VÉLETLENÍTETT ALGORITMUSOK

1.ea.

1. Bevezetés - (Mire jók a véletlenített algoritmusok, alap technikák)

1.1. Gyorsrendezés

Vegyünk egy ismert példát, a rendezések témaköréből, méghozzá a gyorsrendezés algoritmusát. Ebben egy adott S halmazban lévő n db számot rendezünk növekvő sorrendbe (a számokat sor adatszerkezettel tároljuk). A rekurzív algoritmus a következő:

1. ha a sorozat hossza <=1 akkor vége
2. választunk egy F elemet
3. helyezzük F elé az F-nél kisebb, mögé pedig az F-nél nagyobb elemeket
4. gyorsrendezés az F-nél kisebb elemekre
5. gyorsrendezés az F-nél nagyobb elemekre
6. vége

A gondot az F felosztó elem választása jelenti, ugyanis ha a választás nem megfelelő (F valamelyik oldala üres marad), akkor esetleg fölöslegesen hívódik meg a következő rekurzió, ami időveszteséget okoz. Ha ez minden lépésben megtörténik, akkor végeredményben (n-1), (n-2)... fölösleges hívás lesz, ami így azt eredményezi, hogy az algoritmus futási ideje $O(n^2)$. Ismerjük, hogy a legjobb, illetve az átlagos futási idő is $O(n*\log n)$. A következőkben megadunk egy olyan véletlenített algoritmust, aminek várható futási ideje $O(n*\log n)$ lesz. Ezt úgy érjük el, hogy a felosztó elemet egyenletes eloszlás alapján véletlenszerűen választjuk a halmazból (tehát mindegyiket azonos valószínűséggel). Ez az **RQS** (Randomized QuickShort) algoritmus. Mit mondhatunk ennek az algoritmusnak a várható futási idejéről? (a következőkhöz ld. még diszkrét valószínűségi változók, várható érték definíciója)

Az algoritmus egy futását ábrázolhatjuk egy olyan bináris keresőfával is, ahol a k. szinten lévő csúcsok a rekurzió k. mélységében választott felosztó elemek lesznek, és minden

csúcs bal oldali részfájába a tőle kisebb, jobb oldali részfájába a tőle nagyobb elemek kerülnek. A halmaz elemeit jelöljük $s_1, \dots, s_n$ -el, mégpedig úgy, hogy s_i jelentse a rendezés után i. helyre került számot (azaz s_1 a legkisebb, stb.). A fa levelei végül balról jobbra a rendezett sorozatot adják meg. A várható futási idő arányos lesz azzal, hogy a rendezés során hány összehasonlítás történik. Legyen az X_{ij} valószínűségi változó értéke 1, ha s_i és s_j össze lesz hasonlítva és 0, ha nem. Ekkor az összehasonlítások száma $\sum_{i,j} X_{ij}$ lesz. Ennek várható értéke $E(\sum X_{ij}) = \sum E(X_{i,j})$ lesz. A várható érték definíciója alapján $E(X_{i,j}) = P(X_{i,j}=1)*1 + P(X_{i,j}=0)*0 = P(X_{i,j}=1)$. Az RQS futása során a fa bármely csúcspontjában érvényes, hogy az algoritmus össze fogja hasonlítani az aktuális csúcsot (felosztó elemet) a két leszármazott részfa elemeivel, de nem lesz összehasonlítás a két részfa egyik eleme között sem, azaz egy i és egy j elem akkor és csak akkor lesz összehasonlítva, ha valamelyikük felosztó elem. Megfogalmazhatjuk még azt a szabályt is, hogy $i \le j$ elemek akkor és csak akkor lesznek összehasonlítva, ha valamelyikük hamarabb lesz felosztó elem, mint bármelyik közöttük lévő elem ($s_{i+1}, \dots, s_{j-1}$). Ez könnyen belátható úgy, hogy vegyük azt az s_k elemet, ami először lesz kiválasztva a [i;j] intervallumból $i \le k \le j$. Ekkor ha $k \notin \{i, j\}$ akkor a k csúcs bal oldali részfájába kerül i és jobb oldali részfájába j, így nem lesznek összehasonlítva. Ha pedig $k \in \{i, j\}$, akkor mivel az egyikük felosztó elem, össze lesznek hasonlítva. A fenti elgondolások alapján így kiszámolhatjuk annak a valószínűségét, hogy az i és a j össze lesz hasonlítva:

$$\left.\begin{array}{l} P(s_i \ \ lesz\ a\ felosztó\ elem) = \dfrac{1}{j-i+1} \\ \\ P(s_j \ \ lesz\ a\ felosztó\ elem) = \dfrac{1}{j-i+1} \end{array}\right\} \quad P(s_i\, vagy\, s_j \ \ felosztó\ elem\ lesz) = \frac{2}{j-i+1}$$

