

אינדקסים ופונקציות Hash

במחשבים ישנים, בהם שמירת הקבצים נעשתה על-גבי סרט מגנטי, עבודה עם קבצים גדולים, ובעיקר ביצוע פעולות חיפוש שונות באותם קבצים, הייתה מסורבלת. הפעולות האפשריות בעת עבודה עם אותם הקבצים היו Rewind – חזרה לתחילת הקובץ, ו-Next – מעבר לרשומה הבאה. למרות מגבלות אלו, כאשר הקבצים נשמרים כקבצים סידרתיים, כלומר כקבצים ממוינים, שבהם כל הרשומות נרשמות אחת אחרי השנייה לפי סדר המיון, ניתן לבצע פעולות חיפוש באופן יעיל יחסית, למשל, בעזרת חיפוש בינארי.

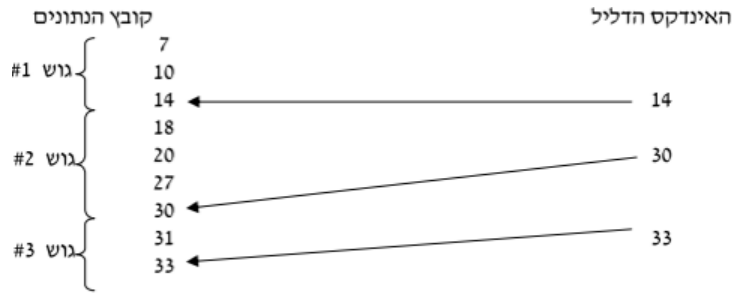
עבודה עם קבצים סידרתיים אינה פשוטה. פעולות כגון הוספת רשומה או מחיקת רשומה דורשות עבודה רבה עם הקובץ, ולעיתים אף את כתיבתו מחדש. התפתחויות טכנולוגיות (כגון פיתוח הדיסקים הקשיחים) ובעקבותיהם חידושים במערכות ההפעלה של המחשבים מאפשרים היום גמישות גדולה ביותר בעבודה עם קבצים, בעיקר מכוון שניתן להגיע בקלות לכל מיקום בקובץ ללא צורך לעבור על כל הרשומות בקובץ. הרעיון של קבצים סדרתיים נשמר. אולם במקום לשמור את הרשומות ממוינות בקובץ, אחת אחר השנייה, אנו מוסיפים בסוף כל רשומה מצביע לרשומה הבאה. באופן כזה, אנו יכולים לקרוא את הקובץ רשומה אחרי רשומה, בהתאם לאופן המיון, אולם פעולות של הוספת רשומה, או מחיקת רשומה, הופכות להיות פשוטות, ואינן דורשות כתיבה מחדש של הקובץ, אלא עדכון של המצביעים בסוף כל רשומה ורשומה.

אינדקס הוא מבנה נתונים (קובץ) המאפשר גישה ישירה לפי מפתח, מבלי שיהיה צורך לסרוק את הקובץ הסדרתי.

