**מגישות:**

אנה קיסנר 319652442

מור מילנדר 205983521

**חלק ראשון**

**מה זה בינה מלאכותית:**

בינה מלאכותית היא ענף של [מדעי המחשב](https://he.wikipedia.org/wiki/%D7%9E%D7%93%D7%A2%D7%99_%D7%94%D7%9E%D7%97%D7%A9%D7%91) העוסק ביכולתנו לתכנת [מחשבים](https://he.wikipedia.org/wiki/%D7%9E%D7%97%D7%A9%D7%91) לפעול באופן המציג יכולות שאפיינו עד כה את הבינה האנושית בלבד. הגדרה דומה לתחום זה ניתנה על ידי [מרווין מינסקי](https://he.wikipedia.org/wiki/%D7%9E%D7%A8%D7%95%D7%95%D7%99%D7%9F_%D7%9E%D7%99%D7%A0%D7%A1%D7%A7%D7%99" \o "מרווין מינסקי): "לגרום למכונה להתנהג בדרך שהייתה נחשבת לאינטליגנטית לו אדם התנהג כך". העוסקים בבינה מלאכותית מבכרים לעיתים הגדרה מצמצמת של מושג זה, כאילו בינה מלאכותית היא "כל מה שאדם יודע לעשות אך עדיין אין אנו יודעים לתכנת אותו". בהתאם להגדרה זו, היכולת לשחק [שחמט](https://he.wikipedia.org/wiki/%D7%A9%D7%97%D7%9E%D7%98) נחשבה ליכולת שבמסגרת הבינה המלאכותית, עד לשלב שבו הצלחנו לתכנת את המחשב כך שהתחיל לנצח.

המבחן המקובל ביותר לבינה מלאכותית הוטבעה בשנת [1950](https://he.wikipedia.org/wiki/1950) על ידי [אלן טיורינג](https://he.wikipedia.org/wiki/%D7%90%D7%9C%D7%9F_%D7%98%D7%99%D7%95%D7%A8%D7%99%D7%A0%D7%92" \o "אלן טיורינג), וידועה בשם "[מבחן טיורינג](https://he.wikipedia.org/wiki/%D7%9E%D7%91%D7%97%D7%9F_%D7%98%D7%99%D7%95%D7%A8%D7%99%D7%A0%D7%92)": מכונה תחשב לתבונית, אם יינתן לאדם, "הבוחן" היושב בחדר סגור, לנהל שיחה באמצעות ממשק מחשב (Console) עם שתי ישויות שנמצאות בחדר השני, כאשר אחת מהן תהיה מכונה והשנייה אנושית, והמשוחח לא יוכל לזהות מי משתי הישויות היא מכונה או אדם מכונה, או-אז המכונה תחשב לתבונית. תחרויות לא רשמיות, המכונות "תחרות טיורינג", נערכות מדי שנה כאשר המשתתפים מנסים להעמיד תוכנות ל"מבחן טיורינג" ואולם, הבוחנים, בדרך כלל פרופסורים מנוסים, מצליחים לזהות בקלות מי משוחח עמם. נכון לשנת 2014, הצליחה תוכנה ב-33% מהמקרים לדמות שיחה ברמה אינטלקטואלית של נער בן 13 שנים. (מעבר מוצלח של מבחן טיורינג הוגדר על ידי טיורינג, כנראה בבדיחות הדעת, כהצלחה של לפחות 30% מהמקרים).

בינה מלאכותית נחקרת בשתי רמות:

* בינה מלאכותית דמוית אנוש, שבה המחשב חושב ומסיק מסקנות באופן דומה לזה של המוח האנושי.
* בינה מלאכותית שאינה דמוית אנוש, שבה המחשב יגיע ליכולות עצמאיות לחשיבה, ללא קשר לדרך שבה המוח האנושי עושה זאת.

בינה מלאכותית מסווגת לפי ביצועים:

* [בינה מלאכותית חזקה](https://he.wikipedia.org/wiki/%D7%91%D7%99%D7%A0%D7%94_%D7%9E%D7%9C%D7%90%D7%9B%D7%95%D7%AA%D7%99%D7%AA_%D7%97%D7%96%D7%A7%D7%94): שבה התוכנה אמורה לבצע חיקוי של פעילות השכל והחשיבה, באופן כללי: פתרון בעיות כללי, וכדומה.
* בינה מלאכותית חלשה, שבה התוכנה מיועדת לבצע פעילות נבונה בתחום יישום מוגדר: משחק שח-מט, גילוי הוכחות בגאומטריה, מומחיות ספציפית, וכדומה.

מכיוון שהניסיונות ליצור בינה מלאכותית חזקה נתקעו בקשיים רבים, עיקר ההתפתחות בחקר הבינה המלאכותית, מאז שנות ה-80, הוא בתחום הבינה המלאכותית החלשה.

מחקר הבינה המלאכותית המודרני עוסק ביצירת מכונות מועילות על מנת לעשות משימות שבני אדם מבצעים, והמצריכות תבונה, באופן אוטומטי, למשל תזמון משאבים - כמו יחידות צבאיות - ביעילות המרבית, מענה על שאלות לגבי מוצרים עבור לקוחות, [הבנת דיבור](https://he.wikipedia.org/w/index.php?title=%D7%94%D7%91%D7%A0%D7%AA_%D7%93%D7%99%D7%91%D7%95%D7%A8&action=edit&redlink=1) ו[שפה](https://he.wikipedia.org/wiki/%D7%A2%D7%99%D7%91%D7%95%D7%93_%D7%A9%D7%A4%D7%94_%D7%98%D7%91%D7%A2%D7%99%D7%AA) וזיהוי [פרצופים](https://he.wikipedia.org/wiki/%D7%A4%D7%A8%D7%A6%D7%95%D7%A3) ב[מצלמות טלוויזיה](https://he.wikipedia.org/w/index.php?title=%D7%9E%D7%A6%D7%9C%D7%9E%D7%AA_%D7%98%D7%9C%D7%95%D7%95%D7%99%D7%96%D7%99%D7%94&action=edit&redlink=1) במעגל סגור. לכן בינה מלאכותית נעשתה לדיצפלינה [הנדסית](https://he.wikipedia.org/wiki/%D7%94%D7%A0%D7%93%D7%A1%D7%94), העוסקת בפתרון בעיות מעשיות. לדוגמה, על-פי שיטות של בינה מלאכותית אורגנו המשאבים הצבאיים של [ארצות הברית](https://he.wikipedia.org/wiki/%D7%90%D7%A8%D7%A6%D7%95%D7%AA_%D7%94%D7%91%D7%A8%D7%99%D7%AA) ב[מלחמת המפרץ הראשונה](https://he.wikipedia.org/wiki/%D7%9E%D7%9C%D7%97%D7%9E%D7%AA_%D7%94%D7%9E%D7%A4%D7%A8%D7%A5_%D7%94%D7%A8%D7%90%D7%A9%D7%95%D7%A0%D7%94), והכספים שנחסכו בעקבות כך עלו בהרבה על ההשקעה של הממשלה האמריקאית בתחום הבינה המלאכותית מאז תחילת המחקר בשנות ה-50. מערכות בינה מלאכותית כיום משמשות בעסקים רבים, [בתי חולים](https://he.wikipedia.org/wiki/%D7%91%D7%99%D7%AA_%D7%97%D7%95%D7%9C%D7%99%D7%9D) ויחידות [צבאיות](https://he.wikipedia.org/wiki/%D7%A6%D7%91%D7%90) סביב העולם, וכן מורכבות ב[תוכנות](https://he.wikipedia.org/wiki/%D7%AA%D7%95%D7%9B%D7%A0%D7%94) וב[משחקים](https://he.wikipedia.org/wiki/%D7%9E%D7%A9%D7%97%D7%A7) רגילים.

קיימים שלושה אלגוריתמים עיקריים, אותם למדנו:

* **A\***
* \***העמדה בתנאים (constraint satisfaction)**
* \***סוכנים לוגיים**

בעבודה זו נעסוק באלגוריתם השני והוא **העמדה בתנאים**.

