מבוא לבינה מלאכותית - 236501

אביב 2023

ת"ב מספר 2

מגישים:

מתן קרקסוני 212644967

שלמה שלו בטסיקס 213016348

חלק א' – ImprovedGreedy

הערה: בסעיפים 1, ו- 2 נניח שאנחנו רובוט מספר 0, והיריב רובוט מספר 1 (בה"כ) – אך באופן כללי ניתן בקלות להגדיר נוסחאות כמעט זהות למקרה ההפוך, ולבדוק מה המספר המזהה שלנו, ולענות לפיו...
הערה זו באה לידי ביטוי בהגדרת G, ובכל רכיביי הגדרת היוריסטיקה h.

1. נגדיר את הבעיה בתור הרביעייה הבאה: (S, O, I, G), כך ש:

$$S = \{((x_1, x_2, \dots, x_{25}), credit_0, credit_1, battary_0, battary_1, package_0, package_1, steps) \mid x_1, x_2, \dots, x_{25} \in \{R_0, R_1, P_0, P_1, D_0, D_1, C_0, C_1, \emptyset\} \wedge credit_0, credit_1, battary_0, battary_1, steps \in \mathbb{N} \wedge package_0, package_1 \in \{0, 1, \emptyset\}\}$$

כל מה שצריך בשביל לאפיין מצב במשחק הוא:
המצב של הלוח (שכולל בעצם את המיקום של כל אובייקט),
המאפיינים של השחקנים (נקודות שנצברו עד כה, סוללה שנותרה, ואיזו (ואם בכלל) חבילה הם נושאים),
וכמות הצעדים שנותרו לשחק.

$$O = \{move\ north, move\ south, move\ east, move\ west, pick\ up, drop\ off, charge\}$$

אלו בעצם כל הפעולות ששחקן מסוים (רובוט) יכול לעשות.

$$I = \left(\begin{pmatrix} R_1 & P_0 & \emptyset & C_1 & C_0 \\ D_1 & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & P_1 & \emptyset & \emptyset & \emptyset \\ R_0 & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & D_0 & \emptyset \end{pmatrix}, 0, 0, 20, 20, \emptyset, \emptyset, 200 \right)$$

המצב ההתחלתי רנדומלי...
כלומר חלק ממנו נקבע רנדומלית (למשל מיקומים של אובייקטים), וחלק ממנו קבוע (למשל העובדה שהרובוטים תמיד מתחילים עם 0 נקודות, ובלי חבילה ברשותם).
אבל בכל זאת כתבנו דוגמה קונקרטית – אשר לקוחה מהמצב ההתחלתי שמג'ונרט כאשר מגדירים את המערכת כפי שהדגמתם בקובץ התרגיל.

$$G = \{((x_1, x_2, \dots, x_{25}), credit_0, credit_1, battary_0, battary_1, package_0, package_1, steps) \in S \mid (battary_0 = 0 \vee battary_1 = 0 \vee steps = 0) \wedge (credit_0 > credit_1)\}$$

כל מצב שבו המשחק נגמר (שזה קורה אם נגמרה לשחקן מסוים הסוללה, או אם בוצעו כבר כמות הצעדים שאפשרנו לבצע), וגם צברנו יותר ניקוד מהשחקן היריב (הניקוד שלנו גדול משלו).

מטרת כל רובוט לצבור יותר ניקוד מהרובוט האחר עד סוף המשחק, המשחק נגמר כאשר לאחד מהרובוטים נגמרת הסוללה או כאשר נגמר מספר הצעדים המקסימלי לכל רובוט (ערך מוגדר מראש, דוגמה בהמשך).

