Algoritmuselmélet és bonyolultságelmélet MSc hallgatók számára

Amortizációs analízis

2018. Előadó: Hajnal Péter

1. Alapfogalmak

Az eddig vizsgált algoritmusok analízise a legrosszabb eset analízisén alapultak. Egy inputon futtattuk és az input hosszának/méretének függvényében egy korlátot adtunk a futási időre.

Sokszor egy algoritmus összetett műveleteket ismétel sokszor. Ha csak egy összetett művelet legrosszabb eset analízisét végezzük el és ezt megszorozzuk a művelet elvégzésének számával, akkor „rosszul járunk". Jobb, ha a művelet átlagos lépésszámával dolgozunk. Ha az algoritmus sokszor végrehajt egy műveletet, akkor nem zavaró ha egyszer sok lépést végez el, míg máskor lényegesen kevesebbet. A teljes algoritmus futási ideje az átlagos lépésszámtól függ.

Ha a fenti „filozófia" szerint elemzünk egy ismételt műveletet, akkor amortizált analízisről beszélünk. A fogalmat legjobban példákon keresztül megismerni.

2. Példák

1. példa: Adott az $0^n = (0, 0, \ldots, 0)$ n-hosszú bitsorozat. Vehetjük úgy is mint a 0 szám kettes számrendszerbeli felírásának számjegyei. $2^n - 1$-szer növeljük meg a bitsorozat által kódolt természetes számot 1-gyel és a megnövelt szám n hosszú (esetleges kezdeti 0-kkal felduzzasztott) kettes számrendszerbeli felírását számoljuk ki. Minden növelés költsége az előző sorozathoz képest megváltoztatott bitek száma lesz. Mi a teljes algoritmus összköltsége?

A feladat nem túl érdekes, de elemzése tanulságos lesz. Ha sorozatunk $01^{n-1} = 011\ldots11$, akkor növelése után elérjük az $10^{n-1} = 100\ldots00$ sorozatot, Ennek költsége n, a lehető legnagyobb költség. Így a $2^n - 1$ növelés összeköltsége becsülthető $(2^n - 1)n$-nel.

Gondolkodjunk másképp. Vegyük észre, hogy minden változtatás a bitsorozatunk a záró 1-es blokkját alakítja át 0-sok blokkjává és az ezt megelőző egyetlen 0 bitet 1-essé írja át. (Ha a bitsorozat 0-ra végződik, akkor a záró 1-esek blokkja üres. Ha a záró 1-es blokk a teljes sorozat, akkor már nincs növelés/leállunk.) Azaz egy c költségű átírásban egy darab $0 \leftarrow 1$ átírás és $c - 1$ darab $1 \leftarrow 0$ átírás.

Tegyük fel, hogy egy számjegy megváltoztatásához valójában ki kell fizetnünk 1 \$-t. Ha a legrosszabb esetre készülne;nk fel, akkor n \$-t kell magunkkal vinnünk egy növeléshez. Ha azonban jó gazdasági érzékkel rendelkezünk, akkor 2 \$-ral boldogulunk. Az egyik \$-unk „0-ról 1-re írás" költsége, a másik \$-unk „1-ről 0-ra írás" költsége. Ha egy egy 0-t 1-esre írunk át, akkor a hozott 2 \$ egyikével ki tudjuk fizetni és az átírt 1-esnek ott tudjuk hagyni a második \$-t mind leendő átírási költséget. Általában igaz lesz, hogy aktuális bitsorozatunk minden 1-es bitjén szerepel egy §.

