Конструкција и анализа алгоритама 2

Материјал са предавања

Миодраг Живковић
е-маил: ezivkovm@matf.bg.ac.rs
УРЛ: www.matf.bg.ac.rs/~ezivkovm

Весна Маринковић
е-маил: vesnap@matf.bg.ac.rs
УРЛ: www.matf.bg.ac.rs/~vesnap
Математички факултет, Београд

Аутори:
проф. др Миодраг Живковић, редовни професор Математичког факултета у Београду
др Весна Маринковић, доцент Математичког факултета у Београду

КОНСТРУКЦИЈА И АНАЛИЗА АЛГОРИТАМА 2

Садржај

Глава 1

Геометријски алгоритми

1.1 Увод

Геометријски алгоритми играју важну улогу у многим областима, на пример у рачунарској графици, пројектовању помоћу рачунара, пројектовању VLSI (интегрисаних кола високе резолуције), роботици и базама података. У рачунарски генерисаној слици може бити на хиљаде или чак милионе тачака, линија, квадрата или кругова; пројектовање компјутерског чипа може да захтева рад са милионима елемената. Сви ови проблеми садрже обраду геометријских објеката. Пошто величина улаза за ове проблеме може бити врло велика, веома је значајно развити ефикасне алгоритме за њихово решавање.

У овом поглављу размотрићемо неколико основних геометријских алгоритама — оних који се могу користити као елементи за изградњу сложенијих алгоритама.

Објекти са којима се ради су тачке, праве, дужи и многоуглови. Алгоритми обрађују ове објекте, односно израчунавају неке њихове карактеристике. Најпре ћемо дати основне дефиниције и навести структуре података погодне за представљање појединих објеката. **Тачка** p у равни представља се као пар координата (x, y) (претпоставља се да је изабран фиксирани координатни систем). **Права** је представљена паром тачака p и q (произвољне две различите тачке на правој), и означаваћемо је са $-p-q-$. **Дуж** се такође задаје паром тачака p и q које представљају крајеве те дужи, и означаваћемо је са $p-q$. **Пут** P је низ тачака p_1, $p_2, \ldots, p_k$ и дужи $p_1 - p_2$, $p_2 - p_3, \ldots, p_{k-1} - p_k$ које их повезују. Дужи које чине пут су његове **ивице** (странице). **Затворени пут** је пут чија се последња тачка поклапа са првом. Затворени пут зове се и **многоугао**. Тачке које дефинишу многоугао су његова **темена**. На пример, троугао је многоугао са три темена. Многоугао се представља низом, а не скупом тачака, јер је битан редослед којим се тачке задају (променом редоследа тачака из истог скупа у општем случају добија се други многоугао). **Прости многоугао** је онај код кога одговарајући пут нема пресека са самим собом; другим речима, једине ивице које имају заједничке тачке су суседне ивице са њиховим заједничким теменом. Прости многоугао ограничава једну област у равни, **унутрашњост** многоугла. **Конвексни многоугао** је многоугао

чија унутрашњост са сваке две тачке које садржи, садржи и све тачке те дужи. **Конвексни пут** је пут од тачака $p_1, p_2, \ldots, p_k$ такав да је многоугао $p_1p_2 \ldots p_k$ конвексан.

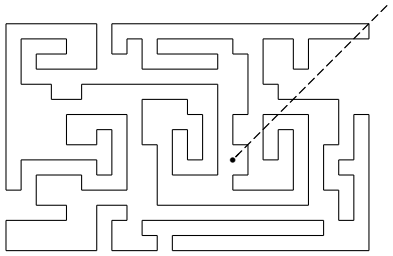
Претпоставља се да је читалац упознат са основама аналитичке геометрије. У алгоритмима које ћемо разматрати наилази се, на пример, на израчунавање пресечне тачке двеју дужи, утврђивање да ли тачка лежи са задате стране праве, израчунавање растојања између две тачке. Све ове операције могу се извести за време ограничено константом, помоћу основних аритметичких операција. При томе, на пример, претпостављамо да се квадратни корен може израчунати за константно време.

Честа неугодна карактеристика геометријских проблема је постојање многих "специјалних случајева". На пример, две праве у равни обично се секу у једној тачки, сем ако су паралелне или се поклапају. При решавању проблема са две праве, морају се предвидети сва три могућа случаја. Сложенији објекти проузрокују појаву много већег броја специјалних случајева, о којима треба водити рачуна. Обично се већина тих специјалних случајева непосредно решава, али потреба да се они узму у обзир чини понекад конструкцију геометријских алгоритама врло исцрпљујућом. Ми ћемо у овом поглављу често игнорисати детаље који нису од суштинског значаја за разумевање основних идеја алгоритама, али читаоцу саветујемо да размотри решавање сваког од специјалних случајева на који се при конструкцији алгоритма наиђе.

1.2 Утврђивање да ли задата тачка припада многоуглу

Разматрање започињемо једним једноставним проблемом.

Проблем. Задат је прост многоугао P и тачка q. Установити да ли је тачка у или ван многоугла P.



Слика 1.1: Утврђивање припадности тачке унутрашњости простог многоугла.

Проблем изгледа једноставно на први поглед, али ако се разматрају сложени неконвексни многоуглови, као онај на слици 1.1, проблем сигурно није једноставан. Први интуитивни приступ је покушати некако "изаћи напоље", полазећи од задате тачке. Посматрајмо произвољну полуправу са теменом q. Види се да је довољно пребројати пресеке са страницама многоугла, све до достизања спољашње области. У примеру на слици 1.1,

идући на североисток од дате тачке (пратећи испрекидану линију), наилазимо на шест пресека са многоуглом до достизања спољашње области (истина, већ после четири пресека стиже се ван многоугла, али то рачунар "не види"; зато се броји укупан број пресека полуправе са страницама многоугла). Пошто нас последњи пресек пре изласка изводи из многоугла, а претпоследњи нас враћа у многоугао, итд., тачка је ван многоугла. Уопште тачка је у многоуглу ако и само ако је број пресека непаран (специјалне случајеве на тренутак занемарујемо).

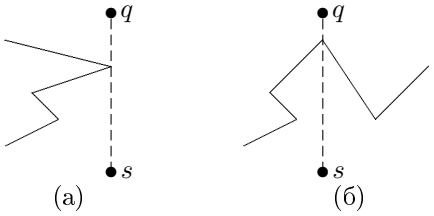
Алгоритам Tačka_u_mnogouglu_1(P, q); {први покушај}
Улаз: P (прост многоугао са теменима $p_1, p_2, \ldots, p_n$ и ивицама $e_1, e_2, \ldots, e_n$), и q (тачка).
Излаз: *Pripada* (Булова променљива, *true* акко q припада P).
begin
    Изабрати произвољну тачку s ван многоугла;
    Нека је L дуж $q - s$;
    $broj := 0$;
    for све гране e_i многугла **do**
        if e_i сече L **then**
            {претпостављамо да пресек није ни теме ни ивица, видети текст}
            $broj := broj + 1$;
    if $broj$ је непаран **then** $pripada := true$
    else $pripada := false$
end

Као што је речено у претходном одељку, обично постоје неки специјални случајеви које треба посебно размотрити. Нека је s тачка ван многоугла, и нека је L дуж која спаја q и s. Циљ је утврдити да ли q припада унутрашњости многоугла P на основу броја пресека дужи L са ивицама многоугла P. Међутим, дуж L може се делом преклапати са неким ивицама многоугла P. Ово преклапање очигледно не треба бројати у пресеке. Други специјални случај је када дуж L садржи неко теме p_i многоугла. На слици 1.2(а) види се пример случаја кад пресек L са теменом не треба бројати, а на слици 1.2(б) пример кад тај пресек треба бројати. Постоји више начина за утврђивање да ли неки овакав пресек треба бројати или не.

- Један начин је следећи: ако су два суседна темена за теме p_i са исте стране L, пресек се не рачуна. У противном ако су два суседна темена са различитих страна дужи L, пресек се броји.

- Други могући начин је следећи: с обзиром на то да разматрамо пресеке праве паралелне са y осом, за сваку ивицу многоугла рачунамо рецимо лево теме, а десно не (темена класификујемо као лево и десно у односу на вредности x координате). Тиме нећемо рачунати пресек налик оном на слици 1.2(а) јер је за обе ивице пресек кроз десно теме, а пресек као на слици 1.2(б) хоћемо једном, јер је за једну од ивица које се сустичу у том темену то лево теме, а за другу десно.

Развијајући овај алгоритам, имплицитно смо претпостављали да радимо са сликом. Проблем је нешто другачији кад је улаз дат низом координата,



Слика 1.2: Специјални случајеви кад права сече ивицу кроз теме.

што је уобичајено. На пример, кад посао радимо ручно и видимо многоугао, лако је наћи добар пут (онај са мало пресека) до неке тачке ван многоугла. У случају многоугла датог низом координата, то није лако. Највећи део времена троши се на израчунавање пресека. Тај део посла може се битно упростити ако је дуж $q - s$ паралелна једној од оса — на пример y-оси. Број пресека са овом специјалном дужи може бити много већи него са оптималном дужи (коју није лако одредити — читаоцу се оставља да размотри овај проблем), али је налажење пресека много једноставније (за константни фактор).

Алгоритам Tačka_u_mnogouglu_2(P, q); {други покушај}
Улаз: P (прости многоугао са теменима $p_1, p_2, \ldots, p_n$ и страницама $e_1, e_2, \ldots, e_n$), и q (тачка са координатама (x_0, y_0)).
Излаз: *Pripada* (булова променљива, *true* акко q припада P).
begin
    $Broj := 0$;
    for све гране e_i многоугла **do**
        if права $x = x_0$ сече e_i **then**
            {Претпостављамо да пресек није ни теме ни страница многоугла}
            Нека је y_i y-координата пресека праве $x = x_0$ са e_i;
            if $y_i < y_0$ **then** {пресек је испод q}
                $Broj := Broj + 1$;
    if *Broj* је непаран **then** $Pripada := true$
    else $Pripada := false$
end

Сложеност. За израчунавање пресека две дужи потребно је константно време. Нека је n број ивица многоугла. Кроз основну петљу алгоритма пролази се n пута. У сваком проласку налази се пресек две дужи и извршавају се још неке операције за константно време. Дакле, укупно време извршавања алгоритма је $O(n)$.

