תכנות מרובה ליבות – תרגול 5#

קונצנזוס:

רוצים שכמות מסויימת של חוטים תחליט החלטה משותפת שהיא ערך שהוצע ע"י אחד החוטים, וכולם יחזירו את אותו ערך.

נשתמש בתבנית אחת למימוש קונצנזוס של מימוש שתי מתודות:

- Propose(value) – הצעת ערך ע"י חוט
- Decide() – החלק החשוב בו מחליט החוט הקורא מה הערך שיחזיר (צריך להיות אותו אחד לכולם כמוגדר לעיל).

Consensus number: n עבור אובייקט, הוא שעבור לכל היותר n חוטים אותו אובייקט פותר קונצנזוס.

דוגמא: peekableQueue

תור שניתן לבצע עליו peek כדי לראות מה הערך הבא שיצא (אך לא מוציא). המס' שלו הוא אינסוף: מכניסים את הערך, ואז מחזירים את top ה-queue, שהוא הראשון שהוכנס לתור.

דוגמא נוספת:

נתון $atomic$ רגיסטר $\left(n, \frac{n(n+1)}{2}\right)$. נראה פתרון ל-n+1 חוטים:

עבור שני חוטים ו-3 מקומות במערך, השתמשנו בתא משותף. הדורס את התא המשותף יודע שהוא השני. מקום אחד שלא נדרס שומר את הערך. אותו דבר נרצה לעשות ל-n חוטים ו-$\frac{n(n+1)}{2}$ מקומות במערך. זו למעשה חצי מטריצה, נבנה חצי מטריצה שנותנת את החיתוכים בין כל החוטים, ומערך נוסף שבו נפרסם לכל חוט שהוא עשה משהו (במקביל לתא שלא נדרס בדוגמא הפשוטה).

כל חוט שכותב, כותב למקום שלו במערך הנפרד, ובכל השורה והעמודה תחתיו – בדרך ידרוס כמובן. ה-n (הארגומנט הראשון) מגדיר כמה מקומות בזיכרון יכולים להכתב ע"י חוט בו"ז באופן אטומי.

ההחלטה של כל חוט תהיה: החוט יבדוק קודם מי עשה השמה (מי שם במערך הנפרד ערך), ולכל אחד כזה צריך ללכת לנקודת ההצטלבות איתו ולראות את מי רואים. אם רואים אותו – אני הגעתי לפניו. החוט יבדוק רק לחוטים שמופיעים בעת הקריאה במערך הערכים.

דוגמא נוספת: Approximate agreement:

בעיה זו פותרת משהו קרוב לקונצנזוס – מוחזר ערך בתוך טווח כלשהו קטן מאפסילון מוגדר מראש. נניח יש אובייקט כזה עבור שני חוטים, מה מספר הקונצנזוס שלו? נשים לב שזה שהוא מוגדר ל-approx(2) (לשני חוטים), זה לא חוסם את מס' הקונ' שלו ל-2, כי ניתן אולי להשתמש בהרבה אובייקטים כאלה ולהשיג קונ' גבוה.

הפתרון: המספר קונ' שלו הוא 1, כי ניתן לבנות אותו עם רגיסטרים אטומיים. הבניה:

$y_i \coloneqq x_i$

$if\ y_j \perp, return\ y_i$ // return me

$while\ (|y_i - y_j| > \epsilon): y_i \coloneqq \frac{y_i + y_j}{2}$ // get me closer to him

נכונות: אם חוט הגיע לשלב בדיקת השני, זה כבר אחרי שעשיתי השמה. אם הוא רואה בשני null, אז הוא לפני שהשני כתב. כשהשני בא לבדוק, זה יהיה אחרי שהראשון ראה null, ולכן זה אחרי שהראשון כתב את ערכו. לכן השני בהכרח יראה את ערכו של הראשון כלא null.

דוגמא נוספת: Trinary register:

יכול להיות במצבים $0,1,\perp$ ויכול לעשות get(), compareAndSet(). צריך לפתור קונ' n ל-k ביטים או בעזרת n רגיסטרים טרינריים וגם בעזרת k רגיסטרים טרינריים.

פתרון ל-k רגיסטרים טרינאריים: חוט שמגיע מציע את המספר שלו (propose). מחזיקים k רגיסטרים טרינאריים. הראשון מצליח לבצע על הביט הראשון compareAndSet, ואז השניים האחרים (אולי יותר) באים ומנסים לבצע ונכשלים. הם עוזרים לראשון: הולכים לחפש בערכים המוצעים ערך שמתחיל ברישא של הביט שנתקלו בו (אחרי שנכשלו לבצע compareAndSet), ומנסים לכתוב את הביט השני של אותו ערך שמצאו (יתכנו כמה, יקחו

את הראשון). אז מישהו מהם מצליח, או הראשון המקורי או מישהו אחר. בגלל שכל הכותבים שומרים לאורך הדרך רישא של מישהו, תמיד יכתב ערך חוקי – והוא יהיה הערך שיוחלט בסוף.

פתרון ל-n רגיסטרים טרינאריים: בונים מערך טרינאריים בגודל n, כל ביט מייצג מזהה-חוט. כל חוט שמגיע מנסה לרשום 1 לביט שלו. אם הצליח, הולך ומנסה לכתוב אפסים לכל האחרים שלפניו. אם בעת ניסיון הכתיבה לא מצליח לכתוב למישהו 0 – אז או שהוא הגיע לפניו ושם כבר 1, או שמישהו שם שם 0. אם אני רואה 1 אני מחזיר אותו, אם אני רואה 0 אני ממשיך לנסות לכתוב אפסים עד ה-1 הראשון שאני רואה (שיתכן וזה אני).

Team consensus:

פותר קונצנזוס כל עוד מגיעים רק שני ערכים שונים. כיצד מייצרים איתו קונ' רגיל? בונים עץ team-con: כך מספר הקונ' שלו הוא אינסוף. בכל רמה בעץ מובטח שלא יוחזרו יותר משני ערכים שונים. מובטח שבסופו של דבר יוחלט ערך אחד בודד. כך פותרים n-con בעזרת כמה שנרצה team-con.