МАТЕМАТИЧКА ГИМНАЗИЈА

У БЕОГРАДУ

МАТУРСКИ РАД

из предмета Програмирање и програмски језици

СЕГМЕНТНО И ФЕНВИКОВО СТАБЛО И ЊИХОВЕ ПРИМЕНЕ У ТАКМИЧАРСКИМ ЗАДАЦИМА

ученик:
Давид Милићевић, 4д

ментор:
Јелена Хаџи-Пурић

Београд, мај 2016.

С А Д Р Ж А Ј

1 УВОД

Од како је Бентли[1] представио сегментна стабла још 1979. године, њихова примена се шири. Од лакших употреба попут RMQ проблема, преко генерализација за више димензија, па све до примена у auto-complete софтверу и развоју одређених врста база података, ова структура је данас врло примењивана у решавању различитих типова проблема.

1994. године, Фенвик[2] представља своју структуру, првенствено намењену за коришћење у алгоритмима компресије података. Иако иницијално није била заснована на сегментим стаблима, веза између ове две структуре је неоспорива.

Због све веће примене ових структура у реалним проблемима, њихова учесталост на такмичењима у програмирању расте, како на домаћим, тако и на страним. На тим такмичењима учествујем још од основне школе, па сам из тог разлога желео да моја тема буде у вези са њима, а како сам и сам имао проблема приликом учења ових структура, надам се да ће мој рад бити од помоћи неком будућем такмичару.

У раду ће прво бити описана структура сегментног стабла и операције над овим стаблом које се користе и све то бити објашњено на неколико такмичарских задатака. Идеја је да задаци не буду много тешки, већ да на адекватан начин илуструју оно о чему се говори. Након тога следи пар мало компликованијих алгоритама који би требало да у већини случајева буду довољни за решавање задатка.

У другом делу се говори о Фенвиковом стаблу, које је интуитивно теже разумети него сегментно, али је разумевање ове структуре и више него корисно, посматрано из такмичарског угла. Такође, овај део је првенствено фокусиран на објашњавање структуре и како и зашто она ради, јер су примери углавном врло слични и два примера довољно добро приказују примерну Фенвиковог стабла.

За имплементацију ове две структуре података коришћен је програмски језик C++, а у приложеним кодовима се могу видети краћа објашњења имплементације описаних алгоритама.

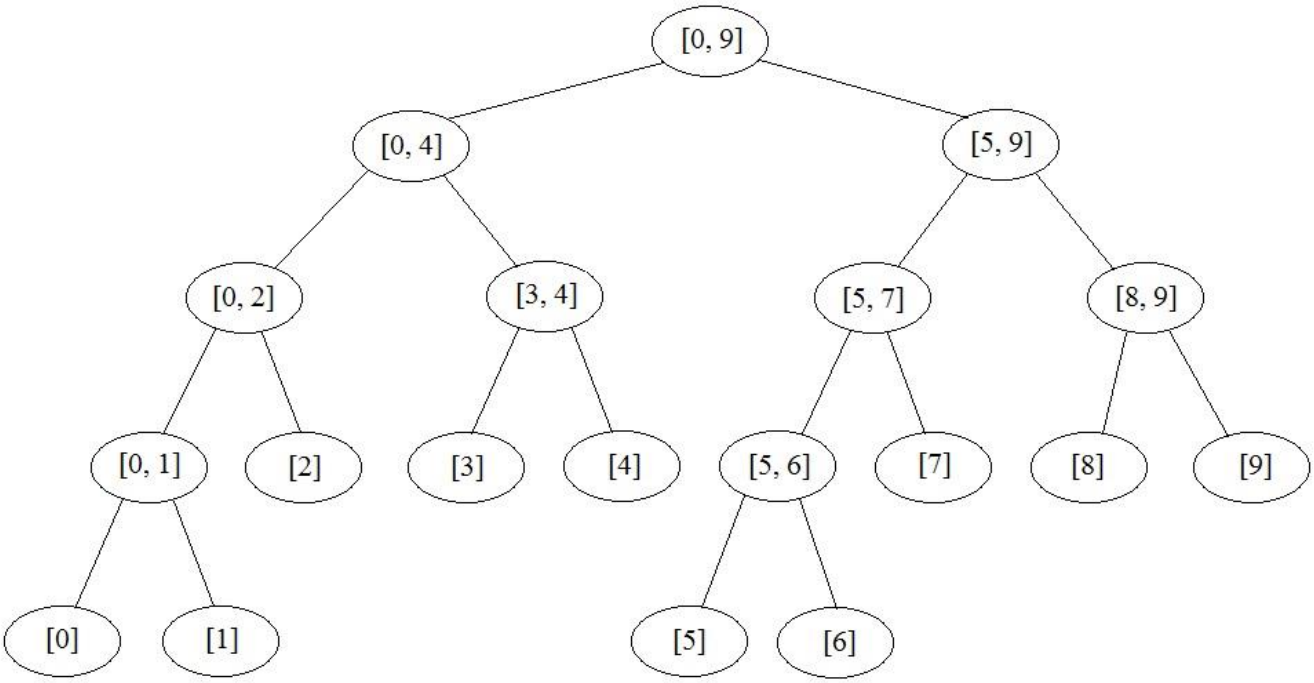
[1] Jon Louis Bentley (рођен 1953), амерички теоријски информатичар
[2] Peter Fenwick, професор са Новог Зеланда

2 СЕГМЕНТНО СТАБЛО

2.1 Структура сегментног стабла и основне операције

Основна идеја је да сваки чвор представља одређени сегмент који је унија сегмената које представљају деца тог чвора. С обзиром да сегментно стабло није проста структура, пре разматрања проблема биће описани структура и основне операције над овим стаблом.

