האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

Programming Workshop in C & C++ - 67312

תרגיל בית 4 – שפת C
סמסטר חורף, 2021/22

מועד פרסום התרגיל: תאריך – 18/11/2020
מועד הגשת התרגיל: תאריך – 02/12/2020 בשעה – 23:59

1 רקע

בתרגיל זה תדרשו לעשות שימוש בכלים שרכשתם במהלך הקורס בכדי לממש ספריה שתכיל מבנה נתונים גנרי מסוג סט גיבוב. מבנה זה יעשה שימוש מאחורי הקלעים במבנה נתונים אחר אותו תממשו (צומת אשר תשמור את המידע באותו תא). קראו היטב את ההוראות המופיעות לאורך המסמך ואת ההנחיות. בנוסף, ודאו כי אתם מבינים היטב את הAPIs אותם אתם נדרשים לממש (יופיעו בקבצים שיסופקו לכם עם התרגיל).

2 הגדרות

נגדיר מספר הגדרות הנוגעות לטבלאות גיבוב (לקחו מקורס מבני נתונים):

1. <u>פונקציית גיבוב (hash function)</u> – פונקציה הממפה מידע מגודל כלשהו ("מפתחות") למידע אחר כלשהו, בגודל סופי. לשם פשטות, ניתן להניח שמדובר בפונקציה מהצורה: $h: U \to \{0, \dots, m-1\}$ כאשר U היא קבוצת איברים כלשהי (למשל: מחרוזות, מספרים וכו') ו- $\{0, \dots, m-1\}$ היא קבוצה סופית של תוצאות אותן ניתן לקבל מהפונקציה h. בנוסף, נרצה שכל פונקציית גיבוב h תקיים; h תהיה קלה לחישוב ותמפה כמה שפחות איברים לאותו התא (כדי למנוע התנגשויות).

2. <u>סט גיבוב (HashSet)</u> – מבנה נתונים המכיל מיפוי של ערכים. הערכים יכולים להיות מספרים, מחרוזות או כל טיפוס נתונים נתמך אחר. היתרון של מבנה נתונים זה הוא שבהינתן פונקציית גיבוב "טובה" (כזו שעונה על התנאים לעיל), פעולות ההוספה, החיפוש וההסרה שלו מבוצעות בסיבוכיות זמן ריצה ממוצעת של $\theta(1)$.

3 מימוש HashSet

נציג מספר נושאים נוספים הנוגעים לפונקציות, טבלאות ומפות גיבוב, להם תדרשו במימוש התרגיל:

3.1 גורם עומס (Load Factor)

לבד מגודל הקלט בפועל, הביצועים של מפת הגיבוב מושפעים משני פרמטרים: גורם עומס עליון וגורם עומס תחתון (upper load factor & lower load factor). גורם העומס מוגדר באופן הבא:

$$\text{LoadFactor} = \frac{M}{\text{capacity}} \quad , \quad \text{capacity} > 0 \ \wedge M \geq 0$$

כאשר M מייצג את כמות האיברים שמבנה הנתונים מכיל ברגע נתון (בפועל), ו-capacity מייצג את כמות הנתונים המקסימלית שניתן לשמור במבנה הנתונים (הקיבולת). גורם העומס העליון והתחתון מגדירים כמה ריק או מלא נסכים שמבנה הנתונים יהיה. כלומר, אם נחצה רף מסוים, נגדיל או נקטין את מבנה הנתונים בהתאם, כך שמחד לא נדרוש הרבה זיכרון מיותר ומאידך נוכל להכיל את כמות האיברים שנרצה.

3.2 גודל הטבלה

בתרגיל זה, גודל מבנה הנתונים (ה-capacity) יהיה חזקה של 2 כאשר הגודל ההתחלתי של הטבלה יהיה 16 (מוגדר בקבוע). שימו לב – בהכרח מתקיים $\text{capacity} \geq 1$.

