Математичка гимназија

МАТУРСКИ РАД

из предмета *Рачунарство и информатика*

Карактеристична бинарна стабла са геометријским применама

Ученик
Андреј Ивашковић, IVd

Ментор
Јелена Хаџи–Пурић

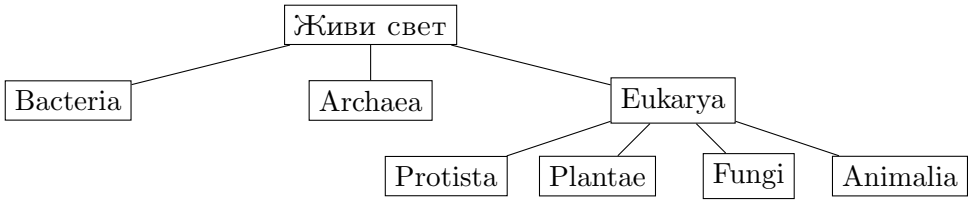
Београд, јун 2014.

Садржај

1 Увод

У основи свих рачунарских програма се налази нека обрада података. Како је неретко број тих података велик, поставља се питање: како их организовати? Одговор: **структуре података** представљају уређене скупове података над којим могу да се врше неке утврђене операције применом одређених алгоритама. Овако описане су свеприсутне у програмирању[1], будући да оне могу бити и једна променљива неког основног типа (број, знак), али и разне организоване колекције елемената (под елементом се подразумева променљива неког простијег типа). Како једна структура не може да ефикасно одговори на сваки упит (обично се испостави да је поступак налажења одговора временски захтеван), јавља се потреба за одабиром структуре података у зависности од задатка — тачније, од операција. Док се структуре података дефинишу и својим операцијама и својим имплементацијама, **апстрактни типови података** (или **апстрактне структуре података**) се дефинишу искључиво својим операцијама. У овом раду ће бити обрађене импементације и примери примена структура података којима је заједничко својство да имају структуру **бинарног стабла**.

Појам стабала је старији од рачунарства и има свој почетак у теорији графова. Није познато када су се први пут проучавала њихова својства, али назив *tree* први користи Кејли[2] у свом раду из 1857. године. Док је уобичајена и најједноставнија математичка дефиниција стабла «неусмерен, ацикличан и повезан граф», он неће бити од нарочите користи при изради ових структура података, где је неопходно увести својеврсну хијерархију међу подацима (пример мотивације је подела сектора неке компаније и све добро конципиране поделе (слика 1), али «породично стабло» није добра аналогија). Зато се некада користи назив «корено стабло» или «уређено стабло». Ради једноставности, овде ће се за њих користити само назив «стабло». Дакле, из свега овог следи да се стабла користе кад год је неопходно представити неку врсту хијерархије међу компонентама, које називамо **чворовима**. Овакво неформално објашњење упућује на то да стабло још увек не може да се назове структуром података, већ само концептом, а евентуално и апстрактним типом података.



Слика 1: Систематика живог света по Воузу

[1] У [2] је дато следеће «интуитивно» објашњење (следи превод):

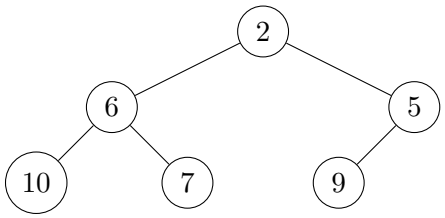
Структуре података су камен темељац рачунарских алгоритама. Конструисање алгоритма је попут консруисања зграде... Да би се то постигло, није довољно познавати функционалност, ефикасност, форму и лепоту. Неопходно је и детаљно знање о техникама изградње... На исти начин, конструисање алгоритма мора да буде засновано на детаљном разумевању техника структура података и цена.

Ово такође илуструје и наслов једне књиге Вирта (Niklaus Wirth (1934–), швајцарски информатичар, добитник Тјурингове награде 1984. године, аутор програмских језика Pascal, Oberon, Modula...): Algorithms + Data Structures = Programs.

[2] Arthur Cayley (1821–1895), енглески математичар

При том се посебно истичу **бинарна стабла**, где је сваки чвор на хијерархијски директно вишем нивоу од највише два чвора (слика 2).

Нека од значајних (не нужно бинарних) стабала и неке од њихових основних примена су наведене:



Слика 2: *Binary min-heap* је пример бинарног стабла

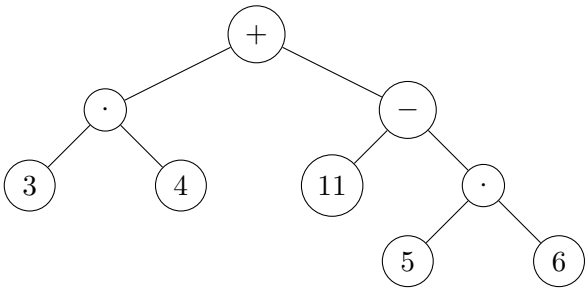
Бинарно стабло претраге. Омогућава брзу манипулацију подацима ако је реч о операцијама додавања новог чвора у стабло, брисања неког постојећег чвора стабла, одређивању да ли се чвор налази у стаблу. Постоји неколико структура података које решавају овај проблем асимптотски оптимално (све операције имају временску сложеност $\Theta(\log n)$, где је n број чворова у стаблу): поред наивног бинарног стабла претраге, ту су и AVL стабло, *red-black tree* итд. Примене су многобројне, а некада су у основи неких других стабала.

Heap. Опште име за читаву фамилију бинарних стабала које врше операције додавања новог чвора, одређивања екстремалног чвора (минимума или максимума), брисања екстремалног чвора (све структуре података које одговарају на овакве упите, не нужно стабла, називају се приоритетним редовима). При томе се тежи да све операције имају временску сложеност $\Theta(\log n)$, где је n број чворова, али у најједноставнијим задацима је одређивање екстремума тривијална операција са $\Theta(1)$ сложеношћу. Уз ситне измене је могуће ефикасно брисање било ког чвора, али задаци у којима се уводе сложеније операције (нпр. постоји неколико независних стабала над којима се врше ове операције, али их је могуће у једном тренутку спојити у једно) захтевају другачије имплементације и некада морају да жртвују $\Theta(1)$ сложеност одређивања екстремума (нпр. биномни *heap*). Примене су честе, посебно у сортирању (*Heap sort*), теорији графова (Дајкстрин алгоритам), симулацији узрочно-последичних дискретних догађања у времену (често у спрези са тополошки сортираним усмереним графовима) итд.

Кодирање и компресија. Хафманов код, који представља популаран начин компресије података без њиховог губитка, често се сликовито приказује стаблима. У овом раду ће бити даље поменут и аритметички код.

Базе података. Пре «победе» релационог модела база података, уз мрежни модел је био популаран и хијерархијски, који директно може да се прикаже стаблима. Данас, B стабло и њему слична се примењују у имплементацији индексне архитектуре и налазе примену у популарним релационим база података (на пример, MySQL базе).

Запис аритметичких израза. Аритметички изрази у стандардној (инфиксној) нотацији могу да се представе у облику стабла: хијерархијска структура омогућава да се одреди редослед којим је неопходно вршити операције (слика 3).



Слика 3: Стабло које одговара изразу $3 \cdot 4 + (11 - 5 \cdot 6)$

Функционисање логичких програмских језика. Као последица претходног, издваја се поступак израчунавања одговора у логичким програмским језицима (Prolog). При том се овде под аритметичким изразима мисли на оне сачињене од логичких операција конјункције, дисјункције, негације итд. Стабло израчунавања одговара се индукује применом тзв. метода резолуције.

Са друге стране, у рачунарству су изузетно значајни појмови познати из геометрије, али и из линеарне алгебре и аналитичке геометрије (области које су веза између геометрије и алгебре): тачка, права, вектор, полигон, конвексни омотач... Уз одговарајуће алгоритме који одговарају на најразличитије упите над њима, могуће се примене у најразличитијим областима, при чему наводим карактеристичне (које се међу собом често преплићу):

Computer Vision. Ова изузетно широка област (буквално преведена као «рачунарска визија») бави се темама које се тичу препознавања облика. Ово је нашло примену у издвајању предмета са слике, идентификацији (отисак прста, снимак вене или дужице), процени кретања предмета на слици, SLAM (*simultaneous localization and mapping* — истовремена локализација и мапирање «посматрача») и многим другим областима.

Линеарно програмирање. Линеарни програми решавају проблеме типа: одредити екстремне вредности линеарне функције $f(x_1, x_2, \ldots, x_n)$ уколико за x_1, x_2, ..., x_n важе ограничења дата системом линеарних неједначина[3]. Они су од изузетног значаја у планирању производње. *Симплекс алгоритам* их решава, а у његовој основи су знања о вишедимензионалним просторима.

Машинско учење. И ова област је изузетно обимна и заснива се највише на хеуристикама. Њена веза са геометријом се успоставља графичким приказом података. Један од основних задатака је тзв. *data mining*, где је неопходно уочити правилности у подацима исте класе. Један од приступа јесте да се подаци представе као тачке у простору (у једноставним примерима је довољна раван), у *кластер* (*cluster*). За неки податак за који се не зна којој класи припада могуће је одредити његову највероватнију на основу сличности са постојећим подацима, углавном одређивањем најближег кластера. Ова класификација се користи у разним препознавањима сигнала нпр. звука, те је могуће издвојити речи и реченице говорника са неког аудио снимка.

[3]Користећи се појмовима линеарне алгебре, поставка овог проблема добија облик: одредити вектор $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ такав да за дате вектор $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$ и дату матрицу $\mathbf{A} \in \mathbb{M}_{m \times n}(\mathbb{R})$ важи да је $\mathbf{c} \cdot \mathbf{x}$ највеће могуће при ограничењима $\mathbf{Ax} \geqslant \mathbf{b}$ и $\mathbf{b} \geqslant 0$.

Овај рад се бави разним геометријским упитима решивих применом бинарних стабала, као и о присуству тих упита у практичним проблемима. Обрађена су следећа стабла: сегментно, Фенвиково и kd. На неким местима се помињу и њихова могућа уопштења за различите метрике и више димензије, али се не улази превише у детаље.

Иако је «званично» и «првенствено» тема овог матурског рада у оквиру информатичке групе предмета, он у себи садржи и појмове из геометрије, математичке анализе, линеарне алгебре, аналитичке геометрије и других области. Такође, ни сама тема не може да се сврста у оквире конкретног предмета, будући да она излази из оквира градива које се обрађује у средњим школама, па и на многим факултетима који имају обиман програм рада из рачунарства — реч је о области која је и даље актуелна: нове примене наведених структура података се и даље истражују.

За имплементацију ових структура података коришћен је програмски језик C++, а разлог овог избора јесте чињеница да он у исто време поседује објектно-оријентисана својства, брзину и добру подршку у смислу библиотека (STL — *Standard Template Library*). Нису све имплементације експлицитно дате у овом «папирном облику», већ само они који илуструју неку значајну идеју. Детаљни кôдови могу да се нађу на приложеном CD-у.

2 Основни појмови

2.1 Бинарна стабла

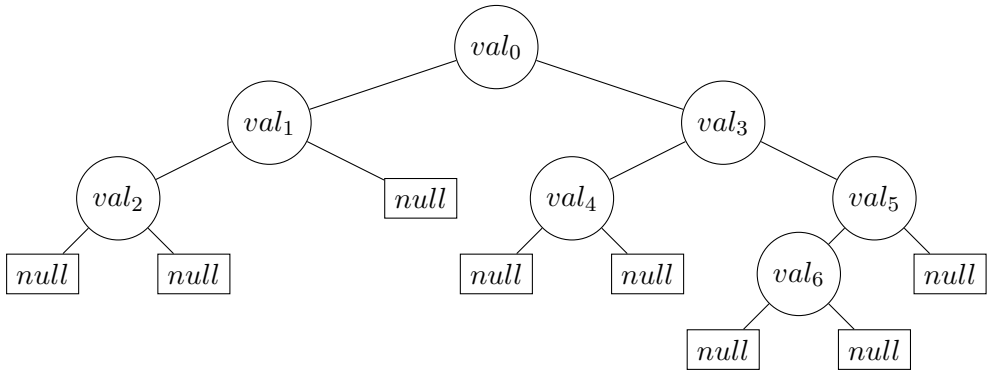
У овом одељку ће бити дата објашњења појмова у вези са кореним стаблима на начин који ће учинити даљу имплементацију врло интуитивном, а тиме ће бити обезбеђен и «шаблон» за имплементацију свих структура података које ће даље бити обрађене. Термин *rooted tree* се уводи ради разликовања ових стабала од истоименог појма у теорији графова, али овде таква терминологија није неопходна.

Поменуто је у уводу да се стабла користе када се уочава извесна хијерархија међу елементима: за сваки чвор u у оквиру стабла сем једног постоји тачно један други чвор v који је на хијерархијски вишем нивоу од њега. Тада кажемо да је u **дете** чвора v, а v **родитељ** чвора u.[4] Једини чвор који нема родитеља назива се **корен**, а чворови који немају ниједно дете називају се **листови**. Као што видимо, кључно својство стабла јесте да се за сваки његов чвор знају и његова деца, а понекад и његов родитељ.

Овиме можемо да прецизније дефинишемо основне појмове. ***k*-нарно стабло** је скуп чворова. **Чвор *a* у *k*-нарном стаблу** је уређени пар (val, N), где се val означава као вредност уписана у тај чвор, а N је низ од највише k показивача на чворове које називамо **децом** чвора a, док је a њима **родитељ**. Никоја два чвора у истом стаблу немају неко исто дете. Чвор u је **предак** чвора v уколико је u родитељ чвора v или постоји чвор w такав да је u родитељ w и w је предак v. Свако стабло има један чвор који је предак свим осталим чворовима и он се означава као **корен**. Све промењиве val у оквиру једног стабла су истог типа.

Неки појмови (нпр. подстабло) имају различито значење у зависности од контекста, тако да ће се тежити да се у сваком њиховом појављивању објасни на шта се тачно мисли.

Један такође значајан концепт је и **висина** (или **дубина**)[5] чвора у стаблу: корен је на висини 0, а сва деца неког чвора u су на висини за 1 већој од висине чвора u.



Слика 4: Имплементација бинарног стабла

Специјалан случај у горњој дефиницији се добија узимањем $k = 2$, чиме се добија **бинарно стабло**. Дакле, сваки чвор бинарног стабла има или ниједно

[4] Ово је контраинтуитивно ако се размишља у контексту породичног стабла, али се у сагласности са појмом наслеђивања у објектно-оријентисаном програмирању.

[5] Стабла се цртају «обрнуто» јер је корен «горе», па су сви његови потомци изнад корена, а листови су највиши делови стабла, што је разлог зашто се чешће користи реч висина.

или тачно једно или тачно два детета. Међутим, даља имплементација ће бити знатно олакшана уколико сматрамо да је број деце увек тачно два, при чему тада омогућимо постојање *null* («празних») чворова.

Оваква интерпретација је кључна за елегантну имплементацију бинарних стабала у конкретним програмским језицима. Уведимо класу `TreeNode`, при чему су њени атрибути променљива `val` неког типа `T` и два показивача који представљају референце на објекте који су такође класе `TreeNode` и зовемо их `leftChild` и `rightChild`. Пошто тип променљиве `val` није унапред одређен, користићемо темплејте. Дакле, основа свих даљих бинарних стабала која ће бити обрађена ће изгледати као у кôду 1. Тада се са чвором у стаблу чије су вредности целобројне најлакше оперише ако се он дефинише преко `TreeNode <int>* node`. STL стил имплементације структура података подразумева уводјење додатне класе `Tree` која у себи садржи само опште методе које представљају операције над целом структуром података.

