UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY

Výskum online problémov z hľadiska advice complexity

2013

Peter Herman

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY

Výskum online problémov z hľadiska advice complexity

(diplomová práca)

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Michal Forišek PhD.

Bratislava, 2013

Peter Herman

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Peter Herman
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Výskum online problémov z hľadiska advice complexity

Cieľ: Advice complexity je novou formou merania množstva informácie, ktoré od seba odlišuje online a offline verziu problému. Prvým cieľom diplomovej práce je zhotoviť prehľad doterajšieho výskumu v tejto oblasti.
Nie je evidentné, či je súčasná definícia advice complexity optimálna, nevýhodou je hlavne skutočnosť, že čerpanie rád od orákula môže prebehnúť offline na začiatku výpočtu. Súčasťou práce by mala byť analýza tohoto problému a návrh možných riešení.
Existujú ešte viaceré triedy online algoritmov, ktoré neboli skúmané z hľadiska tejto zložitosti. V rámci práce by mal byť vhodný online problém analyzovaný ako z hľadiska súčasnej advice complexity, tak aj z nového pohľadu, ak sa nejaký podarí identifikovať.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 30.11.2011

Dátum schválenia: 30.11.2011 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Práca sa zaoberá online problémami z hľadiska advice complexity – teda minimálneho množstva informácie, ktoré musí byť poskytnuté orákulom pre dosiahnutie určitého aproximačného pomeru, respektíve optimality. Venovali sme sa farbeniu grafov, konkrétne farbeniu všeobecných grafov prehľadávaných do hĺbky a grafov vnoriteľných do mriežky. Ďalej sme navrhli nový model pre advice complexity, ktorý umožňuje meniť rýchlosť, ktorou je rada čítaná, a analyzovali jeho správanie na problémoch alokovania disjunktných ciest a pri hľadaní maximálnej súvislej podpostupnosti. Navyše sme sa pokúsili zakomponovať radu do rekurzívnych funkcií, a to konkrétne návrhom triedy primitívnej rekurzie s radou.

Kľúčové slová: poradná zložitosť, online farbenie grafov, alokovanie disjunktných ciest, primitívna rekurzia s radou

Abstract

In our work we analyze advice complexity in online problems. We consider online graph coloring on graphs embeddable into grid and a special case for general graphs. Furthermore we propose a new model for advice complexity allowing control over distribution of advice and we analyze its behavior on problems of disjoint path allocation and search for maximal subsequence. Finally we propose a class of primitive recursive functions with advice.

Key words: advice complexity, online graph coloring, disjoint path allocation, primitive recursion with advice

Obsah

Úvod

Miera zložitosti online problémov bola zvyčajne určovaná v pomere správnosti riešenia k optimálnemu algoritmu, ktorý vedel vopred celý vstup.

Nedávno sa však objavila myšlienka merať túto zložitosť ako množstvo dodatočnej informácie, ktorú náš online algoritmus potrebuje získať od orákula poznajúceho celý vstup, aby bol rovnako efektívny ako algoritmus riešiaci offline verziu problému.[DKP08]

Nie je jasné, či súčasná definícia poradnej zložitosti (advice complexity) presne modeluje tento jav, pretože mimo iné umožňuje vyčerpať celú dodatočnú informáciu na začiatku výpočtu. V tejto práci by sme chceli navrhnúť novú definíciu poradnej zložitosti a analyzovať, ako bude vplývať na poradnú zložitosť to, keď budeme požadovať, aby bolo množstvo informácie rovnomerne distribuované pre spracovanie každej požiadavky.

Zároveň si položíme otázku, v ktorých oblastiach teoretickej informatiky by bolo ešte možné radu pri výpočtoch využiť.

Kapitola 1

Prehľad problematiky

1.1 Čo je online algoritmus

Výpočtové problémy vieme pri ich špecifikovaní rozdeľovať podľa mnohých kritérií. Jedným z týchto kritérií môže byť aj to, či je programu známy celý vstup vopred. Ak sa zamyslíme, uvedomíme si, že v bežnej praxi sa často nevyskytujú problémy v takej forme, v akej sa s nimi stretávajú informatici pri analýze a navrhovaní algoritmov. Jedným z veľmi často sa vyskytujúcich predpokladov - považovaným vlastne za akýsi štandard - je, že algoritmus počas svojho výpočtu vidí celý vstup. Teda predpokladáme, že niekto údaje a požiadavky zozbieral a teraz budú jednorazovo vyhodnotené. Ak sa ale zamyslíme nad reálnym prostredím, uvedomíme si, že existuje veľa problémov, kde nie je možné predikovať celý vstup.

Tieto problémy sú zasadené v takzvanom online prostredí, kde je vstup rozdelený do niekoľkých blokov, ktoré chodia programu postupne. Na tieto bloky musí algoritmus odpovedať priebežne. Jednoduchou analógiou môže byť lekár, ktorému postupne telefonujú pacienti, ktorí sa chcú objednať. Nie je spôsob, ako by lekár vedel predpovedať, kto všetko ochorie a zavolá mu, aby

si dohodol termín. Pri dohodnutí termínu počas telefonátu môže vychádzať iba z termínov, ktoré si dohodol, a musí vybaviť pacienta na telefóne, kým sa mu môže dovolať ďalší. Predpokladáme navyše, že akonáhle je termín dohodnutý, nie je možné ho zmeniť. Algoritmus, ktorý náš myslený doktor používa, musí teda dať odpoveď pre každý z týchto blokov bez toho, aby mal k dispozícii inú informáciu ako to, aké termíny už dohodnuté má.

Online algoritmus teda môže pri výpočte odpovede na i-ty blok využívať iba informáciu z blokov $1, 2, \ldots, i$ a musí odpovedať, kým mu bude sprístupnený ďalší blok. Ak raz odpovie, nemôže už túto odpoveď zmeniť. Skúsme toto definovať formálnejšie pomocou request-answer modelu:[BEY98]

Definícia 1.1.1. *Majme danú vstupnú postupnosť $x = \langle x_1, x_2, \ldots, x_n \rangle$. Online algoritmus A počíta postupnosť výstupov $y = A(x) = \langle y_1, y_2, \ldots, y_n \rangle$, kde $y_i = f(x_1, x_2, \ldots, x_i)$. Cena riešenia je potom daná funkciou $C_A(x) = CENA(y)$.*

Ak by sme chceli hladať typickú ukážku problému, ktorý má zmysel definovať v online prostredí, je to problém stránkovania známy z oblasti správy pamäte.

Definícia 1.1.2. *Problém stránkovania: Máme danú množinu virtuálnych stránok pamäte $M = \{1, 2, \ldots, p\}$. Vstupom je postupnosť požiadaviek na stránky z M: $x = \langle x_1, x_2, \ldots, x_n \rangle$, kde $\forall i : x_i \in M$. Online algoritmus A má k dispozícii buffer $B = \{b_1, \ldots, b_k\}$, o veľkosti k - jeho veľkosť je algoritmu známa na začiatku behu. Pred spustením je buffer inicializovaný na $B = \{1, \ldots, k\}$. Ak v i-tom kroku $x_i \in B$, $y_i = 0$, inak musí A dať ako odpoveď $y_i (\in B)$, ktoré bude v bufferi nahradené x_i. Nový stav buffera teda bude $B = (B \setminus \{y_i\}) \cup x_i$. Cena riešenia je počet výpadkov stránok, teda $C_A(x) = |\{y_i : y_i \neq 0\}|$.*

Ako vidíme na príklade stránkovania, i v informatike sa nachádzajú problémy, ktoré je nutné prirodzene chápať práve v online prostredí. K týmto problémom patria hlavne sieťové algoritmy, napríklad routing. Medzi ďalšie bežne riešené online problémy patria napríklad k-server, scheduling, list update, alebo ski rental.

1.2 Kompetitívna analýza

Intuitívne nie je úplne zrejmé, ako hľadieť na zložitosť online problémov. Jednou z možností, ktorou sa začali informatici zaoberať, je porovnanie výsledku online algoritmu s optimálnym riešením algoritmu OPT poznajúceho celý vstup na začiatku výpočtu (teda de facto s riešením offline verzie problému). Je zrejmé, že online algoritmus tu takpovediac ťahá za kratší koniec - je znevýhodnený oproti algoritmu poznajúcemu celý vstup dopredu. Analýzou tejto zložitosti sa začali zaoberať Sleator a Tarjan [ST85] a túto zložitosť nazvali kompetitívny pomer[1]. Prirodzene nás bude zaujímať asymptotický pomer, pričom však budeme navyše rozlišovať, či je k dorovnaniu potrebná aditívna konštanta. V prípade, že nie je potrebná, budeme hovoriť o striktne kompetitívnom pomere.

Definícia 1.2.1. *Online algoritmus je c-kompetitívny, ak pre každú vstupnú postupnosť x, platí $\exists \alpha \geq 0 : C_A(x) \leq c \cdot C_{OPT}(x) + \alpha$. Ak je $\alpha = 0$ potom hovoríme, že algoritmus je striktne c-kompetitívny.*

Potom môžeme prirodzene zaviesť kompetitívnu zložitosť online problému ako najnižší dosiahnuteľný kompetitívny pomer. Hovorí nám to vlastne o najlepšom pomere k optimu, aký sme schopní dosiahnuť akýmkoľvek online

[1] competitive ratio

algoritmom, ktorý navrhneme. Intuitívne to vlastne hovorí o cene na optimálnosti riešenia, ktorú platíme za to, že nepoznáme celý vstup dopredu.

1.3 Poradná zložitosť

Natíska sa otázka, aký iný pohľad možno nastoliť na zložitosť online problémov. Jednou z možností ako pozerať na obtiažnosť online problému je pozrieť sa, koľko informácie mu chýba oproti offline verzii. Práve takýto pohľad nám ponúka poradná zložitosť, ktorá neskúma faktor, ktorým sa líšia riešenia online a offline verzie problému, ale meraním množstva informácie, ktoré odlišuje online a offline inštanciu. Teda množstvo informácie, ktoré musíme online algoritmu dodať na to, aby bol 1-kompetitívny[2].

Realizujeme to vo všeobecnosti pomocou modelu používajúceho orákulum O, ktoré bude mať k dispozícii celý vstup a bude môcť komunikovať s online algoritmom A. V našej práci budeme zväčša študovať iba také páry (A, O), ktoré riešia problém optimálne, a budeme hľadať takú dvojicu, ktorá využije najmenej informácie.

Predtým ako pristúpime k formalizácii nejakého modelu, skúsme sa pozrieť na poradnú zložitosť trochu intuitívne. Jedným z najjednoduchších online problémov je problém požičiavania lyží[3]. Môžeme sa na to pozrieť ako na dilemu človeka, ktorý by chcel vyskúšať lyžovanie - môže si buď kúpiť vlastné lyže, alebo si ich požičať. Otázne je, koľkokrát sa k lyžovaniu ešte dostane. Od toho závisí, či sa oplatí požičať si opakovane, alebo investovať do vlastných. Keby sme mali k dispozícii veštca, ktorý by videl celú budúcnosť, vedel by nám s istotou povedať ako minimalizovať naše náklady. Navyše by mu stačilo oznámiť nám iba jediný bit informácie.

[2]Respektíve na to, aby sme boli pre zvolené k aspoň k-kompetitívni.

[3]ski rental

Veštci však nie sú zadarmo - väčšinou platí čím viac informácie dodá, tým viac si naúčtuje. Vezmime si trochu ťažší problém, napríklad stránkovanie (definícia 1.1.2). Pri takýchto problémoch už nám nie je jedno, ako presne sa veštca pýtame, môžeme sa totiž dobre udrieť po vrecku. Mohli by sme chcieť v každom kroku poradiť číslo stránky, ktorú chceme vyhodiť, a takto zaplatiť v každom kroku za zhruba logaritmickú informáciu. Čo keby sme si povedali, že chceme vedieť iba 1 bit na každý krok. Stačí položiť otázku trochu inak. Nech si náš veštec zvolí konkrétne optimálne riešenie. Ďalej sa pýtame už iba, či ostane stránka, ktorú sme práve načítali, v tomto riešení v bufferi až do jej ďalšieho použitia? Takto máme vlastne priebežne upravované množiny stránok, ktoré môžeme vyhodiť a ktoré si chceme ponechať. Na konci sa nám podarí zrekonštruovať práve orákulom vybrané optimálne riešenie.

Otázkou ostáva, ako si nechať vyveštiť čo najlacnejšie. Toto presne nám zodpovedá poradná zložitosť. Pristúpme teda k prehľadu modelov, ktoré boli navrhnuté.

