**מבני נתונים – פרויקט מעשי 2 - קובץ תיעוד**

**פרטים אודות מגישות-**

הגר פייתן

* שם משתמש - hagarleap
* תעודת זהות - 206825176

גל קריאל

* שם משתמש - galkariel
* תעודת זהות - 318459666

**חלק ניסויי/תיאורטי**

**שאלה 3 – השוואה בין quadratic probing ל-alternating quadratic probing**

**שאלה 3 - סעיף 1**

גדלי הקבוצות:

( נבחין כי הגודל של זה בדיוק וכי הגודל של זה בדיוק q )

**שאלה 3 - סעיף 2**

מדדנו עבור quadratic probing את כמות השגיאות הנובעות משגיאה כי הטבלה מלאה ומשגיאה כי מפתח כבר קיים במערך.

ראינו כי אין שגיאות שנובעות ממפתח שקיים כבר במערך, אבל קיבלנו בין 0 ל-3 שגיאות בכל ניסוי שנובעות מכך שהטבלה מלאה.

לעומת זאת עבור alternating quadratic probing לא נזרקו שגיאות כלל וכל האיברים הוכנסו לטבלה.

הסבר לשוני בין התוצאות-

ראינו מהסעיף הקודם כי כאשר אנחנו משתמשים במודל של quadratic probingאז כתוצאה סדרת הבדיקות שלנו עוברת רק על קבוצת מספרים בגודל חצי מהמספר הראשוני q () . כלומר פונקציית ה-hash במודל של quadratic probing לא נותנת לנו את כל האינדקסים בטבלה ולכן ייתכנו סדרות של בדיקות שלא עוברות על כל האינדקסים בטבלה כלומר נקבל שגיאות כי הטבלה מלאה למרות שבפועל יש בה עוד מוקמות להכניס איברים.

לעומת זאת כאשר אנחנו משתמשים במודל של alternating quadratic probing ראינו בסעיף הקודם שסדרת הבדיקות שלנו עוברת על קבוצת מספרים בגודל q () כלומר אין חזרות של מספקים ואנחנו עוברים **על כל** האינדקסים בטבלה– כלומר בהכרח נמצא מקום פנוי אם קיים לטובת הכנסת מפתח חדש לכן לא נקבל שגיאות ונצליח להכניס את כל המפתחות.

**שאלה 3 - סעיף 3**



**שאלה 4 – נשווה בין המימושים השונים ל-open addressing**

**שאלה 4 - סעיף 1**

|  |  |
| --- | --- |
| Class | Running Time |
| LPHashTable | 0.510 |
| QPHashTable | 0.498 |
| AQPHashTable | 0.514 |
| DoubleHashTable | 0.704 |

הסבר מילולי להבדלים בזמני הריצה בין סוגי הטבלאות-

נבחין כי זמני הריצה של DoubleHashTable יותר ארוכים באופן יחסי לעומת שאר המימושים. ושאר המימושים רצים בזמנים מאוד דומים.

ראינו בהרצאה כי הסיכוי להתנגשויות כאשר משתמשים בפונקציות hash מהמשפחות האוניברסליות הנו נמוך ( כאשר m זה גודל טבלת ה-hash). בנוסף אנחנו ממלאים רק את חצי הטבלה, ולכן נעריך כי אפילו אם יש התנגשויות לא יהיו "רצפים" גדולים מידי של איברים בטבלה - שעלולים היו לפגוע בזמני הריצה עבור LPHashTable למשל). כלומר בעיית הרצפים (הצטברות ראשונית/משנית) עבור סעיף זה לא תשפיע הרבה על זמני הריצה.

בנוסף לכך, נרצה לציין כי ציפינו ש- LPHashTable ייקח פחות זמן מ- QPHashTableu ומ-AQPHashTable כיוון שראינו במבוא מורחב כי חיבור מספר בודד זה פעולה זולה יותר מהעלאה בחזקה. אבל נראה כי אין משמעות גדולה בזמני הריצה לפעולות חישוב אלה (חיבור והעלאה בחזקה).

עבור LPHashTable, QPHashTableu ו- AQPHashTable מתבצעת אותה כמות של פעולות מודולו וראינו כי זמני הריצה שלהם דומים.

לעומת זאת ב- DoubleHashTable יש יותר פעולות מודולו (אנחנו מפעילים שתי פונקציות hash עבור כל בדיקה להכנסת איבר) ולכן אנו מעריכות כי השוני בזמני הריצה נובע מפעולת המודולו.

