מבני נתונים - הרצאה 7

20 בנובמבר 2011

סיכום עצים

- ייצוג עצים:
באמצעות מערך (עצים בינאריים)
רשימות - כל הבנים או בן בכור.
מעבר למצביעים לבנים לעיתים מחזיקים מצביע לאב.
- מעבר על עצים:
in-order, pre-order, post-order (רק על עצים בינאריים)
על עצים לא בינאריים - BFS (באמצעות תור) וDFS (באמצעות מחסנית).
- עצי חיפוש - ענף ימין גדול מהשורש וענף שמאל קטן מהשורש.
זמן החיפוש - $O(\log n)$.
הכנסה - $O(\log n)$
בהוצאה יש צורך בתיקון העץ - אם מה שמוציאים עלה, לא צריך לתקן.
אם לצומת שמוציאים יש בן יחיד, מוציאים אותו ומעבירים את הבן להיות הבן של אביו.
אם לצומת יש שני בנים, עושים חילוף בינו לבן השמאלי ביותר של תת העץ הימני ואז מורידים את הבן השמאלי ביותר של תת העץ הימני.
- בעיה - אם מכניסים סדרה מסודרת לעץ חיפוש העלות עלולה להיות $O(n)$, לכן משתמשים בעצי חיפוש מאוזנים.
- עצי AVL - ההפרש בין גובה תת העץ הימני לגובה תת העץ השמאלי לכל צומת בעץ הוא לכל היותר 1.
מסתכלים על המסלול מהשורש עד לפעולה שעשינו (הכנסה או הוצאה) ומאזנים, בעלות כוללת של $O(\log n)$.
- עצי 2-3 - הנתונים נמצאים רק בעלים, הצמתים הפנימיים מכילים אינדקסים מפרידים, לכל קודקוד פנימי יש 2 או 3 בנים, וכל העלים באותה רמה.

מיון Sorting

מיונים מתחלקים לשני סוגי מיונים - מיוני השוואה ומיונים שאינם מיוני השוואה (Comparison, Non-Comparison).
במיוני השוואה, הפעולה היחידה היא השוואה, ממיינים בכך שמשווים בין הדברים שממיינים לפי קרי-טריון כלשהו.
במיונים שאינם מיוני השוואה, משתמשים ממש בערכים שממיינים.
אפשר לחלק את המיונים לעוד שתי קטגוריות:
מיונים יציבים ומיונים לא יציבים.
במיונים יציבים, אם $n_i = n_j$ ו$i < j$ אז n_i יישאר לפני n_j גם אחרי המיון, ובמיונים לא יציבים התכונה הזו לא מובטחת.
למשל, נניח שהמספרים שלנו מתחלקים למספרים רגילים ומספרים רומיים, ותחילה הם ממוינים כך שכל המס' הרגילים משמאל והרומיים מימין:

$$1, 5, 2, \mathrm{II}, \mathrm{IV}, \mathrm{I}$$

נרצה למיין לפי גודל המספר, במיון יציב נקבל בהכרח:

$$1, \mathrm{I}, 2, \mathrm{II}, \mathrm{IV}, 5$$

במיון לא יציב נוכל לקבל

$$\mathrm{I}, 1, 2, \mathrm{II}, \mathrm{IV}, 5$$

יעילות מיון השוואה

יש לנו $n!$ פרמוטציות של הסדרה, ואנו צריכים לבחור את הפרמוטציה הנכונה.
כשאני עושה השוואה, אני מקבל שני מצבים - $a > b$ או $b > a$. אחרי k השוואות יש 2^k מצבים. צ"ל מתי $2^k = n!$, אז

$$k = \log_2 (n!)$$

נשתמש בקירוב סטרלינג:

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

אזי

$$\begin{aligned} k &= \log_2 n^n - \log_2 (e^n) + \log_2 \left(\sqrt{2\pi} n\right) \\ &= n \log n - n + \log \left(\sqrt{2\pi} n\right) = \Theta (n \log n) \end{aligned}$$

כלומר כל אלגוריתם שמבוסס על השוואה ממיין בעלות של לפחות $n \log n$.

אלגוריתמי מיון

אלגוריתם 1 Selection Sort

1. עבור על כל המערך ומצא את הערך הנמוך ביותר
2. החלף בינו ובין המקום הראשון במערך
3. הפעל Selection Sort על המערך החל מהמקום השני, אלא אם כן גודל המערך הוא 1.

העלות של אלגוריתם זה היא בקירוב $\frac{n^2}{2}$, כלומר יעילות של $O(n^2)$.

אלגוריתם 2 Bubble Sort

1. עבור על כל זוג שכנים. אם שכן קטן מזה שלפניו, החלף ביניהם.
2. סיים כאשר המערך מסודר.

הבעיה עם אלגוריתם זה היא מס' קטנים שנמצאים בסוף הרשימה ההתחלתית, שגורמים לכך שהאלגוריתם יעשה המון השוואות.

אלגוריתם 3 Cocktail Sort

מבצעים Bubble Sort פעם לפנים ופעם לאחור.

האלגוריתם הזה לא משנה את היעילות בצורה אסימפטוטית והעלות הטיפוסית שלו היא $O(n^2)$ אבל הוא עדיף מבחינת זמן אפקטיבי שהוא עובד, במיוחד אם המערך כמעט מסודר.

אלגוריתם 4 Insertion Sort

עבור על כל הרשימה. עבור כל ערך שהגענו, הזז אותו אחורה עד לערך הראשון (כלומר, הגבוה ביותר) שקטן ממנו.

נשים לב שהזזה אחורה פירושה הזזה שכל הערכים באמצע במקום אחד קדימה. לכן נשים לב שבמצב זה יותר אפקטיבי להשתמש ברשימה מקושרת, ומיון זה טוב כאשר הרשימה כמעט ממויינת בצורה שכל ערך נמצא מס' מקומות קטן מהמקום המתאים לו במערך, אך עדיין הוא $O\left(n^2\right)$.

אלגוריתם 5 Quick Sort

---

1. אם גודל המערך הוא 1, החזר את הערך.
2. בחר pivot
3. בצע Quick Sort על כל הערכים שקטנים או שווים לpivot
4. בצע Quick Sort על כל הערכים שגדולים מהpivot

העלות הטיפוסית של המיון הזה היא $O\left(n\log n\right)$ אך במקרה הגרוע ביותר הוא עדיין $O\left(n^2\right)$ (המקרה הגרוע ביותר הוא כאשר המערך ממוין, ולוקחים את הpivot כאיבר הראשון).

אלגוריתם 6 Heap Sort

---

1. הכנס הכל לערימה
2. הוצא הכל מהערימה

אלגוריתם זה הוא בעלות של $O\left(n\log n\right)$ (גם במקרה הגרוע ביותר) אך הוא לא יציב.