|  |  |
| --- | --- |
| **TECHNICKÁ UNIVERZITA V KOŠICIACH**  Fakulta elektrotechniky a informatiky | |
| **Algoritmus AQ11 v jazyku C#** | |
|  | |
| **2024** | **Ivan Tkachenko** |

**Abstrakt v SJ**

Abstrakt v slovenčine (referát) je povinnou súčasťou každej práce. Je výstižnou krátkou charakteristikou obsahu dokumentu. Abstrakt býva informatívny a zachováva tematické a štýlové vlastnosti práce. Nevyjadruje hodnotiace stanovisko autora. Obsahuje údaje o cieľoch práce, metódach, výsledkoch a záveroch. Text abstraktu sa píše ako jeden odstavec. Abstrakt neobsahuje odkazy na samotný text práce. Mal by mať rozsah asi 250 slov, nemal by presiahnuť jednu stranu. Pri štylizácii sa používajú celé vety, slovesá v činnom rode a tretej osobe. Používa sa odborná terminológia, menej zvyčajné termíny, skratky a symboly sa pri prvom výskyte v texte definujú.

**Kľúčové slová v SJ**

Kľúčové slovo1, kľúčové slovo2, kľúčové slovo3, kľúčové slovo4,...

Kľúčové slová sú slová, ktoré úzko definujú tému práce a pomocou nich je možné jednoduchšie vyhľadať prácu podľa jej odborného významu.

**Abstract**

Text abstraktu v svetovom jazyku je potrebný pre integráciu do medzinárodných informačných systémov (napr. The Network Digital Library of Theses and Dissertations). Ak nie je možné jazykovú verziu umiestniť na jednej strane so slovenským abstraktom, je potrebné umiestniť ju na samostatnú stranu (cudzojazyčný abstrakt nemožno deliť a uvádzať na dvoch stranách).

**Keywords**

Key word 1, Key word 2, Key word 3, Key word 4,...

**Obsah**

[Zoznam obrázkov 4](#_Toc162797910)

[Zoznam tabuliek 5](#_Toc162797911)

[Zoznam symbolov a skratiek 6](#_Toc162797912)

[1. Teoretický popis algoritmu 7](#_Toc162797913)

[2. Teoretický rozbor zvolenej témy 8](#_Toc162797914)

[3. Analýza stavu problematiky 9](#_Toc162797915)

[4. Návrh a implementácia riešenia zvolenej problematiky 10](#_Toc162797916)

[4.1. Vkladanie obrázkov do dokumentu 10](#_Toc162797917)

[4.2. Vkladanie tabuliek do dokumentu 11](#_Toc162797918)

[4.3. Vkladanie grafov do dokumentu 11](#_Toc162797919)

[4.4. Krížové odkazy na použitú literatúru a webový obsah 12](#_Toc162797920)

[4.5. Citovanie z použitej literatúry a zoznam použitej literatúry 12](#_Toc162797921)

[4.5.1. Pribežné ukladanie informácií o použitej literatúre 14](#_Toc162797922)

[5. Postup pri odovzdávaní a tlači záverečnej práce 15](#_Toc162797923)

[5.1. Predbežná kontrola originality práce 16](#_Toc162797924)

[5.2. Odovzdanie a registrácia práce v Univerzitnej knižnici TUKE 16](#_Toc162797925)

[Záver 18](#_Toc162797926)

[Zoznam použitej literatúry 19](#_Toc162797927)

[Prílohy 20](#_Toc162797928)

Zoznam obrázkov

[Obr. 1 Vkladanie popisu pre obrázok 4](#_Toc118285416)

[Obr. 2 Vkladanie popisu k tabuľkám 4](#_Toc118285417)

[Obr. 3 Porovnanie percenta zhody fakúlt pre Bc štúdium za rok 2013 4](#_Toc118285418)

[Obr. 4 Priebežne ukladanie citovanej literatúry 4](#_Toc118285419)

[Obr. 5 Vzor protokolu o kontrole originality 4](#_Toc118285420)

[Obr. 6 Portál pre predbežnú kontrolu originality 4](#_Toc118285421)

