אחזור מידע מהאינטרנט (67782) תרגיל 1: מבנה האינדקס - ניתוח

מוגש על־ידי: מיכאל גרינבאום (211747639), גיא לוי (211744636)

25 באפריל 2021

1 תיאור כללי ודיאגרמה

1.1 תיאור

- ראשית נתאר את המחלקות שלנו באופן גנרי. יצרנו 2 מחלקות מרכזיות:

1. ConcatenatedStringDictionary: זה מחלקה אבסטרקטית שאחראית על ניהול שדה אינדקס של מציעים/תכונות ערכים של סטרינג ארוך, כפי שראינו יש מספר דרכי מימוש לשדה זה באינדקס, מימשנו את האופציות להלן:

(א) ConcatDict: מחלקה של המימוש הטרוויאלי - לכל מילה בסטרינג הארוך שמרנו פוינטר להתחלה שלו (ולפי הפוינטר של המילה הבאה נדע בדיוק איך לקרוא את המילה).

(ב) BlockingDict: מחלקה ששומרת את המצביע לתחילת כל מילה k ולכל מילה את האורך שלה.

(ג) FrontCoding: מחלקה שמממשת גם את הרעיון של BlockingDict וגם עושה front coding לכל בלוק של k מילים בסטרינג הארוך.

2. בכל מימושים אלו של המילון הקריאה היא טריוויאלית ומציאת האינדקס של מילה במילון ניתנת למעשה על־ידי חיפוש בינארי (כפי שהוסבר בהרצאה).

3. Index: למחלקה זו יש שני שימושים:

(א) בניית אינדקס: ראשית האינדקס מקבל את כל המבנה שלו - שמות השדות שלו, גודל כל שדה (בבתים), ושדות של ConcatenatedStringDictionary (שבפועל האינדקס ישמור את מספר השורה המתאים באובייקט ConcatenatedStringDictionary המתאים לאותו שדה).
אובייקט האינדקס קובע במדויק מה האינדקס של כל שדה במערך הבתים (לפי גודל שאר השדות וסדר ההצהרה על השדות באינדקס).
לאחר אפיון הצורה של האינדקס, האינדקס אחראי על הוספת ערכים לשדות שלו - ויש פונקציות מתאימות שמצהירות על התחלת שורה חדשה באינדקס, קבלת שם שדה וערכו (מערך של בתים) ומכניס למיקום המדויק) וסיום הכנסת השורה לאינדקס.
בעת הבנייה האינדקס שמור ברשימה של [מערכי בתים באורכים קבועים] (מערכי הבתים מהווים את השורות של האינדקס, הם תמיד באותו האורך) - בסוף הבנייה ממירים את הרשימה למערך רק של בתים ושומרים לדיסק.

(ב) קריאה מהאינדקס: צריך להפריד בין שדות שונים של האינדקס וכן בין קריאות שונים: קריאה לפי מספר שורה:

i. שדה רגיל: שדה השומר ערך, אז הערך של שדה זה בשורה ה־ i יהיה שמור בתאים

$$\text{index}[\text{field\_first\_index} + i\cdot\text{row\_size} + 0], \ldots, \text{index}[\text{field\_first\_index} + i\cdot\text{row\_size} + \text{field\_size - 1}]$$

כאשר first_field_index זה מיקום השדה בשורה של האינדקס (נקבע בעת אתחול השדות באינדקס), field_size זה גודל השדה בבתים (נקבע בעת אתחול השדות באינדקס) ו־ row_size זה גודל השורות באינדקס בבתים (נקבע בעת אתחול השדות באינדקס).

ii. שדה של ConcatenatedStringDictionary: שדה זה למעשה מכיל מספר i שמהשמעות שלו זה המילה ה־ i באובייקט המחלקה ConcatenatedStringDictionary - אז השדה מחזיר למשתמש את המילה ה־ i באותו אובייקט של מילון. עבור מילונים שהשורות שלהם מסונכרנים עם שורות האינדקס רק נשמור קישור לאובייקט המילון, המשתמש צריך לדעת עם המילון מסונכרן או לא עם האינדקס (האינדקס נבנה על־ידי המשתמש אולי תחת הנחה כזו עבור חלק מהמילונים).

