**תיעוד פרוייקט מעשי 2 בקורס מבני נתונים – מימוש ערימת פיבונאצ'י**

לירון כהן, lironcohen3, 207481268  
יובל מור, yuvalmor, 209011543

**המחלקה FibonacciHeap**

המחלקה מממשת ערימת פיבונאצ'י, ובה כל אחד מצמתי העץ מממש מופע ממחלקת HeapNode, אודותיה נפרט בהמשך.

**שדות FibonacciHeap**

* שדות סטטיים:
  + Double GOLDEN\_RATIO – קבוע סטטי אשר ערכו הוא יחס הזהב (עבור חישובים).
  + Int totalLinks – שדה סטטי השומר את כמות פעולות ה-link שבוצעו בריצת התוכנית.
  + Int totalCuts – שדה סטטי השומר את כמות פעולות ה-cut שבוצעו בריצת התוכנית.
* שדות מופע:
  + HeapNode min – מצביע לצומת עם הערך המינימלי בערימה.
  + HeapNode first – מצביע לצומת הראשון בערימה (שורש העץ הכי שמאלי)
  + Int size – מספר הצמתים בערימה כולה.
  + Int treeNum – מספר העצים בערימה.
  + Int markedNum – מספר הצמתים המסומנים בערימה.

**מתודות FibonacciHeap**

**FibonacciHeap()**

המתודה בונה ערימה ריקה, כלומר מאתחלת את המינימום של הערימה והאיבר הראשון שלה להיות null. בנוסף, מאתחלת את שדות ה-size, treeNum ו-MarkedNum להיות 0.  
סיבוכיות המתודה היא כיוון שהיא קובעת ערכים של שדה.

**isEmpty()**

המתודה מחזירה אמת אם ערך השדה size של הערימה הוא 0, ושקר אחרת.  
סיבוכיות המתודה היא כיוון שהיא בודקת ערך של שדה.

**Insert(int key)**

המתודה יוצרת צומת חדש עם המפתח הנתון ומכניסה אותו לתחילת הערימה (בתור האח הקודם לאיבר הראשון בערימה).

המתודה מעדכנת את מצביעי האחים הרלוונטיים ואת מצביע הצומת הראשון של הערימה.

בנוסף, המתודה מגדילה את ערכי השדות size ו-treeNum ב-1 ומעדכנת את ערך השדה min במידת הצורך. המתודה מחזירה את הצומת שנוצר.

סיבוכיות המתודה היא כיוון שהיא מעדכנת שדות ומבצעת בדיקת תנאי פשוטה.

**InsertTree(HeapNode x) – מתודת עזר**

המתודה מכניסה את הצומת שהתקבל לסוף הערימה (בתור האח הבא לאיבר האחרון בערימה).

המתודה מעדכנת את מצביעי האחים הרלוונטיים.

בנוסף, המתודה מעדכנת את ערך השדה size, מגדילה את ערך השדה treeNum ב-1 ומעדכנת את ערך השדה min במידת הצורך. המתודה מחזירה את הצומת.

סיבוכיות המתודה היא כיוון שהיא מעדכנת שדות ומבצעת בדיקת תנאי פשוטה.

**Link(HeapNode x, HeapNode y) – מתודת עזר**

המתודה מקבלת שני צמתים (שהם שורשים של תתי עצים מאותו הסדר) ומקשרת ביניהם באמצעות השוואה בין המפתחות שלהם, כפי שנלמד בכיתה. המתודה מעדכנת את המצביעים הרלוונטיים ואת השדות treeNum ו-totalLinks.

המתודה מחזירה את הצומת העליון מבין השניים (כלומר בעל המפתח הנמוך יותר).

סיבוכיות המתודה היא כיוון שהיא בודקת תנאים ומעדכנת שדות.

**ClearHeap() – מתודת עזר**

המתודה מנקה את ערכי השדות של הערימה (לצורך הכנסה ממתודת fromBuckets אודותיה יפורט בהמשך). סיבוכיות המתודה היא כיוון שהיא מעדכנת שדות.

**ToBuckets() – מתודת עזר**

המתודה מכניסה את עצי הערימה למערך עצים (המיוצגים מחשבתית ע"י הדליים שהוצגו בכיתה), ומאחדת באמצעות מתודת העזר link את תתי העצים בעלי אותה דרגה.

המתודה מחזירה את מערך העצים כך שבכל דרגה יש לכל היותר עץ יחיד.

הראינו בהרצאה שסיבוכיות המתודה היא WC וכן Amortized.

