

## מבוא למדעי המחשב 67101

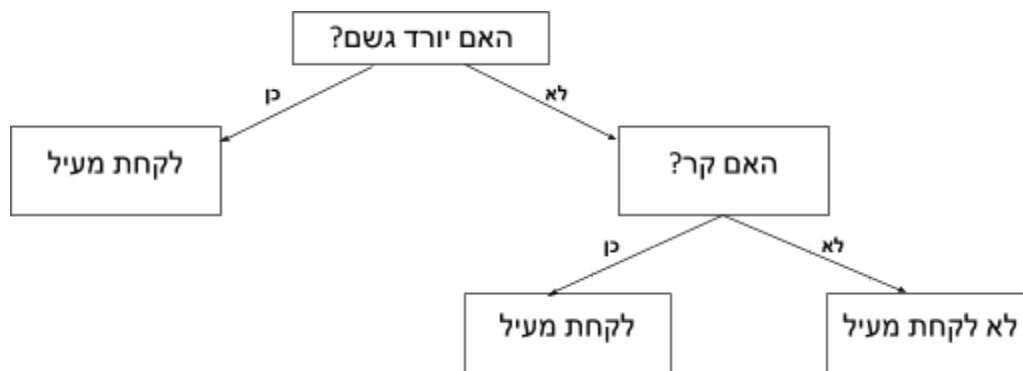
### תרגיל 11 – Decision Trees

להגשה בתאריך 02.01.19 בשעה 22:00

#### הקדמה

בתרגיל זה נתרגל שימוש בעצים כמבנה נתונים. בתרגיל נעבוד עם עצים בינאריים מושרשים, ובפרט נעבוד עם עצי החלטה. עץ בינארי הוא עץ בו לכל קודקוד יש לכל היותר שני בנים. אנו נעסוק בעצים בהם לכל קודקוד יש בדיוק 2 בנים או 0 בנים (כלומר הוא עלה). יש לציין ששורש ללא בנים הוא גם עלה (עץ בעל קודקוד אחד). עץ החלטה הוא עץ בינארי בו כל קודקוד שאיננו עלה מייצג שאלת כן/ולא. לכל קודקוד יש בן אחד המתאים לתשובה "כן" ובן אחד המתאים לתשובה "לא". כשנעבור על קודקודי העץ, בכל שלב נתבונן בשאלה על הקודקוד הנוכחי, נבדוק אם התשובה היא כן או לא, ואז נתקדם לבן המתאים. כל עלה מייצג החלטה שנקבל בהתאם למסלול שעברנו בו.

דוגמה לעץ החלטה פשוט הינה:



בעץ ההחלטה הנ"ל השורש הוא קודקוד המייצג את השאלה "האם יורד גשם?". לקודקוד זה יש שני בנים. הבן השמאלי מתאים לתשובה "כן", ומשום שהוא עלה הוא מייצג החלטה והיא "לקחת מעיל". הבן הימני מתאים לתשובה "לא", והוא מייצג את השאלה "האם קר?". לקודקוד זה שני בנים, הבן השמאלי מתאים לתשובה "כן", ומשום שהוא עלה הוא מייצג החלטה והיא "לקחת מעיל". הבן הימני מתאים לתשובה "לא", ומשום שהוא עלה הוא מייצג החלטה והיא "לא לקחת מעיל".

#### שלד הקובץ ex11.py

על מנת לפתור את התרגיל תקבלו קובץ עם שלד איתו תעבדו.

1. בקובץ זה מוגדרת המחלקה **Node**.

כל אובייקט מסוג **Node** הינו קודקוד בעץ, כאשר **Node** מכיל את השדות הבאים:

- השדה **data**: שדה זה מכיל ערך מסוג **String**. אם ה-**Node** איננו עלה, השדה מייצג את השאלה שנשאלת בקודקוד זה. אם ה-**Node** הינו עלה, השדה מייצג את ההחלטה שהתקבלה בעלה זה.

- השדות **positive\_child** ו- **negative\_child**: אם ה- Node איננו עלה, בשדות אלו יופיעו אובייקטים מסוג Node, כאשר ב- **positive\_child** יהיה הבן התואם לתשובה "כן" על השאלה ב- **data** וב- **negative\_child** יופיע הבן התואם לתשובה "לא". אם ה- Node הינו עלה, בשני השדות יהיה הערך **None**.

