

סדנאות תכנות בשפת C ו-C++ (67315) C++ - תרגיל 6

תאריך ההגשה של התרגיל: 22 ביוני, 2022, בשעה 22:00

נושאי התרגיל: `templates`, C++ standard library, bitwise operators, inheritance, exceptions.

1 רקע

בתרגיל זה נשתמש בידע שצברנו במהלך הקורס ונשלבו בנושאים כדוגמת גנריות ושימוש בספריה הסטנדרטית של C++, לצורך מימוש מבנה הנתונים `HashMap`¹ ושימוש בו. בחלק הראשון של התרגיל (הסבר מפורט בהמשך), נממש את מבנה הנתונים `HashMap`, בעוד שבחלק השני - ניצור מחלקת `Dictionary` (מילון) אשר תירש מ-`HashMap` כך שהמפתחות והערכים הם `string`.

2 הגדרות

להלן מספר הגדרות בסיסיות הנוגעות לטבלאות גיבוב:

- **הגדרה:** פונקציית גיבוב (hash function) היא פונקציה הממפה מידע ממרחב כלשהוא ("מרחב המפתחות") למידע ממרחב אחר, בגודל סופי. לשם הפשטות, נוכל להניח שמדובר בפונקציה מהצורה $h : U \rightarrow \{0, \dots, m-1\}$ כאשר U היא קבוצת איברים כל-שהיא, כמו למשל מחרוזות, מספרים וכדומה, ו- $\{0, \dots, m-1\}$ היא קבוצה סופית של תוצאות אותן ניתן לקבל מהפונקציה (תוצאת פעולת ה-hash).

- **מסקנה מההגדרה הקודמת:** נרצה שכל פונקציית hash, שנסמנה h , תקיים:

1. h תהיה קלה לחישוב,

2. ש- h תמפה כמה שפחות איברים לאותו התא, כדי למנוע התנגשויות רבות.

- **הגדרה:** hash map הוא מבנה נתונים המכיל מיפוי של מפתחות לערכים. המפתחות יכולים להיות מספרים, מחרוזות או כל טיפוס נתונים נתמך אחר - וכך גם הערכים. ברמה האינטרנית, טיפוס נתונים זה עושה שימוש בפונקציית גיבוב כדי **למפות** מפתחות לערכים. הייתרון של מבנה נתונים זה, הוא שבהינתן פונקציית גיבוב "טובה", פעולות ההוספה, החיפוש וההסרה שלו מבוצעות בסיבוכיות זמן ריצה ממוצעת של $\Theta(1)$ (עוד על כך ילמד בקורס מבני נתונים).

¹טבלאת גיבוב - ניתן לקרוא עוד כאן: https://en.wikipedia.org/wiki/Hash_table

3 מימוש HashMap

משהוצגו ההגדרות המקדימות הנדרשות לפתרון התרגיל, נציג להלן מספר נושאים נוספים הנוגעים לפונקציות, טבלאות ומפות גיבוב, להם אתם נדרשים במימוש התרגיל:

3.1 גורם העומס (Load Factor)

לבד מגודל הטבלה בפועל, הביצועים של מפת הגיבוב מושפעים משני פרמטרים: גורם עומס העליון וגורם העומס התחתון (upper load factor & lower load factor). גורם העומס מוגדר כך:

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

כאשר $M \in \mathbb{N} \cup \{0\}$ מייצג את כמות האיברים שמבנה הנתונים מכיל ברגע, בעוד ש- $\text{capacity} \in \mathbb{N}$ מייצג את כמות האיברים המקסימלית שניתן לשמור במבנה הנתונים (כלומר, הקיבולת). אם כך, מהם גורמי העומס התחתון והעליון? אלו הגורמים שמציינים עד כמה נסכים שמבנה הנתונים יהיה ריק או מלא. כלומר, נרצה שכאשר נחצה רף מסוים (threshold) - נגדיל או נקטין את הטבלה בהתאם, כך שמצד אחד לא תדרוש זיכרון רב מדי ומהצד השני תוכל להכיל כמה איברים שנרצה. לשם כך, ברגע שנחצה (ממש) את אותו רף (תחתון או עליון) - נבצע הליך שנקרא re-hashing ובו נגדיל או נקטין את הטבלה, נחשב את ערכי ה-hash בשנית ונמקם שוב את כל האיברים שבטבלה. במילים אחרות, נעתיק את כל הערכים הישנים לטבלה "חדשה". **כברירת מחדל, נגדיר את גורם העומס התחתון להיות $\frac{1}{4}$ ואת גורם העומס העליון להיות $\frac{3}{4}$ (ז"א שכאשר $\frac{M}{\text{capacity}} < \frac{1}{4}$ נקטין את הטבלה ונמפה מחדש את כל הערכים, וכאשר $\frac{M}{\text{capacity}} > \frac{3}{4}$ נגדיל את הטבלה ונמפה מחדש את כל הערכים). בכל מקרה, גורם העומס העליון יהיה בהכרח גדול מהתחתון, ושניהם יהיו בקטע $(0, 1]$.**

3.2 גודל הטבלה

ישנן שתי אפשרויות פופולריות לקביעת הגודל המקסימלי (capacity) של טבלאות גיבוב ברגע נתון: שימוש במספרים ראשוניים או בחזקות של 2. האופציה הראשונה, דהיינו מספרים ראשוניים, טובה מכיוון שפיזור האיברים נעשה בצורה הרבה יותר אחידה. מנגד, הקושי שבבחירה זו הוא שאין דרך קלה לבצע re-hash - כלומר לא נוכל למשל, לבצע חישוב אריתמטי פשוט כמו העלאה בחזקה, כדי לקבל את גודל הטבלה החדש. מנגד, בעוד שהאופציה השנייה - דהיינו שימוש בחזקות של 2 - אינה יוצרת פיזור אחיד טוב מספיק, מאוד קל לבצע בעניינה re-hashing וכן ניתן לחשב בה את המיקום של כל איבר בדרך מהירה יותר, על ידי bitwise operators (נראה זאת בהמשך). **לפיכך, בתרגיל זה, גודל הטבלה יהיה חזקה של 2, כאשר הגודל ההתחלתי הוא 16. שימו לב שבהכרח $\text{capacity} \geq 1$.**

3.3 פונקצית הגיבוב

כאמור לעיל, עלינו לבחור פונקצית גיבוב "טובה"² כדי להגיע למבנה נתונים שפועל ביעילות טובה. פונקצית הגיבוב שנבחר מאוד בסיסית, והיא:

$$h(x) = x \bmod \text{capacity}, \quad \text{capacity} \in \mathbb{N}$$

כאשר capacity מייצג את גודל הטבלה ו- x הוא ייצוג מספרי של הערך שנרצה לשמור בטבלה. **כדי להמיר מחרוזת, מספרים וכיוצ"ב למספר שלם, תוכלו להשתמש בפונקציה `std::hash`.³ ניתן להניח שכל טיפוס שתדרשו להכיל במפה, יתמוך ב-`std::hash`. נשים ♥ שאופרטור ה- modulo ב- C++ אינו פועל באופן זהה לאופרטור ה- modulo המתמטי (כפי שראיתם בתרגיל 1). למשל, בעוד שב- C++ $-3 \bmod 7 = -3$, התוצאה המתמטית היא כידוע 4. כדי לפתור זאת, יהיה עלינו לחשב את הפונקציה ב- C++**

²פונקציית גיבוב תחשב "טובה" אם לכל 2 ערכים שונים $a, b \in U$ הפונקציה תמפה את a, b לאותו הערך בהסתברות נמוכה מאוד (ככול שההסתברות נמוכה יותר כך פונקציית הגיבוב טובה יותר)