**FromBuckets(HeapNode[] B) – מתודת עזר**

המתודה מקבלת מערך עצים אשר נוצרו ע"י מתודת ToBuckets.

המתודה מנקה את שדות הערימה באמצעות מתודת העזר clearHeap, עוברת על המערך ומכניסה את העצים לערימה באמצעות מתודת העזר insertTree.

סיבוכיות המתודה היא כיוון שהיא עוברת על מערך בגודל וקוראת למתודות עזר הפועלות בסיבוכיות .

**Consolidate() – מתודת עזר**

המתודה קוראת למתודות העזר toBuckets ו-FromBuckets אחת אחרי השנייה.

סיבוכיות המתודה היא WC וכן Amortized.

**removeMin() – מתודת עזר**

המתודה מוחקת את הצומת המינימלי בעץ ויוצרת ערימה חדשה ובה תתי העצים של הצומת שנמחק.

המתודה "מדלגת" (באמצעות שינוי מצביעים) על הצומת המינימלי בתוך רשימת האחים שלו.

המתודה מוחקת את הצומת המינימלי ושמה צומת מינימלי אחר במקומו (יעודכן לצומת הנכון לאחר קריאה ל-updateMin).

בנוסף, המתודה יוצרת ערימה חדשה ומכניסה אליה את תתי העצים של צומת המינימום. מעדכנת את ערך השדה של הערימה החדשה לצומת הראשון (יעודכן גם כן לאחר קריאה ל-updateMin) ואת גודל הערימה וכמות העצים בערימה.

המתודה מחזירה את הערימה החדשה שנוצרה.

סיבוכיות המתודה היא מכיוון שהמתודה מבצעת שינויי מצביעים, בדיקות תנאי פשוטות, יצירת ערימה חדשה שגם היא נעשית בזמן קבוע וחישובים פשוטים.

**updateMin() – מתודת עזר**

המתודה עוברת על שורשי הערימה ומעדכנת את ערך השדה min של הערימה לערך המינימלי מבין השורשים.

נציין שעקב תכונות הערימה הערך המינימלי יהיה חייב להימצא באחד השורשים.

סיבוכיות המתודה היא מכיוון שהמתודה מתבצעת לאחר קריאה למתודת העזר consolidate אשר מבטיחה קיום של לכל היותר עץ יחיד מכל דרגה, ובסה"כ עבור איברים בערימה נקבל לכל היותר דרגות.

**deleteMin()**

המתודה קוראת למתודות העזר removeMin, meld, consolidate, updateMin בזו אחר זו.

סיבוכיות המתודה היא WC וכן Amortized, כיוון שזוהי הסיבוכיות הגרועה ביותר של מתודות העזר שנקראות.

**findMin()**

המתודה מחזירה את ערך השדה min של הערימה.   
סיבוכיות המתודה היא כיוון שהיא מחזירה ערך של שדה.

**meld()**

המתודה מקבלת ערימת פיבונאצ'י ומאחדת את הערימה הנוכחית איתה.

המתודה מעדכנת את השדות min, size, markedNum ו-treeNum של הערימה.

בנוסף, המתודה מעדכנת את המצביעים הרלוונטיים.

סיבוכיות המתודה היא כיוון שהיא מעדכנת ערכי שדות.

**size()**

המתודה מחזירה את ערך השדה size של הערימה.   
סיבוכיות המתודה היא כיוון שהיא מחזירה ערך של שדה.

**countersRep()**

המתודה עוברת על שורשי הערימה ומעדכנת מערך מונים בהתאם לדרגות העצים בערימה.

סיבוכיות המתודה במקרה הגרוע היא מכיוון שאם יתבצעו פעולות insert ללא deleteMin, נקבל n שורשים בודדים (n עצים מדרגה 0) ונצטרך לעבור על כולם ולעדכן את המערך.

**delete(HeapNode x)**

*המתודה מוחקת את הצומת מהעץ ע"י קריאה למתודה* decreaseKey *(עבור* delta=*) ולאחר מכן קריאה ל-*deleteMin.

סיבוכיות המתודה היא WC וכן Amortized, כיוון שזוהי הסיבוכיות הגרועה ביותר של מתודות העזר שנקראות.