Коментар. У много случајева једноставан поступак инспирисан обичним алгоритмом (оним којим се ручно решава проблем) није ефикасан за велике улазе. Понекад је пак такав приступ не само једноставан, него и ефикасан. Приступ решавању проблема поступком који се визуелно намеће је обично добар избор. Тиме се може доћи до више корисних запажања о проблему. У овом случају, посматрајући слику, приметили смо да се проблем може решити пратећи неки пут од тачке до спољашњости многоугла.

Интересантан проблем јесте како наћи "оптималну" полуправу кроз дату тачку q, односно ону која сече најмањи број страница многоугла. За свако теме p_i израчунавамо угао који полуправа $q - p_i-$ заклапа са x осом и темена сортирамо растуће према овој вредности. За сваку страницу разликујемо почетно и крајње теме, при чему почетним теменом називамо оно теме коме одговара мањи угао полуправе. Најпре се одреди број b пресека полуправе кроз q и прво теме (у сортираном редоследу) са многоуглом (то се може урадити у времену $O(n)$, где је n број страница многоугла). Претпоставимо да је за i-то теме у сортираном редоследу, $i \geq 1$, одређен број пресека полуправе $q-p_i-$ са страницама многоугла. Приликом преласка на наредно теме p_{i+1} потребно је размотрити две странице кроз то теме и ажурирати вредност бројача b. Вредност бројача b се не мења ако су темена суседна темену r са различитих страна полуправе; бројач се смањује за 2 ако суседним теменима одговарају мањи углови од угла полуправе; бројач се повећава за 2 ако суседним теменима одговарају већи углови од угла полуправе. У сва три случаја је сложеност ажурирања бројача $O(1)$. Током проласка кроз сва темена памтимо најмању до сада добијену вредност бројача b и за који скуп полуправих се она добија (између која два темена треба да пролази дати зрак). С обзиром да је операција ажурирања константне временске сложености, а да то радимо за свако теме, односно $O(n)$ пута, укупна сложеност овог корака је $O(n)$. Укупна сложеност овог алгоритма је због почетног сортирања $O(n \log n)$.

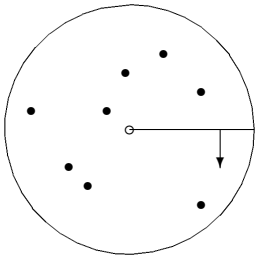
1.3 Конструкција простог многоугла

Скуп тачака у равни дефинише много различитих многоуглова, зависно од изабраног редоследа тачака. Размотрићемо сада проналажење простог многоугла са задатим скупом темена.

Проблем. Дато је n тачака у равни, таквих да нису све колинеарне. Повезати их простим многоуглом.

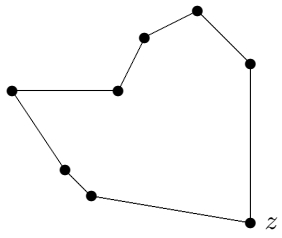
Постоји више метода за конструкцију траженог простог многоугла; уосталом, јасно је да у општем случају проблем нема једнозначно решење. Приказаћемо најпре геометријски приступ овом проблему. Нека је C неки круг, чија унутрашњост садржи све тачке. За налажење таквог круга довољно је $O(n)$ операција — израчунавања највећег међу растојањима произвољне тачке равни (центра круга) до свих n тачака. Површина C може се "пребрисати" (прегледати) ротирајућом полуправом којој је почетак центар C, видети слику 1.3. Претпоставимо за тренутак да ротирајућа полуправа у сваком тренутку садржи највише једну тачку. Очекујемо да ћемо спајањем тачака оним редом којим полуправа наилази на њих добити прост многоугао. Покушајмо да то докажемо. Означимо тачке, уређене у складу са редоследом наиласка полуправе на њих, са $p_1, p_2, \ldots, p_n$ (прва тачка бира се произвољно). За свако i, $1 \leq i \leq n$, страница $p_i - p_{i-1}$ (односно $p_1 - p_n$ за $i = 1$) садржана је у новом (дисјунктном) исечку круга, па се не сече ни са једном другом страном. Ако би ово тврђење било тачно, добијени многоугао би морао да буде прост. Међутим, угао између полуправих кроз неке две узастопне тачке p_i и p_{i+1} може да буде већи од π. Тада исечак који садржи дуж $p_i - p_{i+1}$ садржи више од пола круга и

није конвексна фигура, а дуж $p_i - p_{i+1}$ пролази кроз друге исечке круга, па може да сече друге стране многоугла. Да бисмо се уверили да је то могуће, довољно је да замислимо круг са центром ван круга са слике 1.3. Ово је добар пример специјалних случајева на које се наилази при решавању геометријских проблема. Потребно је да будемо пажљиви, да бисмо били сигурни да су сви случајеви размотрени. На ову појаву наилази се код свих врста алгоритама, али је она код геометријских алгоритама драстичније изражена.



Слика 1.3: Пролазак тачака у кругу ротирајућом полуправом.

Да би се решио уочени проблем, могу се, на пример, фиксирати произвољне три тачке из скупа, а за центар круга изабрати нека тачка унутар њима одређеног троугла (на пример тежиште, које се лако налази). Овакав избор гарантује да ни један од добијених сектора круга неће имати угао већи од π. Могуће је изабрати и друго решење, да се за центар круга узме једна од тачака из скупа — тачка z са највећом x-координатом (и са најмањом y-координатом, ако има више тачака са највећом x-координатом). Затим користимо исти основни алгоритам. Сортирамо тачке према положају у кругу са центром z. Прецизније, сортирају се *углови* између x-осе и полуправих од z ка осталим тачкама. Ако две или више тачака имају исти угао, оне се даље сортирају према растојању од тачке z. На крају, z се повезује са тачком са најмањим и највећим углом, а остале тачке повезују се у складу са добијеним уређењем, по две узастопне. Пошто све тачке леже лево од z, до дегенерисаног случаја о коме је било речи не може доћи. Прост многоугао добијен овим поступком за тачке са слике 1.3 приказан је на слици 1.4.



Слика 1.4: Конструкција простог многоугла.

Описани метод може се усавршити на два начина. Угао φ који права $y = mx + b$ заклапа са x осом добија се коришћењем везе $m = \operatorname{tg}\varphi$, односно из једначине $\varphi = \operatorname{arctg} m$. Међутим, углови се не морају експлицитно

израчунавати. Углови се користе само за налажење редоследа којим треба повезати тачке. Исти редослед добија се уређењем *нагиба* (односно односа прираштаја y- и x-координата) одговарајућих полуправих; то чини непотребним израчунавање аркустангенса. Из истог разлога непотребно је израчунавање растојања кад две тачке имају исти нагиб — довољно је израчунати квадрате растојања. Дакле, нема потребе за израчунавањем квадратних коренова.

Алгоритам Prost_mnogougao$(p_1, p_2, \ldots, p_n)$;
Улаз: $p_1, p_2, \ldots, p_n$ (тачке у равни).
Излаз: P (прост многоугао са теменима $p_1, p_2, \ldots, p_n$ у неком редоследу).
begin
  променити ознаке тако да p_1 буде екстремна тачка;
  {тачка са највећом x-координатом, а ако таквих има више,}
  {она од њих која има најмању y-координату}
  for $i := 2$ **to** n **do**
    израчунати угао α_i између праве $-p_1 - p_i-$ и x-осе;
  сортирати тачке према угловима $\alpha_2, \ldots, \alpha_n$;
  {у групи са истим углом сортирати их према растојању од p_1}
  P је многоугао дефинисан сортираном листом тачака
end

Сложеност. Основна компонента временске сложености овог алгоритма потиче од сортирања. Сложеност алгоритма је дакле $O(n \log n)$.

1.4 Конвексни омотач

Конвексни омотач скупа тачака дефинише се као најмањи конвексни многоугао који садржи све тачке скупа. Конвексни омотач се представља на исти начин као обичан многоугао, теменима наведеним у цикличком редоследу.

Проблем. Конструисати конвексни омотач задатих n тачака у равни.

Обрада конвексних многоуглова једноставнија је од обраде произвољних многоуглова. На пример, постоји алгоритам сложености $O(\log n)$ за утврђивање припадности тачке конвексном n-тоуглу.

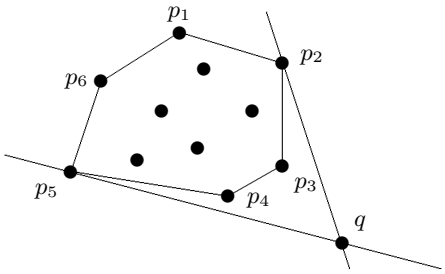
Претпоставимо да је задатак да утврдимо да ли се тачка q налази унутар датог конвексног многоугла $p_1 p_2 \ldots p_n$. Идеја на којој се заснива решење је следећа: ако знамо да се q налази унутар угла $\angle p_i p_1 p_j$, $i < j$, онда се после провере да ли се q и p_j налазе са исте стране праве $p_1 p_m$, $m = \lfloor (i+j)/2 \rfloor$, зна да ли је q унутар угла $\angle p_m p_1 p_j$ (ако је одговор "да", или ако је q на правој $p_1 p_m$), или унутар угла $\angle p_i p_1 p_m$ (у противном). На почетку се проверава да ли је q унутар угла $\angle p_2 p_1 p_n$ (ако није, онда q не припада многоуглу). После највише $O(\log n)$ корака закључује се или да q не припада многоуглу, или да је унутар неког угла $\angle p_i p_1 p_{i+1}$; у другом случају је q у многоуглу акко припада троуглу $\triangle p_i p_1 p_{i+1}$.