³ניתן לקרוא עוד כאן: <https://www.cplusplus.com/reference/functional/hash>

כערך מוחלט. יתרה מכך, אנו עשויים להיתקל בקושי כאשר נחשב את המודולו של INT_MIN , כיוון שאין לו ערך מקסימלי תואם. נפתור זאת על ידי casting ל- long . כלומר $(\% \text{ modulo ב- } C++)$:

$$v \bmod capacity = |(long)v \% capacity|$$

לסיים, נציע חלופה אחרת לחישוב פונקציית הגיבוב: נבחין שפונקציית הגיבוב שהגדרנו עושה שימוש ב- $capacity$, שכאמור לעיל הוא חזקה של 2. לכן, נוכל להשתמש באופרטור הלוגי $\&$ (מיוצג על ידי $\&$) כדי לחשב את אותו הערך, כלומר:

$$v \bmod capacity = v \& (capacity - 1)$$

פתרון זה עדיף, כיוון שאופרטורים לוגיים מהירים יותר מאריתמטיים - אז חישוב האינדקס יהיה מהיר יותר. בנוסף, גם לא נידקק יותר לשימוש בערך מוחלט או ל- long casting.

3.4 אלגוריתם המיפוי - שיטת מיפוי פתוח (Open Hashing).

בהמשך לאמור, קל לראות שפונקציית הגיבוב שנבחרה, $h(x)$, תיצור, במוקדם או במאוחר, התנגשויות. ראשית, אם כמות האיברים שנרצה לשמור במבנה הנתונים תהיה גדולה מ- $capacity$, זה ינבע ישירות מעיקרון שובך היונים. אחרת, גם כאשר $capacity$ גדול מכמות האיברים שנרצה לשמור, אנו עשויים להיתקל במקרים בהם לשני ערכים, x, y *s.t.* $x \neq y$, נקבל ש- $h(x) = h(y)$ ובמקרה זה ניתקל בהתנגשויות. ישנן שתי שיטות לפתרון התנגשויות:

- **Open hashing:** שיטה המאפשרת לשמור יותר מערך אחד בכל תא. התאים, שנקראים "סלים" (buckets) ובנויים מטיפוס נתונים אחר - לדוגמה מרשימה מקושרת. כך, גם אם יש התנגשויות, כל שקורה הוא שהאיבר המתנגש נוסף לרשימה המקושרת.
- **Close hashing:** שיטה לפיה כל תא יכול להכיל רק איבר אחד. במקרים אלו, עלינו למצוא דרך אחרת להתמודד עם התנגשויות ולמפות איברים.⁴

בתרגיל זה, נממש את ה- hash map באמצעות Open hashing.

⁴שיטה מוכרת למימוש closed hashing היא quadratic probing. להרחבה ראו: https://en.wikipedia.org/wiki/Quadratic_probing.

4 חלק א' - המחלקה HashMap

בחלק הראשון נממש את המחלקה הגנרית HashMap, שתייצג מיפוי בין מפתחות לערכים ($key \mapsto value$), המבוסס על טבלת גיבוב. כלומר, מבנה הנתונים ימפה בין מפתחות, מסוג KeyT, לערכים, מסוג ValueT. יש לתמוך ב-API⁵ הבא:

הערות	התיאור	
פעולות מחזור החיים של האובייקט		
	בנאי שמאתחל HashMap ריק.	בנאי ברירת מחדל
יש לוודא שהוקטורים באותו הגודל. המיפוי יהיה $\forall 0 \leq i < n \text{ keys}[i] \mapsto \text{values}[i]$	בנאי המקבל שני וקטורים, אחד שמכיל ערכי KeyT ואחד שמכיל ערכי ValueT ושומר את הערכים במפה.	בנאי 1
	מימוש של בנאי העתקה.	בנאי העתקה
	מימוש destructor.	destructor
פעולות (מטודות)		
המספר שמחזור מטיפוס int.	הפעולה מחזירה את כמות איברי המפה.	size
המספר שמחזור מטיפוס int.	פעולה המחזירה את קיבולת המפה.	capacity
הפעולה תחזיר bool.	פעולה הבודקת האם המפה ריקה.	empty
הפעולה תחזיר אמת אם המפתח איננו קיים ונוסף בהצלחה.	פעולה המקבלת מפתח וערך, ושומרת את המיפוי שהתקבל.	insert
הפעולה מחזירה bool.	הפעולה מקבלת מפתח ובודקת האם הוא קיים במפה.	contains_key
הפעולה תזרוק חריגה במקרה שה-key לא נמצא.	פעולה מקבלת key ומחזירה את ה-value המשוויך אליו.	at
הפעולה תחזיר אמת אם הערך הוסר בהצלחה.	הפעולה מקבלת מפתח ומסירה את הערך המשוויך לו מהמפה.	erase
המספר שמחזור מטיפוס double.	פעולה המחזירה את גורם העומס.	get_load_factor
הפעולה תחזיר int. אם המפתח לא נמצא - יש לזרוק חריגה.	תקבל מפתח ותחזיר את גודל הסל.	bucket_size
הפעולה תחזיר int. אם המפתח לא נמצא - יש לזרוק חריגה.	תקבל מפתח ותחזיר אינדקס הסל.	bucket_index
ה- capacity לא משתנה.	פעולה המסירה את כל איברי המפה.	clear
עליכם לממש const forward iterator בלבד. ה-iterator יחזיר std::pair<KeyT, ValueT>. ⁶	מימוש מחלקת iterator ובנוסף כל הפעולות הנדרשות ל-iterator (לרבות typedefs), בהתאם לשמות הסטנדרטים של C++.	iterator
אופרטורים		
השמה לכל ערכי האובייקט.	תמיכה באופרטור ההשמה (=).	השמה
האופרטור יקבל מפתח ויחזיר את הערך המשוויך לו. אין לזרוק חריגה במקרה זה.	תמיכה באופרטור [].	subscript
בדיקה האם שני סטים מכילים איברים זהים.	תמיכה באופרטורים ==, !=.	השוואה

