טכניקות שימושיות

- פתרון בשלושה שלבים:
  - תנאי עצירה – מקרים טריוויאליים
  - פירוק בעיה גדולה לבעיות קטנות
  - חזרה מפתרון הבעיות הקטנות לפתרון הבעיה הגדולה
- ניתוח באמצעות עץ רקורסיה
- אם גודל העץ הוא T וכל צומת מבצע לכל היותר k עבודה:
  - חסם **תחתון**: זמן ריצה $\leq T$
  - חסם **עליון**: זמן ריצה $\geq T \cdot k$

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?

בינום

- נסמן ב-$\binom{n}{k}$: מספר הקבוצות בגודל k שניתן לבחור מתוך קבוצה בגודל n

- זהות פסקל: $\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$

- בקבוצה של n חתולים שכוללת חתול בשם מיצי, מספר הקבוצות בגודל k:
  - מספר קבוצות בגודל k שכוללות את מיצי: $\binom{n-1}{k-1}$
  - ועוד מספר קבוצות בגודל k **<u>שלא</u>** כוללות את מיצי: $\binom{n-1}{k}$

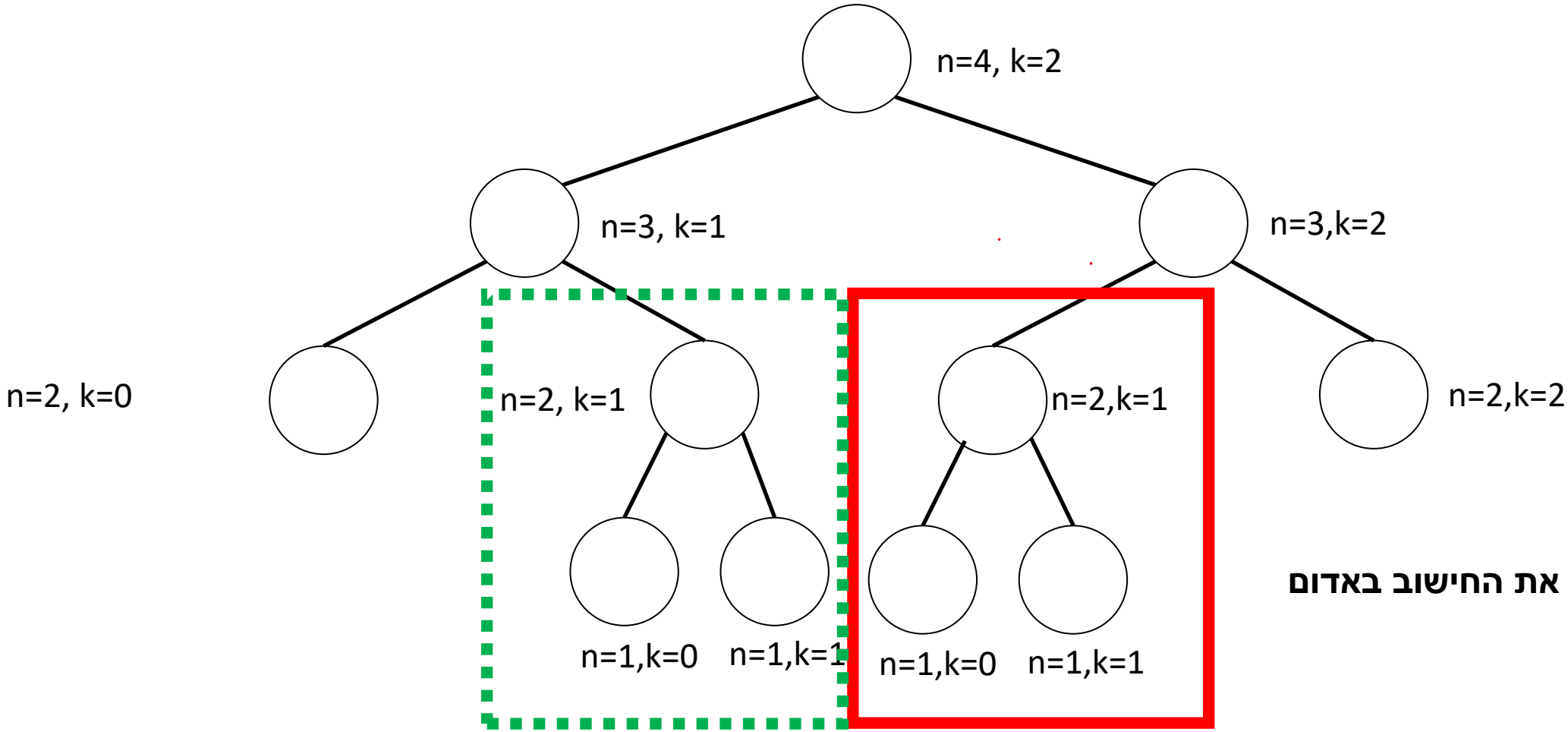
בינום – הגדרה רקורסיבית

תנאי העצירה: $\binom{n}{0} = \binom{n}{n} = 1$

רקורסיה: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

```python
def binom(n,k):
    if n < 0 or k < 0 or n < k:
        return 0
    elif (k==0 or n==k):
        return 1
    return binom(n-1,k-1) + binom(n-1,k)
```

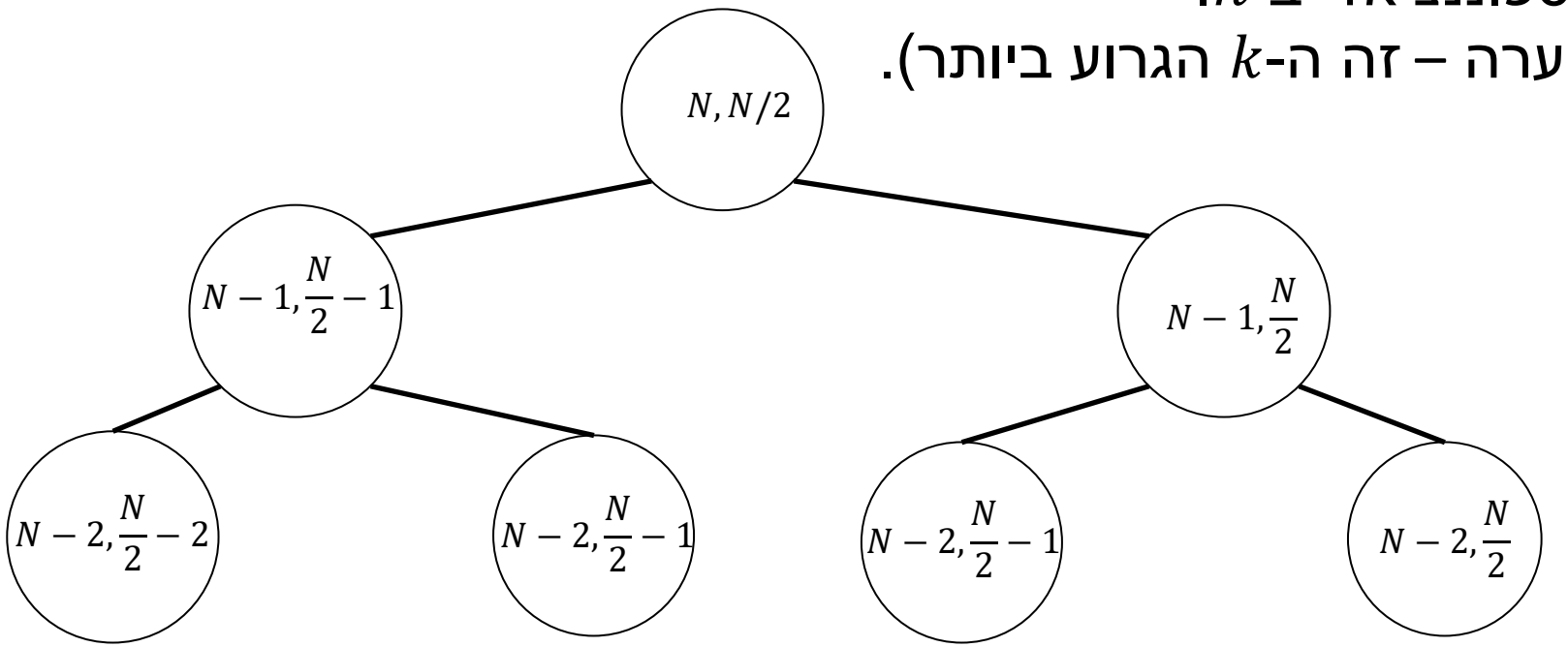
עץ הרקורסיה עבור binom(4,2)