iii. יכול להיות שנרצה לקרוא לפי מילה במילון, לדוגמא לקרוא ערך של שדה באינדקס בהינתן token, כדי לעשות זאת דבר ראשון צריך שאותו המילון יהיה מסונכרן בשורות שלו עם שורות האינדקס, אנחנו לא מוודאים את זה, זו אחריות המשתמש לוודא זאת (אחרת אותו המשתמש לא מבין את מבנה האינדקס אותו יצר). אם קיים סנכרון כזה ניתן לחפש במילון מה האינדקס של המילה הנתונה (אם קיימת) ולהחזיר את ערך השדה באותו מספר שורה שהוחזר על־ידי המילון.

4. נשים לב כי אינדקסים שונים יכולים להיות שותפים לאותו מילון (הקשר של האינדקס למילון זה שהם שומרים קישור לאובייקט ושומרים את אינדקסי המילים המתאימים למילים באותו אובייקט של מילון).

5. בנוסף הכנסנו דחיסה לפונקציות המתאימות, ראשית נפרט את הדחיסות: הפרדנו בין דחיסה של מילים לבין דחיסה של מספרים:

(א) AbstractIntegerCompression: מחלקת אב אבסטרקטית למימושים שונים של דוחס מספרים (דוחס מערך של בתים)

i. GammaCompression, DeltaCompression.

ii. GamDeltaCompression: שילוב של גאמא ודלתא - את כל האינפוטים האי־זוגיים (לפי סדר ההכנסה לפונקציה encode של האובייקט) מצפין לפי גאמא ואת הזוגיים לפי דלתא.

iii. VariantCompression, lengthVariantCompression, groupVariantCompression.

(ב) AbstractStringCompression: מחלקת אב אבסטרקטית למימושים שונים של דוחס רצפי סימבולים (שניתנים לייצוג על־ידי סטרינג).

i. HauffmanCompression

6. המקום בו השתמשנו בדחיסה:

(א) ConcatenatedStringDictionary: בעת שהוספנו מילה לסטרינג הארוך דחסנו אותה (ושמרנו מצביעים בהתאם לסטרינג החדש). בעת שקראנו את המילה מהקובץ פענחנו אותה. נשים לב שהסטרינג הארוך יכול להיות סטרינג גם של מספרים (לדוגמא כשנרצה לשמור רשימה של מספרים בעלת אורך לא מוגדר) - אז השתמשנו בדוחס מתאים בהתאם לסוג הסטרינג.

(ב) בעת שמירת האינדקס שמרנו אותו בדיסק על־ידי דוחס מספרים.

- תיאור לא גנרי:

1. יצרנו 4 מילונים:

(א) מילון של pid: שומר מילון של כל ה pid ממוינים.

(ב) מילון של token: שומר מילון של כל ה token ממוינים.

(ג) מילון של token_(id, freq)_list: שומר לכל token את הרשימה של ה־ (id, freq) כאשר סדר ההופעות שלהם בסטרינג הוא לפי סדר ה token (יש התאמה חח"ע ועל ביניהם לבין הטוקנים הרי) (נשים לב שאין אינטרס לבצע חיפוש בינארי של רשימה, וכן שימוש כזה לא ממוין לפי הערכים במילון לא מאפשר שימוש בפונקציונליות כזו).

(ד) מילון של pid_id_list: שומר לכל pid את ה review id ההתחלתי וכמות ה־ reviews שמתאימים ל־ pid. הערות:

i. ניתן לשמור ככה כי נתון שמשובים על אותו pid הם משובים עוקבים.

ii. אפשר לשמור פשוט באינדקס, אבל מצד שני זה חוסך מקום בזיכרון המרכזי (אז זה לא מיותר לגמרי).