Темена конвексног омотача су неке од тачака из задатог скупа. Кажемо да тачка *припада* омотачу ако је теме омотача. Ковексни омотач може се састојати од најмање три, а највише n тачака. Конвексни омотачи имају

широку примену, па су због тога развијени многобројни алгоритми за њихову конструкцију.

1.4.1 Директни приступ

Као и обично, покушаћемо најпре са директним индуктивним приступом. Конвексни омотач за три тачке лако је наћи. Претпоставимо да умемо да конструишемо конвексни омотач скупа од $< n$ тачака, и покушајмо да конструишемо конвексни омотач скупа од n тачака. Како n-та тачка може да промени конвексни омотач за првих $n-1$ тачака? Постоје два могућа случаја: или је нова тачка у претходном конвексном омотачу (тада он остаје непромењен), или је она ван њега, па се омотач "шири" да обухвати и нову тачку, видети слику 1.5. Потребно је дакле решити два потпроблема: утврђивање да ли је нова тачка унутар омотача и проширивање омотача новом тачком. Они нису једноставни. Ствар се може упростити погодним избором n-те тачке. Звучи изазовно покушати са избором тачке унутар омотача; то, међутим, није увек могуће јер у неким случајевима све тачке припадају омотачу. Друга могућност, која се показала успешном у претходном проблему, је избор екстремне тачке за n-ту тачку.



Слика 1.5: Проширивање конвексног омотача новом тачком.

Изаберимо поново тачку са највећом x-координатом (и минималном y-координатом ако има више тачака са највећом x-координатом). Нека је то тачка q. Јасно је да тачка q мора бити теме конвексног омотача. Питање је како променити конвексни омотач осталих $n-1$ тачака тако да обухвати и q. Потребно је најпре пронаћи темена старог омотача која су у унутрашњости новог омотача (p_3 и p_4 на слици 1.5) и уклонити их; затим се умеће ново теме q између два постојећа (p_2 и p_5 на слици 1.5). **Права ослонца** конвексног многоугла је права која многоугао сече у тачно једној тачки. Многоугао увек цео лежи са једне стране своје праве ослонца. Посматрајмо сада праве ослонца $-q-p_2-$ и $-q-p_5-$ на слици 1.5. Обично само два темена многоугла повезивањем са q одређују праве ослонца (игнорисаћемо специјални случај кад два или више темена многоугла леже на истој правој са q). Многоугао лежи између две праве ослонца, што указује како треба извести модификацију омотача. Праве ослонца заклапају минимални и максимални угао са x-осом међу свим правим кроз q и неко теме многоугла. Да бисмо одредили та два темена, треба да размотримо праве из q ка свим осталим теменима, да израчунамо углове које оне заклапају са x-осом, и међу тим угловима изаберемо минималан и максималан. После

проналажења два екстремна темена p_i, p_j лако је конструисати модификовани омотач. Потребно је елиминисати један од два добијена сегмената низа темена. Посматрају се суседна темена нпр. темена p_i. Једно од њих је са исте стране праве $-p_i - p_j-$ као и тачка q; то теме припада сегменту низа темена које треба заменити новим страницама $p_i - q$ и $q - p_j$. У примеру на слици 1.5 теме p_4, суседно екстремном темену p_5, налази се са исте стране дијагонале $p_5 - p_2$, па се сегмент старог омотача p_3, p_4 замењује са две нове странице $p_5 - q$ и $p_2 - q$.

Сложеност. За сваку тачку треба израчунати углове правих ка свим претходним теменима и x-осе, пронаћи минимални и максимални угао, додати нови чвор и избацити неке чворове. Сложеност додавања k-те тачке је дакле $O(k)$, а видели смо већ да је решење диференцне једначине $T(n) = T(n-1)+cn$ облика $O(n^2)$. Према томе, сложеност овог алгоритма је $O(n^2)$. Алгоритам на почетку такође захтева и сортирање, али је време сортирања асимптотски мање од времена потребног за остале операције.

1.4.2 Увијање поклона

Како се може побољшати описани алгоритам? Кад проширујемо многоугао теме по теме, доста времена трошимо на формирање конвексних многоуглова од тачака које могу бити унутрашње за коначни конвексни омотач. Може ли се то избећи? Уместо да правимо омотаче подскупова датог скупа тачака, можемо да посматрамо комплетан скуп тачака и да директно правимо његов конвексни омотач. Може се, као и у претходном алгоритму, кренути од екстремне тачке (која увек припада омотачу), пронаћи њој суседна темена омотача налазећи праве ослонца, и наставити на исти начин од тих суседа. Овај алгоритам из разумљивих разлога зове се **увијање поклона**.Полази се од једног темена "поклона", и онда се он увија у конвексни омотач проналазећи теме по теме омотача. Алгоритам је приказан на слици **??**. Алгоритам се може преправити тако да ради и у просторима веће димензије.

Алгоритам увијање поклона је директна последица примене следеће индуктивне хипотезе (по k):

Индуктивна хипотеза. За задати скуп од n тачака у равни, умемо да пронађемо конвексни пут дужине $k < n$ који је део конвексног омотача скупа.

Код ове хипотезе нагласак је на проширивању *пута*, а не омотача. Уместо да проналазимо конвексне омотаче мањих скупова, ми проналазимо део коначног конвексног омотача.

Сложеност. Да бисмо додали k-то теме омотачу, треба да пронађемо праву са најмањим углом у скупу од $n - k$ правих. Због тога је временска сложеност алгоритма увијање поклона $O(n^2)$, односно ако коначни омотач има m тачака, сложеност је $O(mn)$, што асимптотски није боље од алгоритма заснованог на проширивању омотача.

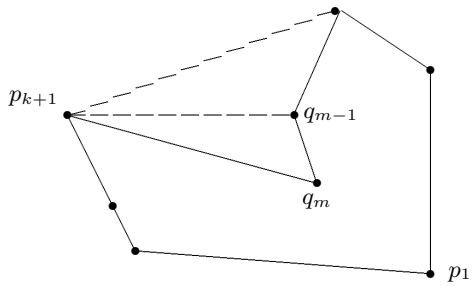
Алгоритам Uvijanje_poklona$(p_1, p_2, \ldots, p_n)$;
Улаз: $p_1, p_2, \ldots, p_n$ (скуп тачака у равни).
Излаз: P (конвексни омотач тачака $p_1, p_2, \ldots, p_n$).
begin
Нека је P празан скуп;
Нека је p тачка са највећом x-координатом
(и са најмањом y-координатом, ако има више
тачка са највећом x-координатом);
Укључи p у скуп P;
Нека је L права кроз p паралелна са x-осом;
while омотач P није завршен **do**
Нека је q тачка за коју је најмањи угао између L и $-p-q-$;
Укључи q у скуп P;
$L :=$ права $-p-q-$;
$p := q$
end

1.4.3 Грејемов алгоритам

Сада ћемо размотрити алгоритам за налажење конвексног омотача сложености $O(n \log n)$. Започиње се сортирањем тачака према угловима, слично као при конструкцији простог многоугла у одељку 1.3. Нека је p_1 тачка са највећом x-координатом (и са најмањом y-координатом, ако има више тачака са највећом x-координатом). За сваку тачку p_i израчунавамо угао између праве $-p_1 - p_i-$ и x-осе, и сортирамо тачке према величини ових углова, видети слику 1.6. Тачке пролазимо редоследом којим се појављују у (простом) многоуглу, и покушавамо да идентификујемо темена конвексног омотача. Као и код алгоритма увијање поклона памтимо пут састављен од дела прођених тачака. Прецизније, то је конвексни пут чији конвексни многоугао садржи све до сада прегледане тачке (одговарајући конвексни многоугао добија се повезивањем прве и последње тачке пута). Због тога, у тренутку кад су све тачке прегледане, конвексни омотач скупа тачака је конструисан. Основна разлика између овог алгоритма и увијања поклона је у чињеници да текући конвексни пут не мора да буде део коначног конвексног омотача. То је само део конвексног омотача до сада прегледаних тачака. Пут може да садржи тачке које не припадају коначном конвексном омотачу; те тачке биће елиминисане касније. На пример, пут од p_1 до q_m на слици 1.6 је конвексан, али q_m и q_{m-1} очигледно не припадају конвексном омотачу. Ово разматрање сугерише алгоритам заснован на следећој индуктивној хипотези.

Индуктивна хипотеза. Ако је дато n тачака у равни, уређених према алгоритму *Prost_mnogougao* (одељак 1.3), онда умемо да конструишемо конвексни пут преко неких од првих k тачака, такав да одговарајући конвексни многоугао обухвата првих k тачака.

Случај $k = 1$ је тривијалан. Означимо конвексни пут добијен (индуктивно) за првих k тачака са $P = q_1, q_2, \ldots, q_m$. Сада треба да проширимо индуктивну хипотезу на $k+1$ тачака. Посматрајмо угао између правих $-q_{m-1} - q_m-$ и



Слика 1.6: Грејемов алгоритам за налажење конвексног омотача скупа тачака.

