PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Implementace logické hry kakuro

2010 Tomáš Slíž

Anotace

Bakalářská práce se zabývá japonskou logickou hrou kakuro. Zkoumá náhodné generování a řešení rébusů, možnosti jeho optimalizace a zrychlení. Zároveň také srovnává různé strategie a přístupy potřebné k vytvoření kvalitního algoritmu pro generování a řešení rébusů v reálném čase.

Děkuji vedoucímu bakalářské práce RNDr. Arnoštu Večerkovi za odbornou pomoc, rady a připomínky při realizaci této bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Cílem této práce je vytvořit program, který umožní generovat a řešit logické rébusy kakuro v reálném čase a zároveň poskytne příjemné a intuitivní uživatelské rozhraní. Součástí aplikace tedy je jak generátor nových her kakuro, tak i jejich řešitel. V části generátoru nových kakuro je důraz kladen na rychlost a efektivnost procesu generování a řešení her kakuro. Herní prostředí implementuje funkce vytvoření nové hry o zadané velikosti a obtížnosti, umožňuje rozehranou hru uložit, případně načíst, součástí je i uživatelský editor, který uživateli umožní vytváření vlastních herních desek. Samotné uživatelské rozhraní prošlo důkladným testováním použitelnosti, jehož cílem je vytvoření ergonomického uživatelského rozhraní, přívětivého k uživatelům.

Požadavky na práci:

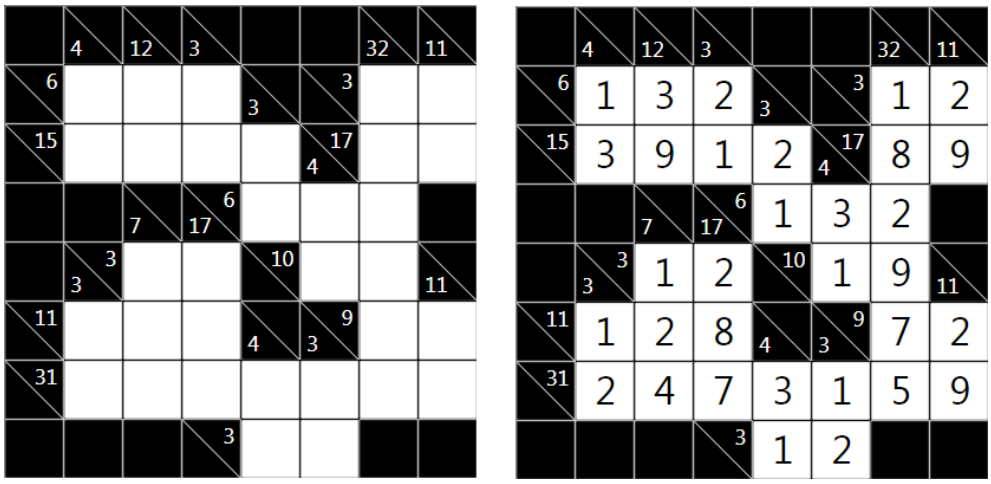
- umožnit uživateli vybrat si a luštit křížovku různého stupně obtížnosti
- kontrola správnosti vyplnění křížovky
- automatické vyřešení křížovky
- možnost uložení a následné načtení rozehrané hry
- vytváření uživatelských křížovek pomocí přívětivého grafického rozhraní a možnost jejich uložení
- zobrazovat celkový čas luštění křížovky
- tisk křížovek
- nápověda s pravidly hry a popisem všech funkcí aplikace
- programátorská dokumentace
- uživatelská dokumentace

2. Teoretická část

V posledních letech se po celém světě zvedla vlna zájmů o japonské logické rébusy. Roku 2004 vyšlo v londýnských novinách The Times několik hlavolamů sudoku. Okamžitě si získali oblibu mnoha čtenářů a během velmi krátké doby se začaly objevovat po celém světě. Po opadnutí prvotní mánie se zájem obrátil i k dalším japonským logickým hrám. Postupně se objevilo kakuro, hitori, nurikabe a další, popularitě sudoku se však přiblížilo pouze kakuro.

2.1. Kakuro

Kakuro, v anglicky mluvících zemích také známé jako cross sums, je momentálně po sudoku druhým nejpopulárnějším japonským logickým rébusem. Jedná se o kombinaci klasické křížovky a sudoku. Herní plán je velmi podobný tomu, s jakým se lze setkat u křížovky, jsou na něm vyplněná a nevyplněná pole. Ve vyplněných polích jsou zadané součty řádků a sloupců. Do nevyplněných polí je třeba doplnit číslice z rozsahu 1–9 tak, aby součet číslic v řádku a sloupci vždy dával číslo uvedené na začátku daného součtu. Příklad kakura a jeho řešení je uveden na obrázku, viz 1.



Obrázek 1. Prázdné a vyřešené kakuro.

Řádky a sloupce se v terminologii hry označují jako segmenty. V rámci segmentu se nesmějí opakovat stejné číslice. Z toho lze snadno odvodit, že nejdelší segment může mít 9 polí a nejvyšší možný předepsaný součet je 45 (1+2+3+4+5+6+7+8+9). Nejnižší možný součet odpovídá číslu 3 (1+2). Důležité ovšem je, že správně zadaný logický rébus kakuro má vždy a pouze jedno jedinné řešení!

2.2. Základní postupy řešení kakura

Základním postupen při řešení kakura je nalezení součtů, které mají omezený počet kombinací číslic, kterými je lze vyplnit. Existují tzv. *jednoznačné součty*, což jsou součty vyplnitelné jen jedinou kombinací číslic. Například součtu 6 v řadě délky 3 lze docílit pouze kombinací čísel [1, 2, 3], součtu 10 v řadě délky 4 zase pouze kombinací čísel [1, 2, 3, 4]. Rozpoznáním těchto jednoznačných součtů si značně ulehčíme hru, u těžších rébusů je téměř nemožné najít řešení bez jejich znalosti.

2.3. Jednoznačné součty

Význam jednoznačných součtů byl vysvětlen v předešlých odstavcích. Celkem existuje 25 jednoznačných součtů (čtyři pro délky součtů 2-7 a jeden pro délku 9). Jejich seznam je uveden v příloze E.

2.4. Hlavní faktory ovlivňující obtížnost kakura

Z předchozích odstavců je jasné, že jedním z hlavních faktorů ovlivňujících obtížnost kakura je počet jednoznačných součtů v něm. Mezi další faktory patří tvar kakura, kde jde hlavně o délku součtů v kakuru - čím delší součty obsahuje, tím je kakuro obtížnější. Velkou roli zde také hraje celková velikost hrací desky, opět čím větší, tím složitější.