Сегментно стабло је бинарно, што значи да сваки чвор има два сина. Ако неки чвор представља интервал $I = [l, d]$ онда његов леви син представља интервал $I_l = [l, (l + d) / 2]$, док десни син представља интервал $I_d = [(l + d) / 2 + 1, d]$. Деобом интервала долазимо до листова који садрже „јединични“ (елементарни) интервал, за разлику од унутрашњих чворова који садрже уније интервала. Још једно битно својство сегментног стабла јесте да је оно балансирано, па је самим тим његова висина сразмерна са $log_2 n$ (где n представља ширину највећег интервала, тј. оног у корену) Најбоље је структуру приказати сликом:



Слика 1: Структура сегментног стабла

Структура овог стабла је погодна за измену одређеног елементарног интервала[3] (*update* операција), као и за постављање упита (*query* операција) над неким интервалом. Посматара се следећи пример:

Дат је низ са n елемената и постоје две врсте упита:

1. На позицију *idx* поставити вредност *val*
2. Који је минимални елемент у интервалу $[l, r]$?

Овај проблем је познат и као RMQ (Range minimum query) и може се применити и на решавање других проблема. Како би се решио проблем неопходно је смислити како ће стабло бити представљено, а затим и имплементирати одређене операције:

Представљање. Испоставља се да је за имплементацију сегментног стабла неретко довољан само један низ, а не читаво стабло. Као и код осталих балансираних бинарних стабала, ако се неки чвор означи индексом i онда ће његов леви син имати индекс $2i$, а десни $2i + 1$. Такође се врло лако може доказати да ће, ако низ има n елемената, стабло имати највише $2n - 1$ чворова, па се за његово представљање може искористити низ са исто толико елемената (у пракси се узимају низови са $4n$ елемената зато што стабло није увек комплетно, па се може десити да индекс неког чвора премаши вредност $2n - 1$). У сваком чвору стабла чува се минимални елемент у интервалу који му одгорвара.

Изградња. Како би се изградило стабло неопходно је у његове чворове уписати иницијалне вредности. Као и код осталих операција, то је најлакше урадити рекурзијом обилазећи стабло у *post-order*[4] обиласку. Тако ће се најпре доћи до неког листа и у њега ће се уписати одговарајући елемент низа. Приликом посете било ког унутрашњег чвора, његово лево и десно подстабло су већ посећени. Као минимум за тренутни чвор узима се мања вредност од левог и десног сина. Како се сваки чвор посећује највише једном, а чворова има највише $2n - 1$, сложеност изградње је $\Theta(n)$.

Измена (update). Потребно је приметити да својство (у овом случају минимум) неког чвора зависи само од својстава његових синова, тако да ће се променом својства неког од синова, својство чвора потенцијално променити. На основу тога, јасно је да ће, када се измени својство неког елементарног интервала, то утицати на његовог родитеља,

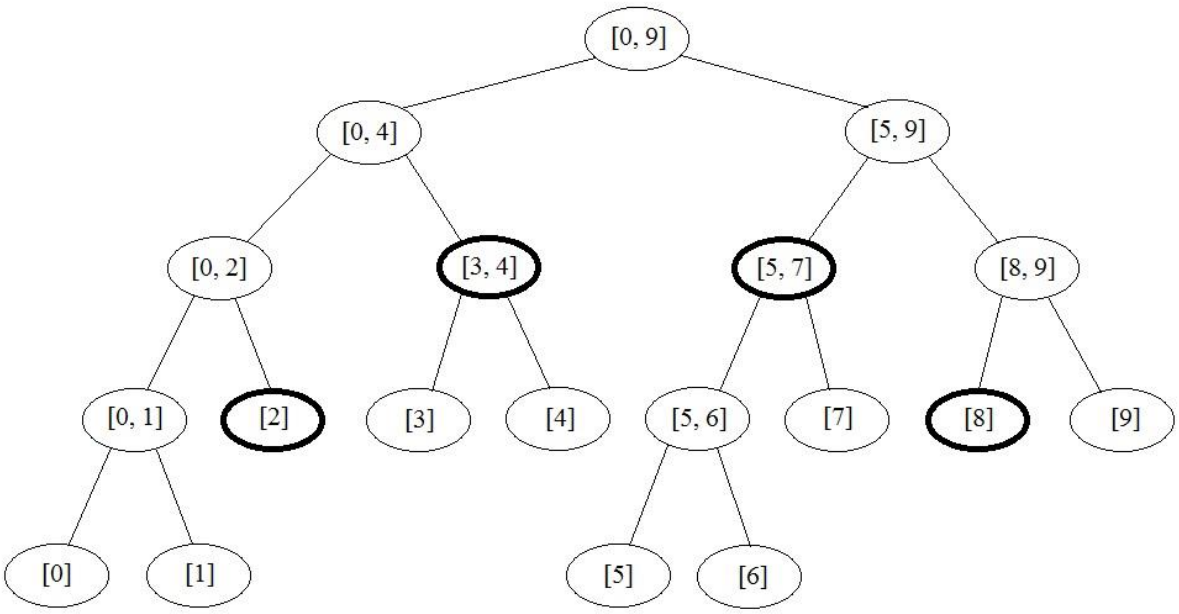
[3] Касније ћемо видети да се уз одређене промене може вршити и измена неког ширег интервала.
[4] *post-order* - обилазак у редоследу: лево подстабло, десно подстабло, тренутни чвор

па на родитеља од родитеља итд. Измениће се највише $log_2 n$ чворова јер је то висина стабла, па је и сложеност ове операције $\Theta(log_2 n)$.

