УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Саша Букуров

ПЛАВО-ЦРВЕНИ ХАКЕНБУШ

мастер рад

Београд, 2021.

Ментор:

др Миодраг Живковић, редовни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

др Предраг Јаничић, редовни професор
Универзитет у Београду, Математички факултет

др Иван Чукић, доцент
Универзитет у Београду, Математички факултет

Датум одбране: 27. септембар 2021.

Садржај

Глава 1

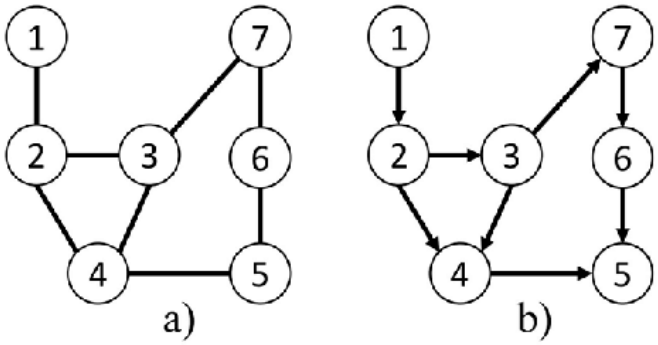
Увод

Тема овог рада је игра хакенбуш, конкретно њена варијанта, плаво-црвени хакенбуш. За почетак дају се одговори на питања шта је игра, како се игра, које варијације постоје. Након тога, фокус ће бити на математичкој репрезентацији игре, где јој је место у теорији игара. Затим следи упознавање са варијантом игре која је уједно и тема овог рада, плаво-црвеним хакенбушом и поступком израчунавања вредности позиције. На крају се приказује имплементација игре и алгоритама оцењивања позиције.

1.1 Графови

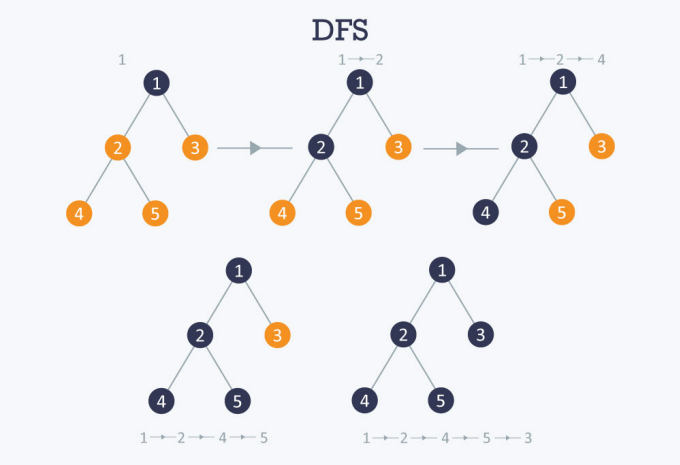
Граф G је структура података која се састоји од скупа чворова V и скупа грана E. Грана одговара пару чворова. Сваки чвор може бити део једне или више грана. Пут од v_1 до v_k је низ чворова $v_1, v_2, ..., v_k$ повезаних гранама $e_i = (v_{i-1}, v_i)$ за $i = 1, 2, ..., k$ [4]. Граф је усмерен, односно неусмерен, ако су му гране уређени, односно неуређени парови. Пример неусмереног и усмереног графа је дат на слици 1.1.

Обиласци графова су веома важни алгоритми у теорији графова. Два основна алгоритма су претрага у дубину и претрага у ширину. За овај рад битан је алгоритам претраге у дубину (DFS, енгл. *depth-first-search*). Принцип алгоритма је описан на примеру обиласка места у округу. Граф G одговара неком округу унутар државе. Чворови су места, а гране су путеви између места. Циљ је обићи сва места. Два места повезује само један пут. Идеја је држати се пута док год се не стигне до места из ког нема новог пута. На раскрсницама се остављају заставице и бира се један од путева ка непосећеним



Слика 1.1: Неусмерен и усмерен граф

местима. Заставица се поставља само први пут када се дође до раскрснице. Када се обиђу сва места до којих воде путеви из раскрснице, враћа се до прве следеће раскрснице која има непосећена места. Ова раскрсница више неће бити посећена. У терминологији графова, од почетног чвора прате се гране које воде до непосећених чворова. Сваки чвор који је посећен се додаје у листу посећених чворова. Када из полазног чвора све гране воде ка посећеним чворовима, претрага је завршена. DFS обилазак је илустрован на слици 1.2.

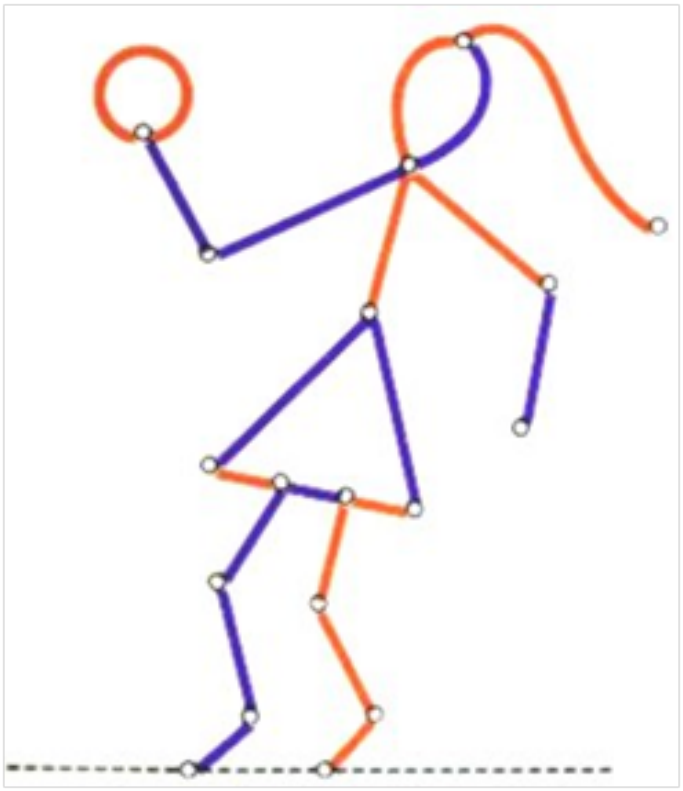


Слика 1.2: Обилазак у дубину

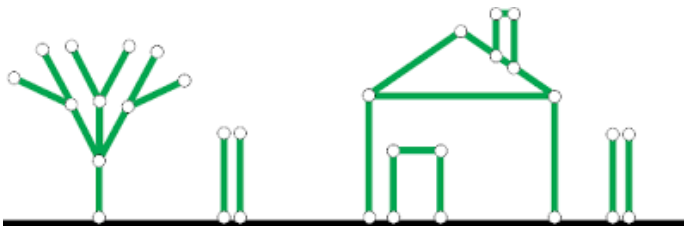
1.2 Основе игре

Хакенбуш је игра за два играча. Постоји једнобојни, зелени хакенбуш, и вишебојне варијанте, плаво-црвени и плаво-црвено-зелени хакенбуш. Цртеж,

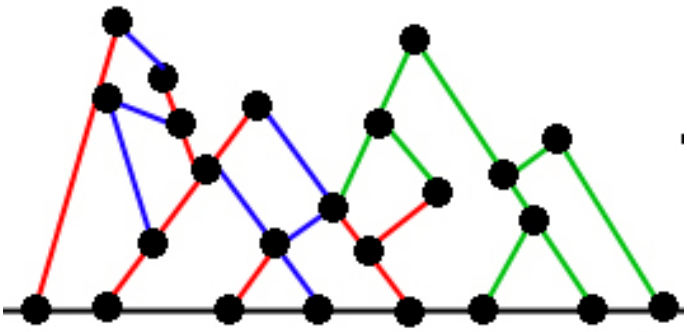
којим се задаје игра, се састоји од хоризонталне линије, коју називамо тло, за коју су повезани и изнад које се налазе сви делови цртежа. Сваки део цртежа повезан је за тло једном или више линија. Боја линија зависи од варијанте игре. Пример цртежа за игру плаво-црвеног хакенбуша је приказан на слици 1.3, зеленог хакенбуша на слици 1.4 и пример плаво-црвено-зеленог хакенбуша на слици 1.



Слика 1.3: Плаво-црвени хакенбуш



Слика 1.4: Зелени хакенбуш



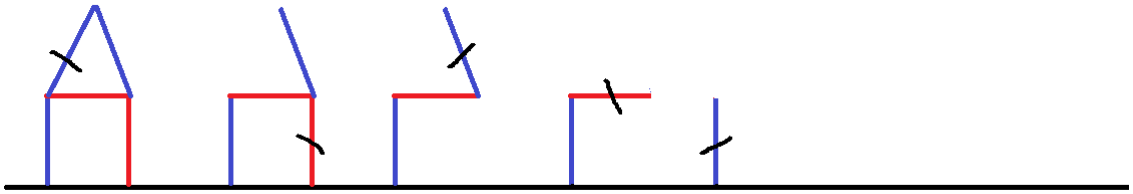
Слика 1.5: Плаво-зелено-црвени хакенбуш

Цртежу који описује игру може се придружити граф, који омогућује формализацију:

- Тло се представља посебним чвором, тзв. 'нулти чвор'.
- Линије су гране графа.
- Остали чворови графа су заједничке тачке грана.
- Боје линија се придружују гранама.

Два играча хакенбуша су *леви* и *десни* играч. *Леви* на свом потезу може да уклони једну плаву грану, док десни може да уклони једну црвену грану кад је на потезу. У варијанти игре где су подржане и зелене гране, оба играча могу да их уклоне. По уклањању гране из графа, такође уклањају се све гране које садрже чворове до којих не постоји пут од нултог чвора. Победник игре је играч који одигра последњи потез, што илуструје пример на слици 1.6.

Игру је лако замислити ако гране представимо као дугуљасте балоне надуване хелијумом, а граф као скуп балона који су међусобно везани кратким концима при чему је макар један причвршћен за тло. Бушењем балона заправо уклањамо грану из графа, а они балони који одлете су гране чији чворови нису повезани путем са нултим чвором.



Слика 1.6: Пример игре плаво-црвеног хакенбуша

Глава 2

Основни појмови и дефиниције

Хакенбуш спада у комбинаторне игре за два играча. У овом поглављу су објашњени основни појмови у вези са комбинаторним играма. Након тога је представљен плаво-црвени хакенбуш и све што је потребно знати о игри како би рачунање позиције и имплементација, које следе касније у тексту, биле јасне.