Mi van, ha az átírás nem csak az utolsó 0 megváltoztatása 1-re. Akkor sincs baj. A fenti gondolatmenet alapján számjegysorozatunk minden 1-ese a leendő 0-ra

írásának költségét „letétben tartja”. Az általános „update” a számjegyek egy végső 1-blokkjának 0-kkal való felülírása és az előtte álló 0-s 1-esé írása. Az 1-esk 0-ra írásának költsége ott van letétben, végül a blokk előtti 0 átírását a hozot 2 $-ból fedezzük, a maradék 1 $-t letétként hagyjuk az új 1-es számjegynél. Így a $2^n - 1$ művelethez $2(2^n - 1)$ $-ra van szükségünk. Ez fedezi az összes költséget. Sőt az algoritmus végén ott van $1^n = 11\ldots11$ szám mindegyik 1-esén 1 $ letét. Tehát a számolási költség pontosan $2(2^n - 1) - n$.

Összefoglalva. Minden növelést 2 költséggel számoltunk el. Tettük ez annak ellenére, hogy minden második növelésünk 1 költségű volt. Ezekben a növelésekben a költség felhasználása az átírásért való fizetés és letét képzése. Általában a korábbi letétekkel és a hozott 2 költséggel gazdálkodtunk (fizettünk átírásainkért és újabb letétet képeztünk. A növelés *amortizációs költsége* 2. A fenti interpretációt a *bankár értelmezésének* nevezzük.

Van egy másik értelmezés is, a *fizikus értelmezése*: Az aktuális szám 1-eseinek számát egy potenciálnak fogjuk fel. Tehát kezdetben 0 potenciálról indulunk. Minden lépésben vizsgáljuk a potenciál változását.

Definiálunk egy amortizált költséget. Ezt absztrakt környezetben írjuk le. Tegyük fel, hogy algoritmusunk egy $\mathcal{K}$ konfigurációi között „sétál”. Egy kiinduló κ_0 konfigurációból indul, majd $e_1, e_2, \ldots, e_L$ összetevőket végez el. e_i során κ_{i-1} konfigurációból κ_i konfigurációba érkezik. Az algoritmus i-edik összetevőjének költsége $c(e_i)$. Továbbá adott egy $P : \mathcal{K} \to \mathbb{R}$ potenciálfüggvény.

Definíció. Algoritmusunk i-edik össszetevőjének amortizált költsége:

$$c_{\text{amort}}(e_i) = c(e_i) + P(\kappa_i) - P(\kappa_{i-1}).$$

Ha a potenciált növeljük, akkor azért is fizetnünk kell az amortizált elszámolásban. Ha viszont potenciált vesztünk, akkor ez fizetési eszköt ad nekünk.

1. Tétel. *A fenti jelöléssel a teljes algoritmus költsége*

$$\left(\sum_{i=1}^{L} c_{amort}(e_i)\right) - \left(P(\kappa_L) - P(\kappa_0)\right).$$

A tétel remélhetőleg nyilvánvaló. Ha valaki egy formális bizonyítást szeretne látni, akkor összegeze az amortizált költségeket. Számolja külön az algoritmuselméleti és a potenciál változásából eredő hozzájárulást. A potenciálváltozás hozzájárulása egy teleszkópikus összeg lesz.

Az absztrakt tárgyalástól térjünk vissza bemelegítő példánkra. Tegyük fel, hogy n bites területünkön az utolsó c bitet írjuk felül, azaz az algoritmuselméleti költség c. Ezen blokkban egy 0-t követ $c - 1$ darab 1-es. A megelőző rész nem változik, így a potenciálváltozáshoz nem járul hozzá. Eredetileg az átírt blokk $c - 1$-gyel járul a potenciálhoz, az átírás után 1-gyel. Így

$$c_{\text{amort}}(e_i) = c + (1 - (c - 1)) = 2.$$

Korábbi eredményeink most a tételre való hivatkozással adódnak.

Csak a nyelvezet és így a gondolkodásmód más, de ugyanazt a matematikai ötletet „kódoljuk”.