[Obr. 7 Obrázok grafického CD média 4](#_Toc118285422)

Zoznam tabuliek

[Tab. 1 Štatistické zhodnotenie percenta zhody za rok 2013 4](#_Toc118285423)

1. Teoretický popis algoritmu

Algoritmus Aq realizuje formu učenia s učiteľom. Vzhľadom na množinu pozitívnych udalostí (príkladov) P, množinu negatívnych udalostí N algoritmus generuje pokrytie C pozostávajúce z komplexov, t. j. spojenia atribučných podmienok, ktoré pokrývajú všetky udalosti z P a žiadne udalosti z N. Algoritmus začína zameraním sa na jednu pozitívnu udalosť e, ktorá sa nazýva semienko, ktoré sa potom zovšeobecní vytvorením všetkých maximálne všeobecných komplexov, ktoré pokrývajú semienko a nepokrývajú žiadne udalosti z N. Táto maximálne všeobecná množina komplexov sa nazýva obálka G(e, N). Všetky udalosti pokryté komplexom c sa odstránia z P. Ak je množina príkladov P prázdna, vráti sa obal C; v opačnom prípade sa z P vyberie ďalší semienko a operácia sa opakuje, kým nie je P prázdna. Kvôli jednoduchosti opisu tu môžeme predpokladať, že komplex je ekvivalentný pravidlu.[[1]](#citation1)

1. Popis postupu a jednotlivých funkcií
   1. Data a Preprocessor

Triedy Data a Preprocessor pracujú spoločne na načítaní údajov z daného súboru údajov, ich transformácii do štruktúry, s ktorou je algoritmus schopný pracovať, a ich uložení v inštancii Data pre budúce použitie algoritmom.

* + 1. Preprocessor

Preprocesor zodpovedný najmä za čítanie a transformáciu údajov zo súboru údajov. Preprocesor je statická trieda, takže nemôže ukladať údaje a inštanciu triedy nemožno vytvoriť.

Metódy:

* + - 1. public static List<string> GetHeaders(string datasetPath)

Metóda preberá ako vstup reťazcovú cestu k súboru údajov. Pomocou balíka CsvHelper načíta názvy hlavičiek zo súboru údajov. Uloží ich do zoznamu reťazcov, ktorý sa potom vráti.

* + - 1. public static List<List<string>> GetRecordsString(string datasetPath)

Metóda preberá ako vstup reťazcovú cestu k súboru údajov. Pomocou balíka CsvHelper načíta každý stĺpec atribútu zo súboru údajov a na začiatok pridá aj stĺpec Id. Uloží ich do dvojrozmerného zoznamu, v ktorom každý uložený zoznam reťazcov predstavuje stĺpec hodnôt jednotlivých atribútov. Tento dvojrozmerný zoznam je návratovou hodnotou.

* + - 1. public static List<List<string>> TransformRecordsIntoRecordsIndividual(List<List<string>> recordsString)

Metóda transformuje zoznam stĺpcov atribútov, ktorý sme získali predchádzajúcou metódou, na dvojrozmerný zoznam, kde každý prvok je zoznam reťazcov, ktorý predstavuje jeden záznam zo súboru údajov.

* + - 1. public static Dictionary<string, List<string>> CreateAttributesDictionary(List<string> headers, List<List<string>> recordsString)

Metóda preberá zoznam hlavičiek a zoznam stĺpcov - atribútov zo súboru údajov ako vstupy a vytvára užitočný slovník. Pomocou tohto slovníka je možné previesť akúkoľvek reťazcovú hodnotu atribútu na číselnú hodnotu. Slovník obsahuje názvy jednotlivých atribútov ako kľúčové hodnoty. Pre každý atribút existuje zoznam hodnôt atribútov. Hodnota na indexe 0 je prázdna, takže žiadna hodnota atribútu nemôže mať index 0. Pomocou toho je možné previesť akúkoľvek reťazcovú hodnotu atribútu na číselnú hodnotu indexu atribútu v príslušnom zozname v slovníku a naopak. A s tým súvisí aj negácia hodnôt atribútov (napríklad Hair : Black -> Hair : Not Black), ak má Black index 2, potom negácia tejto hodnoty bude -2. Výstupom je tento slovník.

