עצים

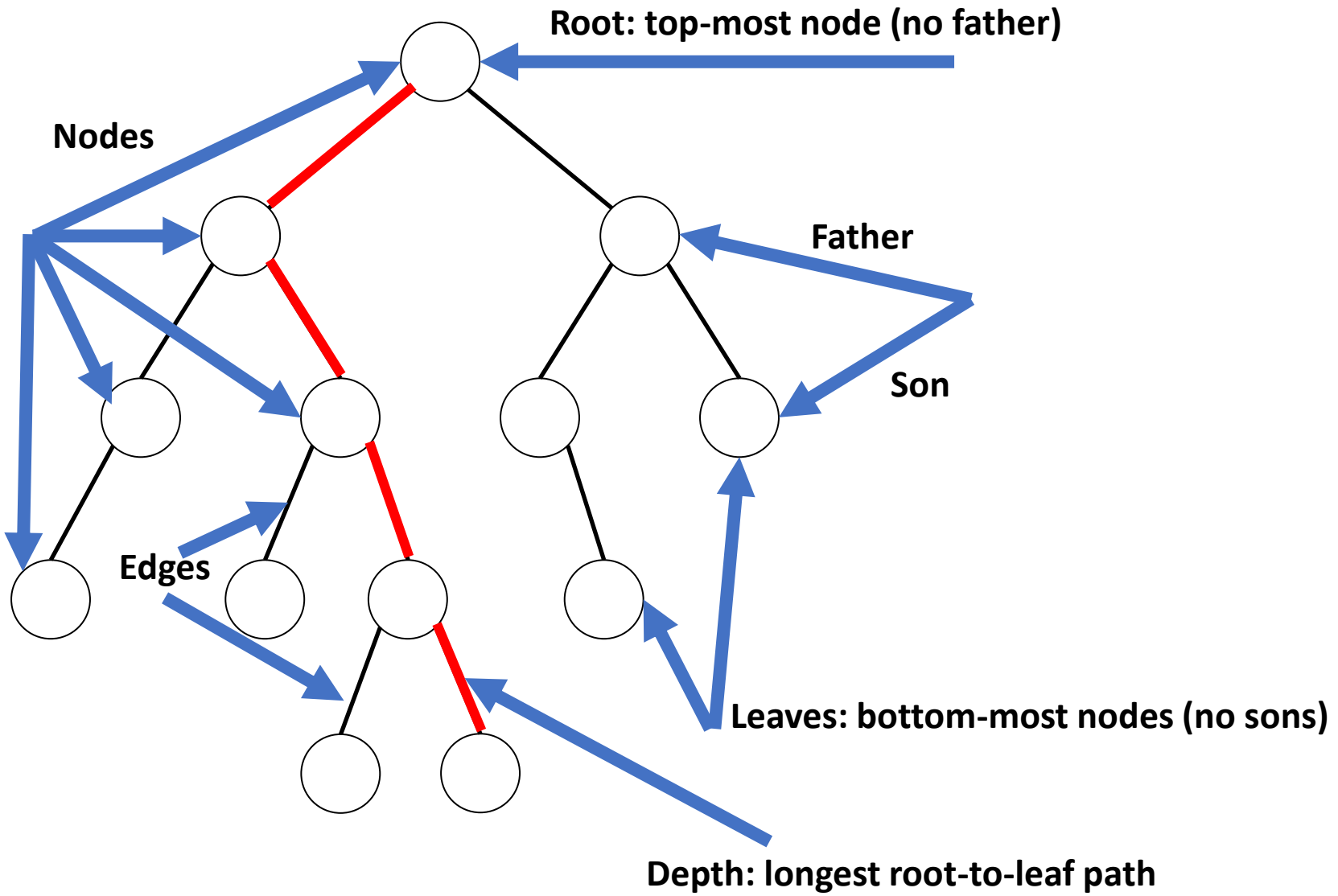
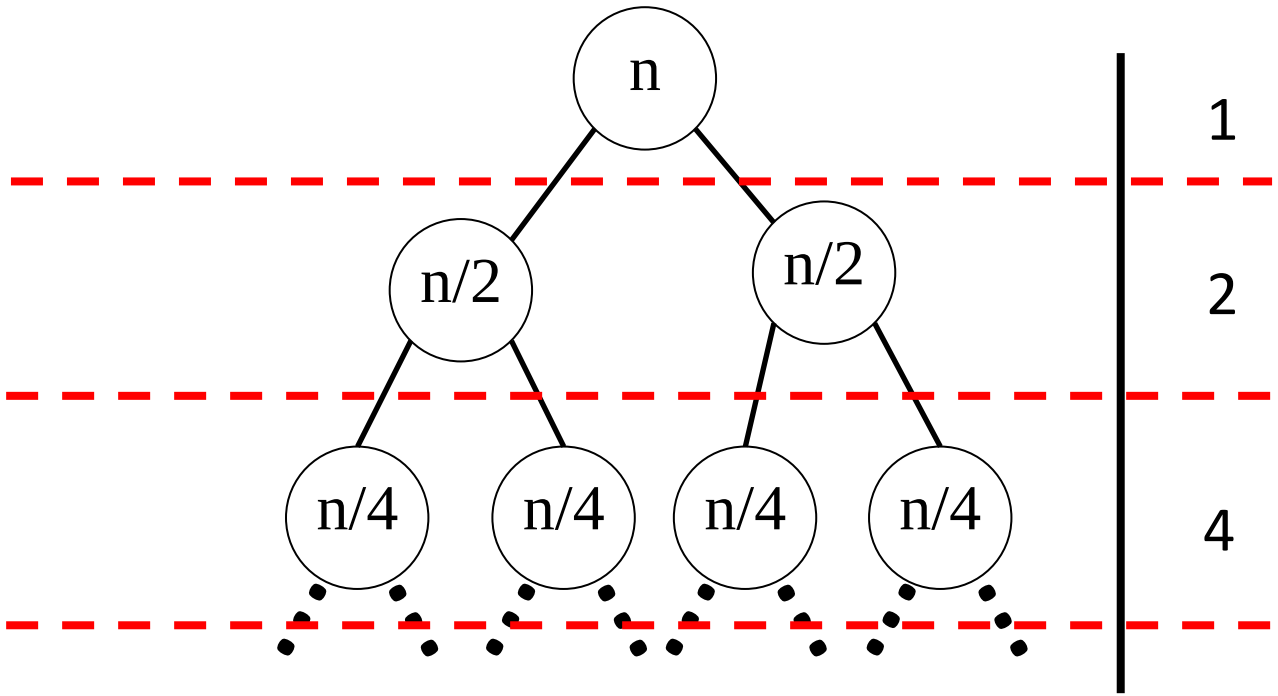


