Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

Vector Quantization Compression

LBG based algorithm

מבוא

מהות הרעיון של Vector Quantization (כימות וקטורי) היא לקחת את נקודות המידע בגרף, למצוא אוסף נקודות היכולות לייצג את כלל נקודות המידע בקירוב, ולהתאים את נקודות המידע המקוריות לנקודות המידע מהאוסף.

באופן מוחשי יותר, במקרה של תמונת RGB, כל פיקסל מיוצג על ידי על ידי 3 בתים. בית עבור כל ממד צבע (אדום, ירוק, כחול). במידה ונרצה לייצג את כמות הבתים, באופן תאורתי (ללא דחיסה מתקדמת לדוגמת JPEG), אנו יכולים לייצג את המידע להרכבת התמונה על ידי 3 בתים עבור כל פיקסל או לעשות מילון עבור הפיקסלים הקיימים ולצמצם בסגנון הופמן (דבר שעדיין ידרוש מקום רב עבור תמונות רבות גוונים). אולם, אם נייצג קירוב של התמונה במספר מצומצם של צבעים, אנו יכולים לייצג כל פיקסל על ידי כמות קטנה יותר של ביטים. ביט אחד עבור שני צבעים, 2 ביטים עבור ארבע צבעים, 3 ביטים עבור שמונה צבעים וכן הלאה...

בפרויקט נעשה שימוש ב:

- מחלקת BufferedImage של ג'אווה על מנת להשיג את ערכי הפיקסלים, ממספר סוגים של קבצי תמונה.
- ווריאציה של Voronoi Regions וווריאציה שלsplitting technique עבור מציאת "נקודות אימון"* התחלתיות.
- KDtree עבור מציאת הנקודה הקרובה ביותר לכל פיקסל.
- windowBuilder עבור הבסיס לממשק.

בנוסף, הפרויקט התמקד לא רק בתמונות RGB, אלה בדחיסה של גרף ונקודות מידע במרחב רב ממדי, RGBA לדוגמא.

*נקודות אימון: נקודות התחלתיות שיהוו מרכז ראשוני למחיצות, ככל שנקודות אלה יותר מדויקות וקרובות למרכזי מקבצים, כך כמות האיטרציות בשלב הסופי תצטמצם ואף עשויה להביא לקירוב טוב יותר עבור הנקודות (עיוות קטן יותר).

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

תוכן עניינים

רעיון אלגוריתם Linde–Buzo–Gray

1. מציאת נקודת אימון התחלתית.
2. פיצול נקודת האימון, מציאת פיקסל קרוב ביותר לכל נקודה, ומרכוז נקודות האימון בהתאם לפיקסלים שלהן.
3. חזרה על שלב שתיים עבור כלל נקודות האימון עד אשר קיים אוסף בגודל המתאים.
4. איטרציות לשיפור מיקום הנקודות עבור הקטנת העיוות, עד אשר השיפור בין האיטרציה הקודמת לאיטרציה הנוכחית קטן לגודל שנקבע מראש להפסקת פעולת האלגוריתם.

מה עשינו בפועל

התוכנה בפועל:

1. מקבלת תמונה בעזרת מחלקת הBuffered Image של ג'אווה.
2. התמונה מומרת למערך פיקסלים של RGB או RGBA.
3. המערך מומר למערך כללי לעבודה במרחב רב ממדי.
4. על ידי שימוש במחלקת הEncoder, אנו מחזירים מערך של פיקסלים כלליים המייצגים את התמונה הדחוסה.
5. בDecoder אנו מחזירים את הפיקסלים ממחלקה כללית לRGB או RGBA.
6. התמונה נשמרת חזרה במקום אותו קבע המשתמש במחשב

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

שלבי ואופן פעולת האלגוריתם שלנו

1. קבלת אוסף נקודות(פיקסלים) וערכיהם. O(N) N-כמות הפיקסלים
2. שימוש בVoronoi regions למציאת אוסף "נקודות האימון התחלתיות" עד אשר לא ניתן לפצל עוד את הגרף (נמצאו כל הגוונים) או עד אשר הגענו לנקודה בה פיצול הגרף יכול להניב יותר נקודות מהנדרש (יותר מ2 בחזקת מספר הממדים). O(Log2(A)*N) A-גודל ערכי הצירים
3. שימוש בsplitting technique למציאת שארית "נקודות האימון". O(n*2^D) D-מספר הממדים n-מספר הפיקסלים במחיצה נוכחית.
4. שימוש בKDtree למציאת "נקודת האימון" הקרובה ביותר עבור כל פיקסל, ולאחר מכן עבור כל איטרציה עד אשר מגיעים לעיוות מינימאלי. O(log2(K)*N) K-מספר המחיצות.

