Математичка гимназија

МАТУРСКИ РАД

- из Рачунарства и информатике -

Рандомизирани алгоритми и структуре података

Ученик:
Марко Грујчић IVд

Ментор:
Јелена Хаџи-Пурић

Београд, јун 2020.

Садржај

1

Увод

Идеја да процес може бити случајан била је присутна од давнина. Коцка и продаја осигурања, који су присутни од античког доба, нам то и показују. Иако су се људи користили случајним догађајима хиљадама година, математички формализам у овој области почиње да се развија тек у седамнаестом веку. Рачунарска наука ступа на сцену знатно касније, па је идеја рандомизације и математичка основа била присутна од самог почетка.

Рандомизирани алгоритми и структуре података су такви да у својој логици садрже одређени степен случајности. Рандомизирани алгоритми деле се на две класе. Лас Вегас алгоритми увек дају тачан одговор, али случајност утиче на време извршења алгоритма. Пример алгоритма овог типа је алгоритам брзог сортирања. Монте Карло алгоритми дају тачан резултат са одређеном вероватноћом, у супротном могу дати погрешан резултат или дати обавештење да је извршење неуспешно. Пример Монте Карло алгоритама су рандомизирани тестови прималности.

Питање је зашто би се уопште користили рандомизирани алгоритми и структуре података. Употребом рандомизације само се проширује скуп алатки које користимо - детерминистички алгоритми који увек дају тачан резултат су подскуп скупа алгоритама који дају тачан резултат са одређеном вероватноћом; такође су подскуп скупа алгоритама који увек даје тачно решење а очекивана временска и просторна сложеност је оптимална. Рандомизација омогућава нам да, у очекивању, заобиђемо „најгоре случајеве" који се намећу неким детерминистичким алгоритмима - неки проблеми који се не решавају ефикасно детерминистичким алгоритмима могу се успешније решавати употребом рандомизације.

Такође су интересантни и неки рандомизирани алгоритми и структуре података који имају сличне перформансе као детерминистички, али доносе друге предности. Најчешћа предност ових алгоритама и структура података је што су једноставнији концептуално и за имплементацију.

2

Хеширање

Једна од најчешћих примена разних структура података је одржавање скупа који се временом мења. Пример би био следећи: великој компанији потребно је да одржава скуп тренутно запослених. У овом примеру постоји универзум U - скуп свих људи који је јако велик, а скуп запослених $S \subset U$ знатно је мањи. Структура података која би се применила на овај проблем мора брзо да обавља следеће операције: уметање, брисање и провера да ли је дати елемент скупа U такође и у скупу S. Структуре података која подржавју ове операције називају се *речници*.

Најједноставније решење оваквог проблема била би употреба повезане листе али ово решење је веома неефикасно из разлога што је време потребно да се изврши провера да ли се елемент налази у скупу пропорционално дужини листе. Низ би био временски знато ефикаснија структура, али због величине универзума није могуће једноставно направити низ величине $|U|$ који ће за сваки могући елемент памтити да ли се налази у S.

Основна идеја технике хеширања је да се уместо низа дужине упоредиве са $|U|$ користи низ дужине реда величине $|S|$.

Нека је n природан број такав да у сваком тренутку важи $|S| \leq n$. H је низ дужине n у ком ће се чувати информације о елементима скупа S, а $h : U \to \{0, 1, ..., n-1\}$ функција која слика елементе универзума у индексе у низу. Низ H назива се *хеш табела*, а функција h назива се *хеш функција*. Када се елемент u додаје у скуп $|S|$ једноставно се записује u на позицији $h(u)$ у хеш табели, док се претрага обавља провером да ли важи $H[h(u)] = u$.

Оваква структура функционисала би изузетно добро уколико би за свака два u, v из скупа S важило $u \neq v \implies h(u) \neq h(v)$, али то не мора да се деси у општем случају. Појава да се два различита елемента скупа S функцијом h сликају у исту позицију у низу назива се *колизија*. Најједноставнији начин превазилажења проблема које ствара колизија је да се у сваком пољу хеш табеле уместо једног елемента чува повезана листа. Сви $u \in S$ за које важи

$h(u) = i$ садржани су у повезаној листи $H[i]$.

Време које је потребно за извршавање сваке од три операције за задату вредност u пропорционално је времену које је потребно да се израчуна вредност $h(u)$ и дужини листе $H[h(i)]$, то јест броју елемената из S који су у колизији са u. Циљ хеш функције биће да што равномерније распореди елементе скупа S у хеш табелу, то јест да се минимизира број колизија за сваку од листи у хеш табели.

Бирање оптималне хеш функције Нека је скуп U неки подскуп скупа природних бројева (када је скуп U пребројив увек могуће пресликати га у неки подскуп скупа природних бројева). Једна од најједноставнијих идеја била би $h(u) = u \mod n$. Ова хеш функција даваће добре перформансе у доста ситуација али радећи над неповољним скупом S ова функција даће много колизија, на пример $n = 128$ а $U = \{2^k | k \in \mathbb{N}_0\}$. У циљу превазилажења ове потешкоће примењује се рандомизација при конструкцији функције h.

Узимајући у обзир рандомизацију најједноставнији је екстрем ове идеје где се при сваком уметању елемента u вредност $h(u)$ одређује униформно случајно из скупа $\{0, 1, ..., n-1\}$.

Лема 2.1. Користећи униформну случајну хеш функцију вероватноћа да су два елемента u и v у колизији једнака је $\frac{1}{n}$

Доказ. Од n^2 могућих вредности пара $(h(u), h(v))$ свака има исту вероватноћу да буде одабрана, пошто n од n^2 могућности значи колизију, вероватноћа да дође до колизије је $\frac{1}{n}$. □

Међутим оваква хеш функција није употребљива - проблем настаје при покушају да се изврши операција претраге или брисања пошто ће за исту вредност u хеш функција сваки пут дати случајан број. Самим тим утврђивање где би u требало да се налази у H није могуће у оптималном времену. Јасно је да сама функција h мора бити детерминистичка, па се рандомизација уводи при самом одабиру функције h.