Diagram courtesy of Data Structures course (Haim Kaplan, Uri Zwick)

ניתוח של max_idx

ב-max_idx יש שורש, שני בנים, ארבעה נכדים, וכו'.. סדרה הנדסית
באורך log n: $1 + 2 + 4 + \cdots + 2^{\log n} = 2n - 1 = O(n)$



בעיית Subset sum

הקלט: רשימה L שמכילה מספרים שלמים, וערך שלם s.
הפלט: האם קיימת תת רשימה $T \subseteq L$ שסכום האיברים בה שווה בדיוק ל s? אם כן, הפלט הוא True אחרת False.

דוגמא 1:

עבור הקלט: $L = [4, -7, 12, 5, 1],\ s = 6$

נחזיר True כי $T = [-7, 12, 1]$ מקיימת שסכום איבריה הוא 6. (גם $T = [5, 1]$ מקיימת זאת)

דוגמא 2:

עבור הקלט: $L = [1, 2, 4, 8, 16],\ s = 32$

נחזיר False כי כל האיברים חיוביים וסכומם הכולל הוא 31.

בעיית Subset sum

הקלט: רשימה L שמכילה מספרים שלמים, וערך שלם s.
הפלט: האם קיימת תת רשימה $T \subseteq L$ שסכום האיברים בה שווה בדיוק ל s? אם כן, הפלט הוא True אחרת False.

<u>המבנה הרקורסיבי של הבעיה:</u>

מקרי הבסיס:
- אם הגענו ל $s = 0$ אז הצלחנו, ולכן נחזיר True.
- אם הגענו ל $s \neq 0$, אבל $L = []$ אז נכשלנו ולכן נחזיר False.

הקריאה הרקורסיבית – אם $s \neq 0$ וגם $L \neq []$:
שימו לב שבהכרח יתקיים ש:
- או שיש תת רשימה T שסכומה s שכוללת את האיבר הראשון ב L,
- או שיש תת רשימה T שסכומה s שלא כוללת את האיבר הראשון ב L,
- או ששתי האפשרויות לא מתקיימות ולכן אין תת רשימה $T \subseteq L$ שסכומה הוא s.

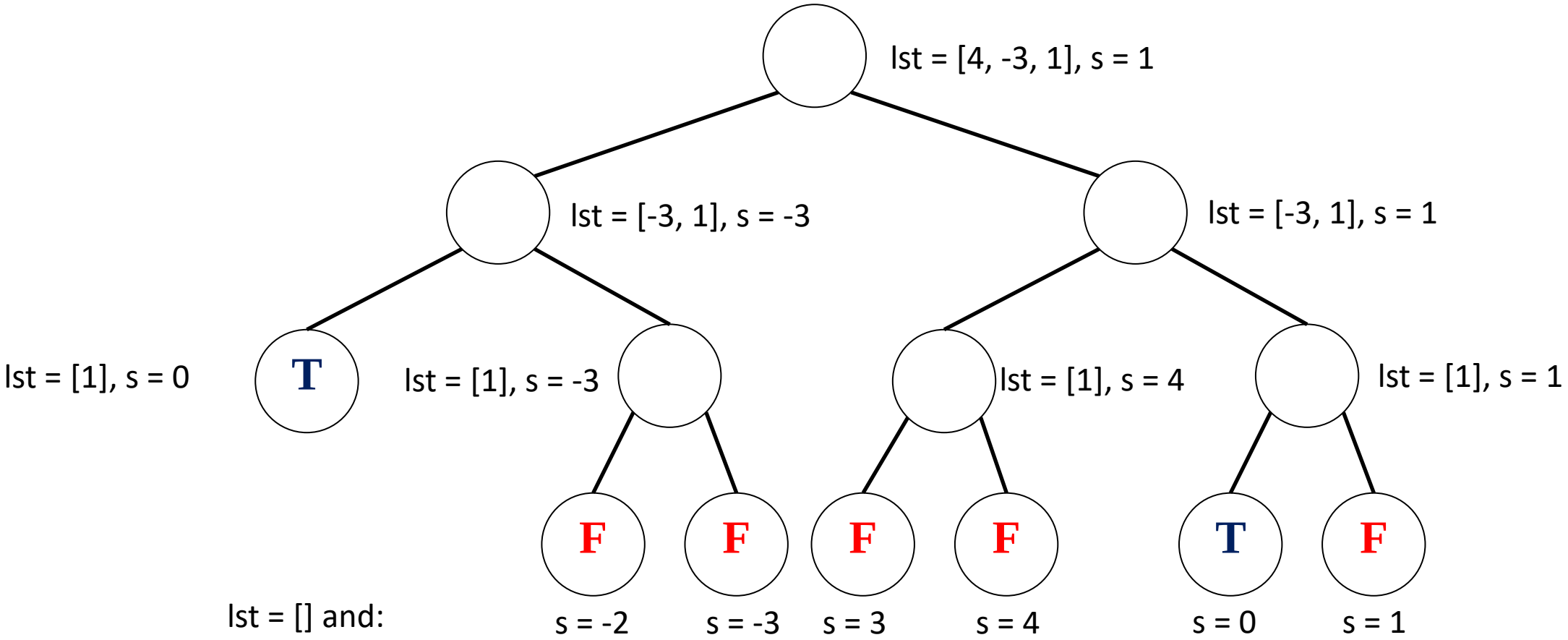
בקוד נבדוק רקורסיבית האם subset_sum(L[1:], s-L[0])==True <u>או</u> subset_sum(L[1:], s)==True.
נחזיר False אם שניהם לא מתקיימים.