מבני הנתונים

Pixel:

מחלקת Pixel משמשת לקבלת ערכי נקודה כללית רב ממדית. המחלקות PixelRGB וPixelRGBA הן מחלקות עזר לקבלת ערכי תמונה ספציפיים לפני המרתם לPixel כללי.

Partition:

במחלקה זאת המשתנים העיקריים הם רשימת הפיקסלים במחיצה וK המחיצה (K הינו ממוצע כל הפיקסלים במחיצה, הCentroid). בנוסף במחלקה פעולות ומשתנים חשובים נוספים, כגון חישוב הDistortion עבור הפיקסלים במחיצה, שמירת התת עץ של אזור Veronoi בו המחיצה נמצאת, ומיקום המחיצה במערך מסוים (עבור חיפושים של הKDtree).

Sphere:

מחלקה המשמשת לחישוב הווקטור אליו תתפצל נקודת האימון אותה אנו רוצים לפצל. המשתנה העיקרי במחלקה הוא מערך הSpheres הפוטנציאליות לפיצול, המדמה מערך רב ממדי. גודלו יהיה שלוש

בחזקת מספר הממדים. המחלקה מהווה אמצעי משני לפיצול נקודות האימון.

Voronoi Tree:

המחלקה העיקרית למציאת נקודות אימון התחלתיות באופן מהיר יותר. המחלקה מורכבת מערכי ההתחלה והסיום של האזור הנוכחי, וכן מערך של אזורים פוטנציאליים אליהם יכול להתפצל. שניים בחזקת מספר הממדים.

KDtree:

עץ חיפוש בינארי רב ממדי עבור פתרון בעיית השכן הכי קרוב בזמן log2(K)*N. במחלקת העץ קיימות הפונקציות לבניית עץ על ידי KDnodes, ופונקציות לחיפוש ומציאת הNode הקרוב ביותר.

Knode:

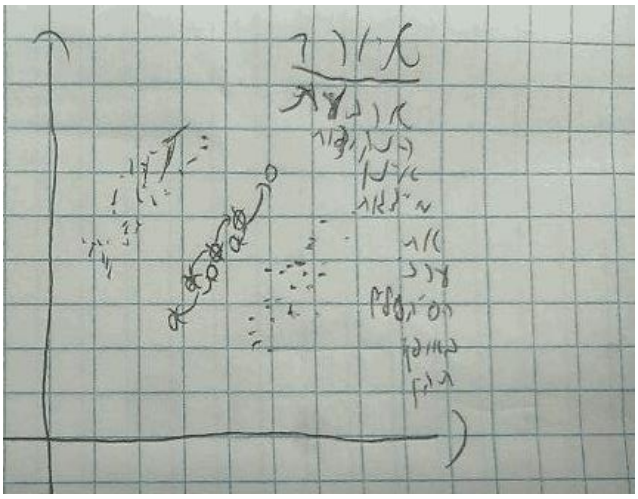
מחלקה עבור תת עץ בKDtree, במחלקה קיימים מצביעים לשתי Knode נוספים (ימין ושמאל), והמחיצה (הפנייה למחיצה), בתוכה הK עם הערך הדרוש לסידור ומציאת המחיצה בעץ.

<u>splitting technique (Greedy sphere variation)</u>

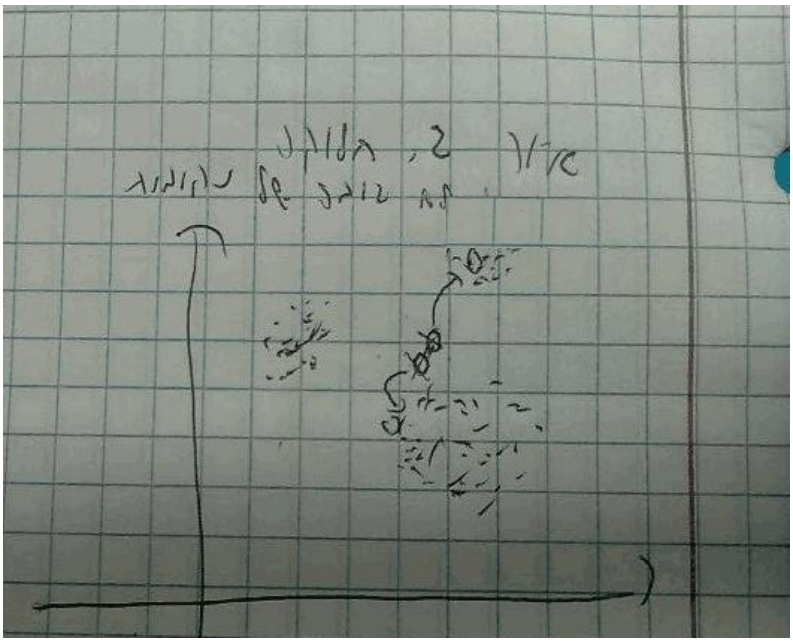
האלגוריתם LBG המקורי פעל בהתחלה מנקודת אימון אחת, ופיצולה על פי וקטור קבוע עד להגעת מספר נקודות אימון רצוי.

