjanak egy búzaszemet, a másodikra kettőt, a harmadikra négyet, és így tovább, minden mezőre az előzőnek a dupláját. A rádzsa - nem ismerve az exponenciális robbanás pokoli erejét - belement az alkuba, és elkezdte hordatni a búzát, míg végül ki nem derült, hogy annyi búza, amennyit az okos brahmin kért, nincs a birodalomban, sem a Földön, de még ha a Földön addig termett összes búzaszemet előkerítenék, az sem lenne elég a kérés teljesítésére. Hogy a brahmint ezután gazdagon megjutalmazták vagy lefejeztették tiszteletlenségért, arról már nem szól a fáma. További olvasnivaló: http://moricz.arrabonus.hu/static/nagykaroly/AL_2006.pdf
  Visszatérve az exponenciális algoritmusokra: nem tűnik soknak, az emberi elme nem is képes vele mit kezdeni, de elképesztő mértékben fel tud duzzadni a szükséges lépésszám.
  Még egy érdekesség ezzel kapcsolatban: Tegyük fel, hogy van egy exponenciális idejű algoritmusunk, melynek segítségével 15 egység hosszú bemenetet fel tudunk dolgozni mondjuk egy nap alatt (ami azt jelenti, hogy 18-hoz nem elég egy hét, 20 elemre kb. egy hónap kell, 24-re nem elég egy év!) És azt gondoljuk, hogy mivel már sokat fejlődött azóta az informatika, lecserélve a számítógépünket egy kétszer olyan gyorsra, majd sokkal nagyobb problémát kezelni tudunk. Az új számítógép rendszerbe állítása után a 15 egység hosszú bemenet kezelése fél nap, 16 egység hosszú problémáé 1 nap! Azaz rengeteg pénz befektetésével mindössze 1-gyel hosszabb bemenet kezelésére vagyunk képesek azonos idő alatt!
  A jelszavaink feltörését célzó ún. brute force algoritmusok ilyenek: egyszerűen végigpróbálgatják az összes lehetséges kombinációját az összes betűnek és számnak, amíg meg nincs a jelszó. Ha jelszavunk 1-2 karakter, gyorsan megy, ha 8-10, akkor már nem olyan gyors. Még egy történet, mellyel kapcsolatban nem találtam forrást, így csak az emlékeimre kell, hogy hagyatkozzak. *Annak idején egy bizonyos titkosítás feltörésére írt ki versenyt egy kriptográfiai cég. A feladat elvégzésére nemzetközi elosztott hálózat szerveződött, melybe mindenki betehette gépének teljesítményét. Az összefogás eredményeként néhány hónap alatt - brute force módszerrel - sikerült is megfejteni a kulcsot. A kódolás úgy emlékszem 60 bites volt.* Ha akár csak 1 bittel is megnöveljük a méretét, már duplaanyi a szükséges idő, ha 10-zel... Amúgy manapság 256 bit körüli kulcsokat használunk, azok megfejtése nyers erővel... hmmm...

Mit kell tudnom?

- Tudni meghatározni, hogy az éppen írt algoritmus lépésszáma hogyan növekszik a bemenet méretének függvényében. Ha pontosan nem is tudod, legalább valami halvány sejtésed legyen róla.
- Ismerni az egyes csoportokat, és ha szóbakerül, tudd, hogy mit kell alatta érteni.