הערה:
אם התכוונתם לשאול על קבוצת מצביים סופיים שלא בהכרח רצויים לנו (כלומר כולל המצבים שבהם המשחק נגמר והפסדנו בו), אז ההגדרה של *G* אמורה להיות:

$$G = \{((x_1, x_2, \dots, x_{25}), credit_0, credit_1, battary_0, battary_1, package_0, package_1, steps) \in S \mid (battary_0 = 0 \vee battary_1 = 0 \vee steps = 0)\}$$

2. נסמן את מרחק מנהטן בין 2 נקודות p_1, p_2, בשם $m(p_1, p_2)$,

לכל $s \in S$, כך ש:

$$s = \left((x_1, x_2, \dots, x_{25}), credit_0, credit_1, battary_0, battary_1, package_0, package_1, steps\right)$$

<u>ניעזר בסימונים הבאים:</u>

r - מיקום הרובוט שלנו,

p_i - המיקום של P_i על הלוח $(x_1, x_2, \dots, x_{25})$ (בלוח של 5x5 כמובן...),

c_i - המיקום של C_i על הלוח $(x_1, x_2, \dots, x_{25})$ (בלוח של 5x5 כמובן...),

d_i - המיקום של D_i על הלוח $(x_1, x_2, \dots, x_{25})$ (בלוח של 5x5 כמובן...).

ונגדיר את היוריסטיקה באופן הבא:

$$smart\_heuristic(s) = 15 * credit_0 + 16 * battary_0 + total(s)$$

כך ש:

הפונקציה $total$ מוגדרת באופן הבא:

$$total(s) = \begin{cases} 15 - m(r, d_i) & ; if\ package_0 = i \neq \emptyset \\ -dist\_to\_station(s) & ; if\ package_0 = \emptyset\ and\ battery_0 \leq 5 \\ -3 * dist\_to\_package(s) & ; else \end{cases}$$

כך ש:

הפונקציה $dist\_to\_station$ מוגדרת באופן הבא:

$$dist\_to\_station(s) = \min\left(m(r, c_0), m(r, c_1)\right)$$

והפונקציה $dist\_to\_package$ מוגדרת באופן הבא:

$$dist\_to\_package(s) = \min(\{m(r, p_i) \mid P_i\ is\ on\ the\ board(not\ picked\ yet \dots)\})$$

<u>הסברים על המרכיבים:</u>

- <u>$smart\_heuristic$</u>

<u>$15 * credit_0$</u> – נרצה תמיד להיות בעלי כמה שיותר נקודות (טריוויאלי) רכיב זה מיועד בשביל שהסוכן יתעדף את הפעולה של הורדת חבילה ביעד שלה כאשר זה מתאפשר לו

<u>$16 * battary_0$</u> – נרצה להיות בעלי כמה שיותר סוללה (בשביל שנוכל לבצע יותר צעדים ובעזרתם לצבור עוד נקודות בהמשך). נציין שהמקדם גבוה יותר בשביל המקרים שבהם הגענו לעמדת טעינה והסוכן בודק האם כדאי להשקיע את הנקודות שלו לטובת סוללה, ואנו רוצים לומר לו שכן

- <u>$total$</u>

☒ נוסיף תעדוף (של 15 נקודות בונוס) <u>לכל המצבים שבהם יש ברשותנו חבילה</u> שאספנו (כי בלי איסוף חבילות לא נצבור נקודות...)
ובנוסף לכך, במצבים אלו, נרצה להתקרב ככל האפשר אל היעד של החבילה שלנו (בכך שנקטין את "הקנס" – שהוא מינוס המרחק שלנו מהיעד הנ"ל)

☒ <u>במצבים שבהם אין לנו חבילה</u>, נפצל למקרים לפי אם יש לנו מעט סוללה או הרבה:

☒ <u>אם יש מעט:</u> נרצה להטעין, אחרת הסוללה תגמר לפני שנספיק לצבור עוד נקודות...
אז נתקדם אל עבר תחנת ההטענה הקרובה אלינו ביותר (להקטין את $dist\_to\_station$)

☒ <u>ואם יש מספיק סוללה:</u> נתקדם במרץ אל עבר החבילה הקרובה אלינו ביותר, אשר נמצאת על הלוח (לא נאספה כבר ע"י הרובוט השני) – בא לידי ביטוי בעזרת ההקטנה של $dist\_to\_package$.

3. רטוב
4. .

4.1. גישה חמדנית עלולה לגרום לנו לפספס את הפתרון האופטימלי,
בכך שתוביל אותנו לעבר פתרון שמורכב מצעדים שיועילו לנו בטווח הקצר (של הצעד הנוכחי),
ולא בטווח הארוך (של ניצחון במשחק).
זאת לעומת minimax אשר משכלל את צעדיי היריב העתידיים ואת צעדי הסוכן שלנו העתידיים
ומתחשב בהם (בצעדים העתידיים) במהלך התכנון והבחירה של הצעד הנוכחי הטוב ביותר
האפשרי.