Универзалне класе хеш функција Функција h бираће се случајно из пажљиво одабраног скупа функција χ које сликају U у $\{0, 1, ..., n-1\}$. Циљ биће да се што боље очува ефикасност униформно случајне хеш функције која проистиче из леме 2.1, па скуп χ мора имати следеће особине:

1. За свака два елемента $u, v \in U$ вероватноћа да за случајно одабрану функцију $h \in \chi$ важи $h(u) = h(v)$ мања је или једнака $\frac{1}{n}$.

2. За сваку функцију $h \in \chi$ ефикасно се рачуна $h(u), \forall u \in U$.

3. Представљање сваке функције је једноставно и компактно.

Због треће особине није могуће случајно бирати функцију из скупа свих функција које сликају U у $\{0, 1, ..., n-1\}$, већину тих функција није могуће представити на други начин сем експлицитним дефинисањем вредности $h(u)$ за сваки $u \in U$.

Лема 2.2. Нека је χ универзална класа хеш функција која слика U у $\{0, 1, ..., n-1\}$, нека је $S \subseteq U$ такав да $|S| \leq n$ и нека је $u \in U$. Нека је X случајна величина једнака броју елемената $s \in S$ за које је $h(s) = h(u)$. Важи $E(X) \leq 1$.

Доказ. Пошто је U универзална класа функција $P\left(h(s) = h(u)\right) \leq \frac{1}{n}$. Нека је X_s индикатор догађаја који узима вредност 1 уколико $h(s) = h(u)$, у супротном 0.

$$E(X) = \sum_{s \in S} E(X_s) = \sum_{s \in S} P\left(h(s) = h(u)\right) \leq n \cdot \frac{1}{n} = 1$$

□

Дизајнирање универзалне класе хеш функција Уместо n величина хеш табеле биће прост број $p \approx n$, и универзум се пресликава у векторе облика $x = (x_1, x_2, ...x_r), r \in \mathbb{N}, x_i \in N_0 \wedge x_i < p, \forall i \in [1, r]$. Један начин да се обави ово пресликавање је да се $u \in U$ представи у систему са основом p, то јест слика се у

$$x_u = \left(u \bmod p, \left\lfloor \frac{u}{p} \right\rfloor \bmod p, .., \left\lfloor \frac{u}{p^{r-1}} \right\rfloor \bmod p\right)$$

где је $x \mod p$ целобројни остатак при дељењу x са p. Уколико је $U = [0, N-1]$ користи се $r = \left\lceil \log_p(N) \right\rceil$.

Нека је A скуп свих вектора облика

$$a = (a_1, a_2, ..., a_r), a_i \in \mathbb{N}_0 \wedge a_i < r, \forall i \in \{1, 2, ..., r\}$$

За свако $a \in A$ дефинише се пресликавање

$$h_a(x) = \left(\sum_{i=1}^{r} a_i x_i\right) \bmod p$$

Овиме дефинисана је класа хеш функција $\chi_A = \{h_a | a \in A\}$ Свако од ових пресликавања испуњава захтеве да се ефикасно рачуна и једноставно представља, па преостаје прва особина да би ова класа функција била универзална.

Лема 2.3. Нека је p прост број и z цео број такав да $z \not\equiv_p 0$, и нека су $\alpha, \beta \in \mathbb{Z}$. Уколико важи $z\alpha \equiv_p z\beta$ следи $\alpha \equiv_p \beta$

Доказ. Из $z\alpha \equiv_p z\beta$ следи $z\,(\alpha - \beta) \equiv_p 0$. Пошто $z \not\equiv_p 0$ важи $(\alpha - \beta) \equiv_p 0$, одакле следи $\alpha \equiv_p \beta$. □

Теорема 2.1. Класа хешфункција χ_A је универзална.

Доказ. Нека су $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$ два различита елемента универзума U. Потребно је доказати да је вероватноћа да је $h_a(x) = h_a(y)$ за случајно одабрано $a \in A$ мања или једнака $\frac{1}{p}$.

Пошто $x \neq y$ постоји индекс j такав да $x_j \neq y_j$. Сада посматрајмо процес случајног одабира вектора a. Нека су прво случајно одабрани сви a_i где $i \neq j$. Након тога случајно се бира вредност a_j. Идеја доказа је да се покаже да без обзира на одабир вредности a_i вероватноћа да је $h_a(x) = h_a(y)$ након случајног одабира a_j једнака је $\frac{1}{n}$, одакле би јасно следило да је и случајним одабиром вредности вектора a вероватноћа да је дошло до колизије $\frac{1}{n}$.

Из дефиниције функције h_a следи да је $h_a(x) = h_a(y)$ ако и само ако

$$a_j(y_j - x_j) \equiv_p \sum_{i \neq j} a_i(x_i - y_i)$$

Пошто су вредности a_i фиксиране, вредност $\sum_{i \neq j} a_i(x_i - y_i)$ једнака је константи коју ћемо обележавати са m. Такође нека се вредност $y_j - x_j$ означава са z.

Сада је довољно доказати да постоји тачно једна вредност $0 \leq a_j < p$ таква да је $a_j z \equiv_p m$. Претпоставимо да постоје различити цели бројеви α и β из $[0, n)$ такви да $\alpha z \equiv_p \beta z$. Из леме 2.2 следи $\alpha \equiv_p \beta$,а пошто α и β припадају интервалу $[0, n)$ следи да су једнаки, што је контрадикција са претпоставком. Овиме доказано је да за тачно једну од n вредности које може узети a_j долази до колизије, па је вероватноћа за колизију једнака $\frac{1}{n}$ због униформно случајног одабира вредности a_j. Овиме је доказана теорема - χ_A јесте универзална класа хешфункција.

□

3

Скип листе

Скип листу први је описао Вилијам Пју, са идејом да представи структуру која има сличне перформансе као детерминистичке структуре попут балансираних бинарних стабала уз једноставнију идеју и имплементацију. Ова структура је речник - чува скуп и подржава операције уметања, брисања и претраге.