1.3.1 Prvý model

Pri prvom definovaní poradnej zložitosť boli zavedené hneď dva rôzne modely, ktoré sa odlišovali spôsobom komunikácie s orákulom. Oba tieto modely však mali isté nedostatky, ktoré okrem občasných technických problémov, ktoré bolo treba riešiť, posielali informáciu, ktorú sme potom nezahrnuli do analýzy. Oba tieto modely sa prvýkrát objavili v [DKP08] a podľa spôsobu komunikácie boli nazvané *helper* a *answerer*.

Helper mód

Orákulum pracujúce v helper móde pošle v i-tom kroku binárny reťazec a_i (môže byť aj prázdny), ktorý bude mať cenu $|a_i|$.

Definícia 1.3.1. *Majme daný online algoritmus A, vstupnú postupnosť $x = \langle x_1, x_2, \ldots, x_n \rangle$, a postupnosť rád $O(x) = \langle a_1, a_2, \ldots, a_n \rangle$ binárnych reťazcov a_i. Online algoritmus s helperom (A, O) vypočíta postupnosť výstupov $y = \langle y_1, y_2, \ldots, y_n \rangle$, kde $y_i = f(x_1, \ldots, x_i, a_1, \ldots, a_i)$. Cena riešenia je $C_{(A,O)}(x) = CENA(y)$ a poradná zložitosť je $B^H_{(A,O)}(x) = \sum_{i=1}^{n} |a_i|$*

V tomto móde je orákulum na začiatku každého kroku nútené odoslať správu s radou, ktorá však môže byť aj prázdny reťazec. Ako je možné poslať dodatočnú informáciu pomocou takéhoto systému? Keďže systém umožňuje prázdne správy, respektíve správy ľubovoľne dlhé, môžeme podať dodatočnú informáciu rozkúskovaním rady. Predpokladajme, že potrebujeme k neprázdnych rád na to, aby sme dosiahli 1-kompetitívny pomer. Označme tieto rady ako $a_1, a_1, \ldots, a_k$. Vidíme, že cena nijako nezohľadňuje počet správ, teda môžeme poslať všetko naraz, alebo to rozdeliť do k správ. V tomto rozdelení vieme zakódovať informáciu. Je zrejmé, že vieme preniesť informáciu rozlišujúcu toľko stavov, koľko je rôznych celočíselných partícií čísla k, teda približne $\frac{1}{4k\sqrt{3}} e^{\pi\sqrt{2k/3}}$.

Answerer mód

V tomto móde môže orákulum posielať informáciu iba v tom prípade, ak ho o ňu algoritmus priamo požiada. Táto rada však už musí byť neprázdny reťazec. Z technických dôvodov je ako rada použitá postupnosť neprázdnych reťazcov, ale použité sú iba tie, ktoré si algoritmus A vyžiada.

Definícia 1.3.2. *Majme daný online algoritmus A, vstupnú postupnosť $x = \langle x_1, x_2, \ldots, x_n \rangle$ a postupnosť odpovedí $O(x) = \langle a_1, a_2, \ldots, a_n \rangle$ neprázdnych binárnych reťazcov a_i. Online algoritmus s answererom (A, O) vypočíta výstupnú postupnosť $y = \langle y_1, y_2, \ldots, y_n \rangle$ nasledujúcim spôsobom:*

1. *v každom kroku i je na základe predchádzajúcich vstupov a rád vygenerovaný bit* $r_i = f(x_1, \ldots, x_i, r_1 * a_1, \ldots, r_{i-1} * a_{i-1})$ [4]

2. *následne sa vypočíta výstup* $y_i = f(x_1, \ldots, x_i, r_1 * a_1, \ldots, r_i * a_i)$

Cena riešenia je $C_{(A,O)}(x) = CENA(y)$ *a poradná zložitosť je* $B^A_{(A,O)}(x) = \sum_{i=1}^{n} |r_i * a_i|$

Keďže ani tento mód nekladie obmedzenie na veľkosť správy a posiela správy ako oddelené entity, je opäť možné rozpartíciovať správu do niekoľkých blokov, do ktorých veľkosti sme schopní zakódovať dodatočnú informáciu, ktorú tento model nemeria. Nemusí byť hneď zrejmé, že nestratíme nejaké bity tohto kódovania, na to, aby sme vedeli, koľkokrát treba radu pýtať. Na to nám stačí jednoduchý algoritmus: Vypýtame si radu, dostaneme k blokov, ktoré nám poradia ako postupovať v nasledujúcich k nejednoznačných miestach. Ak sa nám táto rada minie a objaví sa ďalšia nejasnosť, je zrejmé, že potrebujeme ďalšiu radu. Takto sa nám dostane presne toľko bitov rady ako v helper móde.

Je však zrejmé, že modely sa predsa len odlišujú - a to v inom kanáli, prostredníctvom ktorého možno odosielať informáciu, i keď v menšom množstve. Ak helper pošle na začiatku blok s radou na ďalších k blokov, znamená to, že si môže vybrať na odoslanie nasledujúcej správy ľubovoľný krok výpočtu z intervalu $2, \ldots, k+1$, pretože až $k+1$. blok potrebuje radu z ďalšej správy. Získavame tak ďalších $\log_2 k$ bitov informácie. Obdobný argument samozrejme platí nie len pre časový rozdiel medzi odoslaním prvých dvoch blokov, ale aj pre ľubovoľnú nasledujúcu dvojicu.

[4] operátor „$*$“ je definovaný ako $c * \alpha = \begin{cases} \epsilon & \text{ak } c = 0 \\ \alpha & \text{inak} \end{cases}$

Budeme teda skúmať pomocou týchto modelov, aké množstvo informácie musí algoritmus dostať na to, aby bol optimálny. Prirodzene nás bude zaujímať najmenšie množstvo informácie, s ktorým sme schopní optimalitu dosiahnuť. Aby sme boli schopní lepšie vystihnúť podstatu zložitosti problému, budeme túto zložitosť amortizovať na jeden blok výpočtu. Pre algoritmus A s orákulom O, je teda jeho zložitosť rovná najhoršej bitovej zložitosti, amortizovanej na jeden krok:

Definícia 1.3.3. *Majme online algoritmus A s orákulom O pracujúcim v móde $M \in \{H, A\}$[5]. Bitová zložitosť algoritmu je*

$$B^M_{(A,O)} = \limsup_{n\to\infty} \max_{|x|=n} \frac{B^M_{(A,O)}(x)}{n}$$

Poradná zložitosť online problému P je minimálna bitová zložitosť, ktorú vieme dosiahnuť niektorým párom (A, O):

Definícia 1.3.4. *Uvažujme problém P. Poradná zložitosť P v komunikačnom móde $M \in \{H, A\}$ je $B_M(P) = \min_{(A,O)} B^M_{(A,O)}$, kde minimum berieme cez všetky dvojice (A, O) také, že $\forall x : C_{(A,O)}(x) = C_{OPT}(x)$*

Vidíme, že modely sa od seba líšia, zaujíma nás však, či medzi nimi existuje istý vzťah, alebo dokonca ekvivalencia. Na túto otázku nám zodpovedal sám autor nasledujúcim tvrdením:

Tvrdenie 1.3.1. *Pre každý problém P platí $B_H(P) \leq B_A(p) \leq 0.92 + B_H(P)$*

Keďže algoritmus A je deterministický, závisí vlastne celý výpočet od správ od orákula O. Aby sme vedeli túto skutočnosť formalizovať, zadefinujeme si takzvaný *komunikačný pattern*, ktorý bude opisovať celú komunikáciu prebiehajúcu medzi orákulom a algoritmom, vrátane informácie posielanej v

[5] H a A reprezentujú helper a answerer mód

každom kroku. Keďže algoritmy sú deterministické, je zrejmé, že celý výpočet je daný komunikačným patternom a vstupom. Môžeme teda tvrdiť, že počet existujúcich komunikačných patternov znamená počet existujúcich možných výpočtov dvojice (A, O) na vstupe x.

Definícia 1.3.5. *Majme daný online algoritmus s helperom. Komunikačný pattern je definovaný ako postupnosť rád daných v každom kroku, čiže* $\langle a_1, a_2, \dots a_n \rangle$*, kde* a_i *je binárny reťazec (môže byť i prázdny).*

Lema 1.3.1. *Uvažujme online algoritmus s helperom, a nech vstupná postpnosť* x *má dĺžku* $|x| = n + 1$*. Pre* s *pevne danej veľkosi budeme uvažovať iba komunikačné patterny, v ktorých orákulum pošle v rámci* $n + 1$ *rád najviac* s *bitov. Pre číslo* X *- počet rôznych komunikačných patternov s touto vlastnosťou - platí* $X \leq s(\log 1 + \alpha + 1 + \frac{1}{\ln 2}) + \frac{1}{2}[\log 1 + \frac{1}{\alpha} + \log s] + c$*, kde* $\alpha = \frac{n}{s} > 1$ *a* c *je konštanta.*

Definícia 1.3.6. *Majme daný online algoritmus s anwererom. Pre každý jeho beh, počas ktorého sa opýta* q *otázok, je komunikačný pattern postupnosť* $\langle a_{i_1}, \dots, a_{i_q} \rangle$ *neprázdnych reťazcov - odpovedí, kde* i_j *je krok výpočtu, v ktorom bola položená* j*-ta otázka.*

Napriek tomu, že to nemusí byť intuitívne jasné, opäť môžeme tvrdiť, že výpočet je presne určený dvojicou vstupu a komunikačného patternu. Treba si uvedomiť, že odpovede orákula budú vždy rovnaké a keďže je algoritmus deterministický, položí otázky vždy v rovnakých krokoch výpočtu. Ak dôjde k miestu, kde sa rozhodne opýtať orákula, vždy dostane tú istú odpoveď a teda sa bude správať rovnako aj ďalej. Z toho nám vychádza, že výpočet pre dvojicu (A, O) na vstupe je deterministický.

Lema 1.3.2. *Uvažujme online algoritmus s answererom. Zoberme* q *a* $s \geq q$ *pevne danej veľkosi a uvažujme iba komunikačné patterny, v ktorých sa algo-*

ritmus opýta q otázok, ktoré spolu nie sú dlhšie ako s bitov. Potom existuje $X = \frac{1}{3}(2^{2s+1} + 1)$ rôznych komunikačných patternov s touto vlastnosťou.

Všimnime si, že predchádzajúci vzťah je nezávislý od počtu otázok q, čo je očakávané v súlade s vlastnosťou opísanou skôr.

1.3.2 Druhý model

Kvôli problémom s posielaním dodatočnej informácie prostredníctvom skrytých kanálov vzniknutých v dôsledku diskrétneho charakteru rád, ako aj kvôli zjednodušeniu formalizmu, sa pristúpilo k navrhnutiu nového modelu[BKK$^+$09], ktorý používa princíp bežne používaný pri Turingových strojoch. Úlohou orákula je teraz pripraviť pásku, ktorá obsahuje radu. Algoritmus A teraz môže pristupovať k páske s radou kedykoľvek rovnako ako k vstupu s tým, že na konci bude „spoplatnený“ podľa toho, koľko bitov z pásky s radou prečítal. Je zrejmé, že týmto boli odstránené problémy, ktoré vznikali pri predchádzajúcich dvoch modeloch. Rovnako je tu zrejmé ideové prepojenie so znáhodnenými algoritmami, ktoré používajú pásku s náhodnými bitmi.

Definícia 1.3.7. *Majme daný algoritmus A s radou, vstupnú postupnosť $x = \langle x_1, x_2, \ldots, x_n \rangle$ a pásku s radou Φ. Algoritmus A je c-kompetitívny s poradnou zložitosťou $s(n)$, ak existuje α taká, že $\forall n \forall x \ \exists \Phi$ také, že $C(A^{\Phi}(x)) \leq c \cdot C(OPT(x)) + \alpha$ a počas výpočtu neprečíta viac ako $s(n)$ bitov Φ. Ak $\alpha = 0$ potom hovoríme, že A je striktne c-kompetitívny.*

1.3.3 Tretí model

Tretí model pre poradnú zložitosť, ktorý sme našli, sa vyskytol v článku od [EFKR11] k MTS[6] a problému k-server. V zásade model, ktorý autori

[6] metrical task systems

používajú, dodáva algoritmu pri spracovaní každého dotazu ako radu prvok z definovaného priestoru rád $\mathcal{U}$.

Formálnejšie: Majme množinu $\mathcal{S}$ možných requestov, nech σ je konečná postupnosť požiadaviek z množiny $\mathcal{S}$. t-ty request budeme označovať $\sigma[t] \in \mathcal{S}$.

Deterministický algoritmus je postupnosť funkcií $g_t : S^t \to A_t, t \geq 1$, kde A_t je množina možných odpovedí na t-tu požiadavku.

