Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Nerepetitívne zoznamové farbenia ciest

Diplomová práca

2021
Bc. Alojz Stúpal

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Nerepetitívne zoznamové farbenia ciest

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. RNDr. Robert Lukoťka, PhD.

Bratislava, 2021
Bc. Alojz Stúpal

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Alojz Stúpal
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Nerepetitívne zoznamové farbenia ciest
Non-repetitive list colourings of paths

Anotácia: Nerepetitívne farbenie je vrcholové farbenie grafu také, že pre každú cestu párnej dĺžky, postupnosť farieb na prvej polke cesty nie je identická s postupnosťou farieb na druhej polke cesty.Špeciálne, pre dĺžku cesty dva to znamená, že susedné vrcholy majú rôzne farby. Študujú sa mnohé varianty nerepetitívnych farbení jednym z nich sú zoznamové nerepetitívne farbenia. Cieľom je zistiť či je pre každé priradenie zoznamu farieb dĺžky aspoň k ku každému vrcholu grafu možné nájsť farbenie, kde každý vrchol dostane farbu zo svojho zoznamu. Tento problém je z aloritmického hľadiska mimoriadne ťažký, je Pi^p_2-úplný. Pri štúdiu nerepetitívnych zoznamových farbení je mnoho otvorených otázok. Jednou zo základných je: "Aké je nerepetitívne zoznamové chromatické číslo cesty". Je známe, že toto číslo je buď 3 alebo 4. Cieľom práce je s použitím výpočtových nástrojov získať čiastkové výsledky o nerepetitívnych zoznamových farbeniach ciest.

Vedúci: doc. RNDr. Robert Lukoťka, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 31.10.2019

Dátum schválenia: 12.12.2019 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Veľmi ďakujem svojmu školiteľovi, doc. RNDr. Robertovi Lukoťkovi, PhD., za jeho neochvejnú dôveru v moje schopnosti i za nápady, ktorými ma neustále zahŕňal, a nabádal tak k ďalšej a ďalšej práci. Ďakujem taktiež Jozefovi Šišolákovi za poskytnutie výpočtovej sily v posledných fázach diplomovej práce.

Abstrakt

V práci sa snažíme čiastočne dokázať hypotézu $\pi_{ch}(P_n) = 3$ pre každú cestu $P_n, n \geq 4$, a teda overiť, že pre každú cestu o dĺžke aspoň štyri a pre každé priradenie zoznamov o troch farbách ku každému vrcholu na tejto ceste je možné nájsť aspoň jedno nerepetitívne farbenie, pričom každému vrcholu priradíme farbu z jemu prislúchajúcemu zoznamu. Nerepetitívne farbenie je také vrcholové farbenie grafu, že pre každú jeho podpostupnosť párnej dĺžky sa postupnosť farieb prvej polovice tejto podpostupnosti nezhoduje s postupnosťou farieb druhej polovice. Hypotézu dokážeme za pomoci značne zoptimalizovaného programu, a to pre všetky cesty s dĺžkou menšou alebo rovnou číslu 10. Hypotézu dokážeme s obmedzením na počet možných farieb pre zoznamy vrcholov i pre cesty dĺžky 11 a 12.

Kľúčové slová: nerepetitívnosť, zoznamovosť, farbenie, cesta, graf

Abstract

In our work we are aiming for a partial proof of conjecture $\pi_{ch}(P_n) = 3$ for every path $P_n, n \geq 4$, in other words, to confirm, that for every path of length at least four and for every assignment of lists of three colors to every vertex on this path it is possible to find at least one non-repetitive coloring, whilst assigning one color to every vertex from list associated with it. Non-repetitive coloring is such a vertex coloring, that for every subsequence of even length derived from this coloring the sequence of colors of the first half of this subsequence does not match the sequence of colors of the second half. We will prove the conjecture using considerably optimized program for all paths with length less or equal to 10. With restriction on the number of allowed colors for lists we will also prove the conjecture for paths of length 11 and 12.

Keywords: non-repetitivness, list, coloring, path, graph

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Cieľom tejto práce je čiastočne dokázať hypotézu, že $\pi_{ch}(P_n) = 3$, pre každú cestu $P_n, n \geq 4$, a teda overiť, že pre každú cestu s aspoň štyrmi vrcholmi a pre každé priradenie zoznamov dĺžky 3 ku každému vrcholu cesty je možné nájsť nejaké nerepetitívne farbenie, kde každému vrcholu priradíme farbu z jemu prislúchajúceho zoznamu. Nerepetitívne farbenie je vrcholové farbenie grafu také, že pre každú jeho podpostupnosť párnej dĺžky sa postupnosť farieb prvej polovice tejto podpostupnosti nezhoduje s postupnosťou farieb tej druhej. Problém overenia hypotézy je Π_2^P–úplný. Našou úlohou je implementácia programu, ktorý bude hypotézu overovať. Jeho implementácia je pre účely práce kľúčová. Budeme ho preto v čo najväčšej miere optimalizovať a testovať, aby sme sa spoľahlivo dopátrali k platnosti hypotézy pre čo najdlhšie cesty a s použitím čo najväčšieho počtu možných farieb pre zoznamy vrcholov. Vo všeobecnosti sa budeme snažiť hypotézu dokázať pre všetky cesty dĺžky menšie alebo rovné 10 a pre všetky počty možných farieb taktiež menšie alebo rovné 10.

V prvej kapitole zadefinujeme mnohým známe, no pre našu prácu nevyhnutné pojmy. Zavedieme však i vlastné definície, ktoré nám pomôžu elegantnejšie a čitateľnejšie vyjadrovať svoje myšlienky a dokazovať potrebné tvrdenia.

V druhej kapitole uvedieme doterajšie výsledky týkajúce sa nerepetitívnych zoznamových farbení, prípadne iné, no stále našej téme blízke zistenia.

V tretej kapitole čitateľov informujeme o tvrdeniach, ktoré budú dokazovať správnosť našich implementácií. Pomocou definícií z predchádzajúcich kapitol vyslovíme domnienky, ktoré následne aj formálne dokážeme. Dokázané tvrdenia, vety či lemy nám potom vo štvrtej kapitole uľahčia a urýchlia vyjadrovanie sa, a umožnia tak čitateľom plne sa sústrediť na samotné myšlienky uvádzaných optimalizácií.

V poslednej kapitole, ako sme už naznačili, popíšeme samotnú implementáciu programu. Popíšeme návrh programu, no i jeho základnú, teda nezoptimalizovanú verziu. Vychádzajúc z nej načrtneme čitateľom čo najpresnejšie funkcionalitu implementovaných optimalizácií v chronologickom poradí. Každá podkapitola bude na správnych miestach popretkávaná výsledkami pre práve opisované vylepšenia. Vďaka tomu už neuvedieme samostatnú kapitolu pojednávajúcu o výsledkoch práce, no ich jemné zhrnutie i diskusiu k nim umiestnime do záveru.

Kapitola 1

Základné definície

V tejto kapitole uvedieme najskôr základné definície týkajúce sa grafov, opíšeme niektoré grafové štruktúry, prípadne zavedieme rôzne názvy, ktoré budeme v ďalších častiach práce potrebovať. Vďaka všetkým definíciám popíšeme problém zoznamového nerepetitívneho farbenia ciest. Vysvetlíme i pojmy ako sekvencia a i spôsob, akým sa dá zoznamovosť či nerepetitívnosť vo farbení vyjadriť ako sekvenčný problém.

1.1 Základné pojmy

Definícia 1 (Diestel, [8])**.** *Graf* je dvojica $G = (V, E)$ disjunktných množín, kde prvky E sú dvojprvkové podmnožiny V. Prvky V sú *vrcholy* (prípadne *uzly* alebo *body*) grafu G, prvky E sú jeho hrany.

Tento objekt je, presnejšie povedané, *neorientovaný graf*. Pre vytvorenie *orientovaného grafu* potrebujeme uvažovať, aby každý prvok E bol usporiadanou dvojicou. Hrany $xy, yx \in E$ by teda boli v tomto prípade rozdielne. My budeme v práci používať iba pojem *graf*, pričom budeme mať zakaždým na mysli *neorientovaný graf*. Hranu $\{x, y\}$, kde $x, y \in V$ zas budeme zapisovať ako xy.

Definícia 2 (Diestel, [8])**.** Pre veľkosť množiny vrcholov grafu G s označením $|V|$ budeme používať pojem *rád grafu*.

Definícia 3 (Diestel, [8])**.** Majme vrchol $v \in V$, a hranu $e \in E$. Vrchol v navzeme *incidentný* s hranou e, ak platí $v \in e$. Pojem *koncové vrcholy* bude označovať oba vrcholy x a y, $x \neq y$ pre ktoré platí $x \in e$, $y \in e$, kde $e \in E$. Vrcholy $x, y \in V$ nazveme *susedné*, ak $xy \in E$. Hrany $e, f \in E; e \neq f$ nazveme *susedné*, ak $x \in e; x \in e$, kde $x \in V$.

Definícia 4 (Diestel, [8])**.** Majme vrchol $x \in V$. *Stupňom vrcholu* nazveme počet vrcholov susedných s x.

Definícia 5 (Diestel, [8]). Nech $G = (V, E)$ a $G' = (V', E')$. Zavedieme označenia $G \cap G' = (V \cap V', E \cap E')$ a $G \cup G' = (V \cup V', E \cup E')$. Ak platí $G \cap G' = \emptyset$, potom G a G' nazveme *disjunktné*.

Ak platí $V' \subseteq V$ a $E' \subseteq E$, budeme hovoríť, že G' je podgraf G a písať $G' \subseteq G$.

Definícia 6 (Diestel, [8]). Majme graf $G = (V, E)$. *Sled* S bude označenie pre neprázdnu striedavú postupnosť $v_0e_0v_1e_1...e_{k-1}v_k$, kde $v_i \in V, e_j \in E$ pre $i = 1, 2, ..., k$ a $j = 1, 2, ..., k-1$, pričom opäť platí $e_i = v_iv_{i+1}$ pre všetky $i < k$. Dĺžka takéhoto sledu bude $|S| = k$.

Definícia 7 (Diestel, [8]). *Cestou* P budeme nazývať taký sled S, v ktorom navyše platí, že všetky v_i sú po dvoch rôzne. Dĺžka cesty bude $|P| = |S| = k$.

Definícia 8 (Diestel, [8]). *Ťah* T bude názov pre taký sled S, v ktorom navyše platí, že hrany e_j sú po dvoch rôzne. Dĺžka ťahu bude $|T| = |S| = k$.

Definícia 9 (Diestel, [8]). Majme graf $G = (V, E)$. *Cyklus* C (alebo aj *kružnica*) budeme nazývať taký sled S, v ktorom navyše platí, že všetky v_i sú po dvoch rôzne a $v_0 = v_k$. Dĺžka kružnice bude $|C| = |S| = k$.

Definícia 10 (Diestel, [8]). Ak sú prvky grafu ohodnotené číslom, resp. ak je zadefinovaná funkcia priraďujúca prvkom nejaké hodnoty, graf nazveme *ohodnotený*. *Hranovo ohodnotený graf* je graf, ktorému je pridelená funkcia $h : E \to R$. Číslo $h(e), e \in E$ nazveme *hodnota hrany* alebo aj *cena hrany*. *Vrcholovo ohodnotený graf* je graf, ktorému je priradená funkcia $h : V \to R$. Opäť, Číslo $h(v), v \in V$ nazveme *hodnota vrcholu* alebo aj *cena vrcholu*.

Definícia 11 (Diestel, [8]). *Vrcholové označovanie grafov* definujeme ako zobrazenie $c_1 : V \to S$ a *hranové označovanie grafov* ako zobrazenie $c_2 : E \to S$.

Definícia 12 (Diestel, [8]). Nech $G = (V, E)$ je graf. Zadefinujeme nasledovné:

- *Vrcholovým farbením* nazveme zobrazenie $c : V \to S$, pričom ak sú vrcholy x, y susedné, musí platiť $c(x) \neq c(y)$. Prvky množiny S nazývame farby.
- *Vrcholovým k-farbením* nazveme také vrcholové farbenie, v ktorom $S = \{1, 2, ..., k\}$.
- Graf je *vrcholovo k-zafarbiteľný*, ak má vrcholové k-farbenie.
- *Chromatickým číslom* $\chi(G)$ nazveme najmenšie čislo k také, že G je vrcholovo k-zafarbiteľný.
- *Hranovým farbením* nazveme zobrazenie $c : E \to S$, pričom ak sú hrany e, f susedné, musí platiť $c(e) \neq c(f)$.

- *Hranovým k-farbením* nazveme také hranové farbenie, v ktorom $S = \{1, 2, ..., k\}$.
- Graf je *hranovo k-zafarbiteľný*, ak má hranové k-farbenie.
- *Chromatickým indexom* $\chi'(G)$ nazveme najmenšie čislo k také, že G je hranovo k-zafarbiteľný.

Názov *farbenia grafov* sa od jeho prvého použitia Augustom De Morganom v roku 1852 [22] dodnes zachoval, hoci sa miesto farieb zvyknú používať prirodzené čisla.

1.2 Nerepetitívne a zoznamové farbenia

V nasledujúcom upustíme od označovania cesty $P = v_0e_0v_1e_1...e_{k-1}v_k$ a budeme používať len $P = v_0v_1...v_k$, nakoľko budeme pracovať len s vrcholmi grafov. Taktiež miesto pojmu vrcholové farbenie budeme zvyčajne používať iba slovo farbenie.

Definícia 13. Nech $G = (V, E)$ je graf, nech $c : V \to S$ je farbenie tohto grafu. Hovoríme, že farbenie c je *repetitívne*, ak v grafe G existuje cesta $P = v_0v_1...v_{2r-1}$, pre ktorú platí $c(v_i) = c(v_{r+i})$, pre $i = 0, 1, ..., r - 1$. Ak sa v grafe G nenachádza žiadna takáto cesta, farbenie c sa nazýva *nerepetitívne*.

Definícia 14. Najmenšie také číslo k, pre ktoré má graf G nerepetitívne k-farbenie, sa nazýva *Thueho chromatické číslo*, označuje sa $\pi(G)$.

