

תרגיל בית תכנותי להגשה עד 24.01.19 בשעה 23:50 בהצלחה!

תרגיל זה מנוסח בלשון זכר מטעמי נוחות בלבד והוא מיועד לכל המגדרים.
מתרגל אחראי על התרגיל: שמעון

הוראות:

1. יש להגיש את כל קבצי ה header וה cpp שיצרתם בקובץ zip יחיד בעל השם EXP_ID1_ID2 כאשר ID1 ו ID2 הם מספרי תעודות הזהות של שני בני הזוג. אין צורך להגיש קבצים שסופקו ע"י צוות הקורס.
2. ההגשה תתבצע רק ע"י אחד מבני הזוג.
3. עליכם לוודא לפני ההגשה כי הקוד שלכם מתקמפל ורץ בשרת ה t2 (הוראות מצורפות בקובץ נפרד).
4. זוג שהתרגיל שלו לא יתקמפל בשרת ה t2 או יעוף בזמן ריצה ציונו בתרגיל יהיה 0.
5. שימו לב כי יש לשחרר כל זיכרון שהקצתם. דליפת זיכרון תגרור הורדה בציון.
6. יש לכתוב קוד קריא ומסודר עם שמות משמעותיים למשתנים, למתודות ולמחלקות.
7. יש להקפיד למלא את הדרישות בתרגיל (שימוש בייצוג הנכון, הוספת include רק עבור קבצי מקור שבמפורש הוגדר שניתן להשתמש בהם, סיבוכיות זמן וכו') אי עמידה בדרישות התרגיל תגרור ציון 0.

בתרגיל בית זה אתם מתבקשים לממש בשפת ++c מבנה נתונים דינמי המאפשר שמירה של מפתחות ומידע המשויך למפתח שיקרא LCRS_BalancedTree. מבנה נתונים זה הוא עץ 2-3 בייצוג של left-child-right-sibling.

למבנה הנתונים המתודות הפומביות הבאות:

- **LCRS_BalancedTree (const Key* min ,const Key* max)** – בניית מבנה נתונים חדש כאשר min הוא פוינטר לאובייקט מסוג Key עם המפתח המינימלי האפשרי ו max הוא פוינטר לאובייקט מסוג Key עם המפתח המקסימלי האפשרי. הפונקציה מעתיקה את המפתחות המוצבעים ע"י min ו max ומאתחלת את מבנה הנתונים. סיבוכיות נדרשת – $O(1)$.
- **~ LCRS_BalancedTree ()** – destructor, הפונקציה מוחקת את מבנה הנתונים ואת כל המידע השמור בו. סיבוכיות נדרשת – $O(n)$.
- **void Insert(const Key* nkey, const Value* nval)** – הפונקציה מעתיקה את המפתח ש nkey מצביע עליו ואת המידע המשויך לו ש nval מצביע עליו ומוסיפה למבנה הנתונים את המפתח ואת המידע. סיבוכיות נדרשת – $O(\log n)$.
- **void Delete(const Key* dkey)** – הפונקציה מוחקת ממבנה הנתונים את האובייקט השמור אצלה בעל מפתח השווה למפתח ש dkey מצביע עליו ואת המידע המשויך אליו אם קיים מפתח כזה במבנה הנתונים, אחרת לא עושה כלום. סיבוכיות נדרשת – $O(\log n)$.