4.2. בנוסף לכך, (ספציפי במקרה הזה!) אנו התייחסנו לסוכן שלנו בלבד, ול-מה יהיו הפעולות שיועילו
לנו,
לעומת minimax אשר ישכלל זאת בנוסף להחלטות היריב, שירצה להועיל לעצמו ולהרע לנו,
ויחשוב בצורה יותר מתוחכמת ומתוכננת את הצעד הנוכחי,
כי ייקח גם את הפרמטר הזה בחשבון בניגוד אלינו.

חלק ב:

1) היתרונות של שימוש ביוריסטיקה קלה לעומת קשה:
- המימוש של היוריסטיקה פשוט יותר.
- היוריסטיקה קלה לחישוב, ולכן יש פחות תקורה על החישוב שלה בזמן הריצה ובפרט זמן החישוב שלה קצר יותר. כלומר, נוכל במסגרת הזמן שלנו (האלגוריתם שלנו מוגבל משאבים) להגיע עמוק יותר בעץ ולבדוק יותר מצבים.

החסרונות של שימוש ביוריסטיקה קשה לעומת קלה:

- היוריסטיקה מיודעת יותר, ולכן נבצע פחות צעדים מיותרים והחיפוש יהיה קצר יותר ובפרט הפתרון יכול להיות אופטימלי יותר.
- נוכל להגיע לתוצאות טובות יותר עבור עומק קטן יותר כיוון שהיא מיודעת.

2) לא בהכרח קיים באג באלגוריתם. נשים לב שאלגוריתם minimax משתמש ביוריסטיקה על מנת לבחור את הצעד הבא שלו ויתכן שהיוריסטיקה שדנה בחרה בה אינה מספיק מיודעת (או שאינה מתעדפת ניצחון), מה שמוביל לכך שהחישוב שביצעה על הפעולה שמובילה לניצחון וגם על פעולה אחרת שאינה מובילה לניצחון הניב את אותו הערך ולכן האלגוריתם בחר לבצע את הפעולה השנייה.

3) רטוב 😊

4) השינויים שנצטרך לבצע עבור כל סעיף:

- במקרה זה, כל סוכן (כולל אותנו) רוצה לנצח ולכן בכל צעד הוא יבחר את הערך שיביא אותו למקסימום ולא יהיה אכפת לו מה תהיה ההשפעה של בחירת ערך זה עבור סוכנים אחרים, כיוון שהוא רוצה לקדם את עצמו לניצחון. כאשר בכל פעם בין התורות שלנו יהיו k-1 תורות, ובכל תור כל אחד מהסוכנים יבחר את הערך המקסימלי עבור עצמו ובנוסף <u>גם אנחנו</u> בתור שלנו נבחר את הערך המקסימלי עבור עצמנו על מנת להתקדם לניצחון. לכן סך הכל השינויים שנצטרך לבצע הם שכעת בתור שלנו אנחנו נבחר את הערך המקסימלי עבורנו (מטרתנו הינה לנצח).
- במקרה וכל סוכן רוצה שלא ננצח, בכל פעם כשיגיע תורם הם כולם יבחרו את הצעד שיביא לערך מינימלי של השחקן שלנו ללא התחשבות בהתקדמות שלהם לניצחון, אלא רק מתוך מטרה לגרום לנו לא לנצח. כך שבין כל תור שלנו יהיו k-1 תורות שלהם שהם יבחרו את הערך המינימלי עבור השחקן שלי בכל פעם, <u>ובתור שלנו</u> אנחנו נבחר את הערך שיביא למקסימום עבור השחקן שלנו כי אנו רוצים לנצח (כלומר תור אחד של max, ו-k-1 תורות של min).
- בהנחה שכל סוכן (כולל אותנו לפי ההבהרה בפיאצה) רוצה שהסוכן שאחריו בתור ינצח, בכל פעם שיגיע תורו הוא יבחר את הערך שיביא למקסימום את השחקן שבתור אחריו, ולכן בכל צעד יבחר הערך שיביא למקסימום את השחקן שבתור אחרי, וכיוון שגם הסוכן שלנו רוצה שהשחקן שאחריו בתור ינצח, גם אנחנו בתורנו נבחר את הערך שיביא למקסימום את השחקן שאחרינו.