**שאלה 4 - סעיף 2**

|  |  |
| --- | --- |
| Class | Running Time |
| LPHashTable | 5.551 |
| AQPHashTable | 2.776 |
| DoubleHashTable | 3.836 |

ההסבר לכך שאין לבצע סעיף זה עבור הנה - כפי שראינו בשאלה הקודמת, פונקציית ה-hash במודל של quadratic probing לא נותנת לנו את כל האינדקסים בטבלה, ובפרט נותנת לנו רק חצי מהם. לכן הכנסה של כמות איברים שגדולה מחצי מהטבלה עלולה לגרום לשגיאות "table is full" כיוון שסדרת הבדיקות לטובת הכנסה תהיה רק על חצי מהטבלה.

בניסוי זה אנו מכניסים יותר מחצי איברים של הטבלה ולכן ייתכן ויהיו שגיאות (בשאלה זו לא אמורים להיזרק חריגים) ולכן לא נבצע בסעיף את המימוש עבור .

הסבר להבדל בביצועים (זמני הריצה) לעומת הסעיף הקודם:

LPHashTable- זמני הריצה של מודל זה הנם הארוכים ביותר.

זמני הריצה גדלו בערך פי 10 לעומת הניסוי הקודם (למרות שהקלט גדל רק בערך פי 2) וזו הגדילה הכי גדולה לעומת AQP, DoubleHash.

אנו מעריכות כי הדבר נובע מבעיית ההצטברות הראשונית – נוצרים רצפים ארוכים של תאים תפוסים. סדרות הבדיקה להכניס איבר שיש רצפים ארוכים יהיו יקרות (הרבה פעולות מודלו).

AQPHashTable-

זמני הריצה גדלו בערך פי 5 לעומת הניסוי הקודם (למרות שהקלט גדל רק בערך פי 2) .

אנו מעריכות כי הדבר נובע מבעיית ההצטברות המשנית – זוהי צורה מתונה יותר של הצטברות.

יפה לראות כי בעיית ההצטברות המשנית, באמת מתונה יותר מבעיית ההצטברות הראשונית (זמני ריצה שגדלו פי 5 לעומת זמני ריצה שגדלו פי 10).

DoubleHashTable-

זמני הריצה גדלו בערך פי 5 לעומת הניסוי הקודם (למרות שהקלט גדל רק בערך פי 2) .

בשיטה זו תיאורטית לא אמורים להיות רצפים ארוכים של תאים תפוסים שכן סדרת הבדיקות מחושבת על ידי שתי פונקציות hash שונות.

תוצאות אלה מפתיעות אותנו. שכן היינו מצפות שזמני הריצה יגדלו בערך פי 2 ( כפי שהקלט גדל).

הסבר אפשרי לכך שזמני הריצה גדלו פי 5 בכל זאת יהיה, שכאשר הטבלה לקראת התמלאות יש יותר התנגשויות שיגררו יותר חישובים יקרים.

**שאלה 5 – השפעת מחיקת איברים ב-open addressing**

|  |  |
| --- | --- |
| Iterations | Running Time |
| First 3 iterations | 4.513 |
| Last 3 iterations | 19.45 |

נשווה את זמן ביצוע 3 האיטרציות הראשונות לזמן ביצוע 3 האיטרציות האחרונות.

נבחין כי קיים הבדל משמעותי בין זמני הריצה, ב-3 האיטרציות הראשונות זמני הריצה קצרים משמעותית מ-3 האיטרציות האחרונות.

אנו מעריכות כי הסיבה להבדל הנה שאנו לא מאפסות בכל איטרציה מחדש את המבנה. לכן כשעושים יותר איטרציות יש יותר תאים בטבלה שמסומנים כ-deleted (כיוון שאנחנו מוחקים איברים את כל האיברים שהכנסנו למבנה בכל איטרציה) ולא כ-null . בפונקציית ה-Insert שלנו אנחנו מכניסים איבר בתא שמסומן deleted רק אחרי שעברנו על כל הטבלה לוודא שאין מפתח זהה לזה שאנחנו רוצים להכניס (כיוון שיתכן שהמפתח נמצא בהמשך הטבלה), לעומת זאת עבור תא שמכיל- null אנחנו מכניסים את האיבר החדש ישירות לטבלה שלנו ומסיימים. לכן ככל שיהיו יותר תאים שמסומנים deleted נקבל זמני ריצה ארוכים יותר עבור ההכנסות ולכן זמני ריצה ארוכים יותר עבור הניסוי.