$-q_m-p_{k+1}-$ (видети слику 1.6). Ако је тај угао мањи од π (при чему се угао мери из *унутрашњости* многоугла), онда се p_{k+1} може додати постојећем путу (нови пут је због тога такође конвексан), чиме је корак индукције завршен. У противном, тврдимо да q_m лежи у многоуглу добијеном заменом q_m у P тачком p_{k+1}, и повезивањем p_{k+1} са p_1. Ово је тачно јер су тачке уређене преме одговарајућим угловима. Права $-p_1 - p_{k+1}-$ лежи "лево" од првих k тачака. Због тога q_m јесте унутар горе дефинисаног многоугла, може се избацити из P, а p_{k+1} се може додати. Да ли је тиме све урађено? Не сасвим. Иако се q_m може елиминисати, модификовани пут не мора увек да буде конвексан. Заиста, слика 1.6 јасно показује да постоје случајеви кад треба елиминисати још тачака. На пример, тачка q_{m-1} може да буде унутар многоугла дефинисаног модификованим путем. Морамо да (уназад) наставимо са проверама последње две гране пута, све док угао између њих не постане мањи од π. Пут је тада конвексан, а хипотеза је проширена на $k+1$ тачку.

Алгоритам Grahamov_algoritam$(p_1, p_2, \ldots, p_n)$;
Улаз: $p_1, p_2, \ldots, p_n$ (скуп тачака у равни).
Излаз: $q_1, q_2, \ldots, q_m$ (конвексни омотач тачака $p_1, p_2, \ldots, p_n$).
begin
  Нека је p_1 тачка са највећом x-координатом (а најмањом y-координатом,
    ако има више тачака са највећом x-координатом);
  Помоћу алгоритма *Prost_mnogougao* уредити тачке у односу на p_1;
    нека је то редослед $p_1, p_2, \ldots, p_n$;
  $q_1 := p_1$;
  $q_2 := p_2$;
  $q_3 := p_3$; {пут P се на почетку састоји од p_1, p_2 и p_3}
  $m := 3$;
  for $k := 4$ **to** n **do**
    while угао између $-q_{m-1}-q_m-$ и $-q_m-p_k-) \geq \pi$ **do**
      $m := m - 1$;
    $m := m + 1$;
    $q_m := p_k$ {на слици 1.6 то је p_{k+1}}
end

Сложеност. Главни део сложености алгоритма потиче од почетног сор-

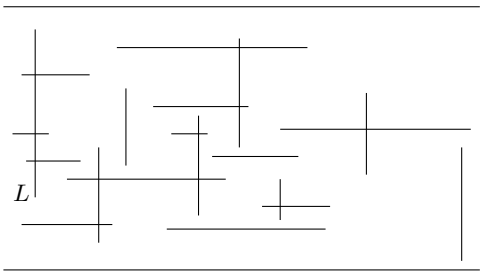
тирања. Остатак алгоритма извршава се за време $O(n)$. Свака тачка скупа разматра се тачно једном у индуктивном кораку као p_{k+1}. У том тренутку тачка се увек додаје конвексном путу. Иста тачка биће разматрана и касније (можда чак и више него једном) да би се проверила њена припадност конвексном путу. Број тачака на које се примењује овакав повратни тест може бити велики, али се све оне, сем две (текућа тачка и тачка за коју се испоставља да даље припада конвексном путу) елиминишу, јер тачка може бити елиминисана само једном! Према томе, троши се највише константно време за елиминацију сваке тачке и константно време за њено додавање. Укупно је за ову фазу потребно $O(n)$ корака. Због сортирања је време извршавања комплетног алгоритма $O(n \log n)$.

1.5 Пресеци хоризонталних и вертикалних дужи

Често се наилази на проблеме налажења пресека. Понекад је потребно израчунавање пресека више објеката, а понекад само треба открити да ли је пресек непразан скуп. Проблеми детекције су обично лакши. У овом одељку приказаћемо један проблем налажења пресека, који илуструје важну технику за решавање геометријских проблема. Иста техника може се применити и на друге проблеме.

Проблем. За задати скуп од n хоризонталних и m вертикалних дужи пронаћи све њихове пресеке.

Овај проблем важан је, на пример, при пројектовању кола VLSI (интегрисаних кола са огромним бројем елемената). Коло може да садржи на хиљаде "жичица", а пројектант треба да буде сигуран да не постоје неочекивани пресеци. На проблем се такође наилази при елиминацији скривених линија када је потребно закључити које ивице или делови ивица су скривени самим објектом или неким другим објектом; тај проблем је обично компликованији, јер се не ради само о хоризонталним и вертикалним линијама. Пример проблема приказан је на слици 1.7.



Слика 1.7: Пресеци хоризонталних и вертикалних дужи.

Налажење свих пресека вертикалних дужи је једноставан проблем, који се оставља читаоцу за вежбање (исто се односи и на хоризонталне дужи). Претпоставимо због једноставности да не постоје пресеци између произвољне две хоризонталне, односно произвољне две вертикалне дужи. Ако покушамо да проблем решимо уклањањем једне по једне дужи (било хоризонталне, било вертикалне), онда ће бити неопходно да се пронађу пресеци уклоњене

дужи са свим осталим дужима, па се добија алгоритам са $O(mn)$ налажења пресека дужи. У општем случају број пресека може да буде $O(mn)$, па алгоритам може да утроши време $O(mn)$ већ само за приказивање свих пресека. Међутим, број пресека може да буде много мањи од mn. Волели бисмо да конструишемо алгоритам који ради добро кад има мало пресека, а не превише лоше ако пресека има много. Дакле циљ нам је да проблем решимо коришћењем алгоритма "осетљивог на излаз", чија сложеност зависи и од величине улаза и од величине излаза. То се може постићи комбиновањем двеју идеја: избора специјалног редоследа индукције и појачавања индуктивне хипотезе.

Редослед примене индукције може се одредити покретном правом (енг. sweeping line) која прелази ("скенира") раван слева удесно; дужи се разматрају оним редом којим на њих наилази покретна права. Поред налажења пресечних тачака, треба чувати и неке податке о дужима које је покретна права већ захватила. Ти подаци биће корисни за ефикасније налажење наредних пресека. Ова техника зове се **техника покретне линије**.

Замислимо вертикалну праву која прелази раван слева удесно. Да бисмо остварили ефекат пребрисавања, сортирамо све крајеве дужи према њиховим x-координатама. Две крајње тачке вертикалне дужи имају исте x-координате, па је довољна једна од њих. За хоризонталне дужи морају се користити оба краја. После сортирања крајева, дужи се разматрају једна по једна утврђеним редоследом. Као и обично при индуктивном приступу, претпостављамо да смо пронашли пресечне тачке претходних дужи, и да смо обезбедили неке допунске информације, па сада покушавамо да обрадимо следећу дуж и да унесемо неопходне допуне информација. Према томе структура алгоритма је у основи следећа. Разматрамо крајеве дужи један по један, слева удесно. Користимо информације прикупљене до сада (нисмо их још специфицирали) да обрадимо крај дужи, пронађемо пресеке у којима она учествује, и допуњујемо информације да бисмо их користили при следећем наиласку на неки крај дужи. Основни део посла је дефинисање информација које треба прикупљати. Покушајмо да покренемо алгоритам да бисмо открили које су то информације потребне.

Природно је у индуктивну хипотезу укључити познавање свих пресечних тачака дужи које се налазе лево од тренутног положаја покретне праве. Да ли је боље проверавати пресеке кад се разматра хоризонтална или вертикална дуж? Кад разматрамо вертикалну дуж, хоризонталне дужи које је могу сећи још увек се разматрају (пошто није достигнут њихов десни крај). С друге стране, кад посматрамо било леви, било десни крај хоризонталне дужи, ми или нисмо наишли на вертикалне дужи које је секу, или смо их већ заборавили. Дакле, боље је пресеке бројати приликом наиласка на вертикалну дуж. Претпоставимо да је покретна права тренутно преклопила вертикалну дуж L (видети слику 1.7). Какве информације су потребне да се пронађу сви пресеци дужи L? Пошто се претпоставља да су сви пресеци лево од тренутног положаја покретне праве већ познати, нема потребе разматрати хоризонталну дуж ако је њен десни крај лево од покретне праве. Према томе, треба разматрати само оне хоризонталне дужи чији су леви крајеви лево, а десни крајеви десно од тренутног положаја покретне праве (на слици 1.7 таквих дужи има шест). Потребно је чувати листу таквих хоризонталних дужи. Кад се наиђе на вертикалну дуж L, потребно је проверити да ли се она сече са тим хоризонталним дужима.

Важно је приметити да за налажење пресека са L овде x-координате крајева дужи нису од значаја. Ми већ знамо да хоризонталне дужи из листе имају x-координате које "покривају" x-координату дужи L. Потребно је проверити само y-координате хоризонталних дужи из листе, да би се проверило да ли су оне обухваћене опсегом y-координата дужи L. Сада смо спремни да формулишемо индуктивну хипотезу.

Индуктивна хипотеза. Нека је задата листа од k сортираних x-координата крајева дужи као што је описано, при чему је x_k највећа од x-координата. Умемо да пронађемо све пресеке дужи који су лево од x_k, и да елиминишемо све хоризонталне дужи које су лево од x_k.

За хоризонталне дужи које се још увек разматрају рећи ћемо да су **кандидати** (то су хоризонталне дужи чији су леви крајеви лево, а десни крајеви десно од текућег положаја покретне праве). Формираћемо и одржавати структуру података која садржи скуп кандидата. Одложићемо за тренутак анализу реализације ове структуре података.

Базни случај за наведену индуктивну хипотезу је једноставан. Да бисмо је проширили, потребно је да обрадимо $(k+1)$-и крај дужи. Постоје три могућности.

1. $(k+1)$-и крај дужи је десни крај хоризонталне дужи; дуж се тада просто елиминише из списка кандидата. Као што је речено, пресеци се проналазе при разматрању вертикалних дужи, па се ни један од пресека не губи елиминацијом хоризонталне дужи. Овај корак дакле проширује индуктивну хипотезу.

2. $(k+1)$-и крај дужи је леви крај хоризонталне дужи; дуж се тада додаје у списак кандидата. Пошто десни крај дужи није достигнут, дуж се не сме још елиминисати, па је и у овом случају индуктивна хипотеза проширена на исправан начин.