בניסיון הראשון למימוש, הלכנו בכיוון זה, והשתמשנו בוקטור קבוע המוסיף 1 לכל ציר. ניסיון זה לא היה מוצלח עקב מספר בעיות. הראשונה הייתה שאם נוצר מצב שהנקודה נפלה בין שני מקבצי פיקסלים הנמצאים משני צדדי הנקודה, אזי עשוי להיווצר מצב שהנקודות המייצגות יהיו בין המקבצים ולא ייצגו את הצבעים כראוי. (ראה איור 1).

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217



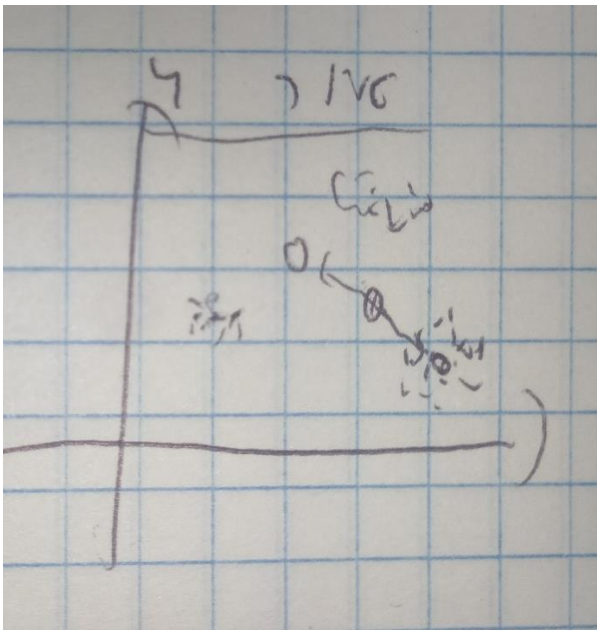
בעיה נוספת היא שיכול להיות מצב שמאוחדים מקבץ מאוד גדול מאוד עם מקבץ גדול אחר תחת צבע אחד (נקודה אחת), ומקבץ פחות גדול מקבל נקודת ייצוג שלמה, אף על פי שעדיף היה לתת נקודת ייצוג למקבץ הגדול מאוד ולאחד בין השניים האחרים (ראה איור 2)

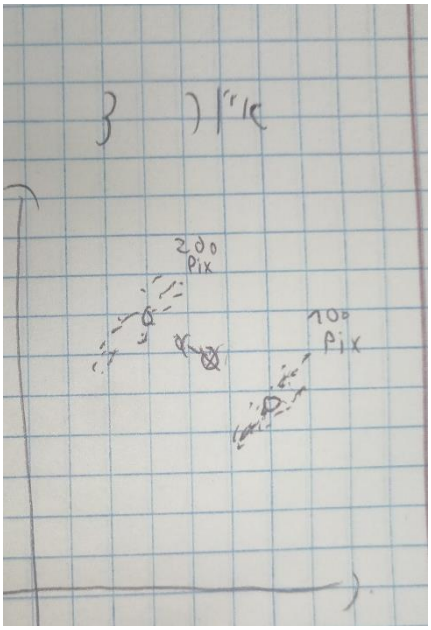


הפתרון הראשון שמצאנו ועבד, בו נעשה שימוש בגרסאות הראשונות של הפרויקט, הינו ווריאציה של הsplitting technique שהחלטנו לקרוא לה greedy sphere מכיוון שהינה מבצעת פתרון מקומי, עבור הגעה לפתרון לבעיה (אם כי לא בהכרח אופטימלי). החיסרון הכי משמעותי הוא שהשיטה דורשת זמן ריצה יקר.

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

עיקר הרעיון הוא למצוא את הכיוון בו נמצאים הכי הרבה פיקסלים, ולפצל את הנקודה העכשווית לכיוונם של הפיקסלים הרבים. הפתרון עוזר משמעותית למנוע את בעיית האיגוד של מקבצים באופן לא יעיל, ואת התופעה שנקודות יכולות להיתקע בין שני מקבצים. (ראה איורים 3 ו4).