Definícia 15. Nech $G = (V, E)$ je graf, nech $L : V \to 2^N$ je funkcia, ktorá danému vrcholu priraďuje množinu farieb (prirodzených čísel) použiteľných na zafarbenie vrcholu v. Túto množinu $L(v)$ budeme nazývať *zoznam farieb* (prípadne len *zoznam*). *Zoznamové farbenie* je farbenie definované zobrazením $c : V \to \mathbb{N}$, pričom $c(v) \in L(v)$ pre všetky vrcholy v.

Definícia 16. Graf je *zoznamovo k-zafarbiteľný*, ak je zoznamovo zafarbiteľný a funkcia $L : V \to 2^N$ priraďuje všetkým vrcholom k–prvkové množiny. *Zoznamové chromatické číslo* $\chi_l(G)$ je najmenšie číslo k také, že G je zoznamovo k-zafarbiteľný.

Definícia 17. Nech $G = (V, E)$ je graf, nech $L : V \to 2^N$ je funkcia, ktorá danému vrcholu priraďuje množinu farieb (prirodzených čísel) použiteľných na zafarbenie vrcholu v. *Nerepetitívne zoznamové farbenie* je nerepetitívne farbenie definované zobrazením $c : V \to N$, kde $c(v) \in L(v)$ pre všetky vrcholy v.

Definícia 18. Graf je *nerepetitívne zoznamovo k-zafarbiteľný*, ak je nerepetitívne zoznamovo zafarbiteľný a funkcia $L : V \to 2^N$ priraďuje všetkým vrcholom k–prvkové množiny. *Thueho číslo výberu* $\pi_{ch}(G)$ je najmenšie číslo k také, že G je nerepetitívne zoznamovo k-zafarbiteľný.

1.3 Sekvencie

Problém nerepetitívneho zoznamového farbenia ciest sa dá popísať nielen grafovými štruktúrami, ale i pomocou sekvencií slov. V nasledujúcom stručne zadefinujeme základné pojmy, a potom popíšeme, ako pomocou nich charakterizovať uvedený problém.

Abeceda Σ je konečná neprázdna množina symbolov (alebo aj písmen).
Slovo nad abecedou Σ je (konečná) postupnosť symbolov zo Σ.
Dĺžka slova w je dĺžka postupnosti, ktorá ho vytvára. Značíme $|w|$.
Prázdne slovo je slovo dĺžky 0, označuje sa ε.
Nekonečné slovo je nekonečne dlhé slovo, teda je to zobrazenie $w : \mathbb{N} \to \Sigma$.
Obojstranne nekonečné slovo je zobrazenie $w : \mathbb{Z} \to \Sigma$.
N-árne slovo je slovo nad abecedou Σ, pričom $|\Sigma| = n$.
Podslovo slova $w = s_1 s_2 ... s_n$ je ľubovoľná postupnosť $s_i s_{i+1} ... s_j$, kde $1 \leq i \leq j \leq n$.
Prefix je podslovo, pre ktoré platí $i = 1$. *Sufix* je podslovo, pre ktoré platí $j = n$.
Štvorec je neprázdne slovo tvaru uu. Slovo nazveme *bez-štvorcové*, ak je neprázdne a žiadne z jeho podslov nie je štvorcom.
Prekryv (*overlap* [6]) je slovo tvaru $xuxux$, kde x je neprázdne. Slovo je *bez-prekryvové* (*overlap-free*), ak je neprázdne a žiadne z jeho podslov nie je prekryv.
Kocka [17] je neprázdne slovo tvaru uuu. Slovo nazveme *bez-kockové*, ak je neprázdne a žiadne z jeho podslov nie je kockou.
Platí, že bez-štvorcové slovo je aj bez-prekryvové a taktiež platí, že bez-prekryvové slovo je tiež bez-kockové.
Reverz slova $w = s_1 s_2 ... s_n$ je slovo $w^R = s_n s_{n-1} ... s_1$. Ak platí $w = w^R$, slovo w voláme palindróm.
Označme *alpha(w)* množinu symbolov, ktoré sa v slove w vyskytujú aspoň raz.

Teraz uvediem ekvivalentnú definíciu pre repetitívnosť (13), zoznamovú k-zafarbiteľnosť (15) i nerepetitívnu zoznamovú k-zafarbiteľnosť (18) pomocou slov a sekvencií.

Definícia 19. Nech $\Sigma = \{0, 1, 2, ..., k\}, k \in \mathbb{N}$ je abeceda nad farbami $0, 1, 2, ..., k$. Nech slovo $w \in \Sigma, w = a_0, a_1, ..., a_n, n \in \mathbb{N}$. Nech w obsahuje štvorec. Potom je tento štvorec ekvivalentný s repetíciou v definícii 13 a slovo w postupnosti $c(v_0), c(v_2), ..., c(v_{2r})$. Ak je slovo w bez-štvorcové, je definícia ekvivalentná nerepetitívnemu farbeniu z definície 13.

Definícia 20. Nech $\Sigma = \{0, 1, 2, ..., k\}, k \in \mathbb{N}$ je abeceda nad farbami $0, 1, 2, ..., k$. Nech slovo $w \in 2^{\binom{\Sigma}{s}}, s \in \mathbb{N}$, a teda $w = a_0, a_1, ..., a_n$, kde $a_i = d_0, d_1, ..., d_s, i \in \mathbb{N}, i < n, d_j \in \Sigma, j \in \mathbb{N}, j < s$. Symbol a_i je ekvivalentnou definíciou zoznamu farieb v definícii 15. Zoznamové s-farbenie môžeme tiež ekvivalentne definovať ako zobrazenie $c : \binom{\Sigma}{s} \to \Sigma$,

kde $c(i) \in a_i$ pre všetky $i \in \mathbb{N}, i < n$, pričom musí platiť $c(i) \neq c(i+1)$ pre všetky $i \in \mathbb{N}, i < n-1$. Ak je navyše slovo w bez-štvorcové, ide o ekvivalentnú definíciu nerepetitívnej zoznamovej s-zafarbiteľnosti z definície 18.

Dodáme, že slovo w z predošlej definície 20 je ekvivalent cesty, ktorej vrcholom prislúcha s–prvková množina farieb.

1.4 Používané definície

Pre účely jednoduchšieho sa vyjadrovania či čitateľnejšieho textu zavedieme ešte iné pojmy, prípadne rozšírime niektoré definície. Pojmy tu uvedené budeme totižto hojne využívať vo všetkých nadchádzajúcich kapitolách práce. V nasledujúcej sekcii ale i v ďalších kapitolách budeme používať symbol $n, n \in \mathbb{N}, n \neq 0$ ako dĺžku cesty alebo dĺžku farbenia. Symbolom $k, k \in \mathbb{N}, k \geq 3$ budeme zas označovať počet povolených farieb. A napokon, $\Sigma = \{0, 1, ..., k\}, k \in \mathbb{N}$ bude znamenať abecedu nad symbolmi (farbami) $0, 1, ..., k$.

Definícia 21. Nech P je cesta, nech $L : V \to 2^N$ je funkcia, pričom $|L(v)| = 3$ pre všetky vrcholy v. Nech prvky v $L(v)$ sú usporiadané. Potom postupnosť usporiadaných trojíc $P_z = (c_{0,0}, c_{0,1}, c_{0,2}), (c_{1,0}, c_{1,1}, c_{1,2}), ..., (c_{n-1,0}, c_{n-1,1}, c_{n-1,2})$, kde $\{c_{i,0}, c_{i,1}, c_{i,2}\} = L(i)$ pre všetky vrcholy $i \in \mathbb{N}, i < n$, budeme volať *zoznamová cesta*. Čísla $c_{i,j}, i \in \mathbb{N}, i < n, j \in \{0, 1, 2\}$ budeme volať *farbami* zoznamovej cesty P_z a príslušnosť farby zoznamovej ceste budeme značiť $c_{i,j} \in P_z$.

Formálny zápis zoznamovej cesty 21 sa odlišuje od skoršej definície cesty 7. Používanie zoznamovej cesty bude pre nás však pohodlnejšie a v texte čitateľnejšie. Definície sa líšia hlavne preto, že je pre naše účely žiaduce zapracovať do objektu cesty pre každý vrchol i zoznam troch farieb, s ktorými budeme v mnohom pracovať. Taktiež sa zdá byť vhodné odstrániť značenia pre vrcholy, keďže tie nebudeme nikde potrebovať.

Takýto zápis zoznamovej cesty 21 sa dokonca odlišuje aj od ekvivalentu cesty v podobe slova w z definície 20. Ak by sme slovo w rozpísali tak, aby sa čo najviac pripodobnilo definícii zoznamovej cesty 21, dostali by sme $w = \{c_{0,0}, c_{0,1}, c_{0,2}\}, \{c_{1,0}, c_{1,1}, c_{1,2}\}, ..., \{c_{n-1,0}, c_{n-1,1}, c_{n-1,2}\}$, teda by išlo o postupnosť 3–prvkových množín. Prvky $c_{i,0}, c_{i,1}, c_{i,2}$ sú navzájom rôzne pre všetky $i \in \mathbb{N}, i < n$ v oboch definíciách, keďže v oboch patria do nejakej spoločnej množiny. Pri zoznamovej ceste ale požadujeme, aby boli trojice navyše usporiadané. Pri slove w sa dá bez ujmy na všeobecnosti tiež vyžadovať, aby platilo $c_{i,0} < c_{i,1} < c_{i,2}$ pre všetky $i \in \mathbb{N}, i < n$, keďže ide o tri rôzne čísla. My však budeme v nasledujúcom pre jednoduchosť pracovať so zoznamovou cestou definovanou ako postupnosť usporiadaných trojíc z definície 21.

Definícia 22. Nech P je zoznamová cesta. Trojicu $(c_{i,0}, c_{i,1}, c_{i,2}) \in P$ pre nejaké $i \in \mathbb{N}, i < n$ budeme volať *trojzoznam* pre vrchol i. Niekedy takúto trojicu nazveme aj *vrchol* zoznamovej cesty.

Definícia 23. Nech P je zoznamová cesta. *Farbením* zoznamovej cesty P nazveme postupnosť farieb $c_0, c_1, ..., c_{n-1}$, kde $c_i, i \in \mathbb{N}, i < n$ je farba patriaca do trojzoznamu zoznamovej cesty P pre vrchol i.

Definícia farbenia by sa dala jednoducho pozmeniť tak, aby bolo farbenie zobrazením, ako tomu bolo v skoršej definícii farbenia 12. Pre prehľadnosť sme však radšej zaviedli definíciu s postupnosťou.

Definícia 24. Nech $C = c_0, c_1, ..., c_{p-1}$ a $D = d_0, d_1, ..., d_{q-1}$, kde $p, q \in \mathbb{N}$ sú farbenia. Nech . je binárna operácia na farbeniach, pričom $C.D = c_0, c_1, ..., c_{p-1}, d_0, d_1, ..., d_{q-1}$. Operáciu . budeme volať *zreťazenie*.

Operáciu zreťazenia budeme používať i na pripojenie jednej farby a k farbeniu C. Zápis $C.a$, prípadne $a.C$, bude skrátený zápis pre $C.A$, prípadne $A.C$, kde $A = a$ je jednoprvková postupnosť farieb.

Definícia 25. Nech A je postupnosť. *Dĺžkou* postupnosti A nazveme počet prvkov postupnosti a zapíšeme ako $|A|$.

Dĺžkou farbenia bude teda počet jeho farieb a dĺžkou zoznamovej cesty bude počet jej trojzoznamov.

Definícia 26. Nech n je párne a nech $C = c_0, c_1, ..., c_{n-1}$ je postupnosť farieb. Nech platí $c_i = c_{i+(n/2)}, i \in \mathbb{N}, i < n/2$. Potom budeme postupnosť C nazývať *repetíciou*.

Definícia 27. Nech P je zoznamová cesta a C nejaké farbenie. Budeme hovoriť, že farbenie C je *repetitívne*, ak obsahuje podpostupnosť, ktorá je repetíciou a že farbenie je *nerepetitívne*, ak takú podpostupnosť neobsahuje. Zoznamovú cestu P budeme nazývať *repetitívnou*, ak sú všetky jej farbenia repetitívne a *nerepetitívnou*, ak je aspoň jedno jej farbenie nerepetitívne.

Definícia 27 je vlastne rozšírením skoršej definície nerepetitívneho zoznamového 3–farbenia (18) na zoznamové cesty.

Definícia 28. Nech P_1 je zoznamová cesta, nech a je farba na zoznamovej ceste P. Budeme hovoriť, že zoznamová cesta P_2 vznikla *substitúciou* zo zoznamovej cesty P_1 nahradením farby a za farbu $b, b \in \mathbb{N}$ a zapisovať $P_2 = P_1\{a \mapsto b\}$, ak platí $\forall i \in \mathbb{N}, i < n, \forall j \in \{0, 1, 2\}$, že ak platí $c_{i,j} \neq a$, tak $d_{i,j} = c_{i,j}$ a zároveň ak platí $c_{i,j} = a$, tak $d_{i,j} = b$ (pričom $d_{i,j} \in P_2$).

Definícia 29. Nech P_0 je zoznamová cesta, nech farby $a, b \in P_0$ a nech farba $p \notin P_0$. Budeme hovoriť, že zoznamová cesta P_1 vznikla zo zoznamovej cesty P_0 *obojstrannou substitúciou* a zapisovať $P_1 = P_0\{a \leftarrow\!\!\rightarrow b\}$, ak platí $P_1 = P_0\{a \mapsto p\}\{b \mapsto a\}\{p \mapsto a\}$.

Dôsledok 1. Definícia 29 nám neprikazuje, aby bola farba p medzi povolenými farbami na zoznamovej ceste P_0 (teda $0 \leq p < k$). Za zmienku tiež stojí, že ani P_0, ani P_1 neobsahuje farbu p. Aj keď teda používa zoznamová cesta P_0 všetky povolené farby, obojstranná substitúcia je stále možná (položíme napríklad $p = k$).

