אלגוריתמים לניהול זיכרון דינמי – 236780

תרגיל בית 1

להגשה עד יום ב' 10/12/2018 ב-12:00 בצהריים בתא של הקורס בקומה 1 בטאוב

1. שיטה שלעיתים משתמשים בשילוב עם אלגוריתם sweep-mark היא lazy sweep. בשיטת ה-lazy sweep מבצעים באיסוף את שלב הסימון כשהתוכנית מחכה אבל לא את שלב ה-sweep. אחרי סיום שלב הסימון נותנים לתוכנית להמשיך לרוץ. כאשר צריך להקצות אובייקט מבצעים חלק מהמעבר של ה-sweep ובונים רשימת מקומות פנויים להקצאה בזיכרון, עד שנתקלים במקום מספיק גדול להקצאת האובייקט ואז מקצים אותו, מפסיקים את ה-sweep, ושוב נותנים לתוכנית להמשיך לרוץ. באלוקציה הבאה ממשיכים לעבור, וכן הלאה. היתרון הוא שההפסקה בריצת התוכנית לצורך האיסוף קְטַנָה כי עושים רק את ה-mark, וגם הלוקליות משתפרת כי השטחים שנוגעים בהם בזמן ה-sweep לרוב הופכים די מהר לשטחים שבהם התוכנית משתמשת (כי האלוקציה נותנת אותם לתוכנית).

כמובן שהיתרונות הנ"ל מתקבלים לתוכניות "רגילות" ובתרגיל זה תתבקשו להראות שבמקרה הגרוע לא מרויחים מזה כלום (ולכן גם להראות שהבנתם את השיטה הזו).

תארו תוכנית שתגרום ל-lazy sweep להתנהג כמו sweep רגיל. כלומר, למרות שה-collector מנסה לבצע את ה-sweep צעד אחר צעד, התוכנית תמיד תכריח אותו לבצע את כל ה-sweep בבת אחת.
הערות:
   א. על התוכנית להצליח להקצות כל בקשה, כלומר היא אינה "עפה" בגלל מחסור בזיכרון.
   ב. הפיתרון הטריביאלי של להקצות ולשחרר רק אובייקט אחד בכל מחזור איסוף אינו מתקבל. יש לאפשר ל-sweep לשחרר לפחות 100 אובייקטים בזמן האיסוף.
   ג. לבסוף, אין להקצות אובייקט אחד שממלא את כל ה-heap. הגודל של כל אובייקט מוקצה חסום ב-1/100 מגודל ה-heap.

2. שאלה זו מתייחסת לאלגוריתם ה-compaction של Jonkers:
   א. כזכור האלגוריתם משתמש בשרשור על-מנת לעדכן את המצביעים להצביע למקום החדש של האובייקט. הסבירו מהי פעולת השרשור וכיצד משתמשים בה.
   ב. הסבירו כיצד ניתן לשלב את השרשור של השלב הראשון של האלגוריתם של Jonkers עם ה-mark של האובייקטים בזמן ה-mark. מה נותר לביצוע בפאזה הראשונה של האלגוריתם המעודכן? ומה בפאזה השנייה?
   ג. האם ניתן להעביר עוד פעילות של הדחיסה לתוך ה-mark (ללא ביצוע heap pass נוסף וללא שימוש בזיכרון נוסף) ואז לוותר על הפאזה הראשונה כולה ולהישאר רק עם מעבר אחד על ה-heap בזמן הדחיסה אחר ה-mark? אם כן - הסבירו כיצד, אם לא - הסבירו מה משתבש בנסיון פשוט לעשות זאת.
   ד. כיצד ייקבע סדר הפוינטרים בשרשרת של אובייקט כאשר יורץ האלגוריתם שתכננתם בסעיף ב'?
   ה. כיצד ישפיע השינוי של סעיף ב' על ה-locality של אלגוריתם הדחיסה? התייחסו רק לשינוי שנגרם ע"י הכנסת השרשור של החלק הראשון אל תוך ה-trace והוצאתו משלב המעבר

הראשון. האם אתם מצפים לשיפור ב-locality? הסבירו.
הניחו שבניהם של אובייקטים נמצאים במקום אקראי כלשהו בזיכרון (ולאו דווקא ליד האובייקטים ההורים).