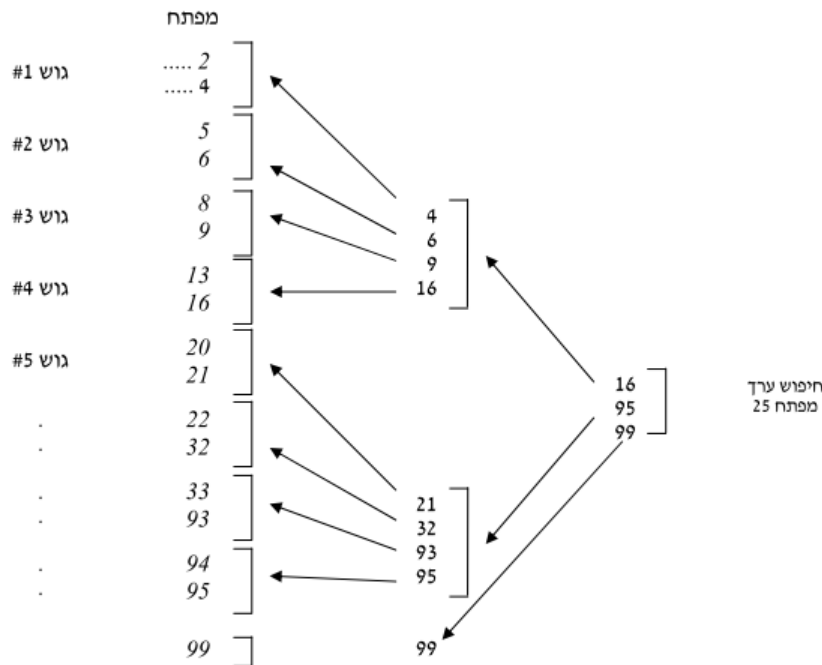


גודל קובץ האינדקס הוא משמעותית קטן יותר מקובץ הנתונים, ולכן, ברוב המקרים, ניתן לשמור אותו בזיכרון, ובכך להגדיל משמעותית את זמני הגישה לרשומות השונות בקובץ הנתונים.

אפשר לחלק את קובצי האינדקס לשתי קבוצות: (1) אינדקס צפוף – אינדקס המכיל רשומת אינדקס לכל ערך של מפתח החיפוש. (2) אינדקס דליל – אינדקס המכיל רשומות אינדקס לחלק מהערכים של מפתח החיפוש. במקרה כזה, המפתחות מחולקים לגושים (ממוינים), וכל כניסה באינדקס תכיל את הערך הנמוך (או הגבוה) באותו גוש של מפתחות. כמו כן, אנו משתמשים בעובדה שקובץ הנתונים הוא קובץ סידרתי, כך שניתן לסרוק את הרשומות, ולהגיע לרשומה הרצויה.



ככל שקבצי האינדקס גדלים יעילותם קטנה. במקרים כגון אלו, בכדי להגדיל את היעילות, ניתן לבנות אינדקסים לאינדקסים עצמם.



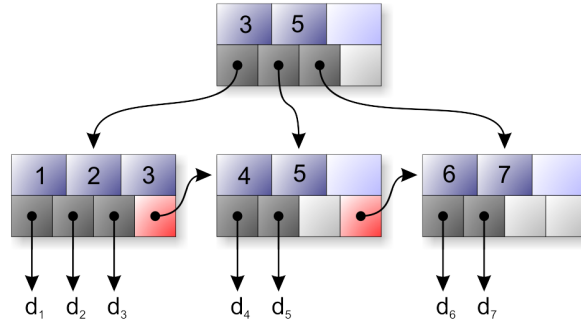
למרות שברוב המקרים אנו עובדים לפי המפתח הראשי, יתכנו מקרים שהם נרצה לבנות אינדקסים נוספים, על מנת שנוכל לבצע שליפות על פי חתכים נוספים באופן יעיל. מכוון שקובץ הנתונים נשמר כקובץ סידרתי, הממוין לפי המפתח הראשי, מימוש של אינדקסים משניים יכול להתבצע אך ורק ע"י אינדקס צפוף.

כאמור, ככל שקבצי האינדקס גדלים יעילותם קטנה, ולכן נהוג לבנות אינדקסים לאינדקסים עצמם. אחת מהשיטות לעשות זאת היא ע"י שימוש במבנה נתונים הנקרא עץ B+ (קיימים מבנה נתונים נוספים עליהם לא נדבר). עץ B+ הוא עץ חיפוש, כלומר עץ ממוין. קדקוד בעץ B+ מכיל n ערכים ו-n+1 מצביעים. עבור כל ערך ישנם שני מצביעים, אחד מצביע לקדקוד בו יש ערכים n מוכים ממנו, ואחד מצביע לקדקוד בו יש ערכים זהים או גדולים ממנו.

ישנם מס' כללים החלים על עצי B+:

1. כל הערכים נמצאים בעלים, וכל העלים נמצאים באותה הרמה
2. אם השורש אינו עלה, חייבים להיות לו לפחות שני בנים.
3. לעץ מסדר n (כלומר שכל עלה מצביע על מקסימום n רשומות), פרט לשורש ולעלים, יש בין $n/2$ ל-n בנים.

סדר העץ (n) נקבע בהתאם לדיסקים בהם נשמר הקובץ ומערכת ההפעלה. בעת ביצוע פעולת קריאה מקובץ, מערכת ההפעלה קוראת את הקובץ בגושים. המטרה שלנו היא שכל צומת בעץ ישכון בגוש משלו (כך, במידה ואנו עובדים עם הדיסק, מספר הפניות שנעשה יהיה מינימאלי). בהתאם לכך ועל סמך המשתנים השונים שאנו שומרים בכל צומת (הערכי המפתחות והמצביעים לצמתים הבאים), נחשב את סדר העץ.