3.1 Услојавање скупа

Посматрајмо уређени скуп

$$S = \{x_1 < x_2 < ... < x_n\}$$

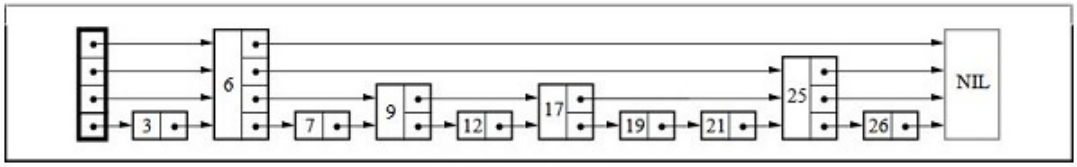
Дефиниција 3.1. *Услојавање* са r слојева уређеног скупа S је r скупова $L_1, L_2, ..., L_r$ за које важи

$$\varnothing = L_r \subseteq L_{r-1} \subseteq ... \subseteq L_1 = S$$

Дефиниција 3.2. За дати уређени скуп C и услојавање над тим скупом, *ниво* неког $x \in S$ се дефинише као

$$l(x) = max\{i | x \in L_i\}$$

За дато услојавање скупа S можемо дефинисати следећу структуру података: слој L_1 представљен је повезаном листом у којој су елементи сортирани. Сваком $x \in L_i$ где је $i > 1$ одговараће поље које показује на поље првог већег из L_i и на поље које одговара вредности x за скуп L_{i-1}. Из практичних разлога додаћемо на почетак и крај елементе $-\infty$ и ∞ тако да је $l(-\infty) = l(\infty) = r$.



Илустрација описине структуре

Рандомизирано услојавање добијамо на следећи начин: слој $l(x)$ је случајна величина са помереном геометријском расподелом параметра $p = \frac{1}{2}$ за свако $x \in S$. x ће бити елемент сваког од скупова $L_1, L_2, ..., L_{l(x)}$. Број слојева r једнак је $\max_{i=1}^{n} l(x_i)$.

Лема 3.1. Очекивани захтевани простор скип листе над рандомизираним услојавањем скупа S са n елемената је $O(n)$.

Доказ. Захтевани простор биће збир n независних померених геометријских расподела. Очекивана вредност тог збира је:

$$E\left(N \cdot G_1\left(\frac{1}{2}\right)\right) = N \cdot E\left(G_1\left(\frac{1}{2}\right)\right) = N \cdot \left(1 + \frac{1}{\frac{1}{2}}\right) = 3 \cdot N = O(n)$$

□

Лема 3.2. Број слојева r у скип листи је $O(\log n)$ са великом вероватноћом - вероватноћом бар $1 - \frac{1}{n^c}$ за константу $c > 1$.

Доказ. За сваки $x \in S$ након услојавања важи $P\left(l(x) > L + 1\right) = \frac{1}{2^L}$. Пошто је $r = \max_{i=1}^{n} l(x_i)$ следи

$$P(r > L + 1) = P\left(l(x_1) > L + 1 \vee l(x_2) > L + 1 \vee ... \vee l(x_n) > L + 1\right)$$

Користећи да важи $P(A \vee B) \leq P(A) + P(B)$ за било која два случајна догађаја добија се

$$P(r \leq L+1) \leq P(l(x_1) > L+1) + P(l(x_2) > L+1) + ... + P(l(x_n) > L+1) = \frac{n}{2^L}$$

За $L \leq \log_2 n$ ово ограничење је тривијално пошто је онда $\frac{n}{2^L} \geq 1$. Међутим, за константу $c > 1$ важиће

$$P(r \geq 1 + c \log_2 n) = \frac{1}{n^{c-1}}$$

одакле следи тврђење леме.

□

Тврђење претходне леме интуитивно је веома јасно: у првом слоју биће n поља која одговарају елементима S, очекује се да ће пола од тих поља имати одговарајуће поље у другом слоју, затим се очекује да ће пола поља из другог слоја имати одговарајуће у трећем. Понављањем поступка интуитивно се види да листа има отприлике $\log_2 n$ слојева.

3.2 Претрага, уметање, и уклањање елемената

Претрага: За задату вредност x потребно је проверити да ли се налази у скупу S. Претрага почиње од поља које одговара вредности $-\infty$ у слоју L_r. Уколико посматрано поље које одговара вредности x у слоју L_1 утврђено је да се x налази у S и претрага се обуставља. У супротном нека поље које се посматра одговара вредности y у слоју L_i. Обављају се следећи кораци:

1) Нека је z најмањи елемент скупа S строго већи од y за који постоји одговарајуће поље у L_i. Уколико је $z \leq x$ ново посматрано поље постаје оно које одговара вредности z у слоју L_i, поступак се понавља.

2) Уколико је z веће од x и $i > 1$ ново посматрано поље одговара вредности x у слоју L_{i-1},поступак се понавља. Уколико је $i = 1$ утврђено је да се x не налази у скупу S.

Пошто табела садржи коначно много поља и датим корацима није могуће да неко поље буде посматрано два пута, претрага се завршава после коначно много корака.

Лема 3.3. Очекивани број корака у једној претраги је $O(\log n)$

Доказ. Посматрајмо поља која смо обишли у претраги уназад. Прво поље у обиласку уназад одговара највећем елементу скупа S мањем или једнаком x у слоју L_1. До поља које одговара $-\infty$ у слоју L_r се стиже на следећи начин:

1) Уколико тренутно посматрано поље одговара $-\infty$ у слоју L_r обилазак је готов.

2) У супротном нека тренутно посматрано поље одговара вредности y у слоју L_i. Уколико је $l(y) > i$ следеће посматрано поље одговара вредности y у слоју L_{i+1}. Ако је $l(y) = i$ следеће посматрано поље одговара вредности z,где је z највећа вредност у L_i мања од y, у слоју L_i.

Из истог разлога као и сама претрага овај обилазак има коначно много корака.

Очекивани број помераја у поље које одговара вредности у истом слоју је $O(1)$ пошто је за сваку вредност за коју важи $l(x) \geq i$ знамо $P(l(x) > i) = \frac{1}{2}$ самим тим број обиђених поља при обиласку у једном слоју биће случајна величина са геометријском расподелом чије је очекивање једнако $\frac{1}{\frac{1}{2}} = 2$. Пошто је r $O(\log n)$ очекивани број корака при претраживању је $O(\log n)$. □

Уметање У скуп C, самим тим и скип листу, додаје се вредност a. Први део уметања обавља се као претрага, са тиме да се за сваки слој памти поље које одговара највећој вредности у скупу $S \cup \{-\infty\}$ $(-\infty < x | \forall x \in S)$ међу обиђеним пољима у том слоју, за које је ознака M_i у слоју i. Уколико је вредност x пронађена у скип листи инсерција се обуставља. У супротном други део полази од слоја L_1. Нека поље M_1 показује на поље X у слоју L_1. Ново поље које одговара вредности a у слоју L_1 се додаје, ново поље показује на поље X, а показивачу поља M_1 ће бити додељена вредност која одговара новом пољу.

