Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Losovanie turnajov švajčiarskym systémom

Bakalárska práca

2023
Martin Hošek

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Losovanie turnajov švajčiarskym systémom

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. Mgr. Tomáš Plachetka, Dr.

Bratislava, 2023
Martin Hošek

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Martin Hošek
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Losovanie turnajov švajčiarskym systémom
Swiss pairing for tournaments

Anotácia: V tejto práci sa študujú zásady losovania turnajov švajčiarskym systémom a rôzne varianty systému losovania. Špecifikujú sa presné pravidlá a preferencie pre vybraný variant. Implementuje sa program pre losovanie a porovnáva sa s existujúcimi implementáciami.

Vedúci: doc. Mgr. Tomáš Plachetka, Dr.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
archív

Dátum zadania: 12.10.2022

Dátum schválenia: 13.10.2022 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcel by som sa poďakovať svojmu školiteľovi doc. Mgr. Tomášovi Plachetkovi, Dr. za vedenie bakalárskej práce, za cenné rady a informácie, ktoré som pri písaní práce, ale aj pri programovaní využil. V neposlednom rade by som sa chcel poďakovať aj za ochotu a čas, ktorý mi venoval.

Abstrakt

V tejto bakalárskej práci robíme stručný prehľad a analýzu losovacích systémov, ktoré sa využívajú v turnajoch. Podrobnejšie sa venujeme švajčiarskemu systému, explicitne definujeme povinné a preferenčné pravidlá, ktoré platia v švajčiarskom systéme. Ďalej popisujeme problém losovania švajčiarskym systémom, definujeme čo je na vstupe a čo na výstupe. Interpretujeme štandardný formát .trf, ktorý sa využíva na ukladanie záznamu o turnaji. Vysvetľujeme, ako sa dá problém losovania švajčiarskym systémom transformovať na problém hľadania maximálneho párovania v neorientovanom grafe. Využívame implementáciu algoritmu Blossom, ktorý rieši problém maximálneho váhovaného párovania. V stručnosti robíme zhrnutie verzií implementácií algoritmu Blossom. Analyzujeme existujúce implementácie losovania švajčiarskym systémom: JaVaFo v jazyku Java a bbpPairings v jazyku C++, porovnávame náš algoritmus s týmito existujúcimi algoritmami.

Kľúčové slová: Maximum weighted matching, švajčiarsky systém, losovacie systémy v turnajoch, algoritmus Blossom, párovanie hráčov

Abstract

In this bachelor's thesis, we provide a brief overview and analysis of draw systems that are used in tournaments. We take a closer look at the mandatory and preferential rules that apply in the Swiss system. Next, we describe the problem of pairing in the Swiss system and define what should be at the input and what should be at the output. We interpret the standard .trf format used to store the tournament record. We explain how the Swiss pairing problem can be transformed into a maximum weighted matching problem in an undirected graph. We use an implementation of the Blossom algorithm that solves the maximum weighted matching problem. Briefly, we summarize the versions of the implementation of the Blossom algorithm. We analyze the existing implementations of Swiss pairing: JaVaFo in Java and bbpPairings in C++, comparing our algorithm with these existing algorithms.

Keywords: Maximum weighted matching, swiss system, tournament pairing system, blossom algorithm, pairing of players

Obsah

Úvod

V tejto práci sa budeme primárne venovať švajčiarskemu systému, pričom opíšeme aj iné systémy losovania turnajov, ktoré sa využívajú. Analyzujeme výhody a nevýhody konkrétnych losovacích systémov a predstavíme algoritmus, ktorý sme vytvorili pre losovanie švajčiarskym systémom, porovnáme ho s existujúcim algoritmom [17] a ukážeme aké knižnice sme v našej implementácii používali.

V prvej kapitole zadefinujeme základné pojmy, ktoré budeme v práci používať, porovnáme losovací systém *round-robin*, *vyraďovací systém* a *švajčiarsky systém*, neformálne zadefinujeme pravidlá losovania švajčiarskym systémom. Problém švajčiarskeho systému spočíva vo vygenerovaní nasledujúceho kola na základe predchádzajúcich odohraných kôl. Vstupom pre tento problém je stav turnaja, ktorý obsahuje relevantné informácie o všetkých predošlých kolách (kto s kým hral, akú mal farbu, aký bol výsledok, kto má koľko bodov). Na uloženie stavu turnaja sa používa štandardizovaný formát TRF [12].

V kapitole 2 presne definujeme problém losovania, definujeme, čo má byť na vstupe, čo na výstupe a explicitne definujeme pravidlá, podľa ktorých sa losuje ďalšie kolo (povinné a preferenčné pravidlá).

V kapitole 3 redukujeme problém losovania švajčiarskym systémom na problém hľadania maximálneho párovania (Maximum Weighted Matching) v neorientovanom neohodnotenom alebo ohodnotenom grafe. Ukážeme, ako sme v našej implementácii využili algoritmus *Blossom*, ktorý rieši tento problém a existuje k nemu aj implementácia v jazyku Java a C++. Spravíme stručný prehľad verzií implementácie algoritmu Blossom. Popíšeme algoritmy redukcie a porovnáme verzie algoritmov (implementácii) na losovanie, s ktorými sme pracovali.

V kapitole 4 popíšeme dôležité detaily implementácie, aké dátové štruktúry sme používali, ako načítame vstup a vypíšeme výstup. Ďalej ukážeme, ako sme využívali implementáciu algoritmu Blossom (BLOSSOM V), aké poskytuje funkcie a ako treba interpretovať výstup BLOSSOM V.

Na konci práce (v kapitole 5) porovnáme viaceré verzie našej implementácie s existujúcou implementáciou $JaVaFo$ a $bbpPairings$.

V prílohe je postup ako spustiť a ako používať našu implementáciu a prehľad, čo obsahuje elektronická príloha.

Kapitola 1

Losovacie systémy

Turnaj je súťaž, ktorá pozostáva z konečného, vopred určeného počtu *kôl*. Kolá tvoria postupnosť, napr. druhé kolo nezačne pred koncom prvého kola.

V každom kole sa hrajú *partie* (*zápasy*) medzi usporiadanými dvojicami hráčov (prirodzene, každý hráč hrá v danom kole najviac jednu partiu). Týmto dvojiciam budeme hovoriť *páry*. V ďalšom texte budeme pod hráčom rozumieť v prípade kolektívnych súťaží tím (zaujímať nás budú len tímy ako celky).

Budeme používať šachovú terminológiu, t.j. o prvom hráčovi v páre budeme hovoriť, že hrá s *bielou* farbou a druhý hrá s *čiernou* (tieto farby napr. v prípade futbalu určujú, kto hrá „doma“ a kto „vonku“). Ak hráč v danom kole nemá súpera, t.j. *je nespárovaný*, tak v tom kole nemá ani žiadnu farbu.

Každá partia končí nejakým *výsledkom*, napr. v šachu sú možné tri výsledky: biely vyhral, čierny vyhral, remíza.

Pod *losovaním kola* r budeme rozumieť množinu párov, ktoré určujú, kto bude s kým hrať v kole r.

Pod *stavom turnaja* po kole r budeme rozumieť dátovú štruktúru, v ktorej sú zapamätané páry v kolách $1 \ldots r$ spolu s výsledkami jednotlivých partií. Stav turnaja môže tiež obsahovať dodatočné informácie ako napr. názov turnaja, mená hráčov, rating hráčov a pod.

Losovací systém je funkcia, ktorá stavu turnaja po kole r priradí množinu párov, ktoré zápasia v kole $r + 1$.

Počet párov v rôznych kolách môže byť rôzny (napr. pri vyraďovacích losovacích systémoch počet dvojíc v jednotlivých kolách klesá).

Počet kôl býva spravidla známy ešte pred začiatkom turnaja.

Pod pojmom *losovanie* alebo *párovanie* teda budeme rozumieť vygenerovanie konkrétneho (nasledujúceho) kola s dodržaním pravidiel pre konkrétny losovací systém.

1.1 Round-robin systém

Ako prvý spomenieme systém *round-robin* (systém každý s každým) [5]. Losovanie round-robin sa využíva v rôznych športoch, napríklad vo futbale. V tomto systéme hrá každý hráč s každým iným hráčom v priebehu turnaja práve raz. V prípade nepárneho počtu hráčov je v každom kole nespárovaný jeden hráč.

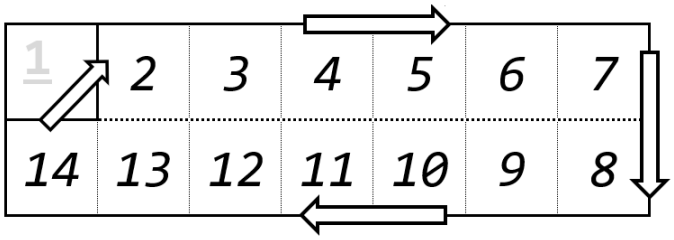
Počet odohraných zápasov v tomto systéme rastie kvadraticky od počtu hráčov, keďže hrá každý s každým. Spolu je teda odohraných $\frac{n(n-1)}{2}$ zápasov resp. $n(n-1)$ zápasov v prípade *double round-robin* (double round-robin znamená, že každý hráč hrá s každým práve dvakrát). V prípade párneho počtu hráčov sa hrá $\frac{n}{2}$ zápasov v $n-1$ kolách, v prípade nepárneho počtu hráčov sa hrá $\frac{n-1}{2}$ zápasov v každom z n kôl. Víťaz je ten hráč, ktorý vyhral najviac zápasov (okrem prípadov, keď sú umožnené remízy).

Nevýhoda tohto systému je veľký počet kôl pri veľkom počte hráčov. Pri malom počte hráčov môže nastať „cyklická“ situácia, napr. pre troch hráčov A, B, C: odohrajú sa tri zápasy *A-B*, *B-C*, *B-C*, kde v prvom zápase vyhrá A, v druhom B, a v treťom C. V tomto prípade bude mať každý hráč rovnaký počet výhier a prehier.

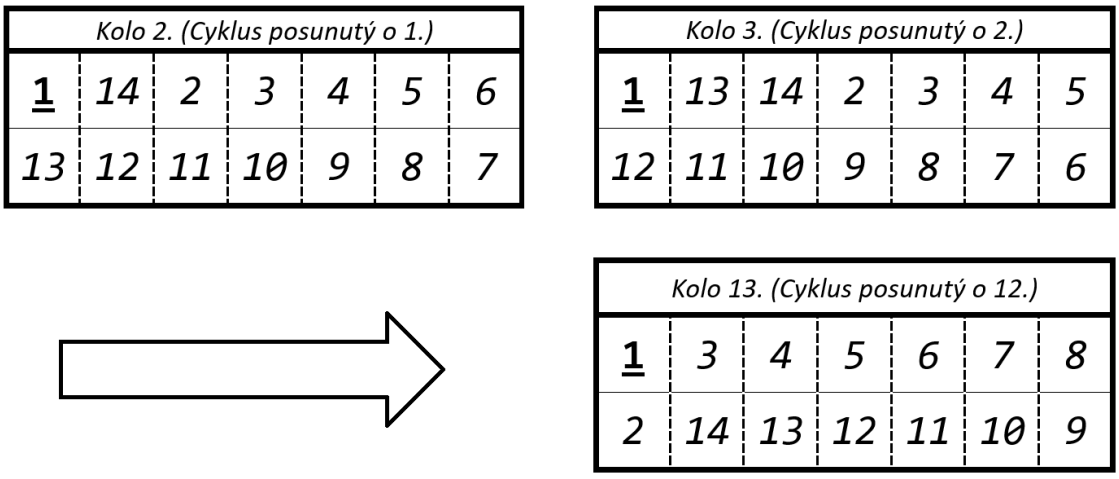
Teraz si v krátkosti opíšeme, akým systémom prebieha losovanie v tomto losovacom systéme. Na rozdiel od švajčiarskeho systému má tento systém presne stanovené, kto s kým má hrať a preto je algoritmus losovania jednoduchší. V nasledujúcom texte budeme číslovať hráčov od 1 po n, kde n je celkový počet hráčov. Ak je dostatok voľných hracích miest, tak všetky zápasy v jednom kole môžu byť hrané súčasne.

Originálnu konštrukciu párovacích tabuliek vyvinul v roku 1886 Richard Schurig a bola nasledovná: Nech n je párny počet hráčov, resp. $n-1$ je nepárny počet hráčov. Vytvoríme si tabuľku veľkosti $\frac{n}{2}$ stĺpcov a $n-1$ riadkov. Každá bunka tabuľky bude obsahovať jeden pár: prvého a druhého hráča z páru. Prvého hráča z páru vyplníme do tabuľky tak, že postupne od ľavého horného rohu štandardným spôsobom po riadkoch vypĺňame jednotlivé bunky tabuľky číslami z postupnosti 1 až $n-1$, 1 až $n-1$, ..., až kým nevyplníme všetky bunky z tabuľky. Druhého hráča z páru do tabuľky doplníme tak, že zoberieme prvých hráčov z dvojíc (tých, ktorých sme doplnili v predošlom kroku) z nasledujúceho riadku a v obrátenom poradí ich napíšeme do aktuálneho riadku. Pre posledný riadok sa považuje za nasledujúci riadok prvý riadok z tabuľky. V prvom stĺpci bude prvý aj druhý hráč vždy rovnaký. V prípade nepárneho počtu hráčov tento hráč v danom kole nehrá. V prípade párneho počtu hráčov sa v prvom stĺpci striedavo (na prvé, druhé, prvé, ... miesto v dvojici) doplní hráč n.

Ďalšia metóda, ktorá generuje losovanie systémom round-robin, sa označuje ako „čistý kruhový turnaj“ z anglického „pure round robin tournament“. V prípade nepárneho počtu hráčov sa pridá pred samotnou aplikáciou algoritmu jeden fiktívny hráč, s nasledujúcim číslom v poradí. V tejto metóde sa rozdelia hráči do dvoch riadkov tabuľky tak, že vo vrchnom riadku bude hráč 1 až hráč $\frac{n}{2}$ a v druhom riadku budú



Obr. 1.1: Ilustrácia algoritmu losovania systémom round-robin, kruhová metóda, kolo 1. Šípky v tomto obrázku ukazujú ktorým smerom sa budú hráči cyklicky posúvať. Hráč 1 je pevne určený, tento hráč sa v tabuľke nebude cyklicky posúvať. V aktuálnom kole budú proti sebe hrať hráči, ktorí sú v rovnakom stĺpci, t.j. hráči 1-14, 2-13, 3-12, 4-11, 5-10, 6-9, 7-8. V prípade, že hráčov by bolo len 13, tak hráč 14 by bol fiktívny a hráč 1 by nehral v tomto kole, keďže je v rovnakom stĺpci, ako fiktívny hráč. Príklad prevzatý zo stránky [5].