Definícia 30. Nech P je zoznamová cesta. Farby $a, b \in P$ budeme volať *disjunktné*, ak sa na zoznamovej ceste P nachádzajú v rôznych trojzoznamoch.

Definícia 31. *Lexikografickým usporiadaním* zoznamových ciest definujeme jednoducho ako lexikografické usporiadanie postupností, nakoľko sme zoznamovú cestu definovali ako postupnosť usporiadaných trojíc 21.

Definícia 32. Nech P_0 je zoznamová cesta. Vytvorme množinu zoznamových ciest $\mathbb{P} = \{P_i\}, i \in \mathbb{N}$, kde P_i je zoznamová cesta, ktorá je výsledkom niekoľkonásobnej aplikácie substitúcie na P_0, pričom zmena farby musí prebehnúť z ľubovoľnej farby na zoznamovej ceste, s ktorou vykonávame substitúciu, na farbu nepatriacu tejto zoznamovej ceste. Lexikograficky najmenšiu inštanciu spomedzi všetkých zoznamových ciest v množine $\mathbb{P}$ budeme nazývať *lexikograficky prvá* alebo aj *lexikograficky najlepšia* zoznamová cesta.

Dôsledok 2. Definícia 32 nám hovorí, že porovnávať zoznamové cesty za účelom zistenia lexikograficky najlepšej inštancie je možné len medzi zoznamovými cestami rovnakej dĺžky a medzi zoznamovými cestami zachovávajúcimi pozície farieb na daných vrcholoch, nie však nutne ich hodnoty. Lexikograficky najlepšia možnosť je preto pre danú zoznamovú cestu vždy len jedna a bude vyhodnocovať repetitívnosť rovnako ako ostatné inštancie zoznamových ciest, s ktorými sme ju porovnávali na základe lexikografického usporiadania. Dôkazom tejto úvahy je jednoducho fakt, že na každú substitúciu v definícii 32 môžeme aplikovať vetu 3.4.

Definícia 33. Nech P je zoznamová cesta. Nech $l, m, l \in \mathbb{N}, l < n, m \in \{0, 1, 2\}$ určujú takú pozíciu farby $a = c_{l,m}, c_{l,m} \in P$, že vrcholy $< l$ neobsahujú farbu a, teda $\forall j \in \mathbb{N}, j < l, \forall o \in \{0, 1, 2\}$ platí, že $c_{j,o} \neq c_{l,m}$. Pozíciu farby danú číslami l, m budeme nazývať *najľavejšia pozícia*, *prvá pozícia*, *najľavejší výskyt* alebo *prvý výskyt* farby a.

Kapitola 2

Doterajšie výsledky

V nasledujúcom budeme pojednávať o dosiahnutých výsledkoch a nastolených hypotézach pojednávajúcich o nerepetitívnych zoznamových farbeniach ciest. Pozrieme sa i na iné výsledky, ktoré však stále dostatočne korelujú s našou témou.

2.1 Thueho — Morsova sekvencia

Definujme *Thueho — Morsovu sekvenciu* [1] ako nekonečnú postupnosť čísel $a_0, a_1, ...$, kde $a_i \in \{0, 1\}$, pre ktorú platí $a_0 = 0, a_{2n} = a_n, a_{2n+1} = 1 - a_n$. Pre názornú ukážku, prvých osem členov Thue — Morsovej sekvencie je tvaru 01101001.

Ekvivalentne môžeme túto sekvencie skonštruovať nasledovne: Nech je prvý člen sekvencie bit 0. Ak máme skonštruovanú sekvenciu x, tak znegujeme každý bit sekvencie, označme znegovanú sekvenciu y, a následne vytvoríme dlhšiu sekvenciu xy.

Označme opäť Thueho — Morsovu sekvenciu ako postupnosť čísel $a_0, a_1, ...$, kde $a_i \in \{0, 1\}$. Vytvorme z nej inú postupnosť rozdielov jej po sebe idúcich členov, teda $b_0, b_1, ...$, kde $b_i = a_{i+1} - a_i$, čiže platí $b_i \in \{-1, 0, 1\}$. Takáto postupnosť neobsahuje štvorec [14] [13]. Ak by sme čísla tejto postupnosti chápali ako tri rôzne farby, zistili by sme (z ekvivalentnosti týchto definícií), že môže existovať nekonečne dlhá nerepetitívna cesta zafarbená tromi farbami. Práve tento objav je pripisovaný nórskemu matematikovi menom Axel Thue a formálnejšie ho vyslovujeme v nadchádzajúcej sekcii.

2.2 Nerepetitívne farbenia

Veta 2.1 (Thue 1906, [21], preklad [5])**.** *Nech P_n je cesta dĺžky $n \in \mathbb{N}, n \geq 4$. Potom $\pi(P_n) = 3$.*

Platnosť tejto vety dokázal Thue v roku 1906 [21] [5, preklad] konštrukčne a veľmi elegantne, ako sme už popísali v predchádzajúcej sekcií. Zistil teda, že určité rekurzívne definované binárne slová, pričom Thueho — Morsova sekvencia je jednou z nich,

neobsahujú prekryvy. Vďaka tejto znalosti sa mu podarilo zostrojiť nekonečný súbor ternárnych slov bez štvorcov. Spomínané binárne slová sa už objavili i v skorších prácach [15] [19], ba dokonca bol Thueov výsledok i viackrát znovuobjavený v rozličných kontextoch a získal mnoho generalizácií a dôležitých aplikácií v rôznych matematických i informatických odvetviach.

Z vety 2.1 ihneď vyplýva, že $\pi(C_n) \leq 4$, pre každý cyklus C_n. Stačí zobrať nerepetitívne 3-farbenie na ceste z ľubovoľných $n-1$ vrcholov, ktorého existenciu nám veta 2.1 zaručuje, a poslednému vrcholu priradiť štvrtú farbu. Dnes dokonca už vieme dokázať, že $\pi(C_n) = 3$ pre $n \geq 18$ [7].

Ak nebudeme uvažovať cesty, ale pozrieme sa na grafy s najvyšším stupňom vrcholu väčším ako dva ($\Delta(G) > 2$), Thueho metódy už fungovať nebudú. Dlho sa nevedelo či je pre takéto grafy vôbec hodnota $\pi(G)$ zhora ohraničená. Dôkaz ohraničenia $\pi(G)$ pre nejakú konštantu c prišiel v roku 2002 [2], no v roku 2006 bol spresnený.

Veta 2.2 (2006, [9]). *Nech G je graf. Potom $\pi(G) \leq 16\Delta(G)^2$.*

Dôkaz vety 2.2 spočíval v ukázaní logaritmicky blízkeho dolného odhadu k číslu $16\Delta(G)^2$ pomocou existencie grafov s Thueho chromatickým číslom rádovo $\Delta(G)^2/log\Delta(G)$. Na dôkaz hornej hranice bola použitá Lokálna lemma [16] [23], a na dôkaz dolnej náhodné grafy [16] [3].

V [9] i v [23] je taktiež dokázaná platnosť $\pi(T) \leq 4$, pre každý strom T. V skúmaní Thueho čísla na stromoch pokračovali napríklad v [4], [12] a ukázali, že $\pi(T) \leq 4^k$, pre stromy T s maximálne k synmi.

Ohraničenosť Thueho čísla pre triedu planárnych grafov ostáva otvoreným problémom. Vedci v práci [20] sa téme planárnych grafov venovali a s použitím spomínanej vety 2.2 pre špecifický typ grafov ohraničenie Thueho chromatického čísla ukázali. My sa tejto téme však nebudeme venovať.

2.3 Nerepetitívne zoznamové farbenia

Ako uvádza [11], výsledok vety 2.2 zostane validný i pri pridaní zoznamov do nerepetitívnych farbení, a teda platí $\pi_{ch}(G) \leq 16\Delta(G)^2$, pre každý graf G. Keďže zameriavame našu pozornosť na cesty, položíme $\pi_{ch}(P_n) \leq 64$, keďže $\Delta(P_n) \leq 2$. Tento odhad bol najprv spresnený na $\pi_{ch}(P_n) \leq 15$, no neskôr bol ešte viac stlačený.

Veta 2.3 (Grytczuk, 2011 [11]). *Nech P_n je cesta dĺžky $n \in \mathbb{N}, n \geq 4$.*
Potom $\pi_{ch}(P_n) \leq 4$.

Teraz uvažujme o dolnom odhade pre $\pi_{ch}(P_n)$. Musí platiť $\pi_{ch}(P_n) \geq \pi(P_n), n \geq 4$, pretože pridaním zoznamov pre vrcholy grafu sa výber farby môže iba sťažiť. Vďaka

vete 2.1 vieme, že platí $\pi(P_n) = 3, n \geq 4$. Z týchto dvoch pozorovaní vyplýva významný dolný odhad pre náš problém.

Veta 2.4. *Nech P_n je cesta dĺžky $n \in \mathbb{N}, n \geq 4$. Potom $\pi_{ch}(P_n) \geq 3$.*

V [9] sú popísané i rôzne ďalšie zistenia na túto tému. V tejto ani v inej práci sa však ešte nepodarilo spojiť vetu 2.3 a vetu 2.4, a dostať tak uzavretý pohľad na hodnotu Thueho čísla výberu pre cesty $\pi_{ch}(P_n)$. Táto otázka tak ostáva otvoreným problémom, pričom sa ale predpokladá, že je hodnota $\pi_{ch}(P_n)$ rovná trom. Túto nepotvrdenú hypotézu sme sa rozhodli v našej práci aspoň sčasti objasniť.

Hypotéza 1. *Nech P_n je cesta dĺžky $n \in \mathbb{N}, n \geq 4$. Potom $\pi_{ch}(P_n) = 3$.*

Kapitola 3

Generovanie zoznamov a farbení

Cieľom našej práce je implementácia programu, ktorý čiastočne vyrieši spomínanú hypotézu, teda že $\pi_{ch}(P_n) = 3$, pre každú cestu $P_n, n \geq 4$. Keďže bude časová zložitosť takéhoto program extrémne vysoká, budeme nútení vymýšľať rôzne optimalizácie, ktorých správnosť je potrebné formálnejšie dokázať. A práve dôkazy tohto typu budú nosnou časťou nasledujúcej kapitoly.

Na úvod ešte vyjasníme zopár pojmov, ktoré sa budú vyskytovať ako v tejto kapitole, tak i v nadchádzajúcich. V celej nasledujúcej kapitole budeme používať symbol $n, n \in \mathbb{N}, n \neq 0$ ako dĺžku cesty (respektíve zoznamovej cesty) alebo dĺžku farbenia. Symbolom $k, k \in \mathbb{N}, k \geq 3$ budeme zas označovať počet povolených farieb. A napokon, $\Sigma = \{0, 1, ..., k\}, k \in \mathbb{N}$ bude značiť abecedu nad symbolmi (farbami) $0, 1, ..., k$.

Veta 3.1. *Nech C_a je repetitívne farbenie a nech C_b je farbenie. Potom je aj farbenie $C_a.C_b$ repetitívne.*

Dôkaz. Označme repetíciu vo farení C_a ako postupnosť $c_r c_{r+1}...c_{r+s}, r, s \in \mathbb{N}, r < n, r + s < n, s > 0, s$ je párne. Označme $|C_a.C_b| = m$. Potom platí $n < m$. Keďže ale $r < n$ a $r + s < n$, musí platiť aj $r < m$ a $r + s < m$, čiže repetícia $c_r c_{r+1}...c_{r+s}$ sa nachádza i vo farbení $C_a.C_b$. □

Veta 3.2. *Nech C je nerepetitívne farbenie. Nech a je farba. Farbenie $C.a$ neobsahuje repetíciu používajúcu farbu a na poslednej pozícii práve vtedy, keď je farbenie $C.a$ nerepetitívne.*

Dôkaz. Implikácia zprava doľava platí triviálne. Implikácia zľava doprava:

Dokážeme obmenené tvrdenie, teda ak je farbenie $C.a$ repetitívne, tak farbenie $C.a$ obsahuje repetíciu používajúcu a. Označme repetíciu v $C.a$ ako postupnosť R. Sporom — nech repetícia R nepoužíva poslednú farbu a. Potom sa ale táto repetícia musí nachádzať vo farbení C, čo je spor s predpokladom, že C je nerepetitívne farbenie. □

Lemma 1. *Nech P_0 je zoznamová cesta a nech $a \in P_0$ je nejaká jej farba. Nech $P_1 = P_0\{a \mapsto b\}$, kde $b \in \mathbb{N}, b < k$. Nech sú farby a, b v P_0 disjunktné. Potom ak v P_0 neexistuje nerepetitívne farbenie, tak neexistuje ani v P_1.*

Dôkaz. Dokážeme obmenené tvrdenie, teda že ak v P_1 existuje nerepetitívne farbenie, tak existuje nerepetitívne farbenie aj v P_0.

Označme nerepetitívne farbenie v P_1 ako $D = d_{0,j_0}, d_{1,j_1}, ..., d_{n-1,j_{n-1}}$, kde $d_{i,j_i} \in P_1, j_i \in \{0, 1, 2\}, i \in \mathbb{N}, i < n$. Skonštruujme farbenie $C = c_{0,j_0}, c_{1,j_1}, ..., c_{n-1,j_{n-1}}$, kde $c_{i,j_i} \in P_0, i \in \mathbb{N}, i < n$. Ukážeme, že farbenie C je nerepetitívne.

Sporom. nech C je repetitívne. Z definície 27 vieme, že v C existuje podpostupnosť farieb s repetíciou. Označme túto repetíciu $c_r, c_{r+1}, ..., c_{r+s}, r, s \in \mathbb{N}, r < n, r + s < n, s \neq 0, s$ je párne a platí $c_{r+i} = c_{r+i+(s/2)}, i \in \mathbb{N}, i < (s/2)$. Prezrieme dva prípady:

- Ak $c_{r+i} \neq a$ (nech $c_{r+i} = p, p \neq a$), potom z definície dostávame, že aj $d_{r+i} = p$. Z rovnosti $c_{r+i} = c_{r+i+(s/2)}$ vieme, že aj $c_{r+i+(s/2)} = p$. Z definície opäť dostávame $d_{r+i+(s/2)} = p$, a teda platí aj $d_{r+i} = d_{r+i+(s/2)}$.