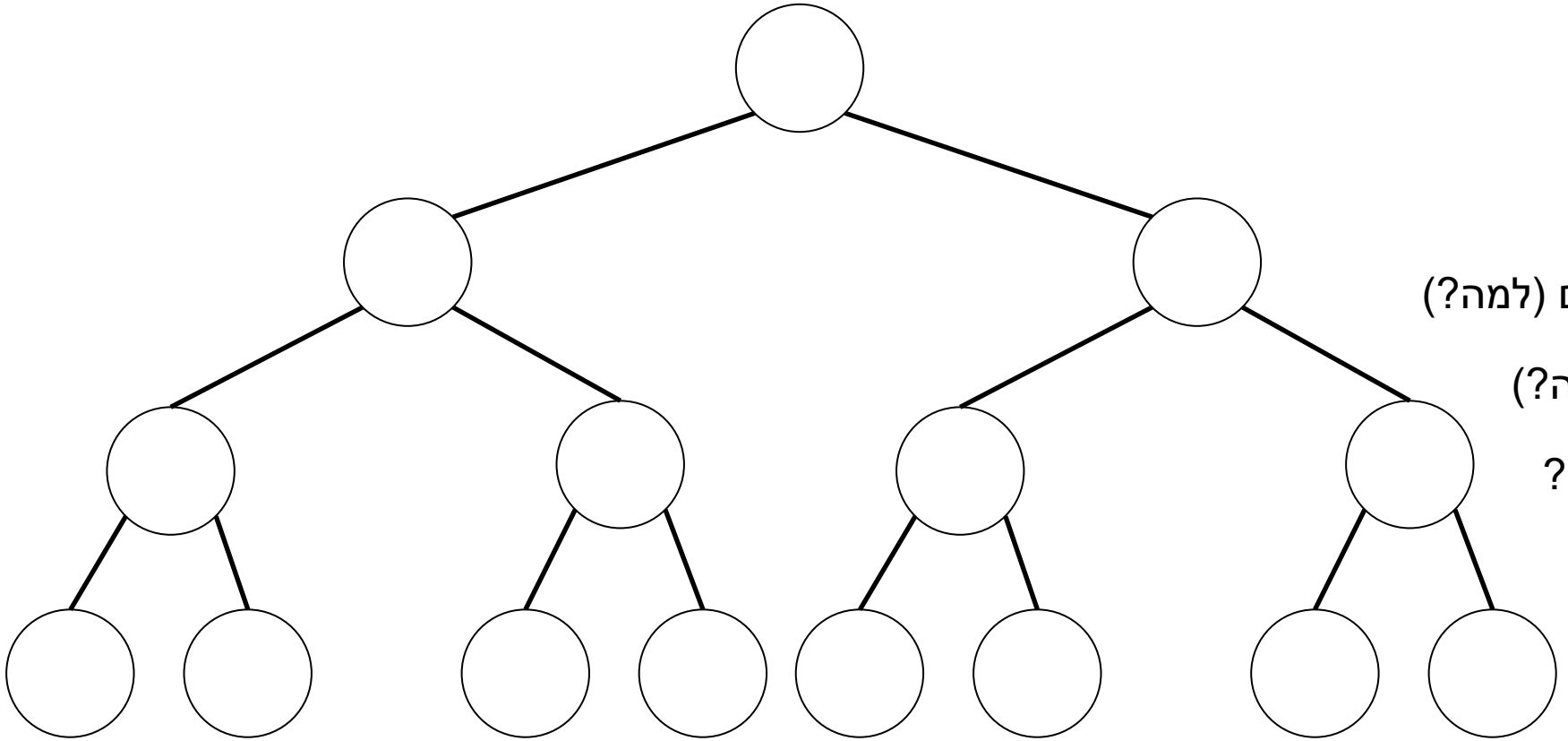
ניתוח זמן ריצה: subset_sum

נסתכל לדוגמא על הקלט [4, -3, 1] ו-$s = 1$.