זה נעשה על ידי עבודה בבסיס ספירה 3 למציאת ערכי הוקטור להזזת הsphere החדשה עבור כל ממד (1+\0\1-), ככה שהפיצול יהיה לכיוון איפה שקיימים מרבית הפיקסלים.

על ידי עבודה בבסיס ספירה 3, אנו יכולים לעבוד במרחב רב ממדי במערך מממד אחד. דוגמא עבור שלושה ממדים:

[0]=-1,-1,-1

[1]=-1,-1,0

[2]=-1,-1,1

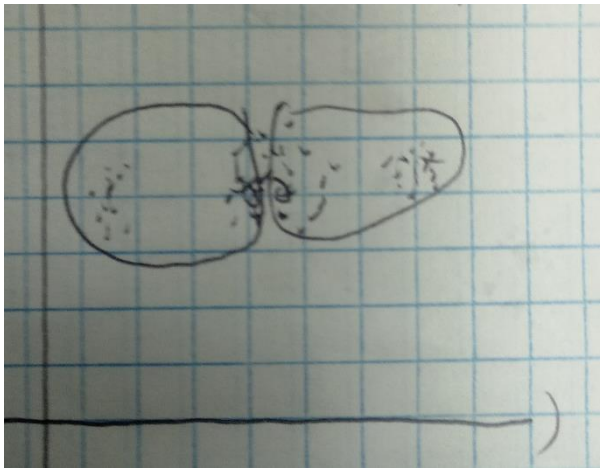
[3]=-1,0,-1

וכן הלאה.

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

על ידי חזקת 3 הקודמת לציר הנוכחי ו2 כפול החזקה הנוכחית ניתן ל"טייל" על כל ציר. על ידי הוספת כיוון כל פיקסל לוקטורים המתאימים, ולאחר מכן בחירת הוקטור עם הכי הרבה פיקסלים בכיוונו שאינו וקטור ה0, אנו יכולים למצוא את הK החדש עבור המחיצה החדשה.

חיסרון בפתרון זה הוא שעדיין קיימות סיטואציות בו יכול להיבחר כיוון לא נכון לפיצול, למשל במקרה בו נתקעים בין שני מקבצים מאוד שווים בכמות הפיקסלים, או אם הנקודה מפוצלת מאמצע מקבץ לכיוון מקבץ גדול יותר מאחוריו (ראה איור 5). בנוסף מכיוון ואנו מפצלים על בסיס הפיקסלים הקיימים במחיצה, יכול להיווצר מצב שנקודה פוצלה מאוד קרוב לנקודה הקיימת במחיצה אחרת.



מכיוון שפתרון הgreedy sphere אינו אופטימלי, ומאוד אינטנסיבי N*log2(K)*3^d כאשר d הינו מספר הממדים, אז על אף שמביא תוצאות טובות החלטנו למצוא ולהשתמש בפתרון יעיל יותר (Voronoi regions), אך הקוד קיים, ונעשה בו שימוש במקרה בו אחת המחיצות איבדה את כל הפיקסלים באחת האיטרציות לתיקון עיוות ואנו מפצלים את המחיצה הנוכחית הגדולה ביותר הניתנת לפיצול, או כאשר פיצול נוסף של הVoronoi tree עשוי להביא למחיצות נעבר לנדרש.

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

Voronoi regions

בפרויקט עשינו ווריאציה של Voronoi region במבנה נתונים של עץ.

מתחילים ממחיצה אחת של כל הגרף, ומפצלים כל פעם את המחיצה הגדולה ביותר לתת אזורי Voronoi עד אשר מגיעים לפיקסל בודד, או עד אשר הגענו לכמות מחיצות שפיצול נוסף עשוי להביא יותר מחיצות מהנדרש (כמות המחיצות הרצויות פחות חזקת 2 במספר הממדים).

באופן רב ממדי, כמות האזורים מוכפלת ב2 עבור כל ממד

ממד אחד: 2

שני ממדים: 4

שלושה ממדים: 8

ארבעה ממדים: 16

וכן הלאה.

במימוש, אופן העבודה הוא שכל עץ הוא אזור, אליו משויכת מחיצה המכילה את כל הפיקסלים באותו אזור.

שלבי הפעולה:

1. מתחילים מאזור אחד המכיל את כל הגרף.
2. מוצאים את האזור בעל כמות הפיקסלים הגדולה ביותר, ומכינים את הערכים לפיצול האזור (כל ממד חלקי שניים).
3. עוברים על כל פיקסל, ושמים אותו בעץ הרלוונטי אליו. אם העץ עדיין לא נבנה בונים אותו, ופותחים מחיצה חדשה. (יכול להיות שבתת אזור לא יהיו פיקסלים כלל, ולו לא תבנה מחיצה כלל).
4. חוזרים על שלבים 2 ו3 עד אשר מגיעים לכמות נקודות אימון שפיצול נוסף עשוי להביא ליותר נקודות ממה שאנו נדרשים.
5. לאחר הפסקת הפעולה, מפצלים את המחיצות הגדולות ביותר שנשארו בGreedy sphere technique ומגיעים לכמות המחיצות הרצויה.