Obr. 1.2: Na tomto obrázku je ukázané, ako tento cyklus funguje. Dĺžka celého cyklu (počet posúvaní a teda počet kôl) je počet nepevných políčok, keďže sa v každom kroku posunie posunú hráči o jednu pozíciu. Na obrázku je počet nepevných políčok 13, preto aj počet kôl bude 13. Príklad prevzatý zo stránky [5].

hráči n[1] až $\frac{n}{2}+1$. Jeden hráč sa označí za fixného - t.j. takého, ktorý nebude v tabuľke meniť svoje miesto. Podrobnejšie vysvetlenie tejto tabuľky je pri obrázku 1.1. Ukážka celého cyklu je na obrázku 1.2. V tejto metóde je zabezpečené, že každý hráč bude hrať rovnaký počet krát ako biely aj čierny.

Ďalšia metóda, ktorá sa využíva je metóda bergerových tabuliek. Autorom tejto metódy bol šachový majster Johann Berger. V tejto metóde sa ako fixný hráč zvolí posledný hráč a hráči sa cyklicky posúvajú proti smeru hodinových ručičiek t.j. doľava o $\frac{n}{2}$, kde n je počet hráčov (vrátane fiktívneho v prípade nepárneho počtu hráčov).

1.2 Vyraďovací systém

Ako druhý opíšeme *Vyraďovací (stromový systém)* [9]. Tento systém sa využíva napríklad v tenise, boxe, futbale [9]. V tomto systéme sa hrá $\lceil log_2(n) \rceil$ kôl, kde n je celkový počet hráčov. V tomto losovacom systéme nie sú žiadne remízy, t.j. každá partia môže mať len dva možné výsledky: prvý hráč z páru vyhral alebo druhý hráč z páru vyhral a teda postúpil do ďalšieho kola. Preto sa môže stať, že hráč, ktorý je dobrý môže skončiť hneď na začiatku, keď sa mu na začiatku nevydarí nejaký zápas (keď prehrá). Jeden z hráčov má výhodu oproti druhému (napríklad že ide prvý, pokiaľ je v konkrétnej hre výhoda ísť prvý, resp. že hrá doma a podobne). Tento problém vieme čiastočne eliminovať tým, že dvaja hráči budú hrať niekoľko (zvyčajne 2 alebo 3 zápasy) proti sebe a až potom sa rozhodne, ktorý hráč z konkrétnej dvojice je lepší a postúpi ďalej.

1.3 Švajčiarsky systém

Ďalší systém, ktorému sa budeme v tejto práci venovať primárne, je *švajčiarsky systém*. Švajčiarsky systém sa využíva hlavne v šachových turnajoch, prípadne aj v iných športoch [7], napr. v hre *Go* [3], *Bridge* [2], *Scrabble* [6]. Tento systém použil v roku 1895 v šachovom turnaji v meste Zürich Julius Müller, preto tento sa tento losovací systém označuje ako švajčiarsky.

Počet kôl v tomto systéme je dopredu určený, avšak nasledujúce kolo sa vylosuje na základe výsledkov z predchádzajúcich kôl, podľa pravidiel, ktoré si nižšie opíšeme.

Jednotlivé zápasy sa teda nelosujú všetky na začiatku, ale sa losujú postupne (po jednotlivých kolách) v závislosti od predchádzajúcich výsledkov. Celkový počet kôl (a teda aj zápasov) nemusí závisieť od počtu hráčov, ktorí sa zúčastňujú turnaja tak, ako je tomu v systéme round-robin. Celkové poradie hráčov vyplynie podľa celkového počtu získaných bodov po poslednom odohranom kole.

[1] Číslo n označuje celkový počet hráčov, vrátane fiktívneho hráča, pri nepárnom počte hráčov.

Pravidlá švajčiarskeho systému sa dajú rozdeliť do dvoch skupín: povinné pravidlá a preferenčné pravidlá. Povinné pravidlá musí spĺňať každé párovanie a preferenčné pravidlá sa aplikujú len vtedy, ak sa dajú aplikovať (a nebudú tým porušené povinné pravidlá). Niektoré pravidlá môžu byť mierne modifikované v závislosti od konkrétneho švajčiarskeho systému, ktorý sa používa.
Medzi povinné pravidlá podľa [11] patrí:

1. Počet kôl je známy na začiatku turnaja.
2. Pre každý pár hráčov platí, že nemôžu v turnaji hrať spolu viac ako jedenkrát.
3. Celkový rozdiel počtu bielych a čiernych farieb, s ktorými hral daný hráč, nemôže prekročiť 2 resp. byť menší ako -2. (Rozdiel sa vypočíta ako **W**-**B**, kde **W** je počet bielych farieb a **B** je počet čiernych farieb).
4. Pre každého hráča platí, že rovnakú farbu môže mať maximálne dvakrát za sebou.
5. Hráč môže byť nespárovaný systémom len raz.
6. Hráč, ktorý už bol nespárovaný systémom alebo dostal kontumačný bod, už nesmie byť v nasledujúcich kolách nespárovaný.

Povinné pravidlá o farbách nemusia v závislosti od konkrétneho variantu platiť v poslednom kole.
Medzi preferenčné pravidlá (presnejšie ich popíšeme v kapitole 2) podľa [11] platí:

1. Ak je to možné, tak sa párujú hráči s rovnakým počtom bodov (prípadne tak, aby rozdiel bodov bol čo najmenší);
2. Párovanie sa snaží vyrovnávať priebežný počet farieb (ak je to možné), t.j. hráč dostane farbu, ktorú mal menej často;
3. Párovanie sa snaží vyrovnávať posledné farby (ak je to možné), t.j. hráč dostane opačnú farbu od tej, ktorú mal naposledy.

Tieto pravidlá tvoria základ švajčiarskeho systému, dodržuje ich každý variant. Existujú 3 používané varianty losovania švajčiarskym systémom: *holandský (Dutch)*, *Dubov*, *Burstein* a *Lim*. Základné rozdiely teraz v krátkosti opíšeme.

Vo všetkých troch variantoch sa hráči rozdelia podľa bodov do skupín (v každej skupine sú hráči s rovnakým počtom bodov).

Vo variante Dutch sa hráči v každej skupine zoradia podľa štartovného čísla. Pokiaľ je to možné, tak sa primárne párujú hráči z prvej polovice skupiny s hráčmi z druhej polovice skupiny. Ak existuje hráč, ktorý sa nedá v danej skupine spárovať, tak prepadne do nižšej skupiny.

Vo variante Dubov sa hráči v jednotlivých skupinách zoraďujú na základe ratingu.

Vo variante Burstein sa hráči v skupine zoraďujú na základe pomocných vedľajších hodnotení (Buchholz, Sonneborn-Berger).

Vo variante Lim je definovaná stredná skupina - skupina s hráčmi, ktoré majú počet bodov rovný polovičke odohraných kôl. Párovanie v jednotlivých skupinách najprv prebieha od najlepšej skupiny po strednú (bez strednej), potom od najhoršej skupiny smerom nahor po strednú skupinu a na koniec sa spáruje posledná stredná skupina so všetkými zvyšnými hráčmi. Ak niektorý hráč v skupine od najlepšej po strednú nemôže byť v danej skupine spárovaný, stane sa z neho downFloater a prepadne do nižšej skupiny. Podobne, ak sa ide smerom od najhoršej skupiny nahor a niektorý hráč nemôže byť spárovaný v danej skupine, tak sa z neho stane upFloater a postúpi do vyššej skupiny.

Kapitola 2

Problém losovania švajčiarskym systémom

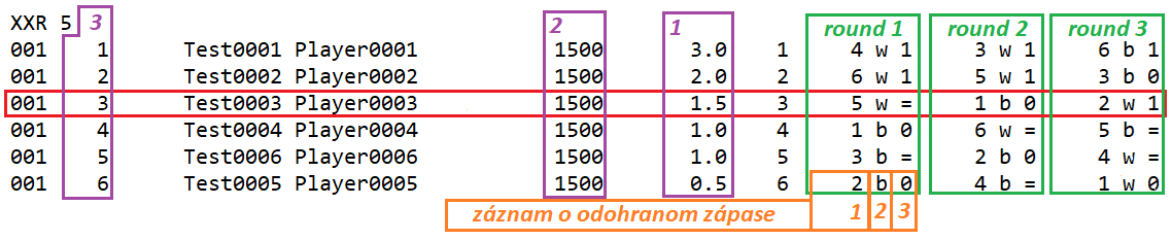
V tejto kapitole opíšeme problém losovania švajčiarskym systémom a bližšie opíšeme preferenčné pravidlá. Na *vstupe* pre losovanie je záznam doteraz odohraného turnaja (stav turnaja) a (voliteľné) parametre turnaja. Na *výstupe* očakávame vylosované ďalšie kolo. Pre každé dve rôzne losovania vieme povedať (rozhodnúť) na základe povinných a preferenčných pravidiel, ktoré z týchto dvoch losovaní je lepšie. Každé *korektné losovanie (platné losovanie)* musí dodržovať povinné pravidlá. Našim cieľom je teda nájsť najlepšie losovanie spomedzi všetkých platných losovaní. Problém losovania je teda optimalizačný problém.

Hráč je pre účely losovania objekt, ktorý má pridelených niekoľko hodnôt (*atribútov*), na základe ktorých bude každý hráč jednoznačne odlíšiteľný od iných hráčov. Budeme ho teda vedieť jednoznačne identifikovať a teda budeme vedieť ľubovoľných dvoch hráčov porovnať a vyrobiť úplne usporiadanie množiny hráčov na základe atribútov, ktoré budú mať daní hráči. Hodnoty týchto atribútov sa dajú získať zo stavu turnaja, resp. vypočítať.

Každý hráč má základné tri atribúty – uvádzame len tie, ktoré sú podstatné pre losovanie, t.j. hráč má aj iné atribúty (napr. meno), ktoré nie sú podstatné pre losovanie. V obrázku 2.1 je vyznačený záznam hráča 3 červeným rámikom a uvádzané atribúty (pre všetkých hráčov) sú vyznačené fialovým rámikom. Atribúty hráča 3 sú samozrejme len v riadku, ktorý je vyznačený červeným rámikom.

1. *Body* - číselný údaj, kde desatinná časť môže byť len ".5", t.j. násobky 0.5 (viac bodov sa pri zoraďovaní považuje za lepšie umiestnenie). Hodnota tohto atribútu zodpovedá počtu bodov, ktoré zatiaľ hráč dosiahol.

2. *Rating* - celočíselný údaj (väčšie číslo sa pri zoraďovaní považuje za lepšie umiestnenie). Hodnota tohto atribútu zodpovedá hodnoteniu, ktoré hráč mal na za-



Obr. 2.1: Príklad záznamu vo formáte .trf. Červeným rámikom je vyznačený hráč s číslom 3, fialovými rámikmi sú postupne vyznačené *id*, *rating* a *body*, zelenou sú vyznačené jednotlivé kolá a oranžovou je vyznačený záznam o odohranom zápase.

čiatku turnaja pridelené na základe hráčovej skúsenosti (hráčovho úspechu) pred turnajom.

3. *ID* – celočíselný údaj (menšie číslo sa pri zoraďovaní považuje za lepšie umiestnenie). Tento atribút je jednoznačný identifikátor hráča - každému hráčovi sa na začiatku turnaja priradí ID (štartovné číslo), pričom platí, že každý hráč má rôzne ID.

Priorita týchto atribútov pri zoraďovaní je vzostupná, t.j. najprv sa zoraďuje podľa bodov, potom podľa ratingu a nakoniec podľa jednoznačného identifikátora (atribútu ID).

Pre každého hráča existuje v každom doteraz odohranom kole práve jeden *záznam o odohranom zápase* (okrem krajného prípadu, keď sa niektorý hráč v priebehu turnaja rozhodne, že už nechce pokračovať a už nebude viac spárovaný - toto môže byť zaznačené záznamom *0000 - Z* alebo prázdnym miestom). Záznam o odohranom zápase je trojica, ktorá obsahuje nasledujúce údaje (v obrázku 2.1 je vyznačený oranžovou farbou jeden príklad):

1. s kým hral hráč, ktorému patrí záznam o odohranom zápase (v prípade že hráč nie je v danom kole spárovaný, tak súper nie je priradený, ale je vyznačené, že hráč v danom kole hráč nehral);

2. s akou farbou hral daný hráč (v súperovom zázname pre dané kolo musí byť uvedená opačná farba), v prípade, že hráč nie je v danom kole spárovaný, tak farba nie je priradená, t.j. farba je žiadna;

3. ako dopadol daný zápas, t.j. s akým výsledkom (v súperovom zázname pre dané kolo musí byť uvedený opačný výsledok, prípadne pri oboch remíza), v prípade, že hráč nie je v danom kole spárovaný, tak výsledok je závislý od toho, pre akú príčinu nebol hráč spárovaný.

Výstup obsahuje spárované dvojice hráčov a tiež informáciu, kto bude mať akú farbu. Za nespárovaných hráčov sú vo výstupe považovaní hráči, ktorí majú ako súpera

uvedeného hráča „0“, t.j. žiadneho.

Teraz si bližšie popíšeme pravidlá losovania: Dodržiavanie povinných pravidiel je zrejmé, dodržiavanie preferenčných pravidiel už nie je jednoznačne zrejmé, preto si zadefinujeme preferenčné pravidlá presnejšie. Preferenčné pravidlá uvedieme v zostupnom poradí, t.j. prvé preferenčné pravidlo je dôležitejšie ako druhé, druhé ako tretie, atď. Ak existuje lepšie párovanie vzhľadom na niektoré preferenčné pravidlo, tak sa toto pravidlo aplikuje aj v prípade, že bude porušené nižšie (číselne vyššie) preferenčné pravidlo. Každé preferenčné pravidlo je definované nezávislé od ostatných, jediná závislosť je v priorite preferenčných pravidiel (v prípade zhody dvoch párovaní, čo sa týka kvality pri niektorom preferenčnom pravidle, sa aplikuje nasledujúce preferenčné pravidlo v poradí).