Уклањање Процес уклањања вредности a из скип листе обавља се као претрага са следећом изменом: уколико тренутно посматрано поље које садржи вредност y у слоју L_i има показивач на поље у истом слоју које одговара елементу a, вредност тог показивача мења се тако да одговара пољу на које на које показује поље које одговара вредности a у L_i у истом слоју. Након тога ослобађа се простор који заузима поље које одговара a у L_i и уклањање се наставља у пољу које одговара y у L_{i-1} или се обуставља уколико је $i = 1$.

Временска сложеност уметања и уклањања је $O(\log n)$ као последица чињенице да је временска сложеност претраге $O(\log n)$.

4

Трип

Трип је први описао Жан Вилмен 1980. године и структуру је назвао Декартово стабло. Реч „трип" први је искористио Едвард Мекреј да опише сличну структуру, за коју је касније користио назив *приоритетно стабло претраге*. Године 1989. ову структуру изнова су описали са рандомизацијом Сесилија Арагон и Рајмунд Зајдел.

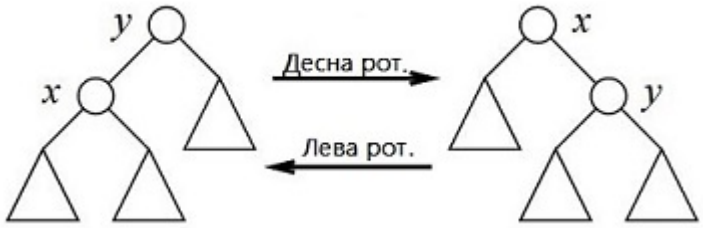
Трип је бинарно стабло у ком сваки чвор има кључ и приоритет. *Инордер* обилазак стабла даје сортиран низ по кључевима (попут бинарног стабла претраге), и за сваки чвор важи да је његов приоритет мањи од приоритета његове деце(попут хипа).

Под претпоставком да су вредности кључева и приоритета јединствене, лако се индуктивно доказује да је структура стабла у потпуности одређена. Пошто чворови трипа имају информационо поље приоритет, као и код хипа корен стабла мора бити чвор са најмањим приоритетом. За сваки преостали чвор знамо ком подстаблу припада по вредности његовог кључа. Оба подстабла такође имају особине трипа, па је њихов корен јединствено одређен. На сличан начин доказује се и да је структура трипа иста као структура бинарног претраге са истим вредностима кључева које се гради уметањем вредности растуће по вредности приоритета која би им одговарала у трипу.

4.1 Операције над трипом

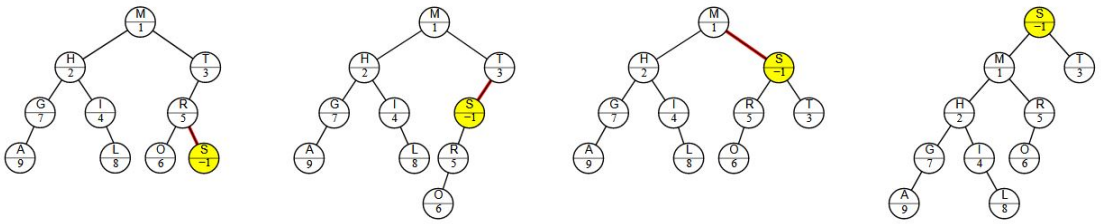
Претрага се обавља као у обичном бинарну стаблу претраге - полази од корена стабла, уколико је вредност траженог кључа једнака вредности кључа корена претрага се завршава. У супротном знамо у ком подстаблу се налази тражени кључ уколико постоји у стаблу - претрага се наставља рекурзивно на одговарајућем детету корена.

Уметање новог чвора z састоји се из два дела. Први део исто се обавља као уметање у обично бинарно стабло претраге. После првог корака чвор са кључем z је лист и стабло је и даље бинарно стабло претраге по кључевима али услов да је хип по приоритетима не мора бити одржан. Други део састоји се од следећег: све док је вредност приоритета оца чвора z већа од вредности приоритета чвора z обавља се ротација (слика).



Десна и лева ротација

Ротацијама одржава се поредак по кључевима, и не могу се појавити нови чворови за које није испоштован хип поредак између тог чвора и оца. Ротација обавља се у константном времену пошто се своди на манипулисање показивачима. На следећој слици приказане су ротације над трипом чији су кључеви слова алфабета а приоритети цели бројеви након првог дела инсерције чвора $(S, -1)$.



Пример ротација

Уклањање чвора са вредности кључа z обавља се на следећи начин: све док z није лист обавља се ротација над z и дететом чвора z са мањим приоритетом. На овај начин одржавају се особине и хипа и бинарног стабла претраге за све чворове(ако изузмемо чвор z и његове односе са децом и оцем, пошто ће бити обрисан). Када је z лист може се уклонити из стабла. Слика *Примери ротација* посматрана са десна на лево одговара уклањању чвора $(S, -1)$ - уклањање фунцкионише као уметање у супротном смеру.

Подела трипа T на $T_<$ и $T_>$ по вредности v такве да $T_<$ садржи све чворове са вредностима кључа мањим од v, а $T_>$ садржи све чворове са кључевима већеим од v обавља се уметањем чвора $(v, -\infty)$. Пошто је вредност приоритета $-\infty$ овај чвор ће бити корен стабла након уметања. Из особина бинарног стабла претраге следи да су сви чворови са вредности кључа мањом од v у левом подстаблу, а сви чворови са вредности кључа већом од v у десном подстаблу, па лево и десно подстабло управо одговарају траженим $T_<$ и $T_>$.

