פרויקט מסכם תכנות מתקדם – תש"פ סמסטר קיץ

תמונת רמות אפור בגודל $N \times M$ הינה מערך דו-מימדי מטיפוס unsigned char בגודל N שורות ו-M עמודות. כל תא במערך מכונה **פיקסל** ומכיל מספר בתחום 0-255 אשר מייצג את גוון האפור כאשר הערך 0 מייצג את הצבע השחור, הערך 255 מייצג את הצבע הלבן, ויתר הערכים מייצגים גווני אפור בין שחור ללבן.

השכנים/הסביבה של פיקסל מסוים מוגדרת ע"י שמונת הפיקסלים אשר נמצאים סביבו. בדוגמא להלן, השכנים של Y,Z צבועים בתכלת.

נגדיר ***סגמנט*** כאוסף פיקסלים השכנים המידיים או העקיפים (שכנים של שכנים וכן הלאה) אשר השוני ברמות האפור של כל אחד מהם מרמת אפור של פיקסל נתון שקרוי ***גרעין הסגמנט (kernel)*** אינו עולה על סף נתון T. לדוגמא, האזור האפור בתמונה הבאה הוא סגמנט עבור סף T=10 וגרעין הסגמנט שבצבע אדום שבו נתונה רמת האפור 95.

	0							M-1
0	Z				101	100	103	
				97	87	100	105	110
					85	**95**	40	
				80	100	94	50	
					70	60		
		Y						
N-1								

על-מנת לייצג תמונה עושים שימוש בהגדרה הבאה:

```
typedef struct _grayImage {
  unsigned short        rows, cols;
  unsigned char         **pixels
} grayImage;
```

על-מנת לייצג מיקום בתמונה, עושים שימוש בהגדרה הבאה:

```
typedef unsigned short imgPos[2];
```

על-מנת לשמור את כל המיקומים של סגמנט, עושים שימוש בעץ שנתון במבנה הבא:

```
typedef struct _treeNode{
    imgPos              position;
    struct _treeNode  **similar_neighbors;  // מערך ילדים
} treeNode;

typedef  struct  _segment {
    treeNode            *root;
    unsigned int        size; /* the total number of tree nodes */
} Segment;
```

גרעין הסגמנט נתון בשורש העץ. השורש ומספר הצמתים בעץ נתונים במבנה **Segment**. כל צומת X בעץ מכיל מיקום של פיקסל P במשתנה **position**. המשתנה **similar_neighbors** של X מכיל מערך של מצביעים לילדים בעץ אשר כל אחד מהם מכיל מיקום ששייך לאחד השכנים בסביבה של הפיקסל P לפי הגדרת הסגמנט. גודל המערך אינו קבוע ותלוי במספר השכנים של **position** שמוכלים בסגמנט ואשר לא קיימים כבר בעץ. כדי לסמן את סוף המערך **similar_neighbors**, התא האחרון שבו מכיל NULL. לדוגמא, אם ל-X יש 5 ילדים, גודל המערך **similar_neighbors** יהיה 6, כאשר התא האחרון יכיל NULL.

סעיף 1

בסעיף זה יש לכתוב את הפונקציה:

Segment *findSingleSegment(grayImage *img, imgPos kernel, unsigned char threshold)

הפונקציה מוצאת סגמנט ע"י סריקת התמונה החל מהמיקום **kernel**. הפונקציה תבדוק את ערכי האפור של שמונת השכנים של **kernel**. השכנים אשר הפרש רמת האפור ביניהם לזו של **kernel** קטנה או שווה ל-**threshold** יוכנסו לעץ כילדיו של **kernel**. התהליך ימשיך בצורה רקורסיבית מהילדים. בשורש העץ יהיה המיקום **kernel**. הטיפוס **unsigned char** של המשתנה **threshold** עושה שימוש בתכונותיו הנומריות של טיפוס זה אשר יכול להכיל ערך מ-0 עד 255 (כך גם בסעיפים 2 ו-5).
שימו לב: כל מיקום בסגמנט צריך להופיע בעץ **בדיוק פעם אחת**.