Упит (query). Да би се пронашао одговор за дати интервал, потребно је прво да се он представи као унија што мање дисјунктних интервала који су садржани у сегментном стаблу. Решење ће бити минимум над свим тим интервалима. Проналажење тих интервала се такође ради рекузивно, и у зависности од тренутно посматраног интервала постоје три случаја:

1. Тренутни интервал је подинтервал траженог интервала: у том случају тренутни интервал у целини учествује у израдњи траженог интервала
2. Тренутни интервал и тражени интервал су дисјунктни: онда се претрага не наставља у тренутном подстаблу јер ниједан интервал сигурно неће градити тражени интервал
3. У преосталим случајевима, када тренутни и тражени интервал имају неки пресек, тренутни интервал неће учествовати у подели, али неко од његових потомака хоће и претрага се наставља дубље у подстаблу

Сложеност упита је $\Theta(log_2 n)$. У приложеном коду се могу видети детаљи имплементације ових операција.



Слика 2: Пример уније интервала за упит [2, 8]

2.2 Основне примене

Пример 1. Дат је низ са n елемената и две врсте операција: променити одређени елемент низа и одредити суму датог интервала.

Овај пример је врло сличан претходно објашњеном RMQ проблему и приказан је како би се увидела сличност са њим. Једина разлика је у томе што се сада у једном чвору више не чува минимум, већ сума свих елемената у том интервалу. Када се промени неки елемент низа, уз помоћ *update* операције промениће се и суме свих интервала у стаблу којима он припада.

Пример 2. Дат је израз који се састоји само од отворених и затворених заграда. Треба подржати две операције: неку заграду заменити са њом супротном и одговорити на питање да ли је цео израз исправан. Израз је исправан ако се заграде могу упарити тако да свака отворена заграда има одговарајућу затворену и ако се између њих налази подједнак број отворених и затворених заграда.

На први поглед делује као да овај задатак нема везе са сегментним стаблима, међутим, поред осталих решења, решење са сегментним стаблом је врло једноставно. У сваком чвору се чувају два податка: број неупарених отворених заграда (o) и број неупарених затворених заграда (z). Цео израз је исправан ако за корен стабла важи да је $o = 0$ и $z = 0$. Када се нека заграда промени, мењају се и вредности o и z за поједине чворове. Нека су *levi* и *desni* леви и десни син тренутно посматраног чвора. Број отворених заграда за тренутни чвор, у том случају је $o = desni.o + max(0, levi.o – desni.z)$. Ова формула важи зато што не постоје неупарене затворене заграде са којима могу да се упаре неупарене отворене заграде десног сина, док неупарене отворене заграде левог сина могу да се упаре са неупареним затвореним заградама десног сина. Аналогно важи и $z = levi.z + max(0, desni.z – levi.o)$.

Пример 3. За дати низ A са n елемената који могу бити негативни треба имплементирати следећа два упита:

- $0\ x\ y$ – променити x-ти елемент у y
- $1\ x\ y$ – исписати $max(A_i + A_{i+1} + \ldots + A_j \mid x \leq i \leq j \leq y)$

Овај задатак се такође може решити са већ постојећим операцијама у стаблу. Сваки чвор сада има четири поља: суму сегмента (*seg*), најбољу префиксну суму (*pref*), најбољу суфиксну суму (*suf*) и најбољу суму која се може остварити у сегменту (*best*). За

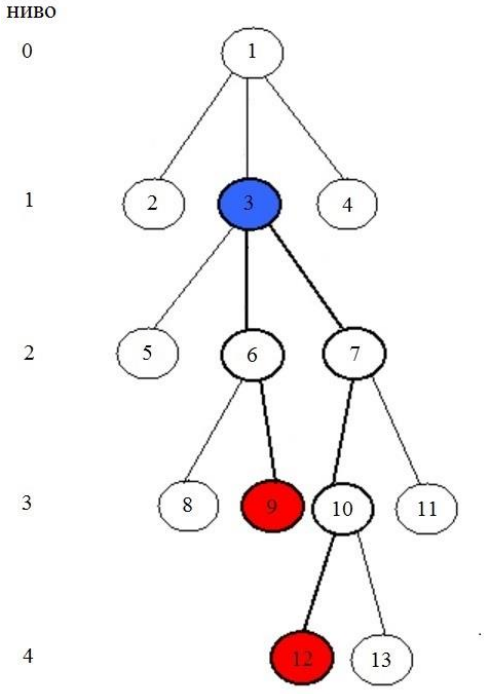
сваки чвор се ове вредности, на основу левог (*levi*) и десног (*desni*) сина мењају на следећи начин:

- *seg = levi.seg + desni.seg*
- *pref = max(levi.pref, levi.seg + desni.pref)*
- *suf = max(desni.suf, desni.seg + levi.suf)*
- *best = max(levi.best, desni.best, left.suf + desni.pref)*

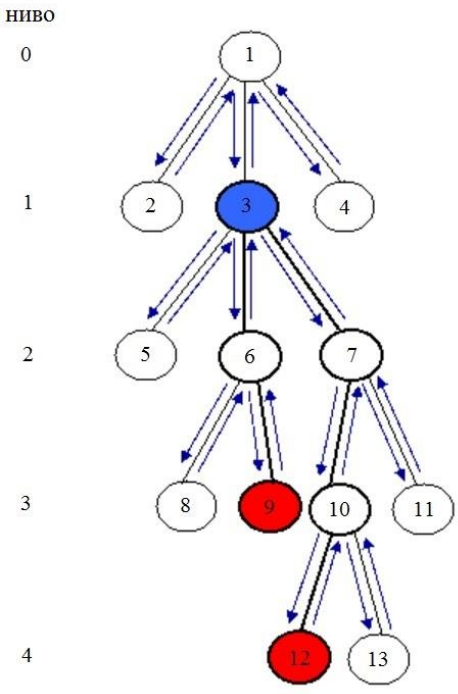
2.3 Аналогија између RMQ и LCA

LCA (Lowest common ancestor) је проблем у коме се за дато уређено стабло поставља питање који је први заједнички предак чворова *x* и *y*. Иако постоје ефикасни алгоритми који нису у вези са сегментним стаблом, овај пример је пре свега занимљив зато што се идеја може применити и у неким другим случајевима.