2.5. Existující implementace

Pro generátor nových kakuro se autorovi práce bohužel nepodařilo získat žádné bližší informace o existujících řešeních, v této kapitole je tedy uvedeno jen srovnání uživatelských prostředí pro hraní kakura. Při srovnávání byl brán zřetel na následující parametry:

- nutnost instalace programu
- komfort a ergonomičnost uživatelského rozhraní
- způsob generování rébusů (v reálném čase, načtením z databáze)
- obtížnost rébusů
- rozmanitost doplňujících funkcí

Cílem tohoto srovnávání bylo získat přehled o existujících programech, jejich řešeních, poučit se z jejich chyb a naopak se inspirovat kvalitními řešeními.

Aplikací s touto tématikou je velké množství, nebylo tedy možné otestovat všechny. Do přehledu byly vybrány programy, které mají zajímavé a neobvyklé funkce. Nutno dodat, že většina z testovaných programů je placených.

2.5.1. DoKakuro

DoKakuro (http://dokakuro.com/) patří do skupiny sofistikovanějších herních prostředí. Umožňuje vracení tahů, uložení hry, nebo například funkci „psaní tužkou“, kdy je možné si do jednotlivých polí kakura poznamenat, která čísla připadají v úvahu pro vložení.

Celé prostředí je vytvořeno pomocí HTML. Problémem z hlediska hratelnosti je fakt, že program není kompatibilní s českou klávesnicí a pro zadávání čísel je třeba přepnout na klávesnici anglickou. Kakura jsou načítána z databáze, složitější rébusy je ovšem nutné si zaplatit. K dispozici je několik úrovní obtížnosti, ovšem jejich odstupňování již není zcela vyvážené. Proto i lehké rébusy mohou začátečníkům dělat problémy.

2.5.2. Kakurolive

Kakurolive (http://kakurolive.com/) je typickým příkladem minimalistického přístupu k uživatelskému rozhraní. Prostředí pro hraní je jednoduché, obsahuje tlačítka pro výběr obtížnosti, kontrolu správnosti, resetování a zobrazení nového kakura.

Celé prostředí je vytvořeno pomocí HTML a Javascriptu. Nové hry jsou načítány z databáze. Hra obsahuje 4 úrovně obtížnosti, které se od sebe liší zejména velikostí řešeného kakura. Hratelnost a zábavnost je na vysoké úrovni.

2.5.3. Kakuro Master

Kakuro Master (http://www.kakuro.com/) je kompletně vytvořený v prostředí Flash. Kakura jsou načítána z předpřipravené databáze, nejsou tedy generována v reálném čase. Hra vyniká pěknou grafikou a animacemi typickými pro flashové hry. Hra obsahuje zajímavou funkci, kdy po najetí myší na součtové políčko jsou zobrazeny veškeré možné kombinace pro daný součet.

Program umožňuje vybírat mezi 4 velikostmi a 3 stupni obtížnosti kakura, ukládat rozehranou hru, vracet tahy. Rébusy spadají do kategorie těžších a tudíž nejsou vhodné pro začínající řešitele. Program celkově působí dobrým dojmem.

2.5.4. Kakuro Cross Sums Puzzle

Kakuro Cross Sums Puzzle (http://www.kakuropuzzle.com/) je hra standartního vzhledu pod operačním systémem Windows. Hra nabízí 4 stupně obtížnosti, lišící se zejména velikostí kakura. Program obsahuje většinu obvyklých funkcí jako je ukládání anebo automatické vyřešení rébusu.

2.5.5. Zhodnocení

Programů pro hraní kakura existuje celá řada, programy šířené zdarma většinou mají méně funkcí a menší kvalitu než placené programy. Také lze obecně říci, že aplikace vyžadující instalaci obsahují více funkcí oproti aplikacím běžícím v internetovém prohlížeči.

Na základě porovnání a zhodnocení funkcí a slabin v současné době existujících programů byly vyvozeny následující závěry:

- programy pro hraní kakura online mají velmi rozdílnou kvalitu, méně kvalitní programy umožňují řešit jen omezený počet rébusů (malá databáze předgenerovaných her)
- některá kakura jsou až příliš obtížná i při zvolené lehké obtížnosti
- některé programy nemají intuitivní ovládání, například není možné vkládat čísla do políček pomocí klávesnice
- mezi standartní nabízené funkce patří zobrazení správného řešení, kontrola správnosti našeho řešení, vymazání všech zadaných čísel v našem řešení, uložení a načtení rozehraných her

2.6. Generátor obrysů

Generátor obrysů je část programu zodpovědná za vytváření prázdných obrysů, do kterých se poté generují samotná čísla součtů. Správný obrys (tvar) kakura musí splňovat několik podmínek. Některé podmínky vycházejí přímo z definice kakura, další souvisejí s nutností regulace obtížnosti generovaných rébusů. Cílem je vytvořit generátor, který vytváří obrysy přiměřené obtížnosti a zároveň je dost rychlý pro generování obrysů v reálném čase.

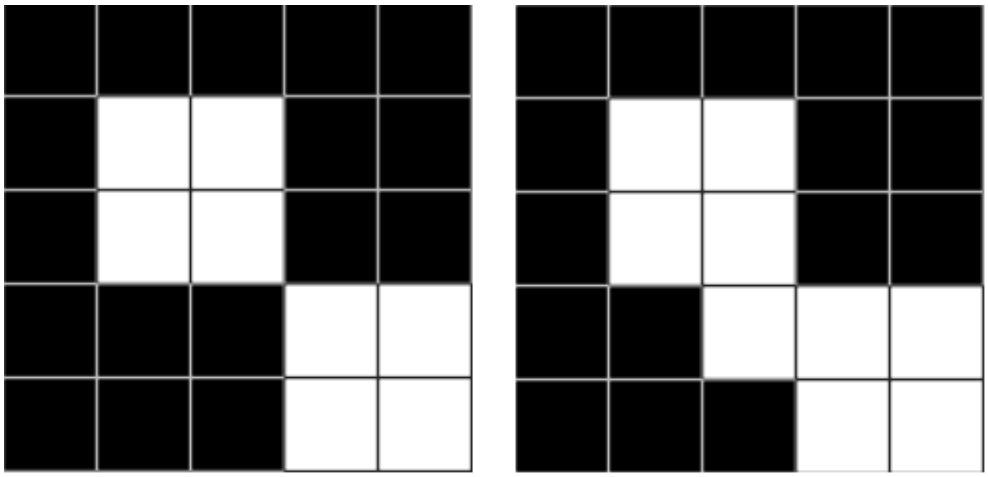
2.6.1. Základní požadavky kladené na tvar kakura

Souvislost - vygenerovaný tvar kakura musí být souvislý. Pokud by souvislý nebyl, získali bychom generováním více dílčích menších kakuro místo jednoho velkého.