Кôд 1

Овај кôд представља основу имплементације свих стабала.

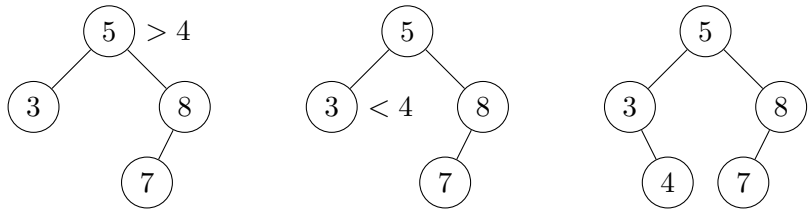
```
template <class T>
class TreeNode
{
    private:
        T val;              // vrednost upisana u cvor
        TreeNode* leftChild; // reference na decu
        TreeNode* rightChild;
    public:
//      ...
}

template <class T>
class Tree
{
    private:
        TreeNode <T>* root; // koren
    public:
//      ...
}
```

Напоменимо да је могуће имплементирати стабла без коришћења референци, али нам такав приступ неће бити нарочито користан.

Пример 1. Покажимо како ово ради на примеру једноставних операција у бинарном стаблу претраге (*binary search tree*, BST) — о њему је било речи у уводу — при чему ћемо претпоставити да никада неће бити наведени исти бројеви. У наивној варијанти је за сваки чвор u задовољено да је вредност која је у њега уписана већа (мања) од вредности уписане у његово лево (десно) дете, а уједно је задовољено да сви потомци левог (десног) детета чвора u имају вредност мању (већу) вредност уписану у чвор u.

При додавању новог елемента се најпре упореди та вредност са кореном, па се установи да ли он треба да буде његов леви потомак или десни потомак. Даље се тај поступак понавља на одговарајућем детету (подстаблу) све док се не дође до *null* чвора.

При испитивању да ли се нека вредност налази у стаблу се најпре испита



Слика 5: Додавање чвора у бинарно стабло претраге

да ли је то корен: ако јесте, поступак је готов; ако није, одреди се да ли би његово евентуално постојање значило да је реч о левом или десном потомку и поступак се понавља на одговарајућем детету (подстаблу) све док се не дође до *null* чвора.

Кôд 2

Имплементација бинарног стабла претраге.

```
template <class T>
class NodeBST
{
    private:
        T val;
        NodeBST* leftChild;
        NodeBST* rightChild;
    public:
    // konstruktori
        NodeBST(T val)
        {
            this -> val = val;
            this -> leftChild = NULL;
            this -> rightChild = NULL;
        }
    // operacije
        void Insert(T x)      // ubacivanje novog elementa
        {
            if (x < this -> val)
                if (this -> leftChild == NULL) // dodavanje cvora
                {
                    NodeBST <T>* newNode = new NodeBST <T>(x);
                    this -> leftChild = newNode;
                }
                else                          // levo podstablo
                    this -> leftChild -> Insert(x);
            if (x > this -> val)
                if (this -> rightChild == NULL) // dodavanje cvora
                {
                    NodeBST <T>* newNode = new NodeBST <T>(x);
                    this -> rightChild = newNode;
                }
                else                          // desno podstablo
                    this -> rightChild -> Insert(x);
        }
        bool Find(T x)        // pretraga, vraca da li se x
        {                     // nalazi u stablu
            if (this == NULL)         // ne postoji
```

```
            return false;
        if (x < this -> val)     // mora da bude u levom podstablu
            return this -> leftChild -> Find(x);
        if (x > this -> val)     // mora da bude u desnom podstablu
            return this -> rightChild -> Find(x);
        return true;             // nadjen
    }
};

template <class T>
class BST
{
    private:
        NodeBST <T>* root;
    public:
    // konstruktori
        BST() { root = NULL; }
    // operacije
        void Insert(T x)
        {
            if (root == NULL)
                root = new NodeBST <T>(x);
            else
                root -> Insert(x);
        }
        bool Find(T x)
        {
            if (root == NULL)
                return false;
            else
                root -> Find(x);
        }
};
```

Иако је «очекивана» сложеност свих операција $\Theta(\log n)$, то се постиже само онда када је стабло *балансирано* тј. сви листови су на истој висини у стаблу. Напоменимо да STL садржи библиотеку `set`, а помоћу ње се врло ефикасно решавају проблеми који траже коришћење неког далеко ефикаснијег (самобалансирајућег) бинарног стабла претраге. △

2.2 Основни појмови о афиним просторима

Овај одељак за циљ има прецизно дефинисање и описивање неких математичких појмова који имају значајне последице, а оне ће овде бити наведене и биће касније коришћене.

Основни задатак аналитичке геометрије јесте установљавање узајамног односа између геометријских појмова и алгебарских једначина. Савремен формалан приступ заснивања геометрије се зато често врши не преко аксиома примењеним на основне појмове тачке, праве и равни, већ управо њиховим дефинисањем преко познатих појмова линеарне алгебре. Предност оваквог начина је, између осталог, могућност уопштења неких концепата: вишедимензионалност, нова дефиниција растојања, хиперравни...

Да би аксиоме класичне геометрије постале теореме у оваквој теорији, најпре је неопходно дефинисати тачке, праве, равни. Зато је кључно претходно

увести појам који је кључна спона линеарне алгебре и аналитичке геометрије: **афини простор**.

Дефиниција 1. **Афини простор** $\mathcal{A}$ је уређена тројка $(\mathbb{S}, \mathbb{V}, f)$, где је $\mathbb{S}$ скуп чије елементе називамо **тачкама**, $\mathbb{V}$ векторски простор, а $f : \mathbb{S}^2 \to \mathbb{V}$ функција која сваком пару тачака $(A, B) \in \mathbb{S} \times \mathbb{S}$ додељује неки вектор $\mathbf{v} \in \mathbb{V}$ тако да су задовољени следећи услови:

1° за сваке три тачке $A, B, C \in \mathbb{S}$ важи $f(A, B) + f(B, C) = f(A, C)$;

2° за сваку тачку $A \in \mathbb{S}$ и вектор $\mathbf{v} \in \mathbb{V}$ постоји и јединствено је одређена тачка $B \in \mathbb{S}$ таква да је $f(A, B) = \mathbf{v}$ (често у ознаци $\overrightarrow{AB} = \mathbf{v}$).

Пример 2. Афини простор $\mathcal{A} = \left(\mathbb{S}, \mathbb{R}^3, f\right)$ који задовољава неопходне услове је основа Еуклидске геометрије. △

Надаље ће бити коришћени појмови **директрисе** (ако је $\mathcal{A} = (\mathbb{S}, \mathbb{V}, f)$ афини простор, тада је векторски простор $\mathbb{V}$ његова директриса) **димензије** и **димензије** афиног простора (што има значење димензије његове директрисе).

Најпре се примети да је у афином простору $\mathcal{A} = (\mathbb{S}, \mathbb{V}, f)$ могуће представити сваку тачку $A \in \mathbb{S}$ као ону за коју је $f(O, A) = \mathbf{v}$ за неку тачку $O \in \mathbb{S}$ и неки вектор $\mathbf{v} \in \mathbb{V}$ на јединствен начин. У том случају $\mathbf{v}$ називом **радијус-вектором** тачке A у односу на **координатни почетак** O. Зато се често при алгебарском представљању тачака неког простора користе одговарајући радијус-вектори, будући да је он карактеристичан само за ту тачку (тачније, овако описано пресликавање је бијективно).

У простору $\mathcal{A}$ из примера 2 могуће је говорити о тачкама као уређеним тројкама реалних бројева, будући да су одговарајући радијус-вектори елементи $\mathbb{R}^3$. Уопште, ако је $\mathbb{R}^n$ векторски простор који одговара неком афином простору $\mathcal{A}$, тада се тачке представљају у облику уређених n-торки реалних бројева. Тада се може рећи да су **координате** неке тачке, у ствари, координате њеног радијус-вектора у односу на базу векторског простора. Зато ће се управо такво представљање тачака користити при каснијем опису и имплементацији структура података.

Дефиниција 2. Афини простор $\mathcal{A}_1 = (\mathbb{S}_1, \mathbb{V}_1, f_1)$ је **афини потпростор** афиног простора $\mathcal{A}_2 = (\mathbb{S}_2, \mathbb{V}_2, f_2)$ (у ознаци $\mathcal{A}_1 \leqslant \mathcal{A}_2$) уколико су задовољени услови:

1° $\mathbb{S}_1 \subseteq \mathbb{S}_2$;

2° $\mathbb{V}_1 \leqslant \mathbb{V}_2$;

3° $f_1 = f_2 \mid_{\mathbb{S}_1}$ (f_1 је рестрикција f_2 на скупу $\mathbb{S}_1$).

Ако су $\mathcal{A}_1$ и $\mathcal{A}_2$ афини простори за које је $\dim \mathcal{A}_1 = n-1$, $\dim \mathcal{A}_2 = n$ и $\mathcal{A}_1 \leqslant \mathcal{A}_2$, тада је $\mathcal{A}_1$ **хиперраван** у $\mathcal{A}_2$.

Пример 3. Хиперравни Еуклидске праве, равни и простора су тачке, праве и равни, респективно. △

2.3 Основни појмови о метричким просторима

Сада ће се размотрити један други елементаран појам. Од нарочитог значаја и за математичку анализу и за геометрију јесте појам **растојања**: на њему се

заснива читава теорија граничних вредности функција једне променљиве; постоји група аксиома Еуклидске геометрије која описује релацију подударности парова тачака. Међутим, управо због чињенице да се поступак одређивања растојања мења у зависности од тога да ли је реч о пару тачака на правој, у равни или у тродимензионалном простору, поставља се питање о могућим уопштењима. У складу са тим је дата дефиниција **метричког простора** (по Фрешеу[6]):

Дефиниција 3. **Метрички простор** $\mathbb{M}$ је уређен пар $(\mathbb{S}, d)$ такав да функција $d : \mathbb{S}^2 \to \mathbb{R}$, која се назива **метрика** или **растојање**, задовољава следеће услове $(\forall A, B, C \in \mathbb{S})$:

1° $d(A, A) = 0$;

2° $A \neq B \Longrightarrow d(A, B) > 0$;

3° $d(A, B) = d(B, A)$;

4° $d(A, B) + d(B, C) \geqslant d(A, C)$ (*неједнакост троугла*).

Будући да ће у овом раду бити посматрано растојање између тачака, то ће овде бити размотрени само они случајеви када је у метричком простору $\mathbb{M} = (\mathbb{S}, d)$ скуп $\mathbb{S}$ облика $\mathbb{R}^k$, где $k \in \mathbb{N}$.

Пример 4. Размотримо неколико неколико метрика које ће касније бити од значаја. Неће се тежити комплетном доказу свих својстава, већ ће евентуално бити наведено објашење зашто важи својство 4 (будући да се 1, 2 и 3 често једноставно доказују).

1° $(\mathbb{R}, d)$, где $d : (x, y) \mapsto |x - y|$, јесте један метрички простор. Он је у основи дефиниције граничне вредности функције једне променљиве, конвергенције низова итд.

2° Сви пред-Хилбертови[7] простори[8] $\mathbb{V}$ могу да буду и метрички: растојање између два вектора одговара норми њихове разлике тј. $d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\| = \sqrt{(\mathbf{u} - \mathbf{v}) \cdot (\mathbf{u} - \mathbf{v})}$. При том четврто својство важи због неједнакости Минковског у варијанти која важи у свим пред-Хилбертовим просторима.

Специјално, од значаја су они којима би могао да се придружи неки афини простор, чиме је могуће говорити о растојању између две тачке. Узимајући $\mathbb{V} = \mathbb{R}^n$ и стандардни скаларни производ добије се **Еуклидова метрика**, а одговарајући **Еуклидов метрички простор** ће бити означен са $\mathbb{E}^n$. Специјално, $\mathbb{E}^1$ је управо описан у 1 (Еуклидска права), а растојање између тачака $A = (x_1, y_1, z_1)$ и $B = (x_2, y_2, z_2)$ у метричком простору $\mathbb{E}^3$ дато је познатом формулом:

$$d(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}.$$

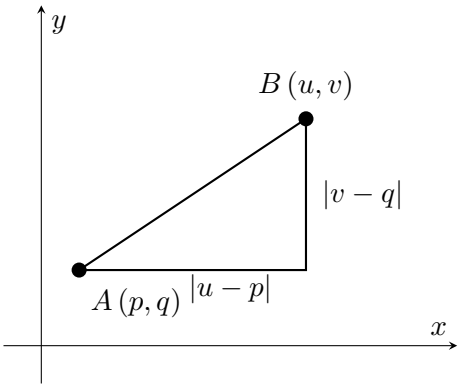
[6] Maurice René Frechet (1878–1973), француски математичар

[7] David Hilbert (1862–1943), немачки математичар

[8] **Пред-Хилбертов простор** $\mathbb{V}$ је векторски простор ком је придружена бинарна операција скаларног производа $\cdot : \mathbb{V}^2 \to \mathbb{R}$ у оквиру уобичајене дефиниције. Сви векторски простори облика $\mathbb{R}^n$ са стандардним скаларним простором су пред-Хилбертови. Могуће је, додавањем још неких услова, дефинисати и **Хилбертове просторе**, о чему је могуће прочитати у АНАЛИЗА 2, стр. 278.

3° **Такси-геометријски метрички простор** (или метрички простор Минковског) биће означен са $\mathbb{T}^n = (\mathbb{R}^n, d)$. Његова метрика d (некада се користи и назив «Менхетн растојање») се дефинише на следећи начин: за $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ је:

$$d(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{n} |a_i - b_i| = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|.$$



Слика 6: Такси-геометријско растојање

Посматрајмо значење овакве метрике. Нека су дате две тачке у равни, $A(p, q)$ и $B(u, v)$. Тада је њихово растојање дато са $d(A, B) = |u - p| + |v - q|$ — другим речима, то је збир дужина уоченог правоугаоника чије су странице паралелне координатним осама (видети слику). У ствари, ако би се кретање неке тачке у равни ограничило на кретање само дуж хоризонтале и вертикале, тада је овако дефинисано растојање управо најмање. Ако би се посматрале тачке са целобројним координатама, примети се смисао назива «Менхетн растојање».

4° У метричком простору $\mathbb{E}^n$ се растојање између вектора $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ и $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ одређује са

$$d(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{n} (a_i - b_i)^2 \right)^{\frac{1}{2}}.$$

Изменом експонената, односно уопштењем

$$d(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{\frac{1}{p}},$$

добије се $\boldsymbol{L_p}$ **метрика**. Одговарајући метрички простор се означава са $\mathbb{L}_p^n$. Доказ неједнакости троугла следи из неједнакости Минковског.

Специјално, сви $\mathbb{L}_2$ метрички простори су Еуклидови, а $\mathbb{L}_1$ су такси-геометријски.

5° У специјалном случају $\mathbb{L}_\infty^n$ се добије да је растојање између веткора $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ и $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ једнако:

$$d(\mathbf{a}, \mathbf{b}) = \max_{1 \leqslant i \leqslant n} |b_i - a_i|. \ \triangle$$

Наредни појам је значајан како у информатици, тако и у математичкој анализи, будући да је у основи дефиниција непрекидности и диференцијабилности реалних функција више промењљивих.