שדות המחלקה מתוארים גם בטבלה הבאה:

Field	Type	Description
<b>data</b>	String	Non-leaf node - The question asked at this node. Leaf node - The decision indicated by this leaf.
<b>positive_child</b>	Node	Non-leaf node - The node that matches a positive answer to the question. Leaf node - None.
<b>negative_child</b>	Node	Non-leaf node - The node that matches a negative answer to the question. Leaf node - None

כמו כן, ל- Node יהיה הבנאי (constructor) הבא:

**Node(data, positive\_child, negative\_child)**

הבנאי ייצור אובייקט מסוג Node עם הארגומנטים כשדות.

2. בקובץ זה מוגדרת המחלקה **Record** (מעטה ואילך Record ייקרא גם רשומה).

- Record מכיל את השדות הבאים:

- השדה **illness**: בשדה זה יהיה String עם שם של מחלה כלשהי.
- השדה **symptoms**: בשדה זה תהיה רשימה של אובייקטים מסוג String שכל אחד מהם מייצג סימפטום אפשרי. שימו לב כי רשימה זו יכולה להיות ריקה.

- ל- **Record** יש את הבנאי (constructor) הבא:

**Record (illness, symptoms)**

הבנאי ייצור אובייקט מסוג Record עם הארגומנטים כשדות.

- לדוגמא, ברשומה מסוימת יכולים להיות השדות הבאים:

```
record.illness == "influenza"
```

```
record.symptoms == ["fever", "fatigue", "headache", "nausea"]
```

3. בקובץ תהיה גם הפונקציה :

**parse\_data(filepath)**

הפונקציה תקבל נתיב של קובץ ותחזיר רשימה של אובייקטים מסוג Record. כל איבר ברשימה מייצג חולה אחד שאובחן במחלה **illness** ויכיל את הסימפטומים **symptoms** שנלוו לה. הפורמט של הקובץ הנ"ל הוא: קובץ **txt**, כאשר כל שורה בו מכילה מילים המופרדות ברווח, המילה הראשונה היא שם מחלה והמילים שאחריה יהיו שמות סימפטומים. תיתכן שורה עם מחלה ללא סימפטומים. בנוסף, הקובץ מסתיים בשורה ריקה. תוכלו לראות קבצים לדוגמה בתיקייה **Data** המצורפת לתרגיל זה.

4. בקובץ זה מוגדרת המחלקה **Diagnoser**.  
Diagnoser מכיל את השדה root שבו יהיה Node שמייצג שורש של עץ.  
למחלקה זו יהיה את הבנאי (constructor) הבא:

**Diagnoser(root)**

הבנאי ייצור אובייקט מסוג Diagnoser וישמור את השורש בשדה root.

בחלק א' של התרגיל יהיה עליכם לממש את יתר המתודות במחלקה זו.

הערה: ניתן להוסיף מתודות למחלקות הקיימות בקובץ השלד לפי הצורך.

## חלק א' – שימוש בעץ החלטה

בחלק זה נניח שכבר בנינו עץ החלטה. עץ ההחלטה כולו ייוצג על ידי שורש העץ שהינו אובייקט מסוג Node. שימו  
לב שבקובץ השלד שקיבלתם נבנה עבורכם עץ ידנית ונכתבה דוגמא לשימוש בו (בחלק השני של התרגיל תבנו עצי  
החלטה). כל הפונקציות בחלק זה הן מתודות של המחלקה Diagnoser. המחלקה Diagnoser תקבל שורש של עץ  
החלטה, תשמור אותו כשדה, וכל יתר הפונקציות ישתמשו באותו העץ. חשוב לציין שעל כל המתודות של Diagnoser  
לא לשנות את מבנה העץ או את המידע השמור עליו כלל.