Délky součtů - délka součtů v kakuru by měla být minimálně dvě pole. Součty délky jedna zde vlastně nemají význam a znamenaly by pouze doplnění čísla ze součtového pole, což je pouhý mechanický přepis hodnoty.

Do součtu v kakuru lze vkládat pouze číslice 1–9 a tyto číslice se nesmí opakovat. Z tohoto omezení vyplývá, že se ve tvaru kakura nesmí vyskytovat součty větší délky než devět polí.

Obtížnost generovaného tvaru - tvar kakura je jedním z významných faktorů ovlivňujících jeho celkovou obtížnost. Tvar obsahující mnoho sousedících



Obrázek 2. Nesouvislý a souvislý obrys kakura.

a křížících se dlouhých součtů je obecně mnohem těžší než tvar, kde je většina součtů krátkých a dlouhé součty nejsou blízko sebe.

Zároveň platí, že z lehkého obrysu lze vytvořit jak lehké, tak těžké kakuro. Z těžkého obrysu ale vznikají pouze těžká a velmi těžká kakura. Z tohoto důvodu není nutné generovat obrysy různých obtížností, pro potřeby hry stačí vytvářet tvary lehké.

2.7. Způsoby generování tvarů

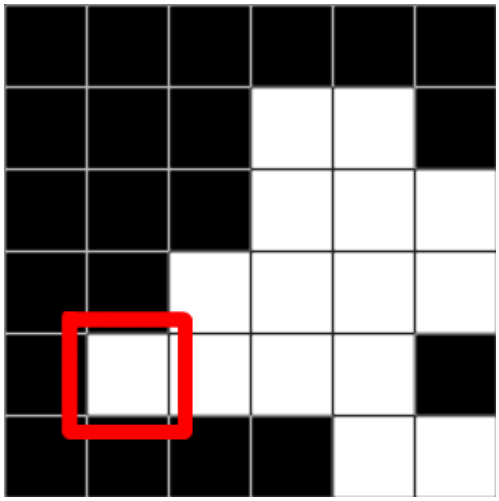
Při tvorbě algoritmu generování tvarů je třeba splnit požadavky uvedené výše. Generování obrysů je možno založit na třech přístupech, konkrétně na přídávání polí, odebírání polí a na kombinaci obou.

2.7.1. Přidávání polí

Tento postup je založen na postupném přidávání políček do prázdné hrací desky. Logické a výhodné je přidávat pole tak, že se připojují k již existujícímu tvaru.

Výhodou tohoto přístupu je, že nám odpadá starost o kontrolu souvislosti generovaného tvaru, výsledný tvar je vždy souvislý. Také je jednoduché zabránit vzniku nepřípustně dlouhých součtů.

Problémem naopak je zamezit vzniku součtů délky jednoho pole. Tyto jednotkové součty při tomto způsobu generování přirozeně vznikají a je třeba vyřešit jejich odstranění. Další nevýhodou je ztížená regulace obtížnosti generovaných tvarů - při přidávání polí do tvaru dochází k prodlužování jednotlivých součtů. Často také dochází k tomu, že se dva původně krátké součty spojí a vznikne tak jeden dlouhý součet.



Obrázek 3. Obrys kakura s nepovoleným jednotkovým součtem.

Algoritmus je možné vylepšit tím, že nebudeme přidávat jen po jednom poli, ale po celém shluku polí. Můžeme zvolit například shluk o velikosti 2x2 pole. Tímto způsobem nám odpadne vznik jednotkových součtů, bohužel se tím redukuje rozmanitost obrysů, které jsme schopni vygenerovat.

2.7.2. Odebírání polí

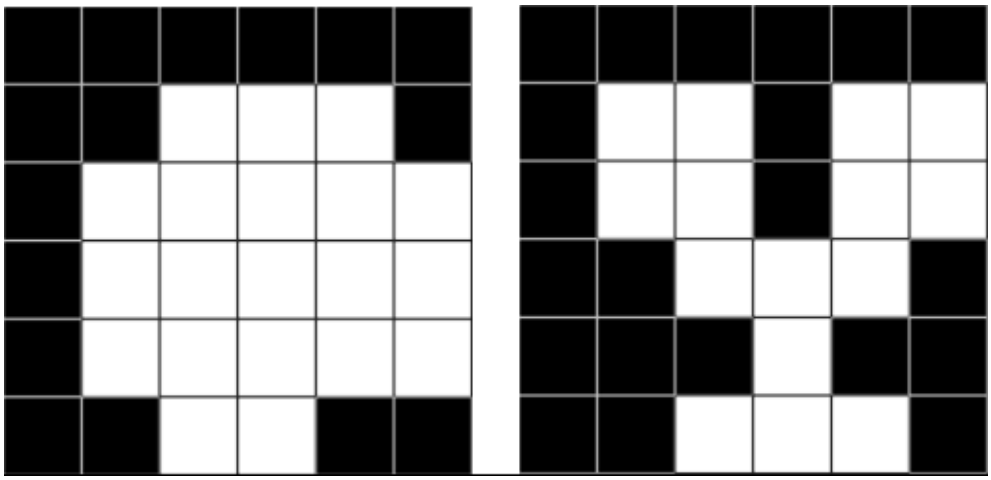
Postup je založen na odebírání polí z hrací desky (přidávání výplně). Na začátku jsou všechny políčka označena jako buňky kakura a cílový tvar vzniká postupným nahrazováním těchto polí výplní.

Výhodou tohoto přístupu je fakt, že je jednoduché splnit podmínku absence jednotkových součtů. Před odebráním buňky se otestuje, jestli touto operací vznikne jednotkový součet a pokud ano, pole se jednoduše neodebere. Také se tímto způsobem dají lehce vytvářet lehké tvary, odebíráním polí totiž dochází ke zkracování jednotlivých součtů, což je žádoucí.

Naopak složitější je splnění posledního požadavku na tvar kakura a to jeho souvislosti. Odebráním pole může kdykoliv dojít k rozpadu kakura a jedinou možností je průběžně testovat celý tvar na jeho souvislost. Zajištění absence nepřípustně dlouhých součtů je také složitější, naštěstí pravděpodobnost výskytu takovýchto součtů není příliš vysoká.