**Cut(HeapNode x, HeapNode y) – *מתודת עזר***

*המתודה מקבלת צומת* x  *ואת ההורה שלה* y*, וחותכת את הקשת ביניהן כפי שראינו בכיתה.*

*המתודה מעדכנת את המצביעים הרלוונטיים, מכניסה את תת העץ ששורשו* x *לערימה ומעדכנת את השדות* markedNum *ו-*totalCuts *בהתאם.*

*סיבוכיות המתודה, כפי שראינו בכיתה היא* O(1)*.*

**cascadingCut(HeapNode x, HeapNode y) *– מתודת עזר***

*המתודה חותכת את* x *מההורה שלו,* y. *אם צריך, פועלת באופן רקורסיבי עד אשר ההורה אינו מסומן.*

*המתודה קוראת למתודת העזר* cut*, ואם ההורה אינו* null *ואינו מסומן, מעדכנת אותו להיות מסומן ומסיימת.*

*אם ההורה מסומן, פועלת רקורסיבית על ההורה וההורה שלו.*

*סיבוכיות המתודה* WC *היא מכיוון שבמקרה הגרוע צריך לבצע כמות חיתוכים כעומק העץ, כלומר כאשר כל חיתוך נעשה בסיבוכיות .*

**decreaseKey(HeapNode x, int delta)**

*המתודה מקבלת צומת וערך* delta*, מקטינה את מפתח הצומת ב-*delta *ומבצעת חיתוכים במידת הצורך באמצעות מתודת העזר* cascadingCut.

*סיבוכיות המתודה במקרה הגרוע היא וכפי שראינו בכיתה, סיבוכיות ה- היא .*

**potential()**

*המתודה מחזירה את פוטנציאל הערימה. כלומר את סכום ערך השדה* treeNum *ופעמיים ערך השדה* markedNum.

*סיבוכיות המתודה היא מכיוון שהיא מחזירה ערכים של שדות ומבצעת פעולה חשבונית פשוטה.*

**totalLinks()**

*המתודה מחזירה את מספר פעולות ה- שבוצעו מתחילת התוכנית.*

*המתודה מחזירה את ערך השדה הסטטי ולכן סיבוכיותה .*

**totalCuts()**

*המתודה מחזירה את מספר פעולות ה- שבוצעו מתחילת התוכנית.*

*המתודה מחזירה את ערך השדה הסטטי ולכן סיבוכיותה .*

**kMin(FibonacciHeap H, int k)**

**המחלקה HeapNode**

המחלקה מממשת צומת בערימת פיבונאצ'י לפי הנלמד בכיתה.

**שדות HeapNode**

* Int key – המפתח של הצומת
* Int rank – דרגת הצומת (מספר הילדים של הצומת)
* HeapNode firstChild – מצביע לבן השמאלי ביותר של הצומת
* HeapNode prevBro – מצביע לאח השמאלי של הצומת
* HeapNode nextBro – מצביע לאח הימני של הצומת
* HeapNode parent – מצביע להורה של הצומת

**מתודות HeapNode**

נציין שכל מתודות המחלקה מבצעות בדיקות פשוטות, אחזורים והשמות לשדות ולכן מתבצעות ב-.

**HeapNode(int key)**

המתודה בונה עצם מסוג HeapNode ומאתחלת את המפתח שלו למפתח שהתקבל כארגומנט.

בנוסף, המתודה מגדירה את מצביעי האחים של הצומת כצומת עצמו.

**getKey()**

המתודה מחזירה את המפתח של הצומת.

**setKey(int key) – מתודת עזר**

המתודה מגדירה את שדה המפתח של הצומת כערך שהתקבל כארגומנט.

**getRank() – מתודת עזר**

המתודה מחזירה את הדרגה של הצומת (מספר הילדים שלו).

**setRank(int rank) – מתודת עזר**

המתודה מגדירה את שדה הדרגה של הצומת כערך שהתקבל כארגומנט.

**getMark() – מתודת עזר**

המתודה מחזירה את שדה הסימון של הצומת (ביט שערכו 1 אם הוא מסומן ו-0 אחרת).

**setMark(int mark) – מתודת עזר**

המתודה מגדירה את שדה הסימון של הצומת כערך שהתקבל כארגומנט.

**getFirstChild() – מתודת עזר**

המתודה מחזירה מצביע לבן השמאלי ביותר של הצומת.

**setFirstChild(HeapNode child) – מתודת עזר**

המתודה מגדירה את הבן השמאלי ביותר של הצומת כערך שהתקבל כארגומנט.

**getPrevBro() – מתודת עזר**

המתודה מחזירה מצביע לאח השמאלי של הצומת (באופן מעגלי).

**setPrevBro(HeapNode prevBro) – מתודת עזר**

המתודה מגדירה את האח השמאלי של הצומת כערך שהתקבל כארגומנט.