2.1 Комбинаторне игре за два играча

У комбинаторној теорији игара постоји доста класификација. Комбинаторне игре се могу поделити на пристрасне и непристрасне игре. Пристрасна игра је игра у којој могући потези зависе не само од тренутне позиције, већ и од играча који треба да одигра потез. Фактор по ком се разликују играчи може бити боја играча. Пример пристрасне игре је шах, где један играч помера само беле фигуре, а други само црне. Код непристрасних игара, једина разлика између два играча је ко игра први. Пример непристрасне игре је ним. Ним је игра за два играча у којој играчи наименично узимају предмете са једне од бар две гомиле, нпр. шибице из кутија. Играч мора да узме барем једну шибицу када је на потезу, а може да узме највише толико шибица колико их има у кутији. У једном потезу се не могу узети шибице из више кутија. Циљ игре је не узети последњу шибицу. Друга подела је везана за доступност информација играчима: постоје игре са потпуном и непотпуном информацијом. Игре са потпуном информацијом подразумевају да [3]:

- Оба играча знају правила игре.

- Оба играча знају ко је на потезу.
- Оба играча знају тренутно и сва претходна стања игре.
- Не постоји догађај у игри који се ослања на вероватноћу (нпр. исход бацања коцкица, извлачење карата).

Пример игре са потпуном информацијом је шах. Игра са непотпуном информацијом је игра која не испуњава један или више услова игара са потпуним информацијама. Пример игре са непотпуном информацијом је *дилема затвореника* (енгл. *prisoner's dilemma*). Два криминалца су ухваћена након пљачке, полиција их испитује раздвојено, немају начин да комуницирају. Ако обојица не признају злочин, могу бити осуђени само за нижи преступ са казном од две године. Ако један призна и тиме изда другог, он служи само годину дана, док други служи четири године. Ако се деси да обојица признају, обојица добијају казну од 4 године. С обзиром да су затвореници раздвојени, немају информацију о другом и тиме не знају туђи потез и из тог разлога се ради о игри са непотпуном информацијом. Већина карташких игара су игре са непотпуном информацијом јер играчи не знају које карте тренутно поседују противници, поред тога мешање карата доводи до непредвидивости која је следећа карта која улази у игру.

Хакенбуш је игра са комплетном информацијом. Оба играча знају како изгледа иницијални граф, виде сваки потез противника и не постоји никаква тајна информација коју један од њих поседује. У зависности од варијанте игра може бити пристрасна игра, плаво-црвени и плаво-зелено-црвени хакенбуш, или непристрасна, зелени хакенбуш.

2.2 Плаво-црвени хакенбуш

Плаво-црвени хакенбуш је игра са следећим правилима:

1. Постоје само два играча, *леви* и *десни*.
2. Постоји почетна позиција.
3. Јасно су дефинисана правила који играч може да одигра који потез од свих могућих.
4. *Леви* и *десни* наизменично вуку потезе до краја игре.

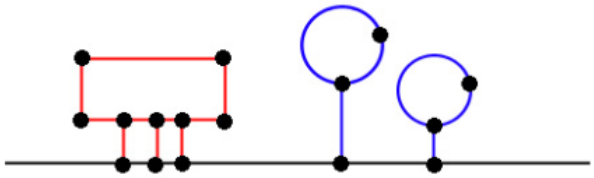
5. Оба играча знају шта се дешава, поседују комплетну информацију.

6. Не постоје потези који се ослањају на вероватноћу(нпр. бацање коцкица, мешање карата).

7. Први играч који не може да одигра потез је изгубио.

8. Правила су тако дефинисана да се игра увек мора завршити јер ће неки играч остати без потеза. Ово се назива услов завршавања. Не постоје партије у којима је могуће да се потези понављају, што би онемогућило да се игра заврши [2].

Пример 1. На слици 2.1 је приказ једноставне позиције плаво-црвеног хакен-буша. У овој позицији *десни* располаже са 10 грана, док *леви* има 6. Граф је подељен на два дела, један само са плавим и један само са црвеним гранама, из чега следи да ниједан од играча не може да утиче на игру другог својим потезима. Под условом да оба играча уклањају гране тако да искористе све потезе, *десни* има 4 потеза предности. Како год *леви* да одигра, играо први или други, не може да победи у овој игри, под условом да *десни* игра оптимално.

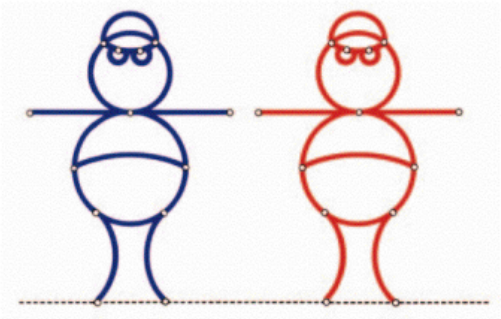


Слика 2.1: Једноставна игра

Дефиниција 1 (Нулта игра)**.** Нулта игра (енгл. *zero-game*) је игра у којој играч који је први на потезу увек губи.

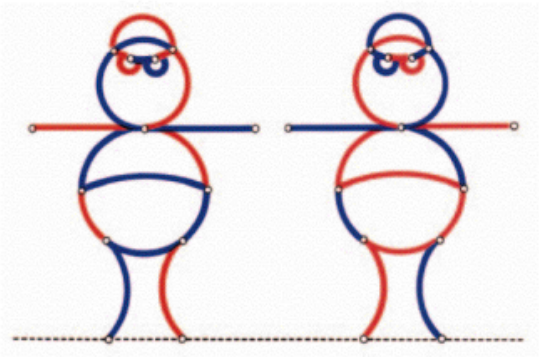
Пример 2. На слици 2.2 налазе се Ивица и Перица. Ивица је представљен са 19 плавих грана, а Перица са 19 црвених грана. Поставка је слична као

у примеру 1, с тим да овог пута оба играча имају на располагању исти број потеза. Дакле, ако *леви* и *десни* играју наизменично, први играч ће први остати без потеза.



Слика 2.2: Ивица и Перица пре препирке

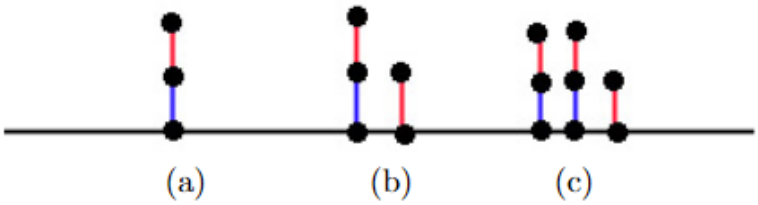
Пример 3. На слици 2.3 поново се налазе Ивица и Перица који су представљени са 19 плавих и 19 црвених грана. Сваки од њих садржи 19 грана, али више нису једнобојни. Ова два графа, ако се боје не узму у обзир, су идентична. Распоред боја грана је инверзан, тј. свака плава грана на Ивици је црвена грана на Перици и обрнуто. Слично као и у примеру 2, играч који је други на потезу треба само да имитира потезе првог и увек ће победити, тј. први ће изгубити.



Слика 2.3: Ивица и Перица после преприке

Постоје позиције кад један од играча има неколико потеза предности, постоји нулта позиција тј. нула потеза предности, а да ли постоји позиција кад је предност мање од потеза?

Пример 4. На слици 2.4(a) *леви* има очигледну предност, ако игра први аутоматски је победио, а ако игра други, *десни* има прилику да одигра потез, али након тога *леви* опет побеђује. Да ли то значи да *леви* има потез предности? Слика 2.4(b) даје одговор на то питање тако што се *десном* додаје цео потез. Да је *леви* имао цео потез предности, додавањем целог потеза *десном* би значило да је граф на слици 2.4(b) нулта игра. Ако *леви* игра први, он брише једину грану коју има, али то оставља *десном* цео потез којим побеђује партију. Ако *десни* игра први, он ће обрисати црвену грану која се ослања на плаву, тиме се добија нулта игра где *леви* игра *леви* први, дакле губи. Ово значи да *леви* увек губи што значи да *леви* нема цео потез предности на слици 2.4(a). Колика је заправо онда његова предност? Спајањем графова 2.4(a) и (b), добија се граф (c). Овим ће се показати да је позиција 2.4(a) вредна $\frac{1}{2}$ потеза, јер су две копије тог графа компезовале додатног потеза *десном* и тиме је добијена нулта игра. Ако *леви* игра први, он брише једну од плавих грана и тиме се добија слика 2.4(b), дакле *десни* увек побеђује. Ако *десни* игра први, има за опцију да брише или самосталну црвену грану или неку која се ослања на плаву. У првом случају, победа *левоī* је очигледна. У другом случају, након што *десни* избрише једну од две горње црвене гране, *леви* треба да брише супротну плаву грану. Ту се добија нулта игра у којој *десни* игра први, дакле *леви* побеђује. С обзиром да је показано да је граф (c) нулта игра и да *десни* има цео потез више, долазимо до закључка да је позиција на слици (a) стварно вредна $\frac{1}{2}$ потеза [2].



Слика 2.4: Позиција вредна пола потеза

Глава 3

Одређивање вредности позиције

Вредност позиције се може схватити и као тренутно стање игре, ко од играча у датом тренутку има предност. У овом тексту, вредност позиције је већ неколико пута споменута. Вредност позиције графа са слике 2.1 је одређена на интуитиван начин, одузимањем броја црвених од броја плавих грана. За компликованије позиције овакав приступ се не може користити, као што је приказано касније у тексту. На примеру 2.4 је доказано да граф са слике (а) има вредност од $\frac{1}{2}$. Начин на који је ово показано је прихватљив за графове од од 2-3 гране, док је за веће непрактичан.

У овој глави су представљени начини рачунања тренутне позиције. Код оцењивања је битно напоменути да се оцена даје у односу на *левог* играча. Ако је оцена x вредност тренутне позиције онда треба да важи да за:

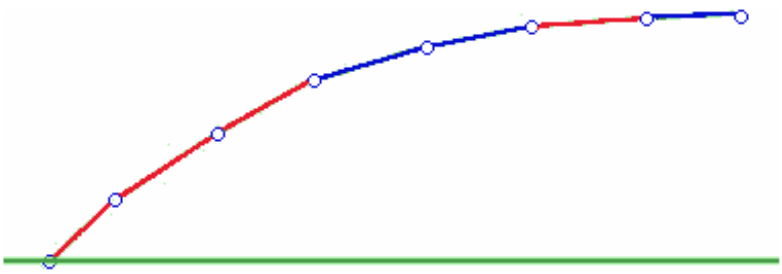
- $x < 0$, *леви* губи,
- $x = 0$, ради се о нултој игри и
- $x > 0$, *леви* побеђује.