Спајање два трипа $T_<$ и $T_>$ за које важи да је вредност кључа сваког чвора у $T_<$ мања од датог v, а вредност кључа сваког чвора $T_>$ већа од v обавља се супротно од поделе: након конструкције трипа чији је корен $(v, -\infty)$, лево подстабло корена $T_<$, а десно $T_>$. Уклањањем корена овако добијеног трипа добијамо тражени спојени трип.

Јасно је да је временска сложеност сваке од операција линеарно пропорционална дубини неког чвора у стаблу - у најгорем случају висини стабла.

4.2 Рандомизација вредности приоритета

Рандомизирани трип је трип за који су вредности приоритета независне равномерно распоређене случајне вредности. Очекивана вредност дубине за сваки чвор је $O(\log n)$, где је n број чворова. За портребе доказа ове тврдње вредност приоритета узимаће вредност реалног броја из интервала $(0, 1)$, чиме је вероватноћа да две вредности кључа имају исти приоритет 0. У пракси могу се користити целобројне вредности, на пример из интервала $[0, 2^{31}-1]$.

Нека је x_k чвор са k-том најмањом вредности кључа. Нека је A_k^i индикатор догађаја који узима вредност 1 уколико је чвор x_i предак чвора x_k, у супротном 0. Јасно је да важи: $dubina(x_k) = \sum_{i=1}^{n} A_k^i$. Јасно следи:

$$E(dubina(x_k)) = \sum_{i=1}^{n} P(A_k^i = 1)$$

Нека је $X(i, k)$ ознака за скуп чворова $\{x_i, x_{i+1}, ..., x_k\}$ уколико је $i < k$, у супротном $\{x_k, x_{k+1}, ..., x_i\}$.

Лема 4.1. За свако $i \neq k$, x_i је предак x_k ако и само ако x_i има најмањи приоритет међу чворовима који припадају $X(i, k)$

Доказ. Уколико је x_i корен стабла, засигурно је и предак чвора x_k. Из хип особине трипа следи да чвор x_i има најмању вредност приоритета у целом

стаблу, па приоритет чвора x_i јесте најмањи у $X(i,k)$.

Уколико је x_k корен стабла, са истим образложењем као претходни случај важи да је x_k предак x_i и да је приоритет чвора x_k мању од приоритета чвора x_i, што је у складу са лемом. Преостаје случај када ни x_i ни x_k није корен стабла. Нека је x_p најнижи заједнички предак чворова x_i и x_k. Уколико $x_i \neq x_p \wedge x_k \neq x_p$ јасно је да је приоритет чвора x_p мањи и од приоритета чвора x_k и од приоритета чвора x_i, и да x_i и x_k припадају у различитим подстаблима чвора x_l.Из претходног и особина бинарног стабла претраге следи $i < l < k \vee k < l < i$ самим тим $x_l \in X(i,k)$. Из хип особине трипа следи да је приоритет чвора x_l мањи од приоритета чвора x_i.

Уколико важи $l = i \vee l = k$ (пошто је подстабло трипа такође трип) доказ се своди на један од прва да случаја.

□

Пошто је расподела вредности приоритета униформна сваки чвор који припада $X(i,k)$ има вероватноћу $\frac{1}{|i-k+1|}$ да има најмању вредност приоритета. Следи:

$$E(dubina(x_k)) = \sum_{i=1}^{n} \frac{1}{|i-k+1|} < 2\sum_{i=1}^{n} \frac{1}{i}$$

Из својстава хармонијског реда следи:

$$E(dubina(x_k)) = 2 \cdot O(\log n) = O(\log n)$$

Самим тим очекивана сложеност свих наведених операција је $O(\log n)$.

5

Брзо сортирање

Алгоритам брзог сортирања развија британски истраживач Тони Хор 1959. године током гостовања на Московском државном универзитету Ломоносов, а први рад на ову тему објављује 1961. Алгоритам брзог сортирања припада класи алгоритама подели па владај, то јест свођење проблема на више инстанци истог проблема са мањим димензијама.

5.1 Опис алгоритма

Дати низ потребно је сортирати растуће. Централна идеја алгоритма брзог сортирања је да уколико постоји елемент низа за који важи да сви елементи низа са индексом мањим од посматраног имају мању или једнаку вредност, а сви елементи низа са индексом већим од посматраног имају већу или једнаку вредност, уколико би сви елементи пре посматраног били сортирани, и сви елементи после били сортирани и цео низ би био сортиран. Овиме се проблем сортирања целог низа своди на два проблема исте природе мањих димензија.

Ова идеја примењује се на следећи начин: бира се елемент низа који има улогу пивот елемента. Алгоритмом партиције низ се трансформише тако да се сви елементи мањи од пивот елемента налазе пре пивот елемента у низу, а сви елементи већи од пивот елемента се налазе после пивот елемента у низу. Алгоритам партиције извршава се на следећи начин:

Иницијализација Први елемент низа и пивот елемент мењају место. Дефинишу се два бројача: $i = 1$ који ће означавати позицију пивот елемнта у низу и $j = 2$ који ће означавати тренутно посматрани елемент низа. Индекс првог елемента у низу је 1.

Обрада Уколико је вредност у низу на месту j већа од вредности низа на месту i вредност j повећава се за један и провера се понавља. Уколико је вредност у низу на месту j мања или једнака вредности пивот елемента мењају се вредности елемената низа на позицијама $i+1$ и j, затим елемената на позицијама i и $i+1$. Након тога повећавају се вредности показивача i и j за 1. Јасно је да ће у сваком тренутку важити $i < j$. Обрада се обуставља када $j > n$.

Применом алгоритма партиције рекурзивно на два добијена низа које разграничава пивот елемент низ се сортира. Базни случајеви рекурзије су низ са једним елементом и празан низ.

Прва непријатност на коју се наилази је да се лошим избором пивот елемента низ дели на два са драстично различитим величинама. Пример би био да се за пивот елемент бира први елемент, а задати низ сортиран је у опадајућем поретку. Применом алгоритма партиције добија се један део низа са $n-1$ елемената, и други који не садржи ни један елемент. Рекурзивни позиви над већим делом дају сличну ситуацију, самим тим сложеност алгоритма у овом случају била би пропорционална квадрату броја елемената низа. Ова препрека превазилази се рандомизацијом избора пивот елемента.

