

בינה מלאכותית – חלק שני

מאת ניר אדר (UnderWarrior)

לפני שנה יצא הגליון הראשון של Digital Whisper, ובו פרסמתי כתבה בנושא בינה מלאכותית. המאמר חיכה להמשך, והנה סוף סוף הוא מגיע ☺

נזכיר בקצרה נקודות חשובות שהוצגו במאמר הראשון:

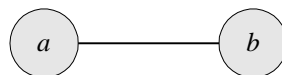
- בינה מלאכותית היא תחום מחקר במדעי המחשב, שהמטרה שלו היא פיתוח אלגוריתמים לתפיסה, הסקת מסקנות ולמידה, וזאת כדי לפתור בעיות מורכבות.
- אחד המושגים החשובים בעולם זה הוא מושג הסוכן האינטליגנטי. סוכן אינטליגנטי זו ישות התופסת את הסביבה שלה ופועלת עליה כדי להשיג מטרות שהוגדרו על ידי אדוניה.
- תחום הבינה המלאכותית עושה בשאלות שונות – איך מייצגים ידע? איך מסיקים מסקנות מתוך מידע קיים? איך בונים מערכת שמשתפרת על סמך ידע חדש (למידה)? ועוד בעיות רבות.

במאמר הראשון נגענו מעט בשאלת ייצוג הידע. הייצוג שהצענו לבעיות הוא ייצוג בעזרת **גרף מצבים**. זה לא הייצוג היחיד, אבל זה ייצוג שרלוונטי להרבה בעיות. **גרף** הוא מושג ידוע במדעי המחשב. בדומה למסמך הקודם, אני רוצה להעביר לכם כמה עקרונות בסיסיים בנושא בלי להכנס לכל התורה המתמטית לעומק.

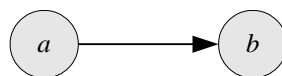
בדפים הבאים נרחיב מעט על תורת הגרפים, נדבר על מספר אלגוריתמים בסיסיים הקשורים לתורת הגרפים ונראה תוך כדי איך זה משתלב במערכות בינה מלאכותית שנבנה. אני משלב 2 נושאים לא פשוטים, אך השילוב ביניהם יאפשר לכם להבין את הדברים בלי להזדקק לכל הרקע המתמטי. למתעניינים – תחום תורת הגרפים ותחום האלגוריתמים הם מהתחומים המרתקים (והקשים) במדעי המחשב – מומלץ בחום!

גרף כללי כלשהו (נסמן אותו באות G) הוא מבנה המכיל קבוצת צמתים V וקבוצת קשתות שנסמן E . צומת נסמן על ידי עיגול, וקשת על ידי קו. קשת יכולה להיות עם כיוון (ואז הקו הוא חץ) או ללא כיוון. לכל קשת יש שתי נקודות קצה, שאינן בהכרח שונות.

איך זה נראה? דוגמאות לגרפים פשוטים:



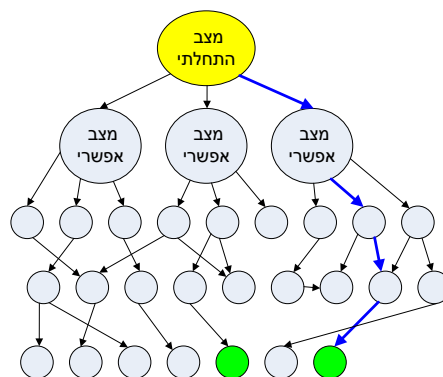
2 צמתים עם קשת לא מכוונת ביניהם



2 צמתים עם קשת מכוונת ביניהם

שימו לב למשהו מעניין – גרף הוא לא בהכרח סופי. יתרה מזאת, כאשר גרף הוא סופי, הוא עדיין יכול להיות גדול מאוד. זו אחת הבעיות העיקריות כשאנחנו מדברים על יצוג בעיות כגרפים, כפי שנראה בהמשך.

במאמר הקודם הראנו איך מייצגים בעיה שאנחנו רוצים לפתור בתור גרף:



אנחנו מתחילים במצב התחלתי ורוצים להגיע לאחד (או לכל) מצבי המטרה.