Využitie *prvého preferenčného pravidla* slúži na maximalizovanie počtu spárovaných hráčov (minimalizácia počtu nespárovaných hráčov). Je zrejmé, že pri nepárnom počte hráčov, bude počet nespárovaných hráčov nepárny a pri párnom počte hráčov bude počet nespárovaných hráčov párny. Maximálny možný počet *prípustných párov* (prípustný pár je pár, ktorý neporušuje povinné pravidlá) v losovaní nazveme *kardinalita maximálneho párovania*, ďalej len *kardinalita*. Kardinalita (**K**) je teda celočíselná hodnota, ktorú dostaneme z nasledujúceho vzťahu: $\mathbf{K} = \frac{\#H - \#N}{2}$, kde $\#H$ je celkový počet hráčov, $\#N$ je počet nespárovaných hráčov. Cieľom prvého preferenčného pravidla je maximalizovať kardinalitu. Pri veľkom počte hráčov a teda veľkom počte prípustných párov bude počet nespárovaných hráčov 0, resp. 1. Pri malom počte hráčov a malom počte prípustných párov (krajné prípady) môže nastať situácia, že pri počte nespárovaných hráčov 0 resp. 1 neexistuje losovanie také, ktoré dodržuje všetky povinné pravidlá – v tomto prípade teda musíme zvýšiť počet nespárovaných hráčov na najmenší možný počet taký, že už bude existovať losovanie, v ktorom budú dodržané všetky povinné pravidlá. Hráč, ktorý je nespárovaný v danom kole, dostáva automaticky kontumačný bod a nemá žiadnu farbu, ani súpera.

Využitie *druhého preferenčného pravidla* slúži na spárovanie hráčov, tak, aby sme minimalizovali rozdiel počtu bodov medzi týmito spárovanými hráčmi. Prioritne sa párujú najlepší hráči (zoradení zostupne od najlepšieho po najhoršieho podľa systému uvedeného vyššie), t.j. hľadáme súpera najprv najlepšiemu hráčovi, potom ďalšiemu v poradí, ktorý nemá ešte pár, až kým nebude spárovaný taký počet hráčov, aby počet nájdených párov bol rovný (maximálnej) kardinalite zistenej pre dané párovanie.

Súper pre hráča H sa vyberá takým istým spôsobom ako prvý hráč z páru podľa predošlého odseku. Nech $B(H)$ znamená počet bodov, ktoré má aktuálne hráč H. Ak pre prvého hráča existujú hráči H_1 a H_2 takí, že $B(H_1) > B(H_2)$, resp. $B(H) - B(H_1) < B(H) - B(H_2)$ a zároveň platí, že pri páre $H - H_1$ by sa znížila kardinalita a pri páre $H - H_2$ zostane kardinalita zachovaná, tak bude platný pár $H - H_2$ i napriek tomu, že

existuje hráč s menším rozdielom bodov.

Využitie *tretieho preferenčného pravidla* slúži na vyrovnávanie farieb v jednotlivých pároch, tak, aby pokiaľ je to možné hráč dostal farbu, ktorú mal menej často. Farbám sa prideľujú číselné hodnoty: $w = +1$ a $b = -1$. Funkcia $F(H)$, kde H je hráč, označuje s ktorou farbou hral častejšie – ak $F(H) = 2$, tak H hral s bielou farbou o 2 hry viac ako s čiernou, ak $F(H) = -1$, tak H hral s čiernou farbou o 1 hru viac ako s bielou a ak $F(H) = 0$, tak H hral s bielou farbou rovnaký počet krát ako s čiernou. Ak pre niektorého hráča H v dvojici platí, že $F(H) = +2$, resp. $F(H) = -2$, tak aby nebolo porušené povinné pravidlo o farbách (3), bude mať hráč *vynútenú farbu*. Obaja hráči v páre nemôžu mať rovnakú vynútenú farbu, keďže by tento pár potom nebol platný kvôli povinnému pravidlu (3). Inak sa zvolí taká farba, aby sa $F(H)$ priblížila k 0. Ak budú mať obaja hráči v páre $F(H) = 1$, $F(H) = 0$ alebo $F(H) = -1$, tak sa aplikuje nasledujúce preferenčné pravidlo. Párovanie je lepšie, ak po odohraní aktuálneho kola bude mať menší počet hráčov vynútenú farbu.

Využitie *štvrtého preferenčného pravidla* slúži na striedanie farieb v jednotlivých pároch, tak aby pokiaľ je to možné hráč dostal opačnú farbu, akú mal naposledy. Funkcia $f(H)$, kde H je hráč, označuje farby, s ktorými hral naposledy – ak $f(H) = 2$, tak H hral s bielou farbou 2 posledné hry, ak $f(H) = -1$, tak H hral s čiernou farbou poslednú jednu hru a ak $F(H) = 0$, tak H ešte nehral žiadnu hru. Ak pre niektorého hráča H v dvojici platí, že $F(H) = +2$, resp. $F(H) = -2$, tak aby nebolo porušené povinné pravidlo o farbách (4), bude mať hráč *vynútenú farbu*. Obaja hráči v páre nemôžu mať rovnakú vynútenú farbu, keďže by tento pár potom nebol platný kvôli povinnému pravidlu (4). Inak sa zvolí taká farba, aby sa $f(H)$ priblížila k 0. Ak budú mať obaja hráči v páre $f(H) = 1$ resp. $f(H) = -1$, tak sa aplikuje preferencia konkrétneho hráča na farbu, ak nemá preferovanú farbu, tak ďalej hráč s vyšším počtom bodov dostane bielu farbu, ak ani to nestačí, tak potom hráč s nižším štartovným číslom (ID) dostane bielu farbu. Opäť platí, že párovanie je lepšie, ak po odohraní aktuálneho kola bude mať menší počet hráčov vynútenú farbu.

Kapitola 3

Maximum Weighted Matching

Na začiatku tejto kapitoly si najprv vysvetlíme pojmy, ktoré budeme používať:

Ohodnotený graf je graf, ktorý má ku každej hrane priradené číslo podľa priority.

Jednotkový graf je graf, ktorý má neohodnotené hrany, resp. každá hrana má rovnakú váhu, napr. 1.

Nech je daný graf **G**, ktorý má konečnú množinu vrcholov **V** (**V** z anglického slova **V**ertex) a množinu hrán medzi vrcholmi **E** (**E** z anglického slova **E**dge). Množina hrán E je teda podmnožina množiny $\{[x, y] \mid [x, y] \in V \times V\}$. Graf môžeme rozumieť aj ako binárnu reláciu nad množinou $V \times V$, kde dvojica [x,y] patrí do relácie, ak existuje v grafe hrana (x,y). Ak budeme písať o jednotkovom grafe, tak takýto graf budeme označovať ako *G(V,E)*, ak o ohodnotenom grafe, tak *G(V,E,w)*, kde **w** (**w** z anglického slova **w**eight) sú hodnoty priradené jednotlivým hranám. Keďže sa v tejto práci venujeme hľadaniu párov, tak všetky grafy obsahujú len neorientované hrany (t.j. $[x, y] = [y, x]$) a zároveň grafy neobsahujú slučku (t.j. pre každú hranu $[x, y] \in E$, kde $x, y \in V$ platí, že $x \neq y$).

Párovanie v grafe je ľubovoľná podmnožina P taká, pre ktorú platí $P \subseteq E$ a zároveň každý vrchol môže byť len v jednej hrane z množiny P, teda každý vrchol je stupňa 0 alebo 1 (z vrcholu vedie práve jedna neorientovaná hrana alebo žiadna).

Perfektný graf je graf, v ktorom existuje *perfektné párovanie*.

Perfektné párovanie v grafe je ľubovoľná podmnožina P taká, pre ktorú platí $P \subseteq E$ a každý vrchol je stupňa 1 (z každého vrcholu vedie práve jedna neorientovaná hrana). Ďalej platí, že $|P| = \frac{|V|}{2}$. Grafy s nepárnym počtom vrcholov teda nemôžu byť perfektné. Perfektný graf je teda špeciálny prípad grafu.

3.1 Blossom algoritmus

V tejto práci sa budeme zaoberať riešením a implementáciou problému losovania z predošlej kapitoly (kapitoly 2) s využitím podproblému Maximum Weighted Matching

(MWM) [4]. Ako vstup problému MWM je neorientovaný graf bez slučiek, kde každá hrana má nejakú váhu (alebo všetky hrany majú jednotkovú váhu v prípade, že hrany sú si navzájom rovnocenné). Podľa toho, či chceme maximálne alebo minimálne párovanie bude výstup problému MWM množina hrán zo vstupného grafu taká, že súčet váh všetkých hrán v tejto množine je minimálny resp. maximálny možný a zároveň každý vrchol sa môže vyskytnúť maximálne v jednej hrane.

Problémom Maximum Weighted Matching sa už v roku 1961 zaoberal Jack Edmonds. V roku 1965 Jack Edmonds publikoval slávny algoritmus *Blossom* [1], ktorý rieši problém MWM v polynomiálnej časovej zložitosti, konkrétne $O(V^4)$, resp. v časovej zložitosti $O(V^2E)$, kde V je počet vrcholov a E je počet hrán vo vstupnom grafe. *Gabow* a *Lawler* nezávisle od seba vylepšili Edmondsov algoritmus Blossom, ktorý mal časovú zložitosť $O(V^3)$ [13] [20]. *Galil* a kol. vylepšili algoritmus na časovú zložitosť $O(VE \log V)$ a následne až na časovú zložitosť $O(V(E + V \log V))$. Implementácia algoritmu Blossom mala viaceré verzie, od *BLOSSOM I* až po dnešnú *BLOSSOM V*. Implementácia BLOSSOM IV a staršie používali pomerne jednoduché dátové štruktúry, nevyužívali prioritné fronty na nájdenie hrany s najmenšou voľnosťou. Otázku, či prioritné fronty môžu použiť na implementáciu, ktorá zlepšuje zložitosť v prípade krajného prípadu zodpovedali *Melhorn* a *Schäfer*, ktorí vyvinuli implementáciu s časovou zložitosťou $O(VE \log V)$. Táto implementácia vykazuje zlepšenie oproti implementácii BLOSSOM IV.

V súčasnosti existuje implementácia algoritmu BLOSSOM V v dvoch programovacích jazykoch: v Jave [18] a v C++ [19]. Implementáciu v oboch programovacích jazykoch spravil Vladimir Kolmogorov. Obe implementácie majú časovú zložitosť $O(EV^2)$. Aj keď existuje algoritmus s lepšou asymptotickou zložitosťou $O(EV \log V)$ [13], implementácia BLOSSOM V je spravená tak, že krajné prípady netrvajú o moc dlhšie ako štandardné prípady ([20], kapitola 3), keďže najviac času trvá spojiť len zopár posledných vrcholov.

V tejto práci sa budeme prevažne venovať implementácii v jazyku Java, keďže implementáciu losovania švajčiarskym systémom sme robili v Jave. Implementácia BLOSSOM V umožňuje mať na vstupe ohodnotený alebo neohodnotený graf. V našom algoritme využívame „podprogram“ BLOSSOM V, ďalej aj ako *BlossomC* pre Blossom v C++ a *BlossomJava* pre Blossom v Jave, ktorý rieši problém Maximum Weighted Matching. Najprv sme pracovali s implementáciou algoritmu BlossomC v C++ (kým sme nenašli, že existuje implementácia aj v Jave). V nasledujúcom texte stručne popíšeme aké nevýhody má BlossomC v porovnaní s BlossomJava.

3.2 Implementácia BLOSSOM v C++

Prvá implementácia, ktorú sme našli a používali, bola implementácia v jazyku C++ [19] (BlossomC). Táto implementácia mala nasledujúce nevýhody, ktoré boli dosť obmedzujúce:

Prvé hlavné obmedzenie je, že na vstupe musel vždy byť perfektný graf, inak táto implementácia vyhodila chybu. Toto sme vyriešili s pomocou algoritmu transformácie ľubovoľného neorientovaného grafu bez slučiek na perfektný graf [14], ktorý si teraz popíšeme.

1. Nech $G = (V, E)$ je vstupný graf (inštancia problému Maximum Weighted Matching), ktorý chceme prerobiť na perfektný graf. Každej hrane priradíme váhu 1 - vyrobíme jednotkový graf. Ďalej vyrobíme kópiu grafu G a označíme ju ako G' (tiež množina V bude V' a množina E bude E').

2. Zjednotíme grafy G a G' do grafu G'' a pre každý vrchol $u \in V$ a každý vrchol $v \in V'$ pridáme v grafe G'' hranu $[u, v]$ s váhou 0. Graf G'' je teda určite perfektný a jeho kardinalita bude rovná výslednej váhe pri maximálnom párovaní na grafe G'', keďže umelo pridané hrany budú mať váhu 0.

Druhé hlavné obmedzenie je, že na vstupe musel byť graf, ktorý má súvislý rad vrcholov, teda vrcholy, ktoré sa číslovali od 0 po $n - 1$. Takýto graf mohol vzniknúť napr. tak, že nejaký hráč v danom kole nemohol hrať[1]. Toto obmedzenie sme riešili tak, že sme použili dátovú štruktúru `Map<vrchol, hráč>`, kde vrchol boli čísla od 0 po $n - 1$ (n je počet hráčov v pôvodnom grafe) a hráč označuje číslo hráča, ktorý je vo vstupnom grafe. Je to teda funkcia, ktorá ku každému vrcholu od 0 po $n - 1$ priradí práve jedného hráča zo vstupného grafu a zároveň neexistujú žiadne dve priradenia vo funkcii také, že dva vrcholy by mali priradeného toho istého hráča.

Tretie obmedzenie je, že sme vstupný graf v správnom formáte museli najprv uložiť do nejakého súboru, potom spustiť BlossomC (v argumente BlossomC sme napísali, odkiaľ má čítať vstupný graf a kde má uložiť výstup) a nakoniec bolo treba interpretovať výstup. Z výstupu nás zaujímala len jedna hodnota - kardinalita - a výstup mal niekoľko riadkov, medzi ktorými sme museli údaj o kardinalite nájsť.

3.3 Implementácia BLOSSOM v Jave

Výhoda BlossomJava v porovnaní s BlossomC bolo, že vstupný graf vieme priamo vygenerovať v programe (nie je teda potrebný externý súbor). BlossomJava poskytuje (okrem iných, ktoré sme nepoužívali v našom programe) nasledujúce metódy:

1. Metóda „`MaximumWeightedMatching(graf, optimalizačný zmysel)`“. Optima-

[1] z rôznych príčin, napr. párovacích, alebo bol hráč chorý a pod.

lizačný zmysel môže byť maximalizovanie váhy vo výslednej množine hrán alebo minimalizovanie tejto váhy.

Na vstupe tejto metódy je ľubovoľný neorientovaný graf bez slučiek. Ako výstup tejto metódy bude množina hrán (množina H) zo vstupného grafu taká, že súčet váh všetkých hrán z tejto výstupnej množiny je maximálny, resp. minimálny možný (v závislosti od optimalizačného zmyslu). Táto metóda však nerieši kardinalitu, t.j. ak existujú dve množiny hrán v grafe H_1 a H_2 také, že kardinalita množiny H_1 je menšia ako kardinalita množiny H_2 ($|H_1| < |H_2|$) a súčet váh v H_1 je väčší ako v H_2, tak táto metóda považuje v prípade maximalizácie váhy H_1 za lepšiu množinu ako H_2 i napriek menšej kardinalite.