⁵ראשי תיבות של: Application Programming Interface
⁶ניתן לקרוא עוד כאן: <https://www.cplusplus.com/reference/utility/pair>

דגשים, הבהרות, הנחיות והנחות כלליות:

- את המחלקה עליכם להגדיר בקובץ `HashMap.hpp` (למה לא נוכל להגדירה בקובץ `cpp`? כי לא ניתן לכלול בקובץ שעובר קומפילציה ישירה מימוש של `templates`).
- כאמור, על המחלקה להיות גנרית. הערך הגנרי הראשון שהמחלקה תקבל הוא טיפוס הנתונים שמייצג את המפתחות, אליו התייחסנו בשם `KeyT`. הפרמטר השני הגנרי שהמחלקה תקבל הוא סוג הנתונים המייצג את הערכים אליהם המפתחות ממפים, נסמנו כ- `ValueT`. **ניתן להניח** כי `KeyT` נתמך על ידי `std::hash`, וכי `KeyT` ו- `ValueT` תומכים ב- `operator=`, `operator==` וכן כי יש לשניהם בנאי דיפולטיבי. שימו לב שניתן להיעזר ב- `T()` כדי לקבל את הערך הדיפולטיבי של `T`.
- **הנכם מחוייבים לשמור את הסלים בתור מערך שמוקצה דינמית** (כל תא במערך הוא סל). את הסלים עצמם ניתן לממש בעזרת מבנה נתונים של STL (למשל `Vector`).⁷
- **דרישות זמן ריצה:** על הפעולות `insert`, `at`, `contains_key`, `erase` (וכמובן `operator[]`) לפעול בסיבוכיות לינארית ביחס לסל שבו האיבר נמצא - כלומר ב- $O(n)$ כאשר n הוא גודל הסל.
- **למותר לציין, אבל יצויין בכל זאת, שאין להשתמש במחלקות של STL באופן שייתר את פתרון התרגיל. למשל, אין לעשות שימוש ב- `std::unordered_map` במקום לממש מפת גיבוב באופן עצמאי. מנגד, אתם בהחלט רשאים (ומצופה מכם) לעשות שימוש ב- STL.**
- **אין להשתמש במצביעים חכמים (Smart pointers) בתרגיל הזה**
- **שימו לב:** ה-API הנ"ל מציג לכם את שמות הפונקציות המחייבות, הפרמטרים, ערכי החזרה וטיפוסיהם. בעת מימוש ה-API, עליכם ליישם את העקרונות שנלמדו בקורס באשר לערכים קבועים (`constants`) ומשתני ייחוס (`references`). **שימוש בקונבנציות אלו הוא חלק אינטגרלי מהתרגיל, עליו אתם מקבלים ניקוד.** עיקרון זה נכון בפרט גם לגבי מימוש ה- `iterator`.