У наставку се приказује како се може израчунати вредност позиције ако се ради о ланцу, стаблу и произвољном графу

3.1 Рачунање вредности ланца

У хакенбуш играма чест случај су ланци, графови код којих не постоји рачвање, већ су све гране у низу, као што је приказано примером на слици 2.4.

За одређивање вредности ланца може се користи метода Елвина Берлекампа (Elwyn Berlekamp, [1][2]).



Слика 3.1: Пример ланца

Први корак код Берлекампове методе је одређивање знака оцене. Прва грана од тла одређује знак оцене, плава за позитивну, црвена за негативну оцену. Свим гранама које су исте боје као прва додељује се вредност 1, а гранама супротне боје вредност 0. Изузетак су гране које спаја *бинарна тачка*. Чвор који спаја прве две гране различитих боја се назива бинарна тачка и тим гранама се не придружује никаква вредност. Све вредности пре бинарне тачке се сабирају као јединице. Вредности након бинарне тачке се посматрају као цифре бинарног броја. У запис тог бинарног броја се додаје 1 на крају. Ако је b_i i-та цифра датог бинарног броја иза бинарне тачке и ако је n број цифара бинарног броја, вредност тог бинарног броја се рачуна по формули:

$$\sum_{i=1}^{n} b_i \cdot \left(\frac{1}{2}\right)^i$$

Та вредност се сабира са бројем јединица леве стране бинарне тачке и тиме је добијена оцена ланца. Ако је прва грана била црвена, резултат се множи са -1.

Доказ. Ланац L дужине n грана се састоји из 3 дела: једнобојни део дужине м грана, бинарна тачка коју чине две гране различите боје и реп дужине n-m-2 грана. Ако знамо да је сума првих n бројева, где је сваки број $\left(\frac{1}{2}\right)^i$ за $i = 1..n$, једнака $\left(\frac{1}{2}\right)^n$, реп се може представити као:

$$b_1 \cdot \left(\frac{1}{2}\right) + b_2 \cdot \left(\frac{1}{4}\right) + ... + b_n \cdot (z_n) = 1 - \left(\frac{1}{2}\right)^n - z_n$$

где је

$$z_n = \sum_{i=1}^{n} (1 - b_i) \left(\frac{1}{2}\right)^i$$

број који представља суму свих степена $\frac{1}{2}$ где је $b_i = 0$.

База индукције: $n = 1$

Постоје два случаја:

За $b_1 = 0$, $\quad z_1 = \frac{1}{2}$ pa se dobija $1 - \left(\frac{1}{2}\right)^1 - z_1 = 1 - \frac{1}{2} - \frac{1}{2} = 0$

За $b_1 = 1$, $\quad z_1 = 0$ pa se dobija $1 - \left(\frac{1}{2}\right)^1 - z_1 = 1 - \frac{1}{2} - 0 = \frac{1}{2}$

Индуктивна хипоза: за $n = k$ добија се израз

$$b_1 \cdot \left(\frac{1}{2}\right) + b_2 \cdot \left(\frac{1}{4}\right) + ... + b_k \cdot \left(\frac{1}{2}\right)^k = 1 - \left(\frac{1}{2}\right)^k - z_k$$

Индуктивни корак: Посматрамо реп дужине $n = k + 1$

Тада је израз из тврђења једнак

$$b_1 \cdot \left(\frac{1}{2}\right) + b_2 \cdot \left(\frac{1}{4}\right) + ... + b_k \cdot \left(\frac{1}{2}\right)^k + b_{k+1} \cdot \left(\frac{1}{2}\right)^{k+1} = 1 - \left(\frac{1}{2}\right)^{k+1} - z_{k+1}$$

где је

$$z_{k+1} = \begin{cases} z_k, b_{k+1} = 1 \\ \\ z_k + \frac{1}{2}, b_{k+1} = 0 \end{cases}$$

Ако се на индуктивну хипотезу дода још и $\left(\frac{1}{2}\right)^{k+1}$ са обе стране једнакости, добија се следећа једнаост:

$$b_1 \cdot \left(\frac{1}{2}\right) + b_2 \cdot \left(\frac{1}{4}\right) + ... + b_k \cdot \left(\frac{1}{2}\right)^k + b_{k+1} \cdot \left(\frac{1}{2}\right)^{k+1} = 1 - \left(\frac{1}{2}\right)^k - z_{k+1} + b_{k+1} \cdot \left(\frac{1}{2}\right)^{k+1}$$

Постоје два случаја $b_{k+1} = 0$ и $b_{k+1} = 1$.
Случај $b_{k+1} = 1$:

$$\begin{aligned}
&= 1 - \left(\frac{1}{2}\right)^k - z_{k+1} + \left(\frac{1}{2}\right)^{k+1} \\
&= 1 - \frac{2^{k+1} - 2^k}{2^{k+1} \cdot 2^k} - z_{k+1} \\
&= 1 - \frac{2^k \cdot (2-1)}{2^{k+1} \cdot 2^k} - z_{k+1} \\
&= 1 - \frac{1}{2^{k+1}} - z_{k+1}
\end{aligned}$$

Случај $b_{k+1} = 0$:

$$\begin{aligned}
&= 1 - \left(\frac{1}{2}\right)^k - z_k \\
&= 1 - \left(\frac{1}{2}\right)^k - z_k + \left(\frac{1}{2}\right)^{k+1} - \left(\frac{1}{2}\right)^{k+1} \\
&= 1 - \left(\frac{1}{2}\right)^k + \left(\frac{1}{2}\right)^{k+1} - z_k - \left(\frac{1}{2}\right)^{k+1} \\
&= 1 - \frac{2^k \cdot (2-1)}{2^{k+1} \cdot 2^k} - (z_k + \left(\frac{1}{2}\right)^{k+1} \\
&= 1 - \frac{1}{2^{k+1}} - z_{k+1}
\end{aligned}$$

□

Пример 5. На слици 3.1 је дат ланац за који је, у овом примеру, демонстрирана Берлекампова метода. Прва грана је негативна, што означава негативну оцену. Бинарна тачка је између треће и четврте гране. С леве стране бинарне тачке збир је 2. С десне стране добијен је бинарни број 010 на који је додат 1, дакле 0101. Вредност тог броја је $\frac{1}{2}+1\cdot\frac{1}{4}+0\cdot\frac{1}{8}+1\cdot\frac{1}{16} = \frac{5}{16}$. Сабирањем леве и десне стране добија се $2\frac{5}{16}$. Као што је на почетку речено, црвена грана значи да је резултат негативан, тако да је коначни резултат -$2\frac{5}{16}$.

Други приступ који може да се примени је метода Теа ван Руда (Tea van Rud) [2]. Приступ је сличан, свакој плавој грани се дода вредност 1, а црвеној вредност -1. Бинарна тачка постоји и у овој методи, али обе гране бинарне

тачке овог пута улазе у рачуницу при чему се не додаје 1 на крају бинарног броја.

Пример 6. У овом примеру се за ланац са слике 3.1 демонстрира метода Теа ван Руда. Свакој плавој се додели вредност 1, а црвеној -1. Од ланца добијена је следећа рачуница: -1-1-1+$\frac{1}{2}$+$\frac{1}{4}$-$\frac{1}{8}$+$\frac{1}{16}$ = -2$\frac{5}{16}$. Може се приметити да ова метода даје исти резултат као и Берлекампова метода у примеру 5.

Берлекампова метода је инвертибилна, што значи да се од оцене може креирати ланац који има ту оцену. Оцена се састоји из два дела, целог броја x и разломка y. Цео број представља низ грана исте боје пре бинарне тачке а разломак после бинарне тачке. Ако оцена не садржи први део, цео број, то значи да бинарној тачки на претходи ниједна грана. Од знака оцене зависи која је прва грана, плус за плаву, минус за црвену грану. У односу на знак, прво се нацрта x грана, а затим гране бинарне тачке. Након тога се разломак y претвори у бинарни запис. Пошто се приликом одређивања оцене на крај бинарног записа додавало 1, сада се та јединица брише. Према знаку и бинарном запису, додају се гране одговарајуће боје. За позитивну оцену 1 је плава грана, а 0 црвена. За негативну оцену важи обрнуто.

Пример 7. Дата је оцена -2−$\frac{5}{16}$. Број -2 преставља две црвене гране пре бинарне тачке, што значи да у ланцу се налазе 3 узастопне црвене гране коју прати једна плава. Вредност $\frac{5}{16}$ у бинарном запису је 0101. Пошто се приликом рачунања оцена бинарном броју додало 1 на крај, та јединица је избачена из записа. Остао је број 010 који представља плаву, црвену и плаву грану. Слика 3.1 одговара овом низу грана.

Као што је већ напоменуто, дате методе су исклучиво за рачунање вредности ланца, не могу се применити на обичне графове попут 2.1 или 2.3.

3.2 Рачунање вредности позиције и правило једноставности

Пример са слике 2.1 је оцењен интуитивном методом и добијен је тачан резултат. Као што је већ речено, ово је веома једноставан пример што је дозволило да се лако израчуна тачна оцена без неке метода и/или правила. Пре удубљивања у сам процес рачунања позиције треба увести нотацију како

се може представити тренутно стање игре. Било која позиција G се може окарактерисати изразом следећег облика:

$$\{l_0, l_1, l_2...|r_1, r_2, r_3...\}$$

где $l_0, l_1, l_2...$, представља прелаз из тренутног стања у ново стање игре, где су оцене новог графа $l_0, l_1, l_2...$ након што *леви* одигра било који од могућих потеза. Слично, $r_1, r_2, r_3...$, су оцене графа наког што *десни* одигра било који од могућих потеза. Као што је већ напоменуто, предност *левоī* се осликава у позитивним вредностима, а *десноī* у негативним. Предност од једног потеза за *левоī* је 1 док је предност од једног потеза за *десноī* -1.