Pre určitý konečný priestor $\mathcal{U}$, ktorý nazývame priestor rád, získava program dodatočnú informáciu pomocou požiadaviek $u_t : \mathcal{S}^* \to \mathcal{U}$. Poradná zložitosť je definovaná ako $\log_2 |\mathcal{U}|$. Pre jednoduchosť možno reprezentovať túto radu ako binárne reťazce dĺžky b.

Formálne je teda deterministický algoritmus s radou postupnosť párov $(g_t, u_t), t \geq 1$, kde $g_t : \mathcal{S}^t \times \mathcal{U}^t \to \mathcal{A}_t$ a $u_t : \mathcal{S}^* \to \mathcal{U}$. Na vstupnej postupnosti $\sigma = (\sigma[1], \ldots, \sigma[n])$ výpočet algoritmu je reprezentovaný funkciou

$$g_t(\sigma[1], \ldots, \sigma[t], u_1(\sigma), \ldots, u_t(\sigma))$$

.

Je to prvý model, ktorý sa snaží o to distribuovať radu rovnomerne počas výpočtu. Problém, keď máme dané obmedzenie šírky kanála s radou, je pomerne zaujímavý. Jediným problémom, ktorý z nášho pohľadu v tomto modeli nastáva, je neschopnosť detekovať sublineárnu zložitosť problému. Vieme, že na optimálne vyriešenie problému ski rental postačuje algoritmu jediný bit. V takto definovanom modeli však musí byť celková rada odovzdaná algoritmu aspoň lineárna - $|\mathcal{U}| = 2$ a každá z n rád použije preto jeden bit.

Preto si myslíme, že zaujímavé by bolo definovať podobný model, ktorý limituje počet bitov poslaných v rámci vyhodnocovania jednej požiadavky, v ktorom nás však zaujíma celková veľkosť odoslanej rady. K tejto myšlienke sa ešte vrátime v neskoršej kapitole.

1.4 Definície vybraných problémov

1.4.1 Online farbenie grafov

Farbenie grafov je v informatike veľmi dobre známy a často riešený problém, ktorý zahŕňa vo svojej všeobecnej verzii veľmi široké spektrum rôznych druhov problémov. Je známe, že všeobecne je tento problém NP-úplný. Z tohto dôvodu nebude veľkým prekvapením, až sa ukáže, že aj výsledky pre online farbenie grafov sú pomerne nelichotivé. Skutočne vo svojej práci[Hal92] Halldórsson ukazuje, že výsledky pre farbenie grafov bez znalosti vstupu sú veľmi slabé. Preto sa bude nutné pozerať na jednotlivé podtriedy tohto problému, kde je už možné nájsť zaujímavejšie výsledky. Väčšinou sa budeme zaoberať buď triedou grafov, ktoré sú v určitom tvare, alebo obmedzíme priestor vstupov na poradie vrcholov spĺňajúce isté požiadavky.

Definícia 1.4.1. *Majme graf $G = (V, E)$, kde $V = 1, 2, \ldots, n$. Tento graf je nášmu algoritmu odkrývaný v krokoch, pričom v i-tom kroku dostáva algoritmus $G_i = G[\{1, 2, \ldots, i\}]$, čiže graf indukovaný prvými i vrcholmi. Algoritmus musí pre vrchol i vrátiť farbu y_i tak, aby tieto farby tvorili na G vrcholové ofarbenie. Cena riešenia je počet farieb, ktoré na ofarbenie algoritmus potrebuje, teda $C_A(x) = |y_i : \forall i|$.*

Vrcholy grafu sú očíslované podľa poradia, v ktorom sa objavujú na vstupe. Treba poznamenať, že algoritmus na začiatku výpočtu nepozná hodnotu n - toto je zámerné, aby nezískaval o vstupe žiadne dodatočné informácie.

Keď spomenieme triedu $OnlineFarbenie(X, Y)$, budeme predpokladať, že graf bude z triedy X a poradie vrcholov na vstupe bude Y. Čo sa týka poradia vrcholov, obmedzuje sa zväčša na poradie vrcholov navštevovaných

pri niektorom prehľadávaní, a to zväčša do hĺbky, alebo do šírky. V silnejšej verzii nám postačí spojitosť indukovaného podgrafu v každom kroku výpočtu.

1.4.2 Najväčšia súvislá podpostupnosť

Problém hľadania najväčšej súvislej podpostupnosti je vo svojej offline podobe veľmi jednoducho riešiteľný problém. My sa zameráme na jeho online podobu, kde bude potrebné po odhalení každého prvku rozhodnúť, či patrí do správneho riešenia, alebo nie. Upozorňujeme čitateľa, že v nami definovanej variante problému sú prvky pre zjednodušenie vždy odhaľované postupne.

Definícia 1.4.2. *Problém najväčšej súvislej podpostupnosti: Máme danú postupnosť celých čísiel $A = (a_1, a_2, \ldots, a_n)$, snažíme sa nájsť takú dvojicu indexov i, j, ktorá maximalizuje $\sum_{k=i}^{j} a_k$. Vstup tvorí $x = \langle x_1, x_2, \ldots, x_n \rangle$, kde $x_k = a_k$. Algoritmus o každom prvku na vstupe musí rozhodnúť, či sa v intervale určenom indexami i a j nachádza, alebo nie.*

1.4.3 Alokovanie disjunktných ciest

Definícia 1.4.3. *Problém alokovania disjunktných ciest: Majme danú cestu $P = (v_1, v_2, \ldots, v_{L+1})$. Vstup je tvorený podcestami P danými dvojicou vrcholov (v_i, v_j), kde $1 \leq i < j \leq L+1$, pričom pre zjednodušenie je dĺžka cesty - L poslaná spolu s prvým krokom výpočtu. Teda $x = \langle (L, v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2}), \ldots, (v_{i_n}, v_{j_n}) \rangle$. Úlohou je vybrať čo najviac podciest tak, aby boli po dvojiciach hranovo disjunktné.*

Jednu vec, ktorú chceme ešte raz čitateľovi zdôrazniť, je, že v nami definovanej verzii problému algoritmus pozná veľkosť cesty, teda pozná hodnotu L - tá je zaslaná spolu s prvým vstupom.

1.5 Teória rekurzívnych funkcií

Teória vypočítateľnosti skúma rôzne výpočtové modely, ktoré boli postupom rokov sformalizované. Za jeden z pre ľudí prirodzenejších výpočtových modelov by sa dali pokladať funkcie. Otázky skonštruovateľnosti a vypočítateľnosti rôznych funkcií sú veľmi zaujímave a rôznorodé. Jednou z jednoduchších tried, ktoré stoja na vrchole hierarchie, by sa určite dala nazvať trieda primitívne rekurzívnych funkcií - $PREC$.

Je to trieda funkcií, ktorých zápis nám poskytuje priamočiary postup vyhodnocovania danej funkcie. Príklad môže byť napríklad jednoduchá funkcia umocňovania, ktorú možno interpretovať ako iterovanie funkcie násobenia, ktorú možno opäť rozložiť na niekoľko sčítaní.

Obsahuje niektoré základné funkcie ako succesor, nulu, alebo jednoduché funkcie na projekciu parametrov. Umožňuje nám z týchto základných funkcií získavať zložitejšie dvomi spôsobmi, a to skladaním funkcií a primitívnou rekurziou. Primitívna rekurzia má však takú vlastnosť, že ak ju vyhodnocujeme pre funkciu f, môže to vyžadovať i vyhodnotenie f pre iné parametre, prvý parameter nám však neustále klesá, a tak nedovolí funkcii opakovať sa príliš veľa krát.

Definícia 1.5.1. *Trieda $PREC$ primitívne rekurzívnych funkcií je definovaná nasledovne:*

1. *Nulárna funkcia z taká, že $z() = 0$, je v $PREC$.*
2. *Unárna funkcia s definovaná predpisom $\forall x : s(x) = x + 1$ je v $PREC$.*
3. *Pre každé $1 \leq k \leq n$ je v $PRECn$-árna funkcia P_k^n definovaná $\forall x_1, \ldots, x_n :$ $P_k^n(x_1, \ldots, x_n) = x_k$.*

4. *Ak $g \in PREC$ je k-árna funkcia a $f_1, \ldots, f_k \in PREC$ sú m-árne funkcie, potom m-árna funkcia $h(x_1, \ldots, x_m) = g(f_1(x1, \ldots, x_m), \ldots, f_k(x_1, \ldots x_m))$ je v $PREC$.*

5. *Ak $f \in PREC$ je k-árna funkcia a $g \in PREC$ je $(k+2)$-árna funkcia, potom $h \in PREC$, kde $h(0, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$ a $h(s(n), x_1, \ldots, x_k) = g(h(n, x_1, \ldots, x_k), n, x_1, \ldots, x_k)$.*

6. *V $PREC$ nie sú žiadne iné funkcie.*

Z definície nám vychádza, že $PREC$ je uzavretá na kompozíciu (bod 4) a primitívnu rekurziu (bod 5). Ekvivalentnou definíciou by bolo nahradiť bod 6 a povedať, že je to najmenšia množina uzavretá na 4 a 5, ktorá obsahuje všetky funkcie z bodov 1 až 3.

V [Zem] bolo ukázané, že primitívne rekurzívne funkcie intuitívne zodpovedajú programom, ktoré používajú iba cykly s ohraničeným počtom opakovaní - teda programom bez while cyklov a rekurzie.

1.5.1 Ackermannova funkcia

Chvíľu sa myslelo, že všetky jednoducho vypočítateľné funkcie sú primitívne rekurzívne, až kým sa v roku 1928 nepodarilo Wilhelmovi Ackermannovi zostrojiť funkciu, ktorá má jednoduchý predpis, podľa ktorého sa vyhodnocuje, napriek tomu však nie je primitívne rekurzívna. Stala sa tak jedným z prvých príkladov totálnej rekurzívnej funkcie, ktorá nie je primitívne rekurzívna[Ack28].

Definícia 1.5.2. *Ackermannova funkcia:*

$$A(m,n) = \begin{cases} n+1 & ak\ m = 0 \\ A(m-1,1) & ak\ m > 0\ a\ n = 0 \\ A(m-1, A(m,n-1)) & ak\ m > 0\ a\ n > 0 \end{cases}$$

Do akej triedy teda patrí Ackermannova funkcia? Je to rozšírenie triedy $PREC$, ktoré pridáva operáciu - minimalizáciu. Intuitívne môžeme chápať túto operáciu tak, že dostane funkciu s niekoľkými parametrami a kým všetky okrem jedného sú určené, hľadá minimálnu hodnotu posledného parametra, pri ktorej nadobúda nulu (pričom pre všetky menšie hodnoty musí byť kladná).

Definícia 1.5.3. *Trieda čiastočne rekurzívnych funkcií RE je definovaná nasledovne:*

1. *Nulárna funkcia z taká, že $z() = 0$, je v RE.*
2. *Unárna funkcia s definovaná predpisom $\forall x : s(x) = x + 1$ je v RE.*
3. *Pre každé $1 \leq k \leq n$ je v RE n-árna funkcia P_k^n definovaná $\forall x_1, \ldots, x_n : P_k^n(x_1, \ldots, x_n) = x_k$.*
4. *Ak $g \in RE$ je k-árna funkcia a $f_1, \ldots, f_k \in RE$ sú m-árne funkcie, potom m-árna funkcia $h(x_1, \ldots, x_m) = g(f_1(x1, \ldots, x_m), \ldots, f_k(x_1, \ldots x_m))$ je v RE.*
5. *Ak $f \in RE$ je k-árna funkcia a $g \in RE$ je $(k+2)$-árna funkcia, potom $h \in RE$, kde $h(0, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$ a $h(s(n), x_1, \ldots, x_k) = g(h(n, x_1, \ldots, x_k), n, x_1, \ldots, x_k)$.*
6. *Ak $g \in RE$ je $k+1$-árna funkcia potom $f(x_1, \ldots, x_k) \in RE$, kde $f(x_1, \ldots, x_k) = \mu(g(y, x_1, \ldots, x_k) = 0)$, pričom platí, že $f(x_1, \ldots, x_k) =$*

y práve vtedy, ak $\forall z < y : g(z, x_1, \ldots, x_k) > 0$ *a súčasne* $g(y, x_1, \ldots, x_n) = 0$.

7. *V RE nie sú žiadne iné funkcie.*

Funkcie z *RE* nemusia byť definované pre všetky vstupy. Preto zadefinujeme ešte jednu triedu.

Definícia 1.5.4. *Do triedy rekurzívnych funkcií REC patria funkcie z RE, ktoré sú definované na všetkých vstupoch.*

Takto definované rekurzívne funkcie už sú turingovsky úplné a zodpovedajú programom s while cyklami [Zem] a pre potreby našej práce sú dostatočne silné na to, aby sme sa už ďalšími konštrukciami nemuseli zaoberať.