ענף שמאלי: לקחת את האיבר הראשון
ענף ימני: לא לקחת את האיבר הראשון

נתעלם מחישוב סלייס, כל קריאה רצה בזמן קבוע.
נקבל חסם תחתון לזמן הריצה (למה?)





אם len(lst) = n:

עומק העץ הוא n (למה?)

ברמת הנמוכה יש 2^n עלים (למה?)

גודל העץ הוא $O(2^n)$ (למה?)

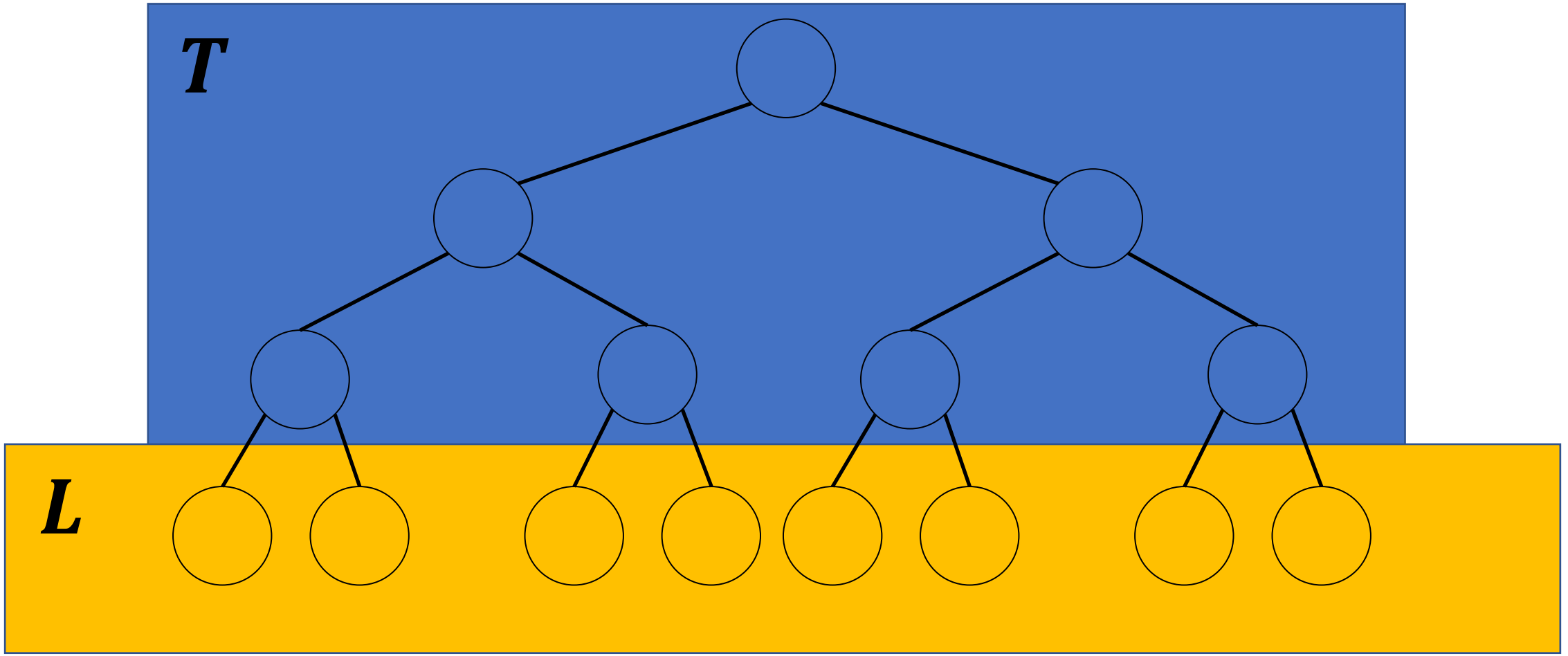
מה זה אומר על זמן הריצה?

האם אפשר לשפר את זמן הריצה כך שיהיה ריבועי ב-n? מה לגבי $O(n^{1000})$?

More on that later…

- כשהקוד יעיל, צריך להראות חסם עליון על הזמן
- לדוגמא: חיפוש בינארי, quicksort...
- כשהקוד לא יעיל, (לפעמים) מספיק להראות חסם תחתון
- לדוגמא: הפתרון ל-subset_sum רץ בזמן של לפחות 2^n
- מה זמן הריצה בפועל? האם זה "משנה"?