Пример 8. Овом нотацијом граф са слике 2.1 се може представити као {-5, -7| -3}. *Леви* играч има вредности потеза -5 и -7 за уклањање плавих грана. Вредност -5 се добија када *леви* уклони било коју плаву грану такву да не додирује тло. Збир преосталих плавих грана је 5, а црвених 10, тако да је 5-10=-5. Вредност -7 се добија када *леви* уклони једну грану, од две могуће, која додирује тло. Коју год црвену грану *десни* да уклони добија се исти резултат, -3, јер број преосталих црвених грана се не мења, увек је 9. Сличним поступком се види да граф са слике 2.3 има вредности {-1|1}.

Поставља се питање, како са датим информацијама доћи до оцене? За одговор на то питање је потребно дефинисати правило једноставности (енгл. *Simplicity Rule*)

Дефиниција 2 (Правило једноставности)**.** За игру G са опцијама (могућим вредностима)

$$\{l_0, l_1, l_2...|r_1, r_2, r_3...\}$$

постоји број x такав да важи да је већи од сваког l_i и мањи од сваког r_j. Број x треба да буде најједноставнији број који испуњава овај услов.

Пре дефинисања вредности броја x, треба поједноставити нотацију. За игру G са опцијама

$$\{l_0, l_1, l_2...|r_1, r_2, r_3...\}$$

нека број a представља максимум из скупа $l_0, l_1, l_2....$, док број b представља минимум из скупа $r_0, r_1, r_2....$

Дефиниција 3. За дате бројеве a и b графа G, нека је x вредност тренутне позиције. Вредност x-а је цео број такав да је најближи нули и испуњава услове да је $x>a$ и $x<b$. У супротном, ако у интервалу (a,b) не постоји цео број, x је рационални број чији је именилац најмањи могући степен двојке.

Да ли се правило једноставности може користи за ланце као и за произвољне графове? Испоставља се да су ланци посебан случај и да се на њих не може применити правило једноставности, што показује следећи пример.

Пример 9. Ланац са слике 3.1, има дате вредности {-3, -2|0, 1, 2}, након свих могућих потеза оба играча. Скупови се своде на вредности {-2|0} и, према правилу једноставности, добија се резултат -1. У примеру 5 резултат који је добијен, има потпуно другачију вредност тако да ове две методе не дају исте резултате.

Слика 3.2 приказује скоро све о чему је било речи везано за оцењивање плаво-црвеног хакенбуша. Средишњи 'лењир' је реална оса са већим маркерима за целе бројеве док се испод линије налазе одговарајући хакенбуш ланци. На слици је приказано Аустралијско бинарно стабло од чвора -3 до чвора 4. Вредност сваког чвора се рачуна помоћу најближег левог и најближег десног, ако постоје. Почетни чвор(чвор {|}), нема ни најближи леви, ни најближи десни. Сви спољашњи чворови са десне стране, у односу на почетни чвор, имају само најближег левог. Сви спољашњи чворови са леве стране, у односу на почетни чвор, имају само најближег десног. Сви унутрашњи чворови имају и најближи леви и десни чвор. Најближи леви и десни чвор су преци тренутног чвора, један он њих је родитељ тренутног чвору, док је други родитељ првог. Вредност чвора се рачуна према правилима резимираним у табели 3.1. Редови табеле табеле редом представљају:

1. Почетни чвор,
2. чворове десно од почетног,
3. чворове лево од почетног и
4. унутрашње чворове.

Што је број једноставнији, ближи је корену стабла.

Пример 10. Чвор чија је вредност $\frac{1}{2}$ има најближе чворове {0| 1}. Чвор 1 је најближи десни чвор и родитељ чвору $\frac{1}{2}$, док је чвор 0 најближи леви чвор и родитељ чвору 1.

0	=	{ | }
n +1	=	{n | }
-n - 1	=	{ | -n}
$\frac{2p+1}{2^{q+1}}$	=	$\{\frac{p}{2^q} \mid \frac{p+1}{2^q}\}$

Табела 3.1: Најједноставнија форма бројева

Пример 11. Позиција на слици 2.1 је представљена као {-5, -7| -3}, па се на основу правила једноставности добија оцена {-5 | -3} = -4. За позицију са слике 2.3 оцена је {-1|1} = 0.
Која је оцена позиције са слике 3.3? За дато стање црвени нема много потеза, шта више, само један једини, док плави има подоста могућности.

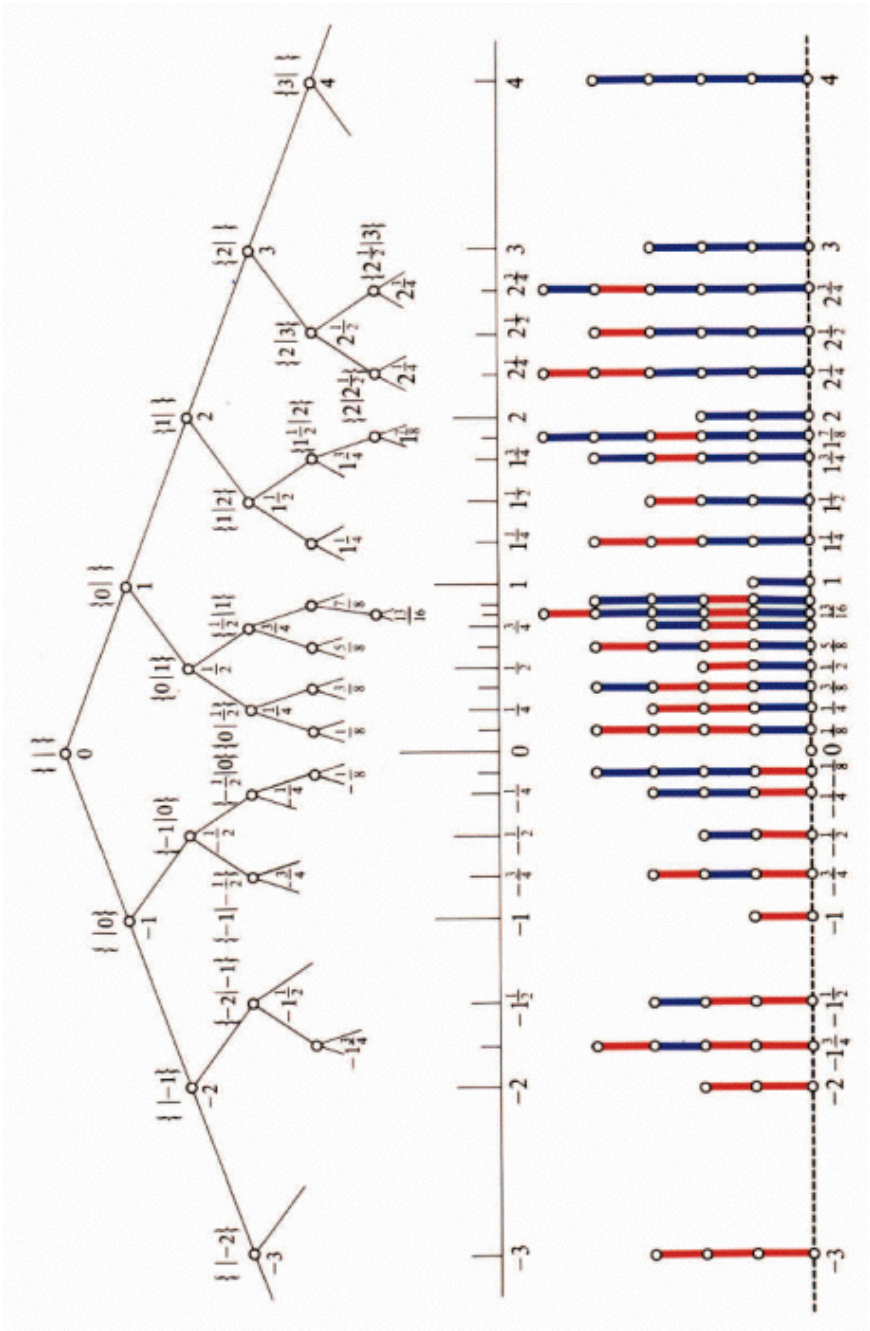
$$\{\frac{1}{2}, 4\frac{1}{2}, 6\frac{1}{2}, 7\frac{1}{2}, 8\frac{1}{2}, 9|10\} = \{9|10\} = 9\frac{1}{2}$$

Овај пример илуструје да $\{n|n+1\} = n + \frac{1}{2}$. Неколико примера, без графа, дати су у табели 3.2.

a	b	x
$2\frac{3}{4}$	$6\frac{1}{2}$	3
-5	$2\frac{5}{8}$	0
0	1	$\frac{1}{2}$
$\frac{1}{4}$	$\frac{5}{16}$	$\frac{9}{32}$
$\frac{1}{4}$	$\frac{7}{16}$	$\frac{3}{8}$
$-2\frac{7}{8}$	$-2\frac{3}{32}$	$-2\frac{1}{2}$

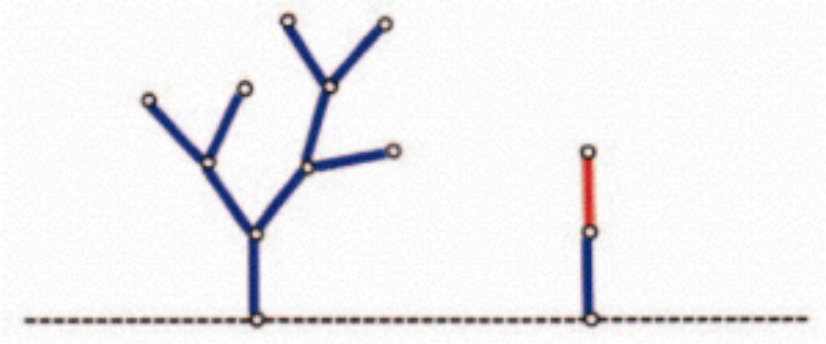
Табела 3.2: Примери примене правила једноставности

Приказани су разни графови и израчунате вредности њихових позиција. Испоставља се да се правило једноставности не може увек искористити за добијање оцене позиције. Размотримо пример на слици 3.4. Применом правила једноставности добија се {0 | -1, 2} = {0 | -1}. Према дефиницији, мора да постоји број x такав да је мањи од b=-1 и већи од a=1, али то овде није тачно.

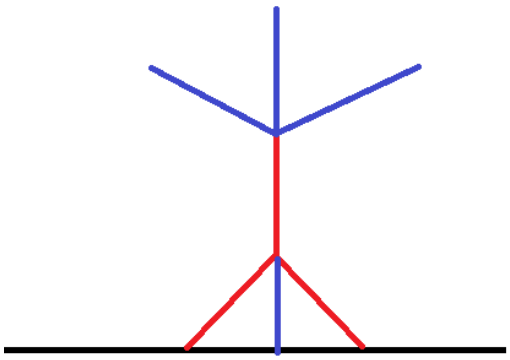


Слика 3.2: Аустралијско дрво бројева, реална оса и хакенбуш ланци

Постоји цела класа графова на које се не може применити правило једноставности. То су стабла. Срећом њихова оцена се може одредити на други начин, о чему ће бити речи у наредној тачки.



