ממ"ן 14

שאלה 1.א.

בייצוג הראשון, קבוצת המשתנים היא שלושת המילים במאוזן ושלושת המילים במאונך. זאת אומרת: .

בייצוג השני, קבוצת המשתנים היא התאים של התשבץ, שנתונות על ידי זוגות של שורה ועמודה. יש 9 תאים- , כאשר .

שאלה 1.ב.

להלן גרף האילוצים של הבעיה:

A1

A2

A3

D1

D2

D3

שאלה 1.ג.

ניתן להגדיר את הבעיה בשני אופנים:

1. התחום של כל משתנה הוא קבוצת ל המילים הנתונות. במקרה זה אין אילוצים אונריים, כי כל משתנה לבדו יכול לקבל כל אחד מהמילים- הבעיה רק מתעוררת כאשר יש שני משתנים שמתנגשים.
2. התחום של כל משתנה הוא כל מילה. במקרה זה לכל משתנה יש את האילוץ האונרי שהוא יכול לקבל מילה רק מתוך קבוצת המילים הנתונה.

שאלה 1.ד.

לכל מילה בתחום של , כל אות במילה חייבת להיות ההתחלה של מילה אחרת. זה נובע מהאילוצים הבינארים של התשבץ. נדגים זאת בכמה מקרים:

אם , אז התחום של D2 הוא כל המילים שמתחילות באות d, וזהו תחום ריק. לכן add ירד מהתחום של A1.

אם , אז התחום של A3 הוא כל המילים שמתחילות באות g, וזהו תחום ריק. לכן bag ירד מהתחום של D1.

לאחר הפעלת כל האילוצים מהסוג הזה, נקבל שהתחום של הוא

לכל מילה בתחום של , כל אות במילה חייבת להיות האמצע של מילה אחרת. זה נובע מהאילוצים הבינארים של התשבץ. נדגים זאת בכמה מקרים:

אם , אז התחום של D3 הוא כל המילים שבאמצעיתן האות h, וזהו תחום ריק. לכן ash ירד מהתחום של A2.

אם , אז התחום של A1 הוא כל המילים שבאמצעיתן האות b, וזהו תחום ריק. לכן bat ירד מהתחום של D2.

לאחר הפעלת כל האילוצים מהסוג הזה, נקבל שהתחום של הוא

לכל מילה בתחום של , כל אות במילה חייבת להיות הסוף של מילה אחרת. זה נובע מהאילוצים הבינארים של התשבץ. נדגים זאת בכמה מקרים:

אם , אז התחום של D2 הוא כל המילים שבסופן האות y, וזהו תחום ריק. לכן aye ירד מהתחום של A3.

אם , אז התחום של A1 הוא כל המילים שבסופן האות b, וזהו תחום ריק. לכן bad ירד מהתחום של D3.

לאחר הפעלת כל האילוצים מהסוג הזה, נקבל שהתחום של הוא

נחזור ונפעיל את האילוצים של ונקבל תחום מצומצם של

נחזור ונפעיל את האילוצים של ונקבל תחום מצומצם של

כעת התחום של הוא . מטעמי סימטריות זה לא משנה איזה ערך נותנים ל-A2 ואיזה ל-D2, אז נניח ללא הגבלת כלליות ש-.

כעת התחום של A1 הוא , והתחום של A3 הוא . נעדכן את הערכים שלהם, נפעיל אילוצים בינאריים ונקבל שהתחום של D1 הוא , והתחום של D3 הוא . אם כן, פיתרון הוא התשבץ הוא:

|  |  |  |  |
| --- | --- | --- | --- |
| D3 | D2 | D1 |  |
| a | o | b | A1 |
| r | a | e | A2 |
| t | f | e | A3 |

מש"ל.

שאלה 1.ה.

להלן התחלת וסוף דיבאג של חיפוש backtrack עם היוריסטיקות MRV, יוריסטיקת הדרגה, LCV. המשמעות של עצירת בדיקה מסוימת היא שנמצא תחום ריק עבור אחד מהמשתנים. התוכנה בjava נמצאת בסוף הממ"ן, כנספח.

trying D3:add

trying A1:boa

trying D2:oaf

Error: Domain of A3 is empty

trying D3:ado

trying D2:add

Error: Domain of A1 is empty

trying D2:ado

Error: Domain of A1 is empty

trying D2:age

Error: Domain of A1 is empty

trying D2:ago

Error: Domain of A1 is empty

trying D2:aid

Error: Domain of A1 is empty

trying D2:ail

Error: Domain of A1 is empty

trying D2:aim

Error: Domain of A1 is empty

trying D2:air

Error: Domain of A1 is empty

trying D2:and

Error: Domain of A1 is empty

trying D2:any

Error: Domain of A1 is empty

trying D2:ape

Error: Domain of A1 is empty

trying D2:apt

Error: Domain of A1 is empty

.

.

.

trying D3:art

trying A1:boa

trying D2:oaf

trying A3:eft

trying D1:bee

trying A2:ear

כפי שניתן לראות, הרצת חיפוש backtracking הגיע לאותו מסקנה שאליו הגיע הפעלת אלגוריתם עקביות הקשת.