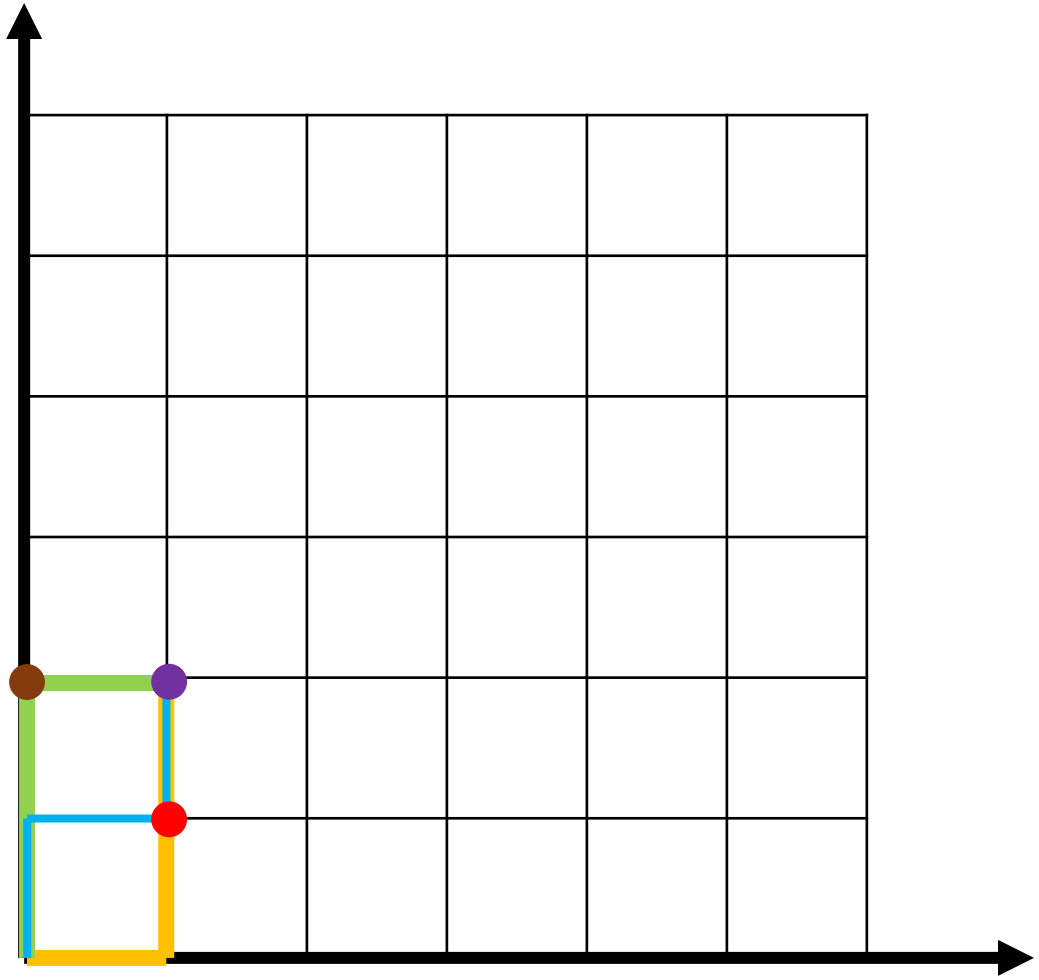
- טקטיקות לחסם תחתון:
  - אם בעץ יש T צמתים, זמן הריצה הוא לפחות?
  - אם בעץ יש L עלים, זמן הריצה הוא לפחות?
- אם P' היא תת-פרוצדורה של P אז $Time(P) \geq Time(P')$



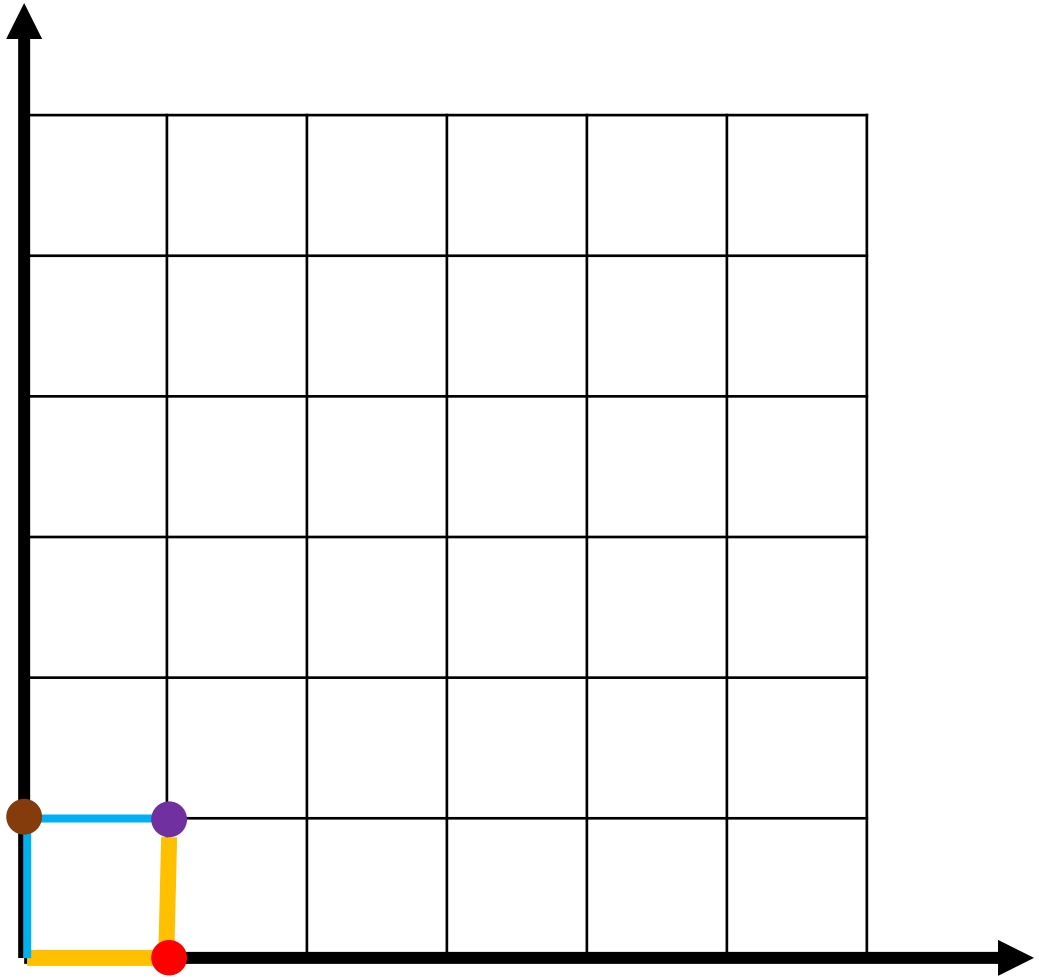
בעיית Count Paths

הקלט: רשימה L באורך d שמכילה שלמים אי-שליליים, נחשוב על הרשימה כנקודה בשריג d-מימדי
הפלט: כמה מסלולים שונים ישנם מ-L לראשית הצירים אם נעים רק על הצירים ולכיוון הראשית תמיד