Kapitola 2

Doterajšie známe výsledky

2.1 Farbenie grafov

Ako bolo uvedené skôr, farbenie grafov je príliš všeobecný problém na to, aby v ňom boli dosiahnuteľné zaujímavé výsledky. Tie však boli dosiahnuté v spojitosti s obmedzovaním sa na konkrétne triedy grafov, respektíve spôsoby prehľadávania.

2.1.1 Farbenie ciest

V tejto časti sa budeme zaoberať výsledkami dosiahnutými pri farbení ciest.

Väčšina výsledkov vo farbení ciest je dôsledkom jednoduchej úvahy, vyjadrenej v nasledujúcej leme.

Lema 2.1.1. *([FKS12]). Nech X je trieda bipartitných grafov. Teda grafov, ktoré sú 2-ofarbiteľné. Online algoritmus s radou pre $OnlineFarbenie(X, \cdot)$ nikdy nemusí pristupovať k páske s radou v prípade, že stupeň pridávaného vrchola v grafe G_k je nenulový.*

Ak sme vybavení touto znalosťou, stačí nám uvedomiť si, že na ceste

je obmedzený počet vrcholov, ktoré môžeme dostať s tým, že v k-tom kroku výpočtu, teda v grafe G_k budú izolované. Na základe tejto jednoduchej úvahy sa nám podarí veľmi presne odhadnúť zložitosť farbenia ciest.

Lema 2.1.2. *([FKS12]). Existuje deterministický online algoritmus, ktorý rieši*
$OnlineFarbenie(Cesta, \cdot)$ *s poradnou zložitosťou* $\lceil n/2 \rceil - 1$ *bitov.*

Lema 2.1.3. *([FKS12]). Akýkoľvek deterministický algoritmus riešiaci* $OnlineFarbenie(Cesta, \cdot)$ *potrebuje prečítať aspoň* $\lceil n/2 \rceil - 1$ *bitov z pásky s radou.*

Z týchto liem nám už triviálne vychádzajú nasledujúce tvrdenia:

Tvrdenie 2.1.1. *([FKS12]). Poradná zložitosť pre* $OnlineFarbenie(Cesta, \cdot)$ *je presne* $\lceil n/2 \rceil - 1$ *bitov.*

Tvrdenie 2.1.2. *([FKS12]). Poradná zložitosť pre* $OnlineFarbenie(Cesta, Spojite)$ *je nula.*

Výsledky tejto časti uzatvára zvláštne poradie vrcholov na vstupe, ktoré je v istom zmysle kombináciou dvoch predchádzajúcich. Označíme ho $2-$*prechod* a znamená, že cesta je prejdená dvakrát z jedného konca na druhý, a každý vrchol je spracovaný v jednom z týchto prechodov. Je zrejmé, že v druhom prechode už nepotrebujeme žiadnu radu, pretože graf už musí prichádzať spojite.

Tvrdenie 2.1.3. *([FKS12]). Poradná zložitosť pre* $OnlineFarbenie(Cesta, 2-$*prechod*$)$ *je* $\beta n + O(\log n)$*, kde* $\beta \approx 0.4057$ *je binárny logaritmus plastickej konštanty(jediný reálny koreň polynómu* $x^3 - x - 1$*).*

2.1.2 Farbenie kružníc

Kružnice sú tiež ešte dostatočne jednoduchou triedou grafov na to, aby nám poskytovali dobré výsledky Treba rozlišovať medzi kružnicami párnej a nepárnej dĺžky, pretože kružnice nepárnej dĺžky možno optimálne ofarbiť 3 farbami na rozdiel od kružníc nepárnej dĺžky, kde stačia farby 2. Toto spôsobí, že kružnice nepárnej dĺžky vieme farbiť bez použitia rady pomocou greedy prístupu.

Tvrdenie 2.1.4. *([FKS]). Existuje deterministický online algoritmus, ktorý rieši*
$OnlineFarbenie(ParneKruznice, \cdot)$ *bez použitia rady.*

Lema 2.1.4. *([FKS]). Existuje deterministický online algoritmus, ktorý rieši* $OnlineFarbenie(Kruznice, \cdot)$ *s poradnou zložitosťou* $\lfloor n/2 \rfloor - 1$.

Dôsledok 2.1.5. *Existuje deterministický online algoritmus, ktorý rieši* $OnlineFarbenie(NeparneKruznice, \cdot)$ *s poradnou zložitosťou* $n/2 - 1$

2.1.3 Farbenie pre všeobecné grafy

V tejto oblasti sa výsledky ukazujú ako nelichotivé. Vyzerá to, že množstvo rady potrebnej k dosiahnutiu optimálneho riešenia je priamo úmerné veľkosti zakódovania optimálneho riešenia pre jednotlivé typy grafov, resp. prehľadávaní.

Drobnou odbočkou, ktorú musíme v tomto momente spraviť, je výlet do kombinatoriky, aby sme boli schopní urobiť odhad na veľkosť kódovania optimálneho riešenia.

Bellove čísla udávajú počet rôznych rozdelení n prvkovej množiny na ľubovoľne veľa neprázdnych disjuntných podmnožín. Označme ich $B(n)$, podľa [DB70] platí nasledujúci vzťah: $lgB(n) = n \log_2 n - n \log_2 \log_2 n + O(n)$.

Stirlingove čísla druhého rádu udávajú počet rôznych rozdelení n prvkovej množiny do presne k neprázdnych disjunktných podmnožín. Označujme ich $S(n,k)$.

Lema 2.1.6. *Hodnota* $max_k(\log_2 S(n,k))$ *je asymptoticky rovná* $n\log_2 n - n\log_2\log_2 n + O(n)$. *Pričom* k, *pre ktoré je* $S(n,k)$ *maximálne je* $k = n/\ln n + O(1)$.

Dôkaz. Je zrejmé, že pre všetky kladné n platí $B(n) = \Sigma_{k=1}^{n} S(n,k)$. Teda existuje k také, že $S(n,k) \geq B(n)/n$ a teda prvá časť tvrdenia je pravdivá. Je ukázané, že pre dostatočne veľké n je k, pre ktoré nastáva maximum vo vzdialenosti menšej ako 1 od $e^r - 1$, kde $re^r = n$. Ďalej v[DB70] je ukázané, že $r = \ln n - \ln\ln n + O(\ln\ln n/\ln n)$. Z tohto je možné ľahko ukázať, že $k = n/\ln n + O(1)$. □

Teraz keď sme už vyzbrojení kombinatorickými odhadmi, môžeme prikročiť k samotným výrokom o farbení grafov. Ak sa pokúšame stanoviť jednoduchý horný odhad, budeme vlastne hladať spôsob ako poslať celé riešenie. Teda rozdelenie vrcholov do niekoľkých disjunktných množín, reprezentujúcich jednotlivé farby. Ak sa teraz zamyslíme, zistíme, že máme v podstate vyhrané - stačí nám poslať číslo veľkosti maximálne $B(n)$.

Tvrdenie 2.1.5. *([FKS]). Poradná zložitosť pre* $OnlineFarbenie(\cdot,\cdot)$ *je najviac* $n\log_2 n - n\log_2\log_2 n + O(n)$.

Tvrdenie 2.1.6. *([FKS]). Poradná zložitosť pre* $OnlineFarbenie(\cdot, BFS)$ *je aspoň* $n\log_2 n - n\log_2\log_2 n + O(n)$.

2.1.4 Pavúčie grafy

Špeciálna trieda grafov, v ktorej všetky vrcholy okrem jedného majú stupeň maximálne 2, sa nazýva pavúčie grafy.

Veta 2.1.7. *([FKS]). Existuje deterministický online algoritmus, ktorý rieši* $OnlineFarbenie(Pavucie, \cdot)$ *s radou veľkosti* $\log_2 \Phi \cdot n + 3\log_2 n + O(1)$.

Kde Φ je zlatý rez, teda $\Phi = \frac{1+\sqrt{5}}{2} \approx 0.69424$.

2.2 Alokovanie disjunktných ciest

V probléme alokovania disjunktných ciest môžeme analyzovať zložitosť podľa dvoch rôznych parametrov - n a L. Pre oba parametre sú známe niektoré ohraničenia.

Pozrime sa najprv na závislosť od n:

Veta 2.2.1. *([BKK+09]). Každý striktne c-kompetitívny algoritmus na alokovanie disjunktných ciest potrebuje prečítať aspoň* $(n+2)/(2c) - 2$ *bitov rady.*

Dôsledok 2.2.2. *Každý optimálny algoritmus pre alokovanie disjunktných ciest potrebuje prečítať aspoň* $n/2 - 1$ *bitov rady.*

Veta 2.2.3. *([BKK+09]). Pre každé* $c > 1$*, existuje c-kompetitívny algoritmus pre alokovanie disjunktných ciest, ktorý prečíta najviac*

$$\min\left\{n\log\left(\frac{c}{(c-1)^{(c-1)/c}}\right), \frac{n\log n}{c}\right\} + 3\log n + O(1)$$

bitov rady.

Nasledujúce výsledky sú známe ohraničenia v závislosti od L:

Veta 2.2.4. *([ea]). Na vyriešenie problému alokovania disjunktných ciest je potrebné presne* $L - 1$ *bitov rady.*

Veta 2.2.5. *([ea]). Pre všetky* $c \in \mathbb{N}^{>1}$ *existuje c-kompetitívny algoritmus, ktorý použije najviac* $\lceil\lceil 4L/(c-1)^2\rceil \log 3\rceil$ *bitov rady.*

Kapitola 3

Naše výsledky

V tejto kapitole prezentujeme naše vlastné výsledky v oblasti poradnej zložitosti.

3.1 Farbenie grafov

Oblasťou, ktorá má dostatočný záber a tým pádom priestor na množstvo výskumu je farbenie grafov. Najprv sa pozrieme na farbenie všeobecných grafov s poradím vrcholov zodpovedajúcim prehľadávaniu do hĺbky a potom sa pozrieme na farbenie niektorých špeciálnych tried grafov.

3.1.1 Farbenie všeobecných grafov pomocou DFS

Výsledky vo farbení všeobecných grafov nevyzerajú zatiaľ veľmi vábne a všetko naznačuje, že väčšina variantov tohto problému bude veľmi ťažká - ekvivalentná s odoslaním správneho riešenia prostredníctvom pásky s radou.

Lema 3.1.1. *Počet existujúcich rôznych ofarbení cesty dĺžky n pomocou $f+1$ farieb je $S(n, f)/(f + 1)$.*

Dôkaz. Označme si zaradom vrcholy cesty $p_1, \ldots, p_n$. Farbiť budeme množinou $0, \ldots, f$. Môžeme predpokladať, že máme postupnosť $a_1, \ldots, a_n$, pričom $a_i \in 1, \ldots, f$ (čo je ekvivalentné rozdeleniu n prvkov do f množín). Ukážeme teraz, ako transformovať takúto postupnosť na ofarbenie cesty $f+1$ farbami. Prvému vrcholu dáme farbu a_1. Následne každému ďalšiemu vrcholu p_k dáme farbu $(f(p_{k-1}) + a_k) \bmod (f+1)$.

Uistíme sa najprv, že takouto transformáciou dostaneme farbenie: toto je triviálne, pretože pričítame v grupe Z_{f+1} čísla najviac veľkosti f, teda žiadne dva za sebou idúce vrcholy na ceste nebudú mať rovnakú farbu.

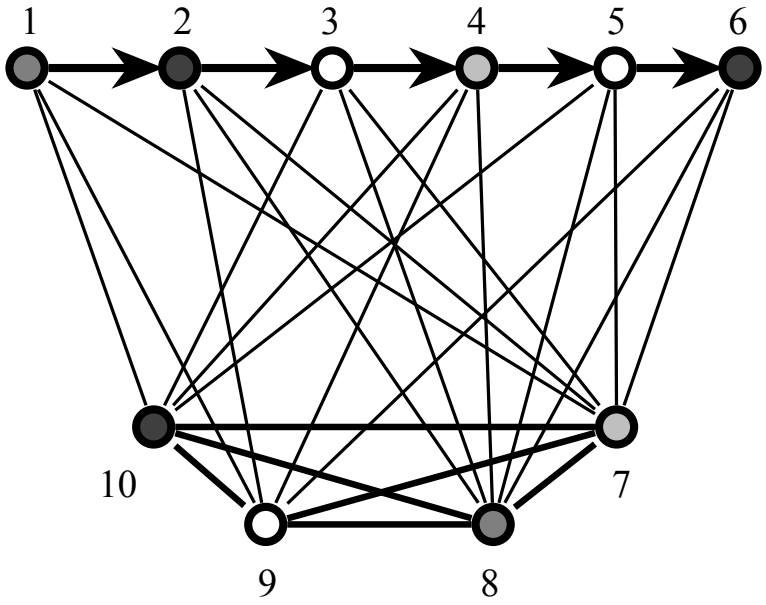
Ďalej sa musíme uistiť, že dve rôzne postupnosti nevygenerujú rovnaké farbenie. Sporom: Nech dve rôzne postupnosti a a b vygenerovali rovnaké farbenie: nájdime prvý prvok v ktorom sa líšia. Nech je to i. Ak je $i = 1$, potom sa $p_1^a = p_1^b$, ale to znamená, že aj $a_1 = b_1$ a teda máme spor s predpokladom. Nech je teda $i > 1$, pričom $p_{i-1}^a = p_{i-1}^b$ a $p_i^a = p_i^b$, z toho ale vyplýva, že $a_i = b_i$ a opäť máme spor. To teda znamená, že postupnosti a a b sú rovnaké. Teda naša transformácia na farbenie je injektívna.