2. példa: Adott n pont a koordináta-síkon: (x_1, y_1), (x_1, y_1), (x_1, y_1), ..., (x_n, y_n) úgy, hogy x-koordinatáik szigorúan monoton növekvőek. Határozzuk meg az input ponthalmaz konvex burkát (mely pontok, milyen körszerűen rendezett sorrendben szerepelnek a konvex borkon). A könvex buroknak lesz egy x koordinátára nézve minimális csúcsa és egy maximális csúcsa. Ez a két csúcs a konvex burok kerületét két ívre osztja. Ezeket (természetes módon) a ponthalmazunk felső és alsó konvex burkolójának nevezzük. A két rész ami együtt (a két extremális pont átfedésével) kiadja a teljes konvex burkot. A két burkoló szimmetrikus szerepet játszik. Elegendő azt tárgyalnunk, hogyan határozható meg a felső burkoló. Ezt részletezzük.

Legyen $\mathcal{P}_i$ az első i pont halmaza és

$$\mathcal{F}_i : E = P_1, P_{i_2}, P_{i_3}, \ldots, P_{i_{\ell-1}}, P_i$$

felső burkolójuk. Szúrjuk be a P_{i+1} pontot ebbe a felső burkolóba. Ezt a beszúrást ismételgetjük az algoritmus során.

A beszúrás nyilvánvaló: Az új felső burkoló az előző felső burkoló egy kezdőszelete lesz. Ez a kezdőszelet $U = P_{i_k}$-ban végződik, majd jön a P_{i+1} csúcs. Tehát az új felső burkoló:

$$\mathcal{F}_i : E = P_1, P_{i_2}, P_{i_3}, \ldots, P_{i_{k-1}}, U = P_{i_k}, P_{i+1}.$$

Az U pont úgy azonosítható, hogy a régi felső burkoló egyetlen U csúcs teljesíti a

$$m(P_{i_{k-1}}P_{i+1}) > m(UP_{i+1}) \leq m(P_{i_{k+1}}P_{i+1})$$

feltételt, ahol $m(AB)$ az A és B pontok egyenesének meredeksége.

A beszúrásra/$U = P_{i_k}$ megtalálására több lehetőségünk van.

I) Például tippelhetünk egy P_{i_j} csúcsra (az egyszerűség kedvéért tegyük fel, hogy $1 < j < \ell$). Három lehetőségünk van:

(a) $m(P_{i_{j-1}}P_{i+1}) > m(P_jP_{i+1}) \leq m(P_{i_{j+1}}P_{i+1})$. Ekkor eltaláltuk a korrekt U-t.

(b) $m(P_{i_{j-1}}P_{i+1}) > m(P_jP_{i+1}) > m(P_{i_{j+1}}P_{i+1})$. Ekkor a keresett U indexe (mint indexelt P) nagyobb mint j.

(c) $m(P_{i_{j-1}}P_{i+1}) \leq m(P_jP_{i+1}) < m(P_{i_{j+1}}P_{i+1})$. Ekkor a keresett U indexe (mint indexelt P) kisebb mint j.

Erre alapítva egy bináris keresést végezhetünk. $\mathcal{O}(\log \ell) = \mathcal{O}(\log i)$ kérdéssel megtaláljuk a keresett P_j-t. Minden „kérdés" költsége $\mathcal{O}(1)$ aritmetikai és összehasonlítási lépés. Tehát a beszúrás költsége $\mathcal{O}(\log i)$. A teljes költség $\mathcal{O}(\log 1 + \log 2 + \ldots + \log(n-1)) = \mathcal{O}(n \log n)$.

II) Egy sokkal naívabb próbálkozás, hogy teszteljük P_{i_ℓ} csúcsot. Ha ez nem jó U szerepére, akkor teszteljük a $P_{i_{\ell-1}}$ csúcsot. Ha ez nem jó U szerepére, akkor teszteljük a $P_{i_{\ell-2}}$ csúcsot. És így tovább. Talán meglepő, de ez az előzőnél jobb algoritmust ad. Mi ennek az oka? Ezt az amortizált analízis válaszolja meg.