2. Metóda „`MaximumWeightedPerfectMatching(graf, optimalizačný zmysel)`“. Optimalizačný zmysel je taký istý ako v predošlej metóde - teda maximalizovanie váhy vo výslednej množine hrán alebo minimalizovanie tejto váhy.

   Na vstupe tejto metódy je ľubovoľný neorientovaný perfektný graf bez slučiek. Ako výstup tejto metódy je perfektné párovanie (s minimálnym alebo maximálnym súčtom váh v závislosti od optimalizačného zmyslu). Keďže je to perfektné párovanie, tak jeho kardinalita (veľkosť výstupnej množiny párov) bude vždy rovná $2|V|$, kde $|V|$ je počet vrcholov vo vstupnom grafe. Ďalšia vyplývajúca vlastnosť, ktorú sme využili v našom algoritme je, že v perfektnom párovaní je každý hráč v niektorej hrane z výstupnej množiny hrán, teda je spárovaný s niekým, preto je kardinalita dodržaná vždy.

3.4 Využitie implementácie BLOSSOM V v Jave

V nasledujúcom texte budeme označovať hľadanie maximálneho párovania s neohodnotenými hranami ako „MaximumMatching“ alebo „MM“, hľadanie minimálneho perfektného párovania s ohodnotenými hranami ako „MinimalWeightedPerfectMatching“, skrátene „MinWPM“ a hľadanie maximálneho perfektného párovania s ohodnotenými hranami ako „MaximalWeightedPerfectMatching“, resp. „MaxWPM“. Keďže nám v prvom rade záleží na kardinalite, tak MM budeme používať len s neohodnotenými hranami (všetky hrany majú rovnakú prioritu) a bude slúžiť len na zisťovanie kardinality vo všeobecnom, aj neperfektnom grafe. V nasledujúcich bodoch opíšeme postup - s opakovaným využívaním metód BlossomJava - ako nájsť najlepšie párovanie pre ďalšie kolo.

Párovanie hráčov sa dá transformovať na problém Maximum Weighted Matching tak, že hráči budú vrcholy v grafe a všetky prípustné páry budú neorientované hrany a tento graf bude vstupom pre BLOSSOM V.

1. Nájsť minimálny počet hráčov, ktorí musia byť nespárovaní (vyrobiť jednotkový graf ako vstup pre MM):

   - vytvoriť vrchol v grafe pre každého hráča, ktorý môže v danom kole hrať (väčšinou všetci hráči, niekedy je dopredu dané, ktorí hráči nemôžu v danom kole hrať z iných príčin ako párovanie, napr. choroba)
   - vytvoriť hrany v grafe s váhou 1, ktoré reprezentujú všetky možné (prípustné) páry
   - spustiť MM a výstup z MM je kardinalita grafu - maximálny počet hrán v grafe, ktoré sa dajú spárovať (t.j. počet možných párov vo výslednom párovaní, ak budeme ďalej písať o kardinalite, tak máme na mysli kardinalitu množiny párov)

2. Určiť konkrétnych nespárovaných hráčov (odspodu): prehľadávať hráčov zoradených podľa určených kritérií[2] vzostupne, s využívaním MM. V predchádzajúcom kroku (1) zistíme, koľko hráčov musíme vyradiť, v tomto kroku ich po jednom vyradíme:

   (a) vymažeme z grafu najhoršieho hráča (aj s jeho hranami)

   (b) vyskúšame (s využitím MM) či sa kardinalita nezmenila (nezmenšila[3])

   (c) ak nie, hráča sme úspešne vymazali z grafu (a teda je pre toto kolo vyradený[4])

   (d) ak sa kardinalita zmenila, tak tohto hráča (aj jeho hranami) vrátime späť do grafu a pokračujeme od bodu (a) s nasledujúcim hráčom v poradí (podľa zoradenia odspodu)[5].

   Ak sme už vyradili taký počet hráčov, aký sme zistili v predchádzajúcej časti, tak táto časť je ukončená a ako výstup prehľadávania bude množina konkrétnych nespárovaných hráčov v danom kole. Výstup z tejto časti bude teda množina vyradených hráčov a tiež perfektný graf zvyšných hráčov (spolu s ich hranami).

3. Hľadanie konkrétnych párov. Na vstupe bude perfektný graf s nevyradenými hráčmi a ich zoradeným zoznamom hrán. Jednotlivé hrany budú mať nasledujúce vlastnosti: v množine zoradených hrán bude vždy na prvom mieste lepší

[2] Hráč s vyšším počtom bodov je lepší, v prípade rovnakého počtu bodov rozhoduje rating (vyšší je lepší) - ak je stále zhoda, tak rozhodne jedinečný atribút - štartovné číslo (menšie je lepšie)

[3] zväčšiť sa nemohla, keďže hráčov z grafu odoberáme

[4] a ďalej sa ním a s jeho vrcholmi nepočíta

[5] keďže sme v predošlej časti pseudokódu (bod 1) zistili kardinalitu, tak musí existovať daný počet hráčov na vyradenie tak, aby kardinalita zostala zachovaná

hráč. Jednotlivé hrany budú zoradené (po skupinách) nasledovne: ako prvá bude skupina hrán s najlepším hráčom, potom skupina hrán s druhým najlepším hráčom ... skupina hrán s najhorším hráčom. V každej skupine budú hrany zoradené podľa preferencie pre konkrétneho hráča (najprirodzenejší súper, druhý najprirodzenejší súper ... najmenej prirodzený súper). Ako výstup z tejto časti bude konkrétna množina párov. Rôzne metódy hľadania konkrétnych párov opíšeme nižšie.

4. Vo výstupnej množine z predošlého bodu ešte treba určiť, ktorý hráč z dvojice bude prvý (bude hrať ako biely) a ktorý druhý (bude hrať ako čierny) - toto sa bude dať spraviť pre každú dvojicu s dodržaním povinných pravidiel, pretože neprípustné dvojice sme vylúčili na začiatku - pred samotným párovaním.

Nazvime túto časť „výroba perfektného grafu“, keďže na túto časť budeme odkazovať aj v ďalšom texte.

Cieľom je minimalizácia počtu volaní metód Blossom, keďže výroba vstupu (vstupný graf) pre Blossom - pre obe metódy je pri veľkých vstupoch príliš dlhá vzhľadom na zvyšok algoritmu. Tiež aj samotný Blossom je časovo náročný pri veľkých vstupoch. Minimalizovať počet volaní Blossom vieme napríklad využitím ohodnoteného grafu (ale ako sme vyššie písali, toto sa dá využiť len s perfektným grafom na vstupe).

3.4.1 Prvá verzia: priamočiare riešenie

Prvé riešenie, ktorým sa dajú nájsť páry v perfektnom grafe spočíva v postupnom prehľadávaní, až kým sa nám nepodarí vyrobiť nejaké perfektné párovanie. Hlavná nevýhoda tohto algoritmu je exponenciálna časová zložitosť $O(2^n)$ - v najhoršom prípade, by sa mohlo stať, že do perfektného párovania by nám chýbalo nájsť už len jednu hranu, ale taká hrana už nie je a keby sme na začiatku tohto algoritmu vybrali inú hranu, tak by sa podarilo vyrobiť perfektné párovanie. Teraz napíšeme pseudokód ako by toto riešenie fungovalo (3.1).

3.4.2 Druhá verzia: využitie grafu s neohodnotenými hranami

Zisťovanie konkrétnych párov zhora (od najviac prioritného páru po najmenej prioritný). Implementačné detaily sú v kapitole 4.5.1. Po nájdení konkrétneho páru budeme pokračovať s hľadaním zvyšných párov (celkový počet párov bude rovný kardinalite zistenej v bode 1 v časti *výroba perfektného grafu*).

Algoritmus 3.1: Algorithm for exponential pairing

```
List<Edge> pairing(List<Player> players,
    List<Players> removedPlayers, TreeMap<Integer, Player> hm) {
  //v liste players su vsetci hraci (zoradeni)

  maxCountOfPlayers = players.size()-removedPlayers.size();
  List<Edge> finalEdges = new ArrayList<>();
  List<Integer> pairedPlayers = new ArrayList<>();
  List<Integer> lastStop = new ArrayList<>();
  /*
    do players treba dat players.size()-krat "0";
    v tomto liste je ulozene, kde skoncil pri ktorom hracovi
    (index) v .getPossiblePlayers()
  */

  for(int c = 0; c<players.size(); c++) {
    Player pl = players.get(c);
    if (removedPlayers.contains(pl)) continue;
    int tmp = finalEdges.size();
    for (int d=lastStop.get(c);
          d<pl.getPossiblePlayers().size(); d++) {
      Integer i = pl.getPossiblePlayers().get(d);
      //porovnava, ci je prvy hrac z dvojice lepsi
      if (pl.compareTo(reader.playersMap().get(i)) < 0) {
        continue;
      }
      /*
        zistuje ci niektory z hracov uz nie je sparovany
        (v niektorom predtym sparovanom pare)
      */
      if (pairedPlayers.contains(i) ||
        pairedPlayers.contains(pl.getId())) continue;
      //zistuje, ci edge je mozna (ci neporusuje pravidla)
      if (!edgeIsPossible(i, pl.getId)) continue;

      finalEdges.add(new Edge(i, pl.getId()));
      pairedPlayers.add(i);
      pairedPlayers.add(pl.getID());
      if (maxCountOfPlayers == pairedPlayers.size())  {
        return pairedPlayers;
      }
      break;
      }
      if (finalEdges.size() != tmp + 1) {
      Edge removed = finalEdges.removeLast();
      pairedPlayers.remove(removed.from());
      pairedPlayers.remove(removed.to());
      c--;
      /*
        zaroven treba dat na prislusny index aktualneho hraca
        (pl) v poli lastStop cislo 0
      */

    }
}
return null;
```

1. Vymažeme z grafu dvoch hráčov (spolu s ich hranami[6], t.j. hranami jedného aj druhého hráča) - hráčov, ktorí sú v najlepšom páre (podľa priority).

2. Vyskúšame (s využitím MM) či sa kardinalita grafu (po vymazaní danej hrany z bodu 1) zmenšila presne o 1, keďže sme odobrali dvoch hráčov a testujeme, či po výbere tohoto páru môžeme spárovať zvyšných hráčov tak, aby výsledná kardinalita grafu zostala nezmenená.

3. Ak áno, týchto hráčov sme úspešne vymazali z grafu (a teda sme určili jeden z párov - pár, ktorý bude vo výslednej množine párov).

4. Ak sa kardinalita zmenšila viac ako o 1, tak týchto hráčov (aj s ich hranami) vrátime späť do grafu, keďže s touto hranou sa nepodarí spárovať zvyšných hráčov tak, aby celková kardinalita zostala zachovaná a pokračujeme od bodu (1) s nasledujúcou hranou v poradí (podľa priority zvrchu).[7]

Ak už sme ukončili túto časť (zistili množinu všetkých hrán, ktorá bude ako výstup (táto množina má kardinalitu takú, akú sme zistili v bode (1) v časti *výroba perfektného grafu*), tak graf už bude prázdny a párovanie máme vyrobené, táto množina bude ako vstup pre bod (4) v časti *výroba perfektného grafu*.

3.4.3 Tretia verzia: váhovanie podľa počtu bodov

Tento algoritmus bude využívať ohodnotený graf a maximálne perfektné párovanie. Detaily tohto algoritmu sú v kapitole 4.5.2.

3.4.4 Štvrtá verzia: hráči v skupinách podľa počtu bodov

Tento algoritmus bude využívať ohodnotený graf a maximálne perfektné párovanie. Implementačné detaily sú v kapitole 4.5.3. Na začiatku tohto algoritmu sa hráči opäť zaradia do skupín podľa počtu bodov. Párovanie bude prebiehať postupne po jednotlivých skupinách, t.j. najprv sa začne párovať najlepšia skupina (podľa počtu bodov), potom ďalšia, ..., až sa na konci bude párovať najhoršia skupina. Nájdenie párov len v konkrétnej skupine zabezpečíme tak, že hranám z danej skupiny[8] nastavíme váhu 1 a zvyšným možným hranám nastavíme váhu 0. Keďže metóda MaxWPM dá na výstupe perfektné párovanie, tak máme garantované, že kardinalita výsledného párovania

[6]Hrana hráča (hrana patrí hráčovi, je hráčova), znamená že hráč je v tejto hrane (jeho `id` sa nachádza v hrane) na pozícii `from` alebo `to`. V ďalšom texte pod hranou hráča máme na mysli túto definíciu hráčovej hrany.

[7]Keďže je na vstupe tejto časti perfektný graf, tak existuje taká množina hrán, s ktorou zostane kardinalita zachovaná.

[8]Hrana z danej skupiny znamená, že je v tejto hrane hráč z danej skupiny (hrana patrí hráčovi z danej skupiny).

bude maximálna možná. Metóda MaxWPM robí maximalizáciu váhy, preto vyberie maximálny možný počet hrán s váhou 1, t.j. spáruje maximálny možný počet hráčov v danej skupine, ktorú aktuálne páruje (páry s váhou 1).

Môže sa však stať, že sa nepodarí spárovať všetkých hráčov z danej skupiny. Hráča, ktorého sa v rámci párovania v jeho skupine nepodarí spárovať, nazývame *downFloater*, teda hráč ktorí prepadol do nižšej skupiny. DownFloaterov môže byť v danej skupine aj viac (a teda párovanie sa snaží minimalizovať počet downFloaterov). Ak sú nejakí downFloateri, tak ich chceme spárovať čo najskôr. Hranám, ktoré patria downFloaterom teda musíme nastaviť váhu tak, aby sa pri párovaní nasledujúcej skupiny párovali primárne. To docielime tak, že váhu nastavíme na hodnotu $(k+1)*(body\ súpera+1)^2$, kde hodnota k je maximálny možný počet párov s jednotkovou váhou, ktorý sa dá vybrať ($\lfloor \frac{h}{2} \rfloor$, kde h je počet hráčov v danej skupine, bez downFloaterov z vyššej skupiny). Takéto váhovanie nám zabezpečí, že hrana s downFloaterom sa bude vyberať primárne, keďže sa používa algoritmus MaxWPM a hodnota $(body\ súpera + 1)^2$ nám prioritne zabezpečí výber hrany, s menším rozdielom (teda keď bude mať súper downFloatera čo najviac bodov). Každý downFloater môže prepadnúť iba o jednu skupinu nižšie, keďže sa v nasledujúcej skupine nutne spáruje (v dôsledku vysokej váhy vzhľadom na súčet hrán hráčov zo skupiny, do ktorej downFloater prepadol). Nemôže sa teda vyskytnúť hrana, v ktorej budú obaja hráči downfloateri, pretože v tom prípade by boli spárovaní vo svojej skupine a nestali by sa z nich downFloateri.