חלק ג:

1) רטוב 😊
2) <u>מבחינת זמן ריצה:</u>

מטרת הסוכן שמימשנו בחלק זה הינה להגיע לפתרון תוך זמן אופטימלי ולחסוך בצעדים מיותרים על ידי גיזום ענפים שפיתוחם בהכרח לא ישנה את הפתרון. לעומת זאת, בחלק הקודם מימשנו סוכן מינימקס מוגבל משאבים – כלומר סוכן המינימקס שמימשנו בחלק הקודם רץ תחת מגבלת זמן הקבועה מראש ולכן זמן הריצה שלו ידוע לנו.

<u>מבחינת בחירת מהלכים:</u>

אלגוריתם מינימקס מוגבל משאבים יבחר את המהלך בעל הערך היוריסטי הגבוה ביותר מבין אלו שהוא הצליח לחשב תחת מגבלת הזמן הנתונה לו. לעומת זאת, עבור הסוכן שאנו מימשנו בסעיף זה <u>אנו נריץ מכל מצב</u> שאנו נמצאים בו את האלגוריתם אלפא בטא לכל מהלך אפשרי של הסוכן על מנת לחשב את המהלך שיביא לנו את הערך המקסימלי (גם אלגוריתם זה משתמש ביוריסטיקה), ונבחר במהלך זה (קיצוץ הענפים מאפשר חיסכון בחישובים ולכן יוכל להתבצע חיפוש טוב יותר).

חלק ד' - Expectimax

1. נניח שיש x מקרים שהסוכן יכול לבחור, והסוכן משחק באופן רנדומלי לחלוטין,
אז נגדיר את ההסתברות שהוא יבחר בכל מקרה מבין x המקרים, להיות: $p = \frac{1}{x}$
כלומר לכל מקרה ניתן הסתברות שווה להיבחר.
2. נשים לב שבסעיף הנ"ל מוגדרים לנו חסם עליון ותחתון לערכיי היוריסטיקה h:

$$\boxed{\forall s: -1 \leq h(s) \leq 1}$$

נוכל לנצל את העובדה הזו בשביל לבצע גיזום לאלגוריתם:

- צומת max
בכל פעם שנתקל בצומת בעלת הערך 1, במהלך חיפוש ערך מקסימלי מבין כל הבנים, נדע שמצאנו ערך מקסימלי
כי 1 הוא החסם העליון של היוריסטיקה, ולכן לא אפשרי למצוא צומת בעלת ערך גדול ממנו בהמשך (אם נמשיך לחפש לא נמצא...)
גם במקרה של חישוב ערך צומת בעזרת תוחלת של ערכי Expectimax רקורסיביים, עדיין כל אחד מהם יהיה לכל היותר 1, ולכן גם התוחלת עצמה.
ולכן נסיק שמצאנו את הערך המקסימלי מבין ערכיי הבנים, וזה מה שחיפשנו, ולכן נוכל לעצור את החיפוש – לגזום את כל שאר הבנים...
- צומת min
באופן דומה, בכל פעם שנתקל בצומת בעלת הערך (-1), במהלך חיפוש ערך מינימלי מבין כל הבנים, נדע שמצאנו ערך מינימלי
כי (-1) הוא החסם התחתון של היוריסטיקה, ולכן לא אפשרי למצוא צומת בעלת ערך קטן ממנו בהמשך (אם נמשיך לחפש לא נמצא...)
גם במקרה של חישוב ערך צומת בעזרת תוחלת של ערכי Expectimax רקורסיביים, עדיין כל אחד מהם יהיה לכל הפחות (-1), ולכן גם התוחלת עצמה.
ולכן נסיק שמצאנו את הערך המינימלי מבין ערכיי הבנים, וזה מה שחיפשנו, ולכן נוכל לעצור את החיפוש – לגזום את כל שאר הבנים...