Дефиниција 4. Нека је $\mathbb{M} = (\mathbb{S}, d)$ један метрички простор и $A \in \mathbb{S}$. Тада је **отворена (затворена) кугла радијуса r** са **центром A** скуп тачака $X \in \mathbb{S}$ који задовољава услов:

$$d(A, X) < r \text{ (односно } d(A, X) \leqslant r).$$

При коришћењу речи кугла углавном ће се мислити на отворену куглу.

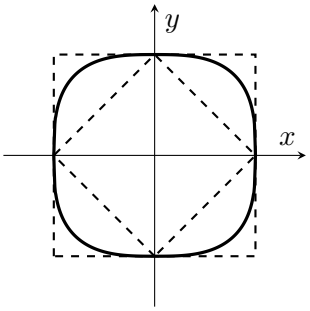
Пример 5. Размотримо кугле у различитим метричким просторима:

1° У Еуклидским метричким просторима $\mathbb{E}^1$, $\mathbb{E}^2$ и $\mathbb{E}^3$ кугле су интервали, унутрашњости кругова и унутрашњости лопти.

2° У простору $\mathbb{L}_\infty^2$ кугла је скуп тачака којој су све координате у опсегу r од одговарајућих координата центра те кугле — реч је о квадрату чије су странице паралелне координатним осама. Даље, у $\mathbb{L}_\infty^3$ кугла је коцка чије су стране паралелне равнима xOy, yOz и zOx. У општем случају, у $\mathbb{L}_\infty^n$ кугла је n-димензионална **хиперкоцка** чије су ивице паралелне одговарајућим векторима ортонормиране базе. У ову n-димензионалну хиперкоцку је уписана одговарајућа кугла у $\mathbb{E}^n$.

3° У такси-геометријским векторским просторима кугле су такође хиперкоцке, али су оне сада уписане у одговарајуће Еуклидске кугле.

4° Посматрајмо $\mathbb{L}_p^2$ метричке просторе и утврдимо њихове кугле за $p > 1$. Већ их смо одредили за $p = 1$, $p = 2$ и $p \to \infty$, одакле закључујемо да са обе стране конвергирају у хиперкоцке. △



Слика 7: Кугла у $\mathbb{L}_3^2$ метричком простору

У већини стабала која ће бити даље наведена биће речи о Еуклидским афиним просторима и углавном о Еуклидским метрикама, али ће понекад бити значајно «пребацити се» на неке друге метрике у зависности од захтева (и то управо због неких својствава кугли).

2.4 Клетва димензионалности

Геометријски проблеми који ће овде бити описани ће, поред компликованог које користи стабла, имати и наивно решење. Углавном ће ти проблеми бити формулисани на нивоу неке праве, равни, евентуално тродимензионалног простора. Међутим, и у теорији и у пракси (постоје проблеми чија је формулација

еквивалентна некој геометријској, при чему то често није дато на експлицитан начин) се посматрају и вишедимензионални простори, те се захтевају уопштења структура података које решавају задатак. Међутим, у неким случајевима ти претходно ефикасни алгоритми губе своју предност у брзини у односу на наивно решење, а разлог томе је чињеница да је временска сложеност операција над структуром података експоненцијална по димензији простора. Ова појава се назива **клетва димензионалности** (енг. *curse of dimensionality*).[9]

У описима свих структура података које могу да се уопште тако да одговарају на неке кључне упите ће бити истражен утицај клетве димензионалности.

[9]Наравно, ово није само у вези са геометријом, јавља се где год је димензија векторског простора над којим се оперише у експоненту функције која означава сложеност алгоритма.

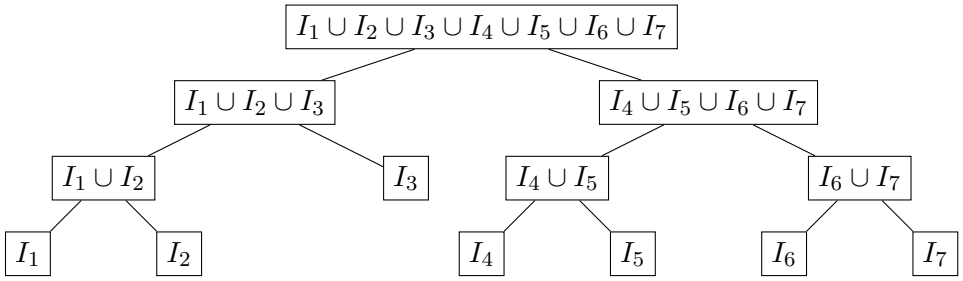
3 Сегментно стабло

Сегментно стабло први уводи Бентли[10] у свом раду из 1977. године [линк ка раду] да би решио један геометријски проблем (о ком ће касније бити речи), али је ова структура података касније добила многе друге интерпретације.

3.1 Општа структура сегментног стабла

Сегментна стабла се јављају у многим варијантама и имплементацијама, али је у основи свега да сваки чвор тог стабла описује неке интервале или сегменте, а да његова деца представљају подинтервале или подсегменте. Зато ћемо, пре разматрања конкретних проблема, описати општу структуру и операције у свим сегметним стаблима.

Нека је дато n тачака на реалној правој: x_1, x_2, ..., x_n, при чему важи $x_1 < x_2 < \ldots < x_n$. Тада издвајамо наредне елементарне интервале и сегменте: $I_1 = (-\infty, x_1)$, $I_2 = [x_1, x_1]$, $I_3 = (x_1, x_2)$, $I_4 = [x_2, x_2]$, ..., $I_{2n-1} = (x_{n-1}, x_n)$, $I_{2n} = [x_n, x_n]$, $I_{2n+1} = (x_n, +\infty)$. У зависности од конкретног задатка, ови елементарни скупови могу да имају различита значења и својства, али се поставља услов да сваки од тих интервала има једну константну особину (вредност неког броја придруженог том скупу или припадност неком надскупу, као у упиту припадности (*stabbing query*, о чему ће бити речи касније)). Уколико се од нас захтева да се испита нека особина уније узастопних $I_a \cup I_{a+1} \cup \ldots \cup I_b$, неретко се користи сегментно стабло.



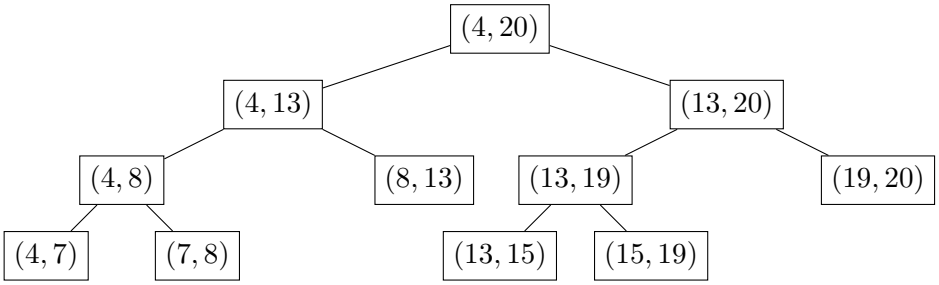
Слика 8: Структура сегментног стабла

Сегментно стабло је бинарно стабло чији сваки чвор говори о некој унији узастопних елементарних интервала и сегмената, те има вредност сачињену из три дела: први одређује особине леве границе (првенствено где се налази), други особине десне границе, а трећи говори о некој карактеристици те укупне уније. Деца неког чвора који представља унију n елементарних скупова представљају $\left\lfloor \frac{n}{2} \right\rfloor$ и $\left\lceil \frac{n}{2} \right\rceil$ елементарних скупова. Корен сегментног стабла описује целу реалну праву, а листови елементарне скупове. Овиме је обезбеђено да стабло буде балансирано и да су забележени подаци о сваком делу реалне праве.

Наравно, ова структура се мало упрости у зависности од конкретног задатка. Тако, на пример, ретко постоји потреба да се за елементарне узму и сегменти облика $[x_i, x_i]$ и интервали (x_i, x_{i+1}). У ситуацијама када је битно само понашање у крајевима, интервали могу да се «прогутају». Некада неће бити потребе за интервалима који има бесконачне границе. У случајевима

[10] Jon Louis Bentley (1953–), амерички теоријски информатичар, професор на Carnegie Mellon универзитету у Питсбургу, аутор књиге Programming Pearls

када никада неће бити могућ приступ самој граници или када се бавимо неком мерљивошћу скупова (рачунамо дужине, површине, запремине), није неопходно оставити и једноелементне сегменте у стаблу. Понекад ће бити довољно само посматрати скупове облика $[x_i, x_{i+1})$.



Слика 9: Пример једног сегментног стабла, једноелементни скупови су избачени

Могуће је осмислити имплементацију сегментног стабла коришћењем низа следећим «триком»: уведе се неколико «лажних» листова тако да их укупно буде 2^k, чиме стабло постаје балансирано и тада је могућа нешто олакшана имплементација (на исти начин као код binary heap).

Сврха сегментног стабла је да велики број упита раније наведеног типа који се једноставним приступима одраде у $\Theta(n)$ (или горем) времену реше у $\Theta(\log n)$ временској сложености. Начин на који се ово постиже јесте тако што се интервал/сегмент у упиту напише као унија што је могуће мање дисјунктних интервала/сегмената представљених сегментним стаблом. Уз њих, ту је и основни упита који наговештава употребу сегментног стабла: измена карактеристике неког елементарног скупа (изводљиво у $\Theta(\log n)$). У уобичајним околностима није могуће додавање нових елемената у сегментно стабло и није могуће брисање (додуше, некада су могући «трикови» који премосте ово ограничење). Остаје још једна операције, а она је уједно и једина која је временски захтевнија: сама изградња сегментног стабла ($\Theta(n)$).

Наведимо сада уобичајене поступке којим се обављају све ове радње, при чему ћемо (ради једноставности) претпоставити да су нам довољни интервали облика (x_1, x_2), (x_2, x_3), $\ldots$, (x_{n-1}, x_n), што ће учинити атрибуте и методе класе доста једноставним.

Кôд 3

Почетак имплементације сегментног стабла.

```
template <class T>
class SegmentTreeNode
{
    private:
        T val;                      // karakteristika opisanog intervala
        double leftBound;           // granice intervala
        double rightBound;
        SegmentTreeNode* leftChild; // reference na decu
        SegmentTreeNode* rightChild;
    public:
//      ...
}
```

```
template <class T>
class SegmentTree
{
    private:
        SegmentTreeNode* root;
    public:
//      ...
}
```

Изградња. Нека нам је дат низ X који представља n сортираних тачака на реалној правој. Тада се од таквог низа прави сегментно стабло полазећи од корена: његов интервал је (x_1, x_n), а интервали који одговарају његовој деци су $(x_1, x_{\lfloor n/2 \rfloor})$ и $(x_{\lfloor n/2 \rfloor}, x_n)$. Даље се формирају одговарајућа подстабла све док се не дође до елементарних интервала. Овај последњи корак подразумева подешавања полазних вредности за карактеристике елементарних интервала, те настаје преплитање са наредном операцијом. Формирање једног чвора се врши у константном времену, а број чворова сегментног стабла је $n + \lceil n/2 \rceil + \lceil \lceil n/2 \rceil /2 \rceil + \ldots + 1 \approx 2n$, те је сложеност ове операције $\Theta(n)$.

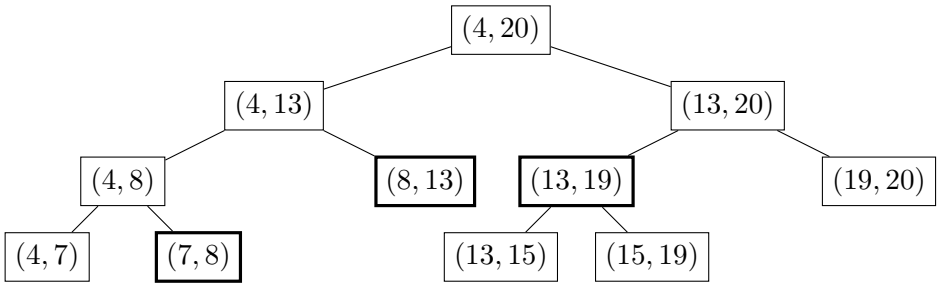
У неким ситуацијама је корисно или једноставније знати на које се елементарне интервале односи сваки чвор (уместо да се памте границе). Обе имплементације су приложене на диску.

Измена. Карактеристика сваког чвора на неки начин зависи од његове деце — ако се ова веза не уочи, сегментно стабло вероватно не решава задатак. Када се најпре измени неки елементарни интервал, то значи да се мења лист, што даље мора да утиче на његовог родитеља, па на родитеља његовог родитеља итд. односно на све његове претке. Зато је имплементација ове операције рекурзивна и неопходно је поћи од корена, па «тражити» одговарајући елементарни скуп. Сложеност ове операције је $\Theta(\log n)$ јер је $\log n$ висина стабла.

Упит. Неопходно је задати скуп за који се одређује одговор представити као унију што је могуће мање дисјунктних скупова садржаних у сегментном стаблу, након чега се испита како они заједно дају коначан одговор. Уколико сигурно може да се напише као унија неких елементарних скупова, тада само разбијање није нарочито сложено јер можемо да разликујемо три случаја: ако тренутно посматрани чвор представља неки подскуп коначног скупа, онда он учествује у разбијању; ако тренутно посматрани чвор представља неки скуп који је дисјунктан са коначин скупом, ни он ни било који његов потомак неће учествовати у разбијању; у свим преосталим случајевима неће тај чвор учествовати у разбијању, али у обзир долазе оба његова детета. Ово је општа шема која подлеже изменама у зависности од самог проблема. Сложеност ове операције је $\Theta(\log n)$.

Ово све важи уколико дати скуп може да се представи као унија елементарних скупова. Уколико не може, поступа се у складу са постављеним захтевом.

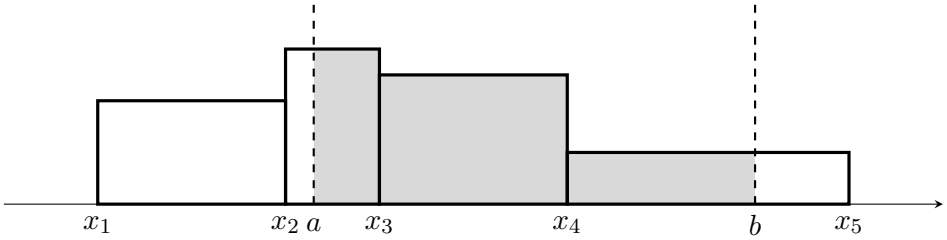
Пример 6. Дат је скуп $n-1$ дисјунктних правоугаоника, при чему сви належу на x-осу у xOy равни у једној узастопној серији: први правоугаоник за једну страницу има $[x_1, x_2]$, други $[x_2, x_3]$, ..., последњи $[x_{n-1}, x_n]$. Операције које могу да се врше над тим правоугаоницима су облика:



Слика 10: Скупови који учествују у изградњи $(7, 19)$

1° за прослеђене a и b одредити површину пресеку фигуре добијене у унији свих правоугаоника и дела равни $a \leqslant x \leqslant b$;

2° промена висине k-тог правоугаоника на h (тако да правоугаоник и даље належе).