2.7.3. Kombinovaný způsob

Poslední možností generování obrysů je kombinace obou výše zmíněných přístupů. Buď je možné vycházet z metody přidávání polí a na konci generování řešit problém jednotkových součtů odebráním příslušných polí, nebo lze začít odebíráním polí s tím, že se nebude testovat souvislost vznikajícího tvaru. Opě-



Obrázek 4. Obrys kakura s dlouhými a krátkými součty.

tovné souvislosti obrysu se poté dosáhne vhodným přidáním polí, které opět spojí jednotlivé komponenty do jednoho obrysu. Problémem v tomto případě je, že opětovným přidáváním polí tam, odkud již byla jednou odebrána, se prodlužují součty.

2.7.4. Porovnání způsobů

Přidávání polí

Výhody:

- souvislost kakura se udržuje přirozeně
- zabránění vzniku dlouhých součtů je snadné

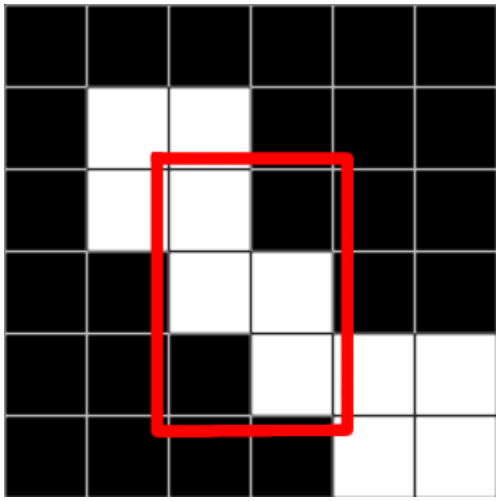
Nevýhody:

- vznikají jednotkové součty a je potřeba je odstraňovat
- přirozeně se generuje obtížnější tvar, toto nelze jednoduše redukovat

Odebírání polí

Výhody:

- nevznikají jednotkové součty
- generují se přirozeně jednoduché, případně středně těžké tvary



Obrázek 5. Nevyjádřitelný obrys kakura pomocí shluků.

Nevýhody:

- v obrysu mohou zůstat nepřípustně dlouhé součty, pravděpodobnost je však malá a navíc s rostoucí velikostí obrysu se téměř nezvyšuje
- je třeba průběžně testovat souvislost kakura, test je algoritmicky jednoduchý, ale ne úplně časově levný (složitost jednoho testu je dána složitostí prohledávání grafu do šířky či do hloubky), tento test je nutné vykonávat opakovaně

Kombinovaná metoda

Výhody:

- je možné více či méně úspěšně eliminovat nevýhody obou předchozích metod

Nevýhody:

- náročnější implementace a nastavení optimálních parametrů generování

Nevýhodou způsobu generování založeného na přidávání polí do kakura je obtížné odladění algoritmu tak, aby bylo možné eliminovat jednotkové součty při zachování generování jednoduchého obrysu. Proto byla zvolena metoda založená na odebírání polí z obrysu. Jako první kandidát byla díky jednodušší implementaci zvolena metoda prostého odebírání buněk s tím, že pokud se ukáže jako nepoužitelná z důvodu velké časové složitosti, použije se metoda kombinovaná.

Jednodušší metoda se nicméně osvědčila a kombinovanou metodu nebylo třeba použít.

2.8. Algoritmus generování obrysů

Celý algoritmus je založen na opakování smyčky `najdi dlouhý součet` – `najdi vhodné pole k odebrání` – `odeber pole`. Na konci generování je pak ještě proveden test na existenci nepřípustně dlouhých součtů, pokud se v obrysu nacházejí, vygenerujeme nový tvar.

2.8.1. Reprezentace herní desky a polí obrysu

Herní deska je reprezentována jako jednorozměrné pole buněk. Každá buňka si uchovává informace o své poloze v rámci herní desky a také o tom, je-li použita či zastává jen úlohu výplně obrysu.

Dále jsou v buňce uchovávána data o řádkových a sloupcových součtech, do kterých buňka patří. Tato data jsou udržována v každé buňce zvlášť, použití sdílených struktur by bylo problematické z důvodu dynamických změn součtů. Každá buňka si tedy udržuje data o souřadnicích počátku a konce řádkového a sloupcového součtu, do kterého patří. Pokud se daný součet mění, jsou všechny buňky v něm zaktualizovány. Tento mechanismus umožňuje udržovat si přehled o délkách jednotlivých součtů a rychle testovat, zda odebráním konkrétní buňky nevzniká jednotkový součet.

Testování souvislosti generovaného tvaru se děje jeho prohledáváním do šířky. Algoritmus si počítá počet nalezených buněk a porovnává ho s celkovým počtem buněk v obrysu. Pokud se počty rovnají, tvar je souvislý, pokud ne, tvar souvislý není.

2.8.2. Výběr pole pro odebrání

Výběr pole pro odebrání z kakura probíhá v několika iteracích. V každé z nich se projde celá herní deska a jsou hledány součty určité délky a větší (konkrétně v první iteraci se hledají součty minimální délky 10, následně 7 a 3, toto nastavení je výsledkem testování algoritmu). Z jednotlivých políček součtu je pak pro odebrání vybráno to, které má dlouhý i druhý kolmý součet. Dále je proveden test, zda odebráním daného pole nedojde ke vzniku jednotkového součtu či narušení souvislosti kakura. Pokud ne, je pole odebráno, pokud ano, je vybráno jiné pole z daného součtu. Pokud není možné odebrat žádné pole součtu, je součet ponechán v původním stavu.

2.8.3. Test na nepřípustně dlouhé součty

Pravděpodobnost vzniku nepřípustně dlouhých součtů není velká a se zvětšující se velikostí desky téměř neroste. Tento test lze tedy provést jen jednou - na konci generování. Pokud deska takovýto součet obsahuje, je vygenerován tvar jiný.

2.8.4. Algoritmus generování

Nejprve jsou náhodně odebrána pole z okraje herní desky. Další náhodnost algoritmu zajišťuje odebrání polí z náhodných míst obrysu. Pseudokód hlavní smyčky algoritmu:

```
generujNovyTvar (int pocetRadku, int pocetSloupcu) {
  while (true) {
    inicializacePlanu(pocetRadku, pocetSloupcu);
    odeberPoleZOkraje();
    odeberPoleZTvaru();
    odeberPoleZDlouhychSouctu(10);
    odeberPoleZDlouhychSouctu(7);
    odeberPoleZDlouhychSouctu(3);
    if (testNaPrilisDlouheSoucty() == true) {
      ulozTvar();
      break;
    }
  }
}
```

Základní pomocné metody, které se provádějí při každém pokusu o odebrání pole kakura jsou `jeMoznoOdebratPole()` a `odeberPole()`. První metoda nejdříve otestuje, zda tvar odebráním pole neztratí souvislost, následně je proveden test na vznik jednotkových součtů. Druhá metoda provádí samotné odebrání pole z obrysu, při této operaci jsou aktualizovány informace o součtech v každé změněné buňce.