3. נניח שהיה ניתן לקבל ממערכת ההפעלה מידע על "מצביעים הפוכים". כלומר, לכל אובייקט היה ניתן לקבל (באופן פלאי) ביעילות רשימה של כל אבותיו ע"י הפעלת קריאת מערכת "getParents" עם כתובת האובייקט. (רשימת האבות כוללת גם אבות שהם שורשים.) בשאלה זו נבדוק כיצד היה ניתן להשתמש בפעולה זו לאיסוף אשפה. כשנאמר "נגיש", "אבות", ו"בנים" נתכוון למושגים האלה ביחס למצביעים המקוריים (ולא ההפוכים).
ניתן להניח שהרוטינה getParents מתבצעת באופן אטומי והיא מודעת תמיד למצב הנוכחי של ה-heap על כל ההזזות שאובייקטים עברו עד נקודת הזמן שבו היא נקראה.
   א. האם בעזרת המידע על המצביעים ההפוכים בלבד ניתן לדעת עבור אובייקט נתון כלשהו אם הוא נגיש מהשורשים או לא (מבלי להתבונן במכוונים ה"רגילים" שבתוך האובייקט)? נמקו.
   ב. במידה וגילינו שאובייקט אינו נגיש (מהשורשים). האם מובטח שגם כל בניו לא נגישים? האם מובטח שגם כל אבותיו לא נגישים?
   ג. תארו אלגוריתם יעיל ככל האפשר ל-compaction בסביבה זו המשתמש במעבר אחד בלבד על ה-heap (בהינתן סימון מי מהאובייקטים חי ומי מת). הסבירו את מחיר האלגוריתם בזמן ומקום.
   ד. נניח שכל תוכניות המחשב בג'אווה בעולם אינן מכוונות יותר משני מצביעים אל אותו אובייקט. (כלומר לכל אובייקט יש לכל היותר 2 אבות). תארו מימוש (יעיל ככל יכולתכם) של getParents שמחזיר תשובה נכונה לאורך כל זמן ריצת התוכנית. ניתן כמובן להשתמש ב-read-barrier או ב-write-barrier, בנוסף אולי לביצוע פעולות נוספות בזמן ה-compaction. הסבירו את העלויות במקום וזמן.
   ה. עתה נניח שבתוכניות ג'אווה ל-98% מהאובייקטים יש לכל היותר שני אבות, אבל לשאר האובייקטים ייתכנו יותר אבות, אך לכל היותר 10. התוכלו לתאר מימוש (יעיל ככל האפשר) של getParents המסוגל לטפל גם ב-2% האובייקטים הפופולאריים?
   ו. נניח שמנסים למקבל את האלגוריתם שהצעתם בסעיף ג' ע"י חלוקת ה-heap לאזורים. כל חוט מתחיל בלבקש אחריות על אזור. בקשה זו מתבצעת בזהירות תוך תיאום עם מנעול כדי שכל חוט יקבל איזור אחד ויחיד. אחרי כן, החוט מבצע compaction לוקלי באזור תוך הזזת האובייקטים בתוך האזור לתחילתו. האם יש בעיה במיקבול כזה בגלל המקביליות של חוטי ה-compaction באלגוריתם?

4. התבוננו בתיקון הבא לאלגוריתם של Lang & Dupont:
באלגוריתם המקורי, לאחר האיסוף מתפנה כל השטח של from-space וגם חלק מהשטח של to-space, בגלל הדחיסה. בתיקון לאלגוריתם, בתום האיסוף, נקבע את from-space להיות, כמו במקור, השטח הבא. אבל את to-space במחזור הבא נקבע להיות גדול יותר. הוא יתחיל מיד במקום שבו סיים המחזור הקודם להעתיק את האובייקטים החיים לתוך to-space שלו. אזור ה-to-space של המחזור הנוכחי יכלול את כל השטח הפנוי ברצף משמאל ל-from-space הנוכחי. אם נמשיך כך שוב ושוב אז נראה שייווצר שטח רצוף ריק ההולך וגדל ונקבל קירוב טוב יותר ל-compaction. נרצה לבחון את החסרונות של התיקון הנ"ל. על-מנת לממש רעיון זה, צריך להחליט האם לתוכנית מותר להקצות בשטח הריק שהולך וגדל באיסופים (תוך שמירת מקום ל-to-space) או לא.

   א. אם אסור להקצות בשטח הריק בין מחזור למחזור אז השטח הפנוי הולך וגדל – מה החיסרון בשיטה זו?

ב. אם מרשים להקצות בשטח הריק (בשמאלו, עד כדי שמירת לפחות שטח אחד שלם ריק שבו לא מקצים כדי שישמש כ- to-space של המחזור הבא), האם השיטה הזו מועילה (יחסית לאלגוריתם המקורי ללא התיקון)? אם כן - הסבירו במה התועלת. אם לא - נמקו.
הניחו שמבצעים איסוף כאשר אין יותר מקום פנוי להקצאה.

בהצלחה!