Дакле, потребно је превести проблем налажења LCA у проблем RMQ. Ово је најлакше објаснити на основу примера. Посматра се DFS (Depth first search) обилазак стабла са почетним чвором у корену.



Слика 3: LCA чворова 9 и 12 је чвор 3

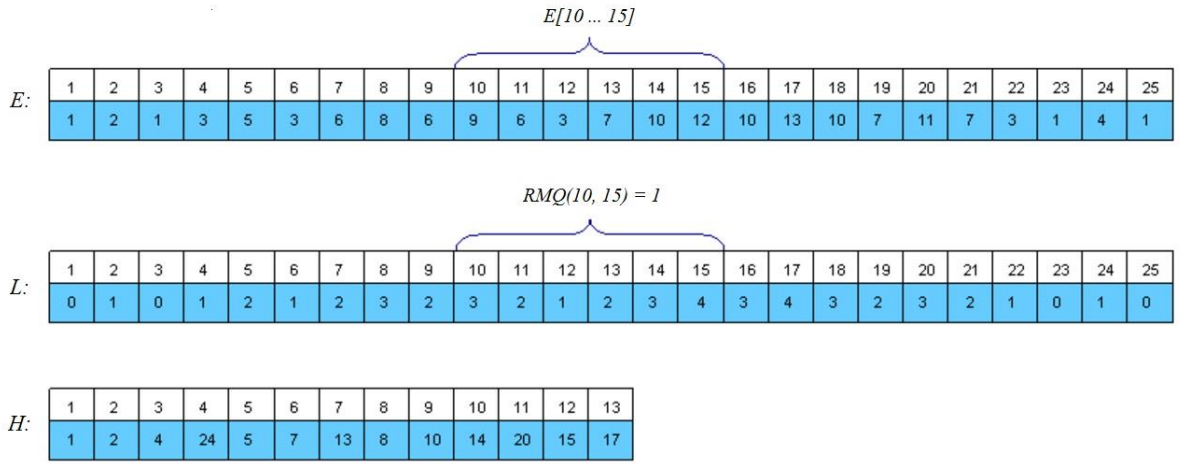


Слика 4: Обилазак

Може се приметити да ће *LCA(x, y)* бити посећен између посећивања чворова *x* и *y*. Из тог разлога кандидати за *LCA(x, y)* су само чворови који су посећени након чвора *x*, а пре чвора *y*. Потребно је направити три низа:

- *E[1, 2N-1]* – чворови посећени у обиласку. *E[i]* је индекс *i*-тог посећеног чвора у обиласку
- *L[1, 2N-1]* – *L[i]* је ниво на коме се налази чвор *E[i]*
- *H[1, N]* – *H[i]* је индекс првог појављивања чвора *i* у низу *E*

Уз претпоставку да је *H[x] < H[y]* (у супротном треба заменити вредности *x* и *y*) је лако увидети да су чворови који су посећени између *x* и *y* заправо *E[H[x] ... H[y]]*. Сада је преостало да од тих чворова пронађемо онај са најмањим нивоом. Одговор на питање је зато *LCA(x, y) = E[RMQ(H[x], H[y])]* ако је функција *RMQ* имплементирана да врати индекс најмањег елемента или *LCA(x, y) = RMQ(H[x], H[y])* у случају када враћа најмањи елемент (што је и случај у приложеном коду).



Слика 5: Пример изгледа низова E, L и H

2.4 Lazy propagation

Проблем код сегментног стабла се јавља када је потребно изменити интервал вредности, а не појединачан елемент. Када би се, као и до сада, за сваки елемент из интервала позивала фукнција *update*, овакав низ операција би у најгорем случају довео до сложености која је сразмерна са $\Theta(n)$, али се у пракси понаша доста лошије. *Lazy propagation* је техника која омогућава да се измене интервала врше са истом асимптотском сложеношћу $\Theta(\log_2 n)$, као што је био случај код појединачних измена.

Техника ће опет бити објашњена на примеру RMQ, само што ће сада бити дозвољено да се мењају вредности из неког интервала (нпр. операција ће захтевати да се елементи из неког интервала увећају за вредност *val*). Као што назив можда сугерише (у буквалном преводу: лење пропагирање), сваком чвору у стаблу се додаје још једна нова променљива: *lazy*. До тренутка када није искоришћена, *lazy* ће бити постављена на $+\infty$. Када се врше измене на интервалу елемената, рекурзијом треба доћи до истог скупа интервала сегментног стабла као и када се поставља упит у стаблу, тј. позива функција *query* (видети слику 2). У сваком од тих чворова, вредност *lazy* поља се поставља на *val* уколико важи $val < lazy$ (у листовима се вредности минимума могу одмах поставити на *val*). Када се позове *query* функција која захтева да се посети син чвора у коме је *lazy* променљива постављена на неку вредност (тј. није $+\infty$) вредност *lazy* се „шаље“ у његове синове (вредности *lazy* поља синова се мења у зависности од *lazy* поља родитеља), док се *lazy* поље родитеља опет поставља на $+\infty$. Ако се, којим случајем, приступа чвору, а да се не посећују његови директни потомци, довољно је само узети у обзир његову *lazy* вредност.