Слика 3.3: Пример оцењивања



Слика 3.4: Проблематичан граф

3.3 Рачунање оцене стабла

На слици 3.5а је једноставан пример стабла. Само једна грана додирује тло, а остатак графа је везан за ту грану, директно или индиректно. Знак и оцена стабла зависе од гране која додирује тло, тј. од боје те гране. Исто као и код ланаца, који су специјални случај стабала (плава грана је позитивна оцена, црвена је негативна оцена). За рачунање оцене стабла потребна је оцена остатка графа тј. оцена дела графа P, где је P део стабла који се добија брисањем гране до тла и повезивањем са тлом свих чворова суседних обрисаној грани. Ако је вредност P дела реалан број x и ако је грана која додирује тло плава, онда се оцена стабла означава са *1:x*. Слично, за црвену грану, оцена стабла се означава са *-1:x*

За реалан број x, број *1:x* (ординална сума 1 и x) има прву вредност из

серије:

$$\frac{x+1}{1}, \frac{x+2}{2}, \frac{x+3}{4}, \frac{x+4}{8}, \frac{x+5}{16}, ...$$

кад бројилац постане већи од 1. Именилац броја на који се ово односи је $x{+}n$ а не $\frac{x+n}{2^{n-1}}$, када се запише као најједноставнији рационални број. Слично, број *(-1):*x имаће прву вредност из серије:

$$\frac{x-1}{1}, \frac{x-2}{2}, \frac{x-3}{4}, \frac{x-4}{8}, \frac{x-5}{16}, ...$$

када бројилац постане мањи од -1[1].

У случају да се стабло рачва на два стабла P и Q, тада је вредност позиције $x{+}y$, где је x вредност за P и y је вредност за Q. Ово је илустровано на сликама 3.5b и c. Да би се добила коначна оцена стабла на овако описан начин, потребно је израчунати оцену остатка стабла, тј. P. Оцена стабла P се рачуна по истом принципу као и коначна оцена, оцена сваке гране зависи од њене боје и оцене претходне гране. Оцењивање креће од листова на доле. Листови узимају оцену 1 за плаву грану и -1 за црвену грану. Вредност листа се добија према правилима из табеле 3.1. На слици 3.6b је приказано оцењено дрво где уз сваку грану стоји оцена тренутне позиције.

Теорема 1. Оцена стабла се добија применом формуле $\frac{x\pm n}{2^{n-1}}$ на сваку грану датог стабла. Знак унутар формуле зависи од боје тренутне гране (за црвене -, а за плаве +). Формула се примењује од листова ка корену.

Доказ. За дато стабло T, вредност сваке плаве гране се рачуна по формули $\frac{x+d}{2^{d-1}}$, а вредност сваке црвене гране по формули $\frac{x-d}{2^{d-1}}$, где је x вредност претходне гране. Коначна вредност стабла је неки број v.

База индукције код стабла је случај када граф садржи само једну грану и та грана додирује тло. Као код листова, вредност је 1 за плаву, тј. -1 за црвену грану.

За индукциону хипотезу претпоставка је да стабло T има вредност v.

У индукционом кораку додаје се нова грана E, у стабло T, таква да настаје ново стабло T'. Грана E се додаје испод стабла T, тако да је то нова последња грана која улази у рачуницу при рачунању вредности стабла T'. Стабло T' има вредност v' која се рачуна по формули 1:v ако је E плава грана, односно -1:v ако је E црвена грана. Треба показати да, ако је E плава грана, вредност v' је позитивна, односно ако је E црвена грана, вредност v' је негативна.

Пре доказивања треба поновити чињеницу која је битна за касније доказивање. У формули $\frac{x \pm d}{2^{d-1}}$, бира се такво d да ако би представили $x \pm d$ као разломак $\frac{p}{q}$, где је НЗД(p,q)=1 ако је v рационалан број, такав да је p>1 за плаву грану, односно p<-1 за црвену грану.
У односу на вредност v, постоје два случаја, када је v цео број и када је v рационалан број.

Први случај је ако је v цео број. У овом случају, услов да је p веће од 1 или мање од -1, је непотребан. Сваки цео број се може представити као рационалан број чији су именилац и бројилац исти и самим тим p никад није 1 односно -1. У односу на знак броја v и знака који очекујемо да добијемо за v', постоје 4 могуће комбинације:

1. $v < 0$, $v' < 0$
   $v' = \frac{v-d}{2^{d-1}} = \frac{v'-1}{1} = v' - 1 < 0,$

2. $v < 0$, $v' > 0$
   $v' = \frac{v+d}{2^{d-1}}$, за $d = -v + 1, v' = \frac{1}{2^{d-1}} > 0,$

3. $v > 0$, $v' < 0$
   $v' = \frac{v-d}{2^{d-1}}$, за $d = -v - 1, v' = -\frac{1}{2^{d-1}} < 0$ и

4. $v < 0$, $v' < 0$
   $v' = \frac{v+d}{2^{d-1}} = \frac{v'+1}{1} = v' + 1 > 0.$

Други случај је ако је v рационалан број. Предуслов је да број v, који је облика $\frac{a}{b}$ испуњава услов НЗД(a,b)=1. Идентично као за целе бројеве, опет се дели на 4 могуће комбинације у односу на знакове бројева v и v'

1. $v < 0$, $v' < 0$
   $v' = \frac{v-d}{2^{d-1}} = \frac{-\frac{a}{b}-1}{1} = -\left(\frac{a+b}{b}\right) < 0,$

2. $v < 0$, $v' > 0$

   - Ако је $a < b$ и $b-a > 1$,онда је $d = 1$ јер $\frac{v+d}{2^{d-1}} = \frac{-\frac{a}{b}+1}{2^{1-1}} = \frac{b-a}{b} > 0$ и $p > 1$
   - Ако је $a < b$ и $b-a = 1$,онда је $d = 2$ јер за $d = 1$ добија се да је $\frac{v+d}{2^{d-1}} = \frac{b-a}{b} > 0$ али је $p = 1$. За $d = 2$ добија се $\frac{2 \cdot b-a}{2 \cdot b} > 0$
   - Ако је $a > b$ и $\lceil \frac{a}{b} \rceil \cdot b - a > 1, d = \lceil \frac{a}{b} \rceil$. Следи $\frac{v+d}{2^{d-1}} = \frac{-\frac{a}{b}+d}{2^{d-1}} = \frac{\frac{d \cdot b-a}{b}}{2^{d-1}} = \frac{d \cdot b-a}{b \cdot 2^{d-1}} = \frac{\lceil \frac{a}{b} \rceil \cdot b-a}{b \cdot 2^{d-1}} > 0$

- Ако је $a > b$ и $\lceil \frac{a}{b} \rceil \cdot b - a = 1, d = \lceil \frac{a}{b} \rceil + 1$ јер за $d = \lceil \frac{a}{b} \rceil, d \cdot b - a = 1$, а из горње тачке се види да је $p = d \cdot b - a$ тако да услов $p > 1$ није испуњен. За $d = \lceil \frac{a}{b} \rceil + 1, p = d \cdot b - a = \lceil \frac{a}{b} \rceil \cdot b + b - a = b + 1 > 0$.

3. $v > 0$, $v' < 0$

   - Ако је $a < b$ и $a - b < -1$,онда је $d = 1$ јер $\frac{v-d}{2^{d-1}} = \frac{\frac{a}{b}-1}{2^{1-1}} = \frac{a-b}{b} < 0$ и $p < -1$
   - Ако је $a < b$ и $a-b = -1$,онда је $d = 2$ јер за $d = 1$ добија се да је $\frac{v-d}{2^{d-1}} = \frac{a-b}{b} < 0$ али је $p = -1$. За $d = 2$ добија се $\frac{2 \cdot a - b}{2 \cdot b} < 0$
   - Ако је $a > b$ и a - $\lceil \frac{a}{b} \rceil \cdot b < -1, d = \lceil \frac{a}{b} \rceil$. Следи $\frac{v-d}{2^{d-1}} = \frac{\frac{a}{b}-d}{2^{d-1}} = \frac{\frac{a-d \cdot b}{b}}{2^{d-1}} = \frac{a-d \cdot b}{b \cdot 2^{d-1}} = \frac{a - \lceil \frac{a}{b} \rceil \cdot b}{b \cdot 2^{d-1}} < 0$
   - Ако је $a > b$ и a - $\lceil \frac{a}{b} \rceil \cdot b = -1, d = \lceil \frac{a}{b} \rceil + 1$ јер за $d = \lceil \frac{a}{b} \rceil, a - d \cdot b = -1$, а из горње тачке се види да је $p = a - d \cdot b$ тако да услов $p < -1$ није испуњен. За $d = \lceil \frac{a}{b} \rceil + 1, p = d \cdot b - a = \lceil \frac{a}{b} \rceil \cdot b + b - a = b + 1 > 0$.

4. $v > 0$, $v' > 0$
   $v' = \frac{v+d}{2^{d-1}} = \frac{\frac{a}{b}+1}{1} = \frac{a+b}{b} > 0$.

За оба случаја су изведени сви могући случајеви и у сваком је показано да се добија вредност одговарајућег знака, колика год она била. Крај доказа. □

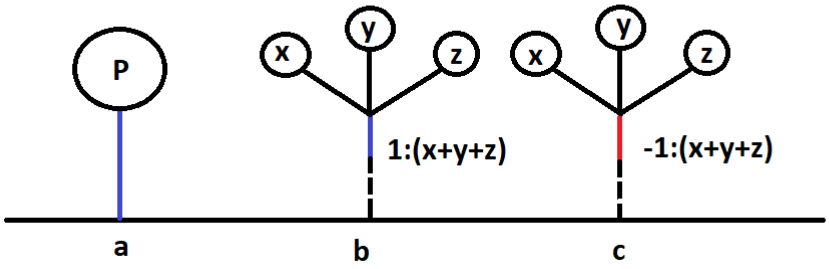
Пример 12. Граф са слике 3.6а се дели на стабло и једну плаву грану, тзв. бесплатан потез. За почетак, треба израчунати оцену стабла, а онда укупну оцену. Стабло је веома једноставно, тражени број је -1:(x+y+z)=-1:9. $\frac{9-1}{1}$, $\frac{9-2}{2} ... \frac{9-10}{512} = -\frac{1}{512}$. Дакле вредност стабла је $-\frac{1}{512}$. Да би се добила оцена игре, потребно је сабрати оцене свих стабала, $1 + \left(-\frac{1}{512}\right) = \frac{511}{512}$.