2. יצרנו 3 אינדקסים:

(א) אינדקס ל־ review: מכיל את השדות הבאים:

i. שדה pid: מכיל מספר המסמן מספר שורה במילון pid (והמספר שורה במילון מצביע ל־ pid המתאים למשוב הנוכחי). [בגודל 4 בתים]

ii. שדה score: מכיל את הscore. [בגודל 1 בתים]

iii. שדה helpfulness numerator: מכיל את הhelpfulness numerator. [בגודל 4 בתים]

iv. שדה helpfulness denominator: מכיל את הhelpfulness denominator. [בגודל 4 בתים]

v. שדה ה־ review length: מכיל את כמות הטוקנים של המשוב. [4 בתים]

(ב) אינדקס ל־ pid: זהו אינדקס חסר שורות, מכיל רק מילונים שמסונכרנים עם השורות שלהם אחד עם השני.

i. שדה pid: מכיל מצביע למילון pid.

ii. שדה pid_id_list: מכיל מצביע למילון pid_id_list.

(ג) אינדקס ל־ token: מכיל את השדות הבאים:

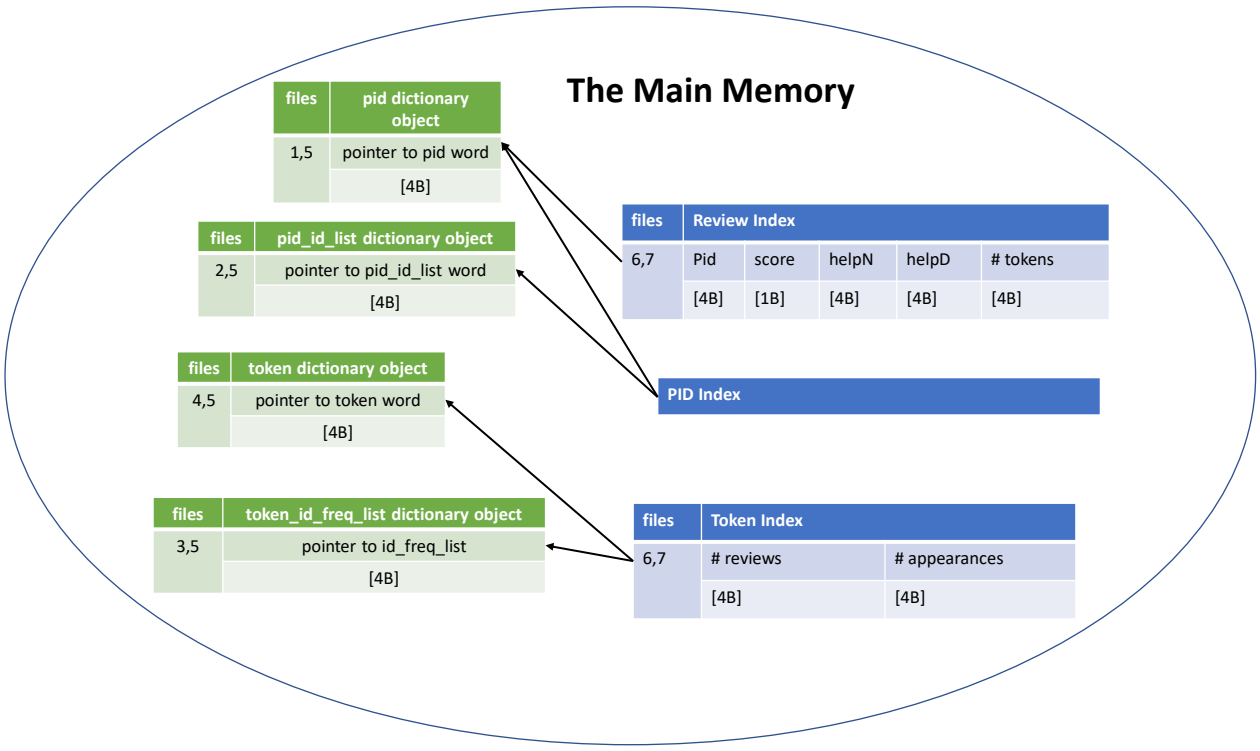
i. שדה token: מצביע למילון token (האינדקס והמילון הזה מסונכרנים בשורות).