ניתן היה לחסוך את החישוב באדום

ניתוח זמן ריצה binom

- נראה שוב חסם תחתון אקספוננציאלי ב-n.
- נקבע $n = N, k = N/2$ (הערה – זה ה-k הגרוע ביותר).
- עץ הרקורסיה:



- נבחין: לכל $j < N/2$, ברמה ה-j:

$$\frac{N}{2} \geq k_j \geq \frac{N}{2} - j > 0 \qquad n_j = N - j > \frac{N}{2},$$

- כלומר, לכל צומת ב-$N/2$ הרמות הראשונות: $n_j > k_j$ וגם $k_j > 0$ (לא תנאי עצירה!).
- מסקנה: גודל העץ לפחות $2^{n/2}$ ← זמן ריצה לפחות $2^{n/2}$.

נוסיף ממואיזציה – כדי להימנע מקריאות רקורסיביות שכבר התבצעו במהלך החישוב

- ממואיזציה: זיכרון עזר (dict, list למשל) שמכיל תוצאות חישוב הפונקציה לקלטים שונים. זהו למעשה סוג של מילון: לכל קלט נשמר הערך המתאים לו.
- המפתח מחושב מהקלט וצריך לתאר אותו היטב
- המילון ממפה מפתחות (=קלטים) לפלטים
- הערה: אם משתמשים ב-dict, פעולות על המילון רצות ב-$O(1)$ בממוצע.

- ממואיזציה לפונקציה רקורסיבית:
  1. נוסיף את זיכרון העזר לקלט הפונקציה.
  2. לפני קריאה רקורסיבית נבדוק האם הקלט במילון:
     i. אם כן, נחזיר אותו
     ii. אם לא, נחשב אותו, נשמור במילון ואז נחזיר פלט
  3. נשתמש בפונקציית מעטפת

דוגמא: binom_fast

```
def binom(n,k):
    if n < 0 or k < 0 or n < k:
        return 0
    elif (k==0 or n==k):
        return 1
     return binom(n-1,k-1) + binom(n-1,k)
```

```
def binom_fast(n,k):
    d = {}
    return binom_mem(n,k,d)

def binom_mem(n,k,d):
    if n < 0 or k < 0 or n < k:
        return 0
    elif (k==0 or n==k):
        return 1
    if (n,k) not in d:
        d[(n,k)] = binom_mem(n-1,k, d) + \
                   binom_mem(n-1,k-1, d)
    return d[(n,k)]
```

האם חייבים מילון?

יותר טבעי לתאר את הזכרון שלנו באמצעות מטריצה (=רשימה מקוננת)

דוגמא: binom_fast

```
def binom(n,k):
    if n < 0 or k < 0 or n < k:
        return 0
    elif (k==0 or n==k):
        return 1
     return binom(n-1,k-1) + binom(n-1,k)
```

```
def binom_fast(n,k):
    mat = [[-1 for i in range(k+1) ] for j in range(n+1) ]
    return binom_mem(n,k,mat)

def binom_mem(n,k,mem):
    if n < 0 or k < 0 or n < k:
        return 0
    elif (k==0 or n==k):
        return 1
    if mem[n][k] == -1:
        mem[n][k] = binom_mem(n-1,k, mem) + \
                    binom_mem(n-1,k-1, mem)
    return mem[n][k]
```