Kapitola 4

Implementácia losovania

V tejto kapitole opíšeme samotnú implementáciu algoritmu losovania švajčiarskym systémom. V našom programe sme využívali rôzne dátové štruktúry (objekty, triedy). Tie najdôležitejšie si popíšeme v tejto kapitole.

4.1 Objekt (trieda) `Reader`

Táto trieda má konštruktor
`public Reader(String arg, boolean clrOff, boolean ignoreEmptyRounds)`,
kde `arg` je vstupný súbor, ktorý má prečítať trieda Reader prečítať; `clrOff` je parameter, ktorý ak je nastavený na true, tak nekontroluje pravidlá farieb v poslednom kole a `ignoreEmptyRounds` je parameter, ktorý ak je nastavený na `true`, tak ignoruje prázdny záznam o odohranej partii (súper je žiadny, výsledok žiadny, farba žiadna pre daného hráča a dané kolo, kde je prázdny záznam). Ak je nastavený na `false`, tak pri prvom výskyte prázdneho záznamu pre daného hráča sa nastaví argument `givenUp` a tento hráč už musí mať ďalej len prázdne záznamy, inak sa vyhodí chyba (hráč sa vzdal a zvyšok turnaja už nebude hrať). Trieda `Reader` má 2 hlavné časti:

1. metóda `read()` - táto metóda ma za úlohu prečítať jednotlivé riadky vstupného súboru a skontrolovať syntax týchto riadkov. Syntax je kontrolovaná podľa štandardného formátu **TRF** [12]. Táto metóda berie do úvahy len riadky podstatné pre párovanie, t.j. riadky s hlavičkou **001** (záznamy hráčov) a **XXR** (záznam o maximálnom počte kôl). Pre každý riadok typu **001** (záznamy hráčov) vytvorí inštanciu objektu `Player` s parametrami, ktoré opíšeme pri vysvetľovaní objektu Player. Keďže v našom programe často pracujeme len s číslami (`id`) hráčov, tak sme vytvorili aj objekt `Map<Integer, Player>` (konkrétne `TreeMap<Integer, Player>`), ktorý má v prípade potreby poskytnúť referenciu na objekt konkrétneho hráča na základe jeho `id`. Návratová hodnota tejto metódy je práve objekt typu `TreeMap<Integer, Player>`, v ktorom sú načítané všetky údaje o každom

hráčovi. Pre riadok typu **XXR** nastaví maximálny počet kôl. Tento riadok musí byť v súbore skôr, ako riadky **001**. V tejto metóde sa zároveň kontroluje (pre každého hráča zvlášť), či počet bodov v **TRF** súbore pre daného hráča je rovnaký ako súčet bodov zo všetkých záznamov o odohraných zápasoch daného hráča.

2. metóda `checkRules(Map<Integer, Player> hm)` - táto metóda má za úlohu skontrolovať dodržanie povinných pravidiel.

   (a) Pre každého hráča a každé kolo platí, že ak bol údaj v zázname hráča X pre dané kolo o spárovaní s hráčom Y, tak aj údaj v zázname hráča Y pre dané kolo musí obsahovať informáciu o sparovaní s hráčom X v danom kole. Ak hráč v danom kole nehral (nebol spárovaný) z rôznych príčin, tak údaj o spárovaní je prázdny, resp. „`0000`“;

   (b) Pre každého hráča a každé kolo platí, že ak bol už hráč X niekedy spárovaný s hráčom Y, tak už nikdy nemôže byť spárovaný hráč X s hráčom Y. Toto pravidlo má výnimku, že ak výsledok zápasu bol *kontumačný (forfeit)* t.j. zápas sa neodohral, tak títo dvaja hráči ešte môžu byť spárovaní;

   (c) Pre každého hráča a každé kolo platí, že hráči, ktorí boli v danom kole spárovaní, musia mať navzájom opačnú farbu, s akou hrali (t.j `w-b`/`b-w`). Ak hráč v danom kole nehral (nebol spárovaný) z rôznych príčin, tak farba je žiadna „`-`“ a pre účely pravidiel o farbách sa tento záznam ignoruje (postupnosť farieb *WWBWBW-* - je pre účely pravidiel o farbách rovnaká ako postupnosť *W-WBWBW-* alebo *WWBW-BW-* alebo *W- -WBWBW*, atď.);

   (d) Pre každého hráča platí, že priebežný rozdiel farieb, nesmie byť väčší ako 2 a zároveň nikdy nemôže mať trikrát po sebe tú istú farbu. Toto pravidlo má výnimku v poslednom kole (ak je atribút `clrOff` true, tak sa toto pravidlo pre posledné kolo ignoruje);

   (e) Pre každého hráča a každé kolo platí, že typ výsledku zápasu s hráčom Y v zázname hráča X musí byť rovnaký ako typ výsledku v zázname hráča Y pre dané kolo. Ak hráč v danom kole nehral (nebol spárovaný) z rôznych príčin, tak je typ výsledku pre nespárovaného hráča „bye“ viď [12];

   (f) Pre každého hráča a každé kolo platí, že hráči, ktorí boli v danom kole spárovaní, musia mať navzájom opačnú hodnotu výsledku (t.j *win-lose*/*lose-win*) alebo (*draw-draw*). Ak hráč v danom kole nehral (nebol spárovaný, z rôznych príčin), tak hodnota výsledku môže byť ľubovoľná (samozrejme musí byť dodržaný typ výsledku - nespárovaný hráč);

   (g) Pre každého hráča platí, že môže byť systémom nespárovaný len raz (t.j. záznam „`0000 - U`“ môže mať len raz) a zároveň ak už niekedy dostal kontumačný bod, tak tiež už nemôže byť nespárovaný systémom.

4.2 Objekt (trieda) Player

Táto trieda má konštruktor

`public Player(Integer id, String name, Integer rating, Double points, Integer rank, List<PlayedGameWith> playedGames)`,

kde `id` je jednoznačný identifikátor hráča (štartové číslo), `name` je meno hráča, `rating` je hodnota, ktorá určuje rating hráča, `points` sú aktuálne body hráča, `rank` je aktuálne priebežné poradie hráča a `playedGames` sú jednotlivé záznamy o odohraných zápasoch. Objekt `PlayedGameWith` teda obsahuje údaje, s kým hráč hral, (prípadne že nebol spárovaný), s akou farbou hral, a s akým výsledkom sa skončil daný zápas pre daného hráča. Pre účely nášho párovania sú podstatné atribúty len `id`, `points` a `playedGames`.

Objekt `Player` má interne uložené aj ďalšie atribúty, ktoré sa inicializujú (aktualizujú) priebežne, počas behu programu. Medzi tieto atribúty okrem pomocných atribútov (privátnych premenných) patria:

1. `unpairedBySystem` typu `int` - hodnota atribútu znamená, že v ktorom kole bol hráč nespárovaný (ak ešte nebol, tak hodnota je -1) a teda ak už bol nespárovaný systémom, tak viac nemôže byť nespárovaný systémom;

2. `canPlayInCurrentRound` typu `boolean` - hodnota atribútu znamená, že či môže daný hráč v danom kole hrať;

3. `possiblePlayers` typu `List<Integer>` - hodnota atribútu obsahuje zoznam hráčov (teda len ich `id`), s ktorými môže daný hráč byť spárovaný[1]. Hráči v tomto zozname sú usporiadaní[2] podľa preferencie v zostupnom poradí (prvý v poradí je najprirodzenejší, posledný najmenej prirodzený).

Objekt `Player` okrem iných pomocných metód pre prácu s týmto objektom obsahuje metódu `public int compareTo(Player o)`, ktorá prekrýva všeobecnú metódu `compareTo` a porovnáva hráčov podľa určených atribútov, konkrétne v našom programe primárne podľa bodov (viac bodov = lepšie) a v prípade zhody podľa štartovného čísla (menšie je lepšie).

4.3 Objekt (trieda) Edge

Objekt `Edge` poskytuje dva typy konštruktorov:

`public Edge(int from, int to)` a

`public Edge(int from, int to, long weight)`,

[1] spárovaný vzhľadom na pravidlo o nemožnosti spárovania dvakrát s tým istým hráčom, ostatné pravidlá (o farbách a pod.) sa riešia neskôr

[2] Toto usporiadanie rieši trieda `getPairing`, podľa komparátora `PGWcomparator`.

kde `from` a `to` znamenajú začiatočný a koncový bod hrany, ktorú daná inštancia triedy `Edge` reprezentuje. Ak je v konštruktore aj argument `weight`, tak sa pre danú hranu nastaví váha z konštruktora, inak je váha nastavená na 1. Trieda `Edge` má okrem základných atribútov (`from`, `to`, `weight`) aj ďalšie atribúty, konkrétne `isChecking` typu `boolean` (tento atribút slúži na označenie počas algoritmu, či sa testuje niektorý hráč, ktorý sa nachádza v aktuálnej hrane) a atribút `isPossible` typu boolean. Tento atribút je `true` práve vtedy, keď je daná hrana ešte k dispozícii a `false` práve vtedy, ak niektorý z hráčov v tejto hrane (alebo aj obaja) je vylúčený (nemôže[3] byť spárovaný z nejakého dôvodu).

Objekt `Edge` obsahuje ešte (okrem iných) dve dôležité metódy:

1. `public Edge switchVerticesInCurrentEdge()`, ktorá ma za úlohu vymeniť vrcholy `from` a `to` (toto je dôležité až v poslednej fáze párovania, keď už je rozhodnuté, ktoré hrany budú vo výslednom párovaní a rozhoduje sa už len o farbách);

2. `public void printEdge(FileWriter myWriter)`, ktorá zabezpečuje výpis hrany do súboru `myWriter`, prípadne na konzolu, ak nie je určený súbor (ak `myWriter == null`).

4.4 Objekt (trieda) `Color`, `ResultOfGame`

Trieda `Color` a `ResultOfGame` slúži ako „obal“ pre stringovú reprezentáciu.

Pre triedu `Color` vieme určiť, ktoré stringové označenie farby sa považuje sa White, ktoré za Black a ktoré za žiadnu farbu. V prípade zmeny nemusíme tieto údaje meniť v celom programe, ale len v tejto triede.

Pre triedu `ResultOfGame` vieme určiť, ktoré stringové označenie výsledku zápasu je akého typu (regulárny zápas, kontumačný zápas) alebo v prípade „bye“ (nespárovaný) sa považuje tento typ za nespárovaného. Ďalej vieme určiť, ktoré stringové označenie výsledku zápasu sa považuje za výhru (1.0 bodu), prehru (0.0 bodu) alebo remízu (0.5 bodu), resp. v prípade typu výsledku „nespárovaný“ to tiež vieme v tejto triede určiť. V prípade zmeny, nemusíme tiež tieto údaje meniť v celom programe, ale len v tejto triede. V prípade, ak by bol systém bodovania iný (napr. 3.0/0.5/0.0), vieme toto v tejto triede ľahko zmeniť a nemusíme to robiť v celom programe.

[3]Hráči sa vylučujú postupne (a teda vlastnosť „nemôže byť spárovaný“ môže vzniknúť aj počas algoritmu) v prípade, že kardinalita je menšia ako maximálna možná kardinalita vzhľadom na počet hráčov.

4.5 Párovací algoritmus (trieda getPairing)

Na začiatku hlavnej metódy `main` (ktorá sa bude spúšťať) sa nastavia pracovné premenné (argumenty) podľa prepínačov, ktoré boli zadané. Bližšie informácie o prepínačoch (argumentoch) a syntaxi sú v popise elektronickej prílohy.

Nasleduje prečítanie súboru (pomocou triedy `Reader`, ktorú sme opísali vyššie). Po prečítaní vstupného súboru sa inicializujú potrebné premenné, vytvorí sa zoznam hráčov `ArrayList<Integer> players` (zoradený zostupne, podľa metódy `compareTo()` objektu `Player`), ktorí môžu v danom kole hrať a vytvorí sa zoznam všetkých prípustných hrán `ArrayList<Edge> edges`[4], s jednotkovou váhou. Výnimku z jednotkovej hrany budú mať len hrany, ktoré obsahujú hráča, ktorý musí byť spárovaný (už bol nespárovaný systémom a pod.) - váha danej hrany zvýši[5] o $n + 1$, kde n je maximálna kardinalita párovania (t.j. zvýšenie o maximálny možný počet hrán v párovaní $+1$), za každého nespárovaného hráča[6]. Zoznam prípustných hrán sa vytvorí tak, že sa postupne prejdú pre všetkých hráčov zoznamy, ktoré vráti pre každého hráča zvlášť metóda `getPossiblePlayers()`, pričom hrana sa do zoznamu pridá iba v prípade, ak je práve prehľadávaný hráč lepší ako hráč s ktorým má vytvoriť hranu (podľa zoznamu z metódy `getPossiblePlayers()`).

Ďalej sa vytvorí graf s vrcholmi reprezentujúcimi `id` hráčov, ktorí môžu v danom kole hrať a prípustnými hranami zo zoznamu edges. Keďže v programe budeme často pristupovať k hranám, ktoré patria konkrétnym hráčom (ktoré obsahujú konkrétneho hráča), tak si ku každému hráčovi vytvoríme zoznam indexov, na ktorých sa tieto hrany nachádzajú v zozname `edges`.

Nasleduje zistenie maximálnej kardinality a vyradenie potrebného počtu hráčov tak, aby sme vedeli z nevyradených hráčov (a ich hranami) vytvoriť perfektný graf. Kardinalitu zistíme jednoducho tak, že spustíme MM (MaximumMatching) s aktuálne vytvoreným grafom a počet hrán vo výsledku MM bude kardinalita. Počet hráčov,

[4] `ArrayList` preto, lebo budeme často pristupovať na konkrétny prvok (hranu) tohto zoznamu a keďže má objekt `Edge` atribút `isPossible`, tak z tohto poľa nebudeme hrany vymazávať (v prípade potreby, keď niektoré hrany nebude možné z rôznych príčin použiť) a teda bude zachované počiatočné poradie.