3.3 פונקציית הגיבוב

כאמור לעיל, עלינו לבחור פונקציית גיבוב "טובה" כדי להגיע למבנה נתונים יעיל. אנו נשתמש בפונקציית גיבוב פשוטה מאוד:

$$\mathrm{h(x) = x \bmod capacity} \ , \qquad \mathrm{capacity} \in \mathbb{N} \ \mathrm{s.t\ capacity} \geq 2$$

כאשר הקיבולת זהה לאופן בו הגדרנו אותה קודם לכן. x הוא ייצוג מספרי של הערך שנרצה לשמור בטבלה. כדי להמיר מחרוזות, מספרים, וכיוצא בזה למספר שלם, נספק לכם פונקציות מתאימות (ראו API למטה). נשים לב שאופרטור מודולו (%) לא עובד כמו האחד המתמטי. למשל, $(-3)\%7 = -3$ אם נחשב בעזרת קוד, אך המודולו המתמטי נותן ערך 4. מסיבה זו ומסיבות נוספות, נחשב את המודולו באופן הבא: $\mathbf{v \bmod capacity = v \ \&\ (capacity - 1)}$ (שימו לב שזה אופרטור and לוגי!).

פתרון זה עדיף, כיוון שאופרטורים לוגיים מהירים יותר מאריתמטיים.

3.4 אלגוריתמי מיפוי

בהמשך לאמור, ניתן לשער שהפונקציה שהגדרנו קודם לכן תגיע במהרה להתנגשויות. כלומר, אנו עלולים להיתקל בשני ערכים x, y שונים זה מזה אך יתקיים $h(x) = h(y)$, וזה שקול כמובן להתנגשות. ישנן שתי שיטות נפוצות לפתרון התנגשויות:

1. <u>מיפוי פתוח (Open Hashing)</u> – שיטה המאפשרת לשמור יותר מערך אחד בכל תא. התאים, הנקראים גם "סלים" (או buckets) ובנויים מטיפוס נתונים אחר (למשל מערך דינמי או רשימה מקושרת). כך, גם במקרה של התנגשות, כל שקורה הוא שהאיבר המתנגש נוסף לרשימה המקושרת.
2. <u>מיפוי סגור (Close Hashing)</u> – שיטה לפיה כל תא יכול להכיל רק איבר אחד. במקרים אלו, עלינו למצוא דרך אחרת להתמודד עם התנגשויות ולמפות איברים.

בתרגיל זה, נשתמש בשיטת מיפוי סגור.

3.5 חישוב מפתח הגיבוב וטיפול בהתנגשויות

כפי שציינו קודם, בתרגיל זה נשתמש בגיבוב סגור, נתונה לכם בתוך הסטראקט של ההאשסט פונקציית גיבוב (אשר אותה תקבלו בתוך האובייקט של ההאשסט). נטפל בהתנגשויות (אם התא מכיל כבר ערך אחר) בשיטת Quadratic probing . החישוב יעשה לפי הנוסחה הבאה:

$$hash(e) \ + \ (i + (i * i))/2 \ mod \ capacity$$

כאשר i מייצג את מספר הנסיון בו ניסינו לגבב את הערך החדש שאנו מוסיפים אל טבלת הגיבוב (הערך הראשון של i יהיה כמובן 0). הערך שיתקבל מחישוב הנוסחה יהיה תמיד שלם (אפשר להוכיח זאת בקלות), תנסו עד capacity פעמים (אזי נעבור לכל היותר על כל הטבלה) , ותהיו אמורים למצוא מקום (היות והנוסחא תתן לנו אינדקסים שונים שיאפשרו לנו לבדוק במקומות שונים וגם כי מקדם העומס לא אמור להגיע ל1).

4 הספרייה HashSet

כאמור, בתרגיל זה נממש את הספרייה הHashSet (סט גיבוב). הספרייה תממש סט גיבוב המבוססת שיטת מיפוי סגור. בצורה סכמתית ניתן לחשוב על זה באופן הבא: כל מפתח במפת הגיבוב ממופה לצומת.

4.1 הספרייה Node

לחלק זה מצורף הקובץ Node.h הכולל את חתימות הפונקציות אותן תדרשו לממש בחלק זה.

שימו לב – מימוש חלק זה קריטי להצלחת מימוש התרגיל כולו. בנוסף, הAPI של חלק זה נבדק בנפרד וגם יחד עם API של הHashSet.

בנוסף, הAPI הנ"ל הינו דרישה אך כמובן שניתן להוסיף פונקציות עזר כראות עיניכם ולפי צרככם (בקובץ הc).

4.2 הספרייה HashMap

לחלק זה כמו לקודם מצורף קובץ HashSet.h הכולל את חתימות הפונקציות אותן תדרשו לממש בחלק זה ואת הקבועים שיסייעו לכם כמו גודל התחלתי ופרמטר הגדילה..

חלק זה הינו החלק העיקרי ועליו יעשו מרבית הבדיקות.