3. $(k+1)$-и крај дужи је вертикална дуж. Ово је основни део алгоритма. Пресеци са овом вертикалном дужи могу се пронаћи упоређивањем y-координата свих хоризонталних дужи из скупа кандидата са y-координатама крајева вертикалне дужи.

Алгоритам је сада комплетан. Број упоређивања ће обично бити много мањи од mn. На жалост, у најгорем случају овај алгоритам ипак захтева $O(mn)$ упоређивања, чак иако је број пресека мали. Ако се, на пример, све хоризонталне дужи простиру слева удесно ("целом ширином"), онда се мора проверити пресек сваке вертикалне дужи са свим хоризонталним дужима, што имплицира сложеност $O(mn)$. Овај најгори случај појављује се чак и ако ниједна вертикална дуж не сече ниједну хоризонталну дуж.

Да би се усавршио алгоритам, потребно је смањити број упоређивања y-координата вертикалне дужи са y-координатама хоризонталних дужи у скупу кандидата. Нека су y-координате дужи која се тренутно разматра y_D и y_G, и нека су y-координате хоризонталних дужи из скупа кандидата $y_1, y_2, \ldots, y_r$. Претпоставимо да су хоризонталне дужи у скупу кандидата задате сортирано према y-координатама (односно низ $y_1, y_2, \ldots, y_r$ је растући). Хоризонталне дужи које се секу са вертикалном могу се пронаћи извођењем две бинарне претраге, једне за y_D, а друге за y_G. Нека је $y_i < y_D \leq y_{i+1} \leq$

$y_j \leq y_G < y_{j+1}$. Вертикалну дуж секу хоризонталне са координатама y_{i+1}, y_{i+2},... y_j, и само оне. Може се такође извршити једна бинарна претрага за нпр. y_D, а затим пролазити y-координате док се не дође до y_j. Иако је полазни проблем дводимензионалан, налажење y_{i+1}, ... y_j је једнодимензионални проблем. Тражење бројева у једнодимензионалном опсегу (у овом случају од y_D до y_G) зове се *једнодимензионална претрага опсега*. Ако су бројеви сортирани, онда је време извршења једнодимензионалне претраге опсега пропорционално збиру времена тражења првог пресека и броја пронађених пресека. Наравно, не можемо да приуштимо себи сортирање хоризонталних дужи при сваком наиласку на вертикалну дуж.

Присетимо се још једном захтева. Потребна је структура података погодна за чување кандидата, која дозвољава уметање новог елемента, брисање елемента и ефикасно извршење једнодимензионалне претраге опсега. На срећу, постоји више структура података — на пример, уравнотежено стабло — која омогућују извршење уметања, брисања и тражења сложености $O(\log n)$ по операцији (где је n број елемената у скупу), и линеарно претраживање за време пропорционално броју пронађених елемената. Претрага започиње тражењем y_D и y_G у стаблу које садржи кандидате $y_1, \ldots, y_r$. Нека су путеви који се при томе прелазе полазећи од корена стабла означени са P_D, P_G, и нека је v последњи заједнички чвор за та два пута. Резултат претраге су кључеви комплетних десних подстабала чворова са пута P_D (испод v) у којима пут наставља левом граном, и (симетрично) комплетних левих подстабала чворова са пута P_G (испод v) у којима пут наставља десном граном. Овим су обухваћени сви чворови у унутрашњим подстаблима која су ограђена путевима P_D и P_G. Током обиласка, такође се проверавају и пријављују сви чворови на путевима P_D и P_G који леже у интервалу.

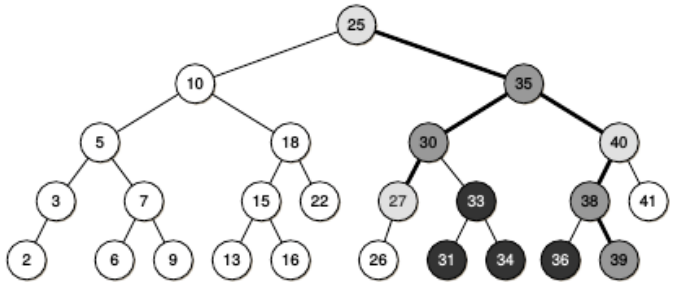
Алгоритам BSP_pretraga_opsega(v, x', x'')
Улаз. v (показивач на корен БСП), $[x', x'']$ (интервал)
Излаз. F (скуп тачака из v које припадају интервалу)

```
1  if v ≠ nil:
2      if x' ≤ v.kljuc:
3          BSP_pretraga_opsega(v.levi, x', x'')
4      if x' ≤ v.kljuc and x'' ≥ v.kljuc:
5          додај v.kljuc у F
6      if x'' > v.kljuc:
7          BSP_pretraga_opsega(v.desni, x', x'')
```

Сложеност. Почетно сортирање према x-координатама крајева дужи захтева $O((m+n)\log(m+n))$ корака. Пошто свако уметање и брисање захтева $O(\log n)$ корака, укупно време за обраду хоризонталних дужи је $O(n \log n)$. Обрада вертикалних дужи захтева једнодимензионалну претрагу опсега, која се може извршити за време $O(\log n + r)$, где је r број пресека ове вертикалне дужи. Временска сложеност алгоритма је дакле

$$O((m+n)\log(m+n) + R),$$

где је R укупан број пресека.



Слика 1.8: Пример једнодимензионалне претраге опсега. Кандидати, њих 25, уписани су у уравнотежено стабло, а исписују се кандидати из интервала [30,39]. Резултат чине тамно сиви и црни чворови.

Алгоритам Preseci$((v_1, v_2, \ldots, v_m), (h_1, h_2, \ldots, h_n))$;
Улаз: $v_1, v_2, \ldots, v_m$ (скуп вертикалних дужи), и$h_1, h_2, \ldots, h_n$ (скуп хоризонталних дужи).
$\{y_G(v_i)$ и $y_G(v_i)$ су мања и већа y-координата дужи $v_i\}$
Излаз: Скуп свих парова дужи које имају пресек.
begin
Сортирати све x-координате у неопадајући низ и сместити их у Q;
$V := \emptyset$;
{V је скуп хоризонталних дужи, тренутних кандидата за пресеке; организован}
{је као уравнотежено стабло према y-координатама хоризонталних дужи}
while Q је непразан **do**
уклонити из Q наредни крај дужи, p;
if p је десни крај дужи h_k **then**
уклонити h_k из V
else if p је леви крај дужи h_k **then**
укључити h_k у V
else if p је x-координата вертикалне дужи v_i **then**
{налажење пресека кандидата са вертикалном дужи}
једнодимензионална претрага опсега од $y_G(v_i)$ до $y_G(v_i)$ у V
end

1.6 Одређивање две најудаљеније тачке у равни

Често је, поред две најближе тачке, важно одредити и две најудаљеније тачке у неком скупу тачака. Нека је S скуп од n тачака у равни и нека је задатак одредити дијаметар $D(S)$ скупа тачака S који се дефинише као највеће растојање неке две тачке из S:

$$D(S) = \max\{d(p,q) \mid p,q \in S\}$$

Притом желимо да утврдимо и које су то две тачке a и b за које важи да је $d(a,b) = D(S)$ (можемо да разликујемо две варијанте овог проблема: наћи било који пар таквих тачака или све парове који задовољавају овај услов). Директан начин да решимо овај проблем састојао би се од рачунања растојања $d(p,q)$ за сваки пар тачака (p,q) и одређивања највеће од тих вредности. С обзиром на то да се рачунање растојања две тачке може урадити у константном времену и да је број парова тачака једнак $\binom{n}{2}$, овај алгоритам је сложености $O(n^2)$. Показаћемо у наставку текста да се овај проблем може решити и алгоритмом сложености $O(n \log n)$.

Лема 1.1. Дијаметар скупа тачака S једнак је дијаметру скупа темена његовог конвексног омотача S'.

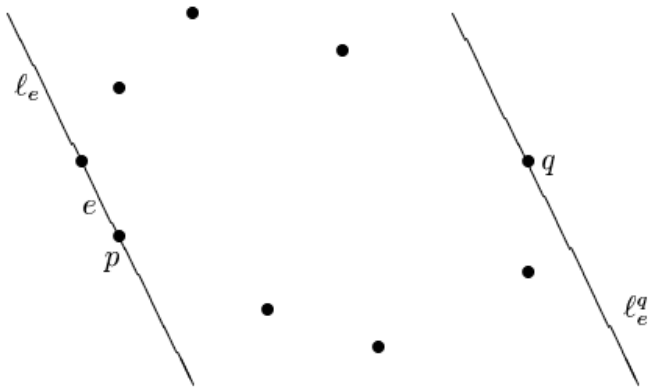
Доказ. Нека су p и q две различите тачке скупа S тако да нису обе у S'. Без смањења општости претпоставимо да $q \notin S'$. Покажимо да онда важи да постоји тачка r у S' тако да је $d(p,q) < d(p,r)$. Без нарушавања општости, претпоставимо да је права кроз тачке p и q хоризонтална и да је p лево од q. С обзиром на то да тачка q не припада конвексном омотачу, мора да постоји теме конвексног омотача r које је на вертикалној правој кроз q или десно од ње. Неједнакост $d(p,q) < d(p,r)$ следи из чињеница да је у троуглу (p,q,r) угао код темена q прав или туп.

Дакле, да бисмо израчунали дијаметар скупа тачака у равни, довољно је размотрити само темена која припадају његовом конвексном омотачу. Другим речима, без смањења општости може се претпоставити да су тачке из скупа S темена конвексног многоугла, при чему су сви углови конвексног омотача мањи од π (ако је угао многоугла у неком темену једнак π, онда то теме не припада конвексном омотачу).

Лема 1.2. Нека је l права кроз тачке p_1 и p_2. Низ растојања преосталих тачака до праве l: $d(p_3,l), d(p_4,l), \ldots, d(p_n,l)$ расте до неке вредности $d(p_i,l)$, а затим опада.