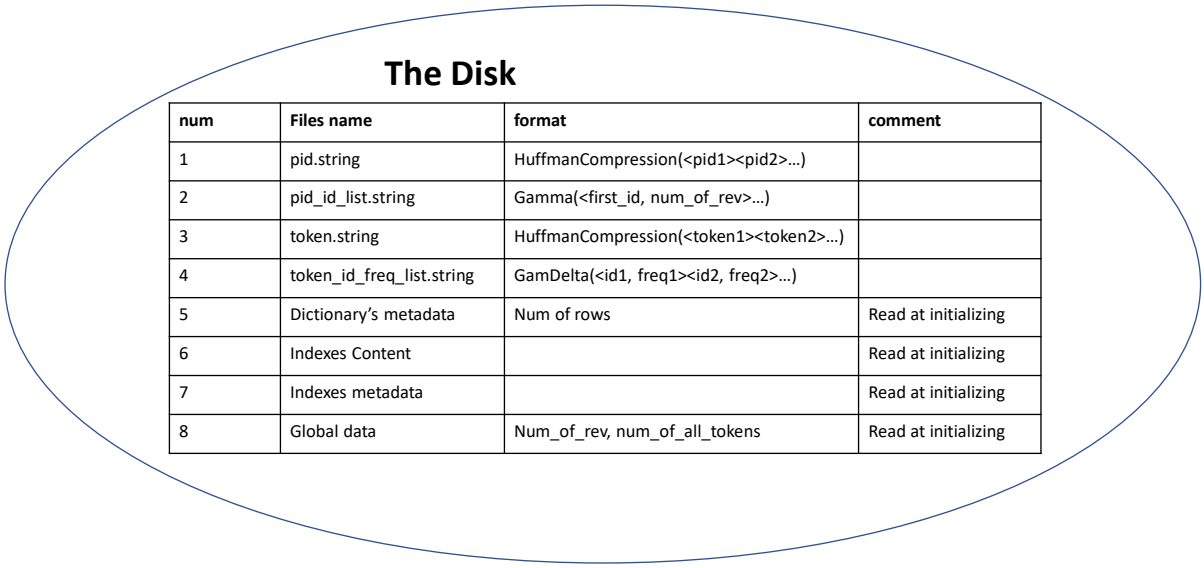
ii. שדה token_(id, freq)_list: מכיל מצביע מילון token_(id, freq)_list. (האינדקס והמילון הזה מסונכרנים בשורות).

iii. שדה num_reviews_contain_token: מכיל את כמות המשובים שמכילים את הטוקן. [בגודל 4 בתים]

iv. שדה num_appearances: מכיל את כמות המופעים של הטוקן באוסף. [בגודל 4 בתים]

1.2 דיאגרמה





The Disk

num	Files name	format	comment
1	pid.string	HuffmanCompression(<pid1><pid2>…)	
2	pid_id_list.string	Gamma(<first_id, num_of_rev>…)	
3	token.string	HuffmanCompression(<token1><token2>…)	
4	token_id_freq_list.string	GamDelta(<id1, freq1><id2, freq2>…)	
5	Dictionary's metadata	Num of rows	Read at initializing
6	Indexes Content		Read at initializing
7	Indexes metadata		Read at initializing
8	Global data	Num_of_rev, num_of_all_tokens	Read at initializing

2 הזיכרון המרכזי Vs הדיסק

- החלקים שנקראים לזיכרון המרכזי בעת אתחול ה־ IndexReader: כל השדות באובייקטי ה Index כפי שמפורט לעיל. בנוסף אנחנו קוראים meta data של המילונים והאינדקסים (כמות השורות במילונים והאינדקסים).
- החלקים שנקראים בעת הצורך: השדות שנשמרים בסטרינג הארוך של המילונים, כפי שמציונים לעיל. אובייקט המילון יודע לאן בתוך הקובץ לפנות כדי לקרוא את מה שרלוונטי.