У приложеном коду постоје даља детаљнија објашњења како би ова техника била јаснија.

Пример 4. Дат је низ са N елемената који су иницијално постављени на 0. Након тога је дато C команди, које могу бити следећих типова:

- *0 p q val* – додати вредност *v* на све елементе у интервалу $[p, q]$
- *1 p q* – исписати суму свих елемената у интервалу $[p, q]$

Идеја је иста као и у примеру 1. Разлика је што сада треба додати *lazy* поље. У њему ће се чувати сума коју треба „послати“ потомцима. Иницијално, *lazy* је постављена на 0. Све остало је исто као и у објашњеном примеру са RMQ, осим што треба водити рачуна да *lazy* сада чува суму, а не најмањи елемент (нпр. више неће бити $lazy = min(lazy, val)$, него ће бити $lazy = lazy + val$).

2.5 Бинарна претрага

Често је потребно пронаћи лист у стаблу који задовољава неко својство. У ту сврху неопходно је претражити стабло. Претрага се врши од корена ка листовима и то у сваком тренутку одлучујући да ли ће се ићи у лево или десно подстабло.

Пример 5. У овом примеру је неопходно одржавати динамички скуп бројева који подржава две основне операције:

- *Insert(S, x)* – ако *x* није у *S*, убаци *x* у *S*
- *Delete(S, x)* – ако *x* јесте у *S*, избриши *x* из *S*

и два типа упита:

- *K-th(S)* – испиши *k*-ти најмањи елемент у *S*
- *Count(S, x)* – испиши број елемената у *S* који су мањи од *x*

Insert и *Delete* су прости примери измене елементарних интервала, док је *Count(S, i)* такође пример простог упита на интервалу [*0*, *i*]. Међутим, како би се реализовао упит *K-th* треба имплементирати бинарну претрагу у стаблу на следећи начин: у сваком чвору у стаблу се памти колико има елемената у левом и десном подстаблу (нека су то бројеви *levo* и *desno*). Уколико је $k \leq levo$, посећује се лево подстабло, док се у супротном случају посећује десно. Процес се понавља док се не дође до неког од листова. Тада је вредност која одговара том листу тражени број.

2.6 Перзистентно сегментно стабло

Перзистентне структуре подататака су структуре које увек одржавају своје претходне верзије након извршених измена. Овакве структуре су ефективно непроменљиве зато што операције над њима не мењају њихову структуру, већ уместо тога стварају нове измењене структуре (нова структура је измењена стара).

Иако задаци који користе ову структуру нису толико учестали, она је овде представљена првенствено због занимљиве идеје, а и да би се показало како сегментно стабло не може увек бити имплементирано уз помоћ низа, већ се мора направити стабло.

Пример 6. За дати низ *a* са *N* елемената одговорити на низ питања типа: „Који би био *k*-ти елемент по величини да је сегмент *a*[*i* … *j*] сортиран?“

Основна идеја: За свако *i* и *j* ($1 \leq i \leq j \leq N$) постоји сегментно стабло. Ако је то случај, онда се свако питање може решити у $\Theta(log_2 N)$ применом бинарне претраге (2.5). Међутим, проблем је што изградња тих сегментних стабала захтева $\Theta(N^3)$ меморије и времена.

Запажање 1: У сегментном стаблу за интервал [*i*, *j*] сваки чвор може бити израчунат преко одговарајућих чворова у сегментним стаблима за интервале [*1*, *i* – *1*] и

[*1, j*]. Сада је неопходно свега *N* сегментних стабала, свако стабло за по један префикс низа *a*.

Запажање 2: Сегментно стабло за све префиксе до *i* је скоро исто као стабло за префикс *i – 1*, осим $\Theta(log_2 N)$ чворова који ће бити промењени (на свакој дубини у стаблу ће бити промењен само један чвор, и то баш онај који садржи елемент који се додаје).

Запажање 3: Елементе низа *a* је неопходно мапирати у вредности које ће припадати интервалу [*1, N*] како би уопште била могућа изградња сегментних стабала.

Изградња стабла: Нека су до сада изграђена стабла за префикс *i*. При изградњи стабла за префикс *i + 1* треба незнатно модификовати функцију *update*. Функција се креће кроз стабло за префикс *i*. Уколико елемент који се додаје не припада тренутно посматраном интервалу значи да се тренутни чвор (заједно са својим потомцима) неће мењати, па је у том случају могуће искористити већ постојећи чвор и само преусмерити показивач на њега. У супротном, треба направити нови чвор и, уколико он није лист, рекурзивно позвати функцију како би се одредила његова деца.

Питања: Одређивање *k*-тог елемента по величини је већ обрађено у примеру 5. Сада је потребно мало изменити бинарну претрагу јер се посматрају два сегментна стабла, она која одговарају префиксима *i – 1* и *j*. Међутим, разлика је врло мала. Број елемената у левом подстаблу је број елемената у левом постаблу за сегментно стабло *j* минус број елемената у левом подстаблу за стабло *i – 1*. Нека је то број *br*. Ако важи $k \leq br$ онда се у левом подстаблу тражи *k*-ти елемент, у супротном се у десном подстаблу тражи (*k – br*)-ти елемент (у левом подстаблу их има *br*).

2.7 Напомене

Углавном се, приликом имплементације, користе низови дужине 2^k. Ако је низ дат на улазу краће дужине, вишак елемената се допуњава погодним вредностима за то (нпр. код задатка са сумом интервала, вишак елемената се допуњава нулама, док се код RMQ задатка вишак елемената допуњава неким великим бројевима). Такви низови се користе не због неке суштинске разлике, већ ради лакше анализе проблема, јер је у том случају сегментно стабло комплетно (има $2^{k+1} - 1$ чворова).