Доказ. Ово тврђење следи директно из чињенице да су тачке скупа S темена конвексног многоугла.

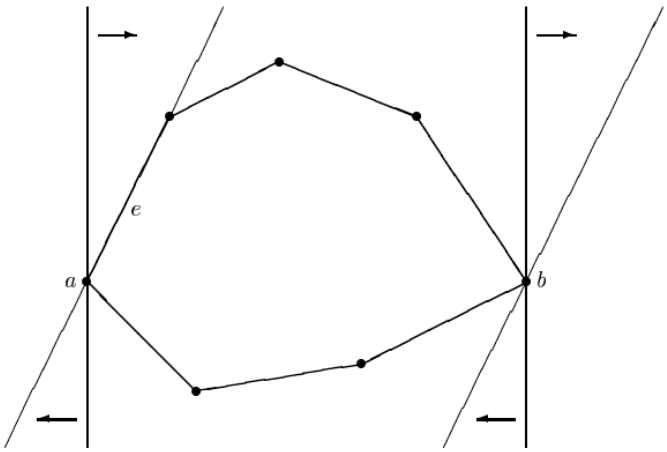
Ако је e ивица конвексног омотача, означимо са l_e праву која садржи e. Нека је L скуп свих парова (p,q) за које постоји ивица e конвексног омотача чија је једна крајња тачка p, тако да важи да је: $d(q,l_e) = \max\{d(r,l_e) | r \in S\}$. У примеру на слици 1.9 тачка q је на максималном растојању од праве l_e. Ако са l_e^q означимо праву кроз q која је паралелна са l_e, тада су све тачке из S између l_e и l_e^q (због конвексности, права l_e^q може да садржи највише још једну тачку скупа S). Стога је пар (p,q) садржан у скупу L.



Слика 1.9: Илустрација дефиниције скупа L. Ивица e има p као крајњу тачку и q је најудаљенија од l_e. Стога скуп L садржи пар (p, q)

Лема 1.3. Дијаметар скупа S једнак је највећем растојању $d(p, q)$ за неки пар из скупа L.

Доказ. Нека су a и b две тачке из S такве да је $d(a, b) = D(S)$. Покажимо да је пар (a, b) или пар (b, a) садржан у L, чиме ће лема бити доказана.



Слика 1.10: Илустрација доказа леме 3. Права кроз a је прва која наилази на ивицу конвексног омотача.

Без нарушавања општости, претпоставимо да је права кроз a и b хоризонтална. Све тачке из $S \setminus \{a, b\}$ су строго између две вертикалне праве кроз a и b (иначе би дијаметар скупа S био већи од $d(a, b)$). Ротирајмо ове две праве симултано у смеру казаље на часовнику око a и b и станимо чим се једна од њих поклопи са страницом конвексног омотача скупа S; приликом ротације ове две праве остају паралелне (слика 1.10). Нека је

e ивица са којом се дешава прво преклапање. Ако је a крајња тачка ове ивице, није тешко закључити да b има највеће растојање до праве кроз e. Стога је пар (a, b) садржан у скупу L. Слично, уколико је b крајња тачка ивице e, онда је пар (b, a) садржан у L.

На основу претходне леме закључујемо да дијаметар можемо израчунати на следећи начин: најпре одредимо скуп L, а затим одредимо и сва растојања одређена елементима из L и пронађемо највеће. Покажимо да скуп L садржи највише $4n$ елемената, при чему је са n означен број ивица конвексног многоугла S. Уочимо једну ивицу e конвексног омотача. Она је ограничена двема тачкама. Постоје највише две тачке скупа S на максималном растојању од праве l_e кроз e. Последица ове чињенице је да ова ивица може да изнедри највише четири пара у L из чега следи да је величина скупа L највише $4n$. Стога, закључујемо да је други корак алгоритма сложености $O(n)$. Остаје проблем одређивања скупа L. Показаћемо да се он може израчунати у времену $O(n)$.

Подсетимо се да смо тачке скупа S нумерисали као $p_1, p_2, \ldots, p_n$ у смеру супротном од кретања казаљке на часовнику. Нека је e ивица ограничена тачкама p_1 и p_2 и нека је l_e права која садржи e. Желимо да пронађемо највише две тачке скупа S које су на максималном растојању од l_e. Лема 1.2 нам даје једноставан поступак за одређивање ових тачака:

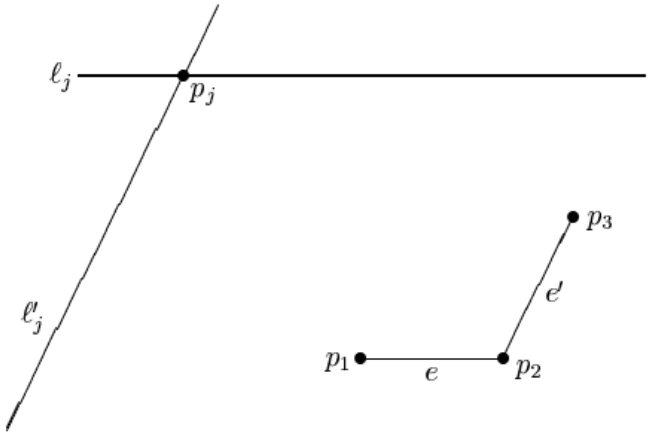
$j \leftarrow 3$
while $d(p_{j+1}, l_e) > d(p_j, l_e)$ **do**
  $j \leftarrow j + 1$

Притом се подразумева да се индекси тачака рачунају по модулу n, тј. да важи $p_{n+1} = p_1$. На крају ове while петље имамо да важи $d(p_3, l_e) < d(p_4, l_e) < \ldots < d(p_j, l_e)$ и $d(p_{j+1}, l_e) \leq d(p_j, l_e)$. Ако је $d(p_{j+1}, l_e) < d(p_j, l_e)$ онда је p_j једина тачка са максималним растојањем до l_e и додајемо два пара (p_1, p_j) и (p_2, p_j) у L. Иначе, имамо $d(p_{j+1}, l_e) = d(p_j, l_e)$ и додајемо четири пара (p_1, p_j), $(p_1, p_{j+1}), (p_2, p_j), (p_2, p_{j+1})$ у L. На овај начин смо обрадили прву ивицу e. Укупно утрошено време је $O(n)$, јер индекс j може бити величине n.

Нека је e' наредна ивица конвексног омотача, која има крајње тачке p_2 и p_3. Размотримо како можемо да ефикасно пронађемо највише две тачке скупа S која су на максималном растојању од $l_{e'}$.

Лема 1.4. Нека је индекс j такав да важи да је p_j на највећем растојању од праве l_e. Слично, нека је индекс i такав да је тачка p_i на највећем растојању од l'_e. Тада је $j \leq i \leq n$ или $i = 1$.

Доказ. Претпоставимо без нарушавања општости да је права кроз p_1 и p_2 хоризонтална. Тада је p_1 лево од p_2. Нека је l_j (аналогно l'_j) права кроз p_j паралелна са l_e (односно l'_e), видети слику 1.11. С обзиром на то да је p_j на највећем растојању од l_e, знамо да су тачке $p_4, p_5, \ldots, p_{j-1}$ све на правој l_j или испод ње. Због конвексности ове тачке морају да буду десно од праве l'_j (показати да је ово тачно и ако је угао у p_2 мањи од $\pi/2$). Стога важи да је $d(p_j, l'_e) > d(p_k, l'_e), k = 4, 5, \ldots, j-1$, дакле $i \notin \{4, 5, \ldots, j-1\}$. Јасно је



Слика 1.11: Илустрација доказа леме 4.

и да је $i \neq 2$ и $i \neq 3$. Према томе, на најдаљу тачку од e' наилази се после тачке p_j.

Последица ове леме је следећа: да бисмо нашли индекс i, не морамо да кренемо од p_4 током обиласка конвексног омотача; можемо кренути од p_j.

У општем случају, нека је e_k ивица конвексног омотача са крајњим тачкама p_k и p_{k+1} и нека је p_u тачка на максималном растојању од праве l_k кроз e_k. Да бисмо пронашли тачку на максималном растојању од праве l_{k+1} која садржи наредну ивицу e_{k+1}, можемо размотрити темена конвексног омотача почев од p_u и размотрити тачке $p_u, p_{u+1}, \ldots, p_n, p_1, \ldots, p_k$ све док не пронађемо максимални елемент низа

$$d(p_u, l_{k+1}), d(p_{u+1}, l_{k+1}), \ldots, d(p_n, l_{k+1}), d(p_1, l_{k+1}), \ldots, d(p_k, l_{k+1})$$

за који знамо да му елементи прво расту, а затим опадају. На овај начин рачунамо скуп L и притом пролазимо максимално два пута око конвексног омотача, те се овај скуп рачуна у времену $O(n)$. Стога смо доказали наредну теорему:

Теорема 1. *Нека је S скуп од n тачака у равни. Дијаметар скупа S се може израчунати у времену $O(n \log n)$. Ако су тачке скупа S дате као темена конвексног многоугла, сортираним редоследом, онда се дијаметар може израчунати у времену $O(n)$.*

Коментар 2. Може постојати већи број тачака које су на максималном растојању, али сви парови таквих тачака морају бити садржани у скупу L. Стога може постојати највише $4n$ парова тачака које су на растојању $D(S)$ и предложени алгоритам их све налази.

Размотримо још неколико занимљивих геометријских проблема.

1.7 Дужина уније дужи на правој

Проблем. За дати скуп од n дужи на правој, одредити укупну дужину коју покривају ове дужи.

Уколико би све дужи биле међусобно дисјунктне, онда бисмо проблем лако решили: сабрали бисмо дужине свих дужи. Међутим, ове дужи се могу преклапати (потенцијално и по неколико њих истовремено), те је потребно развити другачији алгоритам. Претпоставимо да све дужи леже на x-оси. Идеја је искористити технику вертикалне покретне праве која раван прелази слева удесно и која редом пролази скупом крајњих тачака свих датих дужи.