שימו לב:

1. הספרייה חייבת להיות **גנרית** (ומכאן, גם הNode חייב להיות גנרי). ספרייה שתיכתב באופן שאינו גנרי תפסיד נקודות רבות ללא אפשרות לערעור, שכן לא תעברו טסטים (אין ביכולתכם לדעת מה נבחר להכניס לValues).
2. הנכם חייבים להשתמש בספריית הNode מסעיף 4.1. בנוסף, בכל סיטואציה בה תוכלו להשתמש באחת הפונקציות מהAPI של הNode ותבחרו שלא לעשות כך, תוכלו להפסיד נקודות. אחת ממטרות הAPI היא לאפשר כתיבת קוד נקי וברור.

4.3 הקובץ Hash

בקובץ הנ"ל (Hash.h) תקבלו מימוש של מספר פונקציות גיבוב (פשוטות מאוד) לדוגמא בהן תוכלו להשתמש לצורך בדיקות.

4.4 הקובץ Utils

בקובץ הנ"ל (Utils.h) תקבלו מימוש של מספר פונקציות (פשוטות מאוד) לדוגמא בהן תוכלו להשתמש לצורך בדיקות.

5 הרצת התוכנית

את התוכנית שלכם נקמפל יחד עם קובץ main.c חיצוני שאינכם ראיתם. הקובץ ישתמש בפונקציות שונות מתוך הAPI אותו כתבתם.

6 הגשת התרגיל

עליכם להגיש בגיט (כמו בתרגילים הקודמים עד עכשיו) את הקבצים:
Node.c
HashSet.c

לא להגיש את קבצי הh (לא נהיה סלחניים לגבי זה).

7 הנחות מקלות

-אתם יכולים להניח שכל הקצאת זיכרון שאתם עושים תצליח

-אינכם יכולים להניח על הקלט בפונקציות שום דבר אלא אם כן מצוין אחרת בתיעוד בקובץ h

-עליכם לכתוב את הקוד בזהירות רבה, התרגיל קל יחסית ולכן אנו נבדוק את הקוד בדקדקנות.

-גבולות מקדם העומס שניתן לכם בקובץ הקבועים יהיו תמיד גדולים מ0 וקטנים מ1 ומקדם העומס המקסימלי יהיה גדול ממש ממקדם העומס המינימאלי.

-על התכנית שלכם להיות fool proof אזי כי עליכם להזהר ממתכנת זדוני שירצה להפיל לכם את הקוד בטסטים כלשהם לדוגמא על ידי הכנסת סטראקטים שבתוכם יש שדות שמכילים NULL.

8 פיתרון בית הספר

- פיתרון בית הספר המקופל זמין לשימושכם בנתיב:

~labcc/www/ex4

9 הערות ודגשים

- אנא וודאו כי התרגיל שלכם עובר את סקריפט ה-pre-submission ללא שגיאות או אזהרות. הסקריפט זמין בנתיב:

~labcc/presubmit/ex4/run <path_to_submission>

- עליכם לבדוק כי התוכנית מתקמפלת ללא הערות על מחשבי בית ספר לפי הפקודה הבאה:

gcc -Wall -Wextra -Wvla -Werror -g -lm -std=c99 <code_files> <main_file> -o hash_set_prog

- כחלק מהבדיקה האוטומטית תבדקו על סגנון כתיבת קוד. אנא ודאו כי הנכם מבינים את דרישות הcoding style של הקורס במלואן.

- הקפידו על בדיקות של דליפות זיכרון ושאר השגיאות אותן בודק ה-**valgrind**. המשקל אשר יינתן לבדיקות אלו יהיה רב ולכן וודאו זאת היטב.
למען הסר ספק, הvalgrind אמור לרוץ באופן תקין מלא-מלא, כלומר שום שגיאה או הערה מכל סוג שהוא. אם מופיעה הערה כלשהי, גם אם אינה קשורה לדליפת זיכרון באופן ישיר, היא נחשבת כשגיאה ותהיה על כך הורדת נקודות.

- תיעוד – הקפידו על תיעוד הולם של הקוד אותו אתם כותבים.

- נראות הקוד – הקפידו על כל הדגשים התכנותיים – אורך פונקציות, שמות משתנים ושמות פונקציות אינפורמטיביים וכו'.
תהיה בדיקה ידנית ואנחנו נתייחס לדגשים אלה בקפדנות.

בהצלחה!

"I have yet to see a language that comes even close to C"

(Linus Torvalds)