Takýchto postpností existuje $f!S(n,f)$ - pre každé rozdelenie prvkov do množín môžeme vytvoriť $f!$ postupností, ak však berieme farbenie ako delenie do disjunktných množín, musíme vylúčiť postupnosti označujúce rovnaké farbenie. Takých postupností môže byť až $(f+1)!$ pre každé farbenie. Teda počet existujúcich ofarbení je

$$\frac{f!S(n,f)}{(f+1)!} = \frac{S(n,f)}{f+1}$$

□

Veta 3.1.2. *Poradná zložitosť pre* $OnlineFarbenie(\cdot, DFS)$ *je aspoň* $n \log_2 n - n \log_2 \log_2 n + O(n)$.



Obr. 3.1: Jedna z inštancií pre $n = 10$, $m = 6$, $k = 3$. Vrcholy 7 až 10 tvoria kliku.

Dôkaz. Pre zvolené n zostrojíme množinu M_n inštancií problému nasledovne: Nech $m(n) = n - \ln n$ a $k(n)$ je počet farieb (podmnožín), pre ktoré je $S(m(n), k(n))$ maximálne. Podľa lemy 2.1.6 máme $k(n) = m(n)/\log_2 m(n) + O(1)$, teda $m(n) + k(n) + 1 \leq n$ pre dostatočne veľké n. Pre prehľadnosť budeme ďalej v dôkaze uvádzať iba m a k, pričom budeme mať celý čas na mysli $m(n)$ a $k(n)$.

Teraz potrebujeme, aby množina M_n obsahovala dostatočne veľa inštancií. Každú pre jedno ofarbenie m vrcholov k farbami.

Každá inštancia má identický G_m, ktorý je tvorený cestou. Navyše vrcholy $m + 1, \ldots, m + k + 1$ tvoria v grafe kliku. Táto klika reprezentuje množinu farieb na ofarbenie grafu - zjavne každý jej vrchol má rôznu farbu. Teraz môžeme u prvých m vrcholov pomocou tejto kliky vytvoriť inštanciu s ľubovoľným ofarbením. Vrchol, ktorý má byť ofarbený určitou farbou, spojíme hranami s vrcholmi všetkých ostatných farieb nachádzajúcich sa v klike. Týmto

mu prakticky vynútime danú farbu - inak bude potrebných na ofarbenie viac farieb, ako je optimum pri farbení offline algoritmom. Príklad tohto vidíme na obrázku 3.1.

Z lemy3.1.1 vyplýva, že počet ofarbení prvých m vrcholov pomocou $k+1$ farieb je $S(m,k)/(k+1)$.

Počas farbenia prvých m vrcholov teda algoritmus nevie rozlíšiť, ktorú z inštancií má na vstupe, a môže pri ich rozlišovaní použiť iba radu, ktorú dostane. Potrebujeme teda $\log_2 S(n,k)/(k+1)$ bitov rady na rozlíšenie inštancie, ktorú máme na vstupe. Pomocou lemy 2.1.6 a faktu, že $\log_2(n - n/\ln n) = \log_2 n + O(1)$ dostávame:

$$\log_2 \frac{S(m,k)}{k+1} = \log_2 S(m,k) - \log_2 k + 1 =$$

$$= (\log_2 n - n/\ln n)\log_2 n - (\log_2 n - n/\ln n)\log_2\log_2 n + O(n) - O(\log_2 n) =$$

$$= n\log_2 n - n\log_2\log_2 n + O(n)$$

□

3.1.2 Grafy vnoriteľné do mriežky

Jednou z tried grafov sú grafy, ktoré je možné nakresliť do mriežky.

Definícia 3.1.1. *Graf $G = (V,E)$ voláme mriežka veľkosti $m \times n$, ak je v nasledujúcom tvare: $V = \{(a,b) \mid 1 \leq a \leq m \wedge 1 \leq b \leq n\}$ a medzi vrcholmi (a,b) a (c,d) je hrana práve vtedy, keď $((a = c+1) \wedge (b = d)) \vee ((a = c) \wedge (b = d+1))$. Mriežku veľkosti $m \times n$ budeme označovať $G_{m\times n}$.*

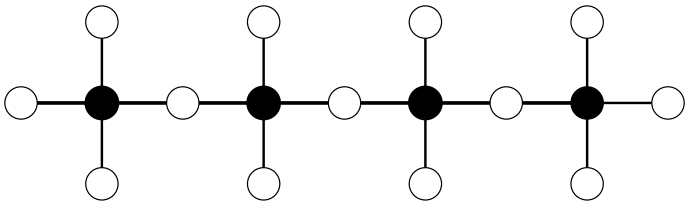
Definícia 3.1.2. *Povieme, že graf G je vnoriteľný do grafu H, ak existuje podgraf $H' \subseteq H$ taký, že $G \simeq H'$.*

Poznámka 3.1.1. Keďže mriežka je bipartitný graf, všetky grafy do nej vnoriteľné budú tiež bipartitné. Označíme ich G_{vn}. Označme G_{vns} spojité grafy z G_{vn}.

Dôsledok 3.1.3. *Keďže G_{vn} sú bipartitné, $OnlineFarbenie(G_{vn}, Spojite)$ sa dá riešiť optimálne deterministickým algoritmom bez rady.*

Všetky naše grafy budú bipartitné. Tým pádom aplikovaním lemy 2.1.1 môžeme prísť k záveru, že hlavnú úlohu v odhadoch bude hrať počet izolovaných vrcholov, ktoré vieme odkryť algoritmu pri online farbení. Keďže sú všetky grafy vnoriteľné do mriežky bipartitné, jednou z ciest ako garantovať izolovanosť vrcholov je odkrývať postupne všetky vrcholy jednej z partícií. Budú nás teda zaujímať najmä grafy, ktoré maximalizujú rozdiel veľkosti medzi partíciami.

Definícia 3.1.3. *Graf G nazývame chlpatým grafom stupňa k, ak je tvorený cestou $v_1, v_2, \ldots, v_{2k+1}$, na ktorej sú ku každému vrcholu s párnym indexom pripojené práve 2 listy. (Obr. 3.2)*



Obr. 3.2: Chlpatý graf stupňa 4.

Poznámka 3.1.2. Je zrejmé, že každý chlpatý graf je vnoriteľný do mriežky.

Lema 3.1.4. *Chlpatý graf stupňa k má maximálny pomer veľkosti partícií dosiahnuteľný pre G_{vns} s $4k + 1$ vrcholmi.*

Dôkaz. Označme partície grafu U a V, pričom U nech je partícia, ktorú chceme minimalizovať.

Ak $k = 0$: Graf je tvorený jediným vrcholom $v \in V$ - toto zjavne U minimalizuje, keďže veľkosť nemôže byť záporná.

Ak $k = 1$: Zvoľme si vrchol z U ako počiatočný - tento môže mať stupeň maximálne 4 (je vnoriteľný do mriežky pre ktorú $\Delta = 4$). Keďže chceme minimalizovať rozdiel medzi partíciami, všetky ostatné vrcholy budú zavesené na tomto vrchole ako listy a teda jeho stupeň bude 4. (Menšiu partíciu už v spojitom grafe zjavne nevieme dosahnuť)

Pre všeobecné k: zoberme graf, ktorý minimalizuje partíciu U pre $4k - 3$ vrcholov. Potrebujeme pripojiť ďalšie 4 vrcholy, ale $\forall u \in U\colon \deg(u) = 4$. Jediný spôsob ako pridať nový vrchol je napojiť ho na vrchol $v \in V$. Pridajme teraz vrchol u, a pripojme ho ku grafu hranou $\{u, v\}$. Zjavne $\deg(u) = 1$ a teda určite nič nepokazíme, ak zvyšné tri vrcholy zavesíme na u, a opäť platí, že $\forall u \in U\colon \deg(u) = 4$.

Je zjavné, že jedným z grafov (ktoré majú ekvivalentné veľkosti partícií U a V), ktoré môžeme získať takýmto postupom, je práve chlpatý graf stupňa k.

□

Veta 3.1.5. *Existuje algoritmus riešiaci* $Farbenie(G_{vns}, \cdot)$*, ktorý potrebuje najviac* $\frac{3}{4}n + O(1)$ *bitov rady.*

Dôkaz. Ako sme už spomenuli, vďaka leme 2.1.1 vieme, že radu potrebujeme iba na vrcholy, ktoré sa nám javia izolované pri ich spracovaní. V leme 3.1.4 sme ukázali, že najhorší prípad predstavujú chlpaté grafy. V nich dosahuje väčšia partícia veľkosť $\frac{3}{4}n$ - toto je teda maximálny počet izolovaných vrcholov, ktoré možno odhaliť počas výpočtu pre $n = 4k + 1$ pre $k \geq 0$.

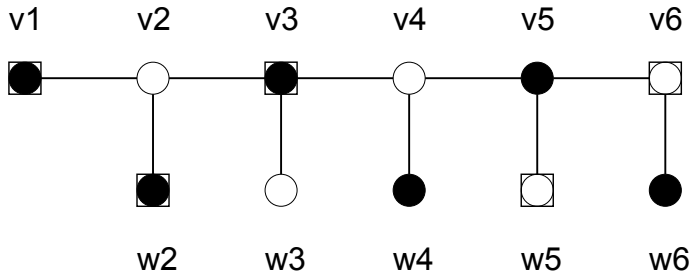
Ako sú na tom zvyšné n? Nemôže dojsť k drastickej zmene, pretože ak by existoval graf veľkosti $n = 4k + c$, kde $c = 2, 3, 4$ kde sa líši veľkosť

väčšej partície o viac ako konštantu - povedzme 5. potom by sme odtrhnutím jedného, dvoch alebo troch vrcholov z väčšej partície dostali spor s lemou 3.1.4 pre dané k.

Stačí nám teda použiť priamočiary postup, v ktorom ak je vrchol x_i izolovaný v grafe G_i, použijeme jeden bit rady na to, aby sme rozhodli, ktorú farbu mu priradíme. Toto nám stačí urobiť pre $i > 1$, pretože prvý vrchol môžeme ofarbiť ľubovoľne. □

Veta 3.1.6. *Algoritmus riešiaci* $Farbenie(G_{vn}, \cdot)$ *optimálne potrebuje aspoň* $\lceil n/2 \rceil - 2$ *bitov rady.*

Dôkaz. Našou úlohou je zostrojiť triedu vstupov, kde online algoritmus potrebuje použiť dostatočne veľa rady. Ako táto trieda vstupov nám poslúžia grafy nasledujúceho tvaru. Vezmime cestu dĺžky $\lceil n/2 \rceil$, označme jej vrcholy $v_1, \ldots, v_2$ a zvyšné vrcholy zavesme postupne od konca ako listy na vrcholy cesty, pričom vrchol zavesený na v_i označíme w_i.



Obr. 3.3: Ukážka grafu pre $n = 11$, $U_k = \{v_1, w_2, v_3\}$, $V_k = \{w_5, v_6\}$

Pre ľubovoľné k také, že $(1 \leq k \leq \lceil n/2 \rceil)$ zostrojme množiny U_k a V_k nasledovne (Obr. 3.3):

$$U_k = \{v_i \mid (i \bmod 2 = 1) \wedge (1 \leq i \leq k)\} \cup \{w_i \mid (i \bmod 2 = 0) \wedge (1 \leq i \leq k)\}$$

$$V_k = \{v_i \mid (i \bmod 2 = 0) \wedge (k+2 \leq i \leq \lceil n/2 \rceil)\} \cup \{w_i \mid (i \bmod 2 = 1) \wedge (k \leq i \leq \lceil n/2 \rceil)\}$$

Je zrejmé, že medzi žiadnymi dvoma vrcholmi z $U_k \cup V_k$ nevedie hrana. Navyše všetky vrcholy v U_k majú rovnakú farbu a všetky vrcholy vo V_k majú rovnakú farbu, inú ako vrcholy z U_k.