3 ניתוח תיאורטי של הגודל

- סימונים:
  - N: כמות המשובים
  - M: כמות הטוקנים הכוללת (כולל הופעות של אותו הטוקן)
  - D: כמות הטוקנים השונים
  - L: האורך הממוצע של טוקן
  - H: ממוצע ה־ id שטוקן של משוב שטוקן מופיע בו
  - F: כמות המשובים השונים הממוצעת שטוקן מופיע בו.
  - G: התדירות הממוצעת של טוקן במשובים בהם מופיעים (לא מחשבים אפסים בממוצע הזה).
  - Z: כמות סימובלים שונים בטוקן ועוד תו מפריד בין מילים
  - A: כמות ה־ pid הייחודיים.
  - B: אורך pid בבתים (פרקטית שווה 10)
  - C: כמות סימבולים ייחודיים בכל ה־ pid ועוד תו מפריד בין מילים
  - X: ממוצע ה־ id המינימלי של כל pid.
  - Y: ממוצע כמות המשובים לכל pid
- ניתוח:
- באינדקס הreview: כל שורה שוקלת $4+1+4+4+4=17$ בתים, יש N שורות וסך הכל $17N$ בתים.
- באינדקס ה־ pid: שומרים מצביעים לשני מילונים ולכן $O(1)$ בתים.
- באינדקס ה־ token: לכל token ייחודי שומרים $4+4=8$ בתים, יש D טוקנים ייחודיים וסך הכל $8D$ בתים.
- במילון pid: שומרים 4 בתים לפוינטר של pid יחיד ושומרים את כל ה־ pid הייחודיים. עושים הופמן לסטרינג, אז ניקח חסם של התפלגות יוניפורמית על הסימבולים - כלומר כל סימבול מיוצג על־ידי $\lceil \log(C) \rceil$ ביטים, כל מילה מקודדת בצורה רציפה של ביטים ומעוגלת לכמות שלמה של בתים - לכן אורך הסטרינג יהיה

$$\left\lceil \frac{\lceil \log(C) \rceil \cdot B}{8} \right\rceil \cdot 8$$

- סך הכל $\left(4 + \left\lceil \frac{\lceil \log(C) \rceil \cdot B}{8} \right\rceil \cdot 8\right) \cdot A$ בתים.

- במילון pid_id_list: שומרים 4 בתים לפוינטר של רשימה, כל ערך ברשימה זה rev id וכמות id, על מספר מקודד על־ידי גאמא, לכן בממוצע ה־ id מקודד באורך

$$2\lceil \log(X) \rceil + 1$$

ובממוצע הכמות משובים מקודדת באורך

$$2\lceil \log(Y) \rceil + 1$$

רשימות רציפות של מספרים כתובים ברציפות בביטים ומעוגלים לכמות שלמה של בתים, לכן

$$\left\lceil 2\frac{\lceil \log(X) \rceil + \lceil \log(Y) \rceil + 1}{8} \right\rceil \cdot 8$$

יש A רשומות וסך הכל

$$\left(4 + \left\lceil 2\frac{\lceil \log(X) \rceil + \lceil \log(Y) \rceil + 1}{8} \right\rceil \cdot 8\right) \cdot A$$

בתים.

- במילון token: שומרים 4 בתים למצביע של טוקן, יש D טוקנים ייחודיים. בממוצע כל טוקן באורך L ואנחנו משתמשים בקידוד הופמן ולכן הקידוד של הטוקן יהיה בממוצע

$$\left\lceil \frac{\lceil \log(Z) \rceil \cdot L}{8} \right\rceil \cdot 8$$

וסך הכל נקבל:

$$\left(4 + \left\lceil \frac{\lceil \log(Z) \rceil \cdot L}{8} \right\rceil \cdot 8\right) \cdot D$$

בתים.