* + - 1. public static List<List<int>> ConvertRecordsToInt(List<List<string>> recordsIndividual, Dictionary<string, List<string>> attributesDict, List<string> headers)

Metóda používa slovník atribútov a hodnôt na prevod reťazcových hodnôt jednotlivých záznamov na číselné. Ako vstupy uvádzame zoznam jednotlivých záznamov, predtým vytvorený slovník a zoznam hlavičiek. Ako výstup dostaneme zoznam, ktorý obsahuje jednotlivé záznamy(zoznam), ktoré obsahujú číselné hodnoty atribútov.

Napríklad:

4 | Female | 61-80 | Yes | No | High | never smoked | Yes |

Prevedie sa na:

4 | 2 | 1 | 2 | 2 | 3 | 2 | 1 |

* + - 1. public static List<List<int>> GetRecordsIndividualNumerical(List<List<string>> recordsString, Dictionary<string, List<string>> attributesDict, List<string> headers)

Užitočná metóda na transformáciu zoznamu, ktorý obsahuje stĺpce - atribúty súboru údajov, na zoznam s jednotlivými záznamami, ktoré obsahujú číselné hodnoty, v jednom volaní. Metóda len využíva predchádzajúce metódy TransformRecordsIntoRecordsIndividual a ConvertRecordsToInt.

Ako výstup dostaneme zoznam, ktorý obsahuje jednotlivé záznamy(zoznam), ktoré obsahujú číselné hodnoty atribútov.

* + - 1. public static List<List<int>> GetPositiveRecords(List<List<int>> recordsNumerical, Dictionary<string, List<string>> attributesDict, List<string> headers, string posAttributeName, string posAttributeValue)

Metóda extrahuje všetky pozitívne záznamy z daného zoznamu záznamov. Ako vstup uvádzame zoznam záznamov, slovník hodnôt atribútu, názov reťazca cieľového atribútu a kladnú hodnotu reťazca atribútu. Jednoducho len prejde každý záznam a vezme každý, ktorý má cieľovú hodnotu. Návratová hodnota je zoznam pozitívnych záznamov.

* + - 1. public static List<List<int>> GetNegativeRecords(List<List<int>> recordsNumerical, Dictionary<string, List<string>> attributesDict, List<string> headers, string posAttributeName, string posAttributeValue)

Funguje takmer identicky ako metóda 2.1.1.7, ale berie len záznamy, ktoré neobsahujú cieľovú hodnotu. Vráti zoznam zaznamenaných záznamov.

* + 1. Data

Trieda Data je zodpovedná hlavne za volanie špecifických metód Preprocesora na danej dátovej množine a uloženie výsledku do inštancie. V podstate po zavolaní správneho konštruktora alebo metódy ReadDataset musíte mať objekt Data, ktorý je pripravený na prácu s algoritmom. Obsahuje tiež metódy pre formátované zobrazenie uložených hodnôt v konzole.

Súkromné ​​atribúty triedy:

* string datasetPath – reťazec cesty k súboru údajov.
* List<List<string>> recordsString – Zoznam záznamov s reťazcovymi hodnotami.
* string targetAttributeName – Reťazec názovu cieľového atribútu.
* string targetAttributeValue – Cieľová hodnota reťazca cieľového atribútu.

Verejné atribúty triedy:

* List<string> Headers – Zoznam hlavičiek súboru údajov.
* Dictionary<string, List<string>> AttributesDict – Slovník hodnôt atribútov.
* List<List<int>> Records – Zoznam záznamov s číselnými hodnotami.
* List<List<int>> PositiveRecords – Zoznam pozitívnych záznamov.
* List<List<int>> NegativeRecords – Zoznam negatívnych záznamov.
* int NumberOfColumns – Počet stĺpcov v množine údajov + stĺpec id.
* int NumberOfRecords – Počet záznamov v množine údajov.

Metódy:

* + - 1. public Data(string datasetPath, string targetAttributeName, string targetAttributeValue)

Konštruktor pre trénovacie dáta. Ako vstupy berie cestu k súboru údajov, názov cieľového atribútu a cieľovú hodnotu atribútu. Volá metódu 2.1.2.3 ReadDataset.