העמדה בתנאים

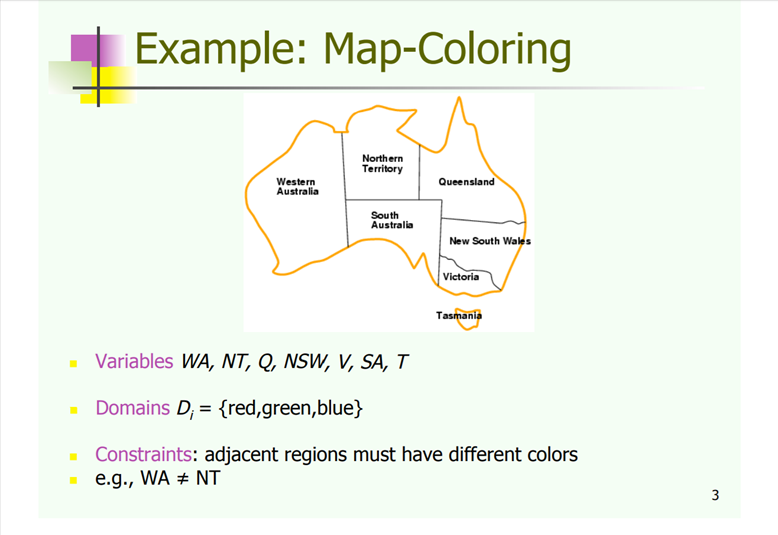
בעיית העמדה בתנאים מוגדרת פורמלית ע"י:

* \*קבוצת משתנים X1,X2,X3,…Xn (לדוגמא מדינות, שעות בלוח זמנים,..)
* \*קבוצה של מגבלות C1,C2,….Cn
* ולכל משתנה Xi יש דומיין Di של ערכים אופציונאליים (לדוגמא צבעים, true\false, שיעורים..).

מטרת האלגוריתם היא לתת לכל משתנה ערך של דומיין אפשרי במסגרת המגבלות

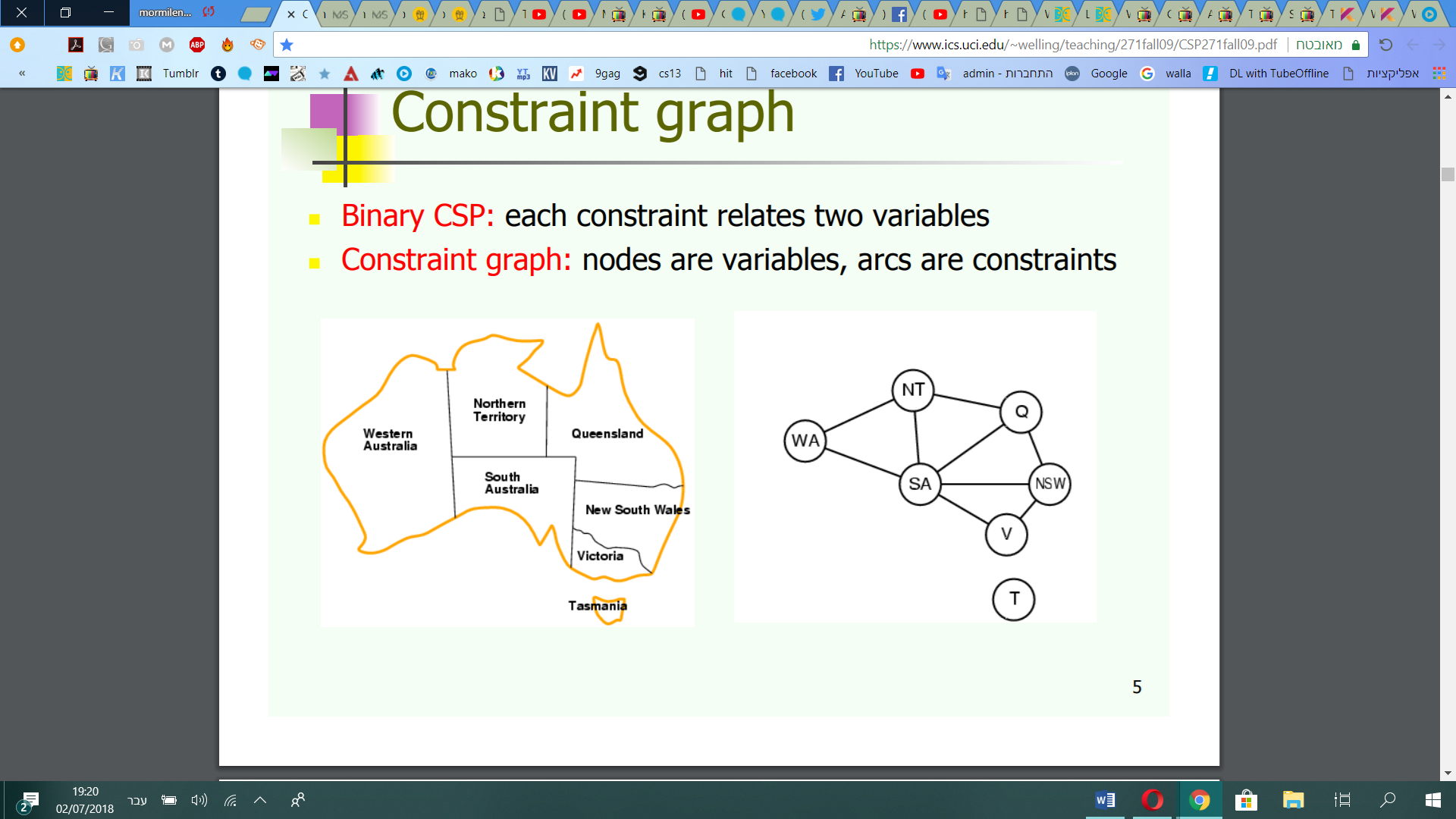
הקיימות.

דוגמא לבעיה כזו היא צביעת מפה כך שלא יהיו שתי ערים שכנות באותו צבע:



ישנם שני סוגי גרפים להצגת ההגבלות בבעיה:

1. בינארי (binary csp)- שבו כל הגבלה מקשרת בין שני משתנים.
2. גרף (constraint graph)- צומת היא משתנה והקשתות הן המגבלות.



בהעמדה בתנאים המגבלות מתחלקות לשלושה סוגים:

1. אונארי- מגבלות המערבות משתנה יחיד. דוגמא לפי צביעת מפות, שמדינה מסויימת לא יכולה להיות בצבע מסויים.( SA ≠ green)
2. בינארי- מגבלות המערבות זוגות של משתנים. דוגמא מצביעת המפות: צבע של מדינה 1 לא יהיה זהה לצבע של מדינה 2. (SA ≠ WA)
3. דרגה גבוהה- מגבלות המערבות שלוש או יותר משתנים. דוגמא מצביעת

המפות: מדינה 1 לא תיהיה באותו צבע כמו מדינה 2 ו 3. (SA ≠ WA ≠ NT)

קיימת גם מגבלה רכה- כאשר ערך מסויים יהיה עדיף על ערך אחר. בדוגמא של צביעת המפה יתכן וצבע אחד עדיף על השני (לדוגמא אדום עדיף על ירוק). לרוב הגבלה זו מיוצגת ע"י עלות עבור כל ערך.

**מבנה אלגוריתם עמידה בתנאים (csp) :**

האלגוריתם עליו מבוסס csp הוא אלגוריתם ה backtracking.

אלגוריתם זה עובד ברקורסיה ומנסה לשייך ערכים לכל המשתנים ובודק אם הערכים

האלו מקיימים את התנאים שהוצבו בבעיה.

אם כן, אז האלגוריתם ימשיך. אחרת, הוא ישנה את הערך האחרון שהוצב (שהפר

את התנאי).