За слику 3.5а рачунање броја *1:x* може се записати као 1:x = $\{0, 1{:}x^L \mid 1{:}x^R\}$. 0 је одсецање плаве гране која везује за тло а x^L и $1{:}x^R$ су одсецања неке плаве односно црвене гране. Ова функција мапира све бројеве у позитивне по принципу једноставности. Дакле 0 као најједноставнији број мапира 1, најједноставнији позитиван број. -1 мапира најједноставнији позитиван број лево од 1 што је у овом случају $\frac{1}{2}$. Све ово се може видети на бинарном стаблу на слици 3.2. Пошто су позитивне вредности оно што тражимо за $x>=0$ прати се десна страна без икаквог рачвања, док за $x<0$ рачва се на првом левом чвору након чега се прати десна страна без било каквог додатног рачвања.

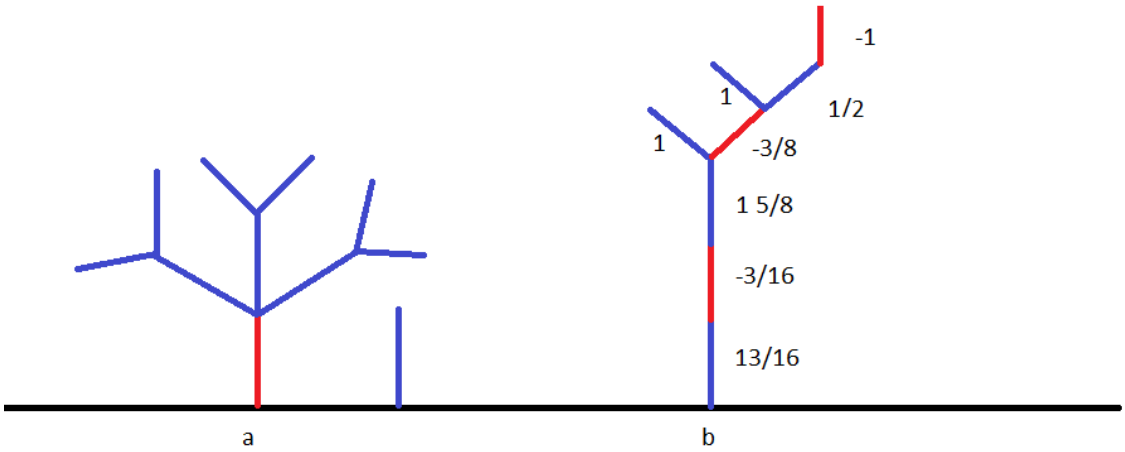
За *-1:x* је идеја иста с тим да је пут којим се иде инверз од 1:x. Лакши приказ претходног тврђења се може видети у табели 3.3.

x	=	-5	-4	-3	-2	-1	0	1	2	3	4
1:x	=	$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$	1	2	3	4	5

Табела 3.3: Табела позитивних вредности за целе бројеве



Слика 3.5: Основе рачунања стабла



Слика 3.6: Пример стабла

Глава 4

Програмска реализација

Програм демонстрира оцењивање тренутне позиције у игри плаво-црвеног хакенбуша као и слободну игру за два играча. Програм је написан у програмску језику *Python* коришћењем стандардне библиотеке.

4.1 Структура програма

Програм се дели на три целине:

- Дефинисање структура,
- Рачунање позиције и
- Слободна игра два играча.

Дефинисање структура

Две класе које се дефинишу у датој имплементацији су грана графа и граф. Класа грана поред дефиниције нема никакве методе. Класа је сачињена од првог и другог чвора и боје гране.

```
1 class Edge:
2     def __init__(self, color, start, end):
3         self.color = color
4         self.v1 = start
5         self.v2 = end
```

Класа граф садржи листу грана, број плавих грана, број црвених грана и речник чворова. За сваки граф мора да постоји чвор који представља тло. У овој имплементацији, тај чвор је '0'. Не постоји могућност да се другачије назове чвор тла осим '0'. Сви остали чворови немају ограничења у смислу како се могу представити тј. назвати. Чланови листе грана су типа *Edge*. Речник чворова за сваки чвор садржи листу његових суседа. У овом речнику кључ је чвор, а вредност је листа суседа. Речник се генерише приликом иницијализације графа. Ако је чвор лист, његов једини сусед је његов отац.

```
class Graph:
    def __init__(self):
        self.edges = []
        self.blue = 0
        self.red = 0
        self.nodes = {}
```

Следи код и објашњене метода графа: свака нова грана графа се додаје у листу грана графа, а у односу на боју нове гране се увећава одговарајући бројач.

```
def add_edge(self, node_from, node_to, color):
    edge = Edge(color, node_from, node_to)
    self.edges.append(edge)
    if color == 'blue':
        self.blue += 1
    else:
        self.red += 1
```

Имплементација алгоритма *DFS* прати објашњење које је дато раније у тексту. Полази се од првог наведеног чвора и рекурзивно се прате његови суседи докле год постоји сусед који није у листи посећених чворова.

```
def DFS(self, node, visited):
    if node not in visited:
        visited.append(node)
        for node2 in self.nodes[node]:
            self.DFS(node2, visited)
```

Брисање гране, *remove_edge*, се извршa коришћењем две методе. Прва метода, *delete_edge*, брише задату грану из графа и ажурира речник суседа. Друга метода, *clean_neighborhood*, помоћу *DFS*-а одређује све гране које више нису унутар графа. Ако грана више није део графа, она се такође брише. Након ове методе за граф важи да постоји пут од тла до сваког чвора.

```
def remove_edge(self, v1, v2):
    self.delete_edge(v1, v2)
    self.clean_neighborhood()

def delete_edge(self, v1, v2):
    for edge in self.edges:
        if edge.v1 == v1 and edge.v2 == v2:
            self.edges.remove(edge)
            self.remove_neighbor(v1, v2)
            if edge.color == 'blue':
                self.blue -= 1
            else:
                self.red -= 1

def clean_neighborhood(self):
    visited = []
    self.DFS('0', visited)
    for edge in self.edges.copy():
        if edge.v1 not in visited or edge.v2 not in visited:
            self.delete_edge(edge.v1, edge.v2)
```

Рачунање оцене позиције

Функција *Hackenbush* позива функције за рачунање оцене позиције у односу на почетно стање графа. Пошто почетни граф може бити само ланац или само стабло, има смисла позвати директно функцију која рачуна оцену за тај случај. Стабло, ланац и обичан граф имају засебне начине рачунања који су изложене у глави 3. Ланац и стабло имају заједничко то да их везује само једна грана за тло тако да се та два случаја касније раздвајају. С обзиром да функција *Simplicity* враћа оцену позиције и најбоље потеза за оба играча, исто мора да се уради и у случају ланца/стабла.

```
def Hackenbush(g):
```

```
    if len(g.nodes['0']) > 1:
        return Simplicity(g)
    else:
        red = ''
        blue = ''
        for edge in g.edges:
            if edge.v1 == '0' or edge.v2 == '0':
                if edge.color == blue:
                    blue = edge.v1 + '-' + edge.v2
                    for edge2 in g.edges:
                        if edge2.color == 'red':
                            red = edge2.v1 + '-' + edge2.v2
                            break
                else:
                    red = edge.v1 + '-' + edge.v2
                    for edge2 in g.edges:
                        if edge2.color == 'blue':
                            blue = edge2.v1 + '-' + edge2.v2
                            break
                break
        return chains(g, g.nodes['0']), blue, red
```

Функција *Simplicity* за дати граф одређује сваки могући потези левог и десног играча. Свака грана из графа се избацује и рачуна се оцена тренутне позиције. Чувају се само прва појављивања сваке оцене. Након обраде свих грана, тј. свих могућих позиција, над скупом оцена се примењује правило једноставности, као што је описано дефиницијом 2, позивом функције *int_ number*, тј. функције *real_ number*, у зависности од случаја. Током проласка кроз граф, памте се тренутне најбоље оцене потеза који оба играча могу да одиграју. Када је обилазак графа готов и када је коначна оцена одређена, функција враћа информацију о оцени и најбољим потезом за оба играча.

```
def Simplicity(g):
    blue_values = []
    red_values = []
    value = 0
    blue_best = -10000000000
    red_best = -10000000000
    red_branch = ''
    blue_branch = ''
    for edge in g.edges:
```

```
10          temp = copy.deepcopy(g)
11          temp.remove_edge(edge.v1, edge.v2)
12          zero_nodes = chain_nodes(g, temp.nodes['0'])
13          color = g.find_color(edge.v1, edge.v2)
14          if len(zero_nodes) > 0:
15              value += chains(temp, zero_nodes)
16          for node in zero_nodes:
17              temp.remove_edge('0', node)
18          value += temp.blue - temp.red
19          if value > red_best and color == 'red':
20              red_best = value
21              red_branch = edge.v1 + '-' + edge.v2
22          if value > blue_best and color == 'blue':
23              blue_best = value
24              blue_branch = edge.v1 + '-' + edge.v2
25          if edge.color == 'blue' and value not in blue_values:
26              blue_values.append(value)
27          elif edge.color == 'red' and value not in red_values:
28              red_values.append(value)
29          value = 0
30      if len(red_values) > 0:
31          b = min(red_values)
32      else:
33          b = 0
34      if len(blue_values) > 0:
35          a = max(blue_values)
36      else:
37          a = 0
38      a = max(blue_values)
39      if a == b:
40          result = 0
41      elif math.fabs(a - b) > 1:
42          if b < 0:
43              result = int_number(a * -1, b * -1) * -1
44          else:
45              result = int_number(a, b)
46      else:
47          x1, y1 = a.as_integer_ratio()
48          x2, y2 = b.as_integer_ratio()
49          x1 *= y2 / y2
50          if b < 0:
51              result = real_number(x2 * -1, x1 * -1, y2) * -1
```

```
52          elif b == 0:
53              result = real_number(x2, x1 * -1, y1) * -1
54          else:
55              result = real_number(x1, x2, y2)
56      return result, blue_branch, red_branch
```

Функцији *chains* прихвата два аргумента, граф и листу чворова. Чворови дате листе морају да испуњавају два услова:

- да у листи суседа имају чвор тла и
- да не постоји пут од датог чвора до било ког другог чвора из листе суседа чвора тла.