- **Value* Search(const Key* key) const** – הפונקציה מחזירה פוינטר למידע המשוך למפתח השווה למפתח ש key מצביע עליו אם קיים מפתח כזה במבנה הנתונים אחרת מחזירה NULL, ללא שינוי של מבנה הנתונים. סיבוכיות נדרשת - $O(\log n)$.
- **unsigned Rank(const Key* key) const** – אם קיים מפתח השווה למפתח ש key מצביע עליו במבנה הנתונים, הפונקציה מחזירה את המיקום של המפתח בסידור הלינארי (בסדר עולה) של המפתחות השמורים במבנה הנתונים אחרת הפונקציה מחזירה 0, ללא שינוי של מבנה הנתונים. סיבוכיות נדרשת - $O(\log n)$.
- **const Key* Select(unsigned index) const** – הפונקציה מחזירה מצביע למפתח במיקום index בסידור הלינארי (בסדר עולה) של המפתחות השמורים במבנה הנתונים. אם לא קיים מפתח במיקום זה הפונקציה מחזירה NULL, ללא שינוי של מבנה הנתונים. סיבוכיות נדרשת - $O(\log n)$.
- **void Print_Values(const Key* key1, const Key* key2) const** – הפונקציה מדפיסה את המידע המשוך למפתחות בטווח [key1, key2] ע"י הפעלת המתודה print() במחלקה Value על כל מידע משוך למפתחות בטווח הנ"ל המאוחסנים במבנה הנתונים. את ההדפסה יש לעשות לפי סדר המפתחות מהקטן לגדול כלומר, להדפיס קודם את המידע המשוך למפתח הכי קטן בטווח [key1, key2] המאוחסן במבנה הנתונים לאחר מכן להדפיס את המידע המשוך למפתח השני הכי קטן בטווח [key1, key2] המאוחסן במבנה הנתונים וכן הלאה. שימו לב כי ערך המפתחות key1 ו key2 לאו דווקא נמצאים במבנה הנתונים וכי זהו טווח סגור כלומר, יש להדפיס גם ערכי מפתחות ששווים ל key1 או key2. נסמן ב k את מספר המפתחות המאוחסנים במבנה הנתונים בטווח [key1, key2]. סיבוכיות נדרשת $O(\log(n) + k)$.

עליכם לחשוב איך מחלקה תשתמש במחלקה אחרת, ואולי להגדיר מחלקות, משתנים ומתודות נוספות כרצונכם.

הערה:

שימו לב, כאשר עליכם להעתיק אובייקט מסוג Key או מסוג Value עליכם לעשות זאת באמצעות המתודה clone() (הסבר על המתודה בהמשך) הקיימת באובייקטים Value ו Key בהתאמה.

הסבר על הקבצים שקיבלתם:

1. Key.h – חתימה של מחלקה אבסטרקטית (לא ניתן ליצור ממנה אובייקטים). מחלקה זו מספקת את הממשק עבור המפתחות שיוכנסו למבנה הנתונים. אינכם יודעים איך ימומש המפתח במבנה הנתונים (ב main שקיבלתם יש דוגמא), המפתח יכול להיות כל מחלקה שיוורשת מ Key ומממשת את המתודות הקיימות ב Key. על מנת ליצור אובייקט Key באפשרותכם להשתמש במתודה clone. מתודה זו מייצרת אובייקט חדש מסוג Key באמצעות אובייקט Key קיים ומחזירה פוינטר לאובייקט החדש. האובייקט החדש שנוצר הוא עותק של האובייקט עליו הפעלנו את הפונקציה clone. כמו כן, הזיכרון של האובייקט החדש מוקצה על ה heap (כלומר באופן דינמי). שימו לב: לאחר שימוש במתודה clone נוצר אובייקט חדש (הפונקציה clone מבצעת את הפקודה new). לדוגמא, הפונקציה Insert יכולה לבצע:

Key* key=nkey->clone();

למשתנה key ייכנס מצביע לאובייקט חדש מסוג Key (שהוא עותק של nkey), כאשר הפונקציה clone שתקרא בזמן הריצה של התוכנית היא זו של המחלקה שירשת מ Key.

2. Value.h – חתימה של מחלקה אבסטרקטית (לא ניתן ליצור ממנה אובייקטים). מחלקה זו מספקת את הממשק עבור המידע שיוכנס למבנה הנתונים. אינכם יודעים איך ימומש המידע במבנה הנתונים (ב main שקיבלתם יש דוגמא) המידע יכול להיות כל מחלקה שירשת מ Value ומממשת את המתודות הקיימות ב Value. על מנת ליצור אובייקט Value באפשרותכם להשתמש במתודה clone. מתודה זו מייצרת אובייקט חדש מסוג Value באמצעות אובייקט Value קיים ומחזירה פוינטר לאובייקט החדש. האובייקט החדש שנוצר הוא עותק של האובייקט עליו הפעלנו את הפונקציה clone. כמו כן, הזיכרון של האובייקט החדש מוקצה על ה heap (כלומר באופן דינמי). שימו לב: לאחר שימוש בפונקציה clone יש לכם אובייקט חדש (הפונקציה clone מבצעת את הפקודה new). לדוגמא הפונקציה Insert יכולה לבצע:

Value* v=nval->clone();

למשתנה v ייכנס מצביע לאובייקט חדש מסוג Value (שהוא עותק של nval), כאשר הפונקציה clone שתקרא בזמן הריצה של התוכנית היא זו של המחלקה שירשת מ Value.

3. LCRS_BalancedTree.txt – קובץ טקסט ובו רשומות חתימות המתודות הפומביות שעליכם לממש במבנה הנתונים כפי שהוגדרו בתחילת התרגיל (מקובץ זה תוכל להעתיק את החתימות על מנת לוודא שיש לך את החתימה המדויקת הדרושה).
4. main.cpp – דוגמת הרצה ודוגמת מימוש של מפתח ושל מידע המשויך למפתח.
5. main_output – פלט לאחר הרצה של main.cpp בשרת ה t2.

אילוצים:

- הקוד שלכם יכול להכיל include רק עבור מחלקות אותן יצרתם ו #include < cstdint >. על מנת לייצג מצביע ל NULL השתמשו במילה השמורה NULL המוגדרת ב cstdint. (בכל מקום שתצטרכו להשתמש ב NULL הוסיפו #include < cstdint >). כמו כן, תצטרכו לעשות include במחלקות שיצרתם ל Key.h, Value.h.

הדרכה:

- אובייקטים מסוג Key תומכים אך ורק באופרטור < במידה ואתם נדרשים, למשל, לאופרטור > האם באפשרותכם לממשו בעזרת האופרטור <?>
- אין להשתמש במתודות המוגדרות במחלקות MyKey ו MyValue ולא מוגדרות במחלקות Key ו Value אלו מימושים ספציפיים ולא בהכרח יהיו מימוש של המחלקות במהלך בדיקת התרגיל.

הנחות:

- הניחו כי המפתחות שיוכנסו למבנה הנתונים יהיו ייחודיים, קטנים ממש מהערך המקסימלי שמאפשר Key וגדולים ממש מהערך המינימלי.
- בקריאה למתודות הפומביות של LCRS_BalancedTree המקבלות כפרמטר מצביע ל Key מובטח כי המתודה תקרא עם ערך מפתח הקטן ממש מהערך המקסימלי האפשרי וגדול ממש מהערך המינימלי האפשרי.

הסבר על תהליך הבדיקה האוטומטית:

אנחנו נריץ את הקבצים שלכם עם פונקציית main שונה מזאת שקיבלתם עם פרסום התרגיל. ב main הבדיקה ייתכן ויתווספו אובייקטים, הפעולות וסדר הפעולות ישתנה, גודל הקלט ישתנה, המימוש של המחלקות MyValue ו MyKey יהיו שונה, וכו'.

במהלך הבדיקה יקומפלו כל הקבצים שהגשתם בתוספת Key.h, Value.h ו main בדיקה בשרת ה t2 עם הפקודה

g++ *.cpp

קובץ ה Key.h וקובץ ה Value.h יהיו בדיוק אותם קבצים שקיבלתם עם פרסום התרגיל. חשוב מאוד שתגישו את כל קבצי ה h וה cpp שיצרתם ותוודאו שהקוד מתקמפל בשרת. בהנחה והקובץ מתקפל, הקוד יורץ והפלט של התוכנית ישווה לפלט חוקי.

המלצות:

1. אל תשאירו את הבדיקה בשרת לרגע האחרון. ייתכן והקוד לא יתקמפל בשרת ותצטרכו לתקנו לפני ההגשה.
2. התחילו מבניית המתודות Insert ו Delete. רק לאחר שמימשתם אותן וביצעתם עליהן בדיקות עבורו לממש את שאר המתודות.