Losovanie turnajov švajčiarskym systémom.

Plán práce letného semestra:

- Spraviť párovací algoritmus
- Spraviť spracovanie outputu z TRF interpretera
- Dokončiť aplikáciu

Výsledok:

- Nájdenie vhodného blackBoxu na párovanie.
- Ošetrenie vstupu pre blackBox
- Teoretické spravenie celého párovacieho algoritmu (pseudokód)
- Otestovanie funkčnosti blossom5 na Linuxe (nevýhoda blossom5: funguje len pod Linuxom => musel som si nainštalovať OS Linux (doteraz som robil na Windowse, aby som mohol na Linuxe spúšťať program v jave z IntelliJ.

Stránka na JavaFo: http://www.rrweb.org/javafo/aum/JaVaFo2_AUM.htm

TRF formát: https://www.fide.com/FIDE/handbook/C04Annex2_TRF16.pdf

Odkaz na TRF interpreter *(a všetky súbory, ktoré sa spomínajú v tomto texte)*:

https://github.com/XL202/JavaFo

Odkaz na riešenie (weighted) maximum matching pomocou blossoms (program v C++, bez externých knižníc, ktorý budem využívať ako blackbox:

https://pub.ist.ac.at/~vnk/software.html

Časť 4. BLOSSOM V, odkaz priamo na program:

https://pub.ist.ac.at/~vnk/software/blossom5-v2.05.src.tar.gz

Ďalšie užitočné linky:

https://depth-first.com/articles/2019/04/02/the-maximum-matching-problem/

https://en.wikipedia.org/wiki/Blossom_algorithm

Manuál na skompilovanie blossom5:

1. Stiahnuť .tar file na linku vyššie.
2. Rozbaliť .tar file
3. Spustiť príkaz Makefile príkazom make v danom priečinku, ktorý sa vytvoril po rozbalení
4. spustiť blossom5 program: `./blossom -e <file_from> -w <file_output>`
5. Viď formát files nižšie.

Pôvodný program na párovanie:

https://github.com/BieremaBoyzProgramming/bbpPairings

Dôvod nepoužitia: vyžadovanie veľa externých knižníc

JavaFo nerieši čo bude v ďalšom kole, preto si vybral možnosť tak, akoby ďalšie kolo už nebolo. Navyše pomocou XXR sa určuje len maximálny počet kôl, nie povinný počet.

Musel som sa rozhodnúť, čomu dám prednosť:

A) Tak aby zostal nespárovaný jeden hráč (v prípade nepárneho počtu hráčov), prípadne 0 (keď je ich párny počet) – minimalizácia počtu nespárovaných hráčov; keď dané spárovanie vzhľadom na platné pravidlá neexistuje, tak vypíše, že sa nedá spárovať.

B) Minimalizácia počtu nespárovaných hráčov – spáruje maximálny možný počet hráčov – t. j. nevyhodí chybu, keď sa nedajú spárovať všetci hráči (resp. v prípade nepárneho počtu *všetci-1*).

Vybral som si možnosť B)

Program blossom5 (ďalej BB) nájde **LEN** perfektné párovanie, t. j. každý vrchol grafu (hráč) musí mať pár, a každý vrchol grafu bude mať práve jedného suseda – každý vrchol je stupňa 1 – vychádza z neho práve jedna hrana.

Vstupný súbor musí mať formát:

```
==================in.txt========================================
<count_of_vertices> <count_of_edges>
<list_of_all_edges: <edgeFrom edgeTo edgeValue>, each in new line>
================================================================
```

Problém je v tom, že všetky vrcholy musia byť v jednom rade, t. j. od 0 po n, lenže niekedy potrebujem spraviť párovanie bez vrcholu v strede radu (napr. `0, 1, 2, 5, 6`), preto musím vrcholy poposúvať tak, aby boli v jednom rade – na to využívam hashmapu (vo funkcii `bbCardinality`, ktorá mi prerobí všetky `edges` tak, aby bol súvislý rad vrcholov od 0 po n.

Ďalší problém je v tom, že nie vždy sa dá spraviť maximum matching (perfektné párovanie), preto potrebujem prerobiť vstup tak, aby sa maximum matching dal spraviť vždy.