* + - 1. public Data(string datasetPath, string targetAttributeName, string targetAttributeValue, Dictionary<string, List<string>> attributesDict)

Konštruktor pre vyhodnocovacie dáta. Ako vstupy berie cestu k množine údajov, názov cieľového atribútu, cieľovú hodnotu atribútu a špecifický slovník atribútov, ktorý musí použiť na konverziu údajov z reťazcov na číselné. Volá metódu 2.1.2.4 ReadDatasetWithExistingDictionary.

* + - 1. public void ReadDataset()

Volá metódy preprocesora a ukladá výsledky do inštancie Data. Vytvára svoj vlastný slovník atribútov a používa sa na čítanie trenovacich súborov údajov. Tiež ukladá počet atribútov a záznamov súboru údajov.

* + - 1. public void ReadDatasetWithExistingDictionary()

Má podobnú funkčnosť ako predchádzajúca metóda 2.1.2.3, ale nevytvára svoj vlastný slovník, ale používa existujúci daný slovník na prevod hodnôt reťazcov na číselné.

* + - 1. Metódy zobrazenia údajov do konzoly:

Jednoduché metódy, ktoré sa používaju na zobrazenie rôznych uložených hodnôt inštancie údajov v konzole. Veľmi užitočné pre fázu vývoja a ladenia.

* public void DisplayHeaders() - Metóda zobrazuje zoznam hlavičiek aktuálne uložených v inštancii.
* public void DisplayRecordsString() - Metóda zobrazí zoznam záznamov s reťazcovymi hodnotami aktuálne uloženými v inštancii.
* public void DisplayNumericalRecords(List<List<int>> records) - Metóda zobrazí zoznam záznamov s číselnými hodnotami, zadaný ako parameter.
* public void DisplayNumericalRecords() - Metóda zobrazí zoznam záznamov s číselnými hodnotami, ktorý je aktuálne uložený v inštancii.
* public void DisplayPositiveRecords() - Metóda zobrazí zoznam pozitívnych záznamov s číselnými hodnotami, ktorý je aktuálne uložený v inštancii.
* public void DisplayNegativeRecords() - Metóda zobrazí zoznam negatívnych záznamov s číselnými hodnotami, ktorý je aktuálne uložený v inštancii.
  1. AQ11

Trieda AQ11 je trieda, ktorá predstavuje algoritmus.

Verejné atribúty triedy:

* public int NumberOfColumns – Počet atributov v trenovacich datoch.
* int NumberOfRecords – Počet záznamov v trenovacich datoch.
* public Data LocalData – Inštancia trénovacich Data uložená v inštancii algoritmu.
* public Data EvaluationData – Inštancia Data pre hodnotenie algoritmu uložená v inštancii algoritmu.
* public List<List<List<int?>>> fullStar – Zoznam, ktorý predstavuje celú obálku G, (každý pozitívný záznam voči každemu negatívnemu záznamu). Obálka obsahuje negácie hodnôt atribútov.
* public List<List<List<List<int?>>>> PositiveFullStar – Zoznam, ktorý predstavuje celú obálku G, (každý pozitívný záznam voči každemu negatívnemu záznamu). Obálka obsahuje pozitivne hodnôty atribútov.
* public int NumberOfTP – Atribút obsahuje počet True Positive prípadov, uložený po vyhodnotení.
* public int NumberOfFP – Atribút obsahuje počet False Positive prípadov, uložený po vyhodnotení.
* public int NumberOfFN – Atribút obsahuje počet False Negative prípadov, uložený po vyhodnotení.
* public int NumberOfTN – Atribút obsahuje počet True Negative prípadov, uložený po vyhodnotení.
* public float Precision – Atribút obsahuje vypočítané Precision po vyhodnotení.
* public float Recall – Atribút obsahuje vypočítané Recall po vyhodnotení.
* public float F1 – Atribút obsahuje vypočítané F1 po vyhodnotení.
* public float Accuracy – Atribút obsahuje vypočítané Accuracy po vyhodnotení.