Metoda `odeberPoleZOkraje()` projde okraje obrysu a náhodně odebere některá pole. Pseudokód této metody:

```
odeberPoleZOkraje () {
  for (vsechna pole na okraji) {
    if (random.next() == true) {
      if(jeMoznoOdebratPole()) {
        odeberPole();
      }
    }
  }
}
```

2.8.5. Výkonnost algoritmu

Výkonnost algoritmu se měřila jako čas potřebný k vygenerování 1000 obrysů daných rozměrů. Konfigurace stroje: Intel Core2Duo 1,66 Ghz, 3 GB RAM, MS Windows 7. Naměřené hodnoty jsou uvedené v tabulce 1.

Z naměřených dat vyplývá, že generátor vytváří obrysy do velikosti 20x20 polí velmi rychle, při generování větších tvarů se rychlost snižuje a u tvarů velikosti 30x30 polí již začíná být rychlost generování nepoužitelná pro generování v reálném čase.

Rozměry tvaru	**Celkový čas (ms)**	**Průměr na jeden obrys (ms)**
10x10	2 890,32	2,89
15x15	10 314,46	10,31
20x20	43 920,12	43,92
25x25	139 326,16	139,32
30x30	343 231,85	343,23

Tabulka 1. Závislost rychlosti generování obrysu na jeho velikosti.

2.9. Generátor nových kakuro

Při vytváření generátoru nových her bylo potřeba vyřešit regulaci obtížnosti generovaných rébusů. Také bylo potřeba vytvářet validní desky, které mají pouze jedno jedinné řešení a to vše v přijatelných časových intervalech.

2.9.1. Pracovní kakuro

Hlavním problémem při generování nového kakura je požadavek na jeho jedinečnost. Není problém vygenerovat kakuro, které toto splňovat nemusí. Takové kakuro bude dále v textu označeno termínem *pracovní kakuro*. Při generování pracovního kakura se začíná prázdným obrysem, který byl vygenerován postupem uvedeným v předchozí kapitole. Jeho políčka se vyplní náhodnými čísly tak, aby bylo dodrženo pravidlo, že v jednom součtu se žádné číslo nesmí opakovat. Poté se dopočítají jednotlivé řádkové a sloupcové součty a pracovní kakuro je hotové.

Vytváření pracovního kakura je ideální příležitostí pro regulaci obtížnosti generovaných rébusů. Nejjednodušší cestou je využití jednoznačných součtů. Při vytváření pracovního kakura se do něho vygeneruje určitý počet jednoznačných součtů odpovídající požadované obtížnosti.

2.9.2. Validní kakuro

Validní kakuro je takové, které splňuje všechny požadavky na něj kladené. Rozdíl oproti pracovnímu kakuru je v tom, že validní kakuro splňuje požadavek jedinečnosti řešení.

Jedinou cestou, jak ověřit splnění podmínky jedinečnosti řešení je kakuro skutečně vyřešit a tím i zjistit, jestli má jen jedno jediné řešení. Pokud pracovní kakuro má opravdu jen jedno řešení, jde o validní kakuro a generování je u konce.

Pokud jen jedno řešení nemá, je nutné vygenerovat nové pracovní kakuro a proces jeho řešení opakovat.

Při vytváření nového pracovního kakura je možné buď vygenerovat celé kakuro znovu nebo použít výsledek předchozí neúspěšné iterace algoritmu a pokusit se poslední pracovní kakuro upravit tak, aby se zvýšila pravděpodobnost, že bude validní.

Vhodný algoritmus pro vyřešení kakura je rekurzivní algoritmus s návratem. Tento algoritmus je v obecné verzi velmi pomalý a proto je nutné ho doplnit o sadu optimalizací, které sníží celkový počet zkoumaných možností.

2.9.3. Vložení jednoznačných součtů

Pro účely vložení jednoznačných součtů do kakura je třeba kakuro reprezentovat jako soubor součtů, nikoliv jako soubor políček. Každý součet má určitou délku a obsahuje odpovídající počet políček. Každé pole kakura je tedy členem jednoho řádkového a jednoho sloupcového součtu.

Nejprve se podíváme na délku součtu. Poté vezmeme náhodně některý z jednoznačných součtů, který je možno vložit do součtu dané délky a vyplníme jím daný součet. Problémem je, že žádný součet není izolovaný a vždy v kolmém směru protíná ostatní součty. Může se tedy stát, že je již vyplněn jeden nebo více součtů kolmých na právě vyplňovaný součet. V tom případě je již do právě vyplňovaného součtu jedno nebo více čísel vloženo. Dále může dojít k tomu, že je již vyplněn jeden nebo více součtů rovnoběžných s právě vyplňovaným součtem a v důsledku toho je vloženo již nějaké číslo v jednom nebo více součtech kolmých na právě vyplňovaný součet.

Oba případy mohou vést k tomu, že číslice jednoznačného součtu nebude možné do součtu vložit. Tato situace z hlediska vyváření kakura není vážnějším problémem. Pokud do daného součtu nelze jednoznačný součet vložit, jednoduše se nevyplní a místo něj se pokusíme vyplnit jiný součet.

2.9.4. Doplnění kakura náhodnými čísly

Po vyplnění části kakura jednoznačnými součty je třeba dovyplnit zbytek kakura náhodnými čísly. Toto je jednoduché. Jediné, co se při vyplňování kontroluje je, jestli není číslo v součtu již uvedeno, čili jestli se neopakuje. Často se stává, že dané číslo na vybrané místo vložit nejde a místo něj se musí vložit číslo jiné. Pokud se stane, že na dané místo nelze vložit žádné číslo, tak se právě generované kakuro zahodí a začne se s vytvářením nového.