לפתרון זה יש יתרון משמעותי, וחיסרון משמעותי.

היתרון הוא זמן הריצה שבמקרה הגרוע ביותר הינו O(N*Log2(A)) כאשר A הוא גודל האזורים בגרף.

החיסרון הוא שאם קיימים פיקסלים מעטים בקצוות האזור, אזי התת אזור יקבל נקודת אימון משלו, אף על פי כמות הפיקסלים המעטה. הפתרון שלנו המפצל את האזור הגדול ביותר עוזר להפחית את ההשפעה, כי אזור המתפצל לרוב יהיה עם כמות פיקסלים רבה מראש, כך שאזורים אלה לרוב בקצוות מקבצים גדולים.

KDtree

לאחר מציאת נקודות האימון ההתחלתיות אנו צריכים למצוא את הK (נקודת האימון) הקרוב ביותר עבור כל Pixel. הסיבה לכך היא שבמהלך פיצול המחיצות וחישוב ממוצע K, חלק מהפיקסלים עכשיו קרובים יותר לממוצע מחיצה אחרת, מהמחיצה בה הם נמצאים. בנוסף לאחר כל תיקון, ממוצעי המחיצות משתנים מחדש, ולכן פיקסלים נוספים עשויים להיות קרובים יותר לממוצע המחיצה, וזאתי הסיבה לאיטרציות. האיטרציות ימשיכו עד אשר השיפור בעיוות שאנו מקבלים בין איטרציה לאיטרציה קטן לגודל מסוים.

את השיפור ניתן לחשב בעזרת הנוסחה $((D_k-D_{k-1})/D_k)*100$ לקבלת אחוז השיפור. התוכנה מפסיקה כאשר מגיעים לגודל שיפור לפי בקשת המשתמש.

בתחילת הפרויקט השתמשנו בתכנון דינאמי מסיבי השומר מערך רב ממדי עבור כל מרחב הצבעים, ובכל תא נשמר הK הקרוב ביותר לערך הפיקסל(הגוון) לאחר שנמצא בפעם הראשונה, או 1- אם עדיין לא נמצא. ככה שהזמן ריצה הוא O(C*K + (N-C)) כאשר C הוא מגוון הצבעים בתמונה K הוא כמות המחיצות וN הוא כמות הפיקסלים.

פתרון זה עבד מאוד מהר, אבל מאוד אינו פרקטי בגרפים רב ממדיים כי עבור כל ממד אנו צריכים להכפיל את גודל המערך בגודל הממד החדש שנוסף לו. במרחב הצבעים מודבר ב256 בשלישית עבור RGB ו256 ברביעית עבור RGBA (4,294,967,296 תאים).

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

כמובן שמאוד נעדיף לא לבצע את החישוב ללא אלגוריתם הגיוני עקב סיבוכיות O(N*K) עבור כל איטרציה.

לאחר חשיבה רבה, וחיפושים נרחבים מצאנו את המבנה נתונים KDtree לחיפוש בינארי במרחב רב ממדי המוריד את זמן הריצה משמעותית לממוצע O(log(K)*N).

רעיון האלגוריתם הוא להשתמש בעץ בינארי, שכל תת עץ מחלק את אזורו לשני חלקים על פי הציר אחרי הציר של הK ממנו הגיע. בסופו של דבר הגרף נתחם לאזורים מרובעים. עבור גרף דו ממדי כל חלוקה תפרוש קו בגרף, עבור שלושה ממדים כל חלוקה תפרוש קיר, עבור ארבעה ממדים כל חלוקה תשים מרובע תלת ממדי בתוך גרף ארבע ממדי, וכן הלאה. לאחר בניית העץ עושים חיפוש השכן הקרוב ביותר עבור כל פיקסל, וככה מוצאים את הK הקרוב ביותר לכל פיקסל.