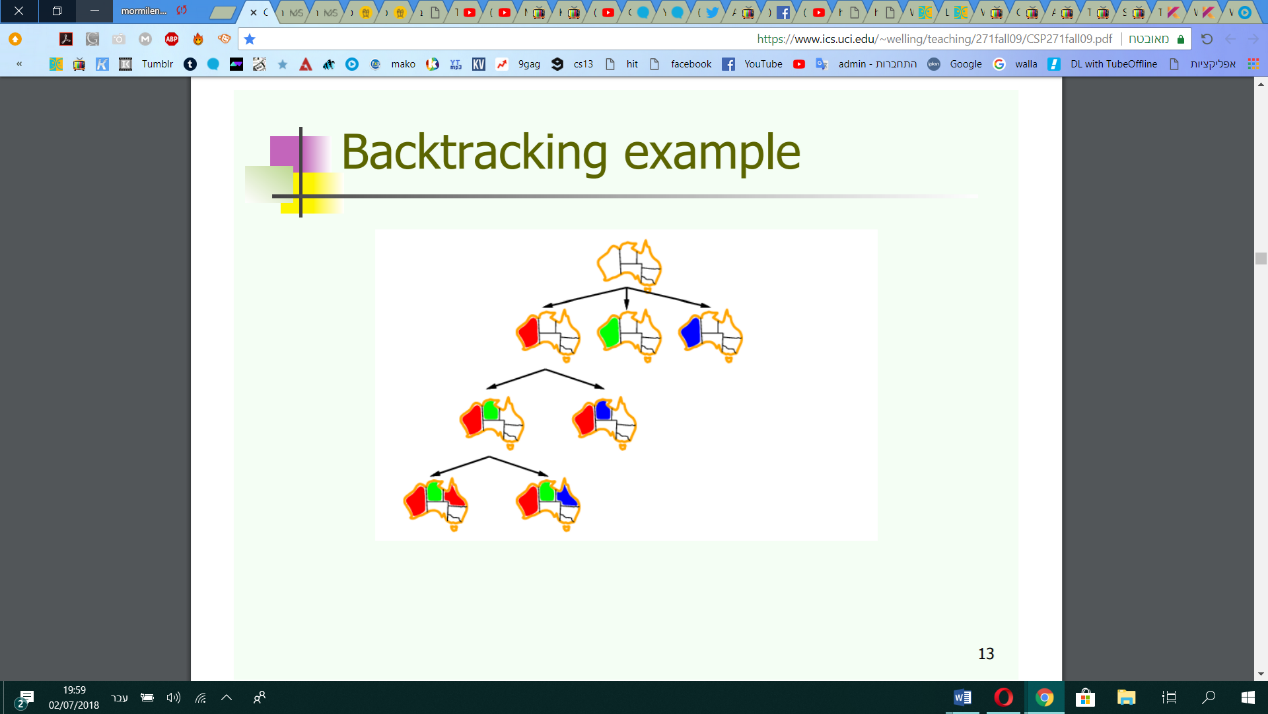
אם מגיעים למצב שאין ערך לתת למשתנה אז זה יחשב ככישלון והאלגוריתם יחזור וישנה את הערכים הקודמים. מכאן מקור השם backtracking כי חוזרים אחורה ומשנים ערכים קודמים כשמגיעים למבוי סתום. (במידה ואין מבוי סתום זו בעיה פולינומיאלית)

אלגוריתם של backtracking עובד בדרך חיפוש לעומק, אם נותנים לו בעיה שאין לה

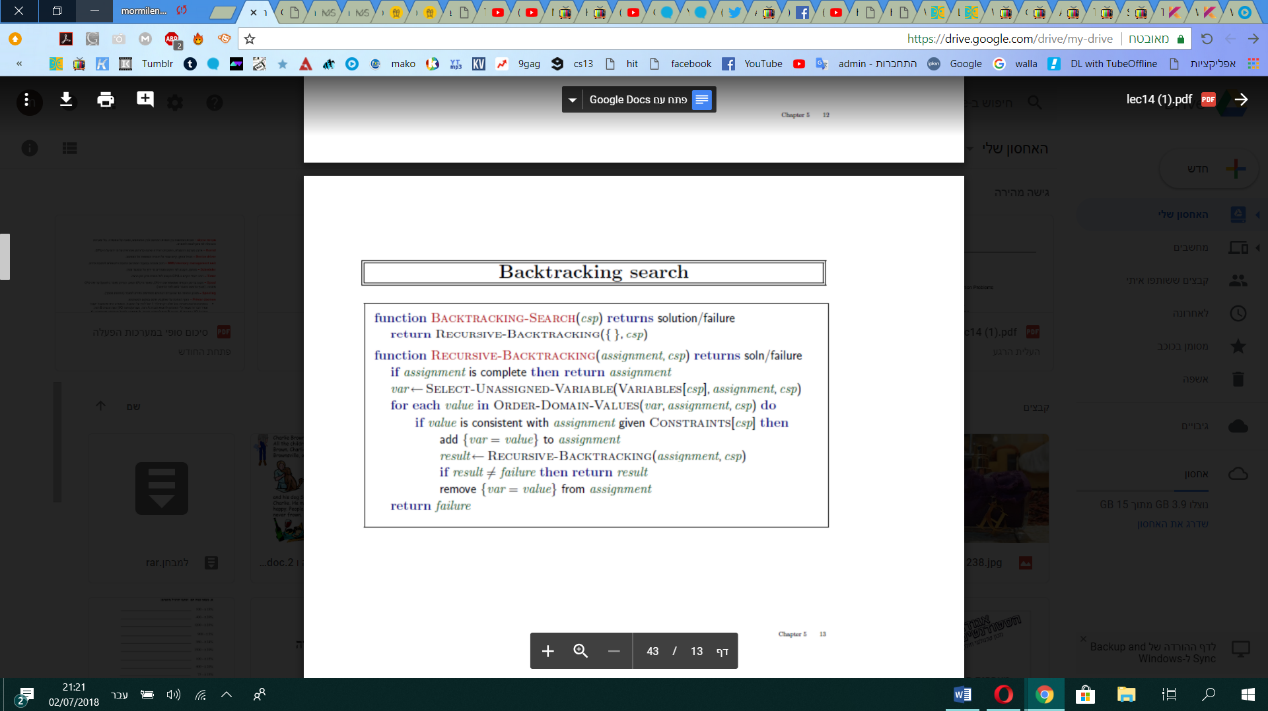
פתרון הוא ימלא ערכים ויחזור כל פעם מחדש עד שיעבור על הכל (כל האפשרויות)

ויודיע שאין פיתרון.

דוגמא להתקדמות ה Backtracking בדוגמת צביעת המפה:



פסאדו קוד ל Backtracking:

..

Backtracking עובד אך הוא אינו יעיל מבחינת זיכרון וזמן.

בכדי לשפר את אלגוריתם Backtracking נצטרך לשאול את השאלות הבאות:

1. מי יהיה המשתנה הבא לו נציב ערך?
2. באיזה סדר נציב בו את הערכים?
3. האם ניתן לזהות מבוי סתום מוקדם יותר?
4. האם ניתן להשתמש במבנה הבעיה?

מכאן שיש מספר דרכים ליעל את האלגוריתם הנקראות **יוריסטיקות**

יוריסטיקה- גישה כלשהי לפתירת בעיה, למידה או גילוי שמממש שיטה פרקטית, שלא מבטיחה להיות אופטימלית, מושלמת, לוגית או רציונאלית אבל כן מספקת בשביל להגיע לתוצאה מידית. כאשר מציאת פיתרון אופטימאלי הוא בלתי אפשרי או אינו פרקטי היוריסטיקה יכולה לשמש כדי לזרז את התהליך מציאת פתרון מספק.