בהנחה שאנחנו יודעים לעבור ממצב למצב – איך נעשה את זה? מדובר על בעיה מסוג בעיות שנקרא **חיפוש בתוך גרף**. קיימים אלגוריתמים רבים לסוג בעיות זה השונים בתכונותיהם ובאופן פעולתם.

נתחיל מהתיאוריה של תורת הגרפים – יש לנו גרפים (מכוונים או לא מכוונים – זה לא משנה לצורך האלגוריתמים הראשונים שנציג) – שאנחנו רוצים לסרוק עד למציאת צומת מטרה.

שתיים מהשיטות הפשוטות ביותר (והמוכרות ביותר) נקראות האחת **Breadth-first search** (ובקיצור BFS) והשניה **Depth-first search** (ובקיצור DFS).

בעית "חיפוש בתוך גרף"

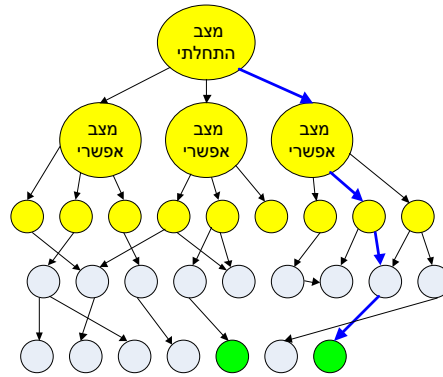
תיאור הבעיה:

- נתון גרף סופי וקשיר G. (גרף קשיר הוא גרף שבו אפשר להגיע לכל הצמתים – אין צומת שאליו אין קשתות).
- תהי צומת התחלה כלשהי, ממנה נתחיל ללכת על קשתות, מצומת לצומת, עד שנבקר את כל הצמתים. במקרה שלנו אנחנו מחפשים את צומת המטרה, אבל כדי להיות בטוחים שנגיע אליו – האלגוריתמים צריכים להבטיח לנו שהם מסוגים לבקר בכל הצמתים.
- אנו מחפשים אלגוריתם שיבטיח שאנו סורקים את כל הגרף, וכן שתהיה לנו אינדיקציה שסיימנו לסרוק את הגרף, מבלי לדעת כמה גדול הוא הגרף. (מבחינת כמות צמתים וקשתות).

הגבלות:

- אנו פועלים ללא תכנון מוקדם, למשל למידת "מפת הדרכים" של הגרף לפני התחלת הסיור, אנו חייבים לקחת החלטה אחר החלטה, מכיוון שאנו מגלים את מבנה הגרף רק תוך כדי הסריקה.
- נכנה בשם **מעברים** את החיבורים בין הקשתות לצמתים.

על מנת שנוכל למצוא אלגוריתם שיפתור בעיות זו, אנו צריכים להשאיר "סימונים" על המעברים כשאנו נעים, כדי שנוכל לזהות בעתיד שהגענו למקום בו כבר היינו.



שלב 3 – סרקנו את הרמה השניה של הבנים

הערות:

- כמו שניתן לראות בשרטוטים האלגוריתם "מסמן" את הצמתים בהם ביקר. במידה ויש מעגלים בגרף הוא לא יבקר בהם שוב.
- האלגוריתם המלא מורכב יותר ופורמלי יותר – אנחנו שומרים על ההגדרה הפשוטה כדי להעביר את צורת החשיבה.
- בגרסה הרגילה של האלגוריתם המטרה היא כאמור לעבור על כל הצמתים בגרפים. כשאנחנו כותבים מימוש עבור בעיה של בינה מלאכותית, לעתים קרובות משנים את האלגוריתם שיעצור כאשר נמצא הפתרון.