1. ממשו את המתודה:

**diagnose(self, symptoms)**

שמקבלת רשימה של סימפטומים "ומאבחנת" איזו מחלה מתאימה להן לפי עץ ההחלטה השמור ב-self.  
הפונקציה תתחיל מהשורש, תבדוק האם הסימפטום עליו שואל השורש נמצא ברשימת הסימפטומים,  
ותתקדם לבן המתאים. כלומר, אם הסימפטום נמצא ברשימה יש להתקדם לבן שתואם לתשובה "כן", ואם  
הוא לא נמצא ברשימה יש להתקדם לבן שתואם לתשובה "לא". לאחר מכן, יש לחזור על אופן הפעולה עד  
שמגיעים לעלה. הפונקציה תחזיר את המחלה שנמצאת על העלה שהגיעה אליו.

2. ממשו את המתודה:

**calculate\_success\_rate(self, records)**

חישוב אחוז הצלחה של עץ:

המתודה תקבל רשימה של אובייקטים מסוג Record, ותשתמש בשורש של עץ ההחלטה השמור ב-self.  
המתודה תחזיר את היחס בין מספר ההצלחות של העץ על הרשומות ב-records לבין מספר הרשומות  
בסה"כ. כדי לעשות זאת, המתודה תעבור על כל אחת מהרשומות ברשימה records, תחשב דיאגנוזה עבור  
הרשימה symptoms מתוך הרשומה (תוך שימוש בשורש העץ שב-self) ותבדוק האם התקבלה המחלה  
illness מתוך אותה רשומה. המתודה תחלק את מספר הפעמים בהן התקבלה המחלה הנכונה במספר  
הרשומות בסה"כ ותחזיר את התוצאה.

3. ממשו את המתודה:

**all\_illnesses(self)**

המתודה תשתמש בשורש עץ ההחלטה השמור ב-self ותחזיר רשימה של כל המחלות השמורות על עלי  
העץ. כלומר, יש להגיע לכל עלי העץ ולשמור את שדה ה-data של כל עלה. על כל מחלה להופיע ברשימה  
פעם אחת בלבד. כמו כן, על הרשימה להיות ממוינת על פי שכיחות המחלות בעץ. מחלה שמופיעה הכי  
הרבה פעמים בעלי העץ תהיה הראשונה ברשימה, ומחלה שמופיעה הכי מעט פעמים בעלי העץ תהיה  
האחרונה ברשימה. עבור מספר מחלות עם שכיחות זהה (מופיעות מספר זהה של פעמים בעלי העץ),  
הסדר ביניהן לא משנה. ניתן וכדאי להשתמש באלגוריתמים שנלמדו בכיתה למעבר על קודקודי עץ.

4. ממשו את המתודה:

**most\_rare\_illness(self, records)**

המתודה תקבל רשימה של records כארגומנט, תעבור על כל אחת מהרשומות ותבדוק לאיזו מחלה נגיע אם נתקדם בעץ לפי רשימת הסימפטומים שלה (כלומר, איזו מחלה העץ מאבחן עבורה). המתודה תעשה זאת עבור כל רשומה ולבסוף תחזיר את המחלה אליה הגענו הכי מעט פעמים. שימו לב כי ייתכן שיש יותר מעלה אחד המייצג כל מחלה.

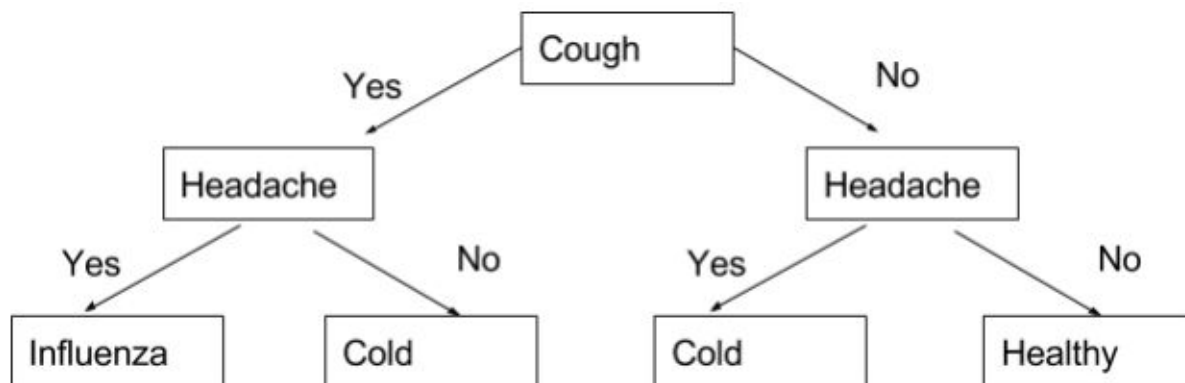
5. ממשו את המתודה:

**paths\_to\_illness(self, illness)**

שמקבלת ארגומנט illness מסוג String.