Metódy:

* + - 1. public AQ11() a public AQ11(Data data)

Konštruktory inštancie algoritmu. Druhý berie ako vstup trovacie dáta v inštancii Data. Volá metódu 2.2.1.2 SetData.

* + - 1. public void SetData(Data data)

Metóda berie inštanciu Data ako vstup a ukladá ju do inštancie AQ11 ako trenovaciu. Tiež vypočíta počet atribútov a počet záznamov a uloží ich.

* + - 1. public void SetEvaluationData(Data data)

Metóda berie inštanciu dát ako vstup a ukladá ju do inštancie AQ11 ako vyhodnocovacie dáta.

* + - 1. public void ApplyAlgorithmOnData(bool display=false)

Metóda, ktorá aplikuje algoritmické metódy na uložené trénovacie dáta. Výsledné fullStar a PositiveFullStar sú uložené v inštancii AQ11. Hodnota parametru display určuje, či sa má výsledné pravidlo zobraziť v konzole.

* + - 1. public void ApplyAlgorithmOnData(Data data, bool display=false)

Funguje rovnako ako predchádzajúca metóda 2.2.1.4, ale teraz s trénovacimi datmi, ktoré sú uvedené ako parameter. Hodnota parametru display určuje, či sa má výsledné pravidlo zobraziť v konzole.

* + - 1. public void DisplayResultingRule()

Metóda používa metódu DisplayPositiveFullStarAsRule a dáva jej uložený PositiveFullStar ako vstupný parameter. Metóda zobrazuje PositiveFullStar ako pravidlo v konzole, ktoré je pre človeka zrozumiteľná.

* + - 1. public List<int?> CreatePartialStarDisjunction(List<int> positiveRecord, List<int> negativeRecord)

Metóda na vytvorenie čiastočnej obálky-disjunkcie (obálka G - pozitívny príklad voči kontrapríkladu) z jedného pozitívneho záznamu a jedného negatívneho záznamu. Vstupy sú jeden pozitívny záznam a jeden negatívny záznam. Výstupom je čiastočná obálka-disjunkcia. Disjunkcia neobsahuje žiadne informácie o ID záznamov alebo cieľovej hodnote.

**Príklad**

Pozitívny záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Negatívny záznam: (| 21 | 1 | 2 | 2 | 2 | 3 | 1 | 2 |)

Výsledná disjunkcia: (| -1 | -2 | -2 | | -3 | |) => (Atribút0: nie 1) ALEBO (Atribút1: nie 2) ALEBO (Atribút2: nie 2) ALEBO (Atribút4: nie 3)

* + - 1. public void DisplayPartialStarDisjunction(List<int?> partialStarDisjunction)

Metóda zobrazenia danej ako parameter obálky (obálka G - pozitívny príklad voči kontrapríkladu).

* + - 1. public List<List<int?>> CreatePartialStarConjunction(List<int> positiveRecord, List<List<int>> negativeRecords)

Metóda na vytvorenie čiastočnej obálky-konjunkcie (obálka G - pozitívny príklad voči všetkým kontrapríkladom) z jedného pozitívneho záznamu a všetkych negatívnych záznamov. V podstate používa metódu 2.2.1.7 CreatePartialStarDisjunction na každý negatívny záznam a ukladá výsledky do zoznamu. Výstupom je čiastočná obálka-konjunkcia.

**Príklad**

Pozitívny záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Negatívne záznamy:

(| 21 | 1 | 2 | 2 | 2 | 3 | 1 | 2 |)

(| 29 | 1 | 4 | 1 | 2 | 2 | 3 | 2 |)

Výsledná konjunkcia: (| -1 | -2 | -2 | | -3 | |), (| -1 | | | | | -3 |) => [ (Atribút0: nie 1) ALEBO (Atribút1: nie 2) ALEBO (Atribút2: nie 2) ALEBO (Atribút4: nie 3) ] AND [ (Atribút0: nie 1) ALEBO (Atribút5: nie 3) ]

* + - 1. public void DisplayPartialStarConjunction(List<List<int?>> partialStarConjunction)

Metóda zobrazenia danej ako parameter obálky (obálka G - pozitívny príklad voči všetkým kontrapríkladom).