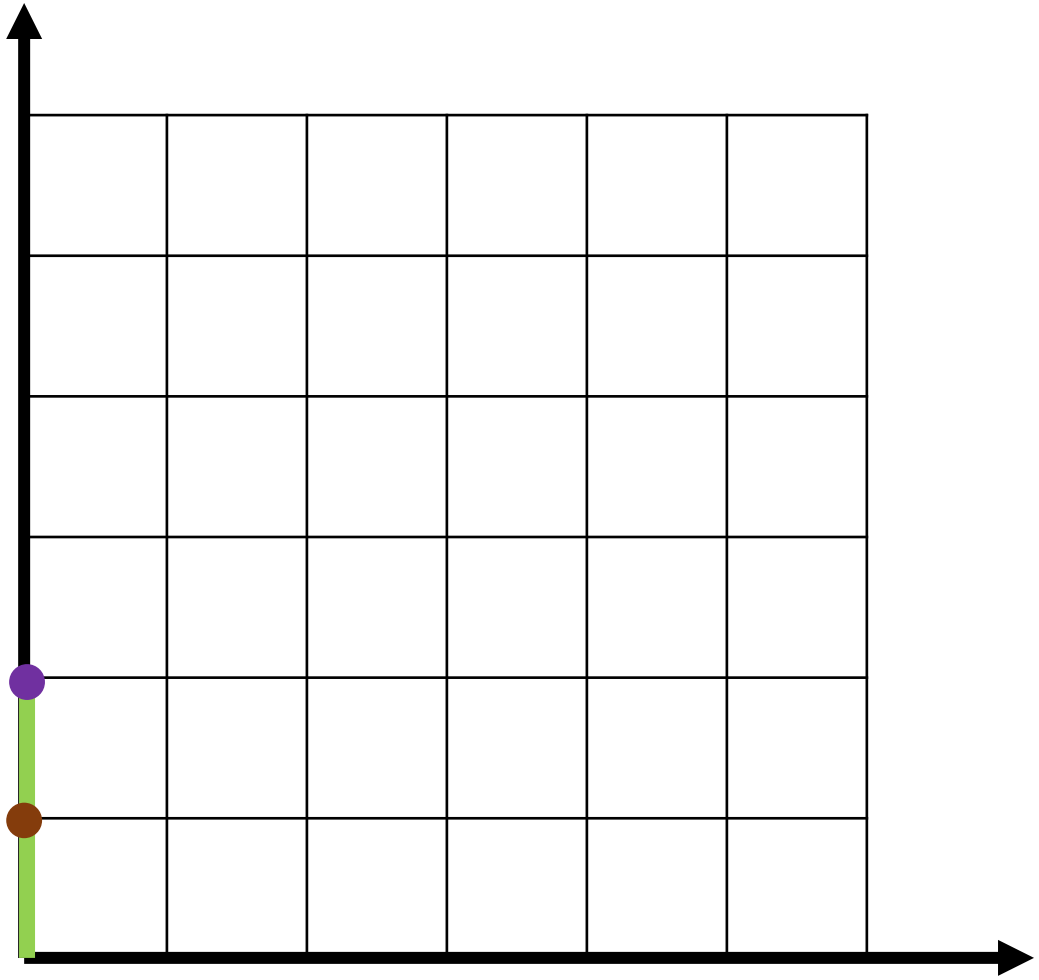
= מספר הדרכים להגיע מ-(1,2)
מספר הדרכים להגיע מ-(0,2)
+
מספר הדרכים להגיע מ-(1,1)