[5] Toto číslo je dostatočné pre $\lfloor 2n \rfloor$ hráčov na vstupe, keďže pri $\lfloor 2n \rfloor$ hráčoch môže byť maximálne n hrán. Keďže každá hrana je jednotková, tak maximálny súčet váh v ľubovoľnom párovaní môže byť len n a zvýšenie o $n+1$ spôsobí zvýhodnenie hrany tak, aby výber tejto zvýhodnenej hrany bol prioritnejší ako výber všetkých zvyšných hrán spolu, ale len s váhou 1. Ak existuje párovanie so všetkými hranami také, že v ňom nebude hráč (ozn. H_S), ktorý musí byť spárovaný (musí byť v niektorom páre), tak zvýšenie váhy v tej hrane, kde je hráč H_S spôsobí výber práve tejto hrany i napriek nižšej kardinalite (a existencií párovania bez hráča H_S), ale so zachovaním povinného pravidla, že hráč H_S nemôže byť dvakrát nespárovaný.

[6] Ak je v hrane jeden nespárovaný hráč systémom, tak váha danej hrany je $(n + 1) + 1 = n + 2$, ak obaja, tak $2(n + 1) + 1 = 2n + 3$, ak žiaden, tak 1.

ktorých musíme vyradiť potom vyrátame ako $\#H - (2K)$, kde $\#H$ znamená počet všetkých hráčov, ktorý boli vo vstupnom grafe (a teda mohli hrať v losovanom kole) a K znamená vypočítanú kardinalitu.

Experimentálne sme zistili, že v prípade nepárneho počtu hráčov (vrcholov) trvá omnoho dlhšie nájsť kardinalitu pomocou MM ako pri párnom počte hráčov. Preto sme spravili vylepšenie, že ešte pred samotným zisťovaním kardinality v prípade nepárneho počtu hráčov skúsime vyradiť jedného (najhoršieho) hráča (keďže je nepárny počet hráčov, tak aspoň jeden musí byť vyradený). Ak sa ho podarí vyradiť, t.j. kardinalita bude $\frac{n}{2}$ (n je počet hráčov bez aktuálne vyraďovaného), tak hráč je úspešne vyradený a zvyšní hráči a ich hrany budú tvoriť perfektný graf, preto môžeme preskočiť bod 1 a 2 z pseudokódu (pseudokód z kapitoly 3, algoritmus výroby perfektného grafu) a hneď ísť na samotné párovanie (bod 3). V prípade, že kardinalita po vyradení nebude $\frac{n}{2}$ (bude menšia), tak sa pokračuje štandardným spôsobom, t.j. od bodu 1 v pseudokóde. V nekrajných prípadoch (väčšina prípadov) sa tohto hráča podarí vyradiť hneď. Vyraďovanie potrebného počtu hráčov prebieha tak ako v postupe napísanom v kapitole 3 (konkrétne bod 2). Pri bode (2a) aplikujeme na graf metódu `removeVertex(id)`, ktorá z grafu automaticky vymaže aj všetky hrany, v ktorých sa nachádzal hráč `id`. Bod (2b) aplikujeme na graf, ktorý sme upravili v bode (2a). V prípade úspešného vymazania hráča (bod 2c) sa danému hráčovi nastaví atribút `cantPlayInCurrentRound` pomocou metódy `cantPlay()`, a zároveň sa pre všetky jeho hrany nastaví atribút `isPossible` na `false`. Ak hráč nebol úspešne vymazaný, tak sa hráč, spolu s jeho vymazanými hranami (v bode (2a)) vráti späť do grafu a pokračuje sa od bodu (2a) s ďalším hráčom v poradí odspodu.

Po vyradení potrebného počtu hráčov nasleduje samotné párovanie, ktoré bude mať na vstupe perfektný graf. Samotné párovanie sme robili viacerými spôsobmi (porovnanie efektívnosti v kapitole 5), preto párovacie programy budú implementovať rozhranie `Pairing`, ktoré obsahuje jedinú metódu, `List<Edge> getMatching()`.

Po dokončení samotného párovania (ešte pred volaním metódy `getMatching()`) sa pole výsledných párov, ktoré vracia metóda `getMatching()` usporiada podľa bodov, ktoré majú hráči v jednotlivých výsledných hranách s využitím komparátora `EdgesOutputComparator`. Pre nespárovaných hráčov (nespárovaných ešte pred zavolaním párovacej triedy) pridáme do poľa výsledných párov hrany $[H, 0]$, kde H je `id` nespárovaného hráča (pre každého nespárovaného hráča jednu).

Keďže metóda `getMatching()` vráti len samotné páry, ešte bude potrebné určiť, kto z páru bude biely a kto čierny. Toto bude mať na starosti metóda

`printPairsOut(String oF, List<Edge> out, Map<Integer, Player> hm)`,

kde `oF (outputFile)` bude výstupný súbor alebo konzola v prípade hodnoty `null`, `out` bude zoznam párov z párovania, a `hm` je mapa objektov `Player` na základe ich `id`.

4.5.1 Trieda PMNW

Prvý spôsob je s využitím triedy `PairingMethodByNotWeighting` (PMNW), ktorý funguje tak, že program nájde v zozname `edges` prvú hranu, ktorá má atribút `isPossible` nastavený na `true` a vymaže z grafu oboch hráčov, ktorí sú obsiahnutí v tejto hrane (spolu s ich hranami) a pomocou MM vyskúša, či sa v danom grafe zmenšila kardinalita[7] presne o 1 (keďže sme našli už jednu hranu, ktorá bude vo výslednom párovaní). Ak sa kardinalita nezmenšila presne o 1, tak sa daní dvaja hráči spolu s ich hranami vrátia späť do grafu a pokračuje sa s ďalšou hranou v zozname edges, ktorá má atribút `isPossible` nastavený na `true`, až kým sa neprejde celý zoznam. Postupne vyraďované hrany (ktoré sme úspešne vyradili a nevrátili späť do grafu) vytvoria finálny zoznam párov, ktoré bude potom vracať metóda `getMatching()`. Tento algoritmus zohľadňuje preferencie len pri každom hráčovi zvlášť, t.j. nezohľadňuje preferencie globálne (aj vzhľadom k iným hráčom), keďže každého hráča páruje samostatne. Priorita hráčov je v tomto algoritme daná pevne pre každého hráča samostatne, podľa toho ako má daný hráč zoradených možných súperov.

4.5.2 Trieda PMGW

Druhý spôsob je s využitím triedy `PairingMethodByGroupWeighting` (PMGW). V tejto triede sa vytvorí graf tak, že pre každú prípustnú hranu zo zoznamu hrán `edges` sa do grafu pridá táto hrana s váhou rovnajúcou sa súčinu bodov, ktoré majú súperi v danej hrane. Váhu každej hrany ešte vynásobíme konštantou 4, keďže po vynásobení bodov s desatinnou časťou .5 nám vzniknú dve desatinné miesta (.25) a keďže máme obmedzený počet desatinných miest (iba 9), tak potrebujeme spraviť z váhy hrany pred aplikáciou pravidiel o preferenciách celé číslo. Tento spôsob slúži na celkovú minimalizáciu rozdielu bodov v spárovaných hranách, preto v krajnom prípade sa môže stať, že práve rozdiel v páre s najlepším hráčom bude väčší ako rozdiely v hranách s horšími hráčmi.

V prípade, že existujú hrany s rovnakým súčinom (t.j. každá porovnávaná hrana obsahuje hráča, ktorý má k bodov a l bodov a teda súčin bude kl[8]), aplikuje sa pravidlo s preferenciou farieb. Najprv sa aplikuje pravidlo o diferencii farieb a potom pravidlo koľkokrát mal hráč aktuálne po sebe poslednú farbu (počet poslednej farby). Snažíme sa eliminovať hrany také, že po odohraní losovaného kola bude mať niektorý z hráč hrany rozdiel farieb (diferenciu) 2, resp. -2 (vynútenú farbu) v prípade pravidla o

[7] maximálny počet párov, ktoré sa dajú v grafe spárovať po odobratí hrany

[8] Rovnaký súčin môžu mať aj hrany s inými hráčmi, napríklad hrana s hráčmi, ktorí majú počet bodov 2 a 4 bude mať rovnakú váhu (2*4=8) ako hrana s hráčmi, ktorí majú počet bodov 8 a 1 (8*1=8). V tomto prípade by sa (ak to perfektné párovanie dovolí) vybrala v párovaní hrana *A*-*B* a *C*-*D*, kde hráči A, B, C, D majú 8, 4, 2 a 1 bod.

diferencií a počet poslednej farby 2, resp. -2 (vynútenú farbu) v prípade pravidla o počte poslednej farby.

Ak má hráč diferenciu 1 alebo 2, tak prirodzený súper z hľadiska farieb je pre tohto hráča je súper s diferenciou -2, -1 alebo 0. Podobne, ak má hráč diferenciu -2 alebo -1, tak prirodzený súper z hľadiska farieb je pre tohto hráča je súper s diferenciou 0, 1 alebo 2. Takúto hranu zvýhodnime tak, že tejto hrane navýšime váhu o hodnotu, ktorá neovplyvní samotné vyberanie párov. Nech n je počet párov, ktoré sa dajú spárovať (polovica z počtu hráčov v perfektnom grafe, ktorý párujeme). Hodnota $d = \lceil \log_{10} n \rceil$ je počet cifier, do ktorých sa zmestí počet párov. Hodnota, o ktorú sa zvýši aktuálna váha hrany (a neovplyvní výber párov z predošlého bodu) bude 10^{-d}. Hodnota d preto, lebo d je počet miest potrebných na zapísanie počtu spárovaných hrán (krajný prípad, keby sme chceli zvýhodniť všetky hrany). Takto navýšené váhy zohľadnia preferencie o diferencii v prípade, ak existujú rovnako dobré hrany (s rovnakým súčinom).

Ak má hráč počet poslednej farby 1 alebo 2, tak prirodzený súper z hľadiska farieb je pre tohto hráča súper s počtom poslednej farby -2, -1 alebo 0.[9] Podobne, ak má hráč počet poslednej farby -2 alebo -1, tak prirodzený súper z hľadiska farieb je pre tohto hráča je súper s počtom poslednej farby 0, 1 alebo 2. Takúto hranu zvýhodnime tak, že jej navýšime váhu o hodnotu, ktorá neovplyvní samotné vyberanie párov a tiež párovanie podľa vyrovnávania diferencií. Hodnota, o ktorú sa zvýši aktuálna váha hrany (a neovplyvní výber párov podľa predošlých dvoch odsekov) je v tomto prípade 10^{-2d}, keďže d je počet desatinných miest vyhradený pre aplikáciu preferenčného pravidla o diferenciách a d je počet miest potrebných na zapísanie počtu spárovaných hrán (opäť krajný prípad, keby sme chceli zvýhodniť všetky hrany). Takto navýšené váhy zohľadnia preferencie o počte poslednej farby v prípade, ak existujú rovnako dobré hrany aj po aplikácii postupu z predošlého odseku.

4.5.3 Trieda PMSW

Tretí spôsob je s využitím triedy `PairingMethodBySemiWeighting` (PMSW). Na začiatku tejto triedy si vytvoríme pole `points` (usporiadanú množinu, keďže pre každú bodovú hodnotu existuje práve jedna skupina), do ktorého uložíme zostupne jednotlivé počty bodov, ktoré hráči na vstupe (`playersSorted`) majú. Vytvoríme graf z prípustných hrán zo zoznamu `edges` (každá hrana bude mať v grafe váhu 0). Ďalej vytvoríme dátové štruktúry

`TreeMap<Double, ArrayList<Integer>> playersGroupMap`,
`TreeMap<Double, ArrayList<Integer>> playersGroupEdgesMapPointers`,
`TreeMap<Double, ArrayList<Integer>> playersGroupLowerEdgesMapPointers`.

Do mapy `playersGroupMap` (PM) uložíme informáciu o tom, do ktorej bodovej

[9] Hráč môže mať počet poslednej farby 0 len v prípade, ak ešte nehral.

skupiny patrí každý hráč (ktorá skupina má akých hráčov).

Do mapy `playersGroupEdgesMapPointers` (PEMP) uložíme pre každú skupinu hrany[10] také, že obaja hráči z hrany patria do tejto skupiny a zároveň je daná hrana prípustná.

Do mapy `playersGroupLowerEdgesMapPointers` (PLEMP) uložíme zvyšné hrany hráčov z danej skupiny, ktoré sme neuložili do predošlej dátovej štruktúry (opäť len prípustné hrany). Tieto hrany sa budú využívať v prípade downFloaterov.

Teraz keď už máme pripravené všetky potrebné štruktúry môžeme postupne párovať všetky skupiny samostatne, od najlepšej po najhoršiu. Pri párovaní hráčov konkrétnej skupiny zmeníme pre každú hranu z ich skupiny (získame ich z mapy PEMP) váhu z 0 na *1 + (číslo, ktoré zohľadňuje farby a je popísané pri predošlom algoritme)*.

Potom spustíme MaxWPM a z výstupu vyberieme hrany, ktoré boli vznikli spárovaním hráčov z aktuálnej skupiny: keďže prvý hráč v hrane je vždy lepší, tak hrana patrí tejto skupine práve vtedy, ak počet bodov prvého hráča je rovný počtu bodov aktuálnej skupiny. Takéto hrany pridáme do poľa `foundedPairs`, ktoré bude potom vracať metóda `getMatching()` a týchto spárovaných hráčov vymažeme z grafu (a teda sa vymažú z grafu aj ich hrany).

Hráči, ktorých sme nespárovali v aktuálnej skupine budú *downFloateri* a ich hranám (z mapy PLEMP) zmeníme v grafe váhu na $(\lfloor \frac{ec}{2} \rfloor + 1) * (\textit{body súpera})^2 + (\textit{číslo, ktoré zohľadňuje farby a je popísané pri predošlom algoritme})$, kde ec je počet hráčov aktuálnej skupiny (bez downFloaterov z predošlej skupiny). Konštanta $\lfloor \frac{ec}{2} \rfloor + 1$ bude stačiť z dôvodu, že viac ako $\lfloor \frac{ec}{2} \rfloor$ hrán s váhou 1 sa nemôže vybrať, keďže hráčov z aktuálnej skupiny je ec. Túto váhu ešte pred aplikáciou preferenčných pravidiel musíme vynásobiť konštantou 4 pre rovnaké dôvody, ako sú popísané v predošlom algoritme. Takto pokračujeme až kým nespárujeme všetky skupiny a potom vrátime finálny zoznam párov pomocou metódy `getMatching()`.