הוספת ערך לעץ B+ נעשית בהתאם לשלבים הבאים:

1. נמצא מקום להוספת האיבר החדש ברמה התחתונה.
2. נוסיף את הערך החדש עם המפתח החדש לאביו.
3. אם דרגת צומת האב תקינה.
4. אחרת, נבצע תיקון לצומת האב.

תיקון צומת אב נעשה ע"י פיצול הצומת לשני צמתים, השמאלי יקבל את ה- $\lfloor (m + 1)/2 \rfloor$ הבנים הראשונים, והימני את ה- $\lfloor (m + 1)/2 \rfloor$ הבנים האחרונים. נמשיך את התיקון באופן רקורסיבי. אם יש צורך לפצל את השורש, נוסיף שורש חדש.



מחיקת ערך מעץ B+ נעשית בהתאם לשלבים הבאים:

1. נמצא את הערך ברמה התחתונה.
2. נמחק אותו מהעץ יחד עם המפתח המתאים לאביו.

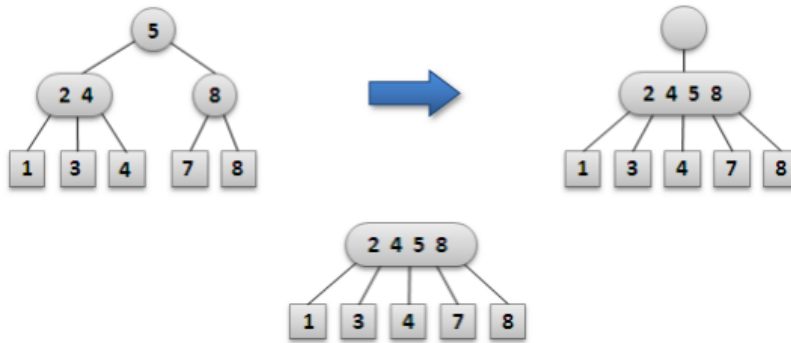
3. אם דרגת צומת האב תקינה, נסיים.
4. אחרת, נבצע תיקון לצומת האב.

תיקון צומת אב נעשה בשני מצבים:

1. אם לאח השמאלי (הימני) של צומת האב הוא בעל יותר מ- $\lfloor \frac{m}{2} \rfloor$ ילדים, נלווה ממנו את בנו הימני (השמאלי) ביותר ונסיים.



2. אחרת (לאח יש בדיוק $\lfloor \frac{m}{2} \rfloor$ ילדים), נאחד את הצומת עם אחיו ונמשיך את התיקון כלפי מעלה באופן רקורסיבי. אם בסיום הפעולה לשורש בן יחיד, נמחק אותו.



בקובץ אינדקס סידרתי, בכדי לפנות לרשומה מסוימת יש צורך לעבור על פני כל הרמות של האינדקס. אם קובץ האינדקס הוא גדול, ולא ניתן לשמור את כולו בזיכרון, יש צורך בגישות לדיסק הקשיח, שהינה פעולה יחסית איטית.

ניתן ליצור קבצים בהם הגישה לכל רשומה היא באופן ישיר, אולם בצורה זו אנו מאבדים את היכולת לסרוק את הקובץ.

בכדי לעשות זאת, אנו בונים מראש קובץ ובו מס' נתון של רשומות "ריקות" מוכנות מראש. כל רשומה, על סמך המפתח שלה, מקבלת מקום בקובץ. את המקום בקובץ אנו מחשבים בעזרת פונקציית hash. השימוש בפונקציית hash הינו יעיל כאשר המפתח אינו רציף (למשל, אם המפתח הוא מס' זהות). במקרים כאלו, אם היינו בונים קובץ ובו מס' הרשומות מחושב על-פי המפתח, היינו מקבלים קובץ ובו הרבה רשומות ריקות, שלא היו מתמלאות אף פעם. פונקציית ה-hash תפקידה הוא להמיר את המפתח מערך אחד לערך אחר, שנמצא בטווח ערכים קטן יותר. כך אנו יכולים לבנות קובץ נתונים קטן יותר, ולהבטיח שמספר הרשומות הריקות בו יהיה קטן משמעותית.