Други проблем су поновљене вредности у низу. Ова непријатност решава се изменом обраде у алгоритму партиције тако да се елементи једнаки пивот елементу групишу у један блок и не обрађују при даљим позивима функције која сортира. Имплементација ове исправке обавља се заменом показивача i са два показивача, i_p и i_k који означавају почетак и крај блока елемената једнаких пивот елементу. Иницијална вредност ових показивача је 1. Када је вредност на месту j мања или једнака вредности пивот елемента прво се мењају вредности елемената низа на позицијама i_k+1 и j, затим на позицијама i_p и i_k+1. Вредности показивача j и i_k се повећавају за 1, а вредност i_p се инкрементира уколико вредност на пољу j пре замена вредности није била једнака вредности пивот елемента.

5.2 Временска сложеност

За потребе доказа претпоставка је да су сви елементи низа различити. У секцији 4 показано је да изградња бинарног стабла претраге полазећи од празног стабла са рандомизираним избором следеће вредности која се умеће у стабло има очекивану временску сложеност $O(n \log n)$.

Посматрањем овог процеса и алгоритма брзог сортирања уочава се да пореде исте парове елемената само у различитом поретку: рандомизирано се бира први елемент. У алгоритму брзог сортирања има улогу пивот елемента, и сви преостали елементи низа се пореде са пивот елементом. У бинарном стаблу претраге ова вредност налази се у корену стабла, самим тим ће свако следеће уметање поредити вредност која се додаје у стабло и корен стабла. Такође, ова два приступа стварају исте подпроблеме - алгоритам брзог сортирања експлицитно дели низ на вредности мање од пивот елемента и вредности веће од пивот елемента, док се у бинарном стаблу претраге обрада при инсерцији за елементе мање од корена наставља се у левом подстаблу, а за веће у десном. Узимајући претходно у обзир индуктивно се доказује да ова два алгоритма имају исти број поређења, па директно следи да је очекивана временска сложеност рандомизираног алгоритма брзог сортирања $O(n \log n)$.

6

Глобални минимални рез

Проблем је следећи: задат је неусмерен мултиграф (два чвора може повезивати више грана) $G(V, E)$. Рез се дефинише као одабир два непразна скупа чворова A и B за које важи $A \cup B = V \wedge A \cap B = \varnothing$. Величина реза (A, B) дефинише се као број грана из E за које важи да им је један одговарајући чвор у A а други у B. Глобални минимални рез је рез најмање величине.

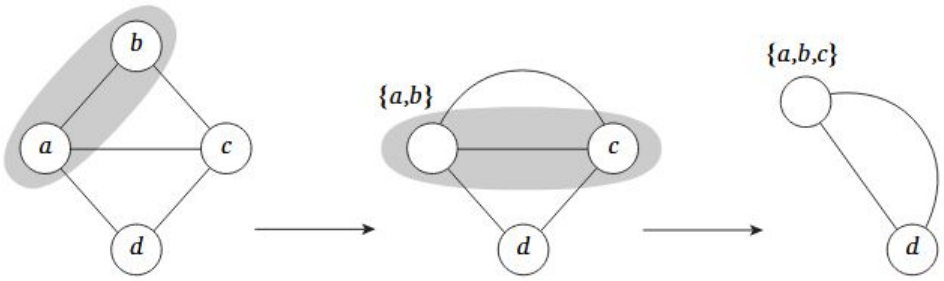
Следећи алгоритам назива се алгоритам контракција. Извршава се на следеци начин:

1) Из скупа E случајно и униформно се бира једна грана (u, v).

2) Над чворовима u и v обавља се *контракција*: полазећи од графа G формира се нови граф G' у ком чворовима u и v одговара чвор w. Остали чворови су очувани у графу G'. Гране које имају један крај у u а други у v бришу се из новог графа. Остале гране су очуване, са тиме да уколико је један чвор који одговара грани једнак u или v ажурира се да буде једнак новом чвору w.

3) Процес се понавља над графом G' све док је број чворова у G' већи од 2.

Новодобијене чворове у графовима добијеним контракцијама погодно је посматрати као *суперчворове* - сваком суперчвору w одговара подскуп полазног скупа човова $S(w) \subset V$ који садржи све чворове који су „прогутани" у контракцијама којима је добијен чвор w. Након извршеног алгоритма преостала су два чвора w_1 и w_2, и добијени рез је $(S(w_1), S(w_2))$.



Пример извршења алгоритма контракција

Тврђење 6.1. Алгоритам контракција даје глобални минимални рез графа G са вероватноћом од бар $1/\binom{n}{2}$, где је n број чворога графа G.

Доказ. Нека је (A, B) глобални минимални рез графа G величине k, а F скуп грана за које је један одговарајући чвор у скупу A, а други у скупу B. Поставља се питање које је доња граница вероватноће да алгоритам контракција дође до реза (A, B).

Вероватноћа да после прве контракције буде сигурно да алгоритам неће дати рез (A, B) је једнака вероватноћи да буде изабрана грана из F, пошто би тада један чвор који припада $S(A)$ и један чвор који припада $S(B)$ били спојени у један суперчвор одакле јасно следи да алгоритам неће доћи до реза (A, B).

Узимајући у обзир да је величина глобалног минималног реза једнака k намеће се да сваки чвор $v \in V$ има степен бар k - у супротном рез $(v, V - v)$ имао би величину мању од k што је контрадикција са претпоставком да је минимални глобални рез графа G величине k. Директно следи да је број грана већи или једнак $\frac{nk}{2}$, па је вероватноћа да је изабрана грана из скупа F мања или једнака од

$$\frac{k}{\frac{1}{2}kn} = \frac{2}{n}$$

Након j контракција тренутно посматрани граф G_j сачињен је од $n - j$ суперчворова. Претпоставка је да ни једна грана из F није одабрана у досадашњем извршавању алгоритма. Пошто за сваки рез графа G_j постоји одговарајући рез графа G степен сваког чвора из G је већи или једнак k. Следи да је број грана графа G_j већи или једнак $\frac{k(n-j)}{2}$ и вероватноћа да у следећој контракцији буде одабрана грана из F је мања или једнака од

$$\frac{k}{\frac{1}{2}k(n-j)} = \frac{2}{n-j}$$

Алгоритам ће дати рез (A, B) уколико ни једна грана из F није одабрана при извршењу алгоритма. Нека је I_j индикатор догађаја који узима вредност 1 уколико у j-тој контракцији није одабрана грана из F. Важи:

$$\begin{aligned}
&P(I_1 = 1) \cdot P(I_2 = 1 | I_1 = 1) \cdot ... \cdot P(I_{n-2} = 1 | I_1 = 1 \wedge ... \wedge I_{n-3} = 1) \\
&\geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n-1}\right) \cdot ... \cdot \left(1 - \frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right) \cdot \left(\frac{n-3}{n-1}\right) \cdot \left(\frac{n-4}{n-2}\right) \cdot ... \cdot \left(\frac{2}{4}\right) \cdot \left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&= \binom{n}{2}^{-1}
\end{aligned} \tag{6.1}$$