Такође, уколико се користи низ дужине 2^k, онда се *update* операција може извршавати итеративно (а не рекурзивно, као што је објашњено) зато што се у том

случају за сваки елемент низа зна који је његов одговарајући лист у стаблу (за i-ти елемент низа чвор је $2^k - 1 + i$ зато што унутрашњих чворова има тачно $2^k - 1$).

У општем случају, сегментна стабла се користе над целом реалном правом и унија свих интервала би требало да представља скуп реалних бројева. На пример, нека је S скуп интервала, а $p_1, p_2, \ldots, p_m$ различите тачке које представљају крајеве интервала. Ове тачке деле реалну праву на $m+1$ делова. У овом случају, елементарни интервали су редом $(-\infty, p_1)$, $[p_1, p_1]$, (p_1, p_2), …, (p_{m-1}, p_m), $[p_m, p_m]$, $(p_m, +\infty)$.

За разлику од њих, овде су објашњена сегментна стабла на испрекиданом скупу елементарних интервала, из разлога што су општа сегментна стабла једноставно уопштење описаних и углавном се користе у геометријским проблемима, који нису толико учестали као проблеми који су наведени и објашњени. Ономе ко је разумео описана стабла сигурно неће бити проблем да се пребаци на њихов општији облик, који је, као што се могло видети, у многим случајевима непотребан.

3 ФЕНВИКОВО СТАБЛО

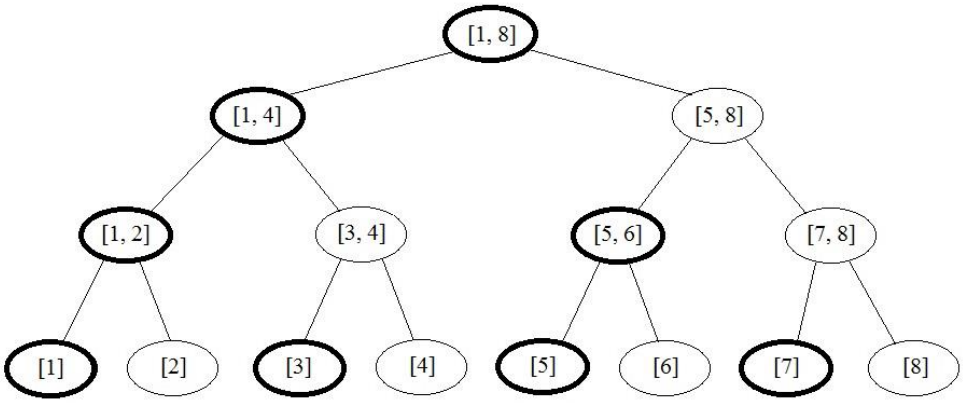
Може се приметити да се задаци у којима се примењују сегментна стабла врло често могу свести на имплементацију две операције:

1) Измена вредности елементарног интервала
2) Одређивање својства које зависи од вредности елемената из опсега [*i, j*]

Углавном се испоставља да се операција 2) може упростити тако да се одређује својство које зависи од елемената из интервала [*1, j*]. У том случају својство за интервал [*i, j*] се може одредити на основу својстава за интервале [*1, i – 1*] и [*1, j*]. Када дође до такве ситуације, углавном је, што због лакше имплементације, што због боље ефикасности, пожељније користити **Фенвиково стабло**[5].

3.1 Структура Фенвиковог стабла

Због једноставности, нека је дужина посматраног низа $n = 2^k$. Над датим низом би требало изградити сегментно стабло. Међутим, може се приметити да за представљање свих интервала облика [*1, i*] нису потребни сви чворови сегментног стабла, већ се скуп посматраних чворова може редуковати.

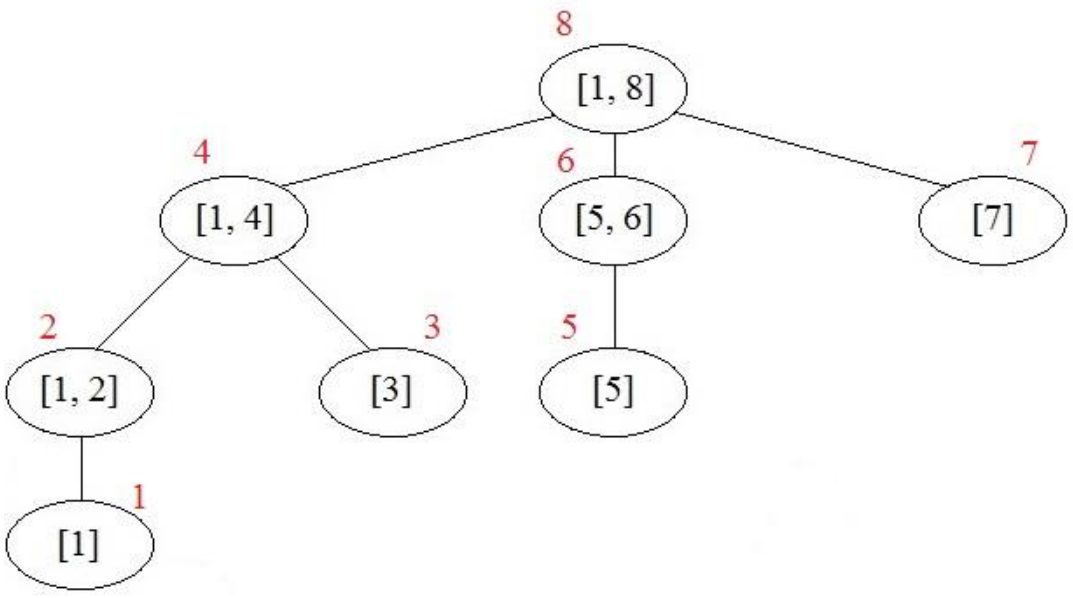


Слика 6: Минималан скуп чворова који су неопходни за представљање интервала облика [1, i] ($1 \leq i \leq 8$)

Пре свега, суштински битна ствар јесте да су сви интервали у означеним чворовима дужине 2^k (за неко k). Очигледно се сваки интервал облика [*1, i*] може представити као унија одређеног броја дисјунктних интервала чије су дужине степени броја *2*. На пример, интервал [*1, 3*] је унија интервала [*1, 2*] и [*3, 3*]; интервал [*1, 7*] је унија [*1, 4*], [*5, 6*] и [*7, 7*]. Одатле потиче идеја да се мало дубље анализира бинарни запис броја *i*.