- Ak $c_{r+i} = a$, potom z definície $d_{r+i} = b$. Z rovnosti $c_{r+i} = c_{r+i+(s/2)}$ vieme, že aj $c_{r+i+(s/2)} = a$. Z definície opäť dostávame $d_{r+i+(s/2)} = b$, a teda platí aj $d_{r+i} = d_{r+i+(s/2)}$.

Ukázali sme, že $d_{r+i} = d_{r+i+(s/2)}, i \in \mathbb{N}, i < (s/2)$, takže farbenie D obsahuje repetíciu $d_r, d_{r+1}, ..., d_{r+s}$, čo je spor s predpokladom, že D je nerepetitívne farbenie. Postupnosť C preto musí byť hľadaným nerepetitívnym farbením.

□

Dôsledok 3. Lemma 1 platí aj ak $b \notin P_0$, pretože podmienka, aby boli farby a, b na zoznamovej ceste P_0 v rôznych trojzoznamoch, je stále naplnená.

Dôsledok 4. Ak $a = b$ z lemmy 1, tak nebudú farby a, b na zoznamovej ceste P_0 v rôznych trojzoznamoch. Je však zrejmé, že bude platiť $P_0 = P_1$, a teda tvrdenie lemmy bude platiť triviálne.

Lemma 1 priveľmi neobmedzuje farbu b — môže sa vyskytovať v P_0, ale nemusí, dokonca môže platiť $b = a$. Jediné obmedzenie na farbu b je, že nemôže nastať situácia, aby na jednom vrchole zoznamovej cesty P_0 boli v trojzozname obe farby a, b, keďže v takom prípade by sme sa po výmene farby a za farbu b dostali do nekonzistentného stavu.

Dôsledok 5. Disjunktnosť farieb a, b v lemme 1 nám zabezpečí i platnosť rovnice $P_0 = P_1\{b \mapsto a\}$, takže opačná implikácia bude taktiež platiť. Tvrdenia "*v P_0 neexistuje nerepetitívne farbenie*" a "*v P_1 neexistuje nerepetitívne farbenie*" (prípadne ich obmenená forma) sú preto v kontexte lemmy 1 ekvivalentné.

Veta 3.3. *Nech P_0 je zoznamová cesta. Nech farby $a, b \in P_0$ sú na zoznamovej ceste P_0 v rôznych trojzoznamoch. Nech $p \in \mathbb{N}$ je farba. Označme $P_1 = P_0\{a \mapsto p\}\{b \mapsto p\}$. Potom platí, že ak v P_0 neexistuje nerepetitívne farbenie, tak neexistuje ani v P_1.*

Dôkaz. Celý dôkaz v podstate pozostáva z dvojnásobného použitia lemmy 1. Vytvoríme zoznamovú cestu $P_{0.5} = P_0\{a \mapsto p\}$. Z lemmy 1, prípadne nejakého jej dôsledku (4, 3) vyplýva tvrdenie "ak v P_0 neexistuje nerepetitívne farbenie, tak neexistuje ani v $P_{0.5}$". Zostrojme teraz zoznamovú cestu $P_{0.5}\{b \mapsto p\} = P_0\{a \mapsto p\}\{b \mapsto p\} = P_1$. Z lemmy 1, prípadne nejakého jej dôsledku (4, 3) vyplýva tvrdenie "ak v $P_{0.5}$ neexistuje nerepetitívne farbenie, tak neexistuje ani v P_1". Vďaka platnosti oboch tvrdení dostávame z tranzitívnosti tvrdenie, ktoré bolo treba dokázať.

□

Veta 3.4. *Nech P_0 a P_1 sú zoznamové cesty. Nech a je farba, pričom $a \in P_0 \,\&\, a \notin P_1$. Nech b je farba, pričom $b \in P_1 \,\&\, b \notin P_0$. Nech platí $P_1 = P_0\{a \mapsto b\}$. Potom ak v P_0 neexistuje nerepetitívne farbenie, tak neexistuje ani v P_1.*

Dôkaz. Dôkaz vety tkvie v triviálnej aplikácii lemmy 1, respektíve jej dôsledku 3. □

Dôsledok 6. Veta 3.4 nám dokazuje platnosť aj obmeneného tvrdenia "*ak v P_1 existuje nerepetitívne farbenie, tak existuje aj v P_0*". A navyše, ako sme spomínali i v dôsledku 3, vďaka disjunktnosti farieb a, b, platí aj tvrdenie s "vymenenými" zoznamovými cestami, teda tvrdenie "*ak v P_1 neexistuje nerepetitívne farbenie, tak neexistuje ani v P_0*" aj s jeho obmenou.

Veta 3.5. *Nech P_0 a P_1 sú zoznamové cesty. Nech farby $a, b \in P_0 \,\&\, a, b \in P_1$. Nech $P_1 = P_0\{a \longleftrightarrow b\}$ Potom ak v P_0 neexistuje nerepetitívne farbenie, tak neexistuje ani v P_1.*

Dôkaz. Rozpíšme obojstrannú substitúciu podľa definície 29 na $P_1 = P_0\{a \mapsto p\}\{b \mapsto a\}\{p \mapsto a\}$, kde $p \in \mathbb{N}, p \notin P_0, p \notin P_1$ je farba. Dôsledok 3 môže byť použitý na všetky tri substitúcie, a tým je veta dokázaná. □

Kapitola 4

Implementácia

Na začiatku kapitoly uvedieme, ako bude vyzerať program, ktorým sa budeme snažiť dokázať či vyvrátiť spomínanú hypotézu 1. Následne budeme v chronologickom poradí uvádzať optimalizácie a vysvetľovať, prečo sme na ne pristúpili. Časovú zložitosť tohto náročného problému výrazne neovplyvníme, no i tak sa ju budeme snažiť zaznačiť a tak zistiť, ako veľmi sa posúvame s každou ďalšou optimalizáciou. Dôkazy, že dané optimalizácie sú korektné, teda že vďaka nim neprehliadneme žiadny protipríklad, alebo naopak, nijaký nesprávny nevygenerujeme, sa nachádzajú v kapitole 3 a v tejto kapitole na ne budeme na vhodných miestach odkazovať. Ešte uvedieme, že pod pojmom *cesta* budeme zakaždým myslieť *zoznamovú cestu* z definície 21.

4.1 Požiadavky a návrh programu

Najskôr uvedieme, ako by mal nami implementovaný program vyzerať a ako bude pracovať.

Rozhodli sme sa pre implementáciu v jazyku C++, nakoľko s ním už máme skúsenosti, a vieme tak zabezpečiť, aby bol program optimálne napísaný.

Implementovaný algoritmus bude musieť vedieť, na ako dlhej zoznamovej ceste a s koľkými farbami má pracovať. Tieto dva parametre však programu neodovzdáme ako vstup, no radšej vytvoríme súbor, ktorý bude tieto hodnoty obsahovať. Zaznamenané čísla búdu konštantami, ešte presnejšie takzvaný *constexpr* v jazyku C++. Pre jednoduchosť budeme aj naďalej spomínané dve hodnoty volať vstupnými hodnotami programu. Takáto interpretácia vstupných údajov nám zabezpečí možnosť používať rôzne triedy, ktoré vyžadujú poznať nejaký ich parameter už v čase kompilácie, napr. trieda *<array>*. Na druhej strane budeme nútení pri testovaní rýchlosti programu pre rôzne hodnoty vstupu zakaždým kód programu kompilovať.

Častá kompilácia nám však prácu nezneprijemní, pretože každá logicky netriviálna funkcia kódu bude dostatočne otestovaná skriptami na to určenými, takže obmieňanie

vstupných parametrov a následná kompilácia sa budú diať len sporadicky a vždy len raz pre dané parametre. Dôkladné testovanie je pre povahu našej práce priam nevyhnutné, nakoľko sa chystáme počítačovo dokázať výpočtovo náročnú hypotézu, na čo nie je možné použiť čo i len trochu chybný algoritmus. Pri hľadaní chýb nám pomôžu i rôzne treťostranové softvéry, najmä pôjde o nachádzanie takzvaných memory leak–ov. Využijeme i rôzne prepínače, ako napríklad "–D_GLIBCXX_DEBUG".

Našou úlohou, ak ju teda detailnejšie popíšeme, je vytvoriť program, ktorý najprv vytvorí všetky možné konštrukcie zoznamových ciest o zadanej dĺžke, z ktorej každý vrchol obsahuje trojzoznam farieb, kde farby sú rôzne a vybrané spomedzi zadaných farieb. Na každej takejto zoznamovej ceste algoritmus následne vygeneruje všetky možné farbenia vyberajúc farbu pre každý vrchol z trojzoznamu určeného pre ten vrchol. Ak sa medzi farbeniami pre nejakú zoznamovú cestu nájde farbenie, ktoré spĺňa podmienky nerepetitívnosti, môžeme túto zoznamovú cestu označiť za prezretú, teda takú, ktorá nespôsobí protipríklad v nami overovanej hypotéze. Za zmienku tiež stojí, že ak zistíme nerepetitívnosť pre nejaké farbenie, nemusíme už ďalšie farbenia pre zoznamovú cestu kontrolovať a môžeme prejsť na nasledujúcu. Bude preto žiadané farbenia nevytvárať všetky naraz, a potom ich kontrolovať, ale naprogramovať generátor, ktorý bude farbenia vytvárať postupne a každé jedno budeme postupne kontrolovať na nerepetitívnosť. Ďalším argumentom pre programovanie generátora je počet farbení, ktorý s narastajúcou dĺžkou cesty rastie exponenciálne. Tieto úvahy platia taktiež pri vytváraní samotných ciest a ich trojzoznamov, preto pri ich implementácii použijeme podobný generátor. Ostáva vyriešiť otázku, ako bude vyzerať kontrola nerepetitívnosti. Avšak časová zložitosť priamočiarej implementácie podmienky nerepetitívnosti (13) pre celú cestu je $O(n^3)$, takže jej použitie nebude priveľmi zaťažovať náš exponenciálne dlhý výpočet.

Okrem dvoch generátorov a kontroly nerepetitívnosti bude program pozostávať i z ďalšej nemenej hlavnej časti, funkcie *main*. Jej priebeh bude nasledovný. V prvom rade vytvorí generátor ciest zo vstupných parametrov. Vypýta si od neho cestu a vďaka nej vytvorí generátor farbení. Z generátora farbení si postupne pýta farbenia a overuje na nich nerepetitívnosť. Ak bude nejaké farbenie nerepetitívne, vypýta si od generátora ciest novú cestu a na nej postup opakuje. Ak však funkcia nenájde nijaké nerepetitívne farbenie pre nejakú cestu, tak je daná cesta protipríkladom a hypotéza 1 neplatí.

Za zmienku stojí i adresárová štruktúra. Hlavný priečinok obsahuje všetky skripty, ktoré je možné skompilovať a spustiť, teda súbory obsahujúce funkciu *main*. Pôjde teda o skripty spúšťajúce hlavný program, ako i skript na otestovanie netriviálnych častí programu. V tomto hlavnom adresári sa nachádzajú dva podadresáre, a to zložka *tests* a *prog*. Priečinok *tests* obsahuje všetky potrebné skripty pre testovanie programu a zložka *prog* zas všetky skripty nevyhnutné pre beh hlavného kódu. Priečinok *prog* bude v neskorších fázach práce obsahovať i ďalšie podadresáre, ktoré oddelia rôzne generátory

od ostatných častí programu. Takýto návrh nám umožní vytvoriť jednoduchý príkaz na kompiláciu všetkých súborov s funkciou *main*, nakoľko len pri kompilácií prilinkujeme obe zložky, priečinok *tests* i *prog*, a ostatné súbory v hlavnom adresári tak z kompilácie vynecháme.

Výsledný algoritmus bude extrémne pomalý, nakoľko ide o Π_2^P–úplný problém. Dúfame však, že sa nám vďaka zapracovaniu optimalizácií podarí dostať na dvojmiestne hodnoty vstupných parametrov. Našim cieľom je teda overiť platnosť hypotézy pre všetky dĺžky ciest menšie alebo rovné 10 a zároveň pre každý počet farieb tiež menší alebo rovný 10.

4.2 Základná verzia programu

Ako prvú sme implementovali základnú verziu popísaného návrhu. Neobsahuje zatiaľ žiadne optimalizácie, dbali sme na vhodné rozdelenie kódu do tried, implementáciu pomocných funkcií. V neposlednom rade sme sa venovali vytvoreniu pomocného programu a viacerých testov, ktoré tento program spustí.