* + - 1. private bool CanBeAbsorbed(List<int?> primaryStar, List<int?> secondaryStar)

Metóda na kontrolu, či sekundárna obláka môže byť absorbovaná pomocou primárnej obálky pri použití absorpčného zákona. Obálky sú uvedené ako parametre. (obálka G - pozitívny príklad voči kontrapríkladu). Výstup je boolovská hodnota.

**Príklad 1**

Primárna obálka: (| -1 | | | | | -3 |)

Sekundárna obálka: (| -1 | -2 | -2 | | -3 | |)

Výsledok: False

**Príklad 2**

Primárna obálka: (| -1 | | -2 | | -3 | |)

Sekundárna obálka: (| -1 | -2 | -2 | | -3 | |)

Výsledok: True

* + - 1. public List<List<int?>> ApplyAbsorptionLawOnConjunction(List<List<int?>> partialStarConjucntion)

Metóda aplikácie absorpčného zákona na čiastočnú obálku-konjunkciu. Metóda odstraňuje akúkoľvek čiastočnú obálku-disjunkciu, ktorá môže byť absorbovaná z čiastočnej obálky-konjunkcie. Používa metódu 2.2.1.11 CanBeAbsorbed s každou kombináciou disjunkcií na kontrolu, či je možné sekundárnu disjunkciu absorbovať. Výstupom je zmenená čiastočná obálka-konjunkcia.

**Príklad**

Obálka-konjunkcia: (| -1 | -2 | -2 | | -3 | |), (| -1 | | | | | -3 |), (| -1 | | -2 | | -3 | |)

Výsledná obálka-konjunkcia: (| -1 | | | | | -3 |), (| -1 | | -2 | | -3 | |)

Vysvetlenie: obálka (| -1 | -2 | -2 | | -3 | |) bola absorbovana obálkou (| -1 | | -2 | | -3 | |) ako bolo preukázané v príklade 2 metódy 2.2.1.11.

* + - 1. public bool IsRecordCoveredByDisjunction(List<int> record, List<int?> disjunction)

Metóda kontroly, či konkrétny daný záznam je pokrytý danou čiastočnou obálkou-disjunkciou. Vstupy sú špecifický záznam a disjunkcia, ktorú chceme skontrolovať. Výstup je boolovská hodnota.

**Príklad 1**

Záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Obálka: (| -1 | -2 | -2 | | -3 | |)

Výsledok: True

**Príklad 2**

Záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Obálka: (| -2 | -4 | -2 | | -3 | |)

Výsledok: False

* + - 1. public void DisplayNotCoverage(List<int> record, List<int?> disjunction)

Metóda zobrazenia konkrétneho daného záznamu, ktorý nebol pokrytý danou obálkovou disjunkciou. Ako vstupy berie jeden záznam a jednu disjunkciu. Hlavne veľmi užitočné pri ladení.

* + - 1. public bool IsRecordCoveredByConjunction(List<int> record, List<List<int?>> conjunction)

Metóda kontroly, či konkrétny daný záznam je pokrytý danou čiastkovou obálkou-konjunkciou. Metóda v podstate iteruje konjunkciou (zoznam disjunkcii) a pomocou metódy 2.2.1.13 IsRecordCoveredByDisjunction kontroluje, či záznam pokrýva každá disjunkcia. Vstupy sú jeden záznam a obálka-konjunkcia. Výstup je boolovská hodnota.

* + - 1. public List<List<int>> SelectCoveredRecords(List<List<int>> records, List<List<int?>> conjunction)

Metóda na výber záznamov, ktoré sú pokryté obálkou-konjunkciou, zo zoznamu záznamov. Pokryté záznamy sú uložené v druhom zozname. Vstupmi sú zoznam záznamov a obálka-konjunkcia. Výstupom je zoznam pokrytých záznamov. V podstate používa metódu 2.2.1.15 IsRecordCoveredByConjunction na každý záznam v zozname a berie len tie, ktoré sú pokryté.