Слика 11: Површина фигуре за $a \leqslant x \leqslant b$

Наивни приступ одговарања на ове операције се врши у временским сложеностима $\Theta(n)$ и $\Theta(1)$, што није повољно уколико се број операција прве врсте доста велик. Међутим, ако посматрамо (x_k, x_{k+1}) као елементарне интервале и на основу њих конструишемо сегментно стабло, добијемо знатно ефикасније решење.

Нека се у сваком чвору који представља интервал (l, r) памти и резултат првог упита када $a = l$ и $b = r$ тј. збир површина узастопне серије правоугаоника. Тада је та карактеристика сваког чвора једнака збиру карактеристика његове деце, док се за листове она рачуна као $h \cdot (r - l)$. Овиме је омогућена изградња сегментног стабла и измена листова.

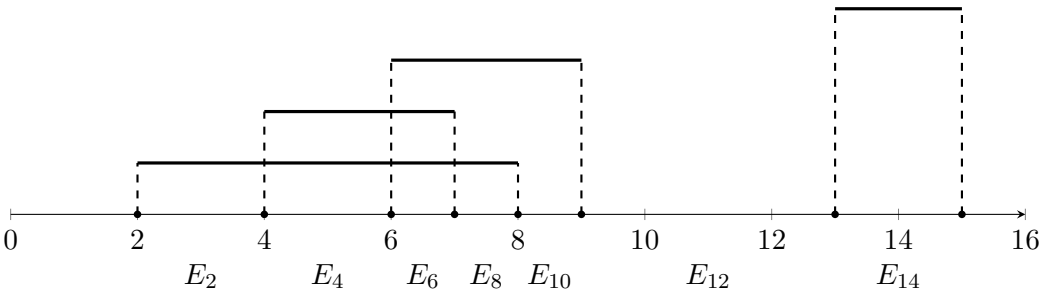
Остаје да се реши операција 1. Приметимо да $a \leqslant x \leqslant b$ обухвата неколико целих правоугаоника из овог скупа и највише два «исечена». Решење представља збир површина свих целих правоугаоника и делове површина крајња два правоугаоника. Међутим, површине целих правоугаоника су већ организоване у збирове у чворовима који нису листови, те се примењује горе описани принцип одабира што је могуће мање чворова да би се описала та унија правоугаоника. Посебно се размотре и «исечени» правоугаоници, који представљају листове које треба анализирати.

Дакле, решење које примењује сементно стабло одговара на обе операције у сложености $\Theta(\log n)$, али захтева препроцесирање сложености $\Theta(n)$. Дакле, ово решење је увек боље од наивног. $\triangle$

3.2 *Stabbing query*

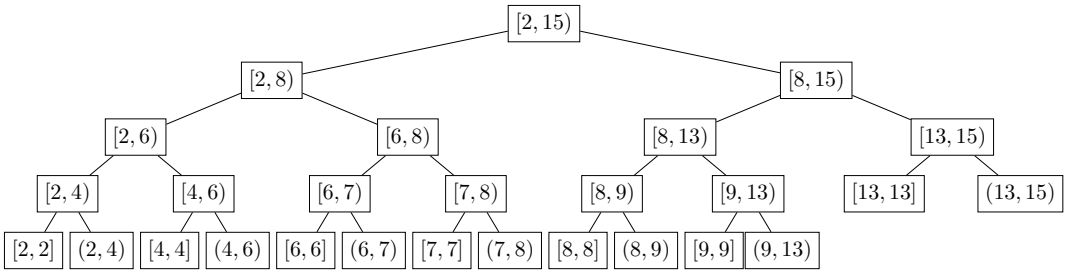
Размотримо сада ситуацију када постоји извесно одступање од поменутог «шаблона» упита над сегментним стаблом: операција измене ће захтевати одабир што је могуће мање чворова који описују неки интервал, а упит ће захтевати приступ неком конкретном елементарном интервалу. Ово се среће у ситуацијама када је један од упита налик ***stabbing query***. Дакле, један од основних задатака је следећи: дат је низ интервала на реалној правој, за сваку даље задату тачку на тој правој одредити у колико се од тих интервала налази.

Основна идеја је да се сви скупови $I_1, \ldots, I_n$ који су задати на самом почетку поделе на елементарне, односно све интервале и једноелементне сегменте E_j $(1 \leqslant j \leqslant m)$ који одговарају пресецима полазних интервала I_k. Примера ради, ако је $I_1 = [2, 8)$, $I_2 = [4, 7]$, $I_3 = (6, 9)$, $I_4 = (13, 15)$. тада можемо да уочимо $E_1 = [2, 2]$, $E_2 = (2, 4)$, $E_3 = [4, 4]$, $E_4 = (4, 6)$, $E_5 = [6, 6]$, $E_6 = (6, 7)$, $E_7 = [7, 7]$, $E_8 = (7, 8)$, $E_9 = [8, 8]$, $E_{10} = (8, 9)$, $E_{11} = [9, 9]$, $E_{12} = (9, 13)$, $E_{13} = [13, 13]$, $E_{14} = (13, 15)$. Добијање скупова E_j од I_k се постиже тако што се најпре издвоје крајеви интервала I_k, након чега се те тачке сортирају у растућем поретку, па се на крају тим редом и процесирају, што једноставно генерише скупове E_j. Наравно, неопходно је обратити пажњу и на могућа поклапања крајева.



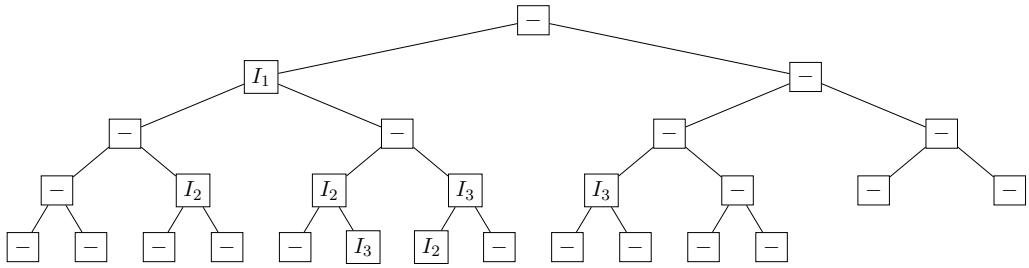
Слика 12: Добијање елементарних скупова

Над овако добијеним елементарним скуповима $E_1, \ldots, E_n$ могуће је конструисати сегментно стабло. Потом се за сваки од I_k примени операција чија структура личи на упит описан на почетку секције: представи се преко што је могуће мање чворова сегментног стабла. При томе се за сваки чвор стабла чувају подаци (на пример, у облику повезане листе) о свим интервалима I_k у чијем представљању учествују.

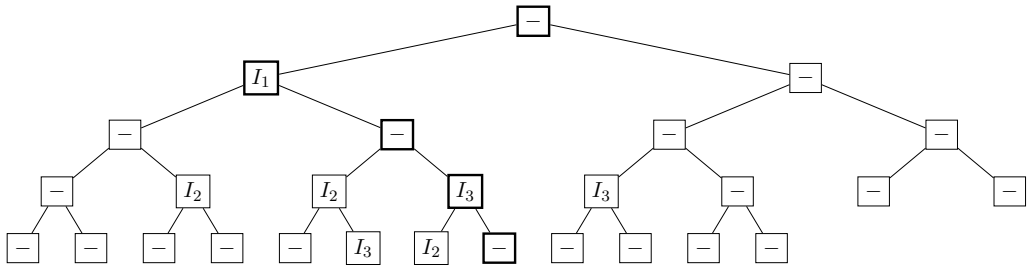


Слика 13: Сегментно стабло над добијеним елементарним скуповима

Када се захтева испис свих интервала I_k који садрже неку тачку A, поступак који се примењује јесте тај да се пође од чвора u који представља елементарни скуп ком A припада (ако постоји, будући да је такође могуће да је A «лево



Слика 14: Разбијања полазних скупова



Слика 15: Тачка 7.5 се налази у скуповима I_1 и I_3

од најлевљег» или «десно од најдеснијег»), након чега се анализирају сви преци u. Сви интервали I_j који су наведени у u и прецима u (у оквиру раније уведених повезаних листа) уједно и представљају све интервале I_k у којима се налази A. Тиме је дат одговор на *stabbing query*.

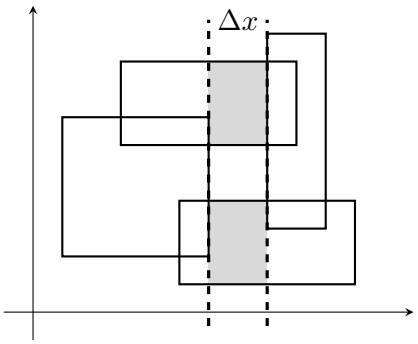
Ово решење се састоји из два дела: препроцесирања и одговора на упит. Препроцесирање се састоји из одређивања елементарних скупова (доминира временска сложеност сортирања, $\Theta(n \log n)$) и конструкције сегментног стабла ($\Theta(n)$), а временска сложеност препроцесирања је $\Theta(n \log n)$. Сваки упит добија одговор у $\Theta(\log n)$ времену, те је овакво решење добро у случајевима када је број упита већи од броја тачака.

Постоји и једно уопштење овог проблема: могуће је додавање нових интервала. Идеју иза решавања оваквог задатка ћемо размотрити у следећем историјски значајном примеру.

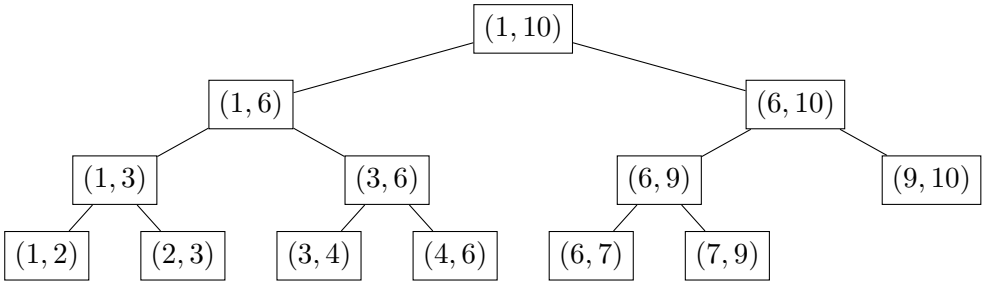
Пример 7. 1977. године Бентли објављује рад (*Algorithms for Klee's rectangle problems*) у ком дефинише сегментна стабла да би решио наредни проблем: за дати скуп од n правоугаоника у равни којима су странице паралелне координатним осама (правоугаоници су задати паровима наспрамних темена (x_k, y_k) и (x'_k, y'_k), где $x_k < x'_k$, $y_k < y'_k$) одредити површину њихове уније.

Овај задатак може да се реши уколико се посматра као «динамичка» варијанта принципа коришћеног у одговарању на *stabbing query*. У претходном примеру је задатак већ у сâмој својој поставци био *on-line*: неопходно је одговорити на неке упите док се мења неко стање. Међутим, у неким проблемима је корисно имати на уму читав улаз и све будуће упите пре формирања самог сегментног стабла. У овом случају ми сâми дефинишемо наше операције, и један природно *off-line* проблем добија другачији, *on-line* карактер.

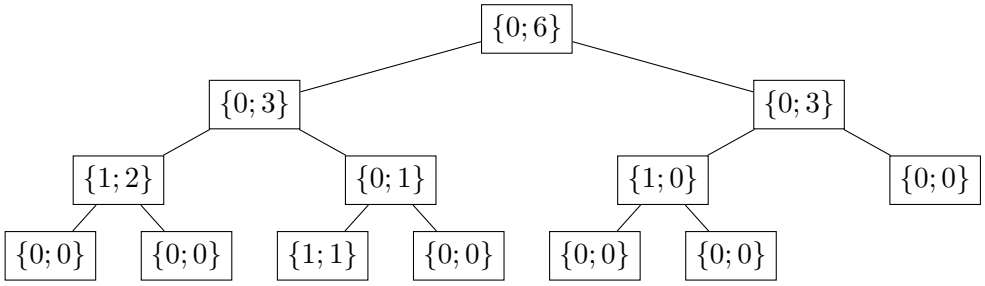
Посматрајмо све интервале (y_k, y'_k) (сваком правоугаонику придружујемо «леви», «улазни», и «десни», «излазни», интервал) и разбијмо их на дисјунктне елементарне интервале. Креирамо сегментно стабло над њима тако да буде спремно за *stabbing query*, с тим што није неопходно меморисати листе интервала у сваком чвору, већ је довољан само број њим. Сваки чвор тог ста-



Слика 16: Илустрација *sweep line* принципа и резултата $\Delta x \cdot root.val$



Слика 17: Сегментно стабло изграђено над скупом правоугаоника



Слика 18: Вредности у чворовима сегментног стабла пред ажурирање одсечка на позицији $x = 8$, а након ажурирања $x = 5$. Вредности су записане у облику $\{$број интервала$; val\}$. Пошто је $root.val = 6$, а $\Delta x = 8 - 5 = 3$, то је «пребрисана» површина једнака $3 \cdot 6 = 18$.

бла представља неки интервал са y-осе који је унија елементарних, те можемо сваком чвору да придружимо вредност *val* која ће представљати актуелну «покривену» дужину (више о томе касније).

Сада се сортирају сви вертикални одсечци по својој x-координати у растућем поретку (у случају поклапања истих је свеједно како ћемо их распоредити), а онда се итерира по њима тим редом (тзв. *sweep line* поступак). Када се наиђе на неки вертикални одсечак, најпре се испита колика је површина «пребрисана» у току померања у односу на претходни одсечак: она је једнака $\Delta x \cdot root.val$, где је Δx растојање актуелног од претходно обрађеног одсечка. Ова вредност се додаје актуелној суми која представља површину.

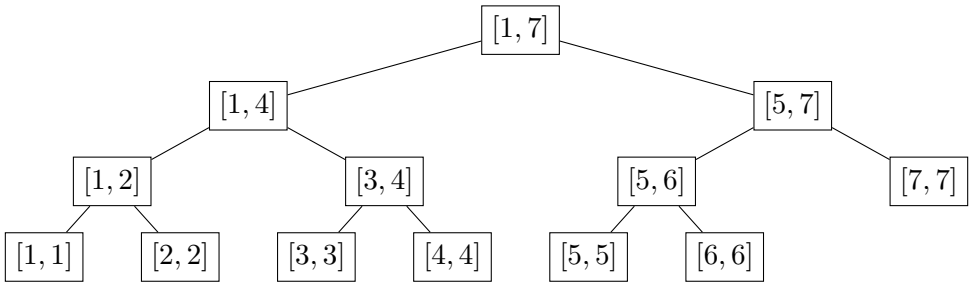
Након тога се поступа различито у случајевима када је реч о «улазном» или «излазном» интервалу: ако је «улазни», тада се «додаје» одговарајући интервал у сегментно стабло (у ствари, само се увећају одговарајуће вредности у чворовима сегментног стабла за 1) и ажурирају се све *val* вредности које су потенцијално измењене; ако је «излазни», тада се тај интервал «избацује» и

опет се ажурирају све потенцијално измењене *val* вредности. Ажурирање *val* вредности неког чвора се врши на основу чињенице да је она једнака или збиру одговарајућих вредности за своју децу (у случају да не улази у састав ниједног интервала) или дужини интервала који тај чвор представља (ако улази). Због природе употребе *val* вредности се испоставља да је ово довољно за исправну (практично једино коришћену) вредност *root.val*.