3. רטוב

חלק ה:

1) נחשב את הנדרש עבור כל שינוי:

- נבחין שמקדם הסיעוף הנוכחי לא ישתנה. ראשית, אם נגדיל את הלוח ל-8x8 מקדם הסיעוף לא ישתנה, כיוון שעדיין כל הפעולות יהיו תקפות כמו מקודם. אם בנוסף להגדלת הלוח נוסיף מחסומים לסביבה, במקרה הטוב מקדם הסיעוף יקטן (לא נוכל לזוז לכיוונים מסוימים), אך במקרה הגרוע מקדם הסיעוף עדיין יישאר כמו שהוא (נוכל לבצע את אותן הפעולות).
  מקדם הסיעוף של הרובוט במקרה זה: 6 – ההסבר לכך הינו שהרובוט יכול לעלות למעלה, ללכת ימינה או שמאלה, לרדת למטה, לחנות ובנוסף הוא יכול לבצע אחת מהפעולות שהן להניח חבילה, להרים חבילה, או להטעין (לא ניתן לעשות יותר מפעולה אחת במצב מסוים מתוך שלושת הפעולות האחרונות שציינתי מעצם הגדרת המשחק).
- כעת, נוספת עוד פעולה שהרובוט יכול לבצע בכל משבצת (אם היא ריקה), והיא להניח עליה בלוק. נשים לב, שכעת במקרה הגרוע מקדם הסיעוף גדל, כיוון שבכל תור יש לרובוט יותר אופציות שהוא יוכל לבצע (למעלה, למטה, ימינה, שמאלה, לאסוף חבילה, להוריד חבילה, להטעין, להניח בלוק על הלוח). בנוסף, הרובוט יכול להניח את הבלוק על כל משבצת שהוא רק רוצה והוא לא צריך להיות עליה או אפילו קרוב אליה. לכן מקדם הסיעוף יגדל לסדר גודל של כמות המשבצות הריקות בנוסף לגודלו הקודם (כיוון שרובוט יכול להוריד בלוק בכל משבצת ריקה – גם אם אינה קרוב אליה).
  מקדם הסיעוף של הרובוט במקרה זה: 27 – נציג את החישוב (נניח כי אין חבילות אשר מונחות על הקרקע כיוון שאנו מחפשים מספר מקסימלי של פעולות בחישוב זה וגם נניח כי משבצות אשר מסמנות יעדים נחשבות ריקות – כלומר ניתן להניח עליהן בלוק):

$$\underbrace{5\cdot 5}_{grid\ size} - \underbrace{2}_{2\ robots} - \underbrace{2}_{charging\ stations} + \underbrace{6}_{possible\ actions\ (original)} = 27$$

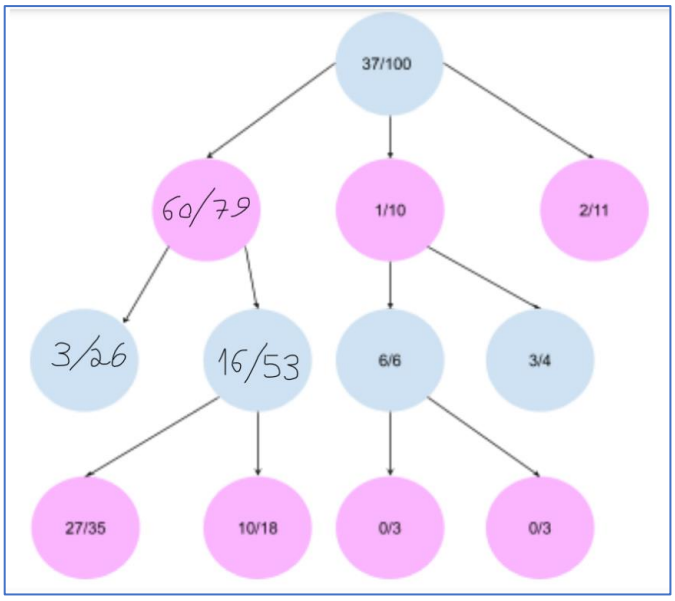
2) נניח ומימשנו את השינוי הנדרש:

- נבחין שלפי המימוש של Improved Greedy, זמן הריצה שלו תלוי במקדם הסיעוף בלבד, בניגוד לשאר האלגוריתמים בהם זמן הריצה תלוי גם במקדם הסיעוף וגם בעומק המירבי של העץ. ולכן, כפי שהסברנו לעיל, בהינתן השינוי שהתווסף מקדם הסיעוף גודל, ובפרט עבור שאר האלגוריתמים $O(b^m)$ יגדל משמעותית יותר מאשר $O(b)$ עבור Improved Greedy וסה"כ נקבל שעבור אלגוריתם זה זמן הריצה שלו יהיה סביר גם לאחר השינוי (נבחין שמקדם הסיעוף גדל בסדר גודל של מספר המשבצות הריקות, וכיוון שאנו לא מעלים אותו בחזקת העומק המירבי זמן הריצה של האלגוריתם עדיין יהיה סביר). בנוסף נשים לב שבמקרה שלנו, ה-Greedy מסתכל אך ורק על השכנים הקיימים במצב הנוכחי ומתוכם הוא בוחר את האופרטור, ולא מבצע קריאה רקורסיבית או ממשיך לסרוק לעומק.
- על מנת להתמודד עם האתגר שנוצר בעקבות הוספת השינוי, נוכל להשתמש באלגוריתם MCTS אשר ראינו בהרצאות. בחרנו באלגוריתם זה כיוון שכפי שלמדנו, אלגוריתם זה מתמודד טוב כאשר יש לנו מקדם סיעוף גדול וכפי שהסברנו לעיל, בעת הוספת השינוי מקדם הסיעוף שלנו גדל ולכן אנו רוצים אלגוריתם אשר יוכל להתמודד טוב עם שינוי זה (MCTS 😊).

האלגוריתם MCTS טוב לבעיות עם מקדם סיעוף גדול כיוון שהוא מתרכז באזורים "מבטיחים" במרחב החיפוש (בעצם מתרכז באזורים עם פוטנציאל גבוה). על ידי כך, MCTS מוריד את הצורך לבצע חיפוש ממצה. בעצם, האלגוריתם מבצע איזון בין exploration ו-exploitation באמצעות דגימה אקראית וזאת על מנת לחקור טווח רחב של אופציות שונות של מהלכים באופן אפקטיבי – מה שמוביל לכך שהוא מתמודד טוב עם בעיות בעלות מקדם סיעוף גדול.

חלק ו' - יבש - שאלה פתוחה - MCTS

1. .



2. בהינתן ש: $C = \sqrt{2}$, והנוסחה הבאה שעליה למדנו בביתה:

$$\text{UCB1(s)} = \underbrace{\frac{U(s)}{N(s)}}_{\text{explotation}} + C \times \underbrace{\sqrt{\frac{\ln(N(s.\text{parent}))}{N(s)}}}_{\text{exploration}}$$

נחשב עבור כל פעולה את כל הצמתים:

- ❖ פעולה ראשונה:
  - **צומת שמאלית:** $\frac{60}{79} + \sqrt{2} \times \sqrt{\frac{\ln(100)}{79}} = 1.100941452$
  - צומת אמצעית: $\frac{1}{10} + \sqrt{2} \times \sqrt{\frac{\ln(100)}{10}} = 1.059705182$
  - צומת ימנית: $\frac{2}{11} + \sqrt{2} \times \sqrt{\frac{\ln(100)}{11}} = 1.09686117$
  - ❖ הצומת השמאלית היא בעלת הערך המקסימלי ולכן היא תיבחר
- ❖ פעולה שנייה:
  - צומת שמאלית: $\frac{3}{26} + \sqrt{2} \times \sqrt{\frac{\ln(79)}{26}} = 0.6951357458$
  - **צומת ימנית:** $\frac{16}{53} + \sqrt{2} \times \sqrt{\frac{\ln(79)}{53}} = 0.7079469168$
  - ❖ הצומת הימנית היא בעלת הערך המקסימלי ולכן היא תיבחר
- ❖ פעולה שלישית:
  - **צומת שמאלית:** $\frac{27}{35} + \sqrt{2} \times \sqrt{\frac{\ln(53)}{35}} = 1.247741309$
  - צומת ימנית: $\frac{10}{18} + \sqrt{2} \times \sqrt{\frac{\ln(53)}{18}} = 1.219741934$
  - ❖ הצומת השמאלית היא בעלת הערך המקסימלי ולכן היא תיבחר