דגשים לגבי מתודות ספציפיות:

- **בנאי 1:** לא ניתן להניח שהוקטורים שיתקבלו יהיו בגודל זהה. אם `keys.size() != values.size()`, אזי יש לזרוק חריגה. כמו כן, אם יש ערכי מפתחות כפולים - אזי עליכם לדרוס את הערכים הישנים עם החדשים (יש להכניס את האיברים למפה לפי סדר מתחילת 2 הווקטורים ועד לסופם).
- `bucket_size` ו- `bucket_index` : יש לזרוק חריגה אם המפתח לא קיים.
- `at` ו- `operator[]` : שימו לב להבדלים שבין `at` ובין `operator[]`, כפי שכבר נידונו בהרצאות ובתרגולים:
- **קריאה:** בעוד שב- `at` תיזרק חריגה כאשר ניגש לאיבר שאינו קיים, כשמדובר על `operator[]` ההתנהגות אינה מוגדרת ותלויה בכם (`no-throw guarantee`).⁸
- **כתיבה:** במקרה שבו פונים ב- `HashMap::operator[]` **לאיבר שלא קיים, עליכם ליצור איבר חדש בטבלה.** גישה זו תאפשר לנו להשתמש בביטויים כמו `map["foo"] = "bar"`.
- **מימוש `iterator`:** שימו לב כי מימוש ה- `iterator` יחייב אתכם לממש מחלקת `iterator`, ולא ניתן להפנות לפעולות `iterator` של STL. עליכם לממש את מחלקת ה- `iterator` כמחלקה פנימית (`nested class`) של `HashMap`. יש לממש אך ורק `const forward iterator` כלומר לא ניתן בעזרת האיטרטור לשנות את תוכן המפה וניתן לקדם את האיטרטור קדימה. זכרו לממש

⁷ניתן לחשוב שאפשר להשתמש בטיפוס נתונים, כמו למשל `vector`, כדי "להחזיק" את הסלים עצמם. גישה זו אינה מדויקת שכן אין לנו שליטה על כמות הזיכרון שמערכת ההפעלה תקצה ל- `vector`. למשל, נניח שנשתמש ב- `constructor` של `vector` שמקבל `capacity`. שימוש זה יבטיח כי הווקטור יוכל להכיל כמות **מינימלית** של איברים בטרם יעבור `resize`. אלא - ופה העיקר - **שימוש זה אינו כופה על STL שלא להקצות כמות זיכרון גדולה יותר**, אם הספריה "סבורה" שכך נכון לעשות. מכאן באה הדרישה האמורה.

⁸התנהגות זו תואמת להתנהגות של `std::unordered_map`. ראו למשל: https://en.cppreference.com/w/cpp/container/unordered_map/operator_at. מוחזר מקום שלא מוגדר בזיכרון. למעשה, כך עובדים גם טיפוסי נתונים אחרים ב-STL. למשל ב- `std::vector` מוחזר מקום שלא מוגדר בזיכרון. כדי להבין מה המשמעות של "התנהגות לא מוגדרת" מומלץ לנסות לבחון את התנהגות `std::unordered_map`.

גם את `begin` ו-`end` ב-`HashMap`. בפרט - חשבו איזה וריאציות צריך? האם נצטרך את `cbegin` ו-`cend`? מצד שני האם נצטרך את `begin` ו-`end`?⁹

5 חלק ב' - המחלקה Dictionary

בחלק זה של התרגיל תצטרכו לממש מחלקת מילון אשר יורשת מהמחלקה `HashMap` בצורה ציבורית ומבצעת ספסיפיקציה כך שגם `KeyT` וגם `ValueT` הם מהטיפוס `std::string`. מחלקת המילון אמורה להרחיב ולשנות את ההתנהגות של המחלקה `HashMap` בעזרת הפעולות הבאות (מטודות): `update` ו-`erase` (הסבר מפורט על אופן הפעולה של כל אחת מהן בהמשך).