Počet volaní MaxWPM a príprava grafu pre MaxWPM zaberá pri veľkých vstupoch nezanedbateľný čas, preto v zvyšnej časti tejto kapitoly popíšeme teoretický algoritmus (teoretické váhovanie) taký, aby stačilo MaxWPM volať len jeden krát. Keďže váhy budú každou skupinou narastať exponenciálne, tak veľmi rýchlo narazíme na obmedzenú aritmetiku (v prípade premennej typu `double` je v implementácii BLOSSOM V vrchné obmedzenie 10^{10} a spodné obmedzenie 10^{-9}) a preto opíšeme tento algoritmus len v teoretickej rovine. Keď chceme zavolať MaxWPM len raz, tak musíme zabezpečiť, aby sa primárne párovala najlepšia skupina (aj s downFloatermi), potom ďalšia, ..., až sa na konci spáruje najhoršia skupina. Zároveň musí platiť, že párovanie nižšej skupiny nesmie ovplyvniť párovanie vyššej skupiny. Teraz popíšeme, ako určiť váhy v jednej (ľubovoľnej) skupine:

[10] Do týchto dátových štruktúr ukladáme len index, kde sa daná hrana nachádza v zozname všetkých hrán `edges`.

V každej skupine máme dva typy hrán. Prvý typ sú hrany hráčov z danej skupiny (obaja hráči v hrane patria do danej skupiny), označme si tieto hrany ako *hrany skupiny*, skrátene HS a druhý typ hrán sú *hrany downFloaterov*, skrátene HD (prvý hráč patrí do danej skupiny a druhý hráč patrí do nižšej skupiny). Výberom váh treba zabezpečiť, že v danej skupine sa budú primárne vyberať HS, až potom HD. Hranám downFloaterom priradíme váhu $s^2 * a$, kde s je počet bodov súpera downFloatera (hráča na pozícií `to`). Parameter a je konštanta, jej význam vysvetlíme na konci tejto podkapitoly. Každá skupina má inú konštantu a každá skupina má vždy len jednu konštantu a. V najhoršom prípade sa pre každého hráča vyberie hrana downFloatera a zároveň v každej takejto hrane bude mať downFloater a súper downFloatera minimálny rozdiel bodov (súper downFloatera bude mať vždy menej bodov ako downFloater). Vtedy bude súčet váh týchto hrán neprevýši $p * s^2 * a$, kde s je najvyšší počet bodov spomedzi všetkých súperov downFloaterov a p je počet možných downFloaterov v danej skupine a teda aj počet hráčov v danej skupine. Váha každej hrany skupiny musí byť vyššia ako súčet všetkých HD, preto *gew* - groupEdgeWeight bude $p * sg^2 * a$, kde sg označuje počet bodov skupiny (vždy vyšší ako najvyšší počet bodov downFloatera). V štandardnom (nekrajnom) prípade sa vyberie $\lfloor \frac{gpc}{2} \rfloor$ hrán skupiny, kde *gpc* je počet hráčov v skupine (groupPlayerCount) a teda maximálny súčet[11] váh hrán z tejto skupiny môže byť $gew * \lceil \frac{gpc}{2} \rceil$. Celkový maximálny možný súčet všetkých hrán (HD a HS) v danej skupine teda je $gew * \lceil \frac{gpc}{2} \rceil = ((p * sg^2) * a) * \lceil \frac{gpc}{2} \rceil$. Teraz, keď už máme váhy v jednotlivých skupinách vyriešené, zostáva ešte vyriešiť, aby hrany z nižšej skupiny neovplyvňovali výber hrán z vyššej skupiny. Na to využijeme parameter a, ktorý bude inicializovaný na hodnotu 1 a pri prechode do ďalšej skupiny sa spraví operácia *a *= ((maximálny súčet z aktuálnej skupiny)+1)*. Váhy hranám v skupinách začneme vyrábať od najhoršej skupiny po najlepšiu skupinu.

4.6 Trieda GP a GRF

Trieda *G*P (`GetPairingByJaVaFo`) robí párovanie podobným systémom ako trieda `getPairing`. Program po spustení načítava prepínače (argumenty) a nastaví potrebné parametre (vstupný súbor, výstupný súbor, cestu k JaVaFo, viac v manuáli podobne ako pri triede `getPairing`), ale namiesto interného algoritmu na párovanie využíva algoritmus párovania JaVaFo.

Trieda *G*RF (`GetRandomGeneratedFileByJVF`), v ktorej sa tiež dajú nastaviť parametre pomocou prepínačov (argumentov), slúži ako náhodný generátor TRF súborov (záznamov o turnaji).

[11] Zaokrúhlenie *gpc* nahor obsahuje prípad vybratia minimálne jednej hrany downFloatera v prípade nepárneho počtu hráčov.

Kapitola 5

Porovnanie nášho algoritmu s JaVaFo

V tejto kapitole spravíme porovnanie dvoch implementácii pomocou nášho algoritmu a porovnanie s implementáciou JaVaFo. Porovnáme kvalitu losovania (ktoré je lepšie) podľa preferenčných pravidiel a tiež čas, ktorý trvá losovanie. Našli sme ešte existujúci algoritmus bbpPairings [15] o ktorom sme zistili, že sa správa tak isto ako JaVaFo, akurát je omnoho rýchlejší, keďže je napísaný v jazyku C++. V popise pre tento algoritmus sa píše, že tiež využíva podproblém Maximum Weighted Matching.

Budeme sa odkazovať na dodržiavanie povinných pravidiel, preto si ich teraz očíslujeme:

1. V turnaji nemôžu byť dvakrát rovnaké dvojice hráčov (každý hráč môže byť spárovaný s každým maximálne raz).

2. Žiadny hráč nemôže mať priebežný rozdiel farieb väčší ako 2.

3. Žiadny hráč nemôže mať po sebe trikrát rovnakú farbu.

Konvencia pre pomenovanie testovacích (vstupných) súborov je nasledovná: súbory so vstupmi budú mať formát `vstup`*`P`*`_`*`RC`*`_`*`RT`*`.trf` alebo `in`*`P`*`_`*`RC`*`_`*`RT`*`.trf`, kde *`P`* znamená počet hráčov vo vstupe, *`RC`* znamená aktuálny počet kôl a *`RT`* znamená maximálny počet kôl. Ak má názov vstupného súboru len 2 argumenty (2 čísla), tak je vynechaný argument *`RT`*. Niektoré testovacie súbory budú mať za argumentmi ešte komentár (pred koncovkou `.trf`).

Na uľahčenie práce s JaVaFo sme napísali „wrapper“, ktorý zavolá JaVaFo. Tento wrapper je trieda `GetPairingByJaVaFo`, ktorú sme popísali v predošlej kapitole. Pre prácu s JaVaFo za účelom náhodného generovania turnajov sme napísali wrapper (triedu `GetRandomGeneratedFileByJVF`), ktorú sme tiež popísali v predošlej kapitole.

5.1 Analýza štandardných vstupov

V tejto podkapitole porovnáme náš algoritmus s JaVaFo na štandardných vstupoch. Séria týchto vstupov sa nachádza v priečinku `src/tests/00`. Výsledky testov jednotlivých vstupov sú napísané v tabuľke 5.1. Čas testovania malých vstupov ovplyvňuje hlavne čas vstupno-výstupných operácii, ktorý je rádovo v 100ms. Pri väčších vstupoch, kde už vstupno-výstupné operácie nemajú na celkový čas veľký vplyv je vidieť časový rozdiel. Z celkových časov nemusí presne vyplývať podľa akej funkcie narastá čas, keďže vstupy sa generujú náhodne a teda voľnosť pri pároch môže byť v každom teste odlišná ale zhruba sa dá vyčítať, ako rastie čas pri každom algoritme s narastajúcou veľkosťou vstupov. V prípade JVF narastá čas približne lineárne. V prípade NWM narastá čas prudko exponenciálne, keďže pri každom páre sa overuje, či sa dá zvyšok spárovať. V prípade SWM a GWM je časový nárast skoro rovnaký, čas pri SWM stúpa trochu rýchlejšie (avšak podstatne menej ako pri NWM) časový nárast pri SWM závisí hlavne od počtu kôl (počtu skupín hráčov) a od počtu downFloaterov.

5.2 Analýza krajných vstupov

V tejto podkapitole porovnáme náš algoritmus s JaVaFo na krajných vstupoch a popíšeme chyby JaVaFo.

Prvá séria krajných vstupov sa nachádza v priečinku `src/tests/01`. Táto séria má za úlohu zistiť, ako sa správa algoritmus v prípade, ak ide párovať už posledné kolo a nepodarí sa dodržať pravidla (1), (2) a (3) súčasne.

Prvý krajný vstup (nachádza sa v súbore `vstup4_2_3.trf`) má za úlohu zistiť, ako sa budú algoritmy správať v prípade, ak sa už dve kolá odohrali, a posledné kolo sa už nebude dať spraviť, kvôli pravidlu (2) a (3). Vynútené páry, vzhľadom k pravidlu (1) budú *1-4* a *2-3*. Keďže podľa pravidiel (2) a (3) musia hrať obaja v prvom páre s bielou farbou (w) a obaja v druhom páre s čiernou farbou (b), tak tu je spor. JaVaFo sa zachová správne a vyhodí výnimku, že párovanie nie je možné na základe platných pravidiel. Takisto správne zareaguje aj náš algoritmus.

Druhý vstup v tejto sérii (cielene zlý - nachádza sa v súbore `vstup4_2_3_zly.trf`) porušuje pravidlo (1). Náš algoritmus zareagoval tak, že vypísal chybu (vyhodil výnimku `SamePairException`) spolu s popisom, kde sa chyba vyskytla (v tomto prípade druhé kolo a hráč s `id` 1). JaVaFo ignoruje kontrolu pravidiel a vygeneruje ďalšie kolo.

Tretí vstup v tejto sérii (nachádza sa v súbore `vstup6_4_5rozdiel_farieb.trf`) má za úlohu zistiť to, čo aj v prvom vstupe, akurát s viacerými hráčmi. JaVaFo vyhodil chybu, keďže podľa pravidla (1) sú vynútené páry *1-2*, *3-4* a *5-6*, ale prvý z týchto párov poruší podmienku (2). Náš algoritmus tento pár (porušujúci pravidlo (2)) vyradí a spáruje len zvyšných - teda výsledkom nášho algoritmu sú 2 páry (*3-4* a *5-6*) a dvaja

Tabuľka 5.1: Porovnanie štandardných vstupov medzi rôznymi algoritmami. Hodnoty v bunkách znamenajú čas v ms. Stĺpec JVF znamená použitie implementácie JaVaFo, stĺpec NWM znamená použitie nášho algoritmu bez váhovania, stĺpec SWM znamená použitie našej implementácie kde sa páruje každá skupina hráčov zvlášť, stĺpec GWM znamená použitie nášho algoritmu s váhovaním a grupovaním. Poznámky k tabuľke sú pod ňou.

Názov vstupu	JVF	NWM	SWM	GWM	poznámka
`in20_1_3.trf`	980	257	222	231	
`in52_4_9.trf`	979	594	677	422	
`in71_3_7.trf`	1378	910	554	453	(1)
`in128_3_10.trf`	1784	2302	1298	1236	(2)
`in129_3_10.trf`	1921	2084	1386	1038	
`in256_3_10.trf`	1967	7467	2813	2254	
`in257_3_10.trf`	2092	5930	2812	2626	
`in512_3_10.trf`	2625	51259	6916	5491	
`in512_10_20.trf`	4140	54392	12247	6234	
`in513_3_10.trf`	2915	54465	8722	6030	
`in600_1_5.trf`	3234	87793	6329	6038	
`in601_1_5.trf`	2711	104961	8151	7538	
`in1024_3_10.trf`	13556	554757	44229	30218	
`in1025_3_10.trf`	4915	572479	31544	26299	
`in2048_2_10.trf`	-	18081	5879	6357	(3) (4)
`in2049_2_10.trf`	-	17145	6669	5933	(3) (4)

(1) Vstup stiahnutý zo stránky [8] (zvyšné štandardné vstupy sú vygenerované pomocou generátora náhodných turnajov, s využitím JaVaFo).
(2) JVF, SWM a GWM mali celkový súčet diferencií 2.0, maximálnu diferenciu 0.5 a počet párov s diferenciou 4; NWM mal celkový súčet diferencií 3.0, maximálnu diferenciu 0.5 a počet párov s diferenciou 6.
(3) Generátor turnajov (s využitím JaVaFo) aj po opakovaných pokusoch nevygeneroval turnaje s 2048 a 2049 hráčmi, s tromi odohranými kolami; s dvomi kolami vygeneroval.
(4) Všetky 4 algoritmy skončili s chybou `heap space` (vyčerpanie miesta), preto sme skúsili náš algoritmus taký, že každému hráčovi bude generovať len troch súperov (troch najlepších súperov, z usporiadanej množiny ktorú vráti metóda `getPossiblePlayers()` a ktorí spĺňajú podmienky pre párovanie). Toto sme docielili pomocou argumentov „`-l`“ a „`3`“. V prípade veľkých vstupov sa výrazne zmenší graf čo sa týka počtu hrán, ale v krajnom prípade nemusí existovať platné losovanie, aj keď bez použitia tohto algoritmu s uvedenými argumentmi by to možné bolo. Algoritmus bbpPairings dokáže vygenerovať rýchlo aj párovanie s viac ako 2000 hráčmi.

hráči (1 a 2) sú nespárovaní.

Druhá séria krajných vstupov sa nachádza v priečinku `src/tests/02`. Táto séria má za úlohu zistiť, či sa berie ohľad na nasledujúce kolá (tie čo nasledujú po aktuálne losovanom).

Prvý krajný vstup (nachádza sa v súbore `vstup8_5_8.trf`) má za úlohu zistiť, či algoritmus uprednostní také páry, že hráči si v nich budú navzájom najprirodzenejší (preferenčné pravidlá budú dodržané v maximálnej možnej miere) pred neprirodzenými pármi, avšak pri výbere neprirodzených párov sa bude dať kompletne[1] losovať aj nasledujúce kolo (po aktuálne losovanom) a v prípade výberu prirodzených párov sa nasledujúce kolo nebude dať vylosovať kompletne. V tomto príklade bude mať tri kolá pred koncom hráč 1 (5.0 bodov) na výber dvoch možných súperov (s dodržaním pravidiel (1), (2) a (3)): hráča 2 (0.0 bodov) alebo hráča 3 (5.0 bodov). Ak algoritmus vyberie prirodzenú možnosť (t.j. pár *1-3*), tak v ďalšom (siedmom) kole nebude existovať kompletné losovanie. Oba algoritmy dajú ako výsledok prirodzené dvojice (v súboroch `vystupJVF8_5_8.txt` a `vystup8_5_8.txt`). Vo vstupe `vstup8_6_8alg.trf` bude záznam zo šiesteho kola, podľa výstupu z algoritmu. Z tohto vstupu vyplynú podľa pravidla (1) páry *1-2*, *8-3*, *4-6* a *7-5*. Na tomto vstupe JaVaFo vyhodí výnimku (keďže pár *1-2* porušuje pravidlo (2)), že párovanie nie je možné na základe platných pravidiel. Náš algoritmus vynechá pár *1-2*, ktorý porušuje pravidlo (2) a výsledkom teda budú páry *8-3*, *4-6* a *7-5* a dvaja hráči (1 a 2) sú nespárovaní.