**getNextBro() – מתודת עזר**

המתודה מחזירה מצביע לאח הימני של הצומת (באופן מעגלי).

**setNextBro(HeapNode nextBro) – מתודת עזר**

המתודה מגדירה את האח הימני של הצומת כערך שהתקבל כארגומנט.

**getParent() – מתודת עזר**

המתודה מחזירה מצביע להורה של הצומת.

**setParent(HeapNode parent) – מתודת עזר**

המתודה מגדירה את ההורה של הצומת כערך שהתקבל כארגומנט.

**חלק המדידות**

**שאלה 1 סעיף א'**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| מספר סידורי | גודל המערך | כמות החילופים במיון רגיל עבור מערך אקראי | כמות החילופים במיון רגיל עבור מערך ממוין הפוך | עלות החיפושים במיון AVL עבור מערך אקראי | עלות החיפושים במיון AVL עבור מערך ממוין הפוך |
| 1 | 10,000 | 25,063,646 | 49,995,000 | 221,782 | 231,358 |
| 2 | 20,000 | 100,023,653 | 199,990,000 | 480,520 | 502,688 |
| 3 | 30,000 | 224,839,071 | 449,985,000 | 777,310 | 788,113 |
| 4 | 40,000 | 401,139,380 | 799,980,000 | 1,056,505 | 1,085,346 |
| 5 | 50,000 | 625,021,906 | 1,249,975,000 | 1,334,497 | 1,386,195 |
| 6 | 60,000 | 900,795,981 | 1,799,970,000 | 1,614,641 | 1,696,195 |
| 7 | 70,000 | 1,225,062,394 | 2,449,965,000 | 1,986,235 | 2,010,660 |
| 8 | 80,000 | 1,599,255,957 | 3,199,960,000 | 2,233,915 | 2,330,660 |
| 9 | 90,000 | 2,032,584,245 | 4,049,955,000 | 2,522,302 | 2,650,660 |
| 10 | 100,000 | 2,504,360,436 | 4,999,950,000 | 2,885,898 | 2,972,357 |

ציפיות ומסקנות מהטבלה

מיון רגיל - השוואה של מערך ממוין הפוך ומערך אקראי

מבחינת קלטי המערכים האפשריים לאלגוריתם המיון, מערך ממוין בסדר הפוך הוא הגרוע יותר מבחינת כמות הפעולות. מערך זה הכי רחוק ממערך ממוין בסדר עולה ולכן נצפה לכמות חילופים גדולה מאוד ביחס למערך אקראי, אשר יכול להגריל סדר ערכים (באמצעות פעולת shuffle) היוצר מערך הקרוב יותר למערך הממוין הדרוש ולכן ידרוש מספר נמוך יותר של חילופים.

מיון AVL – השוואה של מערך ממוין הפוך ומערך אקראי

ראשית נדגיש שעקב אופן בניית המערכים, האיברים בעץ ה-AVL לאחר כל סבב הכנסות יהיו זהים (כלומר איברים מ-0 ועד גודל המערך פחות 1, אשר הוכנסו לעץ בסדר שונה). עקב העובדה שעץ AVL הוא עץ חיפוש, אלגוריתם ההכנסה שלו מסדר את העץ כך שישמור על כללי עץ חיפוש וכן מאזן את העץ. לכן, נצפה שלא יהיה הבדל משמעותי בין מערכים שונים שהוכנסו למיון העץ. נציין שעקב סדר ההכנסה במערך ממוין הפוך, ועקב העובדה שככל שמתקדמים במערך האיברים קטנים יותר וגם העץ גדל, נוכל לשער שעלות החיפושים תגדל מעט יותר בסדר הכנסה זה.

השוואה של כמות החילופים במיון רגיל אל מול עלות החיפושים במיון AVL

מכיוון שעץ AVL הוא עץ מאוזן, עלות חיפוש תהיה ב- (סדר גודל של גובה העץ), לעומת מיון רגיל שבכל איטרציה עובר על סדר גודל של כל המערך כדי לבצע את ההשוואות, כלומר . לכן, נצפה שעלות החיפושים במיון AVL תהיה קטנה משמעותית מכמות החילופים במיון רגיל.

כמות החילופים כתלות בגודל המערך

מכיוון שכמות החילופים במיון רגיל תלויה בגודל המערך וכן גם עלות החיפושים במיון AVL, נצפה שככל שנגדיל את גודל המערך כך יתבצעו יותר חילופים ועלות החיפושים תעלה.