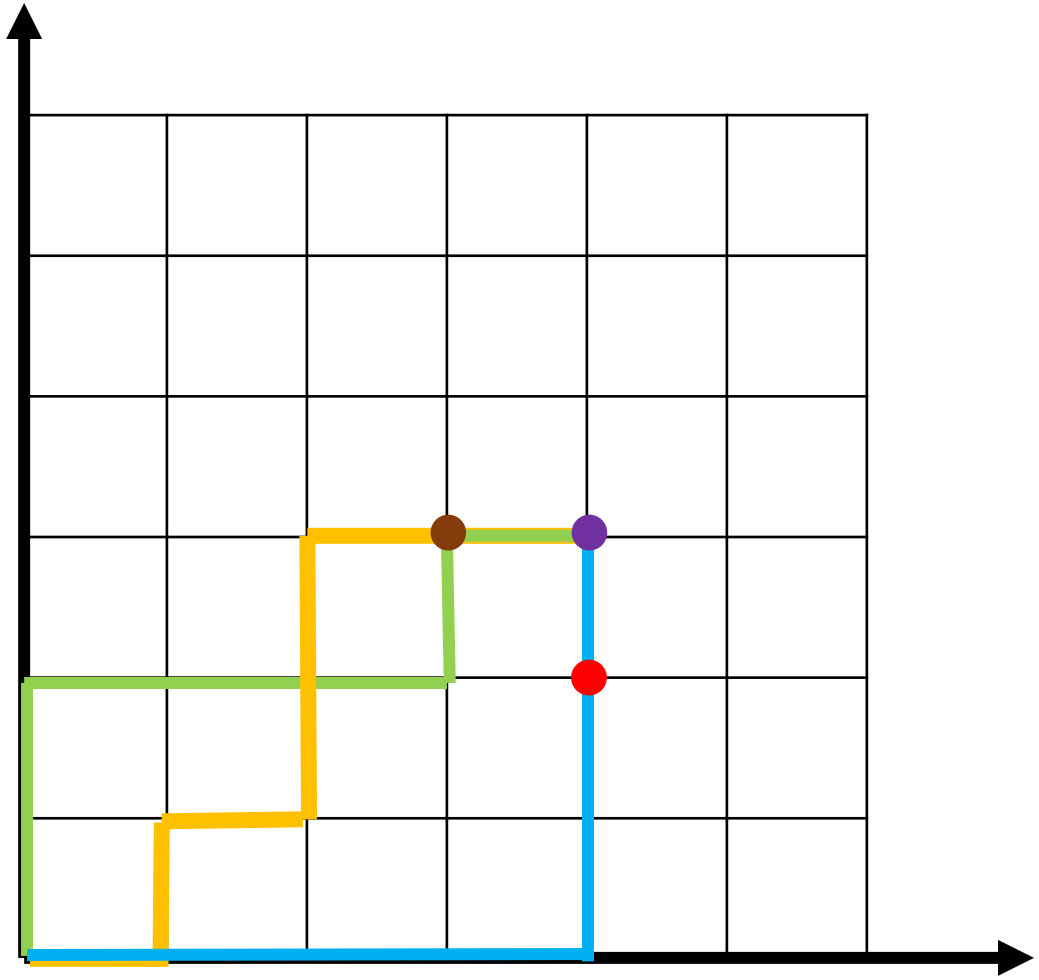


= מספר הדרכים להגיע מ-(1,1)
מספר הדרכים להגיע מ-(0,1)
+
מספר הדרכים להגיע מ-(1,0)



= מספר הדרכים להגיע מ-(0,2)
מספר הדרכים להגיע מ-(0,1)





= מספר הדרכים להגיע מ-(4,3)
מספר הדרכים להגיע מ-(3,3)
+
מספר הדרכים להגיע מ-(4,2)

הפעם ציירנו רק חלק מהמסלולים האפשריים

בעיית cnt_paths

- אם $L = [0, \dots, 0]$ יש מסלול אחד (איזה?)

- אחרת, עבור $L = [a_1, \dots, a_d]$:
  - לכל $a_i > 0$ נחשב את cnt_paths($[a_1, \dots, a_i - 1, \dots, a_d]$)

- נחזיר את הסכום על כל הקריאות

טענה: מספר הקריאות הרקורסיביות של כל צומת $\geq d$

ניתוח זמן ריצה cnt_paths

- נראה שזמן הריצה **לפחות** אקספוננציאלי בגודל הקלט.

- אסטרטגיה: חסם תחתון.
  - נתעלם מזמן הריצה של כל צומת בעץ (למה?)
  - נסתכל רק על תת-עץ (למה?)

- הקלט לדוגמא: $L = [n,n], d = 2$

- טענה – ב-n הרמות הראשונות של עץ הרקורסיה יש לכל צומת שני בנים

- הוכחה – ב-n הרמות הראשונות מתקיים תמיד $L[0], L[1] > 0$

- מסקנה – בעץ יש לפחות 2^n צמתים ← זמן ריצה לפחות 2^n

ניתוח זמן ריצה cnt_paths

- מקרה כללי (יותר): $L = [n, \dots, n], |L| = d$
- בהכללה, ב-n הרמות הראשונות יש d קריאות רקורסיביות
- מסקנה: זמן ריצה לפחות $d^n = 2^{n \log d}$
- אלטרנטיבה: אם cnt_paths(L) = m זמן ריצה $\leq m$ (למה?)
- במחברת (העשרה): הוכחה קומבינטורית $m = \exp(n,d)$
- כמה מסובך לחשב פתרון קומבינטורי?

בעיית העודף

נהגת אוטובוס צריכה להחזיר עודף בסכום מדוייק amount, ובידיה מטבעות מסוגים מוגבלים המפורטים ברשימה coins. יש לה כמות אינסופית של מטבעות מכל סוג. בהינתן amount, coins נרצה לדעת כמה דרכים יש לנהגת להחזיר את העודף?

דוגמא:

עבור הקלט: $\text{amount} = 5, coins = [1,2,3]$

יש 5 אפשרויות: 1+1+1+1+1, 1+2+2, 3+2, 1+1+3, 1+1+1+2

<u>בעיית העודף</u>

<u>פתרון רקורסיבי</u>

מקרי הבסיס:
- אם $amount = 0$ אז יש דרך אחת להחזיר עודף (בה לא מחזירים אף מטבע). לכן נחזיר 1.
- אם $amount < 0$ או $coins = []$ ו $amount \neq 0$, אז אין דרך להחזיר עודף מדוייק ולכן נחזיר 0.