Nech I_k je množina inštancií vstupu, kde množiny U_k a V_k tvoria prefix vstupu, teda platí, že $\forall i\colon 1 \leq i \leq \lceil n/2 \rceil - 1 \Rightarrow x_i \in U_k \cup V_k$. Zjavne indukovaný graf $[x_1, \ldots, x_{\lceil n/2 \rceil - 1}]$ bude obsahovať iba izolované vrcholy.

Zostrojme množinu inštancií M nasledovne: Uvažujme reťazce tvaru $s \in 0\{0,1\}^{\lceil n/2 \rceil - 2}$. Pre každý reťazec s pridáme do M jednu inštanciu podľa nasledovného kľúča. Nech k je počet núl v reťazci. Vyberme ľubovoľnú inštanciu z I_k takú, že $\forall x_i \leq k\colon x_i \in P_x \Leftrightarrow s_i = 0$. (Vždy existuje aspoň jedna takáto inštancia, ak je ich viacero, na výbere nezáleží. Môžeme použiť napríklad lexikograficky najmenšiu)

Pretože takýchto reťazcov existuje $2^{\lceil n/2 \rceil - 2}$, je aj $|S| = 2^{\lceil n/2 \rceil - 2}$, pričom každá z inštancií potrebuje iné ofarbenie izolovaných vrcholov. Podľa Dirichletovho princípu ak existuje deterministický algoritmus, ktorý rieši problém optimálne a použije menej ako $\lceil n/2 \rceil - 2$ bitov rady, musia existovať aspoň dve inštancie v S, ktoré ofarbí rovnako, čím dostaneme neoptimálne riešenie. Tu máme spor a teda každý optimálny algoritmus musí použiť aspoň $\lceil n/2 \rceil - 2$ bitov rady. □

Poznámka 3.1.3. Keďže sme ukázali horný odhad iba pre spojité grafy, uvedieme aspoň triviálny odhad pre všeobecnú podobu: Existuje algoritmus, ktorý rieši $Farbenie(G_{vn}, \cdot)$ optimálne s radou najviac $n-1$. Dôkaz je priamočiara konštrukcia, kde si zakóduje na Φ farby prvých $n-1$ vrcholov, pričom o farbe posledného už vie rozhodnúť algoritmus sám.

3.2 Model pre riešenie online problémov s priebežnou radou

Ako sme už naznačili skôr, môžeme sa niekedy zaujímať o prípad, kedy chceme mať radu rovnomerne rozdistribuovanú medzi jednotlivé požiadavky. Takýto model nám poskytli Emek et al.[EFKR11]. Problém však je, že celá trieda problémov potrebujúcich menej ako lineárne veľa bitov sa nám zleje do jednej triedy požadujúcej práve lineárnu radu. Preto definujeme nový model, kde nás nezaujíma iba odmedzenie počtu bitov, ktoré možno prečítať v jednom kroku výpočtu, ale zároveň sledujeme aj celkový objem rady poskytnutej algoritmu.

Definícia 3.2.1. *Majme daný algoritmus A s radou, vstupnú postupnosť $x = \langle x_1, x_2, \ldots, x_n \rangle$ a pásku s radou Φ. Algoritmus A je c-kompetitívny s poradnou zložitosťou $(s(n), r(n))$ ak existuje α taká, že $\forall n \forall x\ \exists \Phi$ také, že $C(A^{\Phi}(x)) \leq c \cdot C(OPT(x))$ pričom počas výpočtu neprečíta viac ako $s(n)$ bitov Φ a pri spracúvaní x_i neprekročí hlava na Φ pozíciu $i \cdot r(n)$.*

Poznámka 3.2.1. Všimnime si, že model nijako nevynucuje, aby bola prečítaná v i-tom kroku páska s radou až po $i \cdot r(n)$. Program teda môže počkať niekoľko krokov a prečítať v nasledujúcom kroku celú radu za posledných niekoľko krokov. Toto nám však nevadí, pretože takýmto správaním program nemôže nič získať, a vždy existuje ekvivalentný algoritmus, ktorý radu čítal priebežne.

Poznámka 3.2.2. Ak je problém riešiteľný s radou $(O(f(n)), g(n))$ a $h(n)$ je funkcia, pre ktorú platí $\forall n : h(n) \geq g(n)$, potom je problém riešiteľný aj s radou $(O(f(n)), h(n))$.

Rovnako, ak je problém riešiteľný s radou $(\Omega(f(n)), g(n))$ a $h(n)$ je funkcia, pre ktorú platí $\forall n : h(n) \leq g(n)$, potom je problém riešiteľný aj s radou

$(\Omega(f(n)), h(n))$.

Otázkou je, či takýto model prináša niečo nové. Jednou z motivácií by mohlo byť študovať problémy, kde je použitým riešením poslať si celý vstup a riešiť problém offline algoritmom.

3.2.1 Problém najväčšej súvislej podpostupnosti

Je zrejmé, že pointou rady je odoslať informáciu o indexoch i, j. Riešením je teda páska s radou obsahujúca zakódované dve čísla, ktoré nám určia offsety dvoch indexov. Ako však ukážeme, problém môže nastať, ak obmedzíme počet bitov, ku ktorým možno pristúpiť v jednom kroku výpočtu. Ako príklad nám poslúžia nasledujúce vety.

Veta 3.2.1. *Problém maximálnej súvislej podpostupnosti sa dá vyriešiť optimálne*[1] *s radou* $(O(\log N), 2)$.

Dôkaz. Najprv sa pozrieme, akú informáciu ponesie poradná páska Φ. Ako si ukážeme neskor, s jej pomocou už problém jednoducho priamočiaro vyriešime. Potrebujeme na pásku umiestniť dve čísla o veľkosti $O(\log N)$. Keďže však máme užší kanál, nemôžeme ich tam dostať v jednom kroku, musíme teda zároveň odosielať aj informáciu o čiastkových výsledkoch momentálne spracúvaných vstupných blokov. Keďže môže program prečítať 2 bity rady za každú požiadavku, vyriešime problém tak, že vytvoríme defacto dve pásky:

- prvá (označme ju A) bude obsahovať dostatočne dlhý úsek z priamočiareho kódovania riešenia. Teda

$$A = \{0,1\}^* \text{ pričom } a_k = \begin{cases} 1 & \text{ak } i \leq k \leq j \\ 0 & \text{inak} \end{cases}$$

[1] Pripomíname, že slovom optimálne máme na mysli, že je program striktne 1-kompetitívny.

- druhá(označme ju B) bude obsahovať binárne zakódované indexy i a j. Potrebujeme vyriešiť oddeľovač týchto dvoch čísiel, ale bez ujmy na zložitosti môžeme použiť triviálne oddelenie, kde každý druhý bit na páske bude označovať, či predchádzajúci prečítaný bit bol posledný. Toto nám nafúkne pásku iba o konštantný násobok. Teda

$$B = i_1 0 i_2 0 \ldots i_{\lceil \log_2 i \rceil} 1 j_1 0 j_2 0 \ldots j_{\lceil \log_2 j \rceil} 10000 \ldots$$

Samotnú pásku s radou Φ vytvoríme striedavým zápisom týchto dvoch pások.

$$\Phi = a_1 b_1 a_2 b_2 \ldots$$

Ako bude pracovať náš online algoritmus A? V kažom kroku prečíta 2 bity z Φ, môžeme teda predpokladať, že v k-tom kroku má prístup k prvým k bitom A aj B. Je zrejmé, že na zodpovedanie k-tej požiadavky toto stačí, pretože riešenie je zakódované v A, teda vrátime a_k. Akonáhle však dočítame celú informáciu na páske B, poznáme interval, ktorý je riešením, a nepotrebujeme ďalej pozerať radu, pretože už vieme zodpovedať ľubovoľnú požiadavku sami. Keďže B obsahuje iba dve binárne kódované čísla, ktoré sú najviac N, potrebujeme z Φ prečítať iba $O(\log N)$ bitov. A poradná zložitosť je $(O(\log N, 2))$. □

Veta 3.2.2. *Problém maximálnej súvislej podpostupnosti sa dá vyriešiť optimálne s radou* $(\Omega(\log N), 2)$.

Dôkaz. Uvažujme inštancie v tvare $x = \langle a_1, -X, a_2, -X, \ldots, -X, a_n \rangle$, kde X je zvolené tak, aby platilo, že $\forall i : X > a_i$. Je zrejmé, že v takejto inštancii bude maximálna podpostupnosť tvorená jediným prvkom a to maximálnym prvkom. Zvoľme si k a nech a_k je maximálny prvok, navyše nech platí, že $a_1 \leq a_2 \leq \ldots \leq a_k \geq a_{k+1} \geq \ldots \geq a_n$. Všimnime si, že v každom kroku až do

odhalenia prvku a_{k+1} môže byť správnou odpoveďou ľubovoľný z aktuálneho a nasledujúcich prvkov.

Algoritmus teda potrebuje označiť jeden z n prvkov za správny, bez toho aby vedel nejakú užitočnú informáciu vyvodiť z už videných, resp. z práve odhaleného prvku. Na rozlíšenie týchto n prípadov potrebujeme $\log_2 n$ bitov na páske. Teda poradná zložitosť je $(\Omega(\log N), \log N)$.

Ako sme uviedli už v poznámke 3.2.2, ak máme dolný odhad, dodatočným zúžením kanálu s informáciou určite riešenie nezlepšíme. Dostávame teda z odhadu riešiteľnosti s radou $(\Omega(\log N), \log N)$ ako logický záver, že problém je riešiteľný aj s radou $(\Omega(\log N), 2)$. □

Dôsledok 3.2.3. *Problém maximálnej súvislej podpostupnosti sa dá vyriešiť optimálne s radou* $(\Theta(\log N), 2)$.

Prvotný odhad bol, že práve v probléme maximálnej súvislej podpostupnosti nájdeme s našim modelom hranicu, ktorá rozdeľuje logaritmickú a lineárnu radu. Pri bližšiom štúdiu sa nám však bohužiaľ podarilo dokázať aj nasledujúcu silnejšiu verziu vety:

Veta 3.2.4. *Problém maximálnej súvislej podpostupnosti sa dá vyriešiť optimálne s radou* $(\Theta(\log N), 1)$.

Dôkaz. Dolný odhad získame jednoducho z predchádzajúceho výsledku vety 3.2.2, keďže sme ukázali silnejšie tvrdenie. Jednoduchým dôsledkom je, že problém je riešiteľný s radou $(\Omega(\log N), 1)$.

Ako však zostrojíme algoritmus, ktorý použije iba 1 bit na krok? Pozrieme sa trochu bližšie na prípady, ktoré môžu nastať. Ak sa nachádzame v i-tom kroku výpočtu, možnosti sú nasledovné:

Ak $A(x)_{i-1} = 1$ - teda posledný spracúvaný prvok sa nachádzal v hľadanom intervale a zároveň:

- $x_i > 0$ - v tomto prípade nepotrebujeme radu, pretože určite do intervalu patrí aj x_i (pričítaním kladného čísla určite zväčšíme maximalizovanú sumu)
- $x_i < -\sum_{j=1}^{i} A(x)_j \cdot x_j$ - v tomto prípade tiež nepotrebujeme radu, pretože ak by interval presahoval až za x_i, vieme ho zlepšiť tým, že odtrhneme časť pred (resp. za) x_i vrátane
- $x_i = 0$ - v takomto prípade tiež nepotrebujeme radu, pretože interval môžeme natiahnuť o ľubovoľne dlhý úsek núl bez toho, aby sme si narušili výsledok
- $-\sum_{j=1}^{i} A(x)_j \cdot x_j \leq x_i < 0$ - toto je jediný prípad, v ktorom môžeme potrebovať radu, a to v prípade, že $x_{i-1} \geq 0$ - ak je x_{i-1} záporné, opäť radu nepotrebujeme, pretože vieme, že hľadané riešenie musí obsahovať celý aktuálny záporný úsek a kladnú časť za ním (inak by sme opäť jeho odtrhnutím získali lepšie riešenie)

Aby sme si to zhrnuli: jediný prípad, v ktorom potrebujeme radu, ak $A(x)_{i-1} = 1$ je ak $x_i < 0$[2].

Podobnú analýzu urobíme aj pre prípad, v ktorom platí $A(x)_{i-1} = 0$ a súčasne:

- $x_i \leq 0$ - radu nepotrebujeme, ak interval začneme v nasledujúcom kroku, budeme mať aspoň taký dobrý výsledok.
- $x_i > 0$ - ak bolo $x_{i-1} > 0$ radu nepotrebujeme, ak by sa totiž interval začínal na tomto prvku, rozšírením na predchádzajúci by sme dostali le-

[2]Dovolíme si kruto ignorovať prípad, keď je x_i príliš malé a ajtak radu vezmeme.

pšie riešenie. Radu teda potrebujeme iba vtedy, keď bol predchádzajúci prvok záporný.