Pri implementácií prvotného programu sme sa snažili dodržať body zo spomínaného návrhu 4.1. Pri programovaní generátora farbení sme si dopomohli kódovaním poslednej vygenerovanej kombinácie číslom v trojkovej sústave. Každá číslica zakódovanej hodnoty znamená index do trojzoznamu cesty, s ktorou práve pracujeme. Počet číslic takéhoto čísla korešponduje s dĺžkou cesty — ak by bolo kódované číslo menšie ako dĺžka cesty, na jeho začiatok jednoducho pridáme potrebný počet núl. Jednoduchý príklad môže byť číslo 021. Znamená, že nech je cesta akákoľvek, momentálne sme pre farbenie vzali nultý prvok z jej nultého trojzoznamu, druhý prvok z prvého a prvý z druhého trojzoznamu. Napríklad na ceste $(1, 2, 5), (0, 2, 4), (1, 3, 4)$ by tak išlo o farbenie $1, 4, 3$. Takáto implementácia nie je najefektívnejšia, no nateraz nám postačí. Generátor ciest funguje na rovnakom princípe, ibaže on nemôže kódovať čísla v trojkovej sústave, lež v sústave so základom $\binom{k}{3}$, kde k je povolený počet farieb. Ide o počet všetkých možností trojzoznamov, ktorých tri farby sú rôzne a nehľadíme na ich poradie.

```
class SimplePathGenerator
{
    private:
        int length;
        long long nextLists; //next indeces of color lists (allColorSubsets) to be used in nextPath() in decimal
        Path allColorSubsets; //all possible lists of colors (triplets) that will be used on verteces
                              //-------not really a Path -> just to store it nicely-------

    public:
        SimplePathGenerator(int lengthOfPath, int colorsInPath);
        Path nextPath();
};
```

Obr. 1: Hlavička základného generátora ciest

Samotnú funkciu *main* už hlbšie rozoberať nebudene, keďže je dostatočne popísaná v predchádzajúcej sekcií 4.1. Pripájame však obrázok ukazujúci reálny kód tejto funkcie.

```
int main()
{
    bool found;
    SimplePathGenerator pathGenerator = SimplePathGenerator(lengthOfPath, colorsInPath);
    Path nowPath = pathGenerator.nextPath();

    while (!(nowPath.empty()))
    {
        found = false;

        WholeColoringGenerator coloringGenerator = WholeColoringGenerator(nowPath);
        Coloring coloring = coloringGenerator.nextColoring();

        while (!(coloring.empty()))
        {
            if (checkNonRepetitiveness(coloring))
            {
                //this coloring is non-repetitive, so skipping whole path
                found = true;
                break;
            }
            coloring = coloringGenerator.nextColoring();
        }
        if (!found)
        {
            cout << "COUNTEREXAMPLE!" << endl;
        }
        nowPath = pathGenerator.nextPath();
    }
}
```

Obr. 2: Základná verzia funkcie *main*

Implementovaný program bez optimalizácií beží v časovej zložitosti $O(\binom{k}{3}^n 3^n n^3)$, kde n a k sú vstupné parametre — v tomto poradí ide o dĺžku cesty a počet farieb. Od takejto vysokej časovej zložitosti nemáme veľké očakávania, avšak pre zaujímavosť i pre referenčné hodnoty sme program spustili s rôznymi vstupnými hodnotami a dostali výsledky, ktoré sme zapísali do nasledovnej tabuľky 1. Ešte dodáme, že všetky nefinálne výsledky bežali na jednom jadre procesora, 3 vláknach, a to bez prerušovania o priemernej frekvencii $0,987\,GHz$, pričom po prekročení časového limitu dvoch minút sme program zastavili.

Tabuľka 1: Výsledky základného programu. Symbol X značí, že program s danými parametrami úspešné zbehol v čase pod 2 minúty.

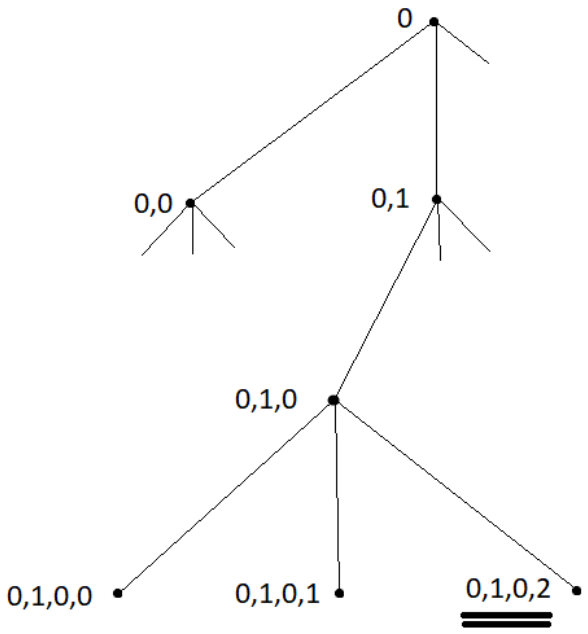
		Dĺžka cesty					
		4	5	6	7	8	9
Počet farieb	4	X	X	X	X	X	
	5	X	X	X			
	6	X	X				
	7	X					
	8	X					
	9						

4.3 Optimalizácia generátora farbení

Základná verzia programu sme dôsledne otestovali. Pri jej dôkladnejšej analýze sme však spozorovali, že vo väčšine vygenerovaných farbení sa nachádza repetícia ihneď na začiatku. Túto skutočnosť sme zachytili na príklade: Vytvárajme farbenia z cesty $(0, 1, 2), (0, 2, 4), ...$ Terajší algoritmus zoberie najskôr všetky farby na pozícii nula z každého zoznamu — vyrobí farbenie $0, 0, ...$. Následne zmení farbenie tak, že z posledného (to jest najpravejšieho) vrcholu vezme farbu na pozícii jedna, ostatné zostanú rovnaké. Ďalšie farbenie bude zasa takmer totožné s predošlým, len z posledného vrcholu sa vezme druhá farba. Ďalšie farbenie vezme z predposledného prvú farbu a z posledného zas nultú, a takto pokračuje, až pokým nevygeneruje všetky. Ľahko nahliadnuť, že pri dlhšej ceste bude veľmi veľa farbení začínajúcich farbami $0, 0$ a keďže už tieto dve farby tvoria repetíciu, každé takéto farbenie bude repetitívne. Ako vidieť vo funkcii *main* 2, repetitívne farbenia nezapríčinia žiaden pokrok a môžeme ich preskakovať.

Rozhodli sme sa implementovať nový generátor farbení, vďaka ktorému nebudeme musieť prezerať farbenia, čo majú krátke repetície na začiatku, a tak zoptimalizujeme terajší kód. Tento generátor už navyše nebude pracovať s číslom v trojkovej sústave, namiesto neho si bude pamätať celú poslednú vygenerovanú možnosť a z nej zakaždým vytvorí nasledujúcu. Hlavnou myšlienkou generátora však naďalej zostáva skoré orezávanie farbení s repetíciou. Naša idea dokonca dokáže viac, a to, že sa vôbec k takýmto farbeniam nedostane. Cieľ dosiahneme generovaním farbenia postupne — generátor vytvorí čiastočné farbenie, ktoré odovzdá v našom prípade funkcii *main*. Tá farbenie okontroluje na nerepetitívnosť. Ak bude farbenie repetitívne, môže ho preskočiť a s ním aj celý podstrom nasledujúcich farbení. Ak bude farbenie nerepetitívne,

musí požiadať generátor o rozšírenie tohto farbenia o ďalšiu farbu. Ak bude farbenie nerepetitívne a zároveň nie čiastočné, môžeme cestu prestať kontrolovať — ako v predošlom postupe (obrázok 2). Popísané generovanie do stromu sa dá na príklade cesty $(0, 1, 2), (0, 1, 2), (0, 1, 2), (0, 1, 2)$ znázorniť nasledovne:



Obr. 3: Zoptimalizované generovanie farbení do stromovej štruktúry. Každá ďalšia hĺbka stromu vyjadruje výber farby na ďalšom vrchole cesty

Na obrázku 3 nie je zakreslený celý strom, ale len vetvy, do ktorých sa program dostane. Algoritmus na takomto príklade najskôr vygeneruje farbenie 0. Otestuje ho a zistí, že je nerepetitívne, takže ho rozvíja ďalej — postupuje v stromu hlbšie. Vygeneruje farbenie $0, 0$, o ktorom zistí, že obsahuje repetíciu, takže už nemusí ísť v strome hlbšie. Generátor preto zmení hodnotu poslednej vygenerovanej farbe — nasledujúce farbenie bude $0, 1$, čo je nerepetitívne, takže ďalšie bude hlbšie, a to farbenie $0, 1, 0$, opäť nerepetitívne. Nasleduje teda farbenie $0, 1, 0, 0$, ktoré obsahuje repetíciu na posledných dvoch nulách, takže generátor pokračuje farbením $0, 1, 0, 1$, ktoré je však tiež repetitívne, a to vďaka dvom po sebe nasledujúcim blokom $0, 1$. Vygeneruje sa preto posledné farbenie, farbenie $0, 1, 0, 2$, ktoré je nerepetitívne a obsahuje farbu pre všetky vrcholy, takže daná cesta je v poriadku a môžeme prejsť na ďalšiu. Dôkaz, že spomínané necelé farbenia obsahujúce repetíciu môžeme preskočiť, uvádzame v 3.1.

Ľahko si povšimnúť, že generovanie zakaždým zmení len poslednú farbu predošlého farbenia — buď pridaním novej farby, alebo priamo zmenou poslednej farby. Zdá sa preto neefektívne kontrolovať nerepetitívnosť celého farbenia, keď je zrejmé, že sa farbenie až na poslednú farbu skontrolovalo už v predošlom kroku. Z tohto dôvodu postačuje vykonať kontrolu nerepetitívnosti len pre poslednú farbu, respektíve pre bloky

všetkých možných dĺžok obsahujúce aj poslednú farbu. Dôkaz tohto tvrdenia uvádzame v 3.2. Časová zložitosť kontroly nerepetitívnosti sa z týchto dôvodov zredukuje z $O(n^3)$ na $O(n^2)$

Po spomínanej optimalizácií sme nútení jemne prerobiť i funkciu *main*, nakoľko ona vykonáva kontrolu repetície, a teda v nej musíme rozlíšiť, či sa treba v strome ponoriť hlbšie, alebo podstrom preskočiť.

```
int main()
{
    bool found;
    SimplePathGenerator pathGenerator = SimplePathGenerator(lengthOfPath, colorsInPath);
    Path nowPath = pathGenerator.nextPath();

    while (!(nowPath.empty()))
    {
        found = false;

        PartialColoringGenerator coloringGenerator = PartialColoringGenerator(nowPath);
        Coloring coloring = coloringGenerator.initialColoring();

        while (!(coloring.empty()))
        {
            if (checkNonRepetitivenessOnLastIndex(coloring))
            {
                if (coloringGenerator.isFullColoring())
                {
                    found = true;
                    break;
                }
                else
                {
                    coloring = coloringGenerator.nextColoring();
                }
            }
            else
            {
                //coloring contains repetition
                coloring = coloringGenerator.skipColoring();
            }
        }
        if (!found)
        {
            cout << "COUNTEREXAMPLE!" << endl;
            nowPath.printPath();
        }
        nowPath = pathGenerator.nextPath();
    }
}
```

Obr. 4: Funkcia *main* po optimalizácií generátora farbení

Po našej prvej optimalizácii sme opäť vyrobili tabuľku s výsledkami, ktorá ukazuje, ako sme algoritmus zrýchlili.

Tabuľka 2: Výsledky programu po optimalizácií generátora farbení. Symbol X označuje predošlé výsledky, symbol + značí, že program s danými parametrami úspešné zbehol v čase pod 2 minúty a predošlej verzii sa to nepodarilo.

		Dĺžka cesty								
		4	5	6	7	8	9	10	11	12
Počet farieb	4	X	X	X	X	X	+	+	+	
	5	X	X	X	+					
	6	X	X							
	7	X								
	8	X								
	9									

Z tabuľky možno usúdiť, že sme si výrazne polepšili na dlhších cestách, čo sa aj na základe našich úvah dalo predpokladať. Pri väčšom množstve farieb sme si však nijako relevantne nepomohli, keďže v takýchto prípadoch extrémne narastá aj počet ciest a nie len farbení, ktoré by sme vďaka optimalizácii preskočili.

4.4 Optimalizácia základného generátora ciest

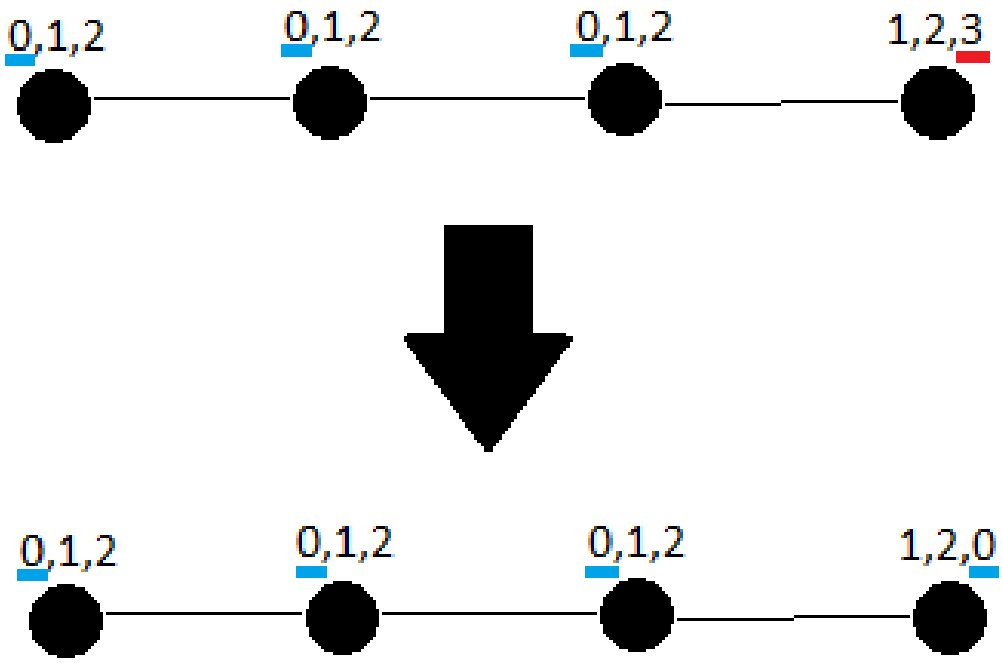
Momentálne náš program preskúmava všetky možnosti ciest a ich trojzoznamov. V nasledujúcom túto skutočnosť zmeníme, nakoľko existujú cesty, a je ich veľmi veľa, ktoré nemusíme vôbec prezerať. Uvedieme dve vylepšenia, ktoré odstránia mnohé cesty, čo nemusia byť algoritmom kontrolované.

V prvom uvedieme, že opäť vytvárame nový súbor, nový generátor, ktorý, ako sme učinili aj s novým generátorom farbení, už nebude pracovať s číslom v exotickej sústave kódujúcim naposledy vygenerované trojzoznami, ale radšej si zapamätáme celý objekt cesty a z neho budeme zakaždým odvodzovať nasledujúcu cestu.

