**פרויקט גמר מערכות הפעלה**

**שאלה 1 :**

התבקשנו ליצור מבנה נתונים של גרף, בחרתי להשתמש במטריצת שכנויות.

כאשר יש צלע יופיע 1 וכאשר אין צלע inf .

|  |
| --- |
|  |

**שאלה 2** : בשאלה זו התבקשנו לממש אלגוריתם למציאת מעגל אויילר או הוכחה שאינו קיים.

נבצע זאת במספר שלבים:

שלב ראשון: נבחן האם כל דרגות הקודקודים הגרף זוגיות, במידה לא קיים מעגל אויילר (נלמד בדיסקרטיים). במקרה זה נדפיס את הקודקוד שדרגתו אי זוגית.

|  |
| --- |
|  |

שלב שני: נבחן האם הגרף קשיר, נעשה זאת באמצעות הפעלת DFS על הגרף. אם הגרף אינו קשיר (בתנאי שיש צלעות בשני רכיבי הקשירות) הוא בפרט לא מכיל מעגל אויילר נדפיס את הקודקוד שהתגלה כלא קשיר.

|  |
| --- |
|  |
|  |

שלב שלישי: אחרי שאנחנו יודעים שקיים נמצא אותו באופן הבא:

* ניצור מטריצת עזר שתהיה העתק של מטריצת השכנויות המקורית.
* יצירת מחסנית שתחזיק את הנוכחי.
* נתחיל מקודקוד מקור כלשהו, נכניס קודקוד שכן אחד שלו למחסנית ונסיר את הצלע בניהם.
* נעבור לקודקוד הבא במחסנית ונכניס קודקוד שכן אחד שלו ונסיר את הצלע בניהם.
* באופן זה נמשיך עבור כל הצלעות עד שנגיע לקודקוד שלא קיימות עבורו צלעות וזה אומר שסיימנו את הקטע ממסלול זה ונכניס אותו לווקטור של אויילר, נמשיך רקורסיבית על התוכן של המחסנית.
* נמשיך כך כל עוד המחסנית מכילה קודקודים.

|  |
| --- |
|  |

**שאלה 3 :**

בשאלה זו התבקשנו ליצור גרף רנדומלי בעזרת דגלים שנקבל מהטרמינל.

נעזרנו בפונקציה getopt\_long על מנת לפרסר בצורה נוחה את הקלט המתקבל.

נבצע בדיקות תקינות על הקלט (בדיקה שבאמת הגיע מספר, בדיקה שמספר הצלעות לא גבוה מידי , בדיקה שכל הארגומנטים אותחלו)

נאתחל את השדות של הגרף, נקצה מקום למטריצת השכנויות ונפעיל את הפונקציה היוצרת גרף רנדומלי.

|  |
| --- |
|  |
|  |
| את יצירת הגרף הרנדומלי נבצע כך:   * יצירת ווקטור שיכיל את כל זוגות הצלעות האפשריות. * ערבוב הווקטור בעזרת הגרעי הרנדומלי שקיבלנו בקלט. * הוספה לגרף את e הצלעות הראשונות בווקטור הרנדומלי (כאשר e זה מספר הצלעות הנדרשות) |
|  |
| לבסוף נפעיל את מציאת מעגל אויילר על הגרף שנוצר ונקבל אחת מהאופציות הבאות:   1. אין מעגל אויילר כי קיימת דרגה אי זוגית בגרף של הקודקוד i . 2. אין מעגל אויילר כי הגרף לא קשיר, הקודקוד s לא מגיע אל הקודקוד i. 3. יש מעגל אויילר והוא : (תצוגה של מעגל אויילר בקודקודים וחצים) |
|  |
| נציג 2 קלטים (seed=42):   1. 5 קודקודים , 10 צלעות – הגרף המלא כאשר כל הדרגות הם 4 לכן קיים מעגל אויילר. 2. 6 קודקודים, 15 צלעות – הגרף המלא כאשר כל הדרגות הם 5 – אין מעגל אויילר. |
|  |
|  |