כמה תכונות מעניינות:

- אנחנו תמיד נמצא את המסלול הקצר ביותר למטרה. אם אנחנו מנסים לפתור בעיה וצריכים למצוא את כמות האופרטורים המינימלית שתביא אותנו לפתרון - BFS יתן לנו פתרון זה.
- אם קיים פתרון – בטוח נמצא אותו. הנושא מאוד ברור אינטואיטיבית, אבל עבור המתעסקים בתורת הגרפים זו נקודה חשובה שדרשה הוכחה.
- באופן דומה – אם לא נמצא פתרון, ואם הגרף סופי, אנחנו נדע שאין פתרון בסופו של דבר.

הרחבה על האלגוריתם והתכונות שלו ניתן למצוא בויקיפדיה:

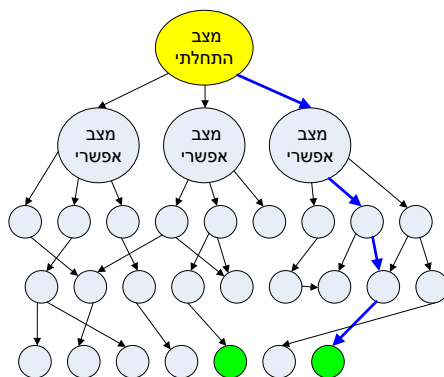
http://en.wikipedia.org/wiki/Breadth-first_search

Depth-first search

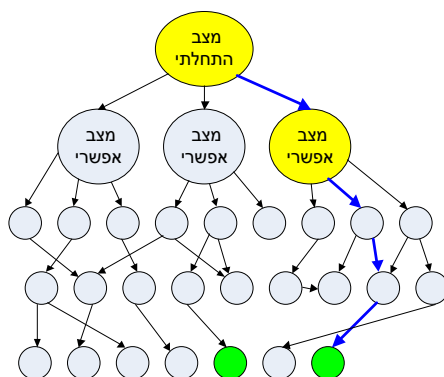
מקור האלגוריתם הוא במאה ה-19 בה שימשה להתמודדות מול בעיות מבוכים.

האלגוריתם בצורה מופשטת:

1. התחל בצומת ההתחלתי.
2. כל עוד לא עברת על כל המעברים בגרף:
 - a. אם קיים בצומת הנוכחי מעבר שבו עדיין לא ביקרת – בקר בו והמשך לצומת שנמצא בצד השני.
 - b. אם כל המעברים מסומנים – חזור אחורה – back tracking – אל הצומת הראשון במסלול שיש לו מעבר לא מסומן והמשך משם.



שלב 1 - מצב התחלתי



שלב 2 – האלגוריתם פונה לאחד הכיוונים – לא תמיד זה בהכרח הכיוון של הפתרון!

אז מי מהאלגוריתמים "טוב" יותר? ואיך מממשים את האלגוריתמים בקוד?

אנחנו מגיעים לנקודה המעניינת. נראה על פניו מהתכונות של BFS שהוא תמיד מוצלח יותר – שני האלגוריתמים מגיעים לפתרון, ו-BFS מחזיר תמיד את המסלול הקצר יותר. למה בכלל צריך את DFS?

בנוסף יש עוד שאלה שנרצה לענות עליה – למה בעצם אנחנו מניחים שאנחנו יכולים להתקדם כל פעם רק שלב אחד? למה אי אפשר "להסתכל" על הגרף ולקפוץ ישר לפתרון?

התשובה לשתי השאלות האלו קשורה: כשאנחנו מדברים על התיאוריה של תורת הגרפים אנחנו מציירים גרפים ו-"רואים" אותם בעיניים. בפועל הגרפים שאנחנו ניצור גדולים לאין שיעור מהדוגמאות הפשוטות שאנחנו מציירים. הבעיה העיקרית היא בעית זכרון. נניח למשל אנחנו בונים גרף עבור משחק, שבו כל מצב במשחק מיוצג כצומת, והמעברים מציינים החלטות במשחק. אם אפשר להזיז כל אחד מ-20 כלים ל-4 כיוונים. במקרה כזה – לכל מצב יהיו 80 (!!!) צמתים בניים.