- במילון token_id_freq_list: שומרים 4 בתים למצביע של רשימה, מקודדים כל רשימה בגאמא באופן רציף, כל id מקודד על־ידי

$$2\lceil \log(H) \rceil + 1$$

וכל freq מקודד על־ידי

$$2\lceil \log(G) \rceil + 1$$

בממוצע יש F רשומות כאלה המעוגלים לכמות שלמה של בתים

$$\left\lceil 2\frac{(\lceil \log(H) \rceil + \lceil \log(G) \rceil + 1)}{8} \cdot F \right\rceil \cdot 8$$

יש רשומה כזו לכל טוקן וסך הכל נקבל

$$\left(4 + \left\lceil 2\frac{(\lceil \log(H) \rceil + \lceil \log(G) \rceil + 1)}{8} \cdot F \right\rceil \cdot 8\right) \cdot D$$

נסכום את הכל ונקבל חסם תיאורטי למקום (כי נתנו חסם להופמן) ועוד $O(1)$.

4 ניתוח תיאורטי של זמן הריצה

- IndexReader(dir): באינדקסים טוענים את הכל, במילונים טוענים רק את הפוינטרים
  - טעינת האינדקס review: $17N$
  - טעינת האינדקס pid: כלום
  - טעינת האינדקס token: $8D$
  - טעינת המילון pid: טוענים $4A$ (רק פוינטרים).
  - טעינת המילון pid_id_list: טוענים $4A$.
  - טעינת המילון token: $4D$.
  - טעינת המילון token_id_freq_list: $4D$
  - טעינת 2 משתנים גלובליים - $O(1)$

סך הכל נקבל

$$17N + 8D + 4A + 4A + 4D + 4D + O(1)$$

- getProductId(int reviewId): ניגשים לאינדקס review בשורה reviewId - 1, בשדה pid מופיע איזה שורה במילון מתאימה, הולכים לשורה הזו במילון, קוראים מהקובץ ומחזירים - סך הכל $(3 + B) \cdot O(1) = O(1)$.
- getReviewScore(int reviewId): ניגשים לאינדקס review בשורה reviewId - 1 ומחזירים את מה שיש בשדה score - סך הכל $O(1)$.

- getReviewHelpfulnessNumerator(int reviewId): ניגשים לאינדקס review בשורה reviewId - 1 ומחזירים את מה שיש בשדה HelpfulnessNumerator - סך הכל $O(1)$.
- getReviewHelpfulnessDenominator(int reviewId): ניגשים לאינדקס review בשורה reviewId - 1 ומחזירים את מה שיש בשדה HelpfulnessDenominator - סך הכל $O(1)$.
- getReviewLength(int reviewId): ניגשים לאינדקס review בשורה reviewId - 1 ומחזירים את מה שיש בשדה ReviewLength - סך הכל $O(1)$.
- getTokenFrequency(String token): עושים חיפוש בינארי במילון token, פענוח ליניארי באורך הקידוד שחסום על־ידי האורך הלא מקודד שהוא הפלט של הקידוד, מוצאים את השורה המתאימה באינדקס token ומחזירים את מה שבשדה Token frequency - סך הכל $\log(D) \cdot O(L) + O(1)$.
- getTokenCollectionFrequency(String token): עושים חיפוש בינארי במילון token (שוב פענוח חסום באורך המילה הלא מקודדת), מוצאים את השורה המתאימה באינדקס token ומחזירים את מה שבשדה Token collection frequency - סך הכל $\log(D) \cdot O(L) + O(1)$.
- getReviewsWithToken(String token): עושים חיפוש בינארי במילון token, מוצאים את השורה המתאימה במילון token_id_freq_list ומחזירים - סך הכל $\log(D) \cdot L + F \cdot O(64)$ (F כפול 32 כי 32 זה אורך הפלט הלא מקודד וקידוד ליניארי באורך הקידוד).
- getNumberOfReviews(): ניגשים למשתנה שנטען באתחול - $O(1)$.
- getTokenSizeOfReviews(): ניגשים למשתנה שנטען באתחול - $O(1)$.
- getProductReviews(String productId): עושים חיפוש בינארי במילון pid, מוצאים את השורה המתאימה במילון pid_id_list ומחזירים - סך הכל $\log(A) \cdot B + Y \cdot O(64)$.