Добија се да је вероватноћа да је добијени рез управо (A, B) већа или једнака од $\binom{n}{2}^{-1}$, одакле тврђење следи директно.

□

Јасно је да је вероватноћа да је после једног извршавања алгоритма контракција добијен оптималан рез мала, зато се алгоритам покреће више пута. После $\binom{n}{2}$ извршсавања алгоритма вероватноћа да алгоритам није пронашао минимални рез мања је или једнака $\frac{1}{e}$:

$$\left(1 - 1/\binom{n}{2}\right)^{\binom{n}{2}} \leq \frac{1}{e}$$

Након $\binom{n}{2} \ln n$ извршавања, вероватноћа да минимални рез није успешно пронађен мања је или једнака $e^{-\ln n} = 1/n$. Алгоритам контракција могуће је имплементирати са временском сложености $O(m)$ па је сложеност са $\binom{n}{2} \ln n$ извршавања једнака $O(mn^2 \log n)$.

7

Налажење најближег пара тачака

Рандомизирано решење проблема проналажења најближег пара тачака први је представио израелски истраживач Мајкл Рабин 1976. године. Постава проблема је следећа: задат је скуп тачака $P = \{p_1, p_2, ..., p_n\}$ где тачка p_i има координате (x_i, y_i) и са $d(p_i, p_j)$ означава се Еуклидска раздаљина тачака p_i и p_j. Потребно је пронаћи две различите тачке p_i и p_j за које је $d(p_i, p_j)$ минимално. Једноставности ради претпоставка је да важи $0 \leq x_i, y_i < 1$ за свако $i = 1, 2, ..., n$ (сваки коначан скуп координата у линеарном времену може се пресликати у овакве координате).

Идеја алгоритма је једноставна: тачке посматрају се у рандомизираном поретку и чува се вредност δ једнака растојању најближег пара међу тренутно обрађеним тачкама. При обради $p \in P$ посматраћемо „околину" тачке ради провере да ли постоји тачка чија је удаљеност од p мања од δ. Уколико таква тачка не постоји вредност δ остаје иста и прелази се на следећу тачку. У супротном чува се нови пронађени пар најближих тачака и мења се вредност δ.

Нека су тачке у случајном поретку означене са $p_1, p_2, ..., p_n$. Алгоритам ће се одвијати у *етапама*; током сваке етапе најближи пар тачака остаће исти. На почетку прве етапе важиће $\delta = d(p_1, p_2)$. Циљ једне етапе је да утврди да δ јесте растојање између две најближе тачке или да пронађе пар тачака p_i, p_j за које важи $d(p_i, p_j) < \delta$. У оквиру једне етапе тачке посматрају се у утврђеном случајном поретку. Етапа се прекида уколико за тренутно посматрану тачку p_i постоји p_j где је $j < i$ и важи $d(p_i, p_j) < \delta$. Тада се започиње нова етапа у којој ће δ имати вредност растојања најближег пара од обрађених тачака: $\delta = \min_{j=1}^{i-1} d(p_i, p_j)$. Обрада тачака у новој етапи наставља се од тачке p_{i+1}.

Број етапа зависиће од поретка којим се посматрају тачке: уколико су p_1 и p_2 тачке са најкраћим растојањем алгоритам састојаће се од само једне етапе, док је у неповољном случају, на пример $p_i = (2^{-i}, 0)$ број етапа једнак $n - 2$.

Највећа препрека за имплементацију је ефикасна претрага околине тачке која се обрађује, то јест провера да ли је δ и даље најкраће растојање између неке две тачке узимајући у обзир нову тачку која се посматра. Идеја ове провере је да се јединични квадрат подели на подквадрате странице $\delta/2$. Формалније, постојаће N^2 подквадрата, где $N = \lceil 1/2\delta \rceil$: за $0 \leq s < N$ и $0 \leq t < N$ подквадрат S_{st} дефинише се као

$$S_{st} = \{(x, y) | s\delta/2 \leq x < (s+1)\delta/2 \wedge t\delta/2 \leq y < (t+1)\delta/2\}$$

Тврђење 7.1. Уколико тачке p и q припадају истом подквадрату важи $d(p, q) < \delta$.

Доказ. Уколико су p и q у истом подквадрату обе координате ове две тачке разликују се за највише $\frac{\delta}{2}$, следи

$$d(p, q) \leq \sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2} = \frac{\delta}{\sqrt{2}} < \delta$$

□

Дефиниција 7.1. Подквадрати S_{st} и S_{uv} су *близу* уколико важи $|s - u| \leq 2 \wedge |t - v| \leq 2$.

Тврђење 7.2. Уколико за $p, q \in P$ важи $d(p, q) < \delta$ подквадрати који садрже ове две тачке су близу.

Доказ. Претпоставимо супротно, нека $p \in S_{st}$ и $q \in S_{uv}$ и подквадрати S_{st} и S_{uv} нису близу. Следи $|s-u| > 2 \vee |t-v| > 2$ па им је по бар једној оси растојање веће од $2 \cdot \frac{\delta}{2} = \delta$, следи $d(p, q) < \delta$ није могуће што је контрадикција. □

Претходна два тврђења уобличиће идеју провере. Претпоставимо да је алгоритам у неком тренутку обрадио тачке $P' = \{p_1, p_2, ...p_{i-1}\}$ међу којима је најкраће растојање пара δ и посматрана тачка је p_i. За сваку тачку из P' сачувано је ком подквадрату припада.