2.9.5. Přehled algoritmického řešení vytváření pracovního kakura

1. vezmi obrys kakura a seznam součtů v něm, seznam součtů promíchej
2. dokud není vyplněn určený počet součtů nebo není dosaženo konce seznamu součtů:
3. vezmi součet, který je na řadě a najdi pro něj existující jednoznačné součty, pokud jich existuje více, nejprve je promíchej
4. vezmi po jednom jednoznačné součty a pomocí rekurzivního algoritmu s návratem se je pokus umístit do součtu, ve chvíli, kdy se podaří umístit první jednoznačný součet, pokračuj opět bodem 2, pokud se nepodaří umístit žádný jednoznačný součet, tento součet přeskoč, vezmi další součet a pokračuj opět bodem 2
5. vezmi obrys kakura částečně vyplněný v předchozím kroku a pro všechna dosud nevyplněná pole:
6. zkus do každého pole v náhodném pořadí vložit čísla 1–9, ve chvíli, kdy úspěšně vložíš do pole nějaké číslo, vezmi další pole a pokračuj krokem 5, pokud žádné takové číslo neexistuje, vezmi další kakuro a pokračuj bodem 1

2.9.6. Vytvoření validního kakura

Kontrola jedinečnosti pracovního kakura je nejobtížnější částí celého generování. Řešení kakura je NP-úplný problém, úloha má nedeterministické polynomiálně složité řešení. Vyřešení kakura je tedy velmi časově náročné. Uspokojivé a rychlé řešení není možné bez dalších optimalizací.

Základním postupem je pokusit se uhodnout řešení, tedy vyplnit obrys náhodnými čísly a poté zkontrolovat, jestli bylo řešení uhodnuto správně. Pokud ne, je třeba hádat dál. Je vhodné použít rekurzivní algoritmus s návratem, pomocí kterého se postupně jednu po druhé zkouší všechny možnosti rozdělení čísel do polí. Celý postup je ve své nejjednodušší verzi extrémně pomalý, počet možností, které je třeba vyzkoušet je 9^x, kde x je počet normálních polí kakura. Navíc hned první řešitelné pracovní kakuro je validní jen málokdy, na získání validního kakura je třeba vyřešit několik desítek, stovek či v případě velkých rébusů tísíců pracovních kakur.

Pro dosažení optimálních časů kontroly jedinečnosti je třeba výše popsaný základní postup vylepšit. Toto je možno dosáhnout pomocí sady drobných vylepšení, které významně sníží celkový počet kombinací, které musíme zkoumat. Tyto vylepšení je možné aplikovat ve dvou fázích, tedy ještě před započetím řešení pomocí hlavního rekurzivního algoritmu anebo přímo za jeho běhu.

Před započetím běhu hlavního algoritmu lze provést omezení celkového počtu procházených možností pomocí jednoznačných součtů. Toto nám výrazně zrychlí

běh algoritmu (například u součtu délky 3 se počet zkoumaných možností sníží z 9^3 na 3^3). Tento krok se dá shrnout takto:

1. vytvoř seznam součtů daného kakura
2. pro každý součet:
3. zjisti, zda se jedná o jednoznačný součet
4. pokud ano, nastav u všech polí daného součtu jako čísla, která je možno do něho vložit, číslice daného jednoznačného součtu, pokud se o jednoznačný součet nejedná, nedělej nic, v obou případech poté pokračuj bodem 2

Toto vylepšení výrazně zrychlí běh algoritmu. Platí to ale za předpokladu, že kakuro obsahuje dostatek jednoznačných součtů. Proto je zrychlení běhu algoritmu patrné pouze u lehkých a středně těžkých kakuro.

Během vykonávání hlavního algoritmu je nutné nějakým způsobem kontrolovat už v okamžiku vkládání čísla, jestli se ono číslo na dané místo může hodit, respektive jestli se na dané místo určitě nehodí. Tímto ořežeme neperspektivní větve algoritmu. Víme, že se v jednom součtu nesmí vyskytovat žádná číslice dvakrát. Dále víme, že je bezpředmětné do součtu vkládat číslo, které je větší než daný součet, tedy cokoliv většího než 5 do součtu 6 přes 2 políčka, nebo naopak příliš malé, tedy cokoliv menšího než 5 do součtu 14 přes 2 políčka.

2.9.7. Přehled algoritmického řešení kontroly jedinečnosti kakura

Jak bylo uvedeno výše, základem řešení jedinečnosti kakura je využití rekurzivního algoritmu s návratem. Seznam všech vyplňovacích polí kakura je uložen v poli `seznam`, po kterém se pohybujeme pomocí promenné `index`. Pseudokód algoritmu vypadá následovně:

```
int pocetReseni = 0;

void vyres (int index) {
  if (index == seznam.length) {    // mame reseni
    ulozReseni();
    pocetReseni++;
    if (pocetReseni == 2) {    // mame 2 reseni, dale nema cenu hledat
      return();
    }
  }
  else {
    // hlavni cast algoritmu
    for (int i=9; i > 0; i--) {
      if (seznam[index].lzeVlozit(i)) {
        seznam[index].vloz(i);
        vyres(index + 1);
        seznam[index].odeber(i);
      }
    }
  }
}
```

V hlavní části algoritmu se zkouší vkládat čísla 1–9, o kontrolu se stará funkce `lzeVlozit()`, která využívá pravidel a mechanizmů uvedených výše. Pokud algoritmus uspěje s vložením čísla, zanoří se rekurzivně hlouběji a pokusí se nějakou číslici vložit do dalšího políčka. Při návratu z rekurze naopak číslici z políčka odebere, aby uvolnil místo pro vložení číslice jiné.

Pokud se během vkládání úspěšně podaří vložit číslici i do posledního políčka, znamená to, že je nalezeno řešení. Je ale nutné pokračovat dále a prozkoumat všechny další možnosti, abychom zjistili, jestli má kakuro ještě nějaké další řešení.

2.9.8. Výkonnost algoritmu

Samotné nalezení jednoho řešení, pokud existuje, je velice rychlá a časově nenáročná operace. My však procházíme veškeré existující kombinace vložení čísel, tudíž časová složitost exponencionálně roste s rostoucí velikostí kakura a také závisí na počtu jednoznačných součtů v něm, tedy na obtížnosti. Tento algoritmus přestal být vyhovující pro generování kakura v reálném čase už od velikosti 12x12 na střední obtížnost.

Z tohoto důvodu se za běhu programu generují kakura potřebné délky a dané obtížnosti na pozadí. Je tak zajištěna rychlá odezva vůči uživateli a zároveň je splněna podmínka generování rébusů v reálném čase. Tento algoritmus lze zároveň využít pro řešitele validních desek kakuro, kde nalezení onoho jediného řešení je otázkou milisekund a to i pro velká a obtížná kakura.

3. Programátorská část

3.1. Návrh aplikace

Aplikace je napsána v jazyce C#, konkrétněji pod Windows Presentation Foundation (WPF). Je rozdělena na dvě hlavní části.