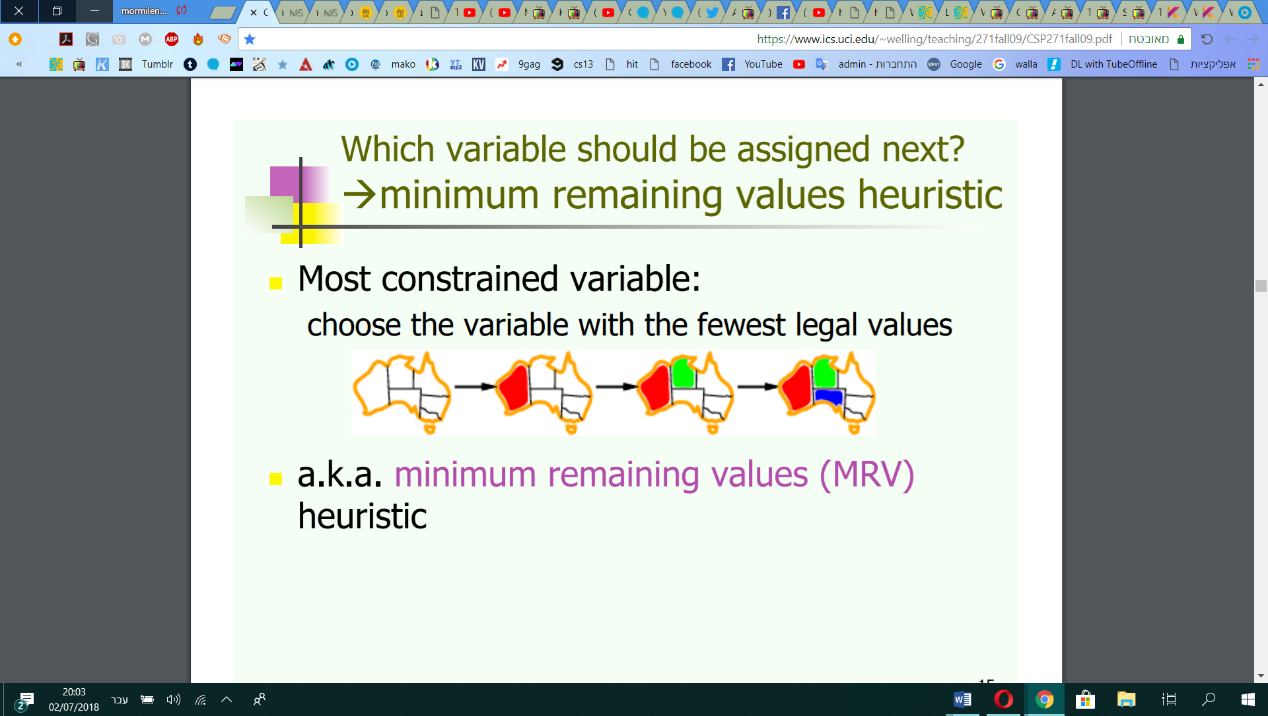
היוריסטיקות הקיימות בבעיית עמידה בתנאים הן:

1. **MRV (minimum remaining values)-**

ביוריסטיקה זו נבחר את המשתנה עם המספר המינימאלי של ערכים חוקיים,

כלומר בוחרים את המשתנה עם מספר האפשרויות הקטן ביותר.

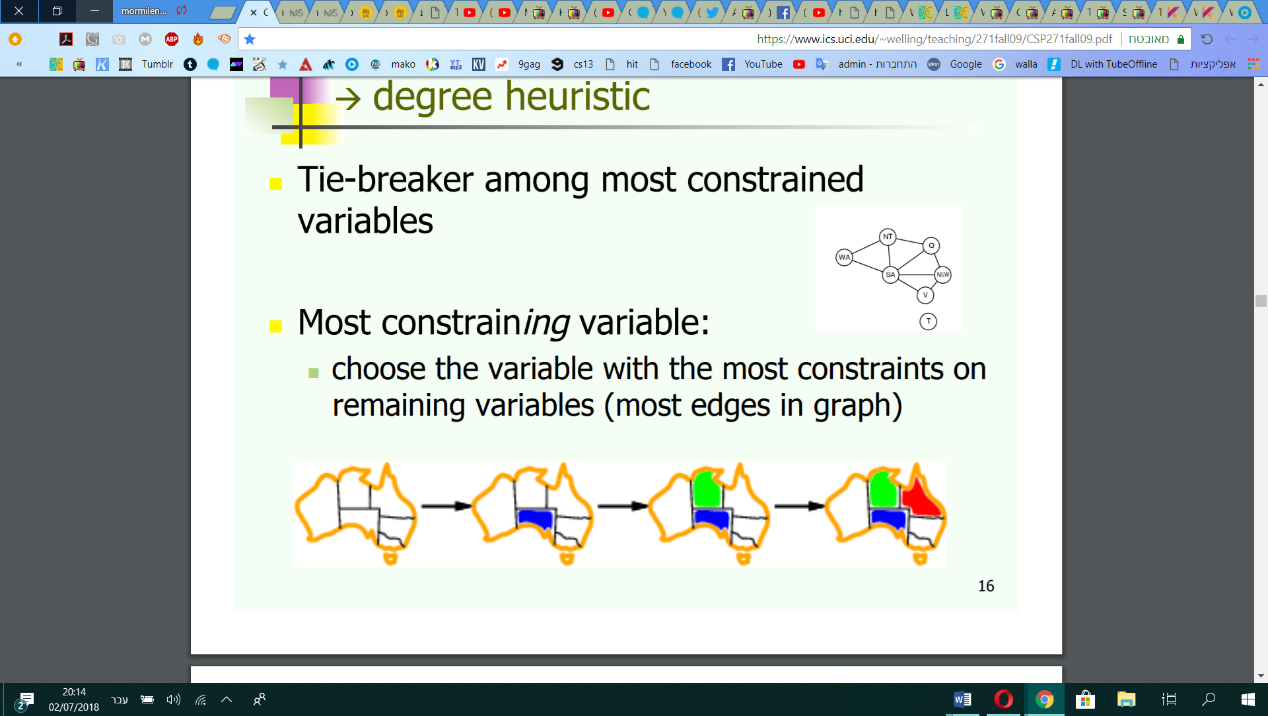
דוגמא לכך בבעיית צביעת המפה:



1. **Degree heuristic-**

ביוריסטיקה זו נבחר את המשתנה שהכי משפיעה על המשתנים האחרים.

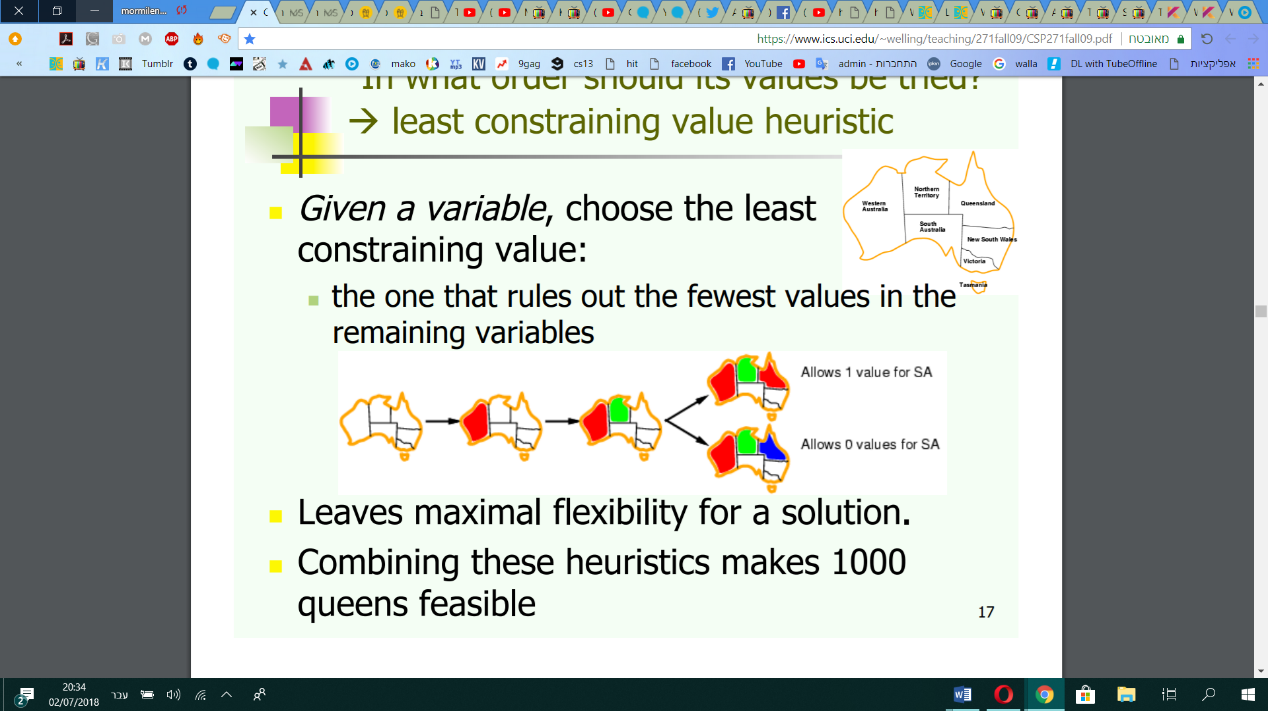
(הצומת בעל הדרגה הגבוהה ביותר) דוגמא לכך בבעיית צביעת המפה:



1. **Least constraining value heuristic-**

ביוריסטיקה זו בהינתן משתנה, נבחר את הערך הכי פחות מגביל, כלומר זה שיפסול

הכי פחות ערכים לשאר המשתנים.