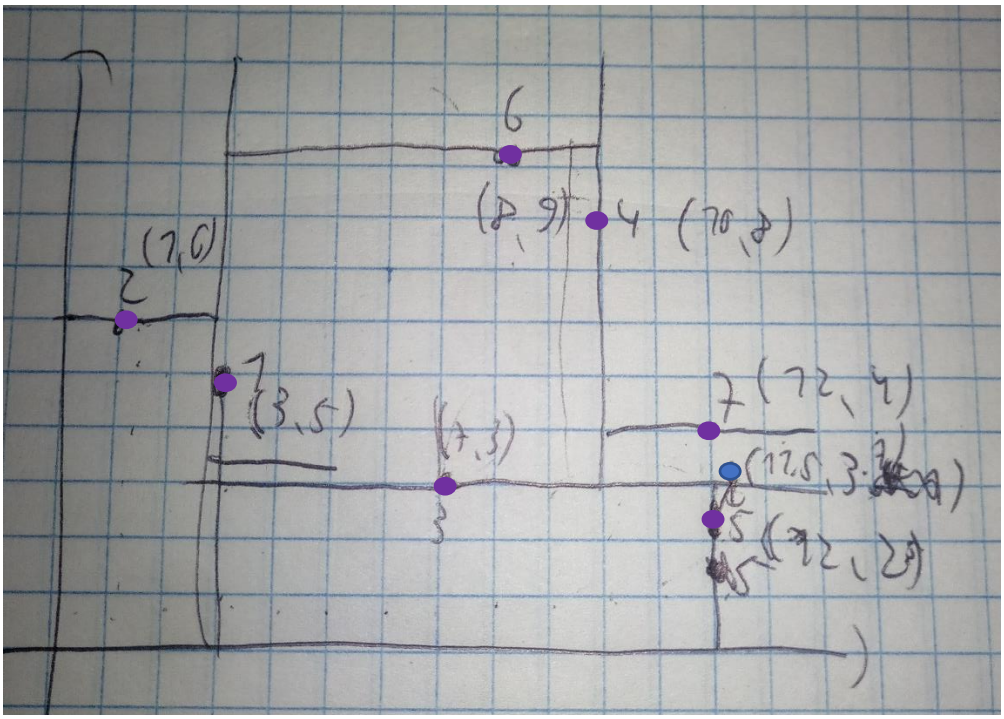
חיפוש השכן הקרוב ביותר:
הבעיה באזורים מרובעים היא שפיקסל הנמצא באזור מרובע מסוים עשוי להיות רחוק מהK שתוחם אותו וקרוב לK שכן. אנו נדרשים לחשב תתי עצים שכנים, עד אשר על פי המרחק המינימאלי עד כה שנמצא לפיקסל רחוק מידי מתתי עצים אחרים. המרחק המינימאלי מתחיל מהK התוחם, ומשתנה אם נמצא K בתת עץ אחר הקרוב יותר. כאשר לא קיימים תתי עצים פוטנציאליים נוספים, הK הקרוב ביותר שנמצא עד כה הוא השכן הקרוב ביותר.

בוריאציה בה השתמשנו, מתחילים את החיפוש ממציאת הנקודה התוחמת את האזור בו נמצא הפיקסל. לאחר מכן מבצעים חיפוש במעלה העץ, שעבור כל הורה, בודקים את תתי העצים תחתיו. ממשיכים זאת עד אשר מגיעים לשורש המקורי, או לאחר שהתרחקנו מאוד מהפיקסל, ולא משנה איזה ציר נבחר, לא נוכל להתקרב אליו.

בוריאציה הנוכחית, ההתרחקות מהפיקסל עובדת במציאת גבול לחיפוש. הרעיון הוא שכאשר המרחק בין אחד הצירים של הנקודה הנוכחית אל הפיקסל, לבדו גדול יותר מהמרחק המינימאלי שנמצא עד כה לפיקסל, אזי אנו יודעים בוודאות ששום נקודה מכיוונה השני

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

אינה יכולה לקרב אותנו לפיקסל. לכל ציר יש שתי נקודות כאלו, אחת הגדולה מספרית מהנקודה, ואחת קטנה מספרית מהנקודה. כאשר הגענו למצב בו כל ציר נפסל משני צדדיו, אזי אין נקודה נוספת היכולה להיות קרובה לפיקסל אותו אנו מחפשים. באיור מטה (איור 6) ניתן לראות מערכת צירים על גרף דו ממדי.



במערכת זאת נקודה 3 תפסול את ציר הX משמאל ונקודה 4 תפסול את ציר הY מלמעלה. מכיוון שאין נקודות מימין לנקודה או כאלו שייפסלו אותה מלמעלה (7 אינה רחוקה מספיק בשביל לפסול), אזי האלגוריתם ירוץ (עד נקודה 1). סך הכל על נקודות 1,7,3,5,4 מכיוון ש6, ו2 ייפסלו עקב המרחק הצירים של 4 ו1 מהמרחק של הפיקסל מ5. אם הייתה מערכת צירים גדולה יותר והיו נקודות גם מימין אשר היו פוסלות את האפשרות לנקודה קרובה יותר שם, אזי האלגוריתם היה מפסיק לאחר שכל ארבעת הגבולות היו נפסלים.