[5] Користе се и следећи називи: **кумулативно стабло** или **binary indexed tree** (BIT)

Интервал [*1, 13*] ће бити унија интервала [*1, 8*], [*9, 12*] и [*13, 13*], а како је *13 = 8 + 4 + 1*, постаје јасно да се интервали бирају тако што се прво узме највећи интервал мањи од *13*, а затим се узимају све мањи и мањи интервали, док њихова дужина у збиру не буде одговарајућа. Тачније, како је $13 = 1101_2$ интервали ће бити дужина 1000_2, 0100_2 и 0001_2. Гарантује се да никад неће бити изабрана два интервала исте дужине (нпр. никада неће бити изабрана два интервала дужине *2* јер би у том случају био изабран одговарајући интервал дужине *4*).



Слика 7: Један од начина представљања Фенвикового стабла

3.2 Операције

Сада ће бити објашњено како се реализују операције 1) и 2) са почетка главе. Како би то уопште било могуће, прво треба смислити начин представљања Фенвиковог стабла. Као што је био случај и са сегментним стаблима, стабло такође не постоји, већ је за представљање довољан низ од *n* елемената, где ће *i*-ти елемент представљати чвор из стабла чији се интервал завршава у тачки *i* (на слици 7 су, између осталог, за сваки чвор приказани индекси из низа који му одговарају).

Ради лакшег разумевања, операције су објашњене на примеру 1 који је коришћен и код сегментних стабала (измена појединачних елеменатат и сума интервала):

1) Када се промени неки елемент низа (нека је то елемент *i*), тада се мења и чвор *i*. Поред тога, неопходно је извршити промене и у свим чворовима који се налазе на путу од корена до чвора *i* (Слика 7: ако се, на пример, промени елемент *2* потребно је променити чворове *2, 4* и *8* јер сви они

садрже тај елемент). Од чвора i до корена се стиже применом битовског оператора and (конјункција): $i = i + i$ and $(-i)$. Тако, након сваког корака, чвор i постаје свој родитељ (касније ће бити објашњено зашто је ово тачно).

2) Да би се одредила сума S која представља суму интервала $[i, j]$, прво треба одредити суме S_{i-1} и S_j које представљају суме интервала $[1, i-1]$ и $[1, j]$, редом. Тада је $S = S_j - S_{i-1}$. Процес одређивања суме S_i је супротан од претходне операције. Кроз стабло се сада креће применом израза $i = i -$ and $(-i)$. Решење је сума вредности у свим чворовима који су посећени.

3.3 *i and (–i)*

Иако се, са свим описаним до сада, Фенвиково стабло може користити без проблема, корисно је разумети значење израза i and $(-i)$ и зашто се баш овај израз примењује на описани начин.

Поменути израз се користи како би се у бинарном запису броја i одредио најлакши бит који је постављен на 1. Зашто је неопходно одредити последњи бит за који то важи биће објашњено касније, док ће сада бити објашњено зашто се за то користи баш овај израз.

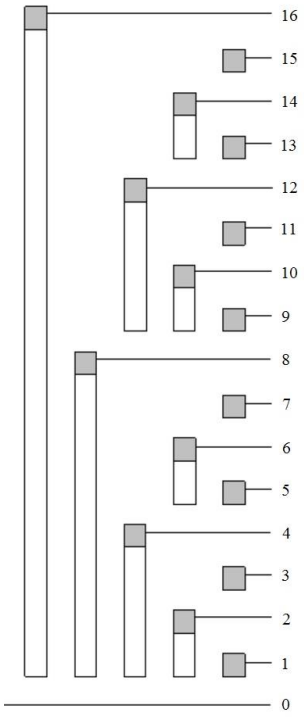
Нека је n број чију је најлакшу јединицу потребно изоловати. У бинарном запису број n се може представити као $a1b$, где a представља бинарне цифре пре последње јединице, а b нуле после ње. Ако се са x^- означи комплемент броја x (број чије су бинарне цифре инвертоване у односу на x: 0 прелази у 1, а 1 у 0) тада важи да је $-n = n^- + 1$ (потпуни комплемент). Дакле, $-n = (a1b)^- + 1 = a^-0b^- + 1$. Пошто се b састоји од свих нула, b^- се састоји од свих јединица. Коначно, $-n = a^-0(0\ldots0)^- + 1 = a^-0(1\ldots1) + 1 = a^-1(0\ldots0) = a^-1b$. Сада је јасно да се применом израза n and $(-n)$, тј. $(a1b)$ and (a^-1b) добија најлакши бит који је постављен на 1.