Így $\sum_{i=1}^{n} \sum_{j=i+1}^{n} p_{ij} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \le \sum_{i=1}^{n} \sum_{j=1}^{n-i+1} \frac{2}{j} \le 2*\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{j} = 2*n*\sum_{j=1}^{n} \frac{1}{j}$.

Mivel az utóbbi szumma egy harmonikus sor, melynek n tagú részletösszege $\approx \log n$, ezért azt kaptuk végeredményben, hogy az RQS algoritmus várható futási ideje

$O(n*\log n)$, azaz várhatóan jobb eredményt tudtunk elérni egy véletlenített algoritmussal, mint egy determinisztikussal. Az is látható, hogy ez az algoritmus biztosan helyes eredményt ad. Az ilyen típusú algoritmusokat **Las Vegas típusú algoritmus**nak nevezzük. Az olyan algoritmusokat, amelyek nem biztos, hogy helyes eredményt adnak (viszont általában kicsi a futási idejük) **Monte Carlo típusú algoritmus**oknak hívjuk. Eldöntési problémákra az utóbbiak kétfélék lehetnek.

a.) egyoldali hibás Monte Carlo algoritmus:

i.) a válasz „igen” ==> P(Alg igent mond)>=C (azaz ekkor hibázhat)

a válasz „nem” ==> P(Alg nemet mond)=1 (ilyenkor nem hibázik)

vagy fordítva

ii.) a válasz „igen” ==> P(Alg igent mond)=1

a válasz „nem” ==> P(Alg nemet mond)>=C

b.) kétoldali hibás Monte Carlo algoritmus:

mindkét esetben nem 0 valószínűséggel hibázik

1.2. Minimális vágás

Egy másik példa a minimális vágás feladat, ahol egy multigráfot kell egy vágással 2 részre osztani úgy, hogy a vágás mérete minimális legyen. (Multigráf: olyan gráf, ahol bármely 2 pont között lehet több él is, azaz az éleknek multiplicitása van). A vágás mérete:= a két csúcshalmaz között átmenő élek száma. Egy egyszerű véletlenített algoritmussal létre tudjuk hozni a kívánt vágást. Az algoritmus (Alg.): Amíg a csúcsok száma>=3 véletlenszerűen választunk 2 pontot (egyenletes eloszlással) és ezeket összeolvasztjuk. Amikor már csak 2 pont van, ez a két pont fogja reprezentálni a két halmazt. Megtalálja-e ez az algoritmus az optimális megoldást, és ha igen, milyen valószínűséggel?

Legyen OPT egy optimális megoldás (több is lehet) és legyen k ennek a mérete.

Lemma: P(Alg. OPT -t ad vissza) $\geq \frac{1}{n^2}$

Biz.:

Nézzük az első összevonást. Az OPT k mérete miatt bármely pont fokszáma >=k, mert ha nem így lenne, akkor létezne k-nál kisebb méretű megoldás (éppen az ilyen pontnál

vágva). Összesen tehát legalább $\frac{n*k}{2}$ él van, míg az OPT-ban k él. Annak valószínűsége, hogy a véletlenített algoritmusunk az első lépésben végez olyan összevonást, ami OPT-beli élet érint $k/\frac{n*k}{2}=\frac{2}{n}$. Jelöljük ε_i -vel azt az eseményt, hogy az i. lépésben nem vonunk össze OPT-beli élt. Ekkor az egyes lépésekben annak a valószínűsége, hogy abban a lépésben nem vonunk össze OPT-beli élt a következő :

$P(\varepsilon_1) \geq 1-\frac{2}{n}$

$P(\varepsilon_2|\varepsilon_1) \geq 1-\frac{2}{n-1}$ (mivel a 2. lépésre már csak n-1 csúcs marad, valamint feltételezzük, hogy az első lépésben sem választott ilyen élt)

⋮

$P(\varepsilon_i \mid \bigcap_{j=1}^{i-1} \varepsilon_j) \geq 1-\frac{2}{n+1-i}$ (itt már csak n+1-i csúcs van)