1. **Arc consistency-**

ביוריסטיקה זו לפני הצבת כל ערך בודקים אם לאחר הצבת ערך זה נגיע

למצב של מבוי סתום, בדיקה זו נעשית ע"י בדיקת האפשרויות הנשארות לשאר

המשתנים בכך שנעבור על ההגבלות המיוצגות ע"י הקשתות בגרף. במידה והצבת

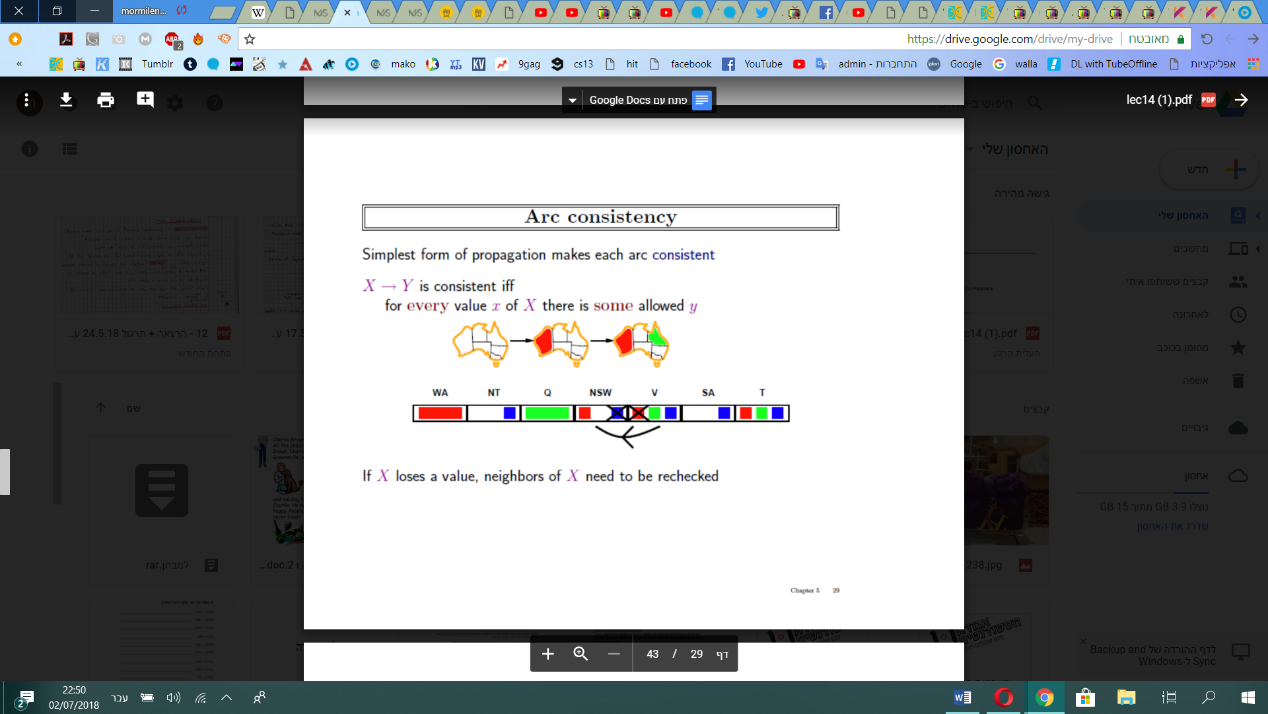
משתנה זה תוביל למבוי סתום כלומר שלמשתנה אין יותר ערכים אפשריים התוכנית

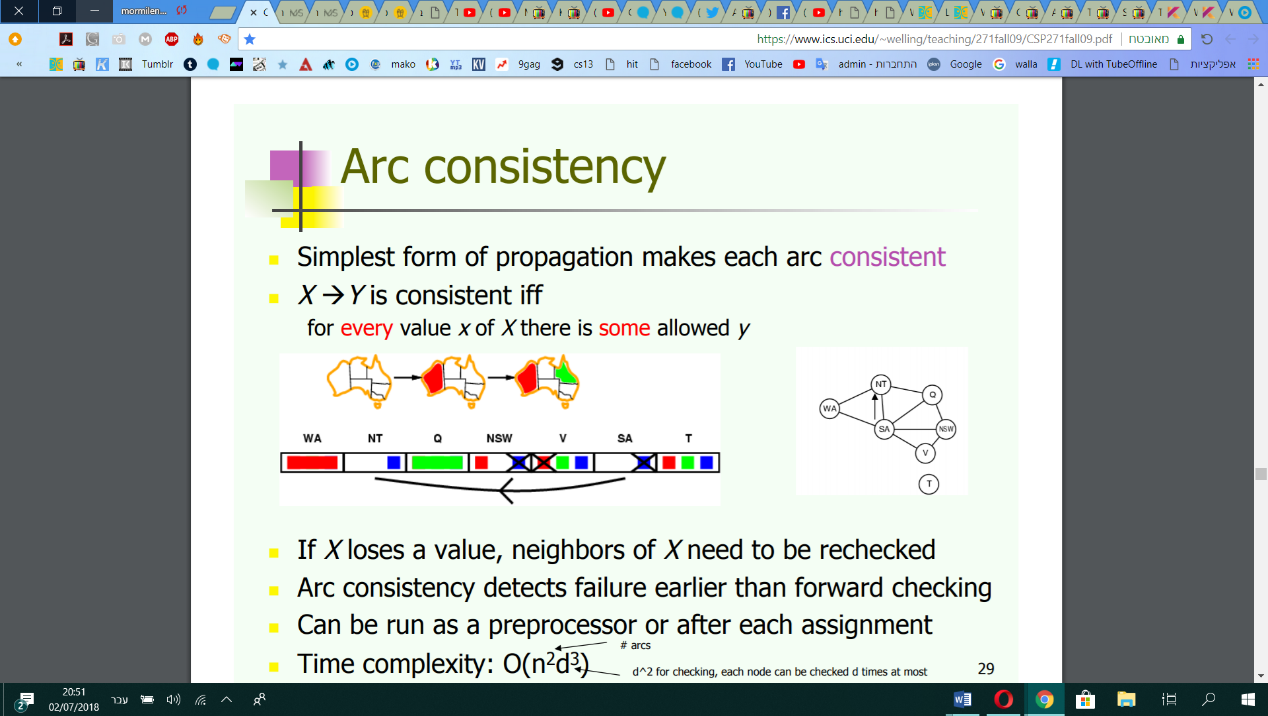
תחזור אחורה.

מהדוגמא של המפה נתן לראות את השלבים של עדכון האפשרויות לצבעים בערים

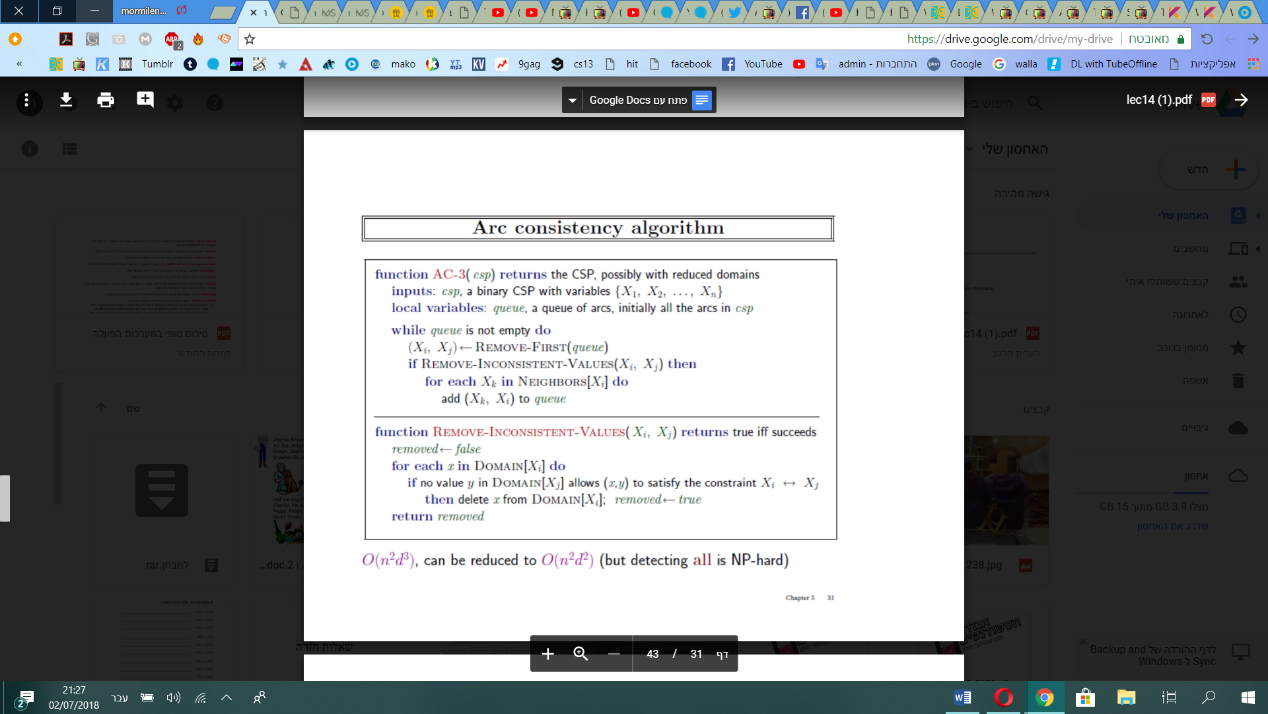
השכנות לזאת שאליה הצבנו את הצבע (הערים השכנות מחוברות בקשתות אחת