ניתן לראות שהתוצאות שהתקבלו תואמות את הציפיות.

ניתוח כמות החילופים ב-Insertion Sort עבור מערך אקראי

מכיוון שמדובר במערך אקראי, נחשב את תוחלת כמות החילופים הממוצעת, על פני כל המערכים האפשריים. נסמן את המערך ב-.

עבור כל איטרציה של הלולאה באלגוריתם המיון, נגדיר עבור   
אינדיקטור .

נשים לב שעבור מתקיים:

נקבל:

*מסכום סדרה חשבונית נקבל:*

*ובסך הכל נקבל:*

*מהגדרת אלגוריתם המיון, מכיוון שמבצעים איטרציה על כל איברי המערך ומבצעים חילופים סך הכל, סיבוכיות האלגוריתם תהיה:*

ניתוח כמות החילופים ב-Insertion Sort עבור מערך ממוין הפוך

נשים לב שמדובר בקלט ספציפי ולכן נחשב באופן ישיר כמה חילופים מתבצעים עבורו.

נניח שהמערך בגודל .

* *עבור האיבר הראשון במערך, נבצע 0 חילופים.*
* *עבור האיבר השני במערך נשווה אותו לראשון, הם יהיו בסדר הפוך ולכן נבצע חילוף אחד.*
* *עבור האיבר השלישי הקטן מהשניים שלפניו, נבצע שני חילופים.*
* *באופן כללי, עבור האיבר ה- נבצע חילופים.*

*נסכום את מספר החילופים ונקבל סדרה חשבונית:*

*ולכן מספר החילופים עבור מערך זה הינו חילופים, כלומר חילופים.*

ניתוח עלות החיפושים ב-AVL insertion sort עבור מערך אקראי

ראשית ננתח את עלות החיפושים כתלות ב- וב- (כמות החילופים שהייתה מתבצעת אם המיון היה רגיל).

נסמן בתור את כמות החילופים שנעשו לאיבר במקום ה- במערך, כך שמתקיים:

*עבור חיפוש של איבר במערך נעבור על של מספר האיברים הנמצאים לפני האיבר במערך וגדולים ממנו (כלומר הוכנסו לפניו לעץ ונעבור דרכם בחיפוש), כלומר קשתות. נסמן את כמות הקשתות הכוללת בתור , ונקבל:*

*כאשר אי השיוויון האחרון נובע ממניפולציה אלגברית על אי שיוויון הממוצעים והשיוויון האחרון נובע מחוקי לוגריתמים.*

*לבסוף נזכור שיש לבצע סריקת של העץ ולכן נקבל .*

*קיבלנו חסם כללי על עלות החיפושים הכוללת ולכן זהו גם חסם על התוחלת המבוקשת.*

*בבירור (מכיוון שזוהי הכמות המקסימלית של ההחלפות עבור איבר במערך), ולכן:*

*ומכאן נקבל .*

ניתוח עלות החיפושים ב-AVL insertion sort עבור מערך ממוין הפוך

נשים לב שמדובר בקלט ספציפי ולכן נחשב באופן ישיר כמה חילופים מתבצעים עבורו.

ראינו בהרצאה שחיפוש איבר עם מפתח בעץ נעשה בסיבוכיות של , ובמקרה הגרוע נקבל קשתות לכל חיפוש.

מחוקי לוגריתמים נקבל:

**שאלה 1 סעיף ב'**

כפי שהראינו בסעיף הקודם, החסם העליון על כמות החיפושים הוא . מלבד עלות החיפושים, עבור חסם על הסיבוכיות נתחשב גם בפעולות איזון שמתבצעות על עץ ה-.

במקרה הגרוע (עבור חסם עליון), נקבל שכל פעולת איזון חסומה ע"י (שמתקבלת עבור שרשרת ובמימוש שלנו מתקבלת תמיד עקב פעולת ). מתבצעת פעולת איזון עבור כל הכנסה, כלומר פעולות איזון ולכן נקבל:

**שאלה 2 סעיף א'**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי | עלות join ממוצע עבור split אקראי | עלות join מקסימלי עבור split אקראי | עלות join ממוצע עבור split של איבר מקסימלי בתת העץ השמאלי | עלות join מקסימלי עבור split של איבר מקסימלי בתת העץ השמאלי |
| 1 | 2.833 | 6 | 2.769 | 16 |
| 2 | 2.769 | 7 | 3.083 | 17 |
| 3 | 2.750 | 10 | 2.929 | 18 |
| 4 | 2.286 | 6 | 2.923 | 18 |
| 5 | 2.700 | 6 | 2.846 | 18 |
| 6 | 2.800 | 5 | 2.667 | 19 |
| 7 | 2.714 | 5 | 2.688 | 19 |
| 8 | 2.733 | 7 | 2.600 | 19 |
| 9 | 3.067 | 7 | 2.867 | 20 |
| 10 | 3.200 | 6 | 3.000 | 20 |