Az összes ε_i esemény bekövetkezésének valószínűsége (tehát hogy egészen addig mellőzzük az OPT-beli élek választását, míg már csak 2 csúcs marad) a fenti valószínűségek szorzata:

$P(\bigcap_{i=1}^{n-3} \varepsilon_i) = P(\varepsilon_1)*P(\varepsilon_2|\varepsilon_1)* \ldots * P(\varepsilon_i \mid \bigcap_{j=1}^{i-1} \varepsilon_j)$,

ami így $\geq(1-\frac{2}{n})(1-\frac{2}{n-1})(1-\frac{2}{n-2})\ldots(1-\frac{2}{3})=(\frac{n-2}{n})(\frac{n-3}{n-1})(\frac{n-4}{n-2})\ldots(\frac{1}{3})=\frac{2}{n(n-1)}>\frac{1}{n^2}$

amivel a lemmát bebizonyítottuk.

Így egy alsó korlátot kaptunk arra, hogy az algoritmus egy optimális megoldást ad vissza, ebből a helytelen eredmény visszaadásának valószínűségére kapunk egy felső korlátot:

P(Alg. Nem ad OPT -t vissza) $\leq 1-\frac{1}{n^2}$. Ez nem túl alacsony érték, de a rövid futási idő miatt az algoritmust függetlenül többször futtathatjuk, ezáltal a hiba valószínűségét csökkentve. Pl ha n^2 alkalommal végrehajtjuk, akkor

P(Alg.^{n2} nem ad OPT -t vissza) $\leq(1-\frac{1}{n^2})^{n^2}$, ami alulról tart 1/e -hez és így egy jobb korlátot ad.

1.3. Bonyolultságelméleti kitekintés

Def.: Az **RP osztály** (Randomized Polynomial Time) azon nyelvek osztálya, amikre létezik olyan véletlenített A algoritmus, ami a nyelv bármely szavára legrosszabb esetben is polinom időben fut, és

- $\forall x \in L \Rightarrow$ P(A elfogadja x-et)>=1/2
- $\forall x \notin L \Rightarrow$ P(A nem fogadja el x-et)=1

Def.: A **co-RP osztály** azon nyelvek osztálya, amikre létezik olyan véletlenített A algoritmus, ami a nyelv bármely szavára legrosszabb esetben is polinom időben fut, és

- $\forall x \in L \Rightarrow$ P(A elfogadja x-et)=1
- $\forall x \notin L \Rightarrow$ P(A nem fogadja el x-et)>=1/2

Mint látható, mindkét fenti osztálybeli algoritmusok egyoldali hibás Monte Carlo algoritmusok. Ha egy probléma mindkét osztály eleme egyszerre, akkor megoldható ”0 oldali” hibával is, azaz biztos helyes megoldás adható (Las Vegas algoritmussal).

Def.: A **ZPP osztály** (Zero-error Probabilistic Polynomial time) azon nyelvek osztálya, amikre létezik várhatóan polinomiális időben futó Las Vegas algoritmus.(ZPP=RP$\cap$co-RP)

Def.: A **PP osztály** azon nyelvek osztálya, amikre létezik olyan véletlenített A algoritmus, ami a nyelv bármely szavára legrosszabb esetben is polinom időben fut, és

- $\forall x \in L \Rightarrow$ P(A elfogadja x-et)>1/2
- $\forall x \notin L \Rightarrow$ P(A nem fogadja el x-et)>1/2

Az ilyen algoritmusok elég gyenge eredményt adnak, ezért hasznosabbak a következő osztály algoritmusai:

Def.: A **BPP osztály** azon nyelvek osztálya, amikre létezik olyan véletlenített A algoritmus, ami a nyelv bármely szavára legrosszabb esetben is polinom időben fut, és

- $\forall x \in L \Rightarrow$ P(A elfogadja x-et)>3/4
- $\forall x \notin L \Rightarrow$ P(A nem fogadja el x-et)>3/4

