תרגיל 3 סנכרון ושרת Kosaraju

גירסא 1.0 - 2 ביוני 2024

בתרגיל נממש שרת על גרף הממש את אלגוריתם Kosarju-Sharir למציאת רכיבי קשירות חזקה בגרף.
השרת יקבל גרפים או שינוים בגרפים, ממספר משתמשים מתקשורת TCP או מstdin
יריץ את האלגוריתם ויחזיר פלטים.

האלגוריתם נמצא ב
https://en.wikipedia.org/wiki/Kosaraju%27s_algorithmhttps://en.wikipedia.org/wiki/Kosaraju%27s_algorithm

שלב 1 - שרת Kosaraju 10 נקודות

קלוט מהקלט הסטנדרטי 2 מספרים. +LF
המספר הראשון מתאר את מספר הקודקודים בגרף.
המספר השני מתאר את מספר הקשתות בגרף.

למספר הראשון (מספר הקודקודים) נקרא n.
למספר השני (מספר הקשתות) נקרא m

קודקודי הגרף ימוספרו כמספרים שלמים מ1 עד n.

לאחר קריאת הזוג הראשון נקרא עוד m זוגות כל זוג ייצג קשת
כלומר הזוג
(1,2) ייצג את הקשת מ 1 ל2.

<u>שימו לב - הגרף הוא גרף מכוון.</u>

לאחר קריאת הגרף בצע את האלגוריתם של kosaraju והחזר כפלט את רכיבי הקשירות החזקה של הגרף.
(כל רכיב קשירות בשורה נפרדת)

דוגמא לקלט

שורה	משמעות
5 5	5 קודקודים 5 קשתות
1 2	קשת בין קודקוד 1 ל 2
2 3	קשת בין קודקוד 2 ל 3
3 1	קשת בין קודקוד 3 ל 1 (רכיב קשירות חזקה 1,2,3)
3 4	קשת בין קודקוד 3 ל 4
4 5	קשת בין קודקוד 4 ל 5

פלט

שורה	משמעות
1 2 3	רכיב הקשירות הראשון הוא 1 , 2 ,3
4	רכיב הקשירות השני הוא 4
5	רכיב הקשירות השלישי הוא 5

שלב 2 - profiling - עשר נקודות

5 נקודות

בחן את מימוש korasaju. (שלב 1) במימוש השתמשנו ברשימה מקושרת.
בחר שני ישומים של רשימה (לדוגמא למי שמשתמש בc++ מימוש של deque ומימוש של list)
בצע profiling לריצה של שלב 1 עם שני המימושים השונים. איזה מהמימושים יעיל יותר במקרה שלנו?

5 נקודות

הצע שני מימושים שונים לגרף.
לדוגמא: מטריצת שכנויות של n*n ולחלופין vector של קודקודים כאשר כל קודקוד מחזיק list של קשתות (ילדים)
בצע profiling לזמן שלוקח

- להריץ את האלגוריתם על הגרף
- לבצע שינויים בגרף (למחוק ולהוסיף קשתות)

כתוב תוכניות בדיקה מתאימות. לאחר הפרופילינג יש להמשיך את העבודה בשלבים הבאים עם המימוש היעיל יותר

שלב 3 - קצת אינטרקציה - 10 נקודות

האלגוריתם שלנו יתמוך באינטרקציה עם לקוחות דרך stdin. כל פקודה תסתיים בLF.
(אפשר להניח שהקלט חוקי)

התוכנית תקבל עכשיו קלט מהstdin ותתמוך בפקודות הבאות

Newgraph n,m

קבל גרף חדש בm זוגות - יש לקלוט m שורות של 2 מספרים (m קשתות כמו בסעיף הקודם)

Kosaraju

חשב את האלגוריתם של kosaraju על הגרף הנוכחי הוצא פלט ל stdout

Newedge i,j

הוסף קשת בין i לj (אפשר להניח שלא קיימת קשת כזאת כלומר הקלט חוקי)

Removeedge i,j

מחק קשת בין i לj (אפשר להניח שקיימת קשת כזאת כלומר הקלט חוקי.)

*** אין פקודות אחרות

שלב 4 - ריבוי משתמשים - 10 נקודות

נמזג את התכנית של שלב 3 ואת הchat של beej. מטרתנו לשלב בינהם כלומר נייצר שרת שיחזיק מבנה נתונים של גרף (משותף לכל המשתמשים) וכל משתמש מחובר יכול להוסיף קשת או להוריד קשת או להורות על חישוב kosaraju על הגרף הנוכחי.