Všimli sme si, že v mnohých prípadoch vygenerovaných ciest sa nachádzajú nejaké dve farby na disjunktných vrcholoch, teda neexistuje trojzoznam, v ktorom by boli obi dve prítomné. Cesty, ktoré majú takéto disjunktné farby vôbec nemusíme generovať, pretože určite niekedy vygenerujeme, alebo sme už vygenerovali cestu, ktorá má na všetkých pozíciách týchto dvoch farieb len jednu farbu. Takáto inštancia cesty je určite ťažšia v zmysle nerepetitívnosti, a teda môžeme cestu s disjunktnými farbami preskočiť. Formálnejší dôkaz, že takéto vylepšenie nepreskočí nijaký protipríklad, uvádzame v 3.3.

Ako príklad aplikácie tejto optimalizácie uvedieme cestu tvaru $(0,1,2),(0,1,2),(0,1,2),(1,2,3)$. Farba 0 sa nachádza na vrcholoch $0,1,2$ a farba 3 je na vrchole 3. Farby 0 a 3 sú teda disjunktné a cestu môžeme preskočiť. Ťažšia inštancia cesty bude

tvaru $(c, 1, 2), (c, 1, 2), (c, 1, 2), (1, 2, c)$, kde c je jedna z povolených farieb okrem 1 a 2. Na nasledujúcom obrázku 5 je znázornený tento príklad s farbou 0 miesto našej farby c pri ťažšej inštancií cesty.



Obr. 5: Názorné zobrazenie príkladu s disjunktnými farbami.

Implementácia optimalizácie bude priamočiara — generátor ciest po vytvorení cesty spustí kontrolu na disjunktnosť všetkých farieb. Ak kontrola nájde disjunktné farby, generátor cestu zahodí a pokračuje na generovanie ďalšej cesty. Pre účely kontroly disjunktnosti si generátor ciest musí navyše pamätať pole, na ktorého pozícii i je množina vrcholov, ktoré obsahujú farbu i v ich trojzozname. Vďaka spomínanému poľu i ostatným úvahám sme naprogramovali takmer optimálny algoritmus pre kontrolu disjunktnosti. Nosné časti funkcie sú v časovej zložitosti $O(n + k)$, kde n je dĺžka cesty a k počet farieb. Avšak inicializácia polí na začiatku funkcie potrebuje až čas $O(n + k^2)$. Dala by sa ale vyňať mimo ňu a v čase $O(n)$ by sme vždy vynulovali tieto polia (v tabuľke o veľkosti $O(k^2)$ vykonávame vždy len $O(n)$ zmien, preto ich vieme v $O(n)$ aj vynulovať), čo by nás vrátilo k optimálnej zložitosti celej funkcie $O(n + k)$. V ďalšej sekcií však použijeme ešte zmenenú implementáciu, a takéto zlepšenie už teda nebude nutné, preto sa ním ani teraz nebudeme zaťažovať.

Vďaka tejto optimalizácii je zrejmé, že cesty, ktoré nebudeme preskakovať, nebudú mať žiadne dve farby na disjunktných vrcholoch. Teda každá farba musí byť s každou inou farbou naraz aspoň v jednom trojzozname. Nech je počet práve používaných farieb na ceste k_p. Najmenší počet výskytov nejakej farby c je $\lceil k_p/2 \rceil$, keďže v každom trojzozname, kde sa farba c vyskytuje, zostávajú dve voľné pozície pre ostatné farby a

každá takáto pozícia môže byť obsadená inou farbou. Keďže je počet všetkých pozícií na ceste dĺžky n rovný $3n$, dostávame nerovnosť $k_p\lceil k_p/2\rceil \leq 3n$. Keďže ale platí $k_p.k_p/2 \leq k_p\lceil k_p/2\rceil$, dostávame užitočný odhad pre maximálny počet farieb na ľubovoľnej ceste, ktorá používa k_p farieb

$$(k_p)^2 \leq 6n. \tag{4.1}$$

Druhá vec, ktorú sme postrehli, je ekvivalentnosť mnohých ciest. Presnejšie povedané, niektoré cesty sa medzi sebou líšia len v použitých farbách. Ak máme napríklad cestu $(0,1,2),(0,1,2),(0,1,2),(0,1,2)$ a zmeníme farbu 0 na farbu 3, dostaneme cestu $(1,2,3),(1,2,3),(1,2,3),(1,2,3)$. Uvedený príklad sme zakreslili na názornom obrázku 6.

Obr. 6: Zobrazenie príkladu dvoch ekvivalentných ciest.

Tieto dve cesty sú ekvivalentné v zmysle, že ak jedna z nich vyprodukuje protipríklad, musí aj druhá a naopak, keď jedna nebude protipríkladom, nemôže byť ani druhá. Formálny dôkaz týchto myšlienok sa nachádza v 3.4. Jedna cesta je v tomto zmysle ekvivalentnosti ekvivalentná s mnohými inými, ktoré náš generátor momentálne vytvára, nie len s jednou ďalšou. Je preto vhodné generátor pozmeniť, aby si z nich vybral len jednu a ostatné preskočil. Otázkou zostáva, ktorá bude tá, čo sa ponechá. Lepšie povedané, aké usporiadanie zvolíme, podľa ktorého zistíme prvú cestu v tomto usporiadaní a práve ňu nepreskočíme. Najintuitívnejšie usporiadanie je v našom prípade lexikografické, a to presnejšie takéto: Cesty prezrieme od nultého (najľavejšieho) vrcholu po posledný (najpravejší), pričom budeme porovnávať farby na vrcholoch tiež najprv nultú, potom prvú a nakoniec druhú. Prvá nerovnosť farieb, na ktorú narazíme, rozhodne o lexikografickosti — cesta s menšou farbou bude lexikograficky skorej ako tá s vyššou farbou.

Pri implementácii sme si uvedomili zásadnú vec. Generátor ciest zakaždým vytvorí len jednu cestu, nie dve, ktoré by sme medzi sebou porovnali. Prvým impulzom bolo vygenerovať všetky ekvivalentné cesty k momentálne vytvorenej ceste a vyrátať, či je tá naša zo všetkých lexikograficky najlepšia, alebo nie je. Avšak uvedomili sme si, že ak máme nejakú cestu, dokážeme k nej vyrobiť lexikograficky najlepšiu v čase $O(n)$, a to tak, že ju prejdeme zľava doprava takisto, ako pri kontrole dvoch ciest v predošlom odstavci. Ak sme na farbe, ktorú sme videli už skôr, do vytváranej lexikograficky najlepšej ju iba prepíšeme na rovnakú pozíciu. Ak však narazíme na farbu, ktorú sme

ešte nevideli, vložíme na to miesto novú farbu, a to najmenšiu takú, aká ešte na ceste nie je. To, že cesty, ktoré môžeme medzi sebou porovnávať lexikograficky, produkujú rovnaký výsledok nerepetitívnosti, vyplýva z definície lexikografickosti 31, a z faktu, že ide len o niekoľko násobné použitie obojstrannej substitúcie 3.5. V našej implementácii dokonca ani nevytvárame popísanú lexikograficky najlepšiu cestu, nakoľko pri prechádzaní vrcholov "zľava doprava" môžeme ihneď kontrolovať či je vygenerovaná cesta lexikograficky prvá a nepotrebujeme túto cestu najprv vytvoriť, a potom ich medzi sebou porovnať. Ľahko nahliadnuť, že ak vygenerujeme lexikograficky najlepšiu cestu, musí mať vždy na prvom vrchole farby $0, 1, 2$. Toto krátke pozorovanie sme zapracovali do generátora tak, že sme natvrdo určili trojzoznam prvého vrcholu na tieto farby a zamedzili sme tiež jeho zmene, čo v konečnom dôsledku signifikantne skrátilo beh programu.

Treba však dodať, že spomínaný postup neponechá len všetky lexikograficky najlepšie cesty, ale i nejaké navyše. Problémové budú farby začínajúce na rovnakých vrcholoch. O nich totiž algoritmus nemá možnosť vedieť, ktorá bude ďalej na ceste lepšie rozmiestnená, čo sa lexikografickosti týka. Ošetrovanie takýchto prípadov preberieme neskôr.

Obe uvedené vylepšenia sme pre prehľadnosť zahrnuli do jednej triedy s názvom "PathRelevanceCheck". Optimalizácie taktiež mierne zmenili časovú zložitosť na $O(2^{\binom{k}{3}} + f(n,k)3^n + g(n,k)n^2)$, kde n je dĺžka cesty, k počet povolených farieb, $f(n)$ je funkcia, ktorá vráti počet vygenerovaných ciest a $g(n)$ zas funkcia, ktorá vráti celkový počet vygenerovaných farbení. Výsledky po optimalizáciách uvádzame opäť v prehľadnej tabuľke.

Tabuľka 3: Výsledky programu po optimalizácií generátora ciest. Symbol X označuje predošlé výsledky, symbol + značí, že program s danými parametrami úspešné zbehol v čase pod 2 minúty a predošlej verzii sa to nepodarilo.

		Dĺžka cesty									
		4	5	6	7	8	9	10	11	12	13
Počet farieb	4	X	X	X	X	X	X	X	X	+	
	5	X	X	X	X						
	6	X	X	+							
	7	X	+								
	8	X	+								
	9	+									
	10	+									

4.5 Výmena generátora ciest

Pri testovaní programu sme zistili zaujímavú skutočnosť, a síce, že najviac času algoritmu zaberie práve generovanie ciest. Na rozdiel od generovania farbení či kontroly nerepetitívnosť je to až neadekvátne veľa. Rozhodli sme sa preto opäť vylepšovať generátor ciest, tentoraz nie len jeho optimalizáciou, ale úplnou výmenou za druhý, inak logicky fungujúci. Náš nový generátor nebude vytvárať cesty vďaka inkrementovaniu predošlej, ale bude meniť pozície pre celú jednu farbu. Popíšme ale postup algoritmu presnejšie. Algoritmus je rekurzívny, pričom so zväčšujúcou sa hĺbkou vnorenia vždy inkrementuje farbu, ktorú má vygenerovať. Taktiež si pamätá, na ktorej farbe sa pri predošlom generovaní skončilo, aby od nej mohol v ďalšom behu pokračovať. Nech algoritmus generuje farbu c. Najprv skontroluje či je daná farba povolená (triviálna podmienka rekurzie). Následne preskočíme farby, ktoré sú menšie ako farba, na akej sa pri predošlom generovaní skončilo, presnejšie, ihneď sa vnoríme do rekurzie s inkrementovanou farbou. V našom príklade sa vnoríme až po farbu c. Nasleduje samotné generovanie farby. Najskôr sa zavolá funkcia na vygenerovanie farby na nasledujúcich pozíciách — tá bude popísaná v nasledujúcom odstavci. Ak bola farba vygenerovaná už na všetkých možných miestach, algoritmus sa raz z rekurzie vynorí s touto informáciou. Ak nebola farba vygenerovaná celá, prejde sa na kontrolu disjunktnosti farby, ako tomu bolo i pri predošlom generátore, avšak teraz nám stačí skontrolovať len generovanú farbu c. Ak je farba c disjunktná, generovanie farby sa zopakuje, a teda sa testujú nasledujúce pozície farby c. Ak nie je, overuje sa nasledujúca podmienka. Ak cesta má vo všetkých trojzoznamoch na každej pozícií validnú farbu, vynoríme sa zo všetkých rekurzívnych volaní a cestu vrátime, pretože cesta musí byť ďalej posunutá do generátora farbení. Ak nie je cesta zaplnená na všetkých pozíciách, vnoríme sa do rekurzie s farbou $c+1$. Po vrátení sa z rekurzie pokračujeme generovaním farby c na ďalších pozíciách, to jest postup generovania farby opakujeme. Názornú ukážku kódu neuvádzame, nakoľko je funkcia dlhá a všetky jej kľúčové prvky sme už popísali a vysvetlili. Čitateľ si ju môže pozrieť na priloženom linku “https://github.com/LordLoles/Diplomovka”, pričom ide o súbor “./prog/PathGenerators/ByColorPathGenerator.cpp” s príslušnou hlavičkou a o funkciu s názvom “nextFullPathGenerator”.

Zostáva nám ešte uviesť, ako bude vyzerať konkrétna funkcia na generovanie danej farby na nasledujúcich pozíciách. Opäť uvažujme, že generovanou farbou je farba c. V prvom je potrebné vygenerovať farby tak, aby bola výsledná cesta lexikograficky najlepšia. Toto sa dá docieliť vcelku jednoducho, a teda už nebude potrebné vykonávať žiadne kontroly lexikografickosti. Stačí, ak sa pri prvom volaní funkcie vygeneruje farba c na prvý vrchol zľava, ktorý ešte neobsahuje tri farby. Táto myšlienka vyplýva z úvah v predošlej podkapitole 4.4 a z vety 3.5. Odhliadnuc od prvého vrcholu, ktorý musí pre účel lexikografickosti obsahovať farbu c, sa bude farba

c generovať spôsobom pripomínajúcim inkrementovanie binárneho čísla. Najlepšie sa však celý spôsob generácie farby vysvetlí rovno na príklade. Majme teda doposiaľ vytvorenú cestu $(0,1,2),(0,1,),(0,2,),(0,1,2),(1,,)$, kde voľné miesta v trojzoznamoch znamenajú zatiaľ žiadnou farbou neobsadené pozície. Nech generujeme farbu 3. Ako prvý vrchol, kam sa farba 3 umiestni, je, pre dosiahnutie lexikografickosti, vrchol jedna, a teda nová cesta bude tvaru $(0,1,2),(0,1,3),(0,2,),(0,1,2),(1,,)$. Ak by sme binárne zaznačili vrcholy v poradí zľava doprava, kde sa farba 3 nachádza či nenachádza, dostali by sme postupnosť bitov 01000. Pri opätovnom volaní funkcie generovania farby sa už postupuje stále rovnako, a to tak, že sa prezerajú vrcholy tentoraz sprava doľava. Ak vrchol obsahuje farbu 3, farba sa odstráni, ak ju neobsahuje a má voľné miesto, farba sa vloží do zodpovedajúceho trojzoznamu na najľavejšiu voľnú pozíciu. Algoritmus teda interne nepracuje s bitovými reťazcami, tie slúžia len na ozrejmenie jeho fungovania. V našom prípade sa teda ako ďalšia vytvorí cesta $(0,1,2),(0,1,3),(0,2,),(0,1,2),(1,3,)$, čo zodpovedá spomínanej bitovej reprezentácií 01001 — z tohto dôvodu sme povedali, že nám postup generátora pripomína inkrementovanie binárneho čísla. Ďalšia postupnosť bitov by mala teda byť 01010, ale keďže je tretí vrchol zaplnený, musíme ho vynechať. Nasledujúca postupnosť preto bude 01100 a prislúchajúca cesta $(0,1,2),(0,1,3),(0,2,3),(0,1,2),(1,,)$. Predposlednou možnosťou sa stane cesta $(0,1,2),(0,1,3),(0,2,3),(0,1,2),(1,3,)$ a poslednou bude opäť tá začínajúca, teda $(0,1,2),(0,1,),(0,2,),(0,1,2),(1,,)$, pričom funkcia odpovie, že už vygenerovala celú farbu 3. Je očividné, že popísaný algoritmus vygeneruje všetky možnosti pre ľubovoľnú farbu (s tým, že prvá pozícia farbu nutne obsahuje), keďže funguje rovnako ako inkrementácia binárneho čísla. Pre jemnú nezhodu pojmu "inkrementácia" s nami vykonávanou operáciou uvádzame, že ak by bitová postupnosť obsahovala iba nezaplnené vrcholy na ceste (neberúc do úvahy generovanú farbu) a navyše by najľavejší z nich vynechala (pre lexikografickosť), išlo by presne o operáciu inkrementovania. Ukážku implementovaného kódu opäť neuvádzame pre jeho priveľkú dĺžku. Funkcia sa však nachádza v tom istom súbore, o akom sme pojednávali v predošlom odstavci a nesie názov "generateNextColor".