Az $U = P_{i_k}$ csúcs megtalása után a $P_{i_{k+1}}$, $P_{i_{k+2}}$, ..., $P_{i_{ell}} = P_i$ csúcsok kiesnek a felső burkolóból. Sőt, soha nem is kerülhetnek vissza. Azaz a naív algoritmus minden sikertelen tesztje egy csúcs kiesésével jár. Legyen $c = \mathcal{O}(1)$ a költsége egy tesztnek. A P_{i+1} csúcs beszúrását $2c$ amortizációs költséggel számoljuk el. Ebből c egy „bekerülési költség". Ezt akkor fizetjük ki az aktuális tesztre, amikor megtaláljuk

az U csúcsot. A másik c költség a bekerült P_{i+1} csúcs „kikerülési költsége”, amit letétbe helyezzük a csúcsnál. Ezt akkor fizetjük ki (amennyiben P_{i+1} nincs a felső burkoló csúcsai között), amikor egy későbbi pont teszteli őt sikertelenül. Tehát minden sikertelen tesztet egy korábbi csúcs letéti összegéből fizetünk. Az amortizáció költség $2c = \mathcal{O}(1)$. A teljes költség $\mathcal{O}(n)$.

3. példa: Dinic-algoritmus. Adott egy $\overrightarrow{G}$ irányított gráf két kitüntetett csúccsal: s és t. A gráf rendelkezik a következő tulajdonsággal:

$$(D) \qquad \text{minden éle rajta van egy legrövidebb } \overrightarrow{st} \text{ úton.}$$

Az algoritmus során rendre egy e élt kapunk az aktuális gráfból, amit elhagyunk, majd a gráf további ritkítását végezzük el (ha szükséges), hogy a (D) tulajdonság továbbra is fennáljon. Az éleket addig kapjuk, amíg gráfunk ki nem ürül.

A (D) tulajdonság miatt gráfunk szintezett:

$$V(G) = S_0 \dot\cup S_1 \dot\cup \ldots \dot\cup S_{\ell-1} \dot\cup S_\ell,$$

ahol $S_0 = \{s\}$, $S_\ell = \{t\}$ és minden él valamely i-re S_i-ből S_{i+1}-be vezet.

Legyen $e = \overrightarrow{xy}$ az aktuális él ($x \in S_i$ és $y \in S_{i+1}$). Az e él elhagyása további ritkítást nem igényel, ha y-ba vezet e-től eltérő él is, és x-ből vezet ki e-től eltérő él is.

Ha x kifoka 1, akkor a korábbi szintek közt lesz olyan él ami már nem lesz rajta $\overrightarrow{st}$ úton. Például az összes x-be futó él ilyen, ezeket el kel hagyni. Ha azon csúcsok, amikből vezetett él x-hez 1 kifokúak voltak, akkor továbbgyűrűzik a hatás visszafelé.

Hasonló a hatás, ha y befoka 1. Ekkor az y-ból kifutó élek válnak feleslegessé (nem lesznek rajta $\overrightarrow{st}$ úton). Törlésük tovább gyűrűzhet előrefelé (a nagyobb indexű szintek felé) a hatás.

Algoritmusunk a fenti gondolatmenet naív végrehajtása: e törlése után követjük a törlés hatását. Előre és hátra is rendre teszteljuk, hogy a megfelelő fok 1 vagy nagyobb és 1 fok esetén tovább törölünk.

Az analízis remélhetőleg már nem okoz gondot. Egy e él esetén $\mathcal{O}(1)$ költsége annak, hogy töröljük gráfunkból és felismerjük gyűrűzik-e előre vagy hátra az él elhagyásának hatása. Ezt a költséget e-re hárítjuk. Ha az x-be befutó élek (vagy y-ból kifutó élek) is a törlendő élek listájára kerülnek, akkor az ezekkel kapcsolatos költségeket már az aktuális él viseli. Az egész algoritmus $\mathcal{O}(|E|)$ lépéssel megvalósítható.