קיימות מס' פונקציות hash נפוצות. (1) חלוקה ושארית - ערכו של המפתח מחולק במספר כלשהו, D , והתוצאה המוחזרת היא השארית המתקבלת מהחלוקה. בכדי לקבל התפלגות אחידה, יש לבחור D שהוא ראשוני וגדול פי 1.25 ממספר הרשומות הצפוי בקובץ. (2) ריבוע ומיצוע - ערך המפתח מועלה בריבוע, ומהתוצאה נשלפות n ספרות המצויות במיקומים $1, m+1, \dots, m$. אם הספרות הן בבסיס B , אז גודל הקובץ צריך להיות B^n . (3) קיפול וחיבור - ספרות המפתח מחולקות לקבוצות (בגודל n). לכל קבוצה מתייחסים כמספר. בכל מספר משנים את סדר הספרות, באופן קבוע מראש, ואת המספרים המתקבלים סוכמים. אם כל קבוצה היא בגודל n ספרות בבסיס B , אז גודל הקובץ צריך להיות B^n .

יתכן ועבור שתי רשומות שונות פונקציית ה-hash תחזיר ערך זהה, ולכן יש צורך במנגנון שיטפל במקרים כאלו. ישנן מס' שיטות לכך:

1. מיעון לשטח גלישה - הרשומה הראשונה תשמר במיקום שהוקצע לה ע"י פונקציית ה-hash, ואילו שאר הרשומות (שממופות לאותה הכתובת) ישמרו בשטח מיוחד שמוקצה לרשומות שהמיקום שלהם כבר תפוס.
2. מיעון פתוח - מען את הרשומה שהמיקום שלה תפוס למיקום חדש בשטח הראשי של הקובץ ע"י הפעלת פונקציית hash חוזרת (דוגמא לפונקציה כזו - גלישה רצופה - התקדם עד שתמצא מקום פנוי).
3. מיעון דליים - דלי הוא שטח רצוף (גוש) המכיל מס' רשומות. פונקציית ה-hash קובעת את כתובת הדלי בו נשמרת הרשומה. כאשר הדלי מתמלא יש שתי אפשרויות. (1) ביצוע hash מחדש על-מנת למצוא דלי חדש, או (2) להשתמש ברשימה משורשרת של דליים.

שימוש בפונקציית hashing אחת קבועה, נקראת hash סטטי. מכוון שקיים קשר בין פונקציית ה-hash לגודל הקובץ, קביעת פונקציית ה-hash חיונית לנו:

- א. אם פונקציית ה-hash מתאימה לגודל הקובץ הנוכחי, הביצועים יהיו ירודים אם הקובץ יגדל.
- ב. אם פונקציית ה-hash מתאימה לגודל הקובץ העתידי, הביצועים יהיו טובים, אבל אנו נבזבז כמות שטח גדולה בקובץ עד שהוא יגדל לגודל הצפוי.
- ג. ביצוע ארגון מחדש של הקובץ מעת לעת הינו תהליך יקר.

הפתרון הוא hash דינאמי, אותו ניתן להשיג ע"י שימוש ב-hash בר הרחבה. Hash בר הרחבה מבוסס על מיזוג ופיצול דליים בהתאם לשינויים בגודל של בסיס הנתונים.

לצורך ביצוע hash בר הרחבה אנו נבחר פונקציית hash אשר תיתן לנו טווח גדול של דליים (מעבר למה שאנו צריכים). נניח שהפונקציה שלנו מחזירה טווח של מספרים טבעיים בייצוג בינארי. מכוון שאנו לא רוצים להשתמש בכל הערכים האפשריים של פונקציית ה-hash (הרבה דליים) נתייחס רק ל- i הספרות המשמעותיות בייצוג הבינארי של הערך שקיבלנו.

במידה והתייחסות רק ל- i הספרות המשמעותיות אינו אפשרי, כלומר יש דלי שהינו מלא, אולם על פי הייצוג רשומה נוספת צריכה להתווסף לאותו הדלי, נגדיל את i ב-1, ונבצע את החלוקה של הדליים מחדש.