- המתודה תעבור בכל המסלולים בעץ ותחזיר רשימה של כל המסלולים שמגיעים לעלה שעליו המחלה illness.
- כדי לייצג מסלול נשים לב שמספיק להגיד מה התשובה שנתנו בכל אחד מקודקודי העץ. כלומר, עבור כל מסלול שמסתיים במחלה illness מתאימה רשימה של ערכי True, False כך שבמקום ה- $i$  ברשימה יהיה True אם בצעד ה- $i$  ענינו "כן", ואחרת במקום ה- $i$  ברשימה יהיה False. שימו לב כי אנו מתחילים לספור את הצעדים בצעד ה-0.
- חשוב לציין כי ייתכן שיהיו כמה מסלולים שיגיעו לאותה המחלה (בעלים שונים בעץ). לכן, ערך ההחזרה אמור להיות רשימה של רשימות. בכל אחת מהרשימות הפנימיות יהיו רק ערכי True, False המייצגים את המסלולים. אם אין אף מסלול המגיע למחלה, יש להחזיר רשימה ריקה. אין חשיבות לסדר הופעת הרשימות.

- לדוגמא, עבור העץ הבא:



הקריאה:

`paths_to_illness(self, "Cold")`

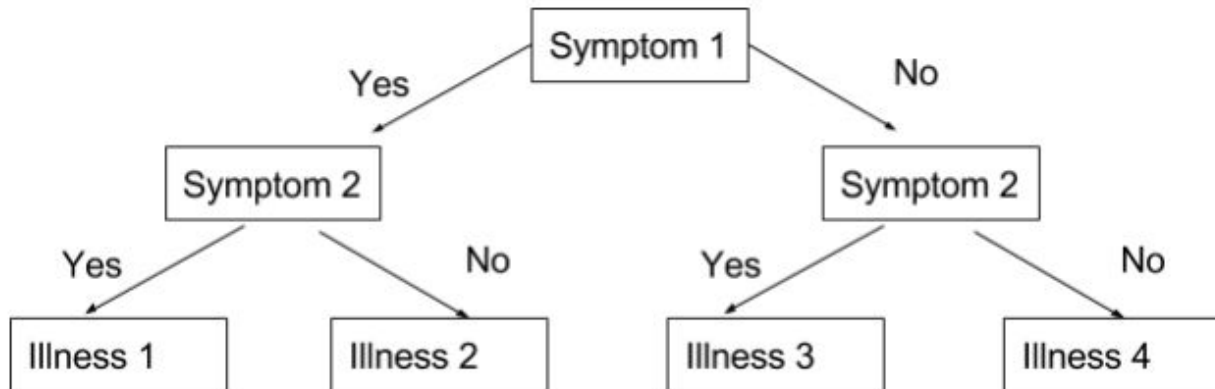
תחזיר את הרשימה:

`[[True, False], [False, True]]`

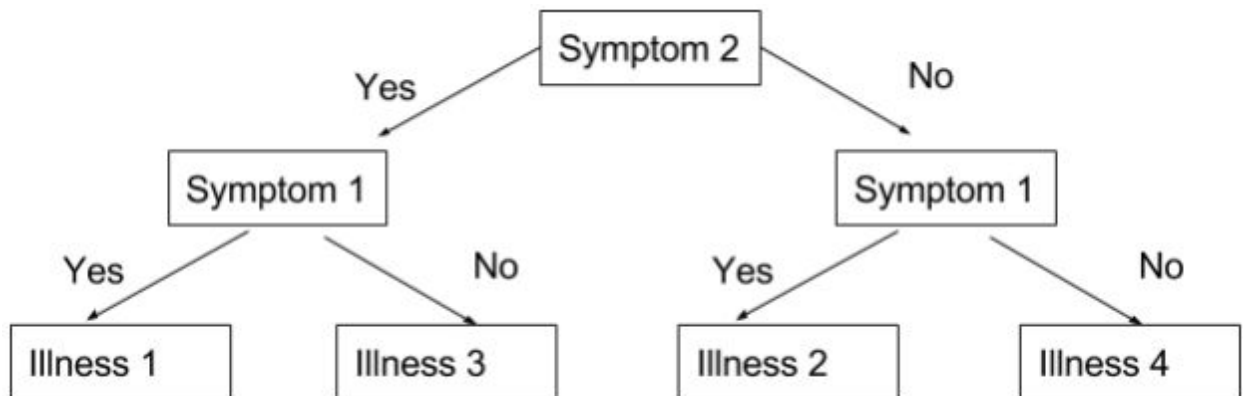
## חלק ב' - בניית עץ החלטה

בחלק זה נבנה עץ החלטה עם אחוז הצלחה גבוה עבור המידע שהתקבל מ-`data_parse`. שימו לב כי אם נשאל על כל אחד מהסימפטומים האפשריים נוכל יחסית בקלות לבנות עץ החלטה אופטימלי (כיצד היינו עושים כזה דבר?). לכן, נרצה להגביל את עצי ההחלטה שלנו לעצים ששואלים על מספר קטן יחסית של סימפטומים.

כמו כן, שימו לב שאם עץ תמיד שואל על רשימת סימפטומים כלשהי, `symptoms`, לפני שהוא מגיע לעלה, סדר השאלות לא משנה. כלומר העץ:



והעץ:



יחזירו בדיוק את אותו אחוז שגיאה. שכנעו את עצמכם שקביעה זו נכונה.

שימו לב כי הפונקציות הבאות (סעיפים 6-7) אינן מתודות של `Diagnoser`.

6. ממשו את הפונקציה:

**`build_tree(records, symptoms)`**

`records` זו רשימה של אובייקטים של המחלקה `Record`, ו-`symptoms` זו רשימה שכל איבריה מסוג `string`. הפונקציה תבנה עץ השואל בדיוק על הסימפטומים ברשימה `symptoms` לפי סדר הופעתם ברשימה, ותחזיר את שורש העץ שנבנה. כלומר, בשורש ישאלו על `symptoms[0]`, בבניו ישאלו על `symptoms[1]`, וכן

הלאה. ניתן לעשות זאת בכך שבשורש העץ נשים את הערך הראשון ברשימה, `symptoms[0]`, ואז בכל אחד מבניו נבנה תת עץ השואל על כל יתר הסימפטומים ב- `symptoms`. באופן זה, בכל עומק בעץ כל הצמתים המקבילים יבדקו את אותו הסימפטום. שימו לב כי זוהי הגדרה רקורסיבית. הערה - ניתן להניח שהרשימות `records, symptoms` לא ריקות. כמו כן ניתן להניח שאין חזרות ברשימה `symptoms` (כל סימפטום יופיע לכל היותר פעם אחת ברשימה).

כשנגיע לעלה בעץ נצטרך לבחור מחלה כלשהי שהיא האבחנה שמספק העץ במקרה זה. "המתמודדים" מבין כל המחלות הן המחלות שמופיעות ברשומות שתואמות את המסלול משורש העץ עד לעלה. נאמר שרשומה תואמת למסלול אם כל הסימפטומים עליהם אמרנו "כן" לאורך המסלול מופיעים ברשימת הסימפטומים ברשומה וכל הסימפטומים עליהם אמרנו "לא" לא מופיעים ברשימת הסימפטומים. כדי למזער את השגיאה של דיאגנוזה עתידית, נבחר לבנות את העץ כך שבכל עלה תופיע המחלה המופיעה במספר המרבי של רשומות התואמות למסלול שמתחיל משורש העץ ומגיע עד לאותו עלה. שימו לב כי העץ לא בהכרח יכיל את כל הסימפטומים או המחלות המופיעים ב- `records`.

