עבודה 4 – חלק תיאורטי

1. \*נסמן ב-h את גובה העץ במהלך ביצוע ה-insert.

|  |  |  |
| --- | --- | --- |
| **טיפוס** | **שימוש** | **סיבוכיות זכרון** |
| null | לחצוץ בין ההכנסות למחסנית (הטיפול בפיצולים מסתיים כאשר שולפים null מסוף התור) | *לכל הכנסה*  *(פעולה בודדת)* |
| Node<T> | בכל פיצול בinsert- אנחנו מכניסים את החוליה שפוצלה (לפני שיצאה מהעץ) – על מנת שנוכל להחזיר אותה למקומה במהלך ה-backtracking. | *לכל פיצול,*  *סה"כ לכל הכנסה*  *(אי אפשר לפצל יותר חוליות מאשר גובה העץ בפעולת הכנסה בודדת)* |
| Node<T> | בכל פיצול בinsert- אנחנו מכניסים את ההורה של החוליה שפוצלה – על מנת שנוכל להחזיר את החוליה שפוצלה למקומה המקורי בעץ במהלך ה-backtracking. | *לכל פיצול,*  *סה"כ לכל הכנסה*  *(אי אפשר לפצל יותר חוליות מאשר גובה העץ בפעולת הכנסה בודדת)* |
| T | בכל פיצול ב-insert אנחנו מכניסים את ה-median value שעלה להורה. כך נוכל לשחזר את האינדקס שלו, ודרכו לגשת לאינדקסים של החוליות החדשות שנוצרו במהלך הפיצול על מנת למחוק אותן מהעץ במהלך ה-backtracking. | *לכל פיצול,*  *סה"כ לכל הכנסה*  *(אי אפשר לפצל יותר חוליות מאשר גובה העץ בפעולת הכנסה בודדת)* |
| Node<T> | בכל insert אנחנו מכניסים את החוליה שאליה הכנסו את הערך, על מנת שנדע מאיפה למחוק אותו במהלך ה-backtracking. | *לכל הכנסה*  *(פעולה בודדת)* |
| T | בכל insert אנחנו מכניסים את הערך שהוכנס לעץ, על מנת שנוכל למחוק אותו במהלך ה-backtracking. | *לכל הכנסה*  *(פעולה בודדת)* |

כל המידע הנ"ל נשמר ב-LinkedList שנקראת backtrackingDeque.

1. \*נסמן ב-h את גובה העץ במהלך ביצוע ה-backtrack.

|  |  |  |
| --- | --- | --- |
| **פעולה** | **מספר החזרות** | **סיבוכיות זמן ריצה** |
| removeKey | פעם אחת לכל הכנסה | *לכל הכנסה* |
| indexOf | פעם אחת לכל פיצול | *לכל פיצול – כי indexOf מבצע עד*  *2\*order-1 איטרציות, וזהו מספר קבוע.*  *סה"כ לכל הכנסה*  *(אי אפשר לפצל יותר חוליות מאשר גובה העץ בפעולת הכנסה בודדת)* |
| removeChild | פעמיים לכל פיצול (בן ימני ושמאלי) | *לכל פיצול – כי removeChild מבצע עד*  *2\*order-1 איטרציות, וזהו מספר קבוע.*  *סה"כ לכל הכנסה*  *(אי אפשר לפצל יותר חוליות מאשר גובה העץ בפעולת הכנסה בודדת)* |
| addChild | פעם אחת לכל פיצול | *לכל פיצול – כי צריך למיין מערך בגודל לכל היותר*  *2\*order-1, וזהו מספר קבוע.*  *סה"כ לכל הכנסה*  *(אי אפשר לפצל יותר חוליות מאשר גובה העץ בפעולת הכנסה בודדת)* |
| לולאת ה-for שנקראת לאחר כל שחזור של פיצול – עדכון המצביע להורה של כל ילד | פעם אחת לכל פיצול | *לכל פיצול – כי הלולאה מבצעת עד*  *2\*order-1 איטרציות, וזהו מספר קבוע.*  *סה"כ לכל הכנסה*  *(אי אפשר לפצל יותר חוליות מאשר גובה העץ בפעולת הכנסה בודדת)* |

1. אם בעץ יש n מפתחות, סיבוכיות זמן הריצה היא – לכל insert, כי דני צריך לשכפל כל מפתח בעץ (כדי לשכפל את כל העץ) במהלך ריצה, ו- לכל backtrack (רק צריך להחליף את העץ הישן בעץ החדש).
2. תלוי מה מבקשים לממש – אם מה שחשוב זה לממש backtrack יעיל, דני ניצח (כי אצלו זה ב- ואצלנו ב-). הפתרון שלנו מנצח בכל היבט אחר – אין פגיעה ביעילות של הinsert (שנשאר ב-, לעומת אצל דני), והפתרון הרבה יותר חסכוני בזיכרון (אצלנו שומרים רק של מידע לכל הכנסה, ואילו אצל דני ).