Формирајмо од свих x координата крајњих тачака дужи низ *tacke* дужине $2n$. Уз x координату тачке потребно је уз сваки члан низа памтити и информацију да ли је тачка почетна и крајња. Сортирајмо затим овај низ према x координати неопадајуће; уколико две или више тачака имају исту вредност x координате, прво се у сортираном низу требају налазити крајње тачке, а затим почетне. Биће нам потребан један наменски бројач br чија је вредност у сваком тренутку број преклопљених дужи непосредно иза покретне праве (а пре наредног краја дужи) и који на почетку иницијализујемо на 0 и променљива *ukupna_duzina* коју такође иницијализујемо на нулу и чија вредност треба да буде збир дужина дужи које је покретна права већ "прегледала". Ова два тврђења нам представљају инваријанту петље алгоритма који треба формулисати. Након тога пролазимо редом кроз елементе низа и за сваки члан низа уколико је вредност бројача већа од нуле променљиву *ukupna_duzina* увећавамо за вредност $tacka[i] - tacka[i-1]$. Приликом наиласка на почетну тачку дужи вредност бројача инкрементирамо, а приликом наиласка на крајњу тачку дужи вредност бројача декрементирамо (дакле у сваком тренутку вредност бројача нам указује на број дужи које се у тој тачки преклапају).

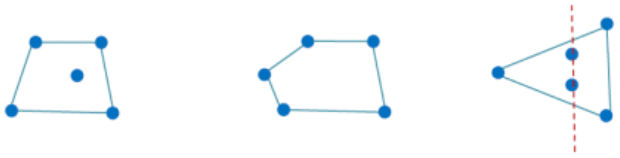
Овај алгоритам (Клее, 1977.) представља илустрацију технике покретне праве. Сложеност овог алгоритма је $O(n \log n)$ што потиче од почетног сортирања крајњих тачака дужи.

1.8 "Happy-ending" teorema

Теорема 3. *За сваки скуп од 5 тачака у равни које су у општем положају постоји четворочлани подскуп који формира темена конвексног четвороугла.*

Ова теорема се може једноставно доказати разматрањем случајева. Уколико су од ових пет тачака четири или пет тачака темена конвексног омотача, онда се сваке четири тачке из овог скупа могу узети као одговор. Ако скуп тачака има облик троугла са две тачке унутар њега, онда се увек могу одабрати две унутрашње тачке и једна од страница троугла. Коју страницу троугла бирамо? С обзиром на то да су тачке у општем положају (тј. никоје три тачке нису колинеарне), права која садржи две унутрашње тачке скупа сече тачно две странице троугла. Бирамо трећу страницу троугла и две унутрашње тачке.

На слици 1.12 приказане су све три могуће конфигурације на које можемо наићи приликом анализе случајева. Овај резултат је Пол Ердеш назвао



Слика 1.12: Различите конфигурације проблема: када се конвексни омотач састоји од пет, четири и три тачке

"happy-ending" теорема јер се двоје његових пријатеља који су радили на овом проблему (Џорџ Секереш и Естер Клајн) венчало. Уопштење овог резултата је теорема Ердеш-Секереш која гласи:

Теорема 4. *Најмањи број тачака за које сваки произвољан распоред садржи конвексни подскуп од n тачака је $2^{n-2}+1$.*

Она је и даље без доказа, али су доказане неке мање прецизне границе.

1.9 Испитивање да ли у скупу од n тачака постоје три колинеарне тачке

Проблем. Нека је задато n тачака у равни. Потребно је испитати да ли међу њима постоје неке три које су колинеарне.

Алгоритам грубе силе би за сваке три тачке проверавао колинеарност, па би сложеност била $O(n^3)$.

За сваку тачку израчунамо нагиб праве кроз њу и сваку од преосталих $n-1$ тачака скупа, а затим тај низ нагиба сортирамо. Након тога потребно је проверити да ли постоје две суседне исте вредности у овом сортираном низу. Ако јесте, онда постоје три колинеарне тачке. Сложеност овог алгоритма је $O(n)$ за рачунање нагиба права, а затим $O(n \log n)$ за њихово сортирање. То радимо n пута, тако да је укупна сложеност алгоритма $O(n^2 \log n)$.

1.10 Испитивање да ли у скупу од n тачака постоји квадрат

Проблем. За датих n тачака у равни потребно је утврдити да ли постоје неке четири које образују квадрат.

Алгоритам грубе силе би за сваке четири тачке проверавао да ли образују квадрат и био би сложености $O(n^4)$. Алгоритам можемо унапредити тако што за сваке две тачке можемо да израчунамо координате потенцијалних квадрата над том страницом (постоје по два таква квадрата). Након тога, потребно је установити да ли се у скупу налазе тачке са датим координатама. Низ тачака можемо сортирати растуће и онда применити бинарну претрагу. Алгоритам је сложености $O(n^2 \log n)$.

1.11 Резиме

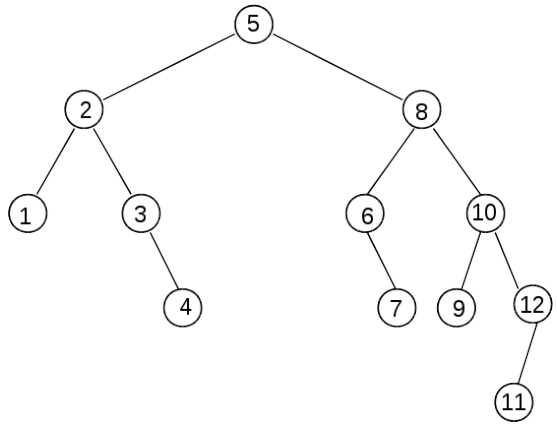
Геометријски алгоритми су на неки начин мање апстрактни од графовских, јер смо навикли да видимо и радимо са геометријским објектима. Међутим, то је само први утисак. Радити са огромним бројем објеката није исто што и радити са малим сликама, па морамо бити опрезни да нас представа коју имамо не наведе на погрешне закључке. Морају се предвидети многи специјални случајеви. Алгоритам за утврђивање припадности тачке унутрашњости многоугла (одељак 1.2) је добар пример. Због тога конструкција геометријских алгоритама захтева посебан опрез.

У конструкцији (дискретних) геометријских алгоритама важну улогу игра индукција. Техника покретне праве, заснована на индукцији, заједничка је за више геометријских алгоритама; такође се често користи декомпозиција. Геометријски алгоритми (изузев најједноставнијих) често захтевају компликоване структуре података, па су многе сложене структуре података развијене управо за ту намену. Овде нисмо разматрали ни једну од таквих специјалних структура података.

Глава 2

АВЛ стабла

АВЛ стабла (која су име добила по ауторима, Адельсон–Вельский и Ландис, 1962.) су структура података која гарантује да сложеност ни једне од операција тражења, уметања и брисања у **најгорем случају** није већа од $O(\log n)$, где је n број елемената. Идеја је уложити допунски напор да се после сваке операције стабло **уравнотежи**, тако да висина стабла увек буде $O(\log n)$. При томе се уравнотеженост стабла дефинише тако да се може лако одржавати. Прецизније, АВЛ стабло се дефинише као бинарно стабло претраге код кога је за сваки чвор апсолутна вредност разлике висина левог и десног подстабла мања или једнака од један (видети пример на слици 2.1).



Слика 2.1: Пример АВЛ стабла.

Као што показује следећа теорема, висина АВЛ стабла је $O(\log n)$.

Теорема 5. *За АВЛ стабло са n унутрашњих чворова висина h задовољава услов $h < 1.4405 \log_2(n+2) - 0.327$.*

Доказ. Нека је T_h АВЛ стабло висине $h \geq 0$ са најмањим могућим бројем унутрашњих чворова. Стабло T_0 садржи само корен, а стабло T_1 корен и једног сина, рецимо десног. За $h \geq 2$ стабло T_h може се формирати

на следећи начин: његово подстабло мање висине $h-2$ такође треба да буде АВЛ стабло са минималним бројем унутрашњих чворова, дакле T_{h-2}; слично, његово друго подстабло треба да буде T_{h-1}. Стабла описана оваквом "диференцном једначином" зову се *Фибоначијева стабла*. Неколико првих Фибоначијевих стабала, таквих да је у сваком унутрашњем чвору висина левог подстабла мања, приказано је на слици 2.2. Означимо са n_h минимални број унутрашњих чворова АВЛ стабла висине h, тј. број унутрашњих чворова стабла T_h; $n_0 = 0$, $n_1 = 1$, $n_2 = 2$, $n_3 = 4, \ldots$. По дефиницији Фибоначијевих стабала је $n_h = n_{h-1} + n_{h-2} + 1$, односно $n_h + 1 = (n_{h-1} + 1) + (n_{h-2} + 1)$, за $h \geq 2$. Видимо да бројеви $n_h + 1$ задовољавају исту диференцну једначину као и Фибоначијеви бројеви F_h ($F_{h+2} = F_{h+1} + F_h$ за $h \geq 1$; $F_1 = F_2 = 1$). Узимајући у обзир једнакост по прва два члана низова, индукцијом се непосредно показује да важи $n_h + 1 = F_{h+2}$. Према томе, за број n унутрашњих чворова АВЛ стабла висине h важи неједнакост

$$n \geq F_{h+2} - 1.$$

Полазећи од израза за општи члан Фибоначијевог низа

$$F_h = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^h - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^h$$

и везе

$$\frac{1-\sqrt{5}}{2} = -\frac{1}{\frac{1+\sqrt{5}}{2}}$$

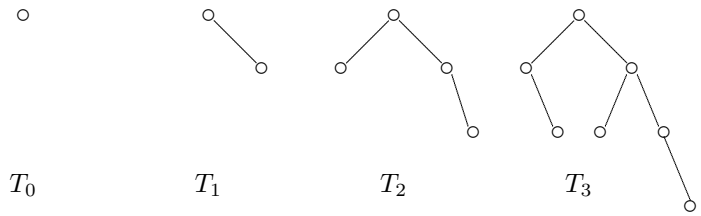
добијамо

$$n \geq \frac{1}{\sqrt{5}}\left(\alpha^{h+2} - \left(-\frac{1}{\alpha}\right)^{h+2}\right) - 1 > \frac{1}{\sqrt{5}}\alpha^{h+2} - 2,$$

где је са $\alpha = (\sqrt{5}+1)/2$ означен позитиван корен карактеристичне једначине $x^2 - x - 1 = 0$ линеарне диференцне једначине $F_n = F_{n-1} + F_{n-2}$. Ова неједнакост еквивалентна је са

$$h < \frac{1}{\log_2 \alpha}\log_2(n+2) - \left(2 - \frac{\log_2 5}{2\log_2 \alpha}\right),$$

чиме је теорема доказана, јер је $1/\log_2 \alpha \simeq 1.44042$ и $\log_2 5/(2\log_2 \alpha) \simeq 0.3277$.