אם עבור עלה מסויים נתקלנו במקרה של שיוויון בין מספר מחלות שונות (כלומר, יש יותר ממחלה אחת שמופיעה במספר המרבי של רשומות התואמות למסלול שמתחיל משורש העץ ומגיע עד לאותו עלה), אז ניתן לבחור כל אחת מהמחלות הנ"ל. בדומה, במקרה בו לא נמצאה אף רשומה תואמת למסלול משורש העץ עד לעלה, ניתן לבחור בכל אחת מהמחלות שמופיעות ברשומות.

## 7. ממשו את הפונקציה

### `optimal_tree(records, symptoms, depth)`

- ניתן להניח כי הרשימות `records, symptoms` לא ריקות וכי  $len(symptoms) \geq depth > 0$ .
- הפונקציה תחזיר את העץ (כלומר אובייקט מסוג `Node` שהוא שורש העץ) עם אחוז ההצלחה הגבוה ביותר שתמיד שואל על מספר סימפטומים השווה ל- `depth`.
- כדי לעשות זאת, הפונקציה תעבור על כל תתי הקבוצות בגודל `depth` של קבוצת הסימפטומים, `symptoms`, תבנה עץ השואל בדיוק על הסימפטומים בתת הקבוצה שנבחרה, ותבדוק את אחוז ההצלחה על אותו העץ (עבור חישוב אחוז הצלחה - בנינו פונקציה בחלק הראשון של התרגיל).
- ניתן להשתמש בפונקציה `combinations` כדי לבנות את כל תתי הקבוצות בגודל מסויים. ניתן לקרוא על הפונקציה ולראות דוגמאות לשימוש בה בקישור הבא:  
<https://docs.python.org/3.6/library/itertools.html#itertools.combinations>
- שימו לב שהפונקציה לא מחזירה רשימה אלא איטרטור (`iterator`). אתם יכולים להתמודד עם ערך ההחזרה שלו ע"י המרתו ל- `list`, או להשתמש בו בלולאת `for` בדרך הבאה:  
`for x in itertools.combinations():`
- ...
- בסוף התהליך הפונקציה תחזיר את השורש של העץ עם אחוז ההצלחה הגבוה ביותר.
- ניתן לבנות עצים אלו בעזרת הפונקציה `build_tree`.

דוגמא עבור הפונקציות 7,6:

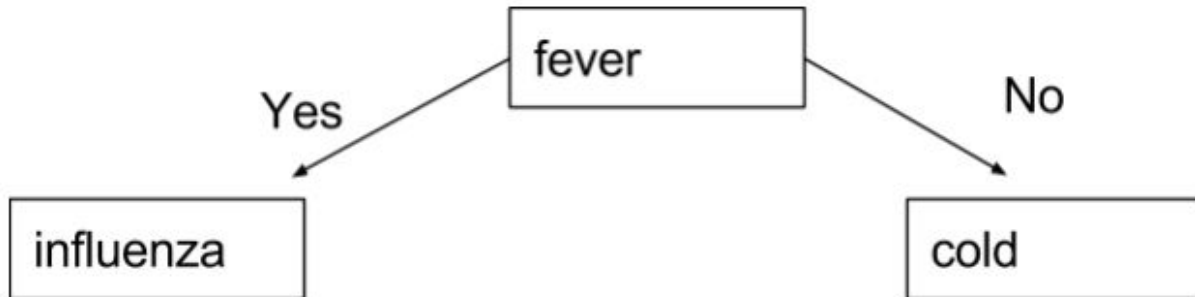
עבור הרשימה `records` הבאה:

```
record1 = Record("influenza", ["cough", "fever"])
record2 = Record("cold", ["cough"])
records = [record1, record2]
```

נצפה ששתי הקריאות:

```
build_tree(records, ["fever"])  
optimal_tree(records, ["cough", "fever"], 1)
```

יחזירו את שורש העץ הבא:



## נהלי הגשת התרגיל:

עליכם להגיש קובץ zip הנקרא ex11.zip אשר מכיל את הקבצים הבאים:

1. ex11.py - עם המימושים שלכם לפונקציות ולמתודות השונות שהוזכרו בתרגיל.
2. README

בהצלחה! :