2. Játékelméleti technikák

2.1. Játékfa kiértékelés

Vegyünk egy olyan speciális játékfát, ahol a leveleket kivéve minden csúcs bináris. Ekkor a „max” csúcsokat elnevezhetjük OR -nak is, (bináris értékekre a „max” művelet ugyanazt az eredményt adja, mint az OR) a „min” csúcsokat pedig hasonló megfontolásból AND -nek. Egy determinisztikus algoritmusnak, ami kiértékel egy ilyen fát, mindig adható olyan bemenet, amire ki kell értékelni az összes levelet, míg véletlenített algoritmust használva itt is javíthatunk az eredményen. Jelölje a pontok fiainak számát d (most 2), és vegyünk egy olyan fát, ami pontosan 2k szintből áll. Ekkor a levelek száma $2^{2n}=4^n$, amit egy determinisztikus algoritmusnak legrosszabb esetben teljesen ki kell értékelnie. Algoritmusunkban ha a gyökér AND akkor válasszuk véletlenszerűen az egyik fiút, amit értékeljük ki rekurzívan

- ha ez 0 $\Rightarrow$ AND=0
- ha ez 1 $\Rightarrow$ menjünk a másik fiúra
    - ha az 0 $\Rightarrow$ AND=0
    - ha az 1 $\Rightarrow$ AND=1

Ha a gyökér OR akkor válasszuk véletlenszerűen az egyik fiút, amit értékeljük ki rekurzívan

- ha ez 1 $\Rightarrow$ OR=1
- ha ez 0 $\Rightarrow$ menjünk a másik fiúra
    - ha az 1 $\Rightarrow$ OR=1
    - ha az 0 $\Rightarrow$ OR=0

Állítás: A kiértékelt csúcsok száma (költség) várhatóan<=3^k.

Biz.(teljes indukcióval):

1. k=0 ra: az állítás triviális
2. Tfh. (n=k) -ra igaz, tehát <=3^k csúcs lesz kiértékelve

3. Bizonyítsuk be, hogy a 2. feltétel mellett az állítás n=k+1 -re is igaz.

a.)

Tfh. a részfában a gyökér egy OR csúcs, ami a kiértékelés során 1 értéket kap. Ez úgy állhat elő, hogy legalább az egyik fia 1 kiértékelést fog kapni. Ha szerencsénk van , ezt a fiút fogjuk először választani, és ekkor a másik ágat már nem is kell kiértékelni. Legrosszabb esetben viszont először egy 0 kiértékelésű fiát választjuk, és csak utána egy 1-est. Tehát ½ valószínűséggel a jó oldali fiút választjuk és ekkor csak ezt az egy részfát kell kiértékelni a feltevés szerinti legfeljebb 3^k költséggel, míg ½ valószínűséggel két részfát kell kiértékelni $2*3^k$ várható költséggel. Ekkor tehát ½ $*3^k$+ ½ $(2*3^k)=(3/2)*3^k$ lesz a várható költség.

Ha a gyökér a kiértékelés során 0 értéket kap, az csak úgy fordulhat elő, hogy megnéztük mindkét fiát (és mindkettő 0 értékű volt). Ebben az esetben tehát $2*3^k$ lesz a várható költség. (Ez a költségesebb az OR esetében)

b.)

Tfh. a gyökér egy AND csúcs, ami a kiértékelés során 1 értéket kap. Ez úgy állhat elő, hogy mindkét fia 1 kiértékelést fog kapni. Mivel ez 2 db 1 értékű OR csúcs kiértékelését jelenti, a várható költség ez esetben $2*[(3/2)*3]= 3^{k+1} <= 3^{k+1}$, tehát ebben az esetben teljesül az indukciós felvetés.

Ha az AND gyökér 0 értéket kap, az ½ valószínűséggel úgy jön létre, hogy megnéztük az egyik fiút, ami OR, 0 értékű (ekkor nincs szükség a másik fiú kiértékelésére), ½ valószínűséggel pedig úgy, hogy kiértékeltünk egy 1 értékű OR fiút, majd utána a másik fiút is (vegyük OR=0 -nak, feltételezve a legrosszabb esetet- az a. pontban látható, hogy az költségesebb). Ennek várható költsége így ½ * [$2*3^k$]+ ½ * [$(3/2)*3^k$ + $2*3^k$]= = 11/4 $*3^k <=3^{k+1}$, tehát ebben az esetben is teljesül az indukciós felvetés, azaz az állítás igaz.

Mivel $n=4^k$ levele van a fának, $\log_4(n)=k$. Ezt az előbb bebizonyított korlátba beírva azt kapjuk, hogy a Költség<= $3^{\log_4 n}$. Mivel $x^{\log_a b}=b^{\log_a x}$, így Költség<= $n^{\log_4 3} \approx n^{0.793}$.

Készítette: Oláh Péter