První část tvoří jádro celé aplikace. Obsahuje třídy umožňující vytvářet rébusy v reálném čase a také třídy umožňující řešit jakýkoliv zadaný rébus kakuro. Celé jádro je zkompilováno do DLL knihovny a je tedy možné jej využít pro další účely.

Druhou část tvoří samotné uživatelské rozhraní. Je postaveno tak, že využívá funkcionalitu jádra aplikace. Není tedy žádný problém jej nahradit jiným uživatelským rozhraním. Důraz byl kladen na přívětivost a jednoduchost ovládání a na celkovou ergonomičnost aplikace s přihlédnutím k zavedeným standartům. Využívá se zde možnosti návrhu vzhledu aplikace v jazyce XAML, což je značkovací jazyk vycházející z XML.

Aplikace běží ve třech vláknech. První vlákno se stará o grafické rozhraní a komunikaci s uživatelem. Druhé vlákno obsluhuje požadavky uživatele (např. na spuštění nové hry nebo její vyřešení). Toto vlákno zároveň v případě potřeby spouští třetí vlákno, které se kvůli větší časové náročnosti na generování obtížných desek stará o jejich „předgenerovávání“ na pozadí aplikace. Tímto přístupem je dosaženo rychlé odezvy aplikace vůči uživateli (nemusí na nic čekat) při zachování požadavku náhodnosti generovaných her a jejich generování v reálném čase.

Pro ukládání rozehraných her je využito formátu XML.

3.2. Popis tříd jádra aplikace

`Cell` - Třída představující jednu buňku hrací desky. Obsahuje informaci o svém typu. Může nabývat tří hodnot a to `EmptyCell` (zarážka, prázdná buňka vymezující obrys desky), `SummaryCell` (součtová buňka obsahující řádkové a sloupcové součty) a `ContentCell` (vyplňovací buňka, do které jsou zadávány číslice tvořící řešení hry). Dále třída obsahuje položky pro uchovávání informací o své poloze v rámcí jednotlivých součtů, o délkách součtů, do kterých spadá a třeba také seznam hodnot, kterými je možné ji vyplnit (u vyplňovacích buněk).

`Board` - Třída představující herní desku. Obsahuje kompletní údaje, jako je její velikost nebo seznam jednotlivých polí (objekty typu `Cell`), která obsahuje. Poskytuje také jednoduchou funkcionalitu pro práci s ní, jako je přidávání a odebírání pole, nastavení hodnoty pole, změnu velikosti a obtížnosti nebo návrat do počátečního stavu.

`Controller` - Statická třída obsahující statické metody pro kontrolu různých pravidel na herní desce. Umožňuje zjistit, zda je deska spojitá, zda neobsahuje jednotkové součty nebo nepřípustně dlouhé součty. Umožňuje nám provádět kontrolu, zda je deska validní.

`Generator` - Obsahuje nejdůležitější funkce, zajišťující hlavní funkcionalitu aplikace. Využívá obou tříd uvedených výše pro generování validních rébusů kakuro. Zajišťuje tedy tvorbu obrysů, tvorbu pracovních i validních kakur. Umožňuje řešit libovolný rébus. Stěžejní jsou zde dvě metody:

- `void GenerateNewBoard(int width, int height, Difficulty dif)` - Metoda vytvoří novou herní desku splňující všechna pravidla, včetně požadavku na jedinečnost řešení. Díky tomu je při jejím volání získáno i řešení desky a to díky volání metody uvedené níže. Jako parametry jí jsou předávány výška, šířka a obtížnost (typ enum).
- `void Solve(int index)` - Rekurzivní metoda, která prozkoumá všechny možnosti řešení hrací desky a zjistí, jestli je řešitelná a jestli má více než jedno řešení. Jako parametr jí je předáván index prvního nevyplněného pole na hrací desce. Toto je z důvodu rekurzivního volání nezbytné.

`LogicManager` - Hlavní třída jádra aplikace, která je v podstatě rozhraním mezi jádrem a okolními třídami. Poskytuje základní funkcionalitu, jako je spuštění nové hry, uložení a načtení hry, zapauzování hry, nápovědu dalšího tahu, kompletní vyřešení rébusu, spuštění generování rébusů na pozadí. Uživatelské rozhraní komunikuje s jádrem pouze prostřednictvím této třídy.

`SaveData` - Třída sloužící jako kontejner pro uchovávání a následné serializování dat do formátu XML. Obsahuje objekt herní desky, uplynulý čas hry a také hash kód pro kontrolu konzistence při opětovném načítání.

`IOManager` - Statická třída obsahující metody `Save(string path, object obj)` a `Load(string path, Type type)`. Jsou koncipovány tak, že umožňují uložení/načtení libovolného serializovatelného objektu na disk/z disku.

3.3. Popis tříd uživatelského rozhraní

`KakuroWindow` - Hlavní třída uživatelského rozhraní, stará se o vykreslení hlavního okna aplikace, obsluhu všech událostí vyvolaných uživatelem. Vše je navázáno na jádro aplikace a využívá tak jeho nabízené funkce.

`Složka Controls` - Obsahuje vlastní uživatelské prvky, které bylo potřeba z různých důvodů vytvořit. Jedná se o jednotlivá pole herní desky, které mohou být tří typů. Obsahuje i uživatelské prvky využívané v okně editoru vlastních desek.

Složka `Dialogs` - Nachází se v ní třídy vytvářející dialogy nové hry nebo okna editoru vlastních desek.

Složka `Themes` - Obsahuje předpisy různých stylů používaných pro změnu vzhledu aplikace.

4. Uživatelská část

4.1. Požadavky na spuštění

Operační systém Microsoft Windows XP nebo novější, Microsoft .NET Framework 4.0. Před spuštěním je potřeba program nainstalovat. Aplikaci spustíte souborem *Kakuro.exe*.

4.2. O programu

Tato aplikace slouží jako herní prostředí pro řešení japonských logických rébusů kakuro. Jednoduše a přehledně umožňuje uživateli řešit konkrétní rébus, který je dle jeho požadavků na velikost a obtížnost vygenerován v reálném čase. Zárukou je, že žádné dva vygenerované rébusy nejsou stejné a program je schopen vygenerovat v podstatě jakýkoliv existující rébus v rámci pravidel kakura. Aplikace dále obsahuje editor, ve kterém si každý může vytvořit svou vlastní jedinečnou desku a to jak tvarem, tak zadanými součty. Hru je možné kdykoliv uložit a načíst, pozastavit, použít nápovědu dalšího tahu nebo si nechat celý rébus vyřešit. Rébus je také možné vytisknout.