Сложеност изградње сегментног стабла је $\Theta(n)$ а сортирања $\Theta(n \log n)$. При анализирању сваког од $2n$ сегмената се у $\Theta(1)$ времену приступа вредности *root.val*, а у $\Theta(\log n)$ се ажурира стабло. Дакле, резултујућа сложеност је баш $\Theta(n \log n)$. △

3.3 Сегментно стабло индуковано низом

Размотримо сада једну дискретну варијанту сегментног стабла (до сада је било речи искључиво о непрекидним). Уместо да од улазних скупова формирамо неке елементарне, у овом случају су нам листовима сегментног стабла представљени једноелементни скупови, а њихови елементи су узастопни цели бројеви. Дакле, уопште нам нису од значаја вредности «између» целих бројева. И над овако конструисаним стаблом је могуће вршити операције сличног типа као и раније: ажурирање вредности и постављања питања о својствима неког опсега.



Слика 19: Пример сегментног стабла над дискретним скупом вредности

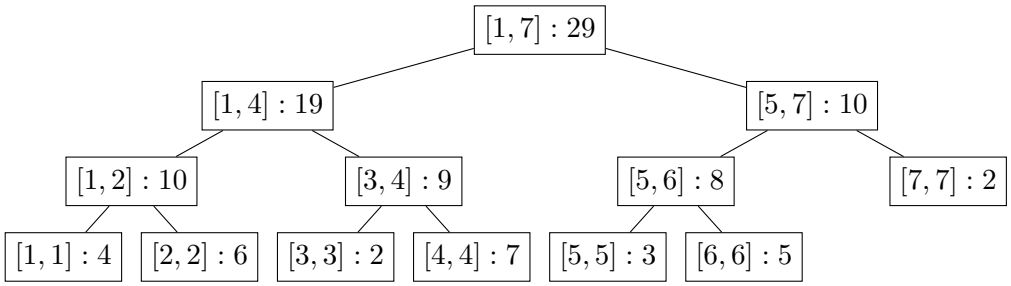
Овакво стабло је корисно посматрати у ситуацијама када је на улазу дат неки низ, вредности придружене листовима ће углавном бити једнаке вредности елемента низа на одговарајућем положају, а постоји довољно једноставна зависност вредности придружене родитељу и вредности придружене детету.

У ствари, никаква суштинска разлика не постоји између дискретних и непрекидних скупова при имплементацији операција, једина разлика јесте чињеница да није јасна геометријска интерпретација проблема у дискретним случајевима. Примене сегментних стабала у овом контексту су изузетно широке и излазе ван домена геометрије. Ипак, због неких значајних последица им посвећујемо посебну пажњу.

Пример 8. Дат је низ A природних бројева, при чему је дужина тог низа једнака n. Над овим низом се врши k операција типа:

1° вредност елемента низа A чији је индекс једнак i се мења на v;

2° захтева се одговор на питање «колики је збир свих елемената низа A чији су индекси у опсегу од i до j?».



Слика 20: Вредности уписане у сегментно стабло над низом $(4, 6, 2, 7, 3, 5, 2)$ у примеру 8

Најпре се изгради сегментно стабло над полазним низом, при чему вредност уписана у неки чвор представља збир свих елемената низа који се налазе у опсегу који се односи на тај чвор (слика 20). Прва од ове две операције је тада еквивалентна ажурирању сегментног стабла, а друга тражењу одговара на упит (неопходно је разбити $[i, j]$ тако да се представи као унија што је могуће мање скупова представљених чворовима у стаблу).

Временска сложеност изградње сегментног стабла је $\Theta(n)$, а сваке операције $\Theta(\log n)$. Дакле, укупна сложеност је $\Theta(n + k \log n)$. Ово је очигледно ефикасније о наивног решења које ради (у најгорем случају) у $\Theta(kn)$ времену.

Овај пример ће поново бити наведен при разматрању Фенвиковог стабла, где ће бити представљено једно мало ефикасније решење. △

3.3.1 RMQ проблем

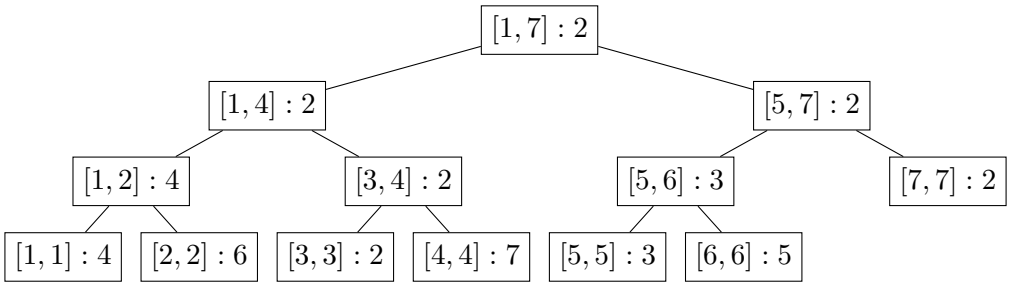
Сегментна стабла индукована низовима су у значајној вези са такозваним ***range minimum query*** (скраћено RMQ) проблемом (слободно преведено као «најмања вредност у опсегу»). Назив потиче од упита који је основна операција од значаја у овом задатку: ако је задат низ A чија је дужина n, која је најмања вредност која се налази између i-те и j-те позиције у A (при услову $1 \leqslant i \leqslant j \leqslant n$)?

RMQ проблем се јавља у две варијанте. Када су подаци над којима се поставља овај упит непроменљиви, постоје врло ефикасна решења (једно од њих користи структуру која се назива *sparse tables*, а друго Картезијска стабла). Са друге стране, у игру улазе сегментна стабла када су ти подаци променљиви.

Као у примеру 8, над низом је могућа конструкција сегментног стабла, при чему вредност придружена неком чвору одговара најмањој вредности у опсегу који тај чвор обухвата. Зато за сваки чвор који није лист важи да је та вредност једнака мањој од вредности уписаној у његову децу (слика 21).

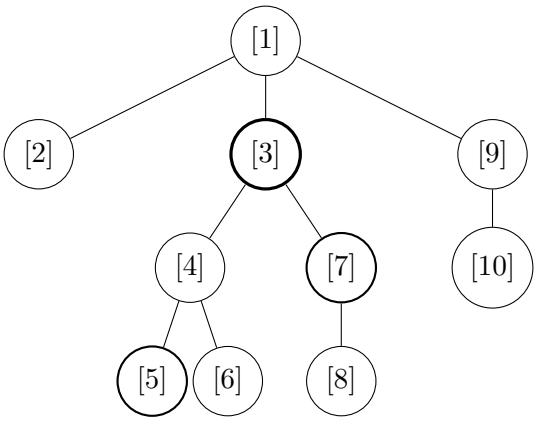
Над овако конструисаним сегментним стаблом се операције врше на уобичајен начин. Када се врши ажурирање неког елемента низа, установи се пут од корена до одговарајућег листа, па се ажурирају сви чворови на том путу полазећи од листа. У случају тражења минимума из опсега $[i, j]$ се «разбије» као унија што је могуће мање скупова на које се односе чворови. Очигледно, изградња стабла ради у $\Theta(n)$, а операције у $\Theta(\log n)$ времену.

Најједноставније примене RMQ су видљиве већ из поставке. Тако се, на пример, можемо неколико пута поставити питање: од оних хотела из ове улице до којих могу да стигнем за не више од t минута, у ком је најмања цена собе?



Слика 21: Изглед изграђеног сегментног стабла у RMQ проблему

Постоје неке примене RMQ које нису из прве очигледне. Једна од најпознатијих је веза са задатком који делом спада у теорију графова, ***lowest common ancestor*** (скраћено LCA) проблемом: за дато уређено стабло (дефинисани су предачки односи) се поставља питање који је први заједнички предак чворова a и b. LCA може директно да се сведе на RMQ проблем (начин на који се то постиже овде неће бити обрађен). Неке директне примене LCA се тичу установљавања природе неких веза у уређеним стаблима, а сложеније нису предмет овог рада. Интересантна је и примена RMQ на стринговима, с тим што тада често долази и до појаве суфиксних стабала.



Слика 22: LCA чворова [5] и [7] је [3]

3.4 *Lazy propagation*

Када се врши операција ажурирања (*Update*) сегментног стабла, неопходно је изменити вредности у $\Theta(\log n)$ чворова. Уколико је природа ажурирања таква да промена није на нивоу појединачног листа, већ читавог опсега, овај број чворова постаје доста већи и сложеност овакве операције би могла да буде реда величине $\Theta(n)$ (промене настају на нивоу целог стабла). Неки од њих ће са врло малом вероватноћом бити размотрени при позиву упита (највише два чвора са исте висине учествују у разбијању скупа над којим се врши упит), посебно листови. Најчешће се приступа вредностима уписаним у чворове блиске корену. Одавде следи основна идеја ***lazy propagation*** технике: при позиву операције *Update* се не измене сви чворови на путу до листа. У ствари, ово је техника одложене измене.

Уместо да се ажурира вредност у великом броју чворова једног сегментног стабла, само се упамти да до измене треба да дође и ажурира се вредност у корену. Када наступи нека друга операција, при разматрању неког чвора

постоји могућност да постоје ажурирања која је неопходно размотрити. Тек тада се обавеза измене «проследи» његовој деци (сем у случају када је реч о листу).

У зависности од конкретног задатка се мења и тежина имплементације (најлакше је за упите који се тичу суме). За сваки чвор у сегментном стаблу је углавном довољно памтити једну измену коју је неопходно применити.

Пример 9. Размотримо сада динамичку варијанту RMQ проблема са следећом изменом: уместо да се мења вредност једног елемента, све вредности из неког интервала се увећавају за неки одређени број x.

Имплементирана су решења овог проблема са и без коришћења *lazy propagation* технике, након чега су мерена времена извршавања при различитим величинама улазног низа (N) и бројем упита (Q), при чему су низ и упити насумично генерисани. Следе табеле са којих могу да се очитају времена извршавања програма.

Одавде се јасно види да без овакве технике у примерима где се користи овакав упит постаје бесмислено не користити *lazy propagation*. △

3.5 Уопштење за више димензије

До сада су била разматрана сегментна стабла у случајевима када су сви скупови били непрекидни одсечци или интервали на некој правој. Међутим, у пракси се често срећу проблеми који захтевају обраду великог броја тачака или скупова који се налазе у просторима димензије 2, 3, 4, ..., k. За такве ситуације је опет могуће користити неку варијанту сегментног стабла.

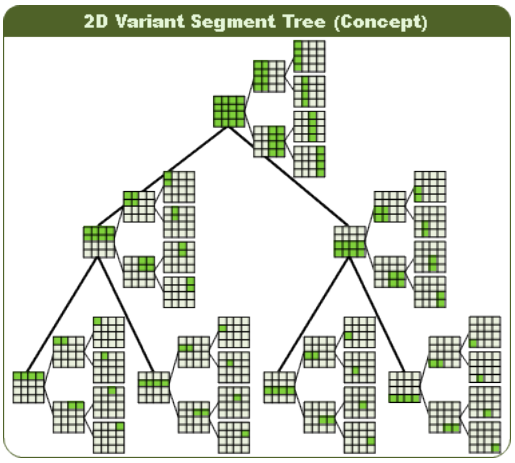
У примеру 5 уведен је појам хиперкоцке. Он се може иопштавати, али нама ће од посебног интереса бити **хиперправоугаоници**. Пошто очекујемо да се у просторима димензије 1 они дегенеришу у сегменте, у просторима димензије 2 у правоугаонике, а у просторима димензије 3 у квадре, тада је природно дефинисати:

Дефиниција 5. Нека је $\mathbb{R}^k$ придружен одговарајући афини простор, а $[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]$ су сегменти. Тада је $[a_1, b_1]\times\cdot\times[a_k, b_k]$ **хиперправоугаоник** (овде је $\times$ Декартов производ).

Дакле, основна идеја је да се изврши сегментација и формира стабло, при чему ће листовима сада бити представљени различити елементарни хиперправоугаоници. У случају димензије $k = 2$ реч је о правоугаоницима. Као што је примећено у досадашњим разматрањима у случају $k = 1$, они ће бити формирати од полазних k-димензионих хиперправоугаоника (који можда нису експлицитно дати на улазу, али се дају одредити у задатку).

Размотримо структуру једног k-димензионог сегментног стабла. Најпре се креира сегментно стабло тако што се узимају у обзир пројекције на x-осу координатног система (ово је основно, једнодимензионо сегментно стабло). Међутим, сваком чвору овако контруисаног стабла се придружи и сегментно стабло реда $k - 1$ које се даље слично дефинише: најпре се конструише над пројекцијама на y-осу, па, ако постоји више димензија, на z итд.

Измене које се врше у оваквим ситуацијама су једноставне, само је неопходно водити рачуна о координатама. Исто тако је лако испитати и временске сложености ових поступака. Нека је димензија сегментног стабла једнака k. Ако се изузме полазно сортирање, изградња захтева $\Theta\left(n^k\right)$ време (у сваком «нивоу» додајемо још $\Theta(n)$ чворова). Када поставимо питање у вези са неким хипер-



Слика 23: Значење сваког чвора 2d сегментног стабла: пример над дискретним скупом (извор: [11])

правоугаоником или вршимо ажурирање неког елемента, најпре изолујемо његов положај по x-координати, а онда у сваком од одабраних чворова првог «нивоа» сегментног стабла радимо изолацију у следећем «нивоу» итд. што даје време $\Theta\left(\log^k n\right)$. Као што видимо, изузетно је изражена клетва димензионалности: временска су експоненцијално лошија што је већа димензија.

Пример 10. Дата је мапа неког региона, при чему је он приказан сегментисан на квадратиће и позната је средња висина региона у оквиру тог квадратића. С времена на време се ажурирају подаци због ранијих грешака у мерењима. С друге стране, редовно се од нас тражи да доставимо податке о тачки најмање надморске висине у некој области.

У ствари, овај задатак се своди на RMQ у две димензије. Уколико бисмо изградили дводимензионално сегментно стабло над датом матрицом, тада на сваки упит можемо да одговоримо у $\Theta\left(\log^2 n\right)$ времену, при чему нам је неопходно $\Theta\left(n^2\right)$ време за изградњу стабла. △

Нажалост, иако је имплементација сегментних стабала у више димензија једноставна, она не даје задовољавајуће резултате. Зато се у сложенијим геометријским захтевима користе неке ефикасније структуре података.

4 Фенвиково стабло

Проблеми који се решавају сегментним стаблом се често јављају у облику који захтева имплементацију две врсте операција:

1° промена вредности карактеристичне за неки елементарни интервал;

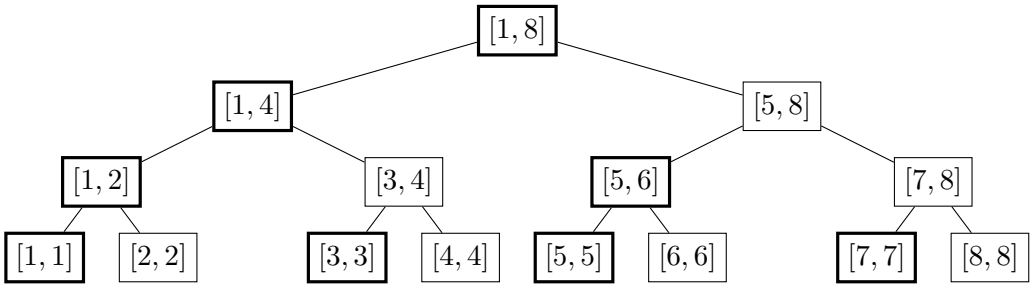
2° одређивање особине која зависи од вредности карактеристичних за све интервале из неког опсега тј. од i-тог до j-тог интервала.