Všimnime si teda, že iba v dvoch z uvedených prípadov potrebuje A prečítať radu, aby sa vedel optimálne rozhodnúť. Zamyslime sa teraz nad tým, ako často tieto prípady môžu nastať:

Najprv prípad, ak $(A(x)_{i-1} = 0) \wedge (x_i > 0) \wedge (x_{i-1} \leq 0)$. V prípade, že $A(x)_i = 0$, vieme s istotou, že $A(x)_{i+1} = 0$ a teda v nasledujúcom kroku radu nebudeme určite potrebovať. Ak je $A(x)_i = 1$, môže sa stať, že budeme potrebovať poradiť. Avšak tento prípad môže nastať iba raz počas celého behu algoritmu.

Ak nastal druhý prípad, kedy potrebujeme radu, teda $A(x)_{i-1} = 1 \wedge x_i < 0$, vieme rovnako povedať, že na výpočet $A(x)_{i+1}$ radu potrebovať nebudeme - ak sme interval ešte neuzavreli, berieme celý záporný úsek, resp. priberieme kladnú hodnotu, ak sme ho už uzavreli, výpočet prakticky skončil.

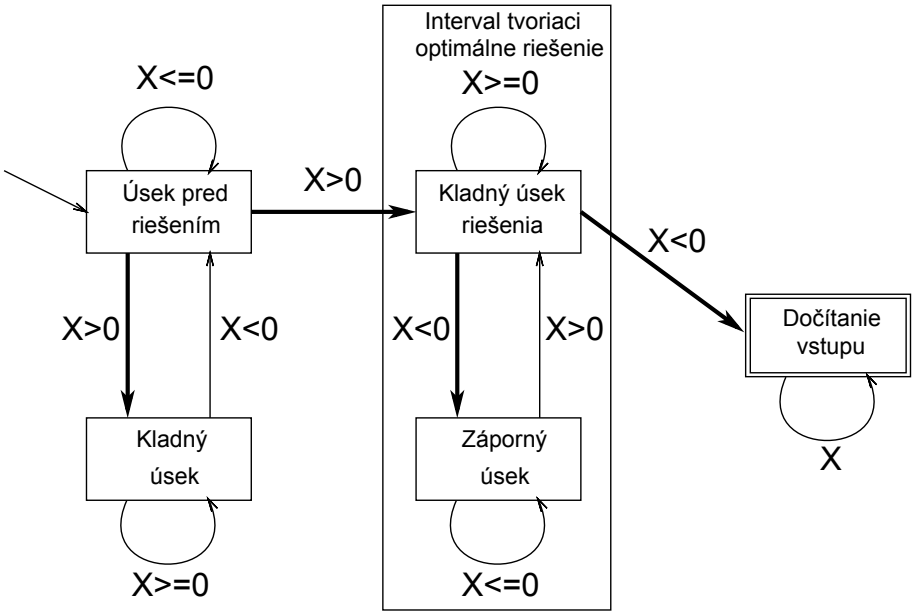
Celý priebeh čítania rady sa dá vyjadriť stavovým diagramom na obrázku 3.4.

Jasne sme ukázali, že po každom kroku, ktorý potrebuje radu, nasleduje aspoň jeden, ktorý radu nepotrebuje - existuje maximálne jedna výnimka k tomuto pravidlu. To znamená, že aspoň každý druhý bit máme voľný a môžeme teda postupne odkomunikovávať riešenie analogicky ako vo vete 3.2.1. Teda na optimálne riešenie nám stačí rada veľkosti $(O(logN), 1)$.

Keďže sme ukázali tesný horný aj dolný odhad, vyplýva z toho, že problém maximálnej súvislej postupnosti riešime s radou $(\Theta(\log N), 1)$. □

3.2.2 Alokovanie disjunktných ciest

Problém alokovania disjunktných ciest nám poskytol príklad, ktorý sme hľadali pre náš model. V ňom sa ukazuje, že množstvo rady potrebnej na opti-



Obr. 3.4: Priebeh spracovania vstupu algoritmom a jeho rozhodovanie. X v diagrame označuje x_i, teda aktuálne spracovávaný prvok. Hrubé šípky označujú hrany, ktoré potrebujú radu (musíme sa rozhodnúť, ktorou z nich opustíme stav).

málne riešenie problému môže kolísať v závislosti od šírky kanálu. Skutočne, pri obmedzení sa na jediný bit rady na každú požiadavku dochádza k zhoršeniu potrebnej zložitosti.

Veta 3.2.5. *Pre každé $c > 1$ stačí na optimálne vyriešenie problému alokovania disjunktných ciest rada $(\Theta(L), c)$.*

Dôkaz. Dolný odhad $L - 1$ môžeme prevziať z vety 2.2.4. Ostáva nám teda ukázať, že nám postačuje $(O(L), c)$ bitov rady na vyriešenie problému.

Úvodom dôkazu ešte poznámku o neceločíselnej funkcii $r(n)$ - je plne v súlade s našou definíciou, a nesmieme na páske prečítať viac ako $\lfloor i \cdot r(n) \rfloor$ bitov v i-tom kroku, teda niektoré časti vstupu môžu mať k dispozícii väčšie množstvo rady.

Nech k je celé číslo zvolené tak, aby platilo $k \cdot c \geq k + 1$. Teraz môžeme vyhlásiť, že na každých k častí vstupu máme k dispozícii radu veľkosti aspoň $k + 1$. Na vybavenie jedného vstupu nám postačuje jeden bit rady - jednoducho si odošleme informáciu o tom, či sa podcesta nachádza v optimálnom pokrytí alebo nie. Navyše každých k krokov získame jeden voľný bit rady.

Keďže podľa vety 2.2.4 sa dá tento problém vyriešiť optimálne s radou $L - 1$, stačí aby sme poslali nášmu algoritmu týchto $L - 1$ bitov a môžeme o všetkých ďalších vstupoch rozhodnúť bez rady. Celú túto radu sa nám určite podarí odkomunikovať v prvých $k \cdot (L-1)$ krokoch výpočtu, pričom použijeme celkovo $(k + 1) \cdot (L - 1)$ bitov rady. Keďže k je konštantné pre zvolené c, je celková rada $O(L)$. □

Veta 3.2.6. *Na optimálne vyriešenie problému alokovania disjunktných ciest je treba radu* $(\Theta(L^2), 1)$.

Dôkaz. Na začiatok ukážeme, že alokovanie disjunktných ciest sa dá riešiť optimálne s radou $(O(L^2), 1)$.

Stačí nám použiť jednoduché priamočiare kódovanie: vezmime si náhodné optimálne riešenie. Vezmime si vstupnú postupnosť $x = \langle x_1, \ldots, x_n \rangle$. Vytvorme z nej podpostupnosť $x' = \langle x'_1, \ldots, x'_k \rangle$ tak, že odstránime z x prvky x_i také, že platí $\exists j : x_j = x_i \wedge j < i$. Postupnosť x' má najviac $\frac{L^2+L}{2}$ prvkov, pretože toľko rôznych podciest existuje. $\Phi_i = 1$ práve vtedy, ak sa podcesta x'_i nachádza v zvolenom optimálnom riešení, inak $\Phi_i = 0$.

Teraz nám stačí jednoducho spracúvať požiadavky podľa nasledujúceho kľúča:

- Ak sme už videli podcestu, ktorá je momentálne na vstupe - zamietame ju (môžeme predpokladať, že ak podcesta je v optimálnom riešení, tak vždy príjmeme už jej prvý výskyt)

- Ak sme ju ešte nevideli - prečítame ďalší bit rady z Φ a podľa neho sa rozhodneme, pričom podcestu zvolíme do riešenia iba vtedy, ak bit na páske je 1.

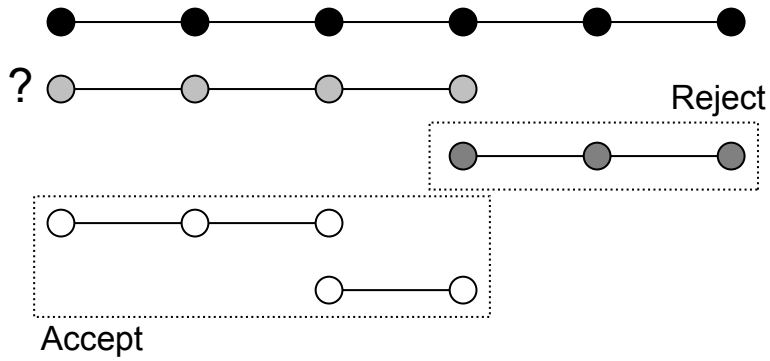
Keďže počet rôznych ciest je dostatočne malý - $\frac{L^2-L}{2}$ - použije tento postup $O(L^2)$ bitov rady.

Teraz ešte potrebujeme ukázať, že na vyriešenie alokovania disjunktných ciest potrebujeme $\Omega(L^2)$ bitov.

Jeden prístup, ako možno chápať online algoritmy s radou, je, že proti sebe hrajú dvaja hráči. Jeden hľadá najlepšie riešenie, zatiaľ čo jeho protivník posiela vstup, pričom má možnosť upravovať ostávajúcu časť vstupu. Rada nám pôsobí ako záväzok - ak protihráč prezradí nejakú informáciu, musí potom vstup skutočne zodpovedať tejto informácii. Ukážeme teraz stratégiu protihráča, ktorá si vynúti dostatočne veľa rady.

Úvodom si musíme uvedomiť, ako je vnútorne závislý vstup. Samozrejme platí, že ak sme už nejakú cestu do riešenia vybrali, automaticky môžeme odmietnuť bez rady všetky cesty, ktoré ju pretínajú. Navyše ak máme dve podcesty $p_1 = (u_1, v_1)$ a $p_2 = (u_2, v_2)$ a platí, že $u_1 \geq u_2 \wedge v_1 \leq v_2$, vieme, že pre každé riešenie, ktoré využíva p_2, existuje aspoň také dobré, ktoré používa p_1 namiesto p_2. Navyše, ak by sme zobrali radu na p_1 vedeli by sme už p_2 rozhodnúť bez rady (pri vhodne volenej stratégii). Ak však zistíme, že dlhšia cesta nie je v riešení, informáciu o tom, ktorá jej podcesta v riešení je, nám to nedáva.

Na vstupe teda budeme určite dávať podcesty od najdlhšej po najkratšiu. Zoberieme si postupnosť všetkých podciest utriedenú zostupne podľa dĺžky a podľa najľavejšieho vrcholu. Tieto postupne dávame algoritmu, ten má iba 3 možnosti (nech je na vstupe podcesta $p_i = (u, v)$), tieto ilustruje obrázok



Obr. 3.5: Pokrývame čiernu cestu disjunktnými podcestami. Ak svetlošedú cestu rozhodneme bez rady, môže súper upraviť zvyšok vstupu tak, že bude obsahovať iba cesty svetlého/tmavého typu a nezískame optimálne riešenie. Pri vypýtaní si rady cestu nepríjmame a zvyšok vstupu môže vyzerať opäť ľubovoľne.

3.5 :

1. Algoritmus sa rozhodne nepoužiť cestu p_i - v tomto prípade môžeme upraviť zvyšok vstupu tak, aby p_i bola v každom optimálnom riešení použitá: ďaĺej na vstupe už bude nasledovať iba niekoľkokrát ľubovoľná cesta nepretínajúca p_i.

2. Algoritmus sa rozhodne použiť cestu p_i - opäť upravíme zvyšok vstupu, a to tak, že na vstupe sa vyskytnú dve disjunktné podcesty p_j a p_k, ktoré sa budú pretínať s p_i.

3. Algoritmus si vypýta radu - keďže máme možnosť prečítať iba 1 bit, musíme sa opýtať, či sa p_i v riešení nachádza[3], alebo nie. V tomto kroku sme schopní získať dodatočnú informáciu iba v prípade, ak p_i patrí do

[3]Keďže zvyšok vstupu už tvoria iba kratšie cesty, sú od tejto odpovedi nezávislé (odpoveď bude 0) a neexistuje teda vlastnosť, ktorá by nám povedala niečo o ďalšom vstupe a zároveň zodpovedala či danú cestu vybrať do riešenia.

optimálneho riešenia, inak žiadnu informáciu naviac nezískame.