מש"ל.

שאלה 2.א.

נגדיר את היחסים הבאים: .

אם כן, ידועים לנו כי הפסוקים הבאים נכונים:

שאלה 2.ב.

אנו צריכים להוכיח כי הפסוק אמיתי.

נסמן

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל .*

*נתבונן בזוג הפסוקיות ו-. נאחד אותם ונקבל פסוק ריק.*

*מכאן ש- ספיק מתוך .*

*מש"ל.*

*שאלה 3.א.*

*ע"פ יוריסטיקת unit-clause, ניתן לפשט את כל הפסוקים ע"פ פסוק 5 (כלומר, לתת ל-S ערך של אמת), ולהוריד מהם את כל המופעים של .*

*כעת, ע"פ יוריסטיקת unit-clause, ניתן לפשט את כל הפסוקים ע"פ פסוק 6 (כלומר, לתת ל-W ערך של שקר), ולהוריד מהם את כל המופעים של .*

*כעת, ע"פ יוריסטיקת unit-clause, ניתן לפשט את כל הפסוקים ע"פ פסוק 3 (כלומר, לתת ל-R ערך של שקר), ולהוריד מהם את כל המופעים של .*

*כעת, ע"פ יוריסטיקת unit-clause, ניתן לפשט את כל הפסוקים ע"פ פסוק 2 (כלומר, לתת ל-Q ערך של שקר), ולהוריד מהם את כל המופעים של .*

*כעת, ע"פ יוריסטיקת unit-clause, ניתן לפשט את כל הפסוקים ע"פ פסוק 1 (כלומר, לתת ל-P ערך של אמת), ולהוריד מהם את כל המופעים של .*

*כעת נשארו רק הפסוקים 4,7, שכעת נראים כך:*

*שאלה 3.ב.*

*פתרון הבעיה המקורית מצריכה מציאת הערך של כל אחד מהמשתנים . כלומר, יש אפשרויות, ולכן זהו מספר המצבים במרחב החיפוש.*

*בעזרת היוריסטיקות, הורדנו את בעיה לבעיה עם 2 משתנים, . כלומר, כעת יש רק אפשרויות, כלומר מצבים במרחב החיפוש.*

*מש"ל.*

*שאלה 4*

1. *המאחד הכללי ביותר הוא .*
2. *לא ניתן לבצע האחדה, כי הביטוי הימיני דורש ששני הפרמטרים של In יהיו זהים, והביטוי בצד שמאל דורש שפרמטר אחד יהיה office-of של השני.*
3. *המאחד הכללי ביותר הוא .*
4. *המאחד הכללי ביותר הוא .*
5. *המאחד הכללי ביותר הוא .*
6. *לא ניתן לבצע האחדה, כי הביטוי הימיני דורש ששני הפרמטרים של F יהיו זהים, והביטוי בצד שמאל דורש שפרמטר אחד יהיה F של עצמו ו-A.*

*מש"ל.*

*שאלה 5*

*ידועים לנו העובדות הבאות:*

*בCNF, העובדות הם:*

*אנו רוצים להוכיח בעזרת רזולוציה כדי לגלות מי הרוצח. מכיוון שהבעיה מוצגת בתחשיב הפסוקים, אין דרך לבטא את כך שמישהו הרג את דני, ולכן נצטרך לבדוק עבור כל בן אדם אם הוא רוצח.*

*נתחיל בגיא: נוסיף את הפסוק ל-KB, ונבצע רזולוציה:*

*בעזרת יוריסטיקת unit-clause, נבחר לחפש את כל הדסיונקצות שמתאחדות עם .*

*לא הצלחנו להגיע לסתירה.*

*ננסה עם דודי. נוסיף את הפסוק ל-KB, ונבצע רזולוציה:*

*בעזרת יוריסטיקת unit-clause, נבחר לחפש את כל הדסיונקצות שמתאחדות עם .*

*ניתן לראות שהגענו לסתירה. לכן הפסוק לא היה נכון. כלומר, דודי רצח את דן.*

*מש"ל.*

נספח:

**import** java.util.\*;

**public** **class** Main {

**static** String[] *assignment*;

**static** **final** String[] ***vars*** = { "A1", "A2", "A3", "D1", "D2", "D3" };

**static** LinkedList<String>[] *domains*;

@SuppressWarnings("unchecked")

**public** **static** **void** main(String args[]) {

*assignment* = **new** String[6];

*domains* = (LinkedList<String>[]) **new** LinkedList[6];

String[] words = { "add", "ado", "age", "ago", "aid", "ail", "aim", "air", "and", "any", "ape", "apt", "arc",

"are", "ark", "arm", "art", "ash", "ask", "auk", "awe", "awl", "aye", "bad", "bag", "ban", "bat", "bee",

"boa", "ear", "eel", "eft", "far", "fat", "fit", "lee", "oaf", "rat", "tar", "tie" };

**for** (**int** i = 0; i < 6; i++) {

*domains*[i] = **new** LinkedList<String>(Arrays.*asList*(Arrays.*copyOf*(words, words.length)));