* + - 1. public List<List<List<int?>>> CreateFullStarDisjunction(List<List<int>> positiveRecords, List<List<int>> negativeRecords)

Metóda na vytvorenie plnej obálky-disjunkcie. (obálka G - všetky pozitívne príklady voči všetkym kontrapríkladom). Vstupmi sú zoznam pozitívnych záznamov a zoznam negatívnych záznamov. Výstup je plna obálka-disjunkcia. Metóda vytvorí spojku pre každý záznam, ktorý nie je pokrytý žiadnou zo obálok v zozname, kým nie sú všetky záznamy pokryté zoznamom obálok (zoznam obálok v tomto pripade je obálka G - všetky pozitívne príklady voči všetkym kontrapríkladom).

* + - 1. public void DisplayFullStarDisjunction(List<List<List<int?>>> fullStar)

Metóda zobrazenia plnej obálky-disjunkcie (vystupnej z methody 2.2.1.17).

* + - 1. public List<List<int?>> TransformNegationsIntoDisjunctions(List<int?> negations)

Metóda transformuje každú negáciu v čiastočnej obál-disjunkcii na disjunkcie obsahujúce pozitivne hodnoty. Vstup je disjunkcia (zoznam), ktorá obsahuje záporné hodnoty. Výstupom je zoznam samostatných pozitívnych disjunkcií pre každý atribút.

**Príklad**

Obálka: (| -2 | | -3 |)

A napríklad pre atribút 0 sú jediné možné hodnoty 1,2. A pre atribút 2 možné hodnoty sú 1,2,3,4.

To znamená, že pre atribút 0 sa hodnota -2 premení na disjunkciu (zoznam), ktorá bude obsahovať jednu hodnotu: (| 1 |).

Pre atribút 1 disjunkcia (zoznam) bude prázdna: ( ).

Pre atribút 2 disjunkcia (zoznam) bude obsahovať všetky kladné hodnoty okrem tej, ktorá bola negovaná. Disjunkcia (zoznam) bude vyzerať takto: (| 1 | 2 | 4 |).

* + - 1. public void DisplayPositiveDisjunctions(List<List<int?>> disjunctions)

Metóda na zobrazenie zoznamu pozitívnych disjunkcií. Vstup je disjunkcia (zoznam), pričom každý prvok je disjunkcia (zoznam) s pozitivnymi prvkami.

* + - 1. public List<List<List<int?>>> TransformNegativeConjunctionIntoPositive(List<List<int?>> conjucntion)

Metóda, ktorá používa metódu 2.2.1.19 TransformNegationsIntoDisjunctions na transformáciu všetkych disjunkcii s negativnymi hodnotami v spojke na disjunkcie s pozitivnymi hodnotami. Vstup je konjunkcia (zoznam zoznamov), ktoré chceme transformovať.

* + - 1. public void DisplayPositiveConjunction(List<List<List<int?>>> positiveConjucntion)

Metóda zobrazenia konjunkcie s pozitivnymi disjunkciami v konzole. Ako vstup používa konjunkciu (zoznam disjunkcií s kladnými prvkami).

* + - 1. public List<List<List<List<int?>>>> TransformFullStarToNumericalPositiveFullStar(List<List<List<int?>>> fullStar)

Metóda transformácie prvkov obálky-disjunkcie na pozitívne. Výstup je plná obálka-disjunkcia iba s pozitivnymi hodnotami. Vstup je fullStar (plna obálka ako vystup z metódy 2.2.1.17 CreateFullStarDisjunction) ktorá obsahuje iba negatívne hodnoty.

1. Popis dát

V tejto kapitole sa rozoberá súčasný stav problému ktorý riešime v záverečnej práci. Sú to rôzne východzie stavy, vstupné hodnoty, aktuálne výstupné hodnoty, podmienky prevádzky, dátový model, aktuálne kalkulácie, nosné vzorce a prepočty hodnôt a premenných.

Čiže ide o podrobný opis skúmanej témy, jej všetkých parciálnych častí, opis jednotlivých tokov informácií medzi časťami systému, podniku, obchodu, aplikácie a podobne.

Ďalej je možné opísať aké funkčné prostriedky sú zvolené pre dosiahnutie riešenia problému, ale v aktuálnom resp. v súčasnom systéme.