Vo vstupe `vstup8_6_8neprirodzeny.trf` bude záznam, keby si algoritmus vybral neprirodzenú možnosť, (t.j. pár *1-2*). V tomto prípade losovanie siedmeho kola bude kompletné, t.j. oba algoritmy budú mať vo výsledku štyri páry (výsledok v súboroch `vystupJVF8_6_8neprirodzeny.txt` a `vystup8_6_8neprirodzeny.txt`).

Vo vyšších kolách JaVaFo ignoruje pravidlá (2) a (3) v poslednom kole. Túto možnosť poskytuje aj náš algoritmus, táto možnosť sa zapne pomocou prepínača `-clrOff`. JaVaFo na vstupe `vstupJVF8_6_7.trf` dá výstup (v súbore `vystupJVF8_6_7.txt`), ktorý porušuje pravidlo (2). Keďže sa losuje posledné kolo, zmenili sme počet kôl zo sedem na osem, aby výnimka v poslednom kole nemala vplyv na tento test.

Tretia séria krajných vstupov sa nachádza v priečinku `src/tests/03`. Táto séria má za úlohu zistiť, či sa preferuje kardinalita párovania pred prirodzenosťou, t.j. ak má hráč na výber prirodzeného a neprirodzeného súpera, pri výbere prirodzeného súpera by sa zvyšok nepodaril spárovať v súlade s pravidlami (1), (2) a (3), ale existuje kompletné losovanie v prípade výberu neprirodzeného súpera.

V tejto sérii testov je len jeden vstup (súbor `vystup6_3_5.trf`) kde sa práve táto podmienka overuje. Prirodzený pár by bol *1-2*, ale potom sa zvyšok nepodarí spárovať, preto oba algoritmy – JaVaFo aj náš – vyberú rovnaký neprirodzený pár, výsledok v

[1] kardinalita bude maximálna vzhľadom na počet hráčov

súboroch `vystupJVF6_3_5neprirodzeny.txt` a `vystup6_3_5neprirodzeny.txt`.

Ďalšie chybné správanie JaVaFo si opíšeme na príklade v priečinku `src/tests/04`. Vo vstupe `vystup6_3_5.trf` je cielená chyba, pri hráčovi 1 je dvakrát údaj, že bol nespárovaný systémom (porušenie povinného pravidla). JaVaFo toto pravidlo ignoruje, náš algoritmus skončí s vyhodením výnimky `RepeatableUnpairedException`.

Záver

V tejto práci sme porovnali niektoré losovacie systémy, ktoré sa používajú v turnajoch: švajčiarsky, round-robin, vyraďovací (stromový). Popísali sme, akým algoritmom prebieha losovanie systémom round-robin.

Zadefinovali sme základné pojmy, ktoré sme v práci používali. Definovali sme problém losovania švajčiarskym systémom (na vstupe je záznam turnaja vo formáte .trf, na výstupe sú spárovaní hráči), spísali sme povinné pravidlá pre losovanie týmto systémom na základe oficiálnych pravidiel [11]. Ďalej sme presne zadefinovali preferenčné pravidlá, určili sme priority preferenčných pravidiel (ktoré je dôležitejšie a ktoré je menej dôležité).

Stiahli sme si program JaVaFo (v jazyku Java), ktorý využíva pri losovaní aj oficiálna medzinárodná šachová federácia (FIDE). Tiež sme analyzovali aj program bbpPairings (v jazyku C++). Zoznámili sme sa aj so štandardným formátom .trf, ktorý sa využíva pri losovaní a ktorý využíva aj samotný program JaVaFo ako formát vstupného súboru, naučili sme sa interpretovať výstup JaVaFo.

Opísali sme, ako sa dá problém losovania švajčiarskym systémom transformovať na problém Maximum Weighted Matching a teda využiť riešenie tohto podproblému v našom algoritme.

Našli sme algoritmus Blossom (od Edmondsa), ktorý rieši problém Maximum Weighted Matching, porovnali sme rôzne verzie implementácií algoritmu Blossom až po aktuálnu BLOSSOM V od Kolmogorova. Zistili sme, že BLOSSOM V v C++ neposkytuje metódu nájdenia maximálneho párovania vo všeobecnom grafe, ale len v perfektnom, zatiaľ čo implementácia v Jave poskytuje metódu nájdenia maximálneho (minimálneho) párovania v perfektnom alebo všeobecnom grafe.

Opísali sme implementáciu nášho algoritmu pre losovanie švajčiarskym systémom, popísali sme dôležité triedy a metódy, ktoré sme využívali a aký vzťah majú dané triedy medzi sebou. Porovnali sme viacero verzií nášho algoritmu a jeho implementácie, postupne od prvej skúšobnej verzie, ktorá bola ešte v jazyku C++, až po aktuálnu verziu programu.

Zoznámili sme sa s knižnicami *jGrapht* a *jHeaps*, ktoré využívala implementácia BLOSSOM V v Jave, tiež sme zistili, ako vyrobiť vstup pre metódy MaximumWeightedPerfectMatching (MWPM) a MaximumWeightedMatching (MWM), ktoré posky-

tuje BLOSSOM V, a ktoré sme využívali. Táto implementácia umožňuje minimálne alebo maximálne párovanie.

Porovnali sme JaVaFo s implementáciou nášho algoritmu, analyzovali sme správanie na štandardných a krajných vstupoch. V štandardných vstupoch sme porovnávali rozdiely bodov v jednotlivých pároch a tiež koľko párov bude mať vynútenú farbu v nasledujúcom kole.

V štandardných vstupoch je náš algoritmus (hráči v skupinách podľa bodov, implementácia `PMSW`) približne rovnako dobre ako JaVaFo. Maximálna a priemerná diferencia bodov v pároch je rovnaká, v niektorých prípadoch sa v minimálnej miere líši počet vynútených farieb. V algoritme váhovania podľa počtu bodov (implementácia `PMGW`) sú diferencie bodov rovnaké, v niektorých prípadoch sa líši v počte vynútených farieb, tiež minimálne. V algoritme bez váhovania (implementácia `PMNW`) je v niektorých prípadoch rozdielna (minimálne rozdielna) aj diferencia bodov v pároch a je vyšší aj počet vynútených farieb.

V krajných vstupoch sa JaVaFo v porovnaní s našou implementáciou nesprával vždy správne. V prípade, ak sme mali na vstupe graf so šiestimi hráčmi a maximálna kardinalita vzhľadom na hrany mohla byť len 2, JaVaFo vyhodilo chybu, že párovanie neexistuje, zatiaľ čo náš algoritmus spároval aspoň tie 2 hrany, ktoré išli spárovať. JaVaFo zároveň nekontroluje logickú syntax vstupného súboru, t.j. akceptuje aj záznam turnaja, ktorý porušil povinné pravidlá (napr. hráč môže byť nespárovaný systémom maximálne raz). V prípade syntaktickej chyby v .trf súbore vypíše chybovú hlášku, z ktorej nie je známe, aký typ syntaktickej chyby bol vo vstupnom súbore, prípadne kde sa táto chyba nachádzala. Náš algoritmus logickú syntax kontroluje, a v prípade syntaktickej chyby vypíše, kde presne je chyba. Ďalej sme zistili, že program bbpPairings (v C++) pracuje tak ako JaVaFo, akurát pracuje omnoho rýchlejšie, keďže je napísaný v C++.

Literatúra

[1] Blossom algorithm. Dostupné z https://en.wikipedia.org/wiki/Blossom_algorithm, [Citované. 19/04/2023].

[2] Duplicate bridge. Hra Bridge (aj tu sa využíva švajčiarsky systém). Dostupné z https://en.wikipedia.org/wiki/Duplicate_bridge, [Citované 22/03/2023].

[3] Go (game). Hra Go (aj tu sa využíva švajčiarsky systém). Dostupné z https://en.wikipedia.org/wiki/Go_(game), [Citované 22/03/2023].

[4] Maximum weight matching. Dostupné z https://en.wikipedia.org/wiki/Maximum_weight_matching, [Citované. 13/05/2023].

[5] Round-robin tournament. Losovací systém round-robin. Dostupné z https://en.wikipedia.org/wiki/Round-robin_tournament, [Citované 22/03/2023].

[6] Scrabble. Hra Scrabble (aj tu sa využíva švajčiarsky systém). Dostupné z https://en.wikipedia.org/wiki/Scrabble, [Citované 22/03/2023].

[7] Swiss-system tournament. Využitie švajčiarskeho losovacieho systému. Dostupné z https://en.wikipedia.org/wiki/Swiss-system_tournament, [Citované 22/03/2023].

[8] Trf example. Príklad súboru .trf. Dostupné z http://www.rrweb.org/javafo/aum/TRFXSample2.txt, [Citované 18/05/2023].

[9] Vyřazovací systém. Stromový losovací systém. Dostupné z https://cs.wikipedia.org/wiki/Vyřazovací_systém, [Citované 22/03/2023].

[10] Naveh Barak a kol. Jgrapht. Knižnica JGraphT (knižnica na prácu s grafmi). Dostupné z https://jgrapht.org, [Citované 19/04/2023].

[11] International Chess Federation. Fide handbook. Pravidlá švajčiarskeho systému (celá kapitola C04, t.j. C0401-C0405). Dostupné z https://handbook.fide.com, [Citované 19/04/2023].

[12] International Chess Federation. Format of trf (tournament report file). Formát súboru „.trf“. Dostupné z https://www.fide.com/FIDE/handbook/C04Annex2_TRF16.pdf, [Citované 22/03/2023].

[13] Zvi Galil, Silvio Micali, and Harold Gabow. An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal on Computing*, 15(1):120–130, 1986. Dostupné z https://doi.org/10.1137/0215009, [Citované 15/05/2023].

[14] Schäfer Guido. Weighted matchings in general graphs. Diplomová práca, Max-Planck-Institut für Informatik in Saarbrücken, máj 2000. Algoritmus transformácie na graf, v ktorom existuje perfektné párovanie. Dostupné z https://homepages.cwi.nl/~schaefer/ftp/pdf/masters-thesis.pdf, kapitola 1.5.1, [Citované 22/03/2023].

[15] jbierma. bbpPairings. Algoritmus bbpPairings pre losovanie švajčiarskym systémom, Dostupné z https://github.com/BieremaBoyzProgramming/bbpPairings, [Citované 13/05/2023].

[16] Dimitrios Michail. Jheaps library. Knižnica JHeaps, ktorá je potrebná pre využívaní Blossom algoritmu v Jave. Dostupné z https://www.jheaps.org/, [Citované 19/04/2023].

[17] Ricca Roberto. JaVaFo, a pairing engine for the FIDE (Dutch) System. JaVaFo: existujúca implementácia losovania švajčiarskym systémom. Dostupné z http://www.rrweb.org/javafo/JaVaFo.htm, [Citované 19/04/2023].

[18] Kolmogorov Vladimir. Package org.jgrapht.alg.matching.blossom.v5. Implementácia algoritmu Blossom V v Jave. Dostupné z https://jgrapht.org/javadoc-1.4.0/org/jgrapht/alg/matching/blossom/v5/package-summary.html, [Citované 19/04/2023].

[19] Kolmogorov Vladimir. Software (Blossom V). Implementácia algoritmu Blossom V v C++. Dostupné z https://pub.ist.ac.at/~vnk/software.html, kapitola 4, [Citované 19/04/2023].

[20] Komogorov Vladimir. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, Júl 2009. Dostupné z https://doi.org/10.1007/s12532-009-0002-8, [Citované 15/05/2023].

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu a súbory s výsledkami testov (v priečinku `src/tests`).

Postup pre inštaláciu a spustenie: Priečinok `src`, ktorý sa nachádza v priečinku `pairing.zip`, je pracovný priečinok. Obsah priečinka `src` je možné dať do iného ľubovoľného pracovného priečinka.

1. Je potrebné sa v konzole prepnúť do pracovného priečinka `src` (resp. do iného, ak si ako pracovný priečinok zvolíte iný) – t.j. príkaz „`cd [path]/src`“.

2. Ak chcete aby bol pracovný priečinok iný ako `src`, tak obsah priečinka `src` je potrebné skopírovať do tohto priečinka.

3. Je potrebné skontrolovať, či máte nainštalovanú `javu` (mal by stačiť príkaz „`javac -version`“ alebo „`java -version`“) – my sme používali verziu `11.0.16`.

4. Ak ste v pracovnom priečinku, tak je potrebné program skompilovať príkazom
   „`javac -d . pairing/getPairing.java`“
   a vytvoria sa skompilované súbory s príponou `.class`.

5. Program spustíte príkazom „`java pairing.getPairing`“ a zobrazí sa Vám manuál.

V priečinku `tests` sú súbory, ktoré sme používali pri testovaní (tie súbory, ktoré sú spomenuté v kapitole 5).

Aktuálna verzia programu má 3 metódy, ktoré sa dajú spustiť (všetky sú v priečinku `pairing`).

- prvá a najdôležitejšia je metóda samotného párovacieho algoritmu, ktorý sa spúšťa pomocou triedy `getPairing`, tak, ako je to popísané vyššie

- druhá metóda umožňuje spustiť párovanie s využitím programu JaVaFo. Táto metóda sa spúšťa pomocou triedy `getPairingByJavaFo`

- tretia metóda umožňuje spustiť náhodné generovanie turnajov (ktoré sme využívali aj my pri testovaní) pomocou triedy `getRandomGeneratedFileByJVF`

Všetky tieto triedy sa dajú skompilovať a spustiť tak, ako je uvedené vyššie, t.j. príkazom

„`javac -d . pairing/[class].java`“

pre kompiláciu triedy „`[class]`“ a „`java pairing.[class] [args]`“ pre spustenie triedy `[class]`, kde `[args]` sú možné argumenty. V prípade, že nie sú uvedené žiadne argumenty, tak sa po spustení zobrazí manuál. Príklad príkazu pre spustenie programu:

„`java pairing.getPairing -i [vstup] -a -o [výstup]`“ .

Grafové knižnice, ktoré sme v programe využívali (JGraphT [10] a JHeaps [16]), sú v priečinku „`org`“ (všetky triedy sa nevyužívajú).