פרוטוקול התקשורת בין השרת ללקוחות - יהיה טקסטואלי (כמו בשלב 3) על גבי port 9034 (הport של beej)

כל משתמש נותן פקודות (כפי שנתנו בשלב 3) או מורה על חישוב האלגוריתם.
משתמש שנותן פקודות שמשנות את הגרף צריך לחכות לשינוי או חישוב קודם שיסתיים.

* שים לב בסעיף זה עדיין לא צריך להגן על חישוב הגרף בעזרת mutex מכיוון שאנחנו עדין עובדים בטרד אחד ולכן לא יהיה שינוי וחישוב לגרף במקביל.

שלב 5 - תבנית reactor - עשר נקודות

בנה ספרית תבניות עם תבנית ל reactor.
הreactor יכול לעבוד עם $select(2)$ או $poll(2)$ לבחירתכם.

הreactor יקבל fd ופונקציה לקרוא לה כאשר הfd חם

```
typedef (void *) (* reactorFunc) (int fd);
typedef reactor;

// starts new reactor and returns pointer to it
void * startReactor ();

// adds fd to Reactor (for reading) ; returns 0 on success.
Int  addFdToReactor(void * reactor, int fd, reactorFunc func);

// removes fd from reactor
Int removeFdFromReactor(void * reactor, int fd);

// stops reactor
int stopReactor(void * reactor);
```

אפשר להניח קלט חוקי (אם מוסיפים fd לreactor הוא לא קיים. אם מוציאים fd מהreactor הוא קיים)

שלב 6 - מימוש שלב 4 בעזרת תבנית (שלב 5) - עשר נקודות

ממש את שלב 4 אבל בעזרת התבנית והספריה שמימשת בשלב 5.

שלב 7 - מימוש מרובה threads - עשר נקודות

חזור למימוש של שלב 4.
אנחנו לא מעוניינים בasync I/O יותר - במקום אנחנו מעוניינים בפתיחה של thread על כל לקוח חדש שנוצר.
(ככה שלשרת יש n+1 טרדים אם יש n לקוחות מחוברים. טרד שעושה accept ועוד n טרדים - אחד לכל לקוח).

שים לב שהגרף הוא משאב משותף ואנחנו צריכים להגן עליו בעזרת mutex. (כדי שלא ישתנה בזמן שטרד אחר עובד עליו - למשל מריץ את האלגוריתם).

שימו לב - המימוש של ה-mutex יכול להיות פשוט, אין צורך בהגנה נפרדת על כתיבות וקריאות.

שלב 8 - תבנית proactor - עשר נקודות

הוסף לספריה (שלב 5) תבנית של proactor.
התבנית תקבל socket עליו היא מאזינה. בכל פעם שמתחבר לקוח (עושים accept) הproactor יצור טרד חדש עם פונקציה שיקבל כפרמטר.

```
Typedef (void *) (* proactorFunc) (int sockfd)

// starts new proactor and returns proactor thread id.
pthread_t startProactor (int sockfd, proactorFunc threadFunc);
// stops proactor by threadid
int stopProactor(pthread_t tid);
```

שים לב שהגרף הוא משאב משותף ושינויים למשאב המשותף חייבים להיות מוגנים בmutex.
בנוסף לא יכול להיות מצב שבו נשנה את הגרף בזמן שטרד אחר עובד עליו (מחשב את האלגוריתם)

שלב 9 - מימוש שלב 7 בעזרת תבנית (שלב 8) - עשר נקודות

ממש את שלב 7 בעזרת הספריה שמימשת בשלב 8.

שלב 10 - producer consumer - עשר נקודות

הוסף לשרת (שלב 9) טרד אחרון שממתין למצב שבו לפחות 50% מהגרף (כלומר אם יש בגרף n קודקודים לפחות n/2 קודקודים) נמצאים ברכיב קשירות אחד.
אם לאחר חישוב kosaraju (כזה שיזם משתמש, אין צורך לבצע חישוב בכל הכנסת קשת בודדת) הגענו למצב כזה יש להעיר את הthread שיוציא הדפסה לstdout

At Least 50% of the graph belongs to the same scc\n

אם לאחר שכבר הגענו למצב שרוב הגרף נמצא באותו רכיב קשירות נגיע למצב שלאחר חישוב kosaraju (עקב הסרת קשתות) הגענו למצב שיש 50% מהגרף שכבר לא שייכים לאותו רכיב קשירות האלגוריתם יוציא הדפסה לstdout

At Least 50% of the graph no longer belongs to the same scc\n

יש להעיר את הטרד הנוסף בעזרת POSIX cond.