Помоћу функције *is_ tree* се одређује да ли се позива функција за рачунање вредности стабла или вредности ланца. Принцип обиласка графа је попут *DFS*-а. Проласком кроз граф, у листу посећених чворова се додаје први чвор сваке гране. Пре додавања у листу посећених чворова, проверава се да ли се тренутни чвор већ налази у листи. У случају да се испуни овај услов и да чвор је већ у листи, значи да постоји рачвање из неке гране и да се рекурзија вратила назад на већ посећен чвор. То значи да је у питању стабло. Ако се овај услов не испуни током обиласка графа, значи да је у питању ланац. У зависности од одговора функције, позива се одговарајућа функција за даље рачунање. Сви међурезултати се сабирају и дају укупну оцену свих стабала и/или ланаца унутар графа.

```
1  def chains(g, nodes):
2      value = 0
3      for node in nodes:
4          visited = []
5          if is_tree(g, node, visited):
6              nodes2 = copy.deepcopy(nodes)
7              nodes2.remove(node)
8              tree = copy.deepcopy(g)
9              for node2 in nodes2:
10                 tree.remove_edge('0', node2)
11             value += tree_value(tree, node, '0', [])
12         else:
13             visited = []
14             colors = []
```

```
            chain_colors(g, node, colors, visited)
            value += chain_value(colors)

    return value

def is_tree(g, node, visited):
    if node in visited:
        return True
    else:
        visited.append(node)
    for edge in g.edges:
        if edge.v1 == node:
            is_tree(g, edge.v2, visited)
    return False
```

За дати почетни чвор ланца одређују се боје грана које га чине, на основу којих се добија и вредност ланца према Теа ван Рудововој методи. Принцип прикупљања боја грана ланца је сличан *DFS*-у. Обе функције се позивају унутар функције *chains*. Резултат *chain_ colors* је улаз за *chain_ value*.

```
def chain_colors(g, node, colors, visited):
    if node not in visited:
        visited.append(node)
        for edge in g.edges:
            if edge.v1 == node or (edge.v2 == node and edge.v1=='0'):
                colors.append(edge.color)
                chain_colors(g, edge.v2, colors, visited)

def chain_value(colors):
    color = colors[0]
    index = 0
    while len(colors) > index and color == colors[index]:
        index += 1
    value = index
    if color == 'red':
        value = value * -1
    power = -1
    while index < len(colors):
        if colors[index] == 'blue':
            value += pow(2, power)
        else:
            value -= pow(2, power)
```

```
        power -= 1
        index += 1
    return value
```

Рачунање вредности стабла се ради рекурзивном функцијом. Излаз из рекурзије је обрада листа. Функција *tree_ value* врши обилазак стабла од корена до сваког листа. Док год не постоји рачвање, пролаз кроз стабло је итеративан и памте се боје грана. Када се наиђе на рачвање, рекурзивно се позива функција за свако од подстабла. Сваки међурезултат се рачуна помоћу функције *fork_ value*. У поглављу 3.3 је детаљно представљен начин на који се рачуна вредност стабла.

```
def tree_value(g, node, previous, color):
    colors = []
    if len(g.nodes[node]) > 2:
        temp = 0
        for neighbor in g.nodes[node]:
            if neighbor != previous:
                temp += tree_value(g, neighbor, node, g.find_color(
   node, neighbor))
        return fork_value(temp, color)
    elif len(g.nodes[node]) == 2:
        node2 = node
        if g.nodes[node][0] != '0' and g.nodes[node][1] != '0':
            colors.append(color)
        else:
            colors.append(g.find_color('0', node))
        while len(g.nodes[node2]) == 2:
            if previous == g.nodes[node2][0]:
                tempnode = g.nodes[node2][1]
            else:
                tempnode = g.nodes[node2][0]
            previous = node2
            colors.append(g.find_color(node2, tempnode))
            node2 = tempnode
        temp = tree_value(g, node2, previous, colors.pop())
        colors.reverse()
        for color in colors:
            temp = fork_value(temp, color)
    elif len(g.nodes[node]) == 1:
        if color == 'blue':
```

```
29              return 1
30          else:
31              return -1
32
33      return temp
34
35  def fork_value(value, color):
36      if color == 'blue':
37          sign = 1
38      else:
39          sign = -1
40      i = 1
41      x, y = value.as_integer_ratio()
42      while True:
43          result = (value + sign * i)/math.pow(2, i-1)
44          x, y = (value + sign * i).as_integer_ratio()
45          if x > 1 and sign == 1:
46              break
47          elif x < -1 and sign == -1:
48              break
49          i += 1
50      return result
```

Слободна игра два играча

Аргументи функције су граф и ко је први на потезу од играча. Са стандардног улаза се читају одговори играча. Подржани су наредни одговори:

- Грана из графа написана у формату x-y, коју играч жели да уклони. У случају да грана не постоји или је грана коју тренутни играч не може да обрише, програм исписује одговарајућу поруку о грешци.
- 'pomoc' ово је наредба да програм израчуна најбољи потез који дати играч може да одигра.
- 'stop' ово је наредба да програм прекине извршавање.

Одговор играча представља грана коју играч жели да избрише. У случају да играч одабере грану која не постоји или нема право да је уклони, добија одговарајућу поруку. Док год оба играча имају гране у графу, игра се наставља. Унос 'stop' је специјалан унос којим се прекида игра. Унос 'pomoc' је

специјалан унос којим играч од

```
def play_game(g, m):
    moves = m
    print(graph(g))
    end = False
    poruka = 'Na potezu je '
    if moves == 'L':
        poruka += 'levi'
    else:
        poruka += 'desni'
    print(poruka)
    for line in sys.stdin:
        found = False
        line = line.strip()
        if line == 'stop':
            print('Prekidanje igre...')
            break
        if line == 'pomoc':
            v, b, r = Hackenbush(g)
            if moves == 'L':
                print('Najbolji potez je ' + b)
                continue
            else:
                print('Najbolji potez je ' + r)
                continue
        v1 = line.split("-")[0]
        v2 = line.split('-')[1]
        for edge in g.edges:
            if edge.v1 == v1 and edge.v2 == v2:
                if moves == 'L' and edge.color == 'red':
                    print('Pogresan potez. Mozete izabrati samo neku
    od plavih grana! Pokusajte ponovo.')
                    found = True
                    break
                elif moves == 'R' and edge.color == 'blue':
                    print('Pogresan potez. Mozete izabrati samo neku
    od crvenih grana! Pokusajte ponovo.')
                    found = True
                    break
                g.remove_edge(v1, v2)
                if g.blue == 0:
```

```
39                      print('Desni je pobednik!')
40                      end = True
41                      break
42                  elif g.red == 0:
43                      print('Levi je pobednik!')
44                      end = True
45                      break
46                  print(graph(g))
47                  if moves == 'L':
48                      moves = 'R'
49                  else:
50                      moves = 'L'
51                  found = True
52                  break
53          if end:
54              break
55          if not found:
56              print('Grana koju ste izabrali ne postoji! Pokusajte
    ponovo.')
57              continue
58          poruka = 'Na potezu je '
59          if moves == 'L':
60              poruka += 'levi igrac.'
61          else:
62              poruka += 'desni igrac.'
63          print(poruka)
```

4.2 Коришћење програма

Предуслов за коришћење програма је да постоји текстуална датотека са улазним подацима за генерисање графа. Потребно је да датотека садржи n линија у којима се налазе имена чворова који чине грану и боја дате гране. Наредни код приказује функцију читања датотеке и формирања графа.

```
1 def initialize_graph(edges_file):
2     file = open(edges_file, "r")
3     g = Graph()
4     for line in file:
5         temp = line.strip().split(' ')
6         g.add_edge(temp[0], temp[1], temp[2])
```

```
        if temp[0] in g.nodes.keys():
            g.nodes[temp[0]].append(temp[1])
        else:
            g.nodes[temp[0]] = []
            g.nodes[temp[0]].append(temp[1])
        if temp[1] not in g.nodes.keys():
            g.nodes[temp[1]] = []
    file.close()
    return g
```

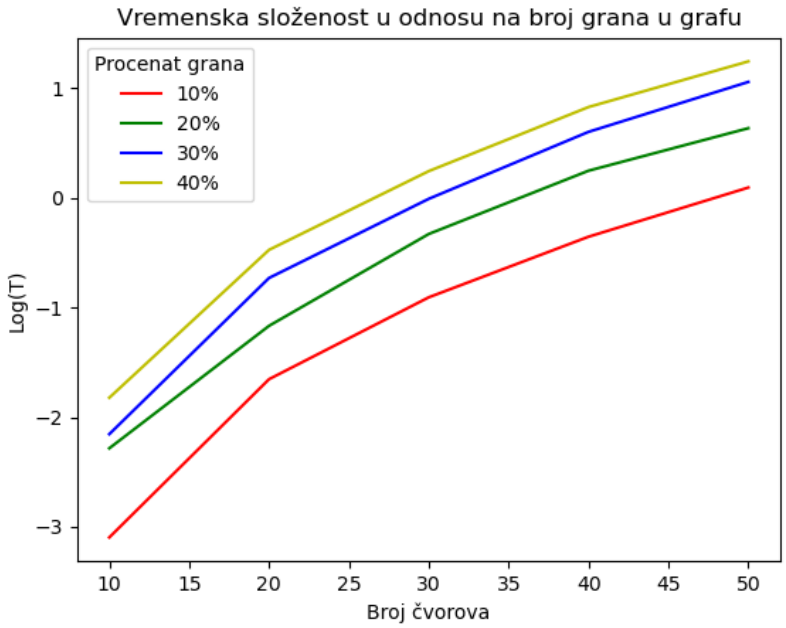
При покретању програма се прослеђује се локација датотеке помоћу које се формира граф. Затим се даје одговор *Slobodna igra* за два играча или *Vrednost pozicije* за рачунање оцене тренутне позиције. При одабиру *Slobodna igra*, наводи се ко игра први са *levi* за левог и *desnog* за десног. Након тога наизменично играчи дају наизменично одговоре. Опис могућих одговора је дат у објашњењу имплементације ове функције. Пример одигране игре уз асистенцију је дат на слици 4.5. Одабиром *Vrednost pozicije*, програм израчунава вредност позиције за унети граф и исписује добијену вредност на стандардни излаз без даље интеракције корисника.

```
def main():
    edges_file = sys.stdin.readline().strip()
    g = initialize_graph(edges_file)
    while True:
        print("Odaberi zeljeni tip.")
        game_mode = sys.stdin.readline().strip()
        if game_mode == 'Slobodna igra':
            print('Odaberite ko igra prvi, levi ili desni:')
            moves = ''
            if sys.stdin.readline().strip() == 'levi':
                moves = 'L'
            elif sys.stdin.readline().strip() == 'desni':
                moves = 'R'
            else:
                print('Pogresan unos. Pokusajte ponovo.')
                continue
            play_game(g, moves)
            break
        elif game_mode == 'Vrednost pozicije':
            start = time.time()
            print('Vrednost pozicije za dati graf je:', Hackenbush(g))
            end = time.time()
```