הנקודה אותה האלגוריתם מחזיר היא הנקודה האחרונה שמצאנו כקרובה ביותר, ובהכרח גם הקרובה ביותר לפיקסל.

במימוש שלנו עקב חוסר זמן, נאלצנו להיעזר במימוש מאתר
https://www.sanfoundry.com/java-program-find-nearest-neighbour-using-k-d-tree-search/

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

נתוני דחיסה

מפעולת האלגוריתם למדנו שהוא דוחס מאוד טוב קבצי "PNG" ו"GIF" אך בקבצי JPEG האלגוריתם דוחס מעט מאוד ואפילו מגדיל את גודל התמונה.

להערכתנו הסיבה לכך היא ש"JPEG" דוחס באיבוד מידע, ובעזרת נוסחאות הנעזרות בהפרשים בין הפיקסלים, ושינוי ערכי הפיקסלים עלול להשפיע על יכולת הדחיסה באופן שלילי. אולם "PNG" ו"GIF" דוחסים ללא איבוד מידע, ובהם הפחתת כמות הגוונים בתמונה מקטינה משמעותית את גודלה. בעמוד הבא ניתן לראות דוגמה ויכולת דחיסה עבור תמונה.

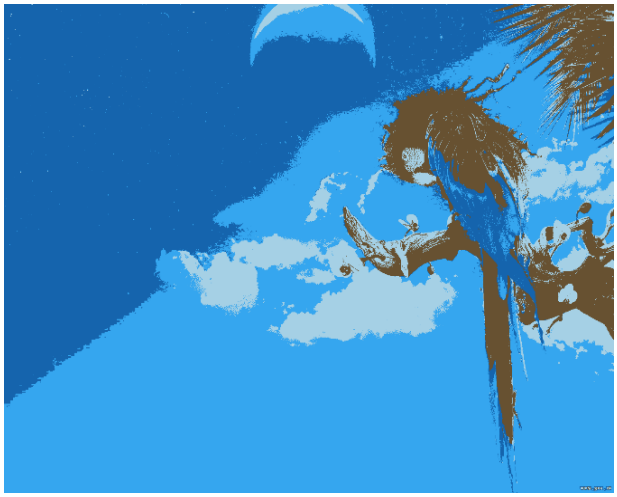
דוגמא לדחיסה וזמני ריצה בשיפור עיוות 0.01

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

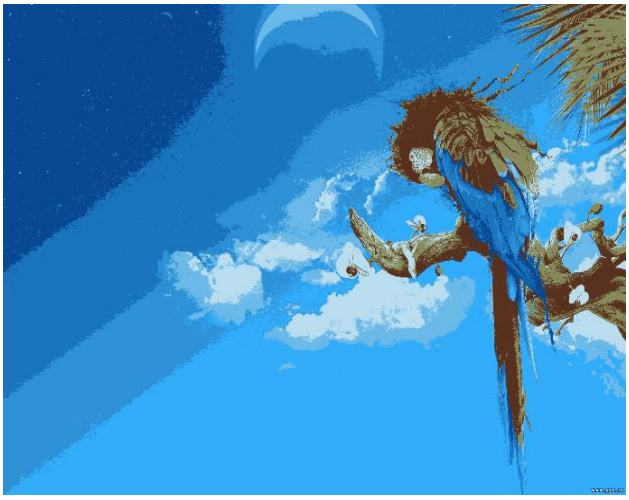
מקורי

2 מחיצות

4 מחיצות



8 מחיצות



32 מחיצות

256 מחיצות

נתוני דחיסה:

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

המרה מקובץ JPG לקובץ PNG

Distortion	0.01
זמן דחיסה עבור 2 מחיצות (גוונים)/ גודל	1.69 שניות/ 64.3KB
זמן דחיסה עבור 4 מחיצות (גוונים)	4.208 שניות/ 106KB
זמן דחיסה עבור 8 מחיצות (גוונים)	6.506 שניות/ 194KB
זמן דחיסה עבור 16 מחיצות (גוונים)	5.034 שניות/ 229KB
זמן דחיסה עבור 32 מחיצות (גוונים)	8.252 שניות/ 347KB
זמן דחיסה עבור 64 מחיצות (גוונים)	13.206 שניות/ 486KB
זמן דחיסה עבור 128 מחיצות (גוונים)	11.324 שניות/ 658KB
זמן דחיסה עבור 256 מחיצות (גוונים)	15.036 שניות/ 820KB
זמן דחיסה עבור 512 מחיצות (גוונים)	13.562 שניות/ 0.99MB
זמן דחיסה עבור 1024 מחיצות (גוונים)	11.342 שניות/ 1.19MB
גודל תמונה מקורי	2.48MB