**שאלה 4 :**

|  |
| --- |
| בתיקיית gcov\_reports נמצאים הדוחות של .cpp.gcov שמציגים את השורות שהגענו אליהם.  כאן נציג את הדוח שיוצא הטרמינל. |
| עבור main.cpp : עם ההרצה של 2 גרף עם מעגל אויילר וגרף ללא מעגל אויילר. |
|  |
| עבור Graph.cpp אתאר אילו הרצות ביצעתי בשביל להגיע לכיסוי:   * חוסר בדגלים , דגלים לא קיימים. * קלט בטווח לא מתאים או לא מספר. * יותר מידי צלעות. * קלט תקין ללא מעגל אויילר. * קלט תקין עם מעגל אויילר. |
|  |
| כעת נבצע בדיקת זליגת זיכרון באמצעות valgrind:  ניתן לראות שאין שום הקצאה של משתנים בheap (כי לא ביצענו new) ושאין זליגת זיכרון. |
|  |
| נציג גם את גרף הקריאות שמתאר את חלוקת הקריאות בין הפונקציות השונות, מי קוראת למי וכמה פעמים כל פונקציה נקראת. באופן זה נזהה צווארי בקבוק.  כדי לראות תצוגה גרפית תקינה נתקין כלי בשם KCachegrind עם הפקודה :  sudo apt install kcachegrind  ואחרי שהרצנו:  Make valgrind-callgrind  נריץ את השורה הבאה:  kcachegrind callgrind.out.\* |
|  |
|  |
|  |
| ניתן לראות כי הפונקציה שצרכה הכי הרבה משאבים היא parseFlags עם 4.38 מיליון הוראות למעבד. |

**שאלה 5 : נכתבה בטעות לא סופק קוד.**

**שאלה 6 :** בשאלה זו התבקשנו לממש תקשורת שרת לקוח, הלקוח ישלח גרף לשרת ויקבל מעגל אויילר אם יש או יקבל שאין.

|  |
| --- |
| צד שרת: (TCP)   * נשתמש באופציות על מנת לפרסר את הפורט שעליו השרת יאזין. |
|  |
| * בפונקציית run\_server נבצע מספר פעולות על מנת לאפשר קבלת תקשורת מלקוח. * הגדרת משתנים ומבנים הדרושים ליצירת socket ו bind * יצירת ה socket המאזין של השרת. * שינוי הגדרת ה socket בפקודה setsocketopt המאפשרת שימוש חוזר בפורט. * הגדרת הכתובת בתור ipv4 והמרת הפורט למבנה מתאים. * יצירת חיבור בין הכתובת לפורט באמצעות bind. * הגדרת התור להמתנה לaccept ל10 לקוחות באמצעות listen. |
|  |
| בלולאה ראשית:   * נכניס אל קבוצת fd את stdin ואת כל הלקוחות. * נמצא את המספר המקסימלי של fd כדי שנדע את הטווח שצריך להעביר ל select. |
|  |
|  |
| בעת קבלת מידע מלקוח ניכנס ללולאה הבאה לפירוש המידע:   * קבלת גודל המטריצה ובדיקת תקינות. * יצירת המטריצה מהגודל המתאים שהתקבל. * בלולאה קבלת שורה אחר שורה של קודקודים במטריצה והזנתם ישירות במטריצה החדשה. * במידה וכל המידע התקבל באופן תקין נעדכן את הפרמטרים המתאימים של המטריצה בפונקציית parseFromMatrix ונפעיל את האלגוריתם שבנינו על הגרף. |
|  |
| * שליחת התשובה תתבצע בפונקציה הבאה: |
|  |
| צד לקוח:   * בדומה לשרת נפרסר את המידע שהתקבל בדגלים בעזרת getopt. |
|  |
| * נחלץ את כתובת הip של hostname באמצעות gethostname. * ניצור את הsocket שאיתו נתחבר לשרת. * ניצור את הstruct של הaddress ונתאים אותם כנדרש. * נבצע connect לאותו ip באותו port שהתקבלו באופציות. |
|  |
| * בשלב זה נקבל מהמשתמש את המטריצה בקלט של stdin ונמלא את המטריצה של הלקוח. * לאחר מכן נשלח אותה שורה אחר שורה בפונקציה הבאה |
|  |