Када се разматра тачка p_i прво се утврђује подквадрат S_{st} ком припада. Уколико постоји тачка у P' која је на растојању мањем од δ од p_i сигурно се налази у неком од квадрата који су близу S_{sp}. Пошто квадрата који су близу квадрату S_{sp} има до 25, проверавају се све тачке које су садржане у подквадратима који су близу S_{st}. Као последица тврђења 7.1 један подквадрат садржи највише једну тачку из P' па се при обради p_i раздаљина између две тачке рачуна до 25 пута.

Преостаје питање како проверити да ли неки подквадрат садржи тачку из P и ако садржи да утврди која је то тачка. За решавање овог проблема

природно се намећу структуре са улогом речника. Универзум U је скуп свих подквадрата, а скуп S који се одржава биће скуп подквадрата који садрже неку тачку из P'. Структура ће за сваку тачку $p' \in P'$ чувати подквадрат ком припада уз назнаку да је p' тачка садржана у том квадрату. Скуп свих подквадрата лако се пресликава у скуп $\mathbb{N}_0$, на пример S_{st} слика се у $s * N + t$ па се једноставно примењује било која од три обрађене структуре података.

При обради сваке тачке извршава се 25 операција претраге. За све пронађене тачке рачуна се растојање до тачке која се посматра. Уколико није пронађен пар са растојањем мањим од δ тренутно посматрана тачка се умеће у структуру и алгоритам се наставља.

Проблем наизглед настаје кад постоји $p' \in P'$ таква да важи $\delta' = d(p', p_i) < \delta$. Тада речник у тренутном облику постаје бескористан. Потребно је за сваку тачку из P' прво обавити уклањање из речника, затим израчунати ком подквадрату припада после промене вредности δ и обавити инсерцију у речник. Пошто се подквадрати сликају у скуп $\mathbb{N}_0$ може се користити иста структура и после промене етапе. Број операција над речником при једној промени етапе пропорционална је $|P'|$, па би укупан број операција над речником наизглед могао да буде пропорционалан квадрату броја тачака, али рандомизација поретка којим се обрађују тачке обезбедиће следећу особину:

Тврђење 7.3. Очекивани број операција над речником је $O(n)$.

Доказ. Операција претраге ће се извршити константан број пута за $n-2$ тачке па је број операција претраге $O(n)$. Преостаје да се утврди очекивани број операција уметања и уклањања.

Пошто се за сваку операцију уклањања извршава одговарајућа операција уметања, и у свим описаним структурама које имају улогу речника асимптотска временска сложеност уметања иста је као и сложеност уклањања, за процену асимптотске временске сложености алгоритма довољно је посматрати само уметања.

Нека је X случајна величина која узима вредност једнаку броју извршених операција уметања, и нека је X_i индикатор догађаја који узима вредност 1 уколико се обрадом i-те тачке у случајном поретку проналази пар тачака на растојању мањем од δ, у супротном узима вредност 0. Јасно је да је

$$X = n + \sum_{i=3}^{n} iX_i$$

Такође важи $P(X_i = 1) \leq \frac{2}{i}$: посматрајмо тачке $p_1, p_2, ..., p_i$ и нека су две тачке са минималним растојањем p и q. $X_i = i$ само ако важи $p = p_i$ или $q = p_i$. Пошто је поредак тачака случајан вероватноћа да се баш p или q налазе на i-тој позицији једнака је $\frac{2}{i}$. Уколико постоји више од једног пара тачака са

једнаким растојањем таквим да не постоји пар тачака са краћим растојањем, тачка p_i никако неће моћи да утиче на вредност δ. Директно следи:

$$E(X) = n + \sum_{i=3}^{n} i \cdot E(X_i) \leq n + 2n = 3n$$

чиме је тврђење доказано.

□

Број операција израчунавања раздаљине, која се извршава у константном времену, мањи је или једнак броју операција претраге, па је очекивана временска сложеност алгоритма $O(n \log n)$ уколико се као речник користе скип листа или трип, или $O(n)$ уколико се користи хеширање.

8

Закључак

Рандомизирани алгоритми проналазе широку примену. Историјски познат пример примене рандомизације била је идеја статистичара Роналда Фишера у пољопривреди. Да би утврдио који хибрид житарице боље опстаје у различитим условима једног пространог имања пројектовао је експеримент који примењује случајност. Овим огледом успешно су упоређени различи хибриди без обзира на спољне услове.

Неке од познатијих савремених примена рандомизације су:

-Рандомизирани тестови прималности

-Савремене C++ библиотеке садрже имплементацију Интросорт алгоритам који је хибрид алгоритма брзог сортирања и алгоритма сортирања уметањем

-У нумеричкој математици рандомизација се примењује при провери да ли је производ две матрице једнак задатој трећој матрици.

-Рандомизација погодна је за поступке брзог индексирања и примењује се у изради пробабилистичких база података.

Са осврtом на садашње околности, рандомизирана контролисана испитивања имају кључну улогу у научно прихваћеним предвиђањима ширења вируса COVID-19.

Уз корисне примене рандомизације, познате су и примери шаљиве бескорисне употребе рандомизације попут чувеног Богосорт алгоритма који сортира низ са n елемената са очекиваном временском сложености $O((n+1)!)$.

Захвалио бих се ментору Јелени Хаџи-Пурић на неизмерној помоћи у реализацији овог рада и за четири године одлично разрађене и савремене наставе програмирања.

Литература

[1] https://web.archive.org/web/20150620160544/http://www.aw-bc.com/info/kleinberg/assets/downloads/ch13.pdf
Датум последњег приступа: 5. октобар 2016.

[2] Motwani, Rajeev; Raghavan, Prabhakar (1995). Randomized Algorithms. New York: Cambridge University Press.

[3] Hoare, C. A. R. (July 1961). "Algorithm 64: Quicksort".

[4] https://web.archive.org/web/20110605030306/http://www.cs.uiuc.edu/class/sp09/cs473/notes/08-treaps.pdf
Датум последњег приступа: 16. јун 2012.

[5] Smid, Michiel (1995). Closest point problems in computational geometry. Max-Planck-Institut für Informatik.