לשנייה).





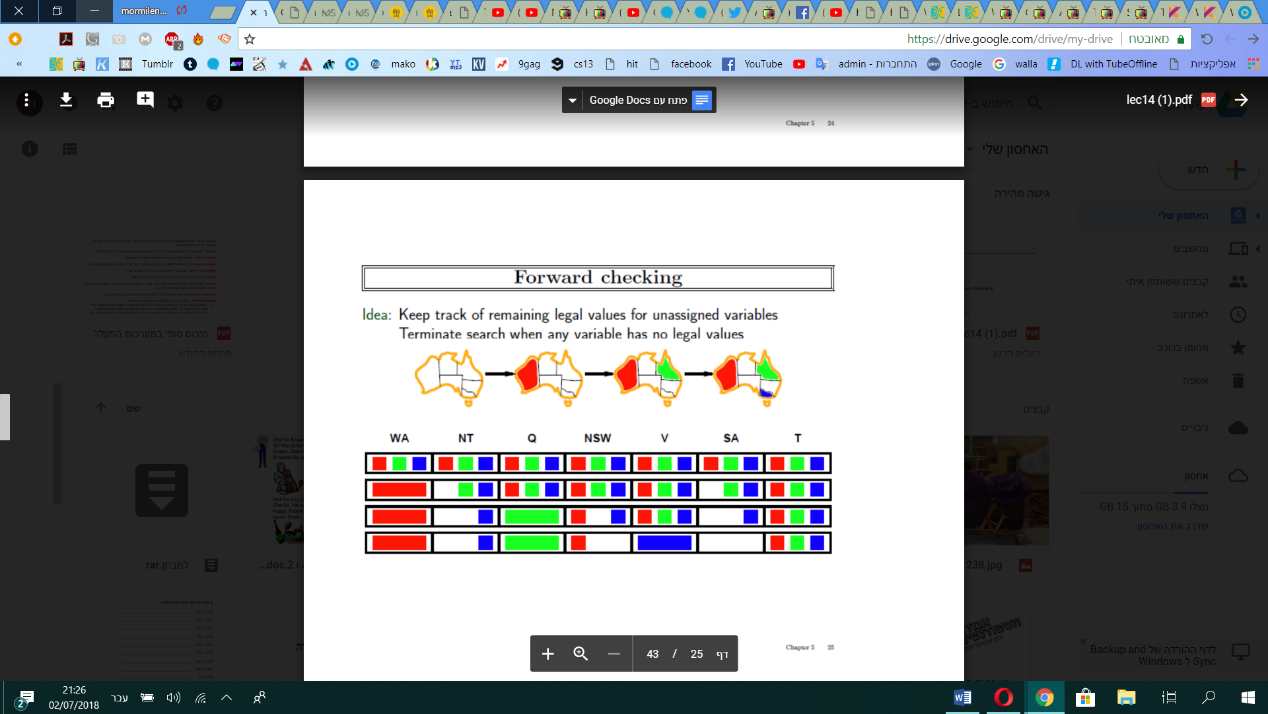
פסאדו קוד ל Arc consistency:



בכדי לממש את היוריסטיקות נשתמש ב **forward checking** .

forward checking- עוקב אחר הערכים החוקיים שנשארו לכל משתנה שעוד לא ניתן

לו ערך ועוצר את החיפוש כשלמשתנה כלשהו אין שום ערך חוקי אפשרי.

מהדוגמא של המפה ניתן לראות את הערכים האפשריים לכל עיר בכל שלב:

דוגמאות לבעיות שאלגוריתם csp פותר:

\*סודוקו

\*בעיית ארבע המלכות

\*צביעת מפות

\*בניית לוחות זמנים

\*בניית מערכת משמרות

**חלק שני (יצוג הבעיה)**

בנית Scheduler:

תכנון משימה- זה מציאת רצף פעולות שיעביר אותנו מנקודת ההתחלה למצב אותו אנחנו רוצים.

תזמון משימה (Scheduling task)- זה מיקום פעולות ידועות בצורה ששומרת על קיום ההגבלות.

בעיית תזמון היא בעיה סטאטית, כל הפעילויות, המשתנים וההגבלות ידועות ובדרך כלל היא נפתרת בעזרת אלגוריתם העמדה בתנאים (csp) סטנדרטי.

**תיאור הבעיה:**

בחרנו לעשות תוכנית שבונה מערכת שעות לסטודנט על פי קריטריונים והגבלות שונות. התוכנית מקבלת רשימת קורסים שניתן להרשם אליהם (רשימת הקורסים אינה מוגבלת למספר מסויים של קורסים, כך שיכולים להיות n קורסים אפשריים) ואת מספר נקודות הזכות שהסטודנט צריך לאסוף באותו סמסטר, כאשר כל קורס מזכה את הסטודנט במספר נקודות שונה כך שהמערכת שהתוכנית מציעה חייבת לשמור על כך שהסטודנט יקבל את מספר נקודות הזכות המדויק שנדרש (לא יותר ולא פחות) ובנוסף היא חייבת לענות על ההגבלות הבאות:

עבור קבוצת קורסים {x1,x2,….,xn}

* ישנם קורסים הנופלים על אותן שעות, אי אפשר לקחת שני קורסים שחופפים בשעות. עבור קורסים xi ו- xj כש i!=j:

end(xi)<=start(xj) || start(xi)>=end(xj)

* ישנם קורסים המחייבים תרגול.

Ǝ course(xi) -> practice(xj)

* ישנם קורסים המחייבים קורסים נוספים (שאם לוקחים אותם בהכרח צריך לקחת את הקורסים הנוספים)

Ǝ course(xi) -> course(xj)

Ǝ course(xi) -> course(xj) ^ course(xk)

Ǝ course(xi) -> course(xj) ^ course(xk) ^ course(xp)

* אי אפשר לקחת יותר מקורס אחד עם אותו מרצה

Class\_taken(xi) ^ professor(xi) = professor(xj) -> cant\_take(xj)

* חייבים לקחת לפחות קורס כללי אחד

Num\_general\_course>=1

* ישנם קורסי חובה שחייבים לקחת

If is\_must(xi) -> take(xi)

* חייבים להגיע למספר מסויים של נקודות זכות (בדיוק אותו מספר, לא יותר)

מספר הנקודות הנדרש לאותו סמסטר - P

Num\_of\_points(taken\_courses)=P

**תיאור המצב הנתון:**

בעבודה שלנו המשתנים (variables) הם הקורסים וה- domains הם true ו- false כאשר true מסמל שהקורס נלקח ו- false שלא.

כל משתנה (קורס) מורכב מהתכונות הבאות:

* זמן התחלה
* זמן סיום
* היום בו הוא מתקיים
* מרצה
* מספר נקודות זכות עבורו
* האם הוא קורס כללי / תרגול / רגיל
* האם הוא קורס חובה
* אילו קורסים נלווים הוא דורש
* התרגולים הקשורים אליו (אם קיימים כאלו)

פרוט הקורסים:

להלן רשימת הקורסים בהם ישתמשנו בבדיקות.