Distortion	0.1
זמן דחיסה עבור 2 מחיצות (גוונים)/ גודל	1.949 שניות/ 64.3KB
זמן דחיסה עבור 4 מחיצות (גוונים)	3.525 שניות/ 122KB
זמן דחיסה עבור 8 מחיצות (גוונים)	3.7 שניות/ 178KB
זמן דחיסה עבור 16 מחיצות (גוונים)	2.604 שניות/ 232KB
זמן דחיסה עבור 32 מחיצות (גוונים)	2.789 שניות/ 346KB
זמן דחיסה עבור 64 מחיצות (גוונים)	3.144 שניות/ 450KB
זמן דחיסה עבור 128 מחיצות (גוונים)	3.717 שניות/ 637KB
זמן דחיסה עבור 256 מחיצות (גוונים)	4.230 שניות/ 803KB
זמן דחיסה עבור 512 מחיצות (גוונים)	4.892 שניות/ 0.99MB
זמן דחיסה עבור 1024 מחיצות (גוונים)	5.406 שניות/ 1.2MB
גודל תמונה מקורי	2.48MB

Distortion	100
זמן דחיסה עבור 2 מחיצות (גוונים)/ גודל	1.379 שניות/ 60.5KB
זמן דחיסה עבור 4 מחיצות (גוונים)	1.675 שניות/ 145KB
זמן דחיסה עבור 8 מחיצות (גוונים)	1.673 שניות/ 182KB
זמן דחיסה עבור 16 מחיצות (גוונים)	1.565 שניות/ 223KB
זמן דחיסה עבור 32 מחיצות (גוונים)	1.655 שניות/ 337KB
זמן דחיסה עבור 64 מחיצות (גוונים)	1.651 שניות/ 428KB
זמן דחיסה עבור 128 מחיצות (גוונים)	2.167 שניות/ 617KB
זמן דחיסה עבור 256 מחיצות (גוונים)	2.543 שניות/ 780KB
זמן דחיסה עבור 512 מחיצות (גוונים)	2.53 שניות/ 0.98MB
זמן דחיסה עבור 1024 מחיצות (גוונים)	4.223 שניות/ 1.2MB
גודל תמונה מקורי	2.48MB

דחיסה מקובץ JPG לקובץ JPG

Dean Moravia, 302491741
Hana Razilov, 311784722
Aviya Shimon, 302813217

Distortion	0.01
גודל עבור 2 מחיצות (גוונים)	109KB
גודל עבור 4 מחיצות (גוונים)	162KB
גודל עבור 8 מחיצות (גוונים)	203KB
גודל עבור 16 מחיצות (גוונים)	199KB
גודל עבור 32 מחיצות (גוונים)	207KB
גודל עבור 64 מחיצות (גוונים)	209KB
גודל עבור 128 מחיצות (גוונים)	212KB
גודל עבור 256 מחיצות (גוונים)	213KB
גודל עבור 512 מחיצות (גוונים)	215KB
גודל עבור 1024 מחיצות (גוונים)	216KB
גודל תמונה מקורי	222KB

Distortion	0.1
גודל עבור 2 מחיצות (גוונים)	109KB
גודל עבור 4 מחיצות (גוונים)	171KB
גודל עבור 8 מחיצות (גוונים)	182KB
גודל עבור 16 מחיצות (גוונים)	201KB
גודל עבור 32 מחיצות (גוונים)	207KB
גודל עבור 64 מחיצות (גוונים)	208KB
גודל עבור 128 מחיצות (גוונים)	212KB
גודל עבור 256 מחיצות (גוונים)	213KB
גודל עבור 512 מחיצות (גוונים)	215KB
גודל עבור 1024 מחיצות (גוונים)	216KB
גודל תמונה מקורי	222KB

Distortion	100
גודל עבור 2 מחיצות (גוונים)	103KB
גודל עבור 4 מחיצות (גוונים)	142KB
גודל עבור 8 מחיצות (גוונים)	178KB
גודל עבור 16 מחיצות (גוונים)	194KB
גודל עבור 32 מחיצות (גוונים)	203KB
גודל עבור 64 מחיצות (גוונים)	205KB
גודל עבור 128 מחיצות (גוונים)	212KB
גודל עבור 256 מחיצות (גוונים)	213KB
גודל עבור 512 מחיצות (גוונים)	215KB
גודל עבור 1024 מחיצות (גוונים)	217KB
גודל תמונה מקורי	222KB