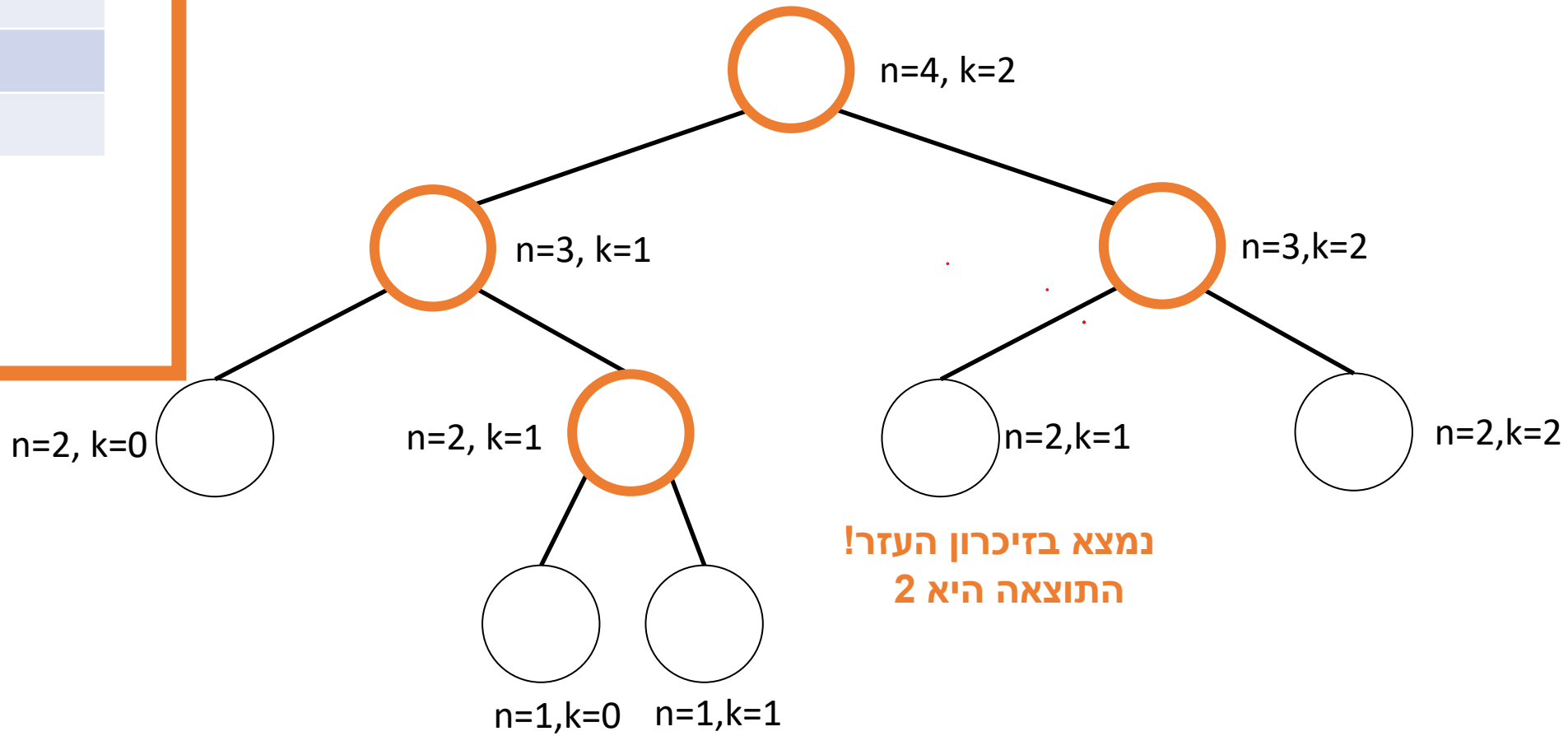
שימו לב, זיכרון העזר הינו מטריצה (ייצוג טבעי):

$$mat[n][k] = \binom{n}{k}$$

תוכן זיכרון העזר:

key	value
(n=2, k=1)	2
(n=3, k=1)	3
(n=3, k=2)	3
(n=4, k=2)	6

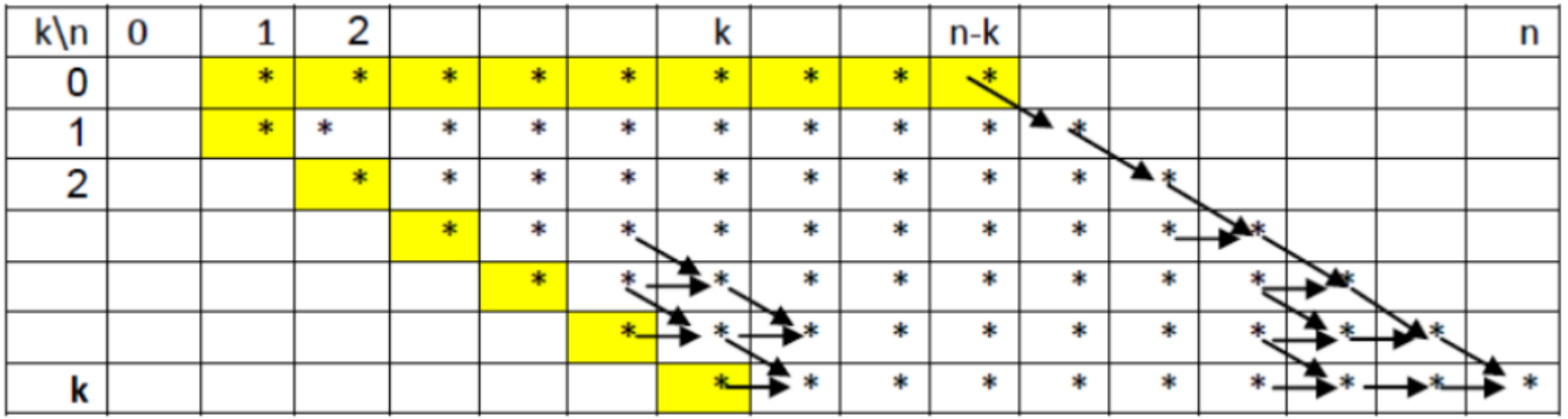
עץ הרקורסיה עבור binom_fast(4,2)



נמצא בזיכרון העזר!
התוצאה היא 2

ניתוח זמן binom_fast

- נבנה טבלה בגודל (k+1)x(n+1).
- התא בעמודה ה-i ובשורה ה-j יתאים לחישוב $\binom{i}{j}$ (מה פשר התאים הריקים?)
- נמתח חץ מתא A לתא B אם במהלך החישוב של B "קראנו" ל-A.
- צהוב: תנאי עצירה.



זמן הריצה:

Num. of visited cells X Num. of visits per cell X time per cell visit (not including recursive calls)

ניתוח זמן binom_fast

- נבחר תא i,j כלשהוא, כלומר: חישוב $\binom{i}{j}$
- התא "נקרא" לכל היותר פעמיים: מהתאים $\binom{i+1}{j}$ ו-$\binom{i+1}{j+1}$
- נניח ש-$\binom{i+1}{j}$ מחושב קודם ונבחין:
  - לא תתבצע קריאה ל $\binom{i+1}{j+1}$ לפני שיסתיים החישוב של $\binom{i+1}{j}$.
  - כשנסיים לחשב את $\binom{i+1}{j}$, המילון יכיל את $\binom{i}{j}$
  - כשנרצה לחשב את $\binom{i+1}{j+1}$, לא נצטרך לחשב את $\binom{i}{j}$ רקורסיבית.
  - ברגע שחושבו $\binom{i+1}{j+1}$ וגם $\binom{i+1}{j}$ לא ניגש אף פעם ל-$\binom{i}{j}$

- נבחר תא i,j כלשהוא, כלומר: חישוב $\binom{i}{j}$
- התא "נקרא" לכל היותר פעמיים: מהתאים $\binom{i+1}{j}$ ו-$\binom{i+1}{j+1}$
- נניח ש-$\binom{i+1}{j}$ מחושב קודם ונבחין:
  - לא תתבצע קריאה ל $\binom{i+1}{j+1}$ לפני שיסתיים החישוב של $\binom{i+1}{j}$.
  - כשנסיים לחשב את $\binom{i+1}{j}$, המילון יכיל את $\binom{i}{j}$
  - כשנרצה לחשב את $\binom{i+1}{j+1}$, לא נצטרך לחשב את $\binom{i}{j}$ רקורסיבית.
  - ברגע שחושבו $\binom{i+1}{j+1}$ וגם $\binom{i+1}{j}$ לא ניגש אף פעם ל-$\binom{i}{j}$

ניתוח זמן binom_fast

- מסקנה: לכל תא בטבלה ניגש לכל היותר פעמיים.
- זמן הריצה (כולל גישה לטבלה): קבוע (בינתיים...)
- גודל הטבלה: $O(nk)$
- מסקנה: זמן ריצה $O(nk)$