У не малом броју проблема решивих сегментним стаблом се или експлицитно врши промена облика операције 2° или је могуће свести је на једноставнији облик који гласи: одређује се особина која зависи од вредности које су карактеристичне до неког почев од првог елементарног интервала. Овај резон је широко применљив и у једноставнијим задацима: збир свих елемената неког низа од i-те до j-те позиције да се одредити користећи збирове од почетка па до позиција $i-1$, односно j. У оваквим ситуацијама се не прибегава имплементацији сегментног стабла, већ се користи нешто ефикаснија структура података коју називамо **Фенвиково стабло**[11].

Мотивација за коришћење Фенвиковог стабла над сегментним је вишеструка. Кључни разлози су једноставнија имплементација и нешто мања потрошња ресурса (измена у константи). Из тог разлога је примена Фенвиковог стабла широко распрострањена. Ипак, држећи се теме (примене у геометрији), овде се неће посветити нарочита пажња њима.

Ипак, наведимо основну структуру једног Фенвиковог стабла. Фенвиково стабло се, као и сегментно, такође гради над неким полазним низом (елементи тог низа су листови) у својој најчешћој варијанти (дискретни подаци). Ради једноставности, претпоставимо да је број елемената овог низа $n = 2^k$, и најпре изградимо сегментно стабло над овим низом.

Приметимо, међутим, да нам за тражено својство свих елемената низа од првог до j-тог нису неопходни сви чворови да би се одржало логаритамско време. Ако је $j = 1$, довољан нам је само први лист; ако је $j = 2$, довољан нам је његов родитељ; ако је $j = 3$, довољан нам је чвор из случаја $j = 2$ и чвор који представља трећи елемент низа; ако је $j = 4$, довољан је чвор који описује прва четири елемента итд. Већ сада уочавамо извесну везу са бинарним записом броја j, што се уочава из правилности којим се бирају чворови.

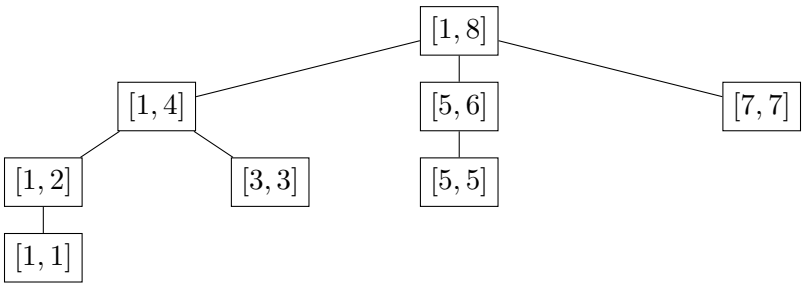


Слика 24: Минималан скуп чворова неопходан за представљање произвољног скупа $[1, j]$, где $1 \leqslant j \leqslant 8$

При представљају броја j узмимо у обзир његов бинарни запис. Узмимо, примера ради, $j = 13 = 01101_2$. Приметимо да се карактеристика $[1, j]$ може

[11] За ово стабло су учестали и неки други називи, између осталих и: ***binary indexed tree*** (BIT), **кумулативно стабло**.

представити као укупна карактеристика следећих подскупова: $[00001_2, 01000_2]$, $[01001_2, 01100_2]$ и $[01101_2, 01101_2]$. А управо се одговарајући чворови налазе у сегментном стаблу! У ствари, стално додајемо све мање степене двојке док не добијемо j: $13 = 8 + 4 + 1$ (мањи су, узимамо чворове различитих висина: ако бисмо одабрали два суседна чвора исте висине, то је исто као да смо одабрали њиховог заједничког родитеља).



Слика 25: Један начин представљања Фенвиковог стабла: видимо да овако није бинарно, али га ипак сматрамо значајним јер представља «контракцију» сегментног стабла

Наравно, овакво решење је могуће, али Фенвиково стабло ће садржати искључиво оне чворове који су неопходни за представљање било ког скупа облика $[1, j]$. Дакле, нису неопходни чак ни сви листови: укупно ће бити баш n чворова у стаблу и то ће j-ти од њих бити последњи који учествује у представљању $[1, j]$.

Погледајмо на шта се тада своде претходно наведене операције:

1° Изменом вредности неког елемента полазног низа постаје неопходно да се ажурирају извесни чворови у стаблу. У случају $j = 13$ и $n = 16$ поступак је следећи: $01001_2 \to 01010_2 \to 01100_2 \to 10000_2$. У ствари, идеја је да се последњи «блок» јединица замени нулама, а да се прва нула веће тежине замени јединицом. Приметимо да је неко тренутно x облика $x = \ldots 00111 \ldots 11000 \ldots 00_2$. Инвертовањем свих битова (непотпуни комплемент) добије се број облика $\tilde{x} = \ldots 11000 \ldots 00111 \ldots 11_2$. Додавањем јединице овој вредности (потпуни комплемент, односно супротан број) добије се $-x = \ldots 11000 \ldots 01000 \ldots 00_2$. x и $-x$ се у бинарном запису поклапају искључиво у цифрама чије тежине одговарају «крајњим нулама» и једној јединици. Тај заједнички део и та прва јединица се добију операцијом бинарне конјункције: x `and` $(-x)$. Додавањем x овоме се добије $\ldots 01000 \ldots 00000 \ldots 00_2$, што је управо следећи тражени број. Дакле, корак је $x + (x$ `and` $(-x))$.

2° Друго кључно питање јесте како разложити неки дати број j на претходно наведен начин: у случају $13 = 01101_2$ било је $01101_2 \to 01100_2 \to 01000_2$. Закључујемо да је неопходно да последњу јединицу у бинарном запису неког броја x заменимо нулом. Постоји врло једноставан начин да се ово уради: користећи чињеницу да се x и $x-1$ разликују само у оним битовима почев од оних најмање тежине па до «најмање» јединице у запису x следи да се следећи број у представљању j добија применом x `and` $(x - 1)$.

Имплементација Фенвиковог стабла нећемо вршити коришћењем стабала и показивача, већ само једног јединог помоћног низа. Зато је учестало коришћење израза «кумулативне табеле».

Понашање у пракси разматрамо у наредном примеру, основној примени ове структуре, где ћемо размотрити и ефикасност.

4.1 Израчунавање префиксних сума

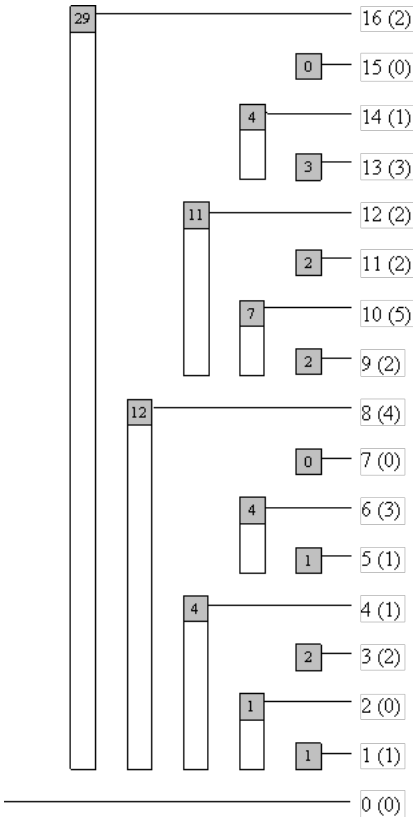
Префиксне суме неког низа A представљају неки низ P_A тако да: $P_A[1] = A[1]$, $P_A[2] = A[1] + A[2]$, ..., односно:

$$P_A[k] = \sum_{i=1}^{k} A[i],$$

уз додатну дефиницију $P_A[0] = 0$.

Основна примена префиксних низова је за ефикасно израчунавање збира $A[i] + A[i+1] + \ldots + A[j]$: то се дâ израчунати преко $P_A[j] - P_A[i-1]$.

Подсетимо се примера 8: низ A је променљив и вршимо велики број упита у којима се захтева збир свих елемената низа A почев од једне и закључно са неком другом позицијом. За ефикасно одређивање овог збира користимо префиксне суме $P_A[k]$ (на основу малопре наведене релације).



Слика 26: Скица «кумулативних табела» (извор: [8])

Уочимо Фенвиково стабло над оваквим низом и одговарајући низ F_A: карактеристика неког сегмента $[i, j]$ (која ће бити уписана у елемент $F_A[j]$) је управо збир свих елемената од i-тог до j-тог. Одређивање $P_A[k]$ се одређује применом операције 2°, а измена неког елемента управо одговара операцији 1°.

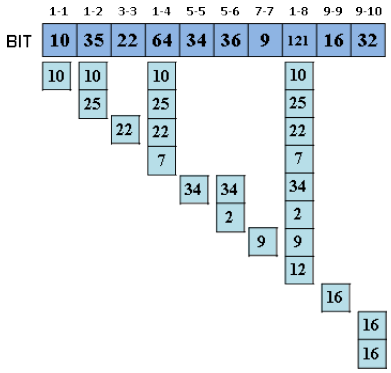
Следи део имплементације:

Кôд 4

Честе једноставне имплементације операција у Фенвиковом стаблу

```
// argumenti: nova vrednost, pozicija izmenjenog elementa,
//           polazni niz, indukovano stablo, dimenzija niza
void Update(int newVal, int pos, int A[], int FA[], int n)
{
    int k = pos;
    int v = newVal - A[pos];
    while (k <= n)
    {
        FA[k] += v;
        k += k & (-k);
    }
}

// argumenti: indeks do kog ide prefiksna suma,
//           indukovano Fenvikovo stablo
int FindPrefixSum(int pos, int FA[])
{
    int k = pos;
    int sum = 0;
    while (k > 0)
    {
        sum += FA[k];
        k = k & (k - 1);
    }
    return sum;
}
```

Операција оваквог типа налази разне примене и редовно се јавља у неким сложенијим проблемима. Ипак, нећемо се даље њима бавити.



Слика 27: Други начин представљања «кумулативних табела» (извор: [11])

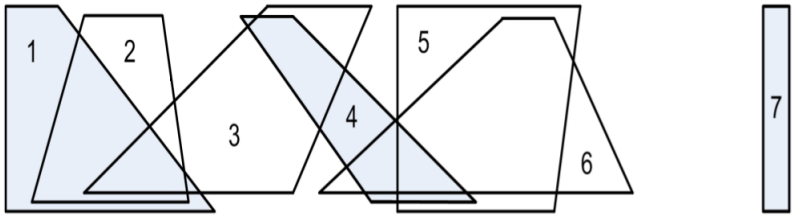
4.2 Поређење Фенвиковог и сегментног стабла

Није могуће решити све задатке решиве сегментним стаблом и Фенвиковим (RMQ), али је тачно обратно. Међутим, имплементација коришћењем сегментног стабла је нешто компликованија, а уједно и мање ефикасна (иако су временске сложености свих операција идентичне, Фенвиково стабло конзистентно поседује предност у пракси јер је број операција је вишеструко мањи).

Lazy propagation стратегија и овде пролази на потпуно исти начин, као и уопштење за више димензије. Дакле, Фенвиково стабло би требало користити кад год је то могуће.

Наведимо, на крају, и један пример из геометрије:

Пример 11. **(Балканска олимпијада из информатике, 2011.)** Размотримо две хоризонталне праве. Трапез T_i између ове две праве има два темена на горњој и два темена на доњој правој (видети слику). Означимо са a_i, b_i, c_i и d_i апсцисе горњег левог, горњег десног, доњег левог и доњег десног темена трапеза T, респективно. Подскуп S ових трапеза се назива независним ако се никоја два трапеза из S не пресецају. Одредити величину највећег независног скупа трапеза ако је број трапеза n, где $n \leqslant 10^5$.



Слика 28: Пример скупа трапеза (врхови и дна трапеза су померени ради видљивости)

Најпре се сви ови трапези сортирају по горњој десној граници и по том поретку се формира једно Фенвиково стабло (над дискретним скупом: сваки лист представља један трапез). Сваки лист представља највећи број дисјунктних трапеза који могу да се одаберу закључно са оним који тај лист представља. Сваки други чвор узима вредност максимума своја два детета, а тиме се остварује могућност динамичког RMQ.

Даље се издвоје доња лева и доња десна темена свих трапеза и заједно сортирају, при чему се за свако памти и да ли је лево или десно и на који трапез се односи. Уколико је реч о левој граници, трапезу се придружи број који је за један већи од највећег броја у Фенвиковом стаблу, при чему се у обзир узимају они трапези чија је горња десна граница лево од горње леве границе тренутног трапеза. Уколико је реч о десној граници, израчунати број се забележи у Фенвиковом стаблу, чиме је омогућено његово «учешће» у резултатима за касније обрађене трапезе.

Коначан резултат је максимум од свих бројева придружених трапезима. Обавља се $\Theta(n)$ упита над Фенвиковим стаблом, те је $\Theta(n \log n)$ укупна временска сложеност оваквог решења. △

5 kd стабло

Претпоставимо да нам је дат скуп тачака у простору. Тада постоји једна класа геометријских упита који су сличног типа и са великим бројем примена:

- Одређивање тачке из овог скупа која је најближа некој датој — ***nearest neighbour*** (NN) упит. Основна примена се јавља у организацији служби са хитним реакцијама: ватрогасна служба, полиција, хитна помоћ. Може да се користи и у савременим *online* апликацијама за одређивања најближег локала неког типа. Овај упит се јавља и у неким хеуристикама и оптимизационим проблемима (проблем трговачког путника).
- Одређивање највећег могућег правоугаоника који не садржи ниједну тачку. Ово може да буде корисно уколико се планира изградња неког објекта што је могуће веће површине.
- *Range searching* упити: одређивање броја тачака у некој области, неког екстремалног својства у области...

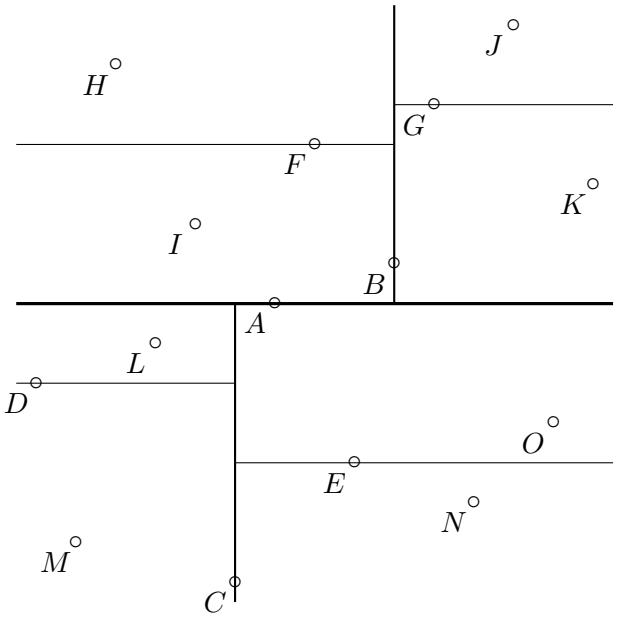
Од читаве ***spatial query*** класе упита (просторни упити), применљивим при раду са базама података у којима су посматрани објекти геометријске природе, посебно смо издвојили ове. Очекујемо да индексирање свих тачака преко низа неће дати задовољавајуће резултате по питању ефикасности, те нам је неопходан некакав **просторни индекс**. За овакве потребе користи се **kd стабло**[12].