5.1 הפעולה erase

הפעולה מקבלת מפתח ומסירה את הערך המשוויך לו מהמפה. הפעולה מרחיבה את המחלקה `HashMap` בכך שאם המפתח לא קיים במילון עליכם לזרוק שגיאה (שימו לב שהתנהגות זו שונה מההתנהגות הסטנדרטית של STL).
הערות והנחיות:

- חישבו איזה שינוי אתם נדרשים לעשות במחלקה `HashMap` כדי שתוכלו לדרוס את הפעולה `erase`.
- את השגיאה שעליכם לזרוק אתם נדרשים לממש בעזרת מחלקה היורשת מ-`std::invalid_argument`¹⁰ ותקרא: `InvalidKey`.

5.2 הפעולה update

הפעולה תקבל שני איטרטורים ותכניס את כל איברי האיטרטור אל המילון. הפעולה לא תחזיר דבר.
הערות והנחיות:

- האיטרטור שיתקבל יהיה לכל הפחות `forward iterator`. הפעולה תקבל 2 איטרטורים, איטרטור התחלה ואיטרטור סיום.
- ניתן להניח כי האיטרטור יכיל איברים מסוג `std::pair<std::string, std::string>` כאשר האיבר הראשון הוא המפתח והאיבר השני הוא הערך.
- לא ניתן להניח שיש ערכים באיטרטור (יכול להתקבל איטרטור ריק), במקרה זה לא יתווסף שום ערך למילון.
- אם יש ערכי מפתחות כפולים עליכם לדרוס את הערכים הישנים עם החדשים.

⁹רמז (למי שהגדיל וקרא את הערת השוליים ©): בנוסף לסוג האיטרטור שביקשנו שתממשו, שימו לב איזה פעולות C++ מחייבת כדי לבצע "איטרציה for-each" (כלומר `for (auto it : map)`).
¹⁰ניתן לקרוא על המחלקה כאן: https://en.cppreference.com/w/cpp/error/invalid_argument

6 נהלי הגשה

- קראו בקפידה את הוראות תרגיל זה ואת ההנחיות להגשת תרגילים שבאתר הקורס.
- זכרו שבתרגיל זה עליכם לקמפל את התוכנית כנגד מהדר לשפת C++ בתקן שנקבע בקורס. כמו כן, זכרו שעליכם **לתעדף** פונקציות ותכונות של C++ על פני אלו של C. למשל, נעדיף להשתמש ב- new ו- delete על פני malloc ו- free, וכן נעדיף להשתמש ב- std::string מאשר ב- char*.
- כמו בתרגילים קודמים, עליכם להגיש את התרגיל דרך הגיט בעזרת הפקודה: git submit. עליכם להגיש **אך ורק את הקבצים**: HashMap.hpp, Dictionary.hpp שימו לב שניתן ואף רצוי להגיש מספר פעמים עד למועד ההגשה ורק ההגשה האחרונה היא הקובעת.
- **שימו לב:** קבצי קוד המקור שתכתבו נדרשים להתקמפל כהלכה עם std=c++14, כנדרש בהוראות להגשת תרגילים שפורסמו באתר הקורס.
- **נזכיר:** כאמור בהנחיות הכלליות להגשת תרגילים - הקצאת זיכרון דינמית מחייבת את שחרור הזיכרון, למעט במקרים בהם ישנה שגיאה המחייבת סגירת התוכנית באופן מיידי עם קוד שגיאה (כלומר קוד יציאה השונה מ- 0). תוכלו להיעזר בתוכנה valgrind כדי לחפש דליפות זיכרון בתוכנית שכתבתם.
- אנא וודאו כי התרגיל שלכם עובר את ה- Pre-submission Script **ללא שגיאות או אזהרות**. קובץ ה- Pre-submission Script זמין להרצה באופן הבא:
ראשית עליכם ליצור קובץ tar הכולל את הקבצים HashMap.hpp, Dictionary.hpp בלבד. ניתן ליצור קובץ tar כדורש על ידי הפקודה:

```
tar -cvf ex6.tar HashMap.hpp Dictionary.hpp
```

שנית להריץ את ה- Pre-submission Script באופן הבא:

```
~labcc2/presubmit/ex6/run <path/to/your/ex6.tar>
```

בהצלחה!!