סעיף 2

נתונה ההגדרה הבאה לאחסון מיקומי הפיקסלים בסגמנט מסוים בעזרת רשימה מקושרת:

```
typedef struct _imgPosCell {
    imgPos                  position;
    struct _imgPosCell      *next, *prev;
} imgPosCell;
```

בסעיף זה יש לכתוב את הפונקציה:

unsigned int findAllSegments(grayImage *img, unsigned char threshold, imgPosCell ***segments)

הפונקציה מוצאת את כל הסגמנטים בתמונה **img** ומחזירה אותם במשתנה הפלט **segments** באופן הבא: כל תא במערך יצביע על התא הראשון ברשימת המיקומים בסגמנט מסוים (זוהי רשימה ללא tail וללא size). הערך המוחזר של הפונקציה יהיה גודל המערך.
על-מנת למצוא את כל הסגמנטים יש להתחיל מהמיקום שמכיל את הערך האפור המינימלי בתמונה, להגדירו כ-**kernel**, ולמצוא את הסגמנט המתחיל ממנו. אח"כ יש לבחור את המיקום שמכיל את הערך האפור המינימלי מאלה שאינם שייכים לסגמנט שכבר מצאנו, להגדירו כ-**kernel**, ולמצוא את הסגמנט שמתחיל ממנו. כך יש להמשיך עד אשר כל פיקסל בתמונה ישתייך לסגמנט מסוים.
על המערך המוחזר לקיים את התכונות הבאות:
א. רשימת מיקומי סגמנט צריכה להיות ממוינת בסדר עולה לפי מיקומי העמודות של הפיקסלים שבסגמנט. אם ישנם שני מיקומים עם אותו מספרי עמודה, יש למיין אותם לפי מיקום השורה שלהם.

ב. המערך **segments** צריך להיות ממוין בסדר יורד על פי גדלי הסגמנטים.

סעיף 3

בסעיף זה יש לייצר תמונה שבה הפיקסלים בכל סגמנט יכילו את אותה רמת אפור. לכל סגמנט יוקצה גוון ספציפי. את רמת האפור של כל סגמנט יש לקבוע בהתאם לכמות הסגמנטים N באופן הבא: ערך האפור של הסגמנט ה-i (מספרי הסגמנטים מתחילים מאפס) יהיה $\left\lfloor i * \frac{255}{N-1} \right\rfloor$. לדוגמא, אם ישנם שלושה סגמנטים, ערך האפור של הראשון יהיה 0, של השני יהיה 127 ושל השלישי 255. הפונקציה תקבל מערך של רשימות סגמנטים כמו זה שיוצרים בסעיף 2. הפונקציה תחזיר תמונה שכל פיקסל מכיל את גוון הסגמנט שהותאם לו.

grayImage *colorSegments(grayImage *img, imgPosCell **segments, unsigned int size)

המשתנה *size* מכיל את גודל המערך **segments**. ניתן להניח שכל פיקסל מופיע בדיוק פעם אחת במערך רשימות הסגמנטים.

סעיף 4 (10 נקודות)

כתבו את הפונקציה:

grayImage *readPGM(char *fname)

הפונקציה מקבלת שם של קובץ טקסט אשר מכיל תמונה בפורמט PGM. הפונקציה תיקרא את הקובץ ותחזיר משתנה מטיפוס **grayImage** אשר יכיל את התמונה.
יש להכיר את מבנה קובץ ה- PGM מ-wikipedia. זהו מבנה פשוט שמצריך מספר דקות להבנתו.

סעיף 5

הוחלט לשמור באופן חסכוני תמונת רמות אפור. לשם כך קובעים מספר רמות אפור Z קטן מ-255, ומשנים את כל ערכי הפיקסלים כך שיהיו בטווח החדש 0-Z. ניתן להניח ש-Z הוא חזקה של 2. לדוגמא, אם Z=32 אזי ערכי הפיקסלים יהיו בטווח 0-31. בדוגמא זו, פיקסל שערכו 255 ישונה ל-31, פיקסל שערכו 192 ישונה ל-24 וכן הלאה. יש לממש את הפונקציה הבאה:

void saveCompressed(char *file_name, grayImage *image, unsigned char reduced_gray_levels)

הפונקציה תשמור את התמונה עם טווח רמות האפור המוקטן בקובץ בינארי באופן הבא:

בתחילת הקובץ יהיו שני משתנים מסוג unsigned short, הראשון יכיל את מספר השורות והשני את מספר העמודות של התמונה. אח"כ יהיה בית אשר יכיל את מספר רמות האפור המוקטן **reduced_gray_levels**. אח"כ, יישמרו הפיקסלים **ברצף** שורה אחר שורה כאשר כל פיקסל מיוצג ע"י מספר הביטים המינימלי הנדרש. בדוגמא לעיל, Z=32, כל פיקסל יישמר בעזרת חמישה ביטים כאשר *אין להשאיר בכל בית שלושה* ביטים מאופסים אלא לשמור את הפיקסלים ברצף אחד אחר השני. בהחלט יתכן שפיקסל יתחיל בבית אחד ויסתיים בבית שאחריו.

לדוגמא, נתונה התמונה בגודל 10 שורות על 12 עמודות שאותה אנו מכווצים ל-32 רמות אפור. נניח כי ערכי שלושת הפיקסלים הראשונים אחרי הכיווץ בשורה הראשונה בעמודות מספר 0, 1, ו- 2 הם 13, 24, 7, בהתאמה, אזי תחילת הקובץ הבינארי תיראה כך (הרווחים אינם מופיעים בקובץ והם לנוחיות התצוגה בלבד). שימו לב כי 3 הפיקסלים יהיו מוכלים ב-2 הבתים שאחרי הבית שמכיל את כמות רמות האפור (הביט השמאלי בכל בית הוא ה-MSB והביט הימני בכל בית הוא ה-LSB):

$$\underbrace{00001010\ 00000000}_{10\ lines}\ \underbrace{00001100\ 00000000}_{12\ columns}\ \underbrace{0100000}_{32\ gray\ levels}\ \underbrace{01101}_{13}\ \underbrace{11000}_{24}\ \underbrace{00111}_{7}\ \ldots$$

במקרה ש-M*N אינו מתחלק ב-**max_gray_level** עם שארית אפס, יש להציב אפסים בפיקסלים העודפים בבית האחרון בקובץ.

סעיף 6

בסעיף זה יש לכתוב את הפונקציה:

```
void convertCompressedImageToPGM(
        char *compressed_file_name, char *pgm_file_name)
```

הפונקציה מקבלת שם של קובץ בינארי עם תמונה דחוסה שיצרתם בסעיף 5 ומייצר ממנה קובץ PGM.
שימו לב שכעת יש להתאים לכל ערך אפור מכווץ ערך בטווח 0-255 ע"י פעולה הפוכה לזו שעשיתם בסעיף 5.

הערה: קבצי PGM ניתנים להצגה ע"י כל תוכנה לקריאת תמונות, לדוגמא, Irfanview החינמית.

הנחיות כלליות

- יש לכתוב קוד יעיל ככל האפשר הן מבחינת זמן ריצה והן מבחינת צריכת זיכרון. יש להקפיד על סיבוכיות פרקטית ואסימפטוטית נמוכות.
- יש לשחרר זיכרון כאשר אין בו צורך.
- יש לסגור כל קובץ לאחר סיום העבודה איתו.
- יש לבדוק הצלחת הקצאת זיכרון דינאמי.
- יש לבדוק הצלחת פתיחת קבצים.
- יש לתעד את הקוד כדי לאפשר הבנה מהירה שלו בזמן הבדיקה.

הנחיות הגשה

- על הפרויקט להכיל **לפחות** שלושה מודולים - כפי שראינו, כל מודול מורכב מקובץ c. וקובץ h.
- יש להגיש קובץ טקסט פשוט בשם README ובו שמות המגישים (באנגלית) ומספרי ת.ז שלהם.
- יש ליצר main ופונקציות בדיקה משלכם אך **אין להגיש אותן**. יש להגיש אך ורק קובץ ZIP שיכיל את קבצי ה-h וקבצי ה-c.
- יש לעשות את הפרויקט בזוגות שכן אחת המטרות היא לתרגל עבודה בצוות, כלומר, חלוקת העבודה כך ששני חברי הצוות יעבדו במקביל ואז שילוב המודולים שנכתבו בדומה לעבודה בפרויקטים בתעשיה. במקרים חריגים תיבחן האפשרות לעשות את הפרויקט לבד.
- הפרויקט מותאם בהיקפו לזוג ולא ניתן לעשותו בשלשה ללא יוצא מן הכלל.