Nový generátor ciest má tiež inú časovú zložitosť ako predošlý, konkrétne $O(\binom{k}{3}^n + f(n,k)3^n + g(n,k)n^2)$, kde n je dĺžka cesty, k počet povolených farieb, $f(n,k)$ je funkcia, ktorá vráti počet vygenerovaných ciest novým generátorom a $g(n,k)$ je funkcia vracajúca počet vygenerovaných farbení.

Taktiež sme učinili významné pozorovanie. Už na začiatku sme si boli vedomí skutočnosti, že ohraničenie počtu farieb k pre danú dĺžku cesty n sa dá zhora odhadnúť a vyrátali sme, že toto ohraničenie rastie so zvyšujúcou sa dĺžkou cesty n rádovo s druhou odmocninou 4.1. Avšak implementácia lexikografickosti nám zabezpečila, že farby s vysokými hodnotami, ktoré nevytvoria lexikograficky najlepšiu možnosť, sa do vygenerovanej cesty vôbec nedostanú. Do algoritmu sme teda pridali premennú, ktorá si

pamätá hodnotu najvyššej použitej farby, z čoho si vieme hravo zrátať počet použitých farieb. Ak sme teda zadali ako vstupný parameter počtu farieb k dosť vysoké číslo, zistili sme, koľko maximálne farieb algoritmu postačuje na vygenerovanie všetkých podmienkam vyhovujúcich ciest. Maximálne počty farieb sme v nasledujúcej tabuľke 4 zaznačili písmenom m. Taktiež, po dosiahnutí maximálneho počtu farieb sme sa už výpočtami s vyšším počtom farieb nezapodievali, preto ich v tabuľke ani neuvádzame. Za zmienku tiež stojí, že súčasná implementácia optimalizuje najmä inštancie s vyšším počtom farieb (ako i časová zložitosť napovedá) a pre dlhšie cesty je, naopak, pomalšia.

Tabuľka 4: Výsledky programu so zmeneným generátorom ciest. Symbol X označuje predošlé výsledky, symbol + značí, že program s danými parametrami úspešné zbehol v čase pod 2 minúty a predošlej verzii sa to nepodarilo. Symbol − hovorí, že program nezbehol v časovom limite, ale predošlej verzii sa to podarilo. Symbol m znamená, že pre danú dĺžku cesty je počet farieb maximálny.

		Dĺžka cesty									
		4	5	6	7	8	9	10	11	12	13
Počet farieb	4	X	X	X	X	X	−	−	−	−	
	5	Xm	X	X	X	+					
	6		Xm	Xm	+						
	7				+m						
	8										
	9										
	10										

Z tabuľky možno vyčítať, že cesty dĺžky menšej ako 8 sme dokázali okontrolovať celé, teda pre všetky relevantné počty farieb. Je tiež vhodné si všimnúť, že maximálne počty farieb v tabuľke veľmi sľubne korešpondujú so vzorcom 4.1, ktorý o maximálnych počtoch farieb pojednáva.

Spomenieme i jedno exemplárne meranie, ktoré sa do tabuľky neuviedlo. V tabuľke 4 uvádzame, že pre vstupné hodnoty $n = 8$ a $k = 6$ program do 2 minút nezbehol. Podarilo sa mu to však v priebehu 3 minút. Keď sme ale zadali $k = 20$, program skončil po 4 a pol minúte, čo nie je ani dvojnásobok. Navyše sme sa dozvedeli, že počet použitých farieb bol 8. Toto meranie výstižne demonštruje naše predošlé úvahy, že veľký počet povolených farieb až tak závažne neovplyvní rýchlosť programu.

4.6 Optimalizácia nového generátora ciest

Po rýchlom teste sme zistili, že najpomalšou časťou našej implementácie je stále generátor ciest. Pri hlbšom debuggovaní sme si uvedomili, že program najviac v generátore ciest spomaľuje veľké množstvo možností pre nejakú farbu. Pridali sme teda doňho jednu funkciu, ktorá ak zistí, že s nasledujúcimi farbami, čiže neuvažujúc momentálnu farbu, nie je možné cestu naplniť, tak s práve generovanou farbou naplní tie vrcholy, ktoré boli problémové. Napríklad pre doposiaľ vytvorenú cestu $(0, 1, 2), (0, ,), (0, 1,), (1, ,)$ s počtom farieb 5 a pri generovaní farby 3 algoritmus zistí, že ak na prvý a tretí vrchol farbu 3 neumiestnime, cesta nebude nikdy celá zaplnená, preto ich tam natvrdo pridá. Samozrejme, ide o názorný príklad, nakoľko v reáli sa vygeneruje farba 3 na prvý vrchol tak či tak, lebo inak by sa porušila lexikografickosť.

```
/*
* Generates given color in 'lastResult' on poistions, where it is not possible for
* the next (remaining) colors to make the path full.
*
* Returns true, if at least one color was added.
* Returns false, if 'lastResult' was unchanged.
*
* O(n)
*/
bool ByColorPathGenerator::generateFillingColors(int color)
{
    set<int> colorUsage = colorsUsage.at(color);
    bool canBeFull = canBeFilled(color + 1);

    if (!canBeFull)
    {
        int remainingColors = colorsInPath() - (color + 1); //not counting with this 'color'
        for (int i = 0; i < length; i++)
        {
            if (!canBeFilledVertex(i, remainingColors) && (!colorUsage.count(i)))
            {
                colorsUsage.at(color).insert(i);
                int pos = freePosInVertex(i);
                lastResult = lastResult.set(i, pos, color);
            }
        }
    }
    return !canBeFull;
}
```

Obr. 7: Funkcia vytvárajúca farby, ktoré sú potrebné, aby cesta mala možnosť byť naplnená.

Funkcia pracuje v čase $O(n)$, ako sa dá vyčítať i v dokumentácii z obrázka 7. Pre pripomenutie uvedieme, že *colorsUsage* je premenná, ktorá si pamätá pre danú farbu množinu vrcholov, na ktorých sa farba momentálne nachádza.

Prišli sme i s druhým nápadom na optimalizáciu, a to s aplikáciou lexikografickosti nie len na prvé výskyty farieb, ale i na všetky dvojice farieb a ich pozícií. Samozrejme, túto podmienku, nazvyme ju lexikografickosť farieb, budeme overovať hneď po vygenerovaní farby, aby sme nezhoršili časovú zložitosť. Všetko opäť vysvetlíme na príklade. Majme cestu $(0, 1, 2), (0, 1,), (0, 2,)$, pričom sme práve vygenerovali farbu 2 a chceme

skontrolovať lexikografickosť farby 2 s ostatnými farbami. Mali by sme pozrieť na všetky ostatné farby, no je zjavné, že nám stačí kontrolovať len tie farby, ktorých prvý výskyt zľava je na rovnakom vrchole ako prvý výskyt našej farby 2. Vieme totiž, že farby začínajúce skôr ako farba 2 sú lexikograficky skorej a taktiež vďaka lexikografickosti cesty vieme, že sú i menšie ako farba 2, a teda sú v poriadku. Pre farby začínajúce neskôr ako farba 2 platí obdobné tvrdenie. Hoci je toto zistenie poľahčujúce, v tomto príklade budeme musieť skontrolovať i tak všetky ostatné farby, teda farby $0, 1$. Predstavu nám opäť výrazne uľahčí zaznačenie si výskytu farby v bitovej postupnosti, ktorej i–ta cifra značí prítomnosť farby na i–tom vrchole cesty. Farba 0 je na všetkých vrcholoch, a teda jej bitová postupnosť bude 111. Pre farbu 1 to bude 110 a pre farbu 2 ide o postupnosť 101. Ak by sme teraz postupnosti pre farby 0 a 1 bitovo porovnali s postupnosťou pre farbu 2, zistili by sme, že sú obe väčšie, a teda sú obe farby na skorších vrcholoch ako farba 2. Daná cesta preto kontrolou lexikografickosti pre farbu 2 prejde. Ak by sme však cestu jemne pozmenili na tvar $(0, 1, 2), (0, 2,), (0, 1,)$, zistíme, že takáto cesta kontrolou neprejde, nakoľko bitová postupnosť pre farbu 2 je "väčšia" ako bitová postupnosť pre farbu 1 ($110 > 101$). Všetky cesty, ktoré touto kontrolou neprejdú, budeme preskakovať. Dôvody, prečo tak môžeme učiniť, vyplývajú priamo z vety 3.5.

Časová zložitosť sa po uvedených vylepšeniach vďaka *O–notácii* nezmenila. Dosiahnuté výsledky sme zhrnuli do nasledovnej tabuľky:

Tabuľka 5: Výsledky programu s vylepšeným novým generátorom ciest. Symbol X označuje predošlé výsledky, symbol + značí, že program s danými parametrami úspešné zbehol v čase pod 2 minúty a predošlej verzii sa to nepodarilo. Symbol m znamená, že pre danú dĺžku cesty je počet farieb maximálny.

		Dĺžka cesty									
		4	5	6	7	8	9	10	11	12	13
Počet farieb	4	X	X	X	X	X	+				
	5	Xm	X	X	X	X					
	6		Xm	Xm	X	+					
	7				Xm						
	8										
	9										
	10										

4.7 Paralelizácia

Nakoľko je v našom záujme otestovať hypotézu 1 pre dĺžku cesty i počet farieb aspoň 10, nebudeme iba optimalizovať kód. Pre dosiahnutie nášho cieľa bude kľúčová paralelizácia. Upravíme funkciu *main* a generátor ciest tak, aby bol ich výsledkom súbor pozostávajúci z neúplných ciest. Pôjde o všetky možnosti ciest s farbami $0, 1$, ktoré do súboru zapíšeme na každý riadok jednu vo formáte "{0 1 -1} {-1 -1 -1} {0 1 -1} {1 -1 -1}", kde kučeravé zátvorky ohraničujú trojzoznam vrcholu a číslo -1 značí nezaplnenú pozíciu. Generovanie takýchto čiastočných ciest, samozrejme, tiež podlieha rovnakým podmienkam, ako predošlé verzie programu — nevytvoria sa teda úplne všetky cesty, len tie relevantné. Ako sme už spomínali, cesty dĺžky 8 sa nám podarilo preskúmať všetky približne za 5 minút. Spomínané súbory sme preto vytvorili len pre cesty dĺžky 9 a 10, pričom súbor pre cesty dĺžky 9 obsahoval 32896 ciest a súbor pre cesty dĺžky 10 až 131328 ciest.

```
int main()
{
    ofstream file ;
    file.open("PathsStarts.txt", ios::trunc);

    if (!file.is_open())
    {
        cout << "Unable to open file";
        return 0;
    }

    bool found;
    int paths = 0;
    ByColorCappedColorsPathGenerator pathGenerator = ByColorCappedColorsPathGenerator(lengthOfPath, 2);
    Path nowPath = pathGenerator.initialPath();

    while (!(nowPath.empty()))
    {
        paths++;
        file << nowPath.to_string() << "\n";
        nowPath = pathGenerator.nextPath();
    }

    file.close();
    cout << "Paths count " << paths << endl;
    cout << "Colors used " << pathGenerator.colorsUsed() << endl;
}
```

Obr. 8: Funkcia main vytvárajúca čiastočné cesty s farbami 0,1 a zapisujúca ich do súboru.

Druhým krokom je vytvorenie funkcie *main* a generátor ciest, ktoré budú vedieť takýto súbor s cestami prečítať a dokončiť generovanie cesty na určitom riadku súboru. Náš program sme napísali tak, aby dokončoval všetky cesty na $(k.n + o)$–tom riadku súboru, kde n je počet častí, na ktoré chceme súbor rozdeliť, o je takzvaný *offset*,

teda konštanta $< n$, vďaka ktorej dokážeme skontrolovať vždy iný riadok spomedzi nasledujúcich n a premenná k sa začínajúc od nuly neustále inkrementuje až pokým nenarazíme na koniec čítaného súboru.

Popísaným spôsobom sme všetky relevantné cesty dĺžky 9 prezreli na troch procesoroch za približne 72 minút. Presnejšie išlo dokopy o 13126 sekúnd, čiže o $3,64$ hodín pre jeden procesor. Odhadovaný čas výpočtu bol pritom $3,25$, čo je dosť blízko výslednému času.

