ReadMe

שם המטלה:

גרפים אקראיים במודל ארדש- ריניה **(Eerdos-Renyi model)**.

בחירת הייצוג של גרף בקוד:

כדי לממש בקוד את כל רעיון הגרפים האקראיים, בדרך כלל משתמשים באחת משני הייצוגים הבאים:

1. מטריצת סמיכויות: מערך דו ממדי בגודל |V|*|V| , בו המספר 1 מייצג צלע בין שני צמתים והמספר 0 מייצג כי אין צלע בין שני הצמתים.

   היתרונות במטריצת סמיכויות הם:

   - הסרה של צומת מהגרף ב**O(1)**.
   - בדיקה האם קיים צלע בין 2 צמתים ב**O(1)**.

   החסרונות הם:

   - משתמש בהרבה מקום (**O(|V|^2)**) ,גם במצב שהגרף דליל (מספר מועט של צלעות).
   - הוספה של צומת לגרף בזמן **O(|V|^2)**.
2. רשימת סמיכויות: מערך של רשימות (וקטורים), גודל המערך שווה לכמות הצמתים, וכל צומת (כל וקטור) בגודל מספר השכנים שלו.
   היתרון ברשימת סמיכויות הוא שמימוש זה תופס פחות מקום **O(|V|+|E|)** ורק במקרה הכי גרוע (גרף שלם) תופס **O(|V|^2).**

במטלה שלנו, נתבקשנו לבדוק 3 תכונות : קשירות, קיום צומת מבודד וקוטר.
ראשית, כדי לדעת האם הגרף קשיר או לא, אנו נפעיל BFS על צומת בגרף. הרצה של BFS על מטריצת סמיכויות היא בסיבוכיות של **O(|V|^2)** , ומצד שני הרצה של BFS על רשימת סמיכויות הוא בסיבוכיות של **O(|V|+|E|)**. לכן, יותר יעיל להשתמש ברשימת סמיכויות.

שנית, בדיקת קיום צומת מבודד בגרף במטריצת סמיכויות תהיה בסיבוכיות **O(|V|^2)** , וזאת משום שעלינו לעבור על כל המטריצה ולחפש שורות אפסים.
לעומת זאת, ברשימת סמיכויות, עלינו לעבור על כל הצמתים, ובכל צומת לבדוק האם יש לו שכן (מספיק אחד), אם יש לו שכן , ניתן לעבור לצומת הבאה , ובכך הסיבוכיות היא **O(|V|)**.

שלישית, חישוב קוטר במקרה הגרוע במטריצת סמיכויות הוא בסיבוכיות של **O(|V|^3)** כלומר הרצה של BFS על כל צומת בגרף.
לעומת זאת, ברשימת סמיכויות, הסיבוכיות הינה **O(|V|*(|E|+|V|))**.

לסיכום, מכיוון שבמטלה שלנו אין צורך בהסרה/הוספה של צומת או בדיקה האם קיימת צלע בין 2 צמתים, השימוש במטריצת סמיכויות למרות יתרונותיה יהיה פחות יעיל ובשל כך בחרנו ברשימת סמיכויות.

בחרנו להשתמש ברשימת סמיכויות מהצורה: `vector<vector<int>>` כלומר, "רשימה של ווקטורים". ווקטורים ב- C++ STL הם מערכים דינאמיים עם היכולת לשנות את גודלם באופן אוטומטי כשאלמנט מוכנס או נמחק אליהם/מהם, דבר המתאים בדיוק לצורך שלנו.
בנוסף, לווקטור יש פונקציות מוכנות אשר משמשות אותנו במהלך התוכנית כגון :
`size()` ו-`push_back()`.

<u>פונקציות בתוכנית:</u>

```
vector<vector<int>> build_random_graph(int V, float p)
```

פונקציה זו מקבלת מס' צמתים והסתברות מסוימת לקיום צלע בין 2 צמתים ומחזירה גרף אקראי.
בתוך הפונקציה רצות 2 לולאות for שתפקידן למלא את רשימת הסמיכויות בשכנים של צמתים.
על מנת להגריל הסתברות כלשהי לקיום צלע השתמשנו בפונקציה rand() שקיבלה סיד time בmain.
הפונקציה פועלת בסיבוכיות: **O(|V|^2)**.

```
int BFS_max_distance(vector<vector<int>>&graph,int root,
vector<bool>& visited, vector<int>& distances)
```