שים לב, שישנם קורסים עם שתי אפשרויות להרצאות בעלות קבוצת תרגולים משותפת.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| שם | סוג | מספר | יום | התחלה | סוף | מס פרופסור |
| C# | רגיל | 1 | 3 | 8 | 11 | 36 |
| Java | רגיל | 2 | 1 | 13 | 16 | 35 |
| Math1\_p1 | תרגול | 3 | 1 | 12 | 13 | 1 |
| Math1\_p2 | תרגול | 4 | 2 | 15 | 16 | 28 |
| Math1\_p3 | תרגול | 5 | 0 | 15 | 16 | 3 |
| Ai | רגיל | 6 | 3 | 8 | 11 | 37 |
| Alg\_p1 | תרגול | 7 | 1 | 11 | 12 | 5 |
| Alg\_p2 | תרגול | 8 | 4 | 10 | 11 | 29 |
| Alg\_p3 | תרגול | 9 | 2 | 8 | 9 | 6 |
| Math2 | רגיל | 10 | 3 | 9 | 12 | 12 |
| Lin\_p1 | תרגול | 11 | 3 | 8 | 9 | 5 |
| Lin\_p2 | תרגול | 12 | 3 | 9 | 10 | 8 |
| Lin\_p3 | תרגול | 13 | 3 | 15 | 16 | 9 |
| Math2 | רגיל | 14 | 4 | 12 | 15 | 20 |
| Auto | חובה | 15 | 4 | 14 | 16 | 13 |
| Auto | חובה | 16 | 1 | 11 | 13 | 18 |
| Math2\_p1 | תרגול | 17 | 3 | 13 | 15 | 21 |
| Math2\_p2 | תרגול | 18 | 4 | 12 | 14 | 22 |
| Math2\_p3 | תרגול | 19 | 1 | 10 | 12 | 16 |
| Log | חובה | 20 | 2 | 8 | 12 | 20 |
| Log | חובה | 21 | 2 | 12 | 16 | 21 |
| Log\_p1 | תרגול | 22 | 2 | 12 | 14 | 23 |
| Log\_p2 | תרגול | 23 | 2 | 10 | 12 | 31 |
| Log\_p3 | תרגול | 24 | 0 | 10 | 12 | 31 |
| Alg | חובה | 25 | 3 | 12 | 15 | 4 |
| b.d | רגיל | 26 | 0 | 10 | 12 | 24 |
| Auto\_p1 | תרגול | 27 | 1 | 13 | 15 | 15 |
| Auto\_p2 | תרגול | 28 | 3 | 10 | 12 | 9 |
| Auto\_p3 | תרגול | 29 | 2 | 14 | 16 | 25 |
| Phil | כללי | 30 | 2 | 12 | 14 | 10 |
| b.d\_p1 | תרגול | 31 | 0 | 12 | 13 | 17 |
| b.d\_p2 | תרגול | 32 | 0 | 13 | 14 | 32 |
| b.d\_p3 | תרגול | 33 | 1 | 13 | 14 | 26 |
| Apps | רגיל | 34 | 1 | 8 | 10 | 34 |
| Math1 | חובה | 35 | 0 | 14 | 16 | 27 |
| Lin | רגיל | 36 | 4 | 8 | 10 | 7 |
| Math1 | חובה | 37 | 0 | 8 | 10 | 2 |
| Culture | כללי | 38 | 3 | 8 | 10 | 12 |
| Met | כללי | 39 | 4 | 10 | 12 | 11 |
| Ant | כללי | 40 | 4 | 8 | 10 | 10 |
| Apps\_p1 | תרגול | 41 | 4 | 10 | 12 | 15 |
| Apps\_p2 | תרגול | 42 | 1 | 12 | 14 | 16 |
| Apps\_p3 | תרגול | 43 | 0 | 14 | 16 | 17 |
| Yoga | כללי | 44 | 1 | 9 | 11 | 12 |
| Java\_p1 | תרגול | 45 | 1 | 11 | 13 | 13 |
| Java\_p2 | תרגול | 46 | 2 | 8 | 10 | 16 |
| Java\_p3 | תרגול | 47 | 3 | 14 | 16 | 18 |
| History | כללי | 48 | 0 | 11 | 13 | 10 |
| C#\_p1 | תרגול | 49 | 2 | 8 | 10 | 18 |
| C#\_p2 | תרגול | 50 | 0 | 9 | 11 | 17 |
| C#\_p3 | תרגול | 51 | 3 | 13 | 15 | 19 |
| Crime | כללי | 52 | 2 | 14 | 16 | 10 |
| Ai\_p1 | תרגול | 53 | 4 | 10 | 11 | 14 |
| Ai\_p2 | תרגול | 54 | 1 | 11 | 12 | 19 |
| Ai\_p3 | תרגול | 55 | 4 | 12 | 13 | 23 |

התלויות בין הקורסים:

* Alg -> lin
* Math2 -> math1 ^ b.d
* Java -> apps ^ b.d ^ ai
* C# -> b.d ^ apps ^ alg
* Ai -> math1 ^ b.d

התוכנית עוברת ברקורסיה ומנסה להוסיף כל פעם קורס אחר, במידה והקורס מפר את אחת ההגבלות התוכנית תחזור אחורה ברקורסיה ותנסה קורס אחר.

במידה והתוכנית מוצאת פיתרון, כלומר מערכת הבנויה מקורסים שסכום מספרי הזכות שלהם שווה לזה שהיא היתה צריכה לאסוף והם לא מפרים אף אחד מהתנאים שהוצגו, היא תדפיס אותו ותעצור. אחרת היא תרוץ על כל האפשרויות ותדפיס בסוף אפס כלומר שאין פיתרון אפשרי.

**חלק שלישי**

טבלת ההרצות:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Points num | base | Base+forword | Eur1 | Eur2 | Eur3 | Arc-3 |
| 19 | @ | @ | 0.037 | 112.038 | 17.076 | @ |
| 20 | @ | @ | @ | @ | @ | @ |
| 21 | 15012 | @ | 0.031 | 0.601 | 16.942 | @ |
| 22 | @ | @ | @ | @ | @ | @ |
| 23 | 14835.2 | @ | 0.036 | 0.092 | 0.073 | @ |
| 24 | 11384.3 | @ | 0.287 | 88.784 | 9.576 | @ |
| 25 | 6220.6 | @ | 0.04 | 2.968 | 2.356 | @ |
| 26 | @ | @ | 0.277 | 11956.4 | 148.638 | @ |
| 27 | @ | @ | 0.264 | 59.706 | 88.962 | @ |
| 28 | 38.266 | 60.597 | 12.929 | 2031.98 | 3.726 | 245.007 |
| 29 | @ | @ | 12.662 | 4209.71 | 169.676 | @ |

* הזמנים מופיעים בשניות
* החלטנו לתת הגבלת זמן של שעה להרצות, במידה והתוכנית רצה מעל לשעה והיא לא הגיעה לתוצאות, זה סומן כ @.

**בסיס:**

ראשית הרצנו את התוכנית ללא כל יורסטיקה וראינו שאחרי הרצה של זמן רב היא עדיין לא מגיעה לתוצאות עבור כל סכומי הנקודות מלבד 28. כנ"ל לאחר הוספת ה forward checking התוכנית עדיין התקשתה להגיע לתוצאות (עבור כל סכומי הנקודות מלבד 28, עבורו לקח יותר זמן מהתוכנית ללא ה forward checking מכיוון שהתוכנית הריצה בדיקות נוספות) .

הגענו למסקנה כי עבור 28 נקודות התוכנית כן מצליחה להגיע לתוצאות מהירות בגלל הסדר בו הקורסים הוזנו למערכת, במקרה יצא שלפי סדר זה ניתן להגיע לתוצאה של 28 נקודות בזמן סביר.

בתוכנית המקורית (ללא יורסטיקות ו forward checking) עבור כמה דוגמאות, השארנו את התוכנית לרוץ עד להגעתה לתוצאה מסויימת כדי לראות שהיא אכן יכולה להגיע לתוצאה ואם כן אז באיזה טווח זמנים היא תגיע אליה.