Robil som to podľa nejakého .pdf dokumentu (ten algoritmus, len teraz neviem naň nájsť link). (Viď file pseudokod_konverzia_pre_BB.txt)

```
int BB(set possible_pairs) {
    //possible_pairs maju vahu 0
    //numbered from 0 to count of possible players (PP) - 1
    //struktura Edge: from, to, weight     (alebo player1, player2)
    pairs_tmp = possible_pairs;
    for(Edge e : possible_pairs) {
         pairs_tmp.add(new Edge(e.from +PP, e.to + PP, PP + 1));
    }
    //zduplikuje graf
    for(int i=0; i<PP; i++) for(int j=i+PP; j<2*PP; j++) {
         pairs_tmp.add(new Edge(i,j,PP+1));
     //pospaja kazde 2 vrcholy z povodneho a duplikovaneho grafu
    }
    return BB(pairs_tmp).count;
}
```

Princíp:

1. zduplikuj graf: pre každú hranu E(from, to, val) pridaj do grafu hranu E(from+count_of_vertices, to+c_o_v, val_max);
   (platné hrany pôvodného grafu - platné dvojice majú hodnotu 1;
2. Pre každý vrchol v pôvodnej časti grafu pridaj hranu do grafu druhej časti s hodnotou val_max.
3. Val_max vypočítaj ako počet všetkých možných hrán (platných) v pôvodnom grafe) + 1
4. Blossom5 vyberá primárne hrany s menšou váhou, preto celkový počet platných hrán bude: output z ((BB) % (počet vrcholov)) (t .j. budú sa ignorovať umelo pridané hrany, ktoré zakomponoval BB do párovania).

file robenie_dvojic.txt

každý hráč bude mať množinu hráčov, s ktorými môže hrať: (aby sa dodržali pravidlá o farbách a také, že nikto nemôže spolu hrať viac ako raz).

Hráčov musí byť párny počet, čo mi zabezpečí vyraďovanie hráčov.

Kmax = maximálna kardinalita (dané ako parameter alebo počet hráčov/2)

Input: all_possible_edges (PE)

zoradenie hráčov podľa bodov

skupina 1(G1): hráči podľa bodov do polovice => zoradené podľa bodov zostupne

skupina 2(G2): zvyšní hráči

nech hráč p má p.with() pole hráčov, s ktorými môže hrať

```
ArrayList<Edge> final_edges;

tmp_e = PE;
bool found = false;
for(Player p: fromG1) {
    for(Player pw: p.with()) {
      tmp_e -= new Edge(p, pw);
      int c = BB(tmp_e);
      if (c = Kmax -1) {
          final_edges.add(new Edge(p,pw));
          found = true;
          break;
      }
    }
    if (found) {
      Kmax -= 1;
      removePlayersPPwFromAllOfRemainingPlayersInBothGroups(p, pw);
      fromG1.remove(pw);
      found = false;
    }
}
```

```
tmp_e = PE;
bool found = false;
for(Player p: fromG2) {
    for(Player pw: p.with()) {
      tmp_e -= new Edge(p, pw);
      int c = BB(tmp_e);
      if (c = Kmax -1) {
          final_edges.add(new Edge(p,pw));
          found = true;
          break;
      }
    }
    if (found) {
      Kmax -= 1;
      removePlayersPPwFromAllOfRemainingPlayersInBothGroups(p, pw);
      fromG1.remove(pw);
      found = false;
    }
}
tmp_e = PE;
bool found = false;
for(Player p: (fromG1+fromG2)) {
    for(Player pw: p.with()) {
      tmp_e -= new Edge(p, pw);
      int c = BB(tmp_e);
      if (c = Kmax -1) {
          final_edges.add(new Edge(p,pw));
          found = true;
          break;
      }
    }
    if (found) {
      Kmax -= 1;
      removePlayersPPwFromAllOfRemainingPlayersInBothGroups(p, pw);
      fromG1.remove(pw);
      found = false;
    }
}
```

File pseudokod_nesparovany.txt

```
BB = BlackBox (program na zisťovanie kardinality)
BB(set of possible edges)
set of possible_pairs [A, B] = pp
set of players_in_pairs = pip;
zistiť kardinalitu pomocou BB
kardinalita = K
max_kardinalita = Km (počet hráčov/2) - zaokruhlene nadol
párnosť (parita počtu hráčov) = p (true/false)

vyradení hráči = vh;

ak K < Km && p = true, tak vyrad(2*(Km-K), pip, pp);
ak K < Km && p = false, tak vyrad(2*(Km-k)+1, pip, pp);

vyrad(int count, set pip, set pp) {
    if (count == 0) return;
    pip_tmp = pip;
    pp_tmp = pp;
    for(Player p: pip) {
     Player p;
     allPairsWithPlayer = findPairsWithPlayer(p);
     pip_tmp -= p;
     pp_tmp -= allPairsWithPlayer;
     k_tmp = BB(pp_tmp);
     if (k_tmp == K) {
         vh.add(hrac s najmensimi bodmi};
         vyrad(count-1, pip_tmp, pp_tmp);
         break;
        }
     pip_tmp += p;
     pp_tmp += allPairsWithPlayer;
//aktualne testovaneho hraca vrat späť do množiny hráčov, tiež aj
jeho hrany, pokracuj s hracom, s druhym najnižším počtom bodov, atď,
až kým sa nenájde taký, ktorého je možné vyradiť *)
    }
}

*ak na začiatku zistí BB, že je možné vyradiť hráča, tak sa určite
nájde taký, ktorého je možné vyradiť
```