פונקציה זו מקבלת גרף (רשימת סמיכויות) ומספר (צומת התחלתי) ומחזירה מרחק מקסימלי מהצומת שנשלח.
בתוך הפונקציה ממומש אלגוריתם הBFS בעזרת וקטורים (distances, visited) ותור.
המערכים יכלו להיות סטטיים, אך בחרנו להשתמש בווקטורים כדי שיהיה ניתן לשנות את מספר הצמתים בגרף ושהתוכנית תצליח לרוץ בכל זאת.
חשוב לציין כי בשל העובדה שאנו עוברים על כל הצמתים השכנים של `root` על פי סדר הופעתם בזיכרון בחרנו להשתמש ב – `iterator` על מנת לייעל את זמן הריצה.
הפונקציה פועלת בסיבוכיות: **O(|V|+|E|)** – סיבוכיות BFS.

```
int CalcNumOfEdges(vector<vector<int>>&graph)
```

// לא שומשה בתוכנית (מייעלת את תהליך חישוב הקוטר בגרף בעל מס' צמתים קטן יחסית)
פונקציה זו מקבלת גרף ומחזירה את כמות הצלעות בגרף.
הפונקציה למעשה סוכמת את כמות האיברים ברשימת סמיכויות, ומחלקת ב2 וכך מוחזרת לנו כמות הצלעות בגרף, לפי הנוסחה: $\sum deg(v) = 2|E|$.
הפונקציה פועלת בסיבוכיות: **O(|V|)** - רצה על כל הצמתים.

```
bool isCompleteGraph(vector<vector<int>>&graph,int numOfEdges)
```

// לא שומשה בתוכנית (מייעלת את תהליך חישוב הקוטר בגרף בעל מס' צמתים קטן יחסית)
פונקציה זו מקבלת גרף ואת כמות הצלעות בגרף.
הפונקציה בודקת האם numOfEdges =(|V|*(|V|-1))/2.
אם מתקיים השוויון, זאת אומרת שהגרף הינו גרף שלם, והפונקציה מחזירה `true`, אחרת מחזירה `false`.
הפונקציה פועלת בסיבוכיות: **O(1)**.

```
bool isTree(vector<vector<int>>&graph,int numOfEdges)
```

// לא שומשה בתוכנית (מייעלת את תהליך חישוב הקוטר בגרף בעל מס' צמתים קטן יחסית)

פונקציה זו מקבלת גרף ואת כמות הצלעות בגרף.

הפונקציה בודקת האם |E|=|V|-1.

חשוב לציין שפונקציה זו תקרא רק לאחר בדיקת הקשירות של הגרף ולכן במידה ותנאי זה מתקיים המשמעות היא שהגרף הינו עץ והפונקציה מחזירה `true`, אחרת תחזיר `false`.

הפונקציה פועלת בסיבוכיות: **O(1)**.

```
int calcTreeDiam(vector<vector<int>>&graph, vector<bool>&
visited, vector<int>& distances)
```

// לא שומשה בתוכנית (מייעלת את תהליך חישוב הקוטר בגרף בעל מס' צמתים קטן יחסית)

פונקציה זו מקבלת גרף (עץ) ומחזירה את קוטר העץ.

מכיוון שאנו פועלים על מקרה פרטי של גרף שהוא עץ, חישוב הקוטר יכול להיות הרבה יותר יעיל.

במקום להריץ BFS על כל אחד מהצמתים כדי לחשב את הקוטר, אנו נריץ BFS פעם אחת על צומת מסוים ולאחר מכן נריץ BFS על הצומת הרחוק ביותר ממנו. כלומר 2 הרצות BFS במקום |V| הרצות.

הפונקציה פועלת בסיבוכיות: **O(|V|+|E|)** – סיבוכיות BFS.

```
bool Is_Isolated(vector<vector<int>> &graph)
```

פונקציה זו מקבלת גרף כלשהו ובודקת האם קיים צומת בודד כלשהו בגרף.

במידה והפונקציה מצאה צומת בודד, הפונקציה תחזיר `true`, אחרת תחזיר `false`.

הפונקציה תרוץ בלולאה על כל הצמתים של הגרף ותבדוק האם יש להם שכנים, ברגע שהיא מצאה צומת ללא שכנים, יוחזר למשתמש שנמצא צומת בודד. אחרת, במידה והסתיימה הלולאה יוחזר למשתמש שלא נמצא צומת בודד.

אנו משתמשים בתוך הלולאה בבדיקה האם כל וקטור (צומת) ברשימת הווקטורים