ניתוח עלות ממוצעת ל-join עבור split של האיבר המקסימלי בתת העץ השמאלי

נשים לב שאם ישנם צמתים בעץ, גובה העץ יהיה כ-. עלות ממוצע תהיה:

*כאשר מבצעים על האיבר המקסימלי בתת העץ השמאלי של העץ, כמות פעולות ה- שנבצע תהיה כגובה העץ כולו, שהוא . בנוסף, ראינו בהרצאה שמחיר פעולת היא כסכום מחירי פעולות ה- המתבצעות בתוך פעולת ה- ומכאן נקבל את העלות המבוקשת.*

ניתוח עלות ממוצעת ל-join עבור split אקראי

כפי שראינו, במקרה הגרוע (האיבר המקסימלי בתת העץ השמאלי) העלות הממוצעת תהיה ובפרט חסם קבוע זה הוא גם חסם גם על התוחלת עבור מקרה אקראי.

ניתוח עלות מקסימלית ל-join עבור split אקראי

בהנחה שהעץ מאוזן, ראינו בתרגול שכמות הצמתים ברמה שבגובה היא (כלומר ברמת העלים יש מהצמתים, ברמה שמעליה יש מהצמתים וכו').

עבור לפי צומת ברמה ה-, ה- המקסימלי יעלה כמספר המקסימלי של הקשתות שעולות "באותו הכיוון" (כלומר שרשרת של בנים ימניים או שרשרת של בנים שמאליים).   
עלות פעולת ה- המקסימלי תהיה כגובה העץ פחות גובה הרמה, כלומר .

נחשב לפי משפט התוחלת השלמה את תוחלת כמות פעולות ה- בחלוקה לפי רמות בהן ה- מבוצע:

ניתוח עלות מקסימלית ל-join עבור split של האיבר המקסימלי בתת העץ השמאלי

עקב המיקום הייחודי של האיבר המקסימלי בתת העץ השמאלי, נשים לב שה- המקסימלי יתקבל   
ב- האחרון.

בהתחלה יבוצעו (גובה העץ פחות 1) פעולות המחברות "מכיוון עלייה שמאלה" (בנים ימניים) ולכן הפרשי גבהי העצים בפעולות אלו יהיו נמוכים.

לעומת זאת, פעולת ה- האחרונה תבצע בין עץ ריק (תת העץ הימני של האיבר המקסימלי בתת העץ השמאלי, שהוא עלה ולכן תת העץ ריק) לבין כל תת העץ הימני של העץ כולו.

הפרש הגבהים במקרה זה הוא וזו תהיה העלות המקסימלית ל- כחלק מ- זה. נקבל:

**שאלת בונוס**

נשים לב שעבור של איבר אקראי, עלות ה- המקסימלי היא אורך הרצף המקסימלי של עליות מהאיבר לשורש "מאותו הכיוון" (כלומר שרשרת של בנים שמאליים או שרשרת של בנים ימניים).

*נשים לב, שאיבר הוא בן שמאלי בהסתברות ובן ימני בהסתברות .*

*לכן, אם נתבונן באיבר האקראי הנבחר וברצף האבות ממנו עד השורש, נרצה לספור את כמות הבנים הרצופים מאותו הכיוון, מה שתואם לצורת התפלגות גיאומטרית אותה ראינו בקורס מבוא להסתברות.*

*ובפרט:*

* *רצף של שני איברים מאותו הכיוון יתקבל בהסתברות ועלות ה- תהיה .*
* *רצף של שלושה איברים מאותו הכיוון יתקבל בהסתברות ועלות ה- תהיה 3.*
* *רצף של כל האיברים מאותו הכיוון יתקבל בהסתברות ועלות ה- תהיה .*

*ממשפט התוחלת השלמה נקבל את הסכום:*

*מחיפוש ברשת האינטרנט נמצא כי אורך הרצף המקסימלי (בדומה לבעיית רצפים בהטלת מטבע) הוא . אם אורך הרצף המקסימלי הוא , עלות ה- תהיה ומכאן נקבל את התוצאה .*