1. Vyhodnotenie

Táto kapitola ja jadrom záverečnej práce a rozsahom by mala byť tretinou z celkového počtu strán.

Na základe poznatkov z predošlých kapitol, by táto kapitola mala obsahovať vlastný návrh riešenia problematiky alebo systému.

Čiže, je potrebné navrhnúť riešenia čiastkových problémov ako sú:

* model správania podniku, aplikácie,
* vnútorné toky informácií,
* ak ide o informačné systémy tak aj databázový model, user case diagram a podobne,
* a taktiež aj celkový popis ako sa vyrieši zadanie práce.

V práci je možné použiť rôzne vzorce, obrázky, tabuľky, ale aj krížové odkazy na literatúru alebo webový obsah a záverom aj zoznam použitej literatúry.

Záver

Záver by mal zachytiť jasnú a presnú prezentáciu dedukcií vychádzajúcich z jadra práce. Musí byť vecnou sumarizáciou vlastného prínosu alebo pohľadu na riešenú problematiku. Zahrnúť možno aj kvantitatívne údaje, ale podrobnosti by sa nemali uvádzať. Záver nemá obsahovať nič, čo nie je v texte práce a musí nadväzovať na úvahy a argumenty v texte práce.

V závere je vhodné poukázať na ďalšie otvorené (doteraz nevyriešené) problémy, ktorým je vhodné venovať pozornosť a ktoré presahujú odporúčaný rozsah práce. Odporúčané sú popisy ďalších navrhovaných aktivít, ktoré priamo vyplývajú zo záverov alebo skúseností získaných v priebehu spracovania práce.

Zoznam použitej literatúry

1. Wojtusiak, J. (2012). AQ Learning. In: Seel, N.M. (eds) Encyclopedia of the Sciences of Learning. Springer, Boston, MA. <https://doi.org/10.1007/978-1-4419-1428-6_845> ISO 690-2: 1997, Information and documentation – Bibliographic references - Part 2: Electronic documents or parts thereof.
2. STN ISO 690:1998 : Dokumentácia - Bibliografické odkazy - Obsah, forma a štruktúra.
3. Zákon č. 183/2000 Z.z. o knižniciach, o doplnení zákona Slovenskej národnej rady č. 27/1987 Zb. o štátnej pamiatkovej starostlivosti a o zmene a doplnení zákona č. 68/1997 Z.z. o Matici slovenskej.
4. Vyhláška č. 131/1997 Zb. Ministerstva školstva Slovenskej republiky zo 7. mája 1997 o doktorandskom štúdiu.
5. LAGOZE, C. a kol. The Open Archives Initiative Protocol for Metadata Harvesting [online]. Protocol Version 2.0 of 2002-06-14. Document Version 2004/10/12T15:31:00Z 2004 [cit. 2004-11-10]. Dostupné na internete:

<http://www.openarchives.org/OAI/openarchivesprotocol.html>.

Prílohy

Príloha A: CD médium – záverečná práca v elektronickej podobe, prílohy v elektronickej podobe.

Príloha B: Používateľská príručka

Príloha C: Systémová príručka

Táto časť diplomovej práce je povinná a obsahuje zoznam všetkých príloh vrátané elektronických nosičov. Názvy príloh v zozname musia byt’ zhodné s názvami uvedenými na príslušných prílohách. Tlačené prílohy majú na prvej strane identifikačné údaje – informácie zhodné s titulnou stranou diplomovej práce doplnené o názov príslušnej prílohy (Systémová príručka, Používateľská príručka). Identifikačné údaje sú aj na priložených diskoch alebo disketách. Ak je médií viac, sú označené aj číselne v tvare I/N, kde I je poradové číslo a N je celkový počet daných médií.

Každá príloha začína na novej strane a je označená samostatným písmenom (Príloha A, Príloha B, ...). Číslovanie strán príloh nadväzuje na číslovanie strán v hlavnom texte.

CD je spravidla grafické s logom univerzity a fakulty. Pozri . Tieto CD robia v Univerzitnej knižnici TUKE.



Obr. 7 Obrázok grafického CD média (vzor)