Суштина структуре kd стабла се заснива на сегментацији простора: најпре се све тачке поделе у два мање-више једнако бројна подскупа које раздваја један услов: однос са неком координатом (све тачке чија је одговарајућа координата мања од те референтне налазе се у једном скупу, оне чија је већа налазе се у другом скупу, а у случају једнакости постоје разлике у зависности од приступа). Након тога се рекурентно врши подела ових подскупова на исти начин, али се сада врши раздвајање по некој другој координати (ако је прво раздвајање било по x координати, даље иде по y, па по z итд...када се дође до последње, поново се дели по x координати).

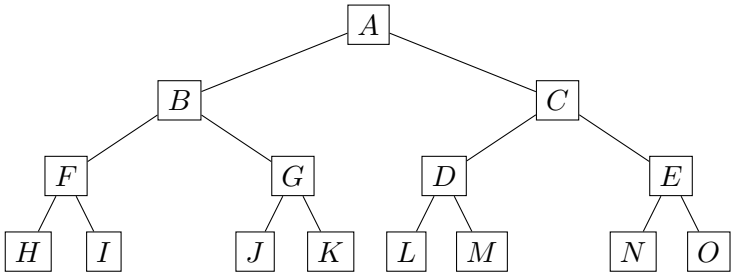
Реч је, дакле, о стаблу над k-димензионим подацима. У случају $k = 1$ се памте тачке које су у сортираном поретку и ово је онда бинарно стабло претраге (пример 1, одељак 2.1). У случају $k = 2$ се оперише са једном равни, у случају $k = 3$ са простором итд.

Испоставиће се да је овако формирано стабло врло подесно за задатке наведеног типа. Да би се стабло креирало, неопходно је одредити ту границу која се повлачи ради поделе на два подскупа. Једноставно би било узети аритметичку средину, али је то незгодно у великом броју случајева јер тачке углавном нису «равномерно» распоређене (наравно, уколико тежимо прављењу балансираног стабла). Да би била извршена подела на два скупа приближно једнаке бројности, неопходно је одредити **медијану** (при томе ћемо игнорисати уобичајену дефиницију ако је број елемената паран: аритметичка средина два средња елемента).

[12]Користи се и запис **k-d** стабло у литератури. При пут када је уведено («Multidimensional binary search trees used for associative searching», Бентли, 1975.) се говорило о «уопштењу бинарног стабла претраге».



Слика 29: Подела простора на сегменте



Слика 30: kd стабло формирано над сегментисаним простором

5.1 *Quickselect* алгоритам

У наивном случају, медијана неког низа се одреди тако што се најпре изврши сортирање, па се узме средњи елемент. Ипак, овакав поступак захтева $\Theta(n \log n)$ време, где је n дужина низа. Од осталих метода издвајамо, због једноставности, ***Quickselect*** алгоритам.

Quickselect има сличну идеју као *Quicksort* алгоритам сортирања. Посматрајмо низ A који има n елемената. Нека се тражи елемент са позицијом k у сортираном низу тј. k-ти најмањи. Најпре издвајамо неки елемент x (који зовемо **пивот**) и испитујемо који су елементи у A мањи, а који већи од њега (у случају једнакости је потпуно свеједно у коју групу га стављамо). Уколико преуредимо низ тако да су сви елементи који су мањи од x пре њега у низу, а они који су већи од x после њега, тада се x налази на истом месту у ком би се налазио у сортираном низу — означимо ту позицију са p. На основу овога можемо да закључимо да ли је он мањи, већи или баш једнак траженом k-том најмањем елементу, те раздвајамо случајеве:

1° уколико је баш $p = k$, тада је x решење;

2° уколико је $p > k$, тада се тражени елемент налази у делу низа закључно са елементом на позицији $p - 1$;

3° уколико је $p < k$, тада се тражени елемент налази у делу низа почевши од елемента на позицији $p + 1$.

У случају да k-ти најмањи елемент није одређен, понавља се поступак, с тим што се пивот бира из допустивог региона. Овај поступак се понавља (дакле, мењају се лева и десна граница у којој се тражени елемент налази) и сигурно ће се коректно завршити.

Посматрајмо рад овог алгоритма на примеру низа $(1, 6, 2, 9, 4, 7, 5)$, где тражимо четврти најмањи елемент:

$$
\begin{gathered}
(|1, \mathbf{6}, 4, 9, 2, 7, 5|), \\
(|1, 4, 2, 5, \mathbf{6}, 7, 9|), \\
(|1, \mathbf{4}, 2, 5|, 6, 7, 9), \\
(|1, 2, \mathbf{4}, 5|, 6, 7, 9), \\
\Big(1, 2, 4, \Big|\boxed{\mathbf{5}}\Big|, 6, 7, 9\Big).
\end{gathered}
$$

У најгорем случају, *Quickselect* ради у $\Theta\left(n^2\right)$ времену (на пример, ако стално бирамо први елемент за пивот, а низ је већ сортиран). Ипак, очекивано и просечно време рада овог алгоритма је $\Theta(n)$ јер је очекивани број итерација за налажење медијане врло мали (исти аргумент правда узимање да је временска сложеност *Quicksort* алгоритма дата са $\Theta(n \log n)$). Предлаже се насумично бирање пивота да би се постигли овакви ефекти.

5.2 Принцип функционисања kd стабла

Ради праћења рада kd стабла неопходно је најпре набројати све операције које је над њим могуће вршити. За неке је оптимизовано (креирање, *NN search*), а за неке уопште није (убацивање новог елемента, брисање).

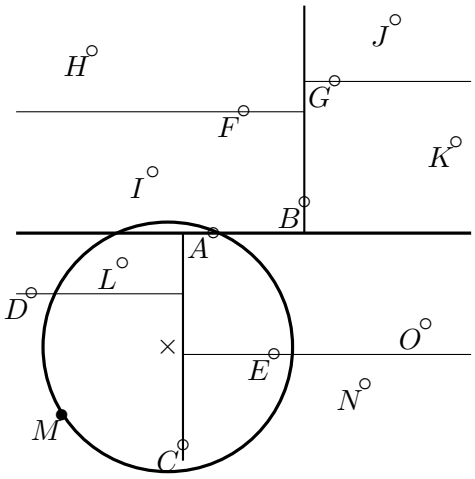
Најпре наведимо помоћне податке у структури стабла. Поред пуких показивача на децу и података о тачки (могуће имплементирати такође преко показивача), требало би памтити и извесне помоћне податке: све оне координате које говоре о ком се делу простора ради (пошто неки сегменти простора имају границе облика $\pm\infty$, постоје различити начини превазилажења овог проблема у зависности од конкретне ситуације: или се зада неки број који излази из предвиђених ограничења за улазне податке или се уводни помоћна променљива логичког типа).

Креирање. За креирање стабла се користи рекурзиван поступак. Реч је о креирању стабла над неким делом A, низа тачака. Уколико не постоји ниједна тачка у издвојеном делу низа A, реч је о *null* чвору. У супротном, користимо поменути *Quickselect* алгоритам, где се тражи $\lfloor n/2 \rfloor$-ти најмањи елемент. Тада се он придружује актуелном чвору и разматрају се деца која се добијају из низа који је модификован након примене *Quickselect*, врши се подела по одговарајућој хиперравни. Критеријум по ком се уреде елементи зависи од висине на којој се чвор налази (испитује се остатак при дељењу са бројем димензија).

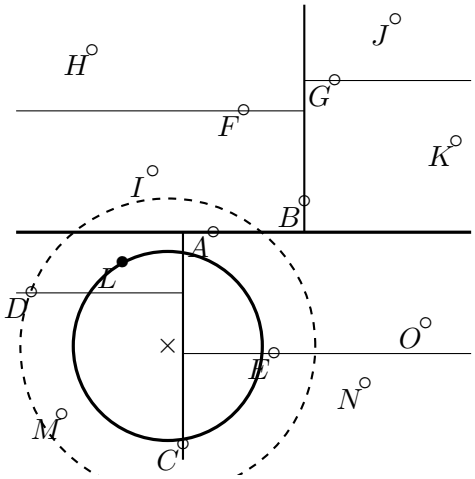
Сложеност креирања kd стабла је $\Theta(n \log n)$ (реч је о очекиваној сложености!). Следи оправдање које се заснива на грубој процени (не и доказ!). Наиме, постоји око $n/2$ листова у стаблу и они су на висини $\log_2 n$. Да би

се стабло конструисало, било је неопходно применити *Quickselect* одређен број пута: једном за низ дужине n, два пута за низ дужине $n/2$, четири пута за низ дужине $n/4$ итд. Дакле, за сваку висину се врши $\Theta(n)$ операција. Следи да је сложеност изградње стабла баш $\Theta(n \log n)$. Интересантно је да овде k уопште не учествује изузев различитих k критеријума по којима се врши сортирање, па су ефекти клетве димензионалности овде минимални.

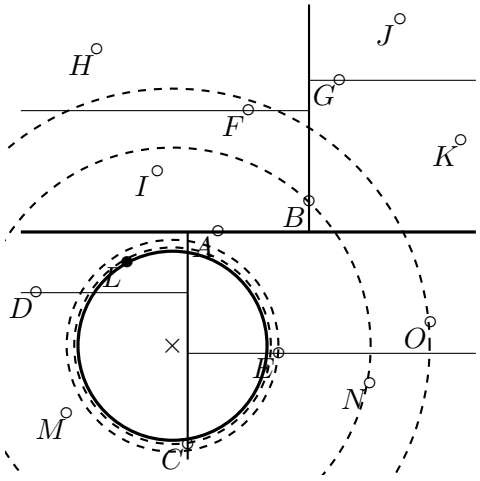
Одговор на NN упит. Најпре се установи у ком се тачно сегменту простора налази поменута тачка — претрага стабла која одређује одговарајући лист је једноставна (уз уобичајене проблеме у случају једнакости). Тачка уочена у том сегменту простора је сигурно један од кандидата и актуелни минимум, најближи сусед. Докле год постоји неки чвор такав да део простора који описује има удаљеност мању од актуелне минималне (тј. уколико одговарајућа кугла пресеца граничну хиперраван), тада је он кандидат за новог најближег суседа.



Слика 31: Први корак у раду претраге: тачка из упита се налази сегменту у ком је M, па је M при избор за најближег суседа



Слика 32: Други и трећи корак у раду претраге: круг сече сегмент у ком је D, али то је много веће од растојања до M; L је следећи избор и то је нови најближи сусед



Слика 33: Преостали кораци у претрази: кандидати су и C, E, N и O, али је на крају баш L тачка која је NN; остатак простора не долази у обзир при претрази

Асимптотска анализа оваквог поступка за одређивање најближег суседа је проблематична. Уколико су тачке насумично генерисане, тада се очекује $\Theta(\log n)$ време. Фридман, Бентли и Финкел у раду из 1977. године доказују да је могућа претрага у логаритамском времену.

Ефекти клетве димензионалности често доводе до претраге много више чворова, чиме се сложеност увећава. У случајевима када је k упоредиво са n рад kd више нема толико предности над линеарном претрагом. Ипак, у пракси се овакви случајеви не разматрају, те узимамо да су ефекти клетве димензионалности минимални.

Интересантно је приметити да се овакав упит може прилагодити на све истакнуте метрике (сва $\mathbb{L}_p^k$ уопштења Еуклидског растојања). За све метрике важи да се у кугли чији је центар тачка из упита и чији радијус даље одређује NN тачка не налази ниједна тачка. У зависности од избора метрике се добијају различити интересантни резултати.

Додавање новог чвора. Убацивање новог чвора је велики проблем за бинарно стабло претраге јер нарушава балансираност, те није неочекивано да се то дешава и овде. Конкретни задаци у којима долази до примене kd стабла углавном «статични» (нема ни додавања ни брисања тачака). Пошто је проблем ребаланса иначе врло компликован, овде му неће бити посвећена нарочита пажња. Постоје нека **уопштења** која заобилазе овај проблем: адаптивно kd стабло, kd B стабло, псеудо kd стабло...

Брисање задатог чвора. Важи исти коментар као и за додавање чвора.

Пример 12. У једном граду постоје улице (хоризонталне на мапи) и авеније (вертикалне на мапи) и оне образују једну целобројну решетку. Време неопходно за прелазак од једне до њој суседне раскрснице означимо са 1. У току ноћи велики број људи зове градски такси, а на неким раскрсницама се налазе такси станице на којима се увек налази бар један такси. У интересу такси службе је да сви клијенти буду услужени што раније. За сваку особу која зада свој положај одредити такси станицу са које ће такси доћи, као и одговарајуће време.

У ствари, тражи се она такси станица чије је Менхетн растојање од путника најмање могуће — ово је једноставан NN упит за $\mathbb{L}_1^2$ метрички простор. Дакле, након креирања 2d стабла над свим станицама се примењују одговарајући упити за сваког путника. △

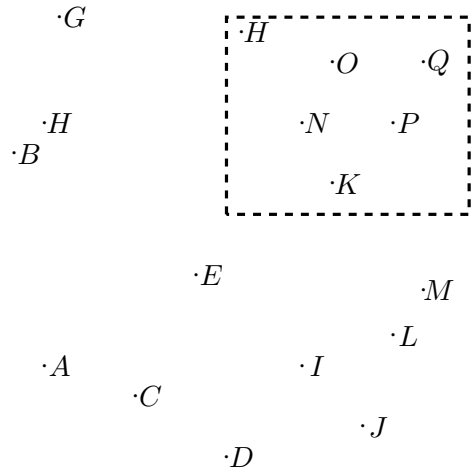
Пример 13. Дат је скуп тачака од N у простору. Задаје се Q тачака, а уз сваку иде упит: одредити ону коцку са центром у тачки из упита такву да су јој ивице паралелне координатних осама и не садржи ниједну другу тачку.

Највећа коцка која не садржи ниједну тачку има бар једну на свом рубу. Тражимо метрику чије су кугле управо коцке чиеј су ивице паралелне координатним осама. Ово се постиже у $\mathbb{L}_\infty^3$ метричком простору, чиме се даље проблем своди на просту примеру NN упита. Временска сложеност овог решења је $\Theta\left((N+Q)\log N\right)$. △

5.3 Интервални упити

kd стабла су изузетно ефикасна кад год се примењује NN упит. Ипак, она се користе и када је операције *range query* типа, као већина њих раније поменутих код сегментних стабала: одређивање неке карактеристике уколико се посматра неки сегмент. Као и при раду са вишедимензионим сегментним стаблима, овде се при упиту ради о неком хиперправоугаонику.

Овде постоје извесне сличности као при раду са сегментним стаблом. Илуструјмо ово на наредном примеру: тражи се број тачака који се налази у хиперправоугаонику из упита.



Слика 34: Колико има тачака у правоуганику?