4.3. Ovládání

Hlavní okno aplikace je rozděleno na několik částí. Nahoře je menu, které obsahuje 4 položky. První je *Hra*, kde můžeme začít novou hru, uložit právě rozehranou hru, případně načíst uloženou hru, dále je zde položka pro vyvolání dialogu tisku a také je z této části menu možné aplikaci ukončit.

Druhou položkou menu jsou *Možnosti*. Z této části menu je možné hru pozastavit a opětovně spustit, nechat si napovědět další tah, provést kontrolu správnosti našeho dosavadního řešení a také si nechat celou hru vyřešit automaticky. Také se zde spouští editor vlastních uživatelských desek.

Třetí položkou menu je položka *Vzhled*. Obsahuje pět předvolených barevných témat a je jen na uživateli, kterému dá přednost.

Poslední položkou menu je *Nápověda*. Pomocí něj můžeme zobrazit informace o programu a autorovi, dále se z tohoto místa otevírá okno s nápovědou k programu.

Pod samotným menu je zobrazena lišta, která pomocí tlačítek s obrázky umožňuje rychlejší přístup k funkcím obsažených v menu, je rozdělena na části podle toho, v jaké položce menu se daná funkcionalita nachází.

Hlavní část okna zabírá vykreslovací plocha, ve které je zobrazena aktuální hraná deska rébusu. Ta se skládá z jednotlivých políček, do kterých je možné po jejich označení zadávat pomocí klávesnice čísla 1–9. Po kliknutí pravým tlačítkem myši na konkrétní buňku se její vzhled mírně změní a je možné si do ní psát

poznámky o možných číslech pro vyplnění. Pokud na to samé pole klikneme levým tlačítkem, můžeme do pole vyplňovat hodnoty řešení.

Ve spodní části okna se nachází lišta zobrazující základní informace o hře, tedy velikost aktuální hrací desky, aktuální zvolenou obtížnost a běžící čas.

Po vložení čísla do posledního nevyplněného pole se zkontroluje správnost řešení a pokud je vše v pořádku, zobrazí se informace o úspěšném vyřešení rébusu a čase, který k tomu bylo zapotřebí.

Samotné okno editoru uživatelských desek je jednoduché. Při jeho spuštění uživatel zadá požadovanou velikost desky (toto je omezeno minimálním počtem 8 polí a maximálním 12 polí). Okno editoru obsahuje vykreslený obrys hrací desky o zadaných rozměrech a v pravé části několik tlačítek pro provádění nejrůznějších akcí. Uživatel si nejprve nakliká tvar hrací desky. Pokud je deska validní a splňuje veškerá pravidla, dostane se ke druhému kroku. Tím je doplnění řádkových a sloupcových součtů. Po vyplnění všech těchto součtů je provedena kontrola jedinečnosti řešení, případně jestli vůbec řešení existuje. Pokud je vše v pořádku, může uživatel tuto svou desku uložit k pozdějšímu řešení nebo ji rovnou začít řešit.

Závěr

Cílem aplikace bylo vytvoření přehledného a uživatelsky příjemného herního prostředí pro řešení a generování náhodných japonských logických rébusů kakuro. Součástí je tedy náhodný generátor nových rébusů a také jejich řešitel. Aplikace umožňuje i tvorbu vlastních rébusů. Vše je generováno v reálném čase, takže hlavního cíle práce se podařilo dosáhnout.

Jako návrh dalšího rozšíření by bylo možné vytvořit grafické prostředí ve 3D nebo rozsáhlým testováním zoptimalizovat a vylepšit stávající algoritmus generování nových rébusů tak, aby se zvýšila jeho použitelnost i u rébusů rozměrů vetších než 15x15.

Reference

[1] Petzold, Charles. *Mistrovství ve Windows Presentation Foundation.* Computer Press, Brno, 2008.

[2] Seta, Takahiro. *The Complexities of Puzzles, Cross Sum and their Another Solution Problems (ASP).* Elektronická publikace, 2001.

[3] *Wikipedia.* Internetová encyklopedie, 2010.

A. Výčet jednoznačných součtů

Součet	Délka součtu	Číslice
3	2	1, 2
4	2	1, 3
16	2	7, 9
17	2	8, 9
6	3	1, 2, 3
7	3	1, 2, 4
23	3	6, 8, 9
24	3	7, 8, 9
10	4	1, 2, 3, 4
11	4	1, 2, 3, 5
29	4	5, 7, 8, 9
30	4	6, 7, 8, 9
15	5	1, 2, 3, 4, 5
16	5	1, 2, 3, 4, 6
34	5	4, 6, 7, 8, 9
35	5	5, 6, 7, 8, 9
21	6	1, 2, 3, 4, 5, 6
22	6	1, 2, 3, 4, 5, 7
38	6	3, 5, 6, 7, 8, 9
39	6	4, 5, 6, 7, 8, 9
28	7	1, 2, 3, 4, 5, 6, 7
29	7	1, 2, 3, 4, 5, 6, 8
41	7	2, 4, 5, 6, 7, 8, 9
42	7	3, 4, 5, 6, 7, 8, 9
45	9	1, 2, 3, 4, 5, 6, 7, 8, 9

Tabulka 2. Výčet jednoznačných součtů.

B. Obsah přiloženého CD

`bin/`
: Instalátor programu Kakuro. Adresář obsahuje i všechny potřebné knihovny a další soubory pro bezproblémové spuštění programu.

`doc/`
: Dokumentace práce ve formátu PDF, vytvořená dle závazného stylu KI PřF pro diplomové práce, včetně všech příloh, a všechny soubory nutné pro bezproblémové vygenerování PDF souboru dokumentace (v ZIP archivu), tj. zdrojový text dokumentace, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty programu Kakuro se všemi potřebnými (převzatými) zdrojovými texty, knihovnami a dalšími soubory pro bezproblémové vytvoření spustitelných verzí programu (v ZIP archivu).

`readme.txt`
: Instrukce pro instalaci a spuštění programu Kakuro, včetně požadavků pro jeho provoz.

Navíc CD/DVD obsahuje:

`data/`
: Ukázková a testovací data použitá v práci a pro potřeby obhajoby práce.

`install/`
: Instalátory aplikací, knihoven a jiných souborů nutných pro provoz programu, které nejsou standardní součástí operačního systému.

U veškerých odjinud převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro materiály, u kterých toto není splněno, je uveden jejich zdroj (webová adresa) v textu dokumentace práce nebo v souboru `readme.txt`.