}

**if** (*backtrack*(0)) {

**for** (**int** i = 0; i < 6; i++)

System.***out***.println(***vars***[i] + ":" + *assignment*[i]);

} **else**

System.***out***.println("failed");

}

**public** **static** **boolean** backtrack(**int** indent) {

**boolean** done = **true**;

**for** (**int** i = 0; i < 6; i++)

**if** (*assignment*[i] == **null**)

done = **false**;

**if** (done)

**return** done;

**int** var = *selectVar*();

**for** (String val : *orderDomain*(var)) {

LinkedList<String>[] backup = (LinkedList<String>[]) **new** LinkedList[6];

**for** (**int** i = 0; i < 6; i++) {

backup[i] = **new** LinkedList<String>(*domains*[i]);

}

**if** (*checkConsistency*(var, val)) {

String output = "";

**for** (**int** i = 0; i < indent; i++)

output += "\t";

System.***out***.println(output + "trying " + ***vars***[var] + ":" + val);

*assignment*[var] = val;

**if** (*inferences*(var,indent+1)) {

**if** (*backtrack*(indent + 1))

**return** **true**;

}

}

*assignment*[var] = **null**;

*domains*=backup;

}

**return** **false**;

}

@SuppressWarnings("unchecked")

**private** **static** **boolean** inferences(**int** var, **int** indent) {

// return true if inferences are okay

**if** (var < 3) {

LinkedList<String>[] newdomains = (LinkedList<String>[]) **new** LinkedList[3];

**for** (**int** i = 0; i < 3; i++) {

newdomains[i] = **new** LinkedList<String>();

}

**for** (**int** i = 3; i < 6; i++) {

**if**(*assignment*[i]!=**null**)

**continue**;

**for**(String str : *domains*[i])

{

**if**(str != *assignment*[var] && str.charAt(var) == *assignment*[var].charAt(i - 3))

newdomains[i-3].add(str);

}

**if**(newdomains[i-3].size()==0)

{

String output = "";

**for** (**int** j = 0; j < indent; j++)

output += "\t";

System.***out***.println(output+"Error: Domain of "+***vars***[i]+" is empty");

**return** **false**;

}

}

**for**(**int** i=3;i<6;i++)

**if**(*assignment*[i]==**null**)

*domains*[i]=newdomains[i-3];

} **else**{

LinkedList<String>[] newdomains = (LinkedList<String>[]) **new** LinkedList[3];

**for** (**int** i = 0; i < 3; i++) {

newdomains[i] = **new** LinkedList<String>();

}

**for** (**int** i = 0; i < 3; i++) {

**if**(*assignment*[i]!=**null**)

**continue**;

**for**(String str : *domains*[i])

{

**if**(str != *assignment*[var] && str.charAt(var-3) == *assignment*[var].charAt(i))

newdomains[i].add(str);

}

**if**(newdomains[i].size()==0)

{

String output = "";

**for** (**int** j = 0; j < indent; j++)

output += "\t";

System.***out***.println(output+"Error: Domain of "+***vars***[i]+" is empty");

**return** **false**;

}

}

**for**(**int** i=0;i<3;i++)

**if**(*assignment*[i]==**null**)

*domains*[i]=newdomains[i];

}

**return** **true**;

}

**private** **static** LinkedList<String> orderDomain(**int** var) {

**return** *domains*[var];// this is fake

}

**private** **static** **boolean** checkConsistency(**int** var, String val) {

// maybe enforce arc-consistency!

**if** (var < 3) {

**for** (**int** i = 3; i < 6; i++) {

**if** (*assignment*[i] != **null**) {

**if** (*assignment*[i] == val || *assignment*[i].charAt(var) != val.charAt(i - 3))

**return** **false**;

}

}

} **else**

**for** (**int** i = 0; i < 3; i++)

**if** (*assignment*[i] != **null**)

**if** (*assignment*[i] == val || *assignment*[i].charAt(var - 3) != val.charAt(i))

**return** **false**;

**return** **true**;

}

**private** **static** **int** selectVar() {

**int** mini = -1;

**boolean** unique = **true**;

**for** (**int** i = 0; i < 6; i++) {

**if** (*assignment*[i] != **null**)

**continue**;

**if** (mini == -1 || *domains*[i].size() < *domains*[mini].size()) {

mini = i;

unique = **true**;

} **else** **if** (mini != -1 && *domains*[i].size() == *domains*[mini].size())

unique = **false**;

}

**if** (mini != -1 && unique)

**return** mini;

mini = -1;

**int** mincount = 0, count;

**for** (**int** i = 0; i < 6; i++) {

**if** (*assignment*[i] != **null**)

**continue**;

count = 0;

**if** (i < 3) {

**for** (**int** j = 3; j < 6; j++) {

**if** (*assignment*[j] == **null**)

count++;

}

} **else** {

**for** (**int** j = 0; j < 3; j++) {

**if** (*assignment*[j] == **null**)

count++;

}

}

**if** (count >= mincount) {

mini = i;

mincount = count;

}

}

**return** mini;

}

}