מבני נתונים 8

שחר פרץ

5 במאי 2025

מרצה: עמית ויינשטין

המציאות היא שיש לנו $O(1)$ "טוב" ו־$O(1)$ "רע". הסיבה? כי פעולה כמו חיבור מספרים מבוצעת כל cycle, לעומת גישה לזכרון שנבתצעת כל כמה סייקלים. אז במחשב של 2GHz:

Thing	CPU cycle	L1 Cache	Ram	Disk (swap)
Cycle Time	0.5ns	1ns	100ns	1-10ms

Where ns=10^{-9} and ms=10^{-6}

שימו לב – לא אנחנו קובעים מה נמצא ב־L1 cache. לנהל את כל זה זה התפקיד של מערכת ההפעלה. לפעולות הגישה לעיל קוראים IO Operations.

אם משהו לא נמצא ב־L1 cache אז נקרא לזה cache miss, ואם מה־RAM עוברים ל־disk זהו page fault. בגישה לזכרון, אם יש cache-miss נביעpage רלוונטי מ־RAM אל ה־cache. גודל page טיפוסי הוא 4KB. ה־HEAP בהכרח ב־RAM וה־stack (עמית לא בטוח לחלוטין) או by definition ב־cache או במקום אחר, כך או אחרת משמעותית יותר קרוב.

נבחין שעצים, רשימות מקושרות וכו, הרבה יותר יקרים כי ה־IO operations שלנו יכולים להיות מדפים שונים. זאת בניגוד למערכים שנמצאים בדף יחיד.

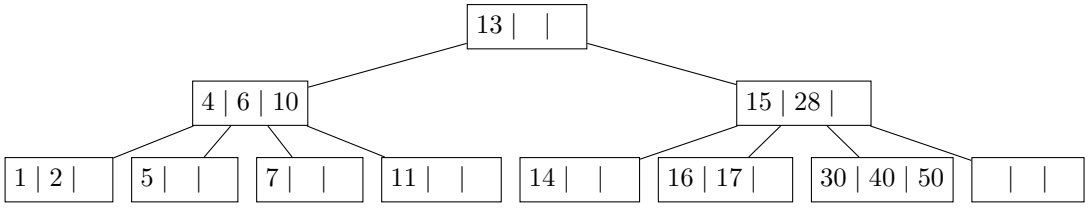
נדבר על כמות פעולות IO כפונקציה של הגודל של העץ. כרגע אנחנו עושים $\log n$ גישות לזכרון שכולן פעולות כבדות. אך לא נספור אותל בצורה אסימפטוטית אלה בצורה יותר מדוייקת. רעיון: שכל צומת יוכל לקפוץ "4 רמות קדימה".

בצורה שקולה, נשמור תת־עץ יותר רדוד. ניקח node־ים מכמה שכבות ואז נוכל להתייחס אליהם כאל node אחד. או כאילו נשמוד עץ יותר רדוד.

אז מה נעשה? נשמור node-ים יותר גדולים, שיכילו יותר הפניות, ואז אם הוא יחזיק k ערכים לפי גודל נוכל לדאוג ל־$\log_k n$ פעולות IO כבדות.

לשם כך המציאו B-trees ב־1972. בחורים בשם Bayer-Mcreight. לסימון, עץ (n, m) יהיה B-tree שבו יש בין ל־m ילדים. נוכל לדאוג לכך שכל העלים באותו העומק (ובפרט כאשר נצטרך להגדיל את העומק, נעשה זאת לכל הילדים בבת־אחת). השורש מיוחד כי לא ניתן לחייב שיש לו d ילדים, אלא בי 0 ל־m ילדים.

דוגמה לעץ $(2, 4)$:



נניח בעץ $(d, 2d)$ מעומק h, כמה קודקודים יש? תשובה:

$$d^{h-1} \cdot 2 \leq \text{leafs}$$

(הכפל ב־2 מגיע מהשורש). $d^h \leq (d-1)d^{h-1} \cdot 2 \leq n$. ואז $h \leq \log_d n$. העלות בחיפוש עצמו בתוך ה־node תגדל – $\log d$ כי צריך לעשות בו חיפוש בינארי. אז הסיבוכיות לכאורה $\log 2_d \log_d n$, זה לא גרוע יותר אסימפטוטית ועדיין משמעותית פחות פעולות IO. אגב, יש מימושים ששומרים מערכים רק בעלים, זה לא כזה קריטי בסקאלה גדולה.

מכאן ואילך תהנו באנימציות של המצגת שאין לי ואני לא עומד לעשות. לא נעשה AVLs אלא "נפעפע" למלה צמתים. לא מחזיקים מצביע לאבא, ונשמור על מחסנית הרקורסיה את ה־node־ים שעברנו בהם. כאשר נוסיף node חדש, נקווה שיש מקום, אחרת נעלה למעלה ונקווה ששם אין overflow, כאשר יש overflow נעשה split ונעלה למעלה, וכו' ברקורסיה. אפשר להראות שזה amortized ב־$O(1)$ (בלי מחיקות). לכך נקרא הכנסה bottom-up. החסרון הוא העליה מעלה בזמן התיקונים (ונוכל רק לקוות שרשימת הצמתים שעברנו בהם ב־cache). גישה אחרת אומרת, שבזמן הירידה נפצל קודקודים מלאים מחשש שאחרי כן נצטרך לעלות הכל.

למחיקות: כמו בעצים רגילים נעשה החפה בין predecessor ו־successor, ומוחק מעלה. עתה כדי לאזן יש לנו שתי אפשרויות. הראשונה היא borrow fuse, שדורש לוויה מצומת מחובר לאבא של העלה ("ללוות מהאח"). אם אי אפשר לעשות borrow נעשה fuse, כלומר לאחד שני עלים

("להתאחד עם אח"). כל הדברים האלו עולים עוד פעולת IO וסידור קטן של הדברים בפנים. גם כאן אפשר להראות ש־amortized זה בסדר.

. . . . . .

שחר פרץ, 2025

קומפל ב־LaTeX וצור באמצעות תוכנה חופשית בלבד