**שאלה 7 :**

התבקשנו לבחור 4 אלגוריתמים והפעלתם על הגרף של הלקוח לפי בחירה, תוף שימוש בfactory ן strategy .

|  |
| --- |
| נבחרו האלגוריתמים הבאים:   * Mst * Max flow * Path cover * SCC   -יצרנו אובייקט אבסטרקטי בשם algorithms שכל האלגוריתמים ירשו ממנו ויממשו את הפונקציה הווירטואלית הטהורה בשם activate .  -כמו כן מימשנו factory ליצירת כל אובייקט תוך שימוש בפולימורפיזם בעזרת האובייקט האבסטרקטי.  -האובייקט strategy קיבל מצביע לאלגוריתם וקריאה לפונקציית execute שלו הפעיל את הactivate המתאימה של האלגוריתם הנכון.  בהמשך אציג רק את algorithms , factory , strategy ואת אופן השילוב בין שניהם ע"י השרת. |
|  |
|  |
|  |
|  |

**שאלה 8 :**

|  |
| --- |
| בשאלה זו התבקשנו לממש leader follower עם threads מרובים.   * נגדיר תור של משימות taskQueue שיכיל גרף וfd של socket . * נגדיר mutex להגנה על הגרף. * נגדיר condition variable לטובת הודעה לthread על הגעה של משימה חדשה. * נגדיר משתנה גלובלי אטומי שיצביע על פעילות השרת. * נגדיר פונקציה בשם workerFunction שתהיה הפונקציה שהthread מפעיל , בפונקציה זו נריץ את כלל האלגוריתמים על הגרף הנתון. |
|  |
|  |
| תיאור כרונולוגי של התהליך:   * האזנה של השרת יצירת הthreads והמתנה ב accept לconnect מלקוח. * בצד הלקוח שליחת ה connect , בחירת גרף רנדומלי או נתון והעברת הנתונים בהתאמה לשרת. * בצד השרת המידה ונבחר גרף רנדומלי מייצר גרף חדש לפי הנתונים שהתקבלו (קודקודים וצלעות), במידה והגיע גרף נתון מקבל אותו. * יצרת הstruct של המשימה החדשה שהתקבלה והכנסתה לתור המשימות, הפעלת notify של condition variable כדי שהthread הפנוי הבא ייקח את המשימה ויבצע אותה. * חישוב האלגוריתמים ושליחת תשובה ללקוח. |
| גרף רנדומלי: |
|  |
| הפרדה בין מטריצה נתונה לבין בחירה ברנדומלי: |
|  |
| הכנסת המשימה לתור והודעה בעזרת condition variable: |
|  |

**שאלה 9:**

ארבעת האובייקטים הפעילים יהיו אחד לכל אלגוריתם.

נציג את הנקודות המרכזיות בקוד:

|  |
| --- |
| הגדרות ראשוניות לכל thread:   * הגדרנו תור משימות * Condition variable * Mutex   הוספנו למבנה של ה task את התוצאה שנבנית בתהליך. |
|  |
| פיצלנו את הפונקציה הראשית (מתרגיל 8) שהייתה לכל thread ל4 פונקציות כך שכל סוג של thread קיבל אחת אחרת.  בכל פונקציה נבצע מספר פעולות:   * הגנה בעזרת mutex בזמן עבודה על התור המשותף. * הוצאת המשימה מהתור. * ביצוע המשימה בעזרת strategy ו factory (יצירת האובייקט שיבצע). * הכנסת התוצאה לresult של המשימה. * נעילה נוספת של התור של הthread הבא בpipe. * הכנסת המשימה לתור שלו. * הודעה ל CV שלו שיש משימה חדשה.   למשל הthread שאחראי לחישוב mst קיבל את הפונקציה הבאה |
|  |
| התאמה הThreads לפונקציות: |
|  |
| התחלת הpipline |
|  |