Vidíme, že je teda nutné, aby si algoritmus v každom kroku vypýtal bit rady. Aby sme mu neposkytli žiadnu dodatočnú informáciu, žiadna z dotazovaných podciest sa v riešení nachádzať nebude. Až sa dostaneme na cesty dĺžky 1, algoritmus sa už vie rozhodnúť aj bez rady. Avšak v tomto momente sme už spotrebovali $\frac{L^2-3L}{2} - 1$, a teda potrebujeme $\Omega(L^2)$ bitov rady. □

3.3 Primitívne rekurzívne funkcie s radou

Natíska sa otázka, ako by mohla vyzerať trieda primitívne rekurzívnych funkcií rozšírených o radu. Nepodarilo sa nám nájsť žiadny výskum v tejto oblasti. Pristúpime teda k zadefinovaniu tejto triedy sami.

Zjavne nemá zmysel obmedzovať poradnú funkciu na triedu $PREC$, vďaka uzavretosti na kompozíciu by to znamenalo, že presná kópia tejto funkcie aj s radou sa už nachádza v $PREC$. Zaujímavou alternatívou sa ukazuje použiť funkciu rekurzívnu. Náš model primitívne rekurzívnych funkcii s radou bude teda používať rekurzívnu poradnú funkciu, ktorá bude mať o jeden parameter viac, ako funkcia, ktorú počítame. Pričom tento dodatočný parameter bude udávať poradové číslo bitu rady, ktorý má táto funkcia vrátiť. Ak by sme dovolili vracať poradnej funkcii hodnotu priamo, mohla by táto defacto nahradiť minimalizáciu a zosilnila by našu triedu na rekurzívne funkcie.

Definícia 3.3.1. $A-PREC = \bigcup_{a \in REC} M_a$, *kde* M_a *je množina spĺňajúca nasledujúce podmienky:* [4]

1. *Funkcia* a *je v* M_a.

2. *Nulárna funkcia* z *taká, že* $z() = 0$, *je v* M_a.

[4] Podmienky $2-6$ sú ekvivalentné s podmienkami $3-7$ z definície 1.5.1

3. *Unárna funkcia s definovaná predpisom $\forall x : s(x) = x + 1$ je v M_a.*

4. *Pre každé $1 \leq k \leq n$ je v M_a n-árna funkcia P_k^n definovaná $\forall x_1, \ldots, x_n :$ $P_k^n(x_1, \ldots, x_n) = x_k$.*

5. *Ak $g \in M_a$ je k-árna funkcia a $f_1, \ldots, f_k \in M_a$ sú m-árne funkcie, potom m-árna funkcia $h(x_1, \ldots, x_m) = g(f_1(x1, \ldots, x_m), \ldots, f_k(x_1, \ldots x_m))$ je v M_a.*

6. *Ak $f \in M_a$ je k-árna funkcia a $g \in M_a$ je $(k+2)$-árna funkcia, potom $h \in M_a$, kde $h(0, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$ a $h(s(n), x_1, \ldots, x_k) = g(h(n, x_1, \ldots, x_k), n, x_1, \ldots, x_k)$.*

7. *V M_a nie sú žiadne iné funkcie.*

Poznámka 3.3.1. Z definície triviálne vyplýva, že $A-PREC \subseteq REC$. Rekurzívne funkcie obsahujú všetky funkcie z $A-PREC$ vrátane poradných funkcií a sú uzavreté na nadmnožinu vlastností $A-PREC$.

Popíšme si trochu motiváciu za našou definíciou. Vďaka tomu, že čiastočne kopírujeme definíciu primitívne rekurzívnych funkcii, zabezpečíme si, že každá primitívne rekurzívna funkcia v $A - PREC$ je. Nasleduje časť, ktorú musíme vykonať pre každú možnú poradnú funkciu: vložíme ju do množiny a urobíme uzáver na kompozíciu a primitívnu rekurziu - toto v zásade znamená, že umožníme funkciám aby používali hodnoty poradnej funkcie vo svojom predpise. Keď urobíme zjednotenie všetkých takýchto množín, dostaneme vlastne všetky existujúce funkcie z $PREC$, ktoré majú navyše možnosť využívať ľubovoľnú poradnú funkciu (ale vždy v celej funkcii maximálne jednu).

Jednou z vecí, ktoré nám uľahčia prácu s primitívne rekurzívnymi funkciami, je kódovanie do čísel. Do jednej premennej môžeme zakódovať ľu-

bovoľne veľa čísel napríklad v tvare $(a, b, c) = 2^a 3^b 5^c$. Navyše zápis každej primitívnej rekurzie vieme zakódovať do čisel. Z každej primitívne rekurzívnej funkcie vieme dostať jedno číslo, ktoré jej zodpovedá. Navyše môžeme aj všetky argumenty funkcie zakódovať do jediného. Funkciu, ktorá zodpovedá zápisu čísla n, budeme označovať f_n. Ak n nekóduje validnú funkciu, potom $f_n \equiv 0$.

Definícia 3.3.2. *Ak M je množina funkcií, potom $U_M(n, x)$ nazveme univerzálnou funkciou pre množinu M, ak platí: $f_n \in M \Rightarrow U_M(n, x) = f_n(x)$.*

Veta 3.3.1. *Nech $g \in PREC$. Vezmime $M = \{f \mid f \in REC \wedge \forall x\colon \log_2 f(x) \leq g(x)\}$. Potom $U_M \in A-PREC$.*

Dôkaz. Naša funkcia bude mať nasledujúci tvar:

$$\begin{aligned} U_M(n, x) &= u(g(x)) \\ u(0) &= a(n, x, 1) \\ u(y+1) &= u(y) + 2^{y+1} \cdot a(n, x, y+2) \end{aligned}$$

V podstate si iba zostrojí horné ohraničenie vstupu a vypýta si prostredníctvom rady výsledok. Ten sa vďaka predpokladu zmestí do $g(x)$.

Poradná funkcia a bude zvolená tak, aby jej hodnoty spĺňali nasledujúcu rovnosť (vracali y-ty bit $f_n(x)$):

$$a(n, x, y) = (f_n(x)/2^{y-1}) \bmod 2$$

Je zrejmé, že $a \in REC$, pretože $f_n \in REC$. Škaredá a technická je časť, kde treba zostrojiť a simulovať f_n, a týmto detailom sa v tomto dôkaze venovať nebudeme. Je však zrejmé, že to zvládneme, pretože je algoritmicky vypočítateľná. □

Z predchádzajúcej vety je jasné, že akonáhle vieme funkcii ohraničiť veľkosť výstupu primitívne rekurzívnou funkciou, je táto funkcia v $A-PREC$.

Veta 3.3.2. $PREC \subsetneq A-PREC$

Dôkaz. Nech $\Phi(n,x) = U_{PREC}(n,x)$, teda univerzálna funkcia pre unárne primitívne rekurzívne funkcie.

Zjavne $\Phi(n,x)$ nie je v $PREC$. Ak by bola, potom $g(n) = \Phi(n,n) + 1$ je tiež v $PREC$, ale potom $\exists y\colon f_y \equiv g$ a $g(y) = \Phi(y,y) + 1 = f_y(y) + 1 = g(y) + 1$.

Vezmime funkciu $f(n,x) = \operatorname{sgn}(\Phi(n,x))$. Zjavne[5] $f \notin PREC$. Zároveň však podľa vety 3.3.1 platí $f \in A-PREC$. □

Našu prácu ukončíme hypotézou, ktorú sa nám zatiaľ nepodarilo formálne dokázať, i keď sa nazdávame, že máme opodstatnené argumenty, ktoré preukazujú jej platnosť.

Hypotéza 3.3.1. $A-PREC \subsetneq REC$

Ako príklad nám poslúži Ackermannova funkcia. Hlavným argumentom je, že ani s 1 bitovou radou nie sme schopní primitívne rekurzívnymi funkciami vygenerovať dostatočne veľké číslo, ktoré by sme mohli vrátiť ako výsledok funkcie.

Jedným spôsobom, ktorým sa pravdepodobne dá dôjsť k tomuto výsledku, je štrukturálna indukcia na funkciu, kde ukážeme istý horný limit veľkosti čísla, ktoré je schopné daná časť funkcie vytvoriť. Predpokladáme, že takýmto postupom by sme ukázali, že sme schopní ohraničiť zhora veľkosť výstupu primitívne rekurzívnou funkciou.

[5]Rozhodujeme či zápis funkcie je prim. rekurzívny, potom aj $g \equiv \neg\operatorname{sgn}(f)$ je v $PREC$ a diagonalizáciou už ľahko ukážeme, že toto nemôže byť.

Ďalší spôsob, ktorým by sa dalo postupovať, je, že by sa nám podarilo ukázať, že vieme k funkcii $f \in A - PREC$ vyrobiť ekvivalentnú funkciu z $PREC$ skúšaním všetkých možností - v každom bode, kde sa berie rada by sa vyskúšali obe možnosti. Ak by sme dostali v niektorej vetve skutočne hľadanú hodnotu, vieme ju už jednoducho overiť. Toto by síce znamenalo exponenciálne spomalenie, to nám však v tomto prípade príslušnosť do tried neovplyvní. Takto by sme dostali spor s tým, že Ackermannova funkcia nie je primitívne rekurzívna.

Technicky sa však tieto dôkazy ukázali príliš komplikované a zatiaľ sa nám ich nepodarilo zostrojiť.

Záver

V tejto práci sme sa venovali niekoľkým oblastiam, do ktorých zasahuje poradná zložitosť. Jednalo sa hlavne o analýzu online problémov.

V problematike farbenia grafov sme sa zaoberali niektorými triedami grafov a poradiami ich prezentovania na vstupe. Ukázali sme odhad pre všeobecné grafy prehľadávané do hĺbky a dolné a horné ohraničenie pre grafy vnoriteľné do mriežky.

Zadefinovali sme nový model, ktorý núti algoritmus čítať radu postupne, a venovali sme sa štúdiu dvoch problémov v tomto novom modeli - hľadanie najväčšej súvislej podpostupnosti a alokovanie disjunktných ciest. Pre oba sa nám podarilo ukázať tesné odhady množstva rady potrebného na ich riešenie. Navyše na probléme alokovania disjunktných ciest sa nám podarilo preukázať opodstatnenosť nášho modelu, keďže celkový objem rady dosahoval asymptoticky rozdielne hodnoty pre rôznu rýchlosť jej sprístupňovania.

Na záver sme opustili online prostredie a venovali sa využitiu rady v teórii rekurzívnych funkcií. Zadefinovali sme triedu primitívne rekurzívnych funkcií s radou a ukázali, že takto definovaná trieda je silnejšia ako primitívne rekurzívne funkcie.

Nadviazať na náš výskum sa dá mnohými smermi. Jednak je tu možnosť analyzovať rôzne online problémy s pomocou nášho modelu využívajúceho priebežnú radu. Prípadne je tu priestor na dokázanie, respektíve vyvrátenie

našej hypotéze o vzťahu primitívne rekurzívnych funkcií s radou a rekurzívnych funkcií.

Literatúra

[Ack28] Wilhelm Ackermann. Zum hilbertschen aufbau der reellen zahlen. *Mathematische Annalen*, 99:118–133, 1928.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online computation and compatitive analysis*. Cabridge University Press, New York, NY, USA, 1998.

[BKK+09] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. On the advice complexity of online problems. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *ISAAC*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer, 2009.

[DB70] N.G. De Bruijn. *Asymptotic Methods in Analysis*. Dover Books on Mathematics. Dover Publications, 1970.

[DKP08] Stefan Dobrev, Rastislav Kralovic, and Dana Pardubská. How much information about the future is needed? In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat, and Mária Bieliková, editors, *SOFSEM*, volume 4910 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2008.

[ea] Hans-Joachim Böckenhauer et al. On the Advice Complexity of the Disjoint Path Allocation Problem. Personal communication, to appear.

[EFKR11] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, May 2011.

[FKS] Michal Forišek, Lucia Keller, and Monika Steinová. Advice Complexity of Online Graph Coloring. Personal communication, to appear. Paper submitted to Information and Computation in 2012.

[FKS12] Michal Forisek, Lucia Keller, and Monika Steinová. Advice complexity of online coloring for paths. In Adrian Horia Dediu and Carlos Martín-Vide, editors, *LATA*, volume 7183 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2012.

[Hal92] Magnús M. Halldórsson. Parallel and on-line graph coloring algorithms. In Toshihide Ibaraki, Yasuyoshi Inagaki, Kazuo Iwama, Takao Nishizeki, and Masafumi Yamashita, editors, *ISAAC*, volume 650 of *Lecture Notes in Computer Science*, pages 61–70. Springer, 1992.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[Zem] Marek Zeman. Súvis rekurzívnych funkcií a programovacích jazykov. Fakulta Matematiky, Fyziky a Informatiky, Univerzita Komenského. 2008.