3. עבור הפעולה הראשונה, ישנן 3 אפשרויות (צומת שמאלית, אמצעית, וימנית),
מבין אלו שלא נבחרו (האמצעית והימנית) – הצומת בעלת ערך גדול יותר היא הימנית, אז מכיוון שאנחנו מחפשים כמות מינימלית של ניצחונות נוספים (שנסמן ב- n) שיובילו לכך שצומת אחרת תיבחר – נשתמש בה בתור זאת שתיבחר בסוף (תהיה בעלת הערך הגדול ביותר בסוף)...
אז בשביל שהימנית תיבחר – צריך שהיא תהיה בעלת הערך הגדול יותר מבין השלושה.
נניח יש עוד n ניצחונות (ונחפש את ה- n המינימלי), זה אומר שיש n+100 משחקים סה"כ עכשיו, וזה אומר שהמשחקים האלה עכשיו צריכים לבוא לידי ביטוי גם בערכים שיש לצמתים נוספים, ליתר דיוק, עכשיו יש n+79 משחקים בצומת השמאלית (ואותם כמות ניצחונות, כי צומת זו שייכת ליריב).
נבדוק מהו ערך ה- *n* המינימלי שעבורו ערך הצומת הימנית יהיה גדול יותר:

<u>עבור *n=0*:</u> נשאר במצב המקורי ללא שינוי...
<u>ננסה לבדוק עבור *n=1*:</u> ואכן נקבל ש:

❖ פעולה ראשונה:

- צומת שמאלית: $\frac{60}{n+79}+\sqrt{2}\times\sqrt{\frac{\ln(n+100)}{n+79}}=\frac{60}{80}+\sqrt{2}\times\sqrt{\frac{\ln(101)}{80}}=1.089673392$
- צומת אמצעית: $\frac{1}{10}+\sqrt{2}\times\sqrt{\frac{\ln(n+100)}{10}}=\frac{1}{10}+\sqrt{2}\times\sqrt{\frac{\ln(101)}{10}}=1.060741434$
- **צומת ימנית:** $\frac{2}{11}+\sqrt{2}\times\sqrt{\frac{\ln(n+100)}{11}}=\frac{2}{11}+\sqrt{2}\times\sqrt{\frac{\ln(101)}{11}}=1.097849197$

נשים לב שאכן צומת אחרת (הימנית) עכשיו בעלת הערך הגדול ביותר, ולכן היא תבחר.
כלומר מצאנו את מספר הניצחונות המינימלי (שהוא 1) הדרוש כדי שצאצא של אב קדום ביותר אחר יבחר בשלב ה- *selection*.

4. אם נניח שהנוסחה שעליה למדנו בכיתה:

$$\mathrm{UCB1}(s)=\underbrace{\frac{U(s)}{N(s)}}_{\text{explotation}}+C\times\underbrace{\sqrt{\frac{\ln(N(s.\text{parent}))}{N(s)}}}_{\text{exploration}}$$

נשארת קבועה ולא ניתן לשנותה, והדבר היחיד שניתן לשנות הוא את $N(s)$.
ונרצה להוביל לכך שהרכיב שאחראי לביצוע exploration יהיה גדול יותר מהרכיב שאחראי לביצוע exploitation, באופן יחסי לאותם רכיבים בהתאמה כפי שהם עכשיו – לפני השינוי שלנו.
נשים לב שבכל רכיב $N(s)$ מופיע אחרת: ברכיב של exploration הוא מופיע במכנה ובשורש.
לעומת זאת, ברכיב של exploitation הוא מופיע במכנה אך ללא שורש.
לכן, אם נגדיל את $N(s)$: הרכיב של exploration יקטן במעט (בזכות השורש),
לעומת הרכיב של exploitation שיקטן בהרבה...

נסמן את $N(s)$ עד לפני הסעיף הזה בתור $N(s)$ הישן: $N_{old}(s)$,
ונסמן את הנוסחה החדשה שלנו בתור $N(s)$ החדש: $N_{new}(s)$.
ונציע שינוי לנוסחה באופן הבא: $N_{new}(s)=25*N_{old}(s)$
וכך נגיע למצב שבו הרכיב של exploration יקטן רק פי 5,
לעומת הרכיב של exploitation שיקטן פי 25.
וכך נשיג את מטרתנו.