אם נבדוק רק 3 רמות למטה, יהיו 82,432,000 מצבים! שמונים ושניים מיליון! אם נבדוק רמה נוספת, גם אם כל מצב ייוצג על ידי byte בודד, יגמר לנו כבר הזכרון במחשב. הנושא הראשון אם כך שאנחנו מתייחסים אליו הוא **סיבוכיות המקום** (זכרון) של האלגוריתם.

עכשיו – איך בעצם אנחנו מממשים את האלגוריתמים? המימוש המקובל הוא בכל פעם שאנחנו לוקחים צומת, ובחרים להכנס לאחד הבנים שלו, שאר הבנים נכנסים למבנה נתונים של מחסנית (או ערימה). כאשר הגענו למבוי סתום, לוקחים את הצומת הבא משם.

- BFS יפתח בכל פעם את כל הרמה (הבנים שמתחתיו) במלואה, יכניס את כולם למחסנית, וימשיך לרמה הבאה. DFS לעומת זאת מפתח את הבנים, ובוחר רק אחד מהם לפיתוח.
- במקרה של המשחק המשוגע שתיארתי למעלה, DFS יכניס בכל שלב למחסנית 80 צמתים חדשים (ואולי יפחית כשהוא מסמן צומת או מגיע למקום ללא מוצא), ואילו BFS יכניס 80^N צמתים, כש-N הוא הרמה הנוכחית של הסריקה.

שאלה חשובה שניה – כמה זמן לוקח לכל אלגוריתם להחזיר תשובה? כאן הנושא משתנה בהתאם לאופי הבעיה – אם הגרף עמוק מאוד, אבל הפתרון נמצא ברמה 2 – אז BFS תמיד ימצא אותו במעבר על הרמה השניה, ואילו ל-DFS יקח זמן רב. אם הפתרון נמצא בבנים, יתכן שדווקא DFS יהיה יעיל יותר. לא נכנס במאמר זה למתמטיקה ולכל חישובי **סיבוכיות הזמן** של האלגוריתמים, אבל זו בהחלט נקודה חשובה.



מי מהאלגוריתמים יותר טוב? התשובה היא "תלוי בבעיה". יתרה מזאת, מלבד שני אלגוריתמים אלו קיימים אין ספור אלגוריתמים נוספים. חלקם פותרים בעיות באלגוריתמים שהוצגו – וחלקם מציגים גישות שונות לחלוטין. הבעיה בכל מקרה היא אותה הבעיה – חיפוש כל הצמתים/צומת מטרה) בתוך גרף.

איך משתמשים במה שראינו היום?

כפי שכתבתי ניסיתי במאמרים אלה להכנס מעט יחסית לרקע המתמטי מסביב לאלגוריתמים ולהציגם יותר כ"כלי עבודה". מה אנחנו יודעים עד עכשיו?

- אנחנו רוצים לגרום למחשב לפתור בעיה. בין אם זוהי החלטה על צעד במשחק שאנחנו כותבים, ובין אם זה פתרון של כל בעיה אחרת.
- למדנו שאנחנו צריכים למצוא ייצוג לבעיה, ולמצב של "העולם" - על מנת שהמחשב יוכל לענות על שאלה צריכים לייצג את הבעיה בצורה שהוא מביא. הדגמנו ייצוג בעזרת גרפים, כאשר כל מעבר בין צמתים זו החלטה.
- כשהמידע מאורגן בצורת גרף – צריך לבצע עליו חיפוש, וכך גרמנו למחשב לפתור בעיה! התוצאה של החיפוש תהיה רצף הקשתות (הצעדים) שאנחנו צריכים לעשות על מנת להגיע לפתרון המבוקש. (או אולי – התוצאה תהיה פשוט הערך של הצומת – בהתאם למה שנרצה).

מה הלאה? במאמר הבא נראה דוגמא ספציפית לבעיה, לייצוג ולפתרון בעזרת הטכניקות שהוצגו.

בנוסף נכנס יותר לנושא המגבלות שלנו (זכרון, זמן), לייצוג נכון של הבעיות ונראה גם כיצד מטפלים בבעיות מסובכות יותר על ידי שינוי של האלגוריתמים.