Креирањем kd стабла долази до сегментације простора. Сваком чвору доделимо број који говори о броју тачака унутар њега (очигледно збир вредности у деци увећан за један, при чему је у листовима једнак јединици). Упит се разматра полазећи од корена и наредни рекурзиван поступак се примењује при обради неког чвора:

1° уколико је реч о *null* чвору, не ради се ништа и ова грана рекурзије се завршава;

2° уколико је одговарајући сегмент подскуп хиперправоугаоника из упита, тада све тачке садржане у овом чвору улазе у избор и ова грана рекурзије се овде завршава;

3° уколико не постоји пресек са посматраним хиперправоугаоником, не разматра се ниједан даљи потомак из овог подставља и ова грана рекурзије се овде завршава;

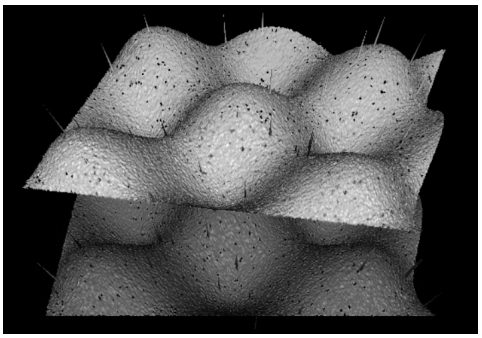
4° у супротном, провери се да ли се тачка придружена актуелном чвору налази унутар хиперправоугаоника, након чега се разматрају деца тог чвора.

Јасно је да овај поступак даје жељене резултате. Анализа ефикасности није претерано компликована, али је корисно продискутовати је. Код сегментног стабла је овакав поступак давао логаритамску сложеност, али је био ограничен на једну димензију (2d сегментно стабло се показало као неефикасно). Међутим, овде је кључна разлика управо тâ што хиперправоугаоници из упита не могу да се прикажу као унија неких сегмената који се појављују у стаблу, чиме је далеко повећан број обиђених чворова. Доказује се да је временска сложеност овог упита дата са $\Theta\left(n^{1-\frac{1}{k}}\right)$, где је n број тачака, а k број димензија. У случају 2d података је реч о $\Theta\left(\sqrt{n}\right)$, али је овде клетва димензионалности изузетно изражена (чак и када заборавимо да Θ крије константу).

Дакле, kd стабла нису погодна за интервалне упите. За ове, а и многе друге, од користи је примена R стабала (која у већини варијанти нису бинарна).

5.4 Примене

Најчешће примене kd стабала се заснивају на NN упиту. У складу са тим се јављају као предложено решење у великом броју проблема у којима се траже некакве хеуристике — нека NN спаривања се природно намећу у пракси. Размотрићемо ***iterative closest point*** (ICP) хеуристику, применљиву у различитим проблемима, као и веома актуелан ***ray tracing*** проблем.



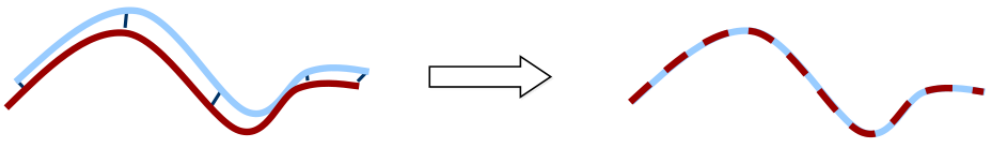
Слика 35: Која изометрија слика први рељеф у други? (извор: рад са Стенфорда, ефишн веријенц)

Претпоставимо да су нам дата нека два скупа тачака (у 2d, 3d или неком другом простору), A и B, који су на неки начин корелисани: то су два скупа једнаке кардиналности, свака тачка из првог скупа одговара тачно једној тачки из другог и обратно. За скуп тачака се користи назив ***point cloud***. Дакле, постоји нека трансформација (тражимо бијекцију) која пресликава A у B — ова операција се зове ***point cloud stitch***. Посматрајмо мало конкретније: реч је о неким истакнутим тачкама у простору у два различита временска тренутка које је посматрач уочио, при чему се он у међувремену креће. Тада је неопходно одредити ону ригидну трансформацију (директну изометријску трансформацију) која што боље пресликава први скуп у други. Овде ћемо

тачке приказивати у наредном облику: поред стандардних координата ћемо додати још једну «помоћну» која је увек једнака јединици. У случају 2d простора пишемо тачке у облику $\begin{pmatrix} x & y & 1 \end{pmatrix}^T$, а у случају 3d користимо **кватернионски**[13] **запис**: $\begin{pmatrix} x & y & z & 1 \end{pmatrix}^T$.

ICP је хеуристика која се користи да би се та трансформација τ пронашла и подсећа на познате итеративне нумеричке методе за решавање једначина (како оних облика $f(x) = 0$, где је f позната реална функција, тако и матричних, диференцијалних, система диференцијалних...). На тренутак оставимо по страни чињеницу да се тражи бијекција, већ за сваку тачку из A тражимо ону из B у коју желимо да се преслика. За ово ћемо користити њој најближу тачку[14] из B (обично се користи стандардна Еуклидска метрика) — ово је корак који се ефикасно обави применом kd стабала и временска сложеност овог корака је $\Theta(n \log n)$. Оправдајмо овај избор: уколико смо већ одредили тражену трансформацију, тада долази до мање-више поклапања свих тачака из $\tau(A)$ и B. Најпре дефинишемо неке критеријуме конвергенције (нпр. да је просечно растојање између упарених тачака довољно мало). Уколико радимо у 3d простору, трансформацији τ придружујемо матрицу која има следећи облик:

$$\mathbf{I} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$



Слика 36: Узима се најближа тачка из другог point cloud-а

Приметимо да матрица ове трансформације (помножена са кватернионом даје кватернион) садржи две подматрице: **T** и **R**. Подматрица **T** одговара транслацији за вектор $\begin{pmatrix} T_x & T_y & T_z \end{pmatrix}^T$. Подматрица **R** одговара некој ротацији у Еуклидском простору и добије се као производ неке три матрице ротације око различитих оса, односно:

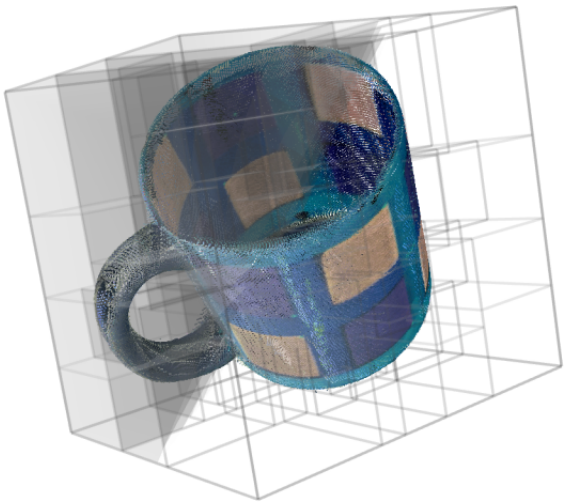
$$\mathbf{R} = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & -\sin\omega \\ 0 & \sin\omega & \cos\omega \end{pmatrix}.$$

Важно је напоменути да τ, којој придружујемо **I**, представља трансформацију која се састоји од композиција једне транслације и једне ротације, при чему се прво примењује ротација, па онда транслација. Овако може да се представи свака ригидна трансформација.

Након што се упаре тачке, неопходно је оценити матрицу која врши претпостављено пресликавање. За критеријум најбоље оцене може да се користи

[13] **Кватерниони** су проширење скупа комплексних бројева. Могу да се запишу у облику уређене четворке (a, b, c, d) или у алгебарском облику $a + bi + cj + dk$, при чему $i^2 = j^2 = k^2 = -1$, а правила сабирања и множења подсећају на правила при раду са комплексним бројевима.

[14] Ово је *point to point* варијанта, постоји и *point to plane* приступ који је ефикаснији.



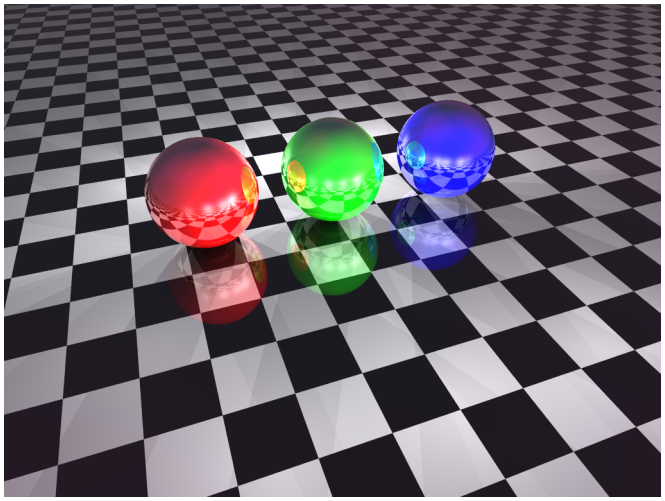
Слика 37: За одређивање најближе тачке користе се kd стабла (извор: [12])

метод најмањих квадрата, али се некада, због потреба брзине, користе бржи и нешто непрецизнији критеријуми. При раду са 3d координатама је овај корак нешто компликованији, те се нећемо њиме бавити. У сваком случају, тако се добије нека оцењена трансформација τ_1 са матрицом $\mathbf{I_1}$. Уколико је задовољен критеријум конвергенције, тада је $\tau = \tau_1$. У супротном, поново се групишу тачке користећи исти критеријум (сада је најближа тачка можда нека друга!) и добије се трансформација τ_2, па се поступак можда опет понови и добије се τ_3, τ_4, ..., τ_n и ту се достигне конвергенција. Тада је $\tau = \tau_n \circ \ldots \tau_2 \circ \tau_1$, односно $\mathbf{I} = \mathbf{I_n} \cdot \ldots \cdot \mathbf{I_2} \cdot \mathbf{I_1}$. Ово је комплетан опис ICP алгоритма.

Постоји могућност да не дође до конвергенције. Да би се избегла бесконачна петља, обично се постави горња граница за n (у бесплатно доступној и обимној PCL библиотеци је *default* вредност једнака 50). Некада се, додуше, јави проблем тако што алгоритам не конвергира решењу, већ тек неком локалном минимуму (функције одступања). Практичне примене овог поступка се уочавају у проблемима који се тичу локализације посматрача (нпр. робота) или мапирању простора (може и оба: ***simultaneous localization and mapping*** (SLAM)). За ове потребе се користе специјализовани сензори који процењују удаљености објеката, праве «дубинске слике» (*grayscale* слика на којој интензитет сиве одређује удаљеност), након чега се уоче одговарајуће значајне тачке у простору. ICP хеуристика се користи и у неким другим ситуацијама које захтевају поређење два *point cloud*-а (проблеми класификације и препознавања).

Слика 38: Слика у боји и њена дубинска варијанта (извор: [13])

Ray tracing (праћење зрака) је алгоритам који се јавља у рачунарској графици и применљив је углавном уколико се ради о систему који не захтева превелику брзину или није интерактиван (*real time* системи, попут рачунарских игара, углавном не користе овако детаљне ефекте). Реч је о проблему генерисања реалистичног приказа полазећи од 3d модела, тачке у којој се налази посматрач и извора светлости — кључни проблем се тиче ефеката геометријске оптике попут сенки, одбијања (рефлексије) и преламања (рефракције) светлости.



Слика 39: Једна генерисана слика (извор: [10])

Нећемо улазити у детаље овог алгоритма, али ћемо се осврнути на део у ком се посматра «први пресек» једног зрака (математички моделованог као полуправа) са објектима који се налазе на сцени. У оваквим ситуацијама се користи *surface area* хеуристика (SAH) и захтева коришћење некакве ефикасне структуре података за анализу. Данас ту улогу све чешће имају баш kd стабла јер имају асимптотски оптимално $\Theta(n \log n)$ време, при чему је n број троуглова на које је простор подељен применом триангулације објеката.

6 Закључак

Обрађена стабла нису једина, а могуће су и њихове различите варијанте, уопштења, комбинације. Њихове примене расту развојем тржишних захтева који се тичу просторних података (посебно у мобилним технологијама) и рачунарској графици.

Због сврхе и предвиђеног обима овог матурског рада, нису обрађена наредна стабла (често из разлога што нису бинарна): **интервално**, **quadtree**, **octree**, **M-tree**, **R**... На неким местима су само поменута нека од многобројних уопштења и хибрида. Ипак, трудио сам се да пружим колико-толико комплетан увод у неке основне појмове и средства која се сматрају нужним у поменутим областима: све поменуто је углавном основа која потиче из седамдесетих и осамдесетих година двадесетог века. Не зна се каква будућност предстоји структурама података са геометријским применама: велики је број операција за које није нађено асимптотски оптимално решење. По мом мишљењу, тренутно постоји некаква стагнација у теоријском свету и питање је када ће се појавити нова структура.

Сегментно и Фенвиково стабло, као и идеје иза њих (нарочито «бележење података за претходних 2^k»), имају све учесталија појављивања на домаћим и међународним такмичењима у програмирању (међу њима и Међународна информатичка олимпијада — IOI). Рећи да ме је велики број «цака» у вези са њима и навео да напишем овај матурски рад није толико далеко од истине.

Не може се порећи да су примене све присутније у захтевима корисника. Зато ову тему сматрам једном од најзначајнијих при разматрању примена алгоритама и структура података.

Желео бих да се захвалим свима који су ми омогућили да овај рад буде написан. Најпре бих издвојио своју менторку, професорку **Јелену Хаџи-Пурић**, због корисних савета о томе које теме рад треба да покрије, као и начин на који треба да буде написан. Захваљујем се и свим професорима информатичких предмета који су ми предавали у Математичкој гимназији на инспирацији и подршци: **Невенки Спалевић**, **Бранислави Бајковић-Лазаревић**, а посебно бих истакао и свог разредног старешину **Мијодрага Ђуришића**. Велики је и значај **проф. др Драгана Урошевића**, на чијим часовима додатне наставе (у оквиру припрема за ученичка такмичења) сам се први пут упознао са неким од ових бинарних стабала. На крају, ту је и велики утицај ИС Петница, где сам се, у оквиру семинара примењене физике и електронике, сусрео са врло интересантним применама неких од ових структура података.

Извори

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, The MIT Press, 2001.

[2] Udi Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, 1989.

[3] Душан Аднађевић, Зоран Каделбург, *Математичка анализа 2*, Математички факултет, Круг Београд, 2011.

[4] Jerome H. Friedman, Jon Louis Bentley, Raphael Ari Finkel, *An Algorithm for Finding Best Matches in Logarithmic Expected Time*, ACM Transactions on Mathematical Software (TOMS) Volume 3 Issue 3, 1977.

[5] Jon Louis Bentley, *Multidimensional binary search trees used for associative searching*, Communications of the ACM Volume 18 Issue 9, 1975.

[6] Ingo Wald, Vlastimil Havran, *On building fast kd-Trees for Ray Tracing, and on doing that in* $O(N \log N)$, Technica Report UUSCI-2006-009, 2006.

[7] Драган Урошевић, *Структуре података за ефикасно пребројавање над скуповима тачака*, Настава математике LVIII 1-2, Друштво математичара Србије, Београд, 2013.

[8] *Binary Indexed Trees*, `www.topcoder.com`

[9] *Range Minimum Query and Lowest Common Ancestor*, `www.topcoder.com`

[10] `www.cise.ufl.edu/ jlpaez/pages/slides.html`

[11] `www.csie.ntnu.edu.tw/~u91029/Sequence.html`

[12] `pointclouds.org`

[13] `vision.in.tum.de/data/datasets/rgbd-dataset`