Odhadovaný čas na skontrolovanie ciest dĺžky 10 sme stanovili na 230 hodín na jeden procesor. Program sme rozdelili a spustili na mnohých procesoroch a dokopy sme na výpočet potrebovali 1031949700 milisekúnd, čo predstavuje $286,65$ hodín. Tu je už rozdiel odhadu a reálne stráveného času badateľnejší.

Učinili sme i odhad pre cesty dĺžky 11. Vytvorený súbor pre takéto cesty obsahuje 524800 neúplných ciest. Pri rozdelení súboru na 10000 častí trval výpočet jednej časti 6115810 milisekúnd. Celkový beh programu pre cesty dĺžky 11 tak vychádza na $16988,36$ hodín na jeden procesor, pričom, ako tomu bolo v predošlých prípadoch, bude náš odhad pravdepodobne menší ako skutočný čas výpočtu.

Nakoniec, uvádzame i tabuľku zobrazujúcu dosiahnuté výsledky vďaka paralelizácii. Jej účel je však len doplňujúci, nakoľko sa už nedá porovnávať s predchádzajúcimi, pretože čas výpočtu ani počet procesorov nebol rovnaký.

Tabuľka 6: Výsledky programu pre paralelizovaný program. Symbol X označuje predošlé výsledky, symbol + značí novodosiahnuté výsledky. Symbol m znamená, že pre danú dĺžku cesty je počet farieb maximálny.

		Dĺžka cesty									
		4	5	6	7	8	9	10	11	12	13
Počet farieb	4	X	X	X	X	X	X	+			
	5	Xm	X	X	X	X	+	+			
	6		Xm	Xm	X	X	+	+			
	7				Xm	+	+	+			
	8					+m	+m	+m			
	9										
	10										

4.8 Neimplementované vylepšenia

Vytvorený program i zo všetkými opisovanými optimalizáciami je dostupný na adrese "https://github.com/LordLoles/Diplomovka".

Avšak výsledná implementácia stále prechádza cesty, o ktorých by sme dopredu vedeli povedať, že určite nevytvoria protipríklad pre hypotézu 1. Veľa takýchto nepotrebných ciest sa nám už podarilo odfiltrovať. Máme však nápady i na ďalšie vylepšenia, ktoré sa nám už naimplementovať nepodaria. Sú teda silnými kandidátmi na smery, ktorými by sa práca mohla v budúcnosti uberať.

Dobrým nápadom sa zdá byť preskúmanie lexikografickosti i na otočenej ceste. Teda vytvoriť lexikograficky najlepšiu cestu, ako tomu je doteraz a pozrieť sa či nie je lexikograficky lepšia tá istá cesta, no prezeraná sprava doľava. Ak je, cestu môžeme vynechať, pretože istotne niekedy vygenerujeme aj tú opačnú. Takáto optimalizácia by mala vynechať presne polovicu ciest spomedzi všetkých, ktoré nemajú v prvej polovici rovnaké trojzoznamy ako v otočenej druhej — alebo nepresne, no výstižnejšie povedané, ktoré nie sú palindrómy.

Inou možnosťou je pozrieť sa na samotné generovanie farieb pre cesty. Totiž ak sa vyskytne neúplne zaplnená cesta, ktorá má len jeden nie zaplnený vrchol, na ktorom je zatiaľ farba c, ďalšie generované farby musia byť na tomto vrchole, aby sme nevyprodukovali cestu s disjunktnými farbami. Napríklad pre cestu $(0,1,2),(0,,),(1,,),(1,,)$ a pre generovanú farbu 3, musíme túto farbu umiestniť na vrchol k nule, keďže inak by boli farby 0 a 3 disjunktné. Za uváženie stojí i nutnosť byť s jednou jednotkou (uvažujúc terajší príklad), čiže na jednom z posledných dvoch vrcholov. No zistili sme, že vo všeobecnosti by takéto vylepšenie zapríčinilo prílišné rozvetvenie všetkých možností, nakoľko by farba 3 mohla byť pri jednej jednotke, pri druhej alebo oboch. Ak by navyše cesta obsahovala farbu 1 takýmto spôsobom na viacerých, nie len dvoch vrcholoch, alebo by nebodaj obsahovala nejakú inú farbu podobne rozmiestnenú ako teraz farba 1, kontrolovanie všetkých týchto možností by sa priveľmi nelíšilo od terajšieho riešenia.

Najsofistikovanejším sa zdá byť nasledovné pozorovanie. Ak sa nejaká farba c na ceste nachádza len v dvoch trojzoznamoch a navyše ak bude medzi týmito dvomi vrcholmi viac ako polovica vrcholov cesty, teda budú dostatočne vzdialené, tieto trojzoznamy neovplyvnia nerepetitívnosť. Presnejšie, ak na tejto ceste budeme vytvárať farbenie, môžeme farbu c na dvoch vzdialených pozíciách vybrať vždy, nakoľko podpostupnosti obsahujúce obe farby nebudú repetíciami. Či je farbenie nerepetitívne preto plne závisí od vrcholov medzi spomínanými dvomi vrcholmi (ale aj od častí na začiatku a konci cesty, tie sú však kratšie). Ak ale usudzujeme, že menšie cesty sme už skontrolovali, môžeme prehlásiť, že takéto časti cesty už skontrolované boli, a teda celú cestu môžeme preskočiť. Dokonca sme usúdili, že ak je spomínaná medzera medzi jednou farbou c väčšia ako polovica dĺžky cesty, môžu byť v ľavej i pravej časti farby c rozmiestnené ľubovoľne.

Hodnotné sa zdá byť i zamyslenie o použití predošlej myšlienky na cesty, ktoré majú spomínanú medzeru medzi vrcholmi s nejakou farbou c menšiu ako polovicu

dĺžky cesty. Ak by táto medzera pridaním jedného trojzoznamu mala dĺžku väčšiu ako polovica dĺžky cesty, dá sa k medzere pridať každý možný trojzoznam a otestovať nerepetitívnosť — výsledky si niekde zapísať a následne ich používať pre generované inštancie ciest.

Záver

V práci sme sa venovali počítačovému dôkazu hypotézy $\pi_{ch}(P_n) = 3$, pre každú cestu $P_n, n \geq 4$. Tento Π_2^P–úplný problém sme zvládli overiť pre všetky možnosti zoznamových ciest dĺžky 10 a menej. Hypotézu sme pre tieto cesty overili tak, že sme preskúmali všetky možné zoznamové cesty so všetkými relevantnými možnosťami trojzoznamov pre každý vrchol a ukázali sme existenciu nerepetitívneho farbenia pre každú jednu zoznamovú cestu. Keďže išlo o počítačový dôkaz a výpis nerepetitívneho farbenia pre každú cestu by bol nereálny, uspokojili sme sa s faktom, že program nenašiel protipríklad, teda nenašiel zoznamovú cestu, v ktorej sú všetky jej farbenia repetitívne. Ukázali sme, že generovanie necelých farbení do stromovej štruktúry na konkrétnej ceste je oveľa efektívnejšie, a oveľa skôr tak dospejeme k nerepetitívnemu farbeniu. Generovanie farbení do stromu nám napomohlo i v kontrole nerepetitívnosti, nakoľko tá už mohla byť prevádzaná len na poslednej farbe z uvažovaného farbenia. Ukázali sme tiež, že ľubovoľná výmena farieb na zoznamovej ceste nezmení fakt, či je repetitívna alebo nie. Takáto výmena pritom môže prebiehať na farbách patriacim ceste i na farbách nepatriacim. Vďaka týmto pozorovaniam sme usúdili, že cesty obsahujúce disjunktné farby môžeme v algoritme vynechávať. Z rovnakého dôvodu môžeme vynechávať i cesty, ktoré nie sú lexikograficky najlepšie spomedzi tých, ktoré dostaneme permutáciou jej farieb, pričom vymieňame všetky výskyty farby naraz. Ďalej sme zistili horný odhad na maximálny počet povolených farieb na ceste $k^2 \leq 6n$, čo sa nám overilo i vďaka myšlienke, že lexikografickosť ciest nepovolí príliš vysoké hodnoty pre farby. Táto myšlienka nám dovolila skúmať nerepetitívne zoznamové farbenia ciest v podstate len s jedným parametrom, a teda s dĺžkou ciest, nakoľko nám bolo umožnené rátať vždy s maximálnym možným počtom farieb. Implementovanými algoritmami sme dokázali platnosť hypotézy pre dĺžky ciest najviac 8 pri behu programu na jednom procesore. Následne sme vytvorili i paralelizovateľnú verziu programu, ktorou sme overili ostávajúce cesty s dĺžkami najviac 10.

Tabuľka 7: Výsledky získané spojením rôznych implementácií. Symbol X označuje, že hypotéza pre danú dĺžku a daný počet farieb platí. Symbol m znamená, že pre danú dĺžku cesty je počet farieb maximálny.

		Dĺžka cesty									
		4	5	6	7	8	9	10	11	12	13
Počet farieb	4	X	X	X	X	X	X	X	X	X	
	5	Xm	X	X	X	X	X	X	X		
	6		Xm	Xm	X	X	X	X			
	7				Xm	X	X	X			
	8					Xm	Xm	Xm			
	9										
	10										

K možným pokračovaniam pre implementačnú časť práce sme sa už vyjadrili v časti 4.8. Nakoľko beh terajšej implementácie pre cesty dĺžky 11 odhadujeme na 16988, 36 hodín na jeden procesor, bolo by rozumné najskôr naprogramovať spomínané optimalizácie a až potom overovať hypotézu pre nasledujúce dĺžky ciest. Ďalším možným pokračovaním práce by mohlo byť nadviazanie na dokázané čiastkové výsledky a odraziť sa z nich pri dôkladnejšom teoretickom skúmaní problému nerepetitívneho zoznamového farbenia ciest.

Presná hodnota Thueho čísla výberu pre cesty $\pi_{ch}(P_n)$ ostáva stále otvoreným problémom. Thueho číslo výberu pre cesty je v mnohých matematických i informatických odvetviach hojne využívané. Hypotézu, že je táto hodnota rovná trom, sme dokázali overiť pre všetky cesty s dĺžkou najviac 10, čím sme, dúfame, dostatočne napomohli spomínaným odvetviam v ich nasledujúcich bádaniach.

Literatúra

[1] Jean-Paul Allouche, Jeffrey Shallit, et al. *Automatic sequences: theory, applications, generalizations*. Cambridge university press, 2003.

[2] Noga Alon, Jarosław Grytczuk, Mariusz Hałuszczak, and Oliver Riordan. Nonrepetitive colorings of graphs. *Random Structures & Algorithms*, 21(3-4):336–346, 2002.

[3] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

[4] János Barát and Péter Varjú. On square-free vertex colorings of graphs. *Studia Scientiarum Mathematicarum Hungarica*, 44(3):411–422, 2007.

[5] Jean Berstel. *Axel Thue's papers on repetitions in words: a translation*, volume 20. Départements de mathématiques et d'informatique, Université du Québec à Montréal, 1995.

[6] Julien Cassaigne. Counting overlap-free binary words. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 216–225. Springer, 1993.

[7] James D Currie. There are ternary circular square-free words of length n for n>= 18. 2002.

[8] Reinhard Diestel. Graph theory, 2000. [Citované 2020-2-2] Dostupné z `http://www.dcs.fmph.uniba.sk/~haviarova/uktg/#materialy`, preklad Peter Hraško: Teória grafov.

[9] Jaroslaw Grytczuk. Nonrepetitive colorings of graphs a survey. *International journal of mathematics and mathematical sciences*, 2007, 2007.

[10] Jarosław Grytczuk, Jakub Kozik, and Piotr Micek. New approach to nonrepetitive sequences. *Random Structures & Algorithms*, 42(2):214–225, 2013.

[11] Jarosław Grytczuk, Jakub Przybyło, and Xuding Zhu. Nonrepetitive list colourings of paths. *Random Structures & Algorithms*, 38(1-2):162–173, 2011.

[12] Andre Kündgen and Michael J Pelsmajer. Nonrepetitive colorings of graphs of bounded tree-width. *Discrete Mathematics*, 308(19):4473–4478, 2008.

[13] M_ Lothaire. *Combinatorics on words*, volume 17. Cambridge university press, 1997.

[14] Christian Maudutt. Multiplicative properties of the thue-morse sequence. *Periodica Mathematica Hungarica*, 43(1):137–153, 2002.

[15] Harold Marston Morse. Recurrent geodesics on a surface of negative curvature. *Transactions of the American Mathematical Society*, 22(1):84–100, 1921.

[16] Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):1–15, 2010.

[17] Seyyed Hamoon Mousavi Haji. Repetition in words. Master's thesis, University of Waterloo, 2013.

[18] Edita Máčajová. Prednášky z teórie grafov, 2019. [Citované 2020-2-2] Dostupné z `http://www.dcs.fmph.uniba.sk/~macajova/TG`.

[19] Eugéne Prouhet. Mémoire sur quelques relations entre les puissances des nombres. *CR Acad. Sci. Paris*, 33(225):1851, 1851.

[20] Neil Robertson and Paul D Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986.

[21] A Thue. Uber unendliche zeichenreichen. norske vid selsk. skr. i. mat. nat. kl. christian, 7: 1–22, 1906. *English translation:[9].*

[22] V Voloshin. Graph coloring: History, results and open problems. *Alabama Journal of Mathematics, Spring/Fall*, 2009.

[23] David R Wood. Nonrepetitive graph colouring. *arXiv preprint arXiv:2009.02001*, 2020.

[24] Xiao Zhou and Takao Nishizeki. Edge-coloring and f-coloring for various classes of graphs. In *Graph Algorithms And Applications I*, pages 241–258. World Scientific, 2002.

[25] Erika Fecková Škrabuľáková. On some of the thue type graph colouring concepts, 2016. [Citované 2020-2-2] Dostupné z `http://www.st.fmph.uniba.sk/~stupal1/diplomovka/res/On%20some%20of%20the%20Thue%20type%20graph%20colouring%20concepts.pdf`.