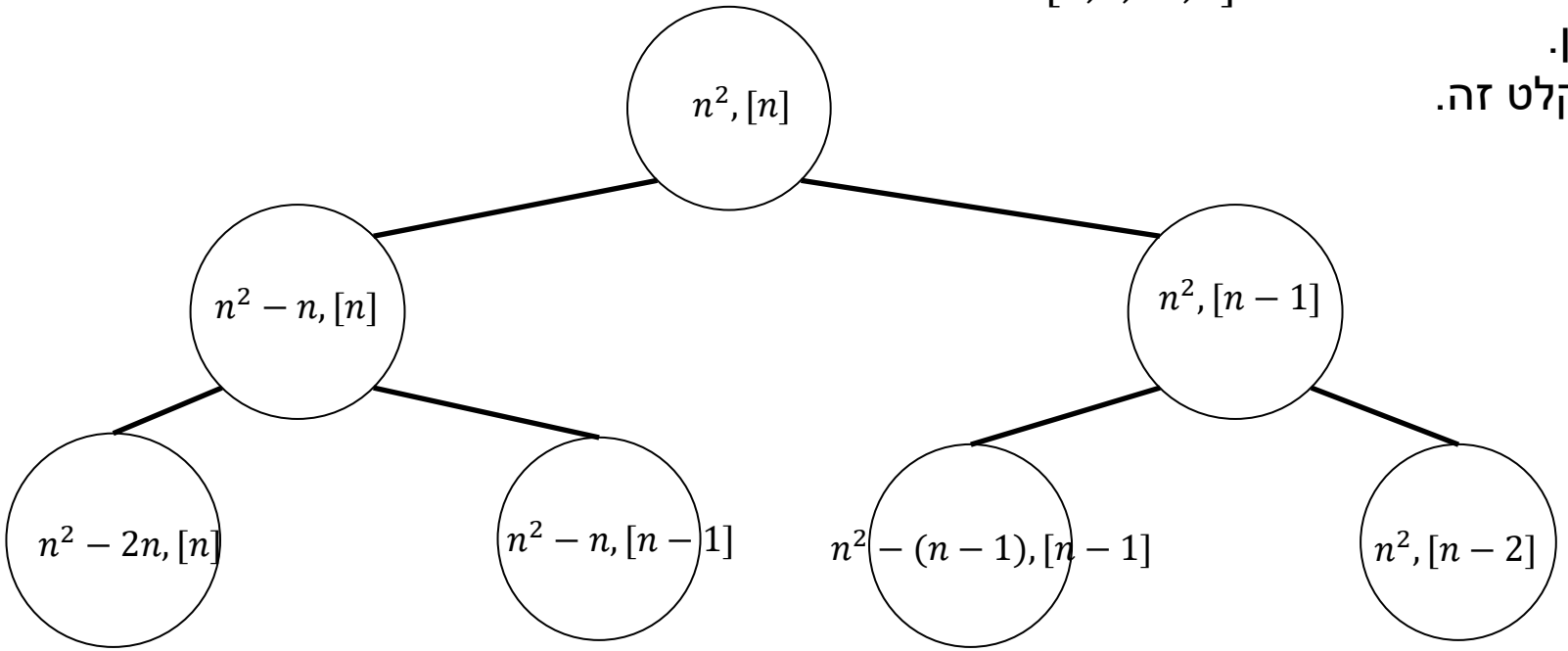
צעד הרקורסיה – אם $amount > 0$ וגם $coins \neq []$:
נבחר מטבע מהרשימה (נניח את המטבע האחרון), וכעת יש שתי אפשרויות:
- או שנשתמש במטבע האחרון ואז נבדוק כמה אפשרויות יש להחזיר את שארית הסכום.
- או שלא נשתמש במטבע האחרון כלל ואז נבדוק כמה אפשרויות יש להחזיר את הסכום (ללא שימוש במטבע האחרון).

סכום התוצאות הנ"ל ייתן את מספר האפשרויות הכולל להחזיר את העודף amount.

ניתוח זמן ריצה – בעיית העודף

נראה שזמן הריצה של הפונקציה שכתבנו גרוע מאד. כלומר, נראה שזמן הריצה הוא לפחות אקספוננציאלי בגודל הקלט.

- נסתכל על מקרה פשוט בו רשימת המטבעות היא $coins = [1,2,\dots,n]$ והסכום הוא $amount = n^2$.
  כל סכום גבוה מספיק יעבוד כאן.
  נצייר את עץ הרקורסיה עבור קלט זה.



ניתן להראות באינדוקציה שבכל צומת ברמה ה k של הרקורסיה , עבור $k < n$, מתקיים:

$$len(coins) \geq n - k > 0, \qquad amount \geq n^2 - nk > 0,$$

לכן, לכל צומת ב n הרמות הראשונות של עץ הרקורסיה יהיו בדיוק 2 בנים.

ניתוח זמן ריצה – בעיית העודף - המשך

לשם פשטות, נניח שכמות העבודה בכל צומת היא O(1). סיבוכיות הזמן היא לפחות כמות הצמתים בעץ כפול O(1) עבודה לצומת.

נראה שמספר הצמתים הוא לפחות אקספוננציאלי ב n:
ב- n הרמות הראשונות בעץ לכל צומת יש שני בנים. כמות הצמתים הכוללת ב n הרמות הראשונות היא:

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1$$

שימו לב שספרנו רק חלק מצמתי העץ, אבל לא את כולם.