מסקנה: לפי ההרצות הראשונות הגענו למסקנה כי התוכנית מתקשה להגיע לתוצאות ולכן נדרשות יורסטיקות בכדי שהיא תוכל להגיע לתוצאות בזמן סביר.

**יוריסטיקה 1:**

היוריסטיקה הראשונה היא יוריסטיקת ה – MRV, בה התוכנית בוחרת את המשתנה עם הכי פחות אפשרויות. היוריסטיקה מוצאת את הקורס הכי "בעיתי" , כלומר בוחרים את הקורס המתקיימים בו הכי הרבה מהמקרים הבאים:

* מתבטלים לו תרגולים בגלל אחת ההרצאות שלו (בגלל שעות ההרצאה או המרצה בה)
* קורס שהוא חובה
* קורס הדורש קורסים קודמים

ואת הקורס הכי "בעיתי" משבצים קודם (בכל הרצה מחפשים את הקורס הבעיתי הבא).

מסקנה: לאחר ההרצה ראינו כי התוכנית הגיע לתוצאות בזמנים מהירים כמעט לכל המקרים. דבר זה הגיוני מכיוון שבעזרת יוריסטיקה זו התוכנית שיבצה קודם את הקורסים הבעייתים ורק לאחר מכן עברה לאלו שהיה קל יותר לשבץ, בכך חסכה הרבה בדיקות אשר היו מובילות למבוי סתום וגורמות לתוכנית לרוץ הרבה יותר זמן.

**יוריסטיקה 2:**

יוריסטיקה זו היא יוריסטיקת ה- Degree. זו יוריסטיקה בה בוחרים את המשתנה לפי השפעתו, בוחרים את הקורס שהכי ישפיע על קורסים אחרים, כלומר הוא יבטל הכי הרבה קורסים אחרים בכך שהוא מופיע בשעות בהן מתקיימים הכי הרבה קורסים ויש לו את המרצה הכי פופולרי (במקרה זה אי אפשר יהיה לקחת את כל שאר הקורסים עם אותו מרצה).

מסקנה: לפי ההרצות היוריסטיקה משפרת חלקית את פעילות התוכנית. במקרים הקלים היא מגיעה לתוצאות בזמן סביר אך עבור מקרים קשים יותר עדיין לוקח לה קרוב לשעה ואף יותר (במקרה של 26 נקודות לקח לה קרוב לשלוש וחצי שעות) להגיע לתוצאות. מכאן שהיוריסטיקה שיפרה את הביצועים בהשוואה עם המצב הבסיסי אבל לא ברמה של יוריסטיקה 1.

היוריסטיקה הזו בוחרת את הקורס הכי משפיע, כלומר הקורס שפוסל הכי הרבה קורסים אחרים, ומשבצת אותו. בכך בכל פעם אנחנו נשארים עם קבוצת קורסים קטנה יותר לבחור ממנה, מה שיכול לחסוך בכמות הצעדים אך לא בהכרח מונע ריצות מיותרות והגעה למבוי סתום (במקרים מסויימים אך עלול להוביל לכך, במידה ואפשר להגיע לתוצאה מהירה יותר ללא שימוש בקורס שנבחר).

**יוריסטיקה 3:**

יוריסטיקה זו היא יוריסטיקת ה- Least constraining value והיא בוחרת את הערך שפוסל הכי פחות אפשרויות, מכיוון שבתוכנית שלנו הערכים הם true ו- false ,כלומר אם לקחת או לא את הקורס, יוצא שיוריסטיקה זו היא ההפך מהיוריסטיקה השניה, היא תבחר את הקורס שהכי פחות משפיע על הקורסים האחרים במידה ונבחר אותו שזה הקורס שמתקיים בשעות בהן מתקיימים הכי פחות קורסים והמרצה שלו מלמד הכי פחות קורסים.

מסקנה: ראשית היוריסטיקה משפרת את הביצועים של התוכנית הבסיסית אך הביצועים שלה פחות טובים מאלו של היוריסטיקה הראשונה, בדומה ליוריסטיקה השניה ביוריסטיקה זו הביצועים שונים עבור כל מקרה, מכיוון שלא תמיד עדיף לקחת את הקורס שהכי פחות משפיע על האחרים, מכאן שגם יוריסטיקה זו חוסכת ריצות רבות אבל עדיין יכולה להוביל לריצות מיותרות למבוי סתום. מכיוון שהיוריסטיקות האלו שונות עבור כל מקרה (לפעמים עדיף לבחור את הקורס שיפסול הכי הרבה קורסים ולפעמים עדיף לקחת את זה שיפסול הכי פחות).מכאן שניתן לראות לפי ההרצות את היחס בין היוריסטיקה השניה והשלישית: קיימים מקרים בהם היוריסטיקה השניה עדיפה (לדוגמא עבור 21 נקודות), מקרים בהם היורסטיקה השלישית עדיפה (לדוגמא עבור 26 נקודות) ומקרים שהתוצאות בינהם דומות (לדומא עבור 25 נקודות).

**יוריסטיקה 4:**

יוריסטיקה זו היא יוריסטיקת ה arc-3. התוכנית בודקת לפני שעושים כל צעד אם לאחר שנעשה אותו נגיע למצב שלא ניתן יהיה לאסוף את מספר הנקודות הרצוי, כלומר או שהקורס שבחרנו גורם לאוסף הנקודות להיות גבוה ממספר הנקודות הרצוי או שסך כל הנקודות של כל הקורסים שנשארו שאפשריים לבחירה (עומדים בתנאים) ביחד עם הנקודות שאספנו עד כה עדיין לא מוביל למספר הנקודות הרצוי (כלומר בחרנו קורסים שביטלו קורסים רבים מדי).

מסקנה: היוריסטיקה לא שיפרה את זמני ההרצות לעומת התוכנית הבסיסית ואף לקח לה יותר זמן מהתוכנית הביסיסית עבור 28 נקודות. זאת ניתן להסביר בכך שבתוכנית שלנו הארק ידע אם הצעד שעשינו יוביל למבוי סתום רק לאחר שיבנה את מרבית המערכת (רק כשיגיע לקורסים האחרונים שנשארו לבניית המערכת), ולכן היוריסטיקה חוסכת רק את הצעדים האחרונים מכל ההרצות מה שלא משפר משמעותית את זמני הריצה. עבור מקרה של בניית מערכת המובילה למבוי סתום רק לאחר שתיבנה כמעט כל המערכת והתוכנית תגיע למספר נקודות הקרוב למספר הרצוי (מה שיקח זמן) רק באחד הצעדים האחרונים לבניית המערכת הוא יראה שלא יצא לו סכום נקודות רצוי והוא יחזור לאחור, ובכך יחסך מספר צעדים קטן ולכן זמן הריצה לא ישתפר משמעותית.

בנוסף לכך עבור המקרה בו התוכנית הבסיסית כן הצליחה להגיע לתוצאה בזמן סביר, עבור 28 נקודות, מכיוון שיוריסטיקת הארק מריצה בדיקות נוספות יוצא שהזמן שלוקח להגיע לתוצאה הוארך.

**מסקנות סופיות:**

יוריסטיקות 2,1 ו-3 יעלו את התוכנית וגרמו להגעה לתוצאות בזמן טוב יותר, כאשר יוריסטיקה אחת נתנה את התוצאות הכי טובות ולכן היא הכי יעילה.

לעומת יוריסטיקה 1 יוריסטיקות 2 ו-3 תלויות בסדר הזנת הקורסים, מכיוון שיתכן ויש כמה קורסים המשפיעים על הקורסים האחרים באותה רמה ומה שיקבע איזה קורס ילקח יהיה הסדר בו הם הוזנו (הקורס הראשון שימצא מתאים) ולכן הן יכולות לתת תוצאות שונות לסדרי קורסים שונים (כשמזינים את אותם הקורסים בסדר שונה), לעומתם יוריסטיקה 1 בונה סדר משלה ובכך לא תלויה בסדר בהם מופיעים הקורסים.

בנוסף לכך עבור מקרים בהם אין תוצאה (לא ניתן לבנות מערכת עם 20 או 22 נקודות זכות לפי הקורסים שהזנו) עבור כל היוריסטיקות התוכנית לא הגיע לתוצאה (שאין תשובה) בזמן סביר, מכיוון שלשם כך היא צריכה לרוץ על כל האפשרויות השונות להרכבת המערכת.