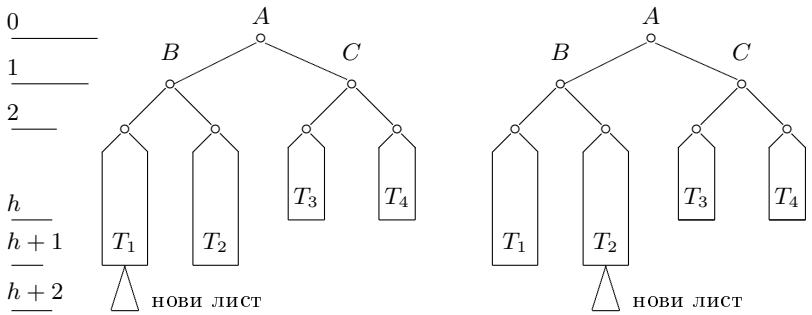
Слика 2.2: Фибоначијева стабла T_0, T_1, T_2 и T_3.

Остаје још да видимо како се после уметања, односно брисања елемента из АВЛ стабла, може интервенисати тако да резултат и даље буде АВЛ стабло.

Приликом уметања новог броја у АВЛ стабло поступа се најпре на начин уобичајен за бинарно стабло претраге: проналази се место чвору, па се у стабло додаје нови чвор са кључем једнаким задатом броју. Чворовима на путу који се том приликом прелазе одговарају разлике висина левог и десног подстабла (**фактори равнотеже**) из скупа $\{0, \pm 1\}$. Посебно је интересантан последњи чвор на том путу који има фактор равнотеже различит од нуле, тзв. **критични чвор**. Испоставља се да је приликом уметања броја у АВЛ стабло *довољно уравнотежити подстабло са кореном у критичном чвору* - у то ћемо се уверити када видимо детаље поступка уравнотежавања.

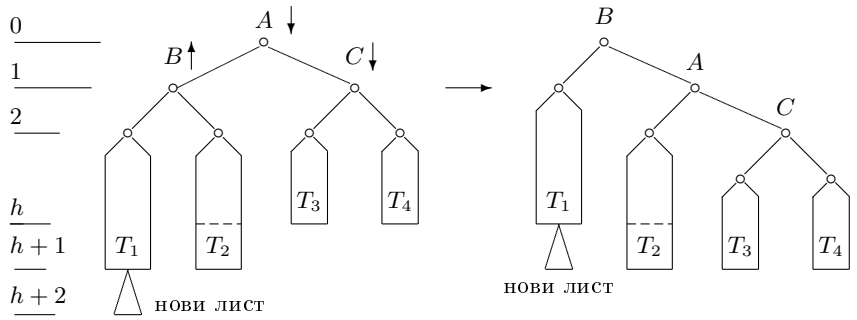
Претпоставимо да је фактор равнотеже у критичном чвору једнак 1 (видети слику 2.3). Нови чвор може да заврши у:

- десном подстаблу – у том случају подстабло коме је корен критични чвор остаје АВЛ
- левом подстаблу – у том случају стабло престаје да буде АВЛ и потребно је интервенисати. Према томе да ли је нови чвор додат левом или десном подстаблу левог подстабла, разликујемо два случаја, видети слику 2.3.
  - У првом случају се на стабло примењује **ротација**: корен левог подстабла B (видети слику 2.4) подиже се и постаје корен подстабла (коме је корен био критичан чвор), а остатак стабла преуређује се тако да стабло и даље остане БСП. Стабло T_1 се "подиже" за један ниво остајући и даље лево подстабло чвора B; стабло T_2 остаје на истом нивоу, али уместо десног подстабла B постаје лево подстабло A; десно подстабло A спушта се за један ниво. Пошто је A критичан чвор, фактор равнотеже чвора B је 0, па стабла T_1 и T_2 имају исту висину. Стабла T_3 и T_4 не морају имати исту висину, јер чвор C није на путу од критичног чвора до места уметања (слика 2.4). Нови корен подстабла постаје чвор B, са фактором равнотезе 0.

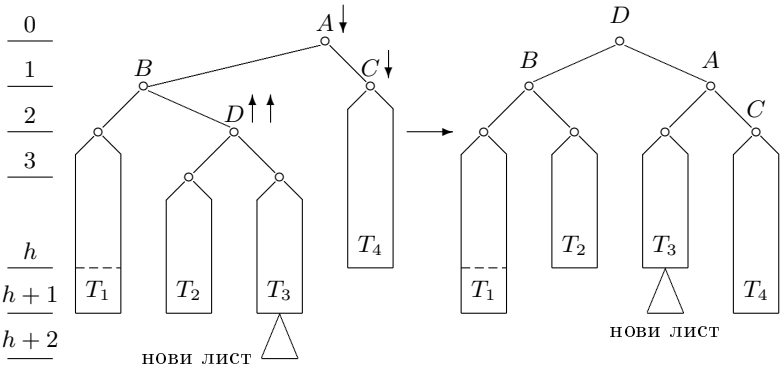


Слика 2.3: Уметања која ремете АВЛ својство стабла.

  - Други случај је компликованији; тада се стабло може уравнотежити **двоструком ротацијом**, видети слику 2.5. Нови чвор подстабла уместо критичног чвора постаје десни син левог сина критичног чвора.



Слика 2.4: Уравнотежавање АВЛ–стабла после уметања — ротација.

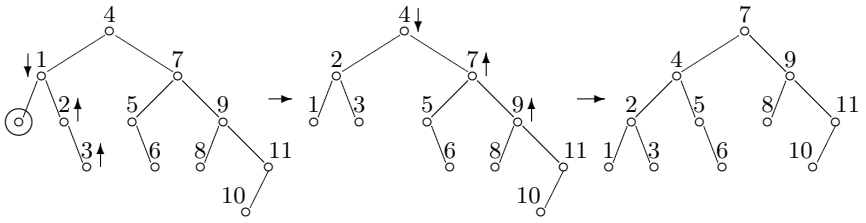


Слика 2.5: Уравнотежавање АВЛ–стабла после уметања — двострука ротација.

Запажамо да у оба случаја висина подстабла коме је корен критични чвор после уравнотежавања остаје непромењена. Уравнотежавање подстабла коме је корен критичан чвор због тога не утиче на остатак стабла. Уз сваки чвор стабла чува се његов фактор равнотеже, једнак разлици висина његовог левог и десног подстабла; за АВЛ стабло су те разлике елементи скупа $\{-1, 0, 1\}$. Уравнотежавање постаје неопходно ако је фактор неког чвора ± 1, а нови чвор се умеће на "погрешну" страну. После уравнотежавања висине изнад критичног чвора остају непромењене. Комплетан поступак уметања са уравнотежавањем дакле изгледа овако: идући наниже приликом уметања памти се последњи чвор са фактором различитим од нуле – критичан чвор; кад се дође до места уметања, установљава се да ли је уметање на "доброј" или "погрешној" страни у односу на критичан чвор, а онда се још једном пролази пут уназад до критичног чвора, поправљају се фактори равнотеже и извршавају евентуалне ротације.

Брисање је компликованије, као и код обичног БСП. У општем случају се уравнотежавање не може извести помоћу само једне или две ротације. На пример, да би се Фибоначијево стабло F_h са n чворова уравнотежило после брисања "лоше" изабраног чвора, потребно је извршити $h-2$, односно $O(\log n)$ ротација (видети слику 2.6 за $h = 4$). У општем случају је граница за потребан број ротација $O(\log n)$, На срећу, свака ротација захтева кон-

стантни број корака, па је време извршавања брисања такође ограничено одозго са $O(\log n)$. На овом месту прескочићемо детаље.



Слика 2.6: Уравнотежавање Фибоначијевог стабла T_4 после брисања чвора, помоћу две ротације.

АВЛ стабло је ефикасна структура података. Чак и у најгорем случају АВЛ стабла захтевају највише за 45% више упоређивања од оптималних стабала. Емпиријска испитивања су показала да се просечно тражење састоји од око $\log_2 n + 0.25$ поређења. Основни недостатак АВЛ стабала је потреба за додатним меморијским простором за смештање фактора равнотеже, као и чињеница да су програми за рад са АВЛ стаблима доста компликовани. Постоје и друге варијанте уравнотежених стабала претраге, на пример $2-3$ стабла, B-стабла и црвено-црна стабла.

Литература

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press, 2002.

[2] М. Живковић, Алгоритми, Математички факултет, 2000.

[3] E. Horowitz, S. Sahni, S. Rajasekaran, Computer Algorithms, Computer Science Press, 1997.

[4] I. Parberry, W. Gasarch, Problems on Algorithms, Second Edition, 2002.