Сада се може прећи на објашњавање операција:

1) Као што је раније речено, од почетног чвора n треба проћи кроз све чворове на путу до корена, тако да у сваком кораку треба одредити родитеља тренутног чвора. Анализом бинарних записа неког чвора и његовог родитеља закључује се да се родитељ добија тако што се последњи низ јединица замени нулама, а прва нула веће тежине замени јединицом. То се врло лако постиже сабирањем броја n и броја који се добија од броја n тако што се узме само јединица најмање тежине, тј. броја n and $(-n)$. На пример,

родитељ чвора *6* је чвор *8* ($0110_2 + 0010_2 = 1000_2$). Пошто је висина стабла log_2N, где је N број елемената низа, сложеност ове операције је $\Theta(log_2N)$.

2) Друга операција је за нијансу једноставнија од прве. Нека је r позиција најлакше јединице у бинарном запису броја n. Тада чвор n одговара интервалу $[n - 2^r + 1, n]$. За одређивање броја 2^r опет ће се користити израз n and $(-n)$. Сада треба интервал $[1, n]$ поделити на одговарајуће интервале чије су дужине степени двојке.



Слика 8: Одговорности

На слици 8 су приказане одговорности за чворове до *16* (интервали који одговарају датим чворовима). Ова операција се може извршити тако што се на решење дода сума чвора n, а затим се из истог чвора уклони најлакша јединица у бинарном запису и овај процес се понавља док се не дође до нуле. На пример, за чвор $n = 13$ разлагање је $01101_2 \rightarrow 01100_2 \rightarrow 01000_2$. Јасно је да је ово најлакше постићи применом израза $n = n - n$ and $(-n)$. За представљање броја N у бинарном систему потребно је log_2N цифара, а како се у сваком кораку мења једна цифра бинарног записа, сложеност ове операције је такође $\Theta(log_2N)$.

Детаљи имплементације се могу погледати у приложеном коду.

Пример 7. RMQ проблем (коришћен при објашњавању сегментних стабала).

Главна разлика у односу на претходно описани проблем је у томе што се у чворовима више не чува сума интервала, већ минимални елемент у одговарајућем интервалу. Другу операцију, упит за минимум на одређеном интервалу, је потребно незнатно изменити. Уместо збира свих посећених чворова, она треба да врати минимум, што је једноставно уз претходне измене стабла. Међутим, прва операција, тј. промена вредности елемента постаје компликованија. Немогуће је само проћи кроз све чворове до корена, већ је неопходно прво променити одговарајући елемент у низу, а затим за сваки чвор покренути упит који је најмањи елемент у њему одговарајућем интервалу. Ту вредност треба заменити новом вредношћу уколико је она мања од претходне. С обзиром

да се у овом примеру у сваком кораку покреће по један упит, сложеност ове операције постаје $\Theta(log_2^2 N)$.

3.4 Напомене

Сваки проблем који је решив Фенвиковим стаблом, може се решити и сегментним, док супротно не важи. Међутим, када год је то могуће, требало би користити Фенвиково стабло, а разлози су вишеструки. Иако је концептуално теже схватити Фенвиково стабло, његова имплементација је поприлично лакша у односу на сегментно. Такође, ефикасност је на страни Фенвиковог стабла (Фенвиково стабло је редуковано сегментно, па самим тим има мање чворова; имплементација Фенвиковог стабла је итеративна, док је имплементација сегментног рекурзивна; број операција које се извршавају у Фенвиковом стаблу је и до неколико пута мањи, у зависности од случаја).

Није наведено више примера из разлога што су сви примери суштински исти и захтевају мало промена у коду. Такође, у приложеним кодовима за пример 7 и пример на којем је објашњавано стабло су приказане две различите имплементације функције *query*. Једна од њих је углавном довољна за решење проблема.

Оно што није било наведено, првенствено због сложености анализе проблема, јесте да је уз помоћ Фенвиковог стабла такође могуће вршити измене на неком интервалу, а не само појединачних елемената. Тај проблем није могуће реализовати уз помоћ само једног стабла, већ су неопходна два, па је јасно да се проблем доста компликује. Иако није објашњено детаљније, међу приложеним кодовима налази се и код који решава овај случај. Наравно, имплементација се у пракси опет показује бољом него код сегментних стабала.

4 ЗАКЉУЧАК

На крају, може се приметити да скоро сви примери који су обрађени захтевају познавање искључиво сегментног или Фенвиковог стабла и ни на који начин нису повезани са неким другим областима. Једини разлог за такав приступ лежи у томе што су ове структуре довољно распрострањене да је о њиховим применама тешко писати у једном раду.

Како њихова распрострањеност расте, несумњиво је да ове структуре постају неопходне једном успешном такмичару, тако да се надам да ће ономе ко буде читао овај рад, то пружити довољно добар увод и бити подлога за даље напредовање и изучавање сегментних и Фенвикових стабала, као и да ће некоме мало искуснијем пружити објашњење за нејасноће.

Захвалио бих се професорки Јелени Хаџи-Пурић, не толико због њене помоћи на изради рада, колико због њене помоћи у припремању за такмичења када је она била потребна и односа према часовима програмирања који су у мени пробудили додатну жељу да напредујем и схватим да је баш то оно што волим и желим да радим.

5 ЛИТЕРАТУРА

[1] Миодраг Живковић, Алгоритми, poincare.matf.bg.ac.rs/~ezivkovm

[2] Robert Sedgewick, Kevin Wayne, Algorithms Fourth Edition, Addison-Wesley, Westford, Massachusetts, 2011.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms Third Edition, The MIT Press, Cambridge, Massachusetts, 2009.

[4] Binary Indexed Trees, www.topcoder.com

[5] Range Minimum Query and Lowest Common Ancestor, www.topcoder.com

[6] Segment tree, en.wikipedia.org/wiki/Segment_tree

[7] blog.anudeep2011.com/persistent-segment-trees-explained-with-spoj-problems

[8] e-maxx.ru/algo