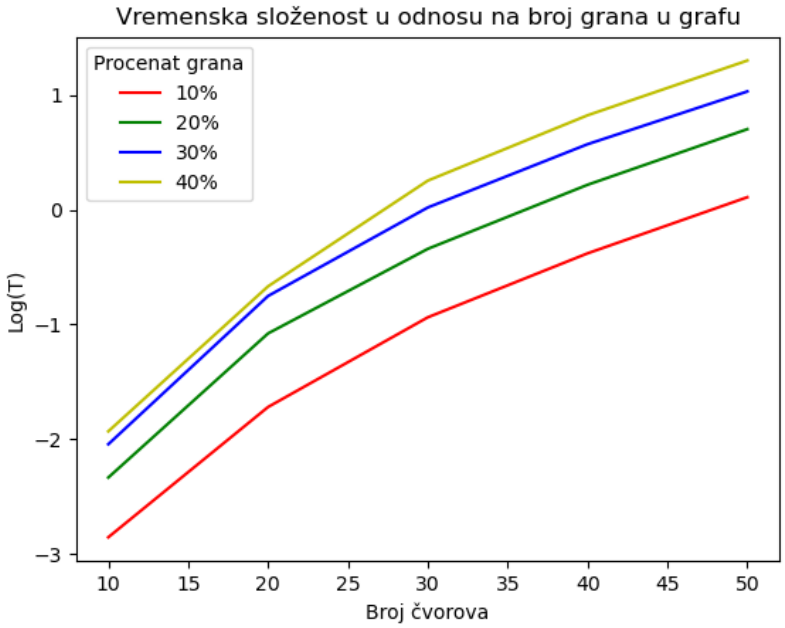
```
23              print((end-start)/60)
24              break
25          else:
26      print("Pogresno uneta vrednost. Podrzane vrednosti su: Vrednost
        pozicije i Slobodna igra")
```

4.3 Резултати тестирања

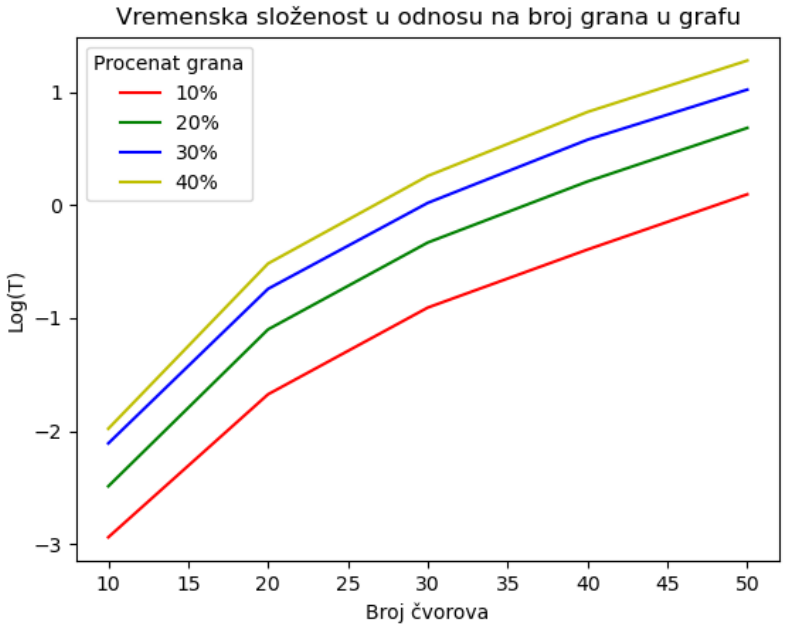
За тестирање програма генерисани су насумични графови. Насумичан граф садржи n чворова и $a \cdot \frac{(n+1)\cdot n}{2}$ грана, где је a параметар који узима вредност процента грана које су, од $n\frac{(n+1)\cdot n}{2}$ могућих, део графа. Проценат a узима вредност из скупа $\{10\%, 20\%, 30\%, 40\%\}$, док n узима вредност из скупа $\{10, 20, 30, 40, 50\}$. Крива на сваком дијаграму представља зависност времена у односу на раст броја чворова. Свака крива одговара једној од вредности параметра a. За свако n и a, генерише се κ графова и узима просек њиховог извршавања. Следи приказ дијаграма за различите вредности κ. Циљ је видети да ли и на који начин κ утиче на криву.



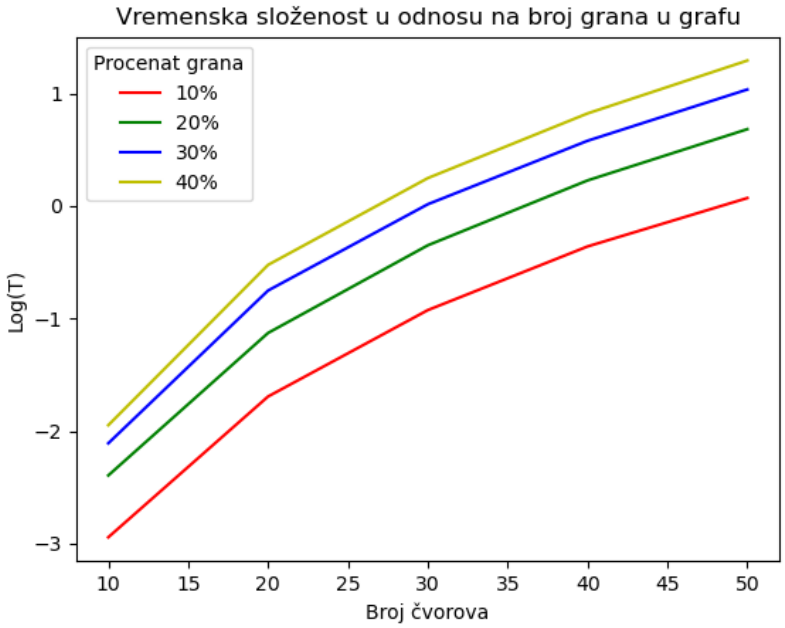
Слика 4.1: Групе од по 5 графова



Слика 4.2: Групе од по 10 графова



Слика 4.3: Групе од по 20 графова



Слика 4.4: Групе од по 50 графова

Може се приметити да криве имају идентично понашање за различито κ и да већи број графова по групи не утиче значајно на криву. Приметно је да граф одступа од експоненцијалне функције од времена на сваком дијаграму.

4.4 Ограничења

Приликом рачунања оцене тренутне позиције постоје нека ограничења:

- Ако стабло садржи циклус, то није валидно стабло и за такво стабло се не може добити оцена.
- Ланац чији почетак и крај су везани за тло, нема оптимизовано рачунање.
- Случај графа са слике 3.4 није теоријски покривен, тако да имплементација не даје тачне резултате.
- Не постоји детекција претходно описаних ограничења.

Ограничење рачунања оцене позиције зависи од типа графа који се даје као улаз. За ланац ограничење је око 1000 грана јер је ограничење програмског

језика *Python* 1000 рекурзивних позива. Ово ограничење се може повећати, али већ на 1200 грана се долази до прокорачења стека (енгл. *stack overflow*). За генерисање графа коришћена је функција која насумично додаје гране које не постоје у графу док се не достигне жељени број грана. Из тог разлога увек може да постоји вероватноћа да ће се генерисати граф за који постоји подграф, који се добија уклањањем једне гране, такав да у себи садржи ланац од преко 1000 грана што доводи до претходно напоменутог проблема.

```
C:\Users\Sasa\anaconda3\python.exe C:/Users/Sasa/Documents/Master/Hackenbush.py
Odaberi zeljeni tip.
Slobodna igra
Odaberite ko igra prvi, levi ili desni:
levi
['0-1:red', '0-2:blue', '0-3:red', '1-3:red', '1-2:blue', '2-3:red', '2-4:red', '2-5:red', '3-6:blue']
Na potezu je levi
pomoc
Najbolji potez je 0-2
0-2
['0-1:red', '0-3:red', '1-3:red', '1-2:blue', '2-3:red', '2-4:red', '2-5:red', '3-6:blue']
Na potezu je desni
pomoc
Najbolji potez je 0-1
0-1
['0-3:red', '1-3:red', '1-2:blue', '2-3:red', '2-4:red', '2-5:red', '3-6:blue']
Na potezu je levi
pomoc
Najbolji potez je 1-2
1-2
['0-3:red', '1-3:red', '2-3:red', '2-4:red', '2-5:red', '3-6:blue']
Na potezu je desni
pomoc
Najbolji potez je 0-3
0-3
Desni je pobednik!

Process finished with exit code 0
```

Слика 4.5: Симулација игре

Глава 5

Закључак

У овом овом раду је приказана комбинаторна игра за два играча, плаво-црвени хакенбуш. За репрезентацију цртежа игре коришћени су графови зарад лакшег теоријског представљања и имплементације. Дато је објашњење за рачунање позиције описаних типова графа. За имплементацију је коришћен програмски језик Python и имплементацијом су покривена сви типови графа уз одређена ограничења. Програм подржава могућност слободне игре два играча.

Рад се може проширити другим варијантама игре хакенбуш попут зеленог хакенбуша и плаво-зелено-црвеног хакенбуша.

Литература

[1] John Horton Conway. In *On Numbers and Games*, 2007.

[2] John H. Conway and Richard K. Guy Elwyn R. Berlekamp. In *Winning Ways for Your Mathematical Plays*, 2002.

[3] David Wolfe Michael H. Albert, Richard J. Nowakowski. In *Lessons in play*, 2002.

[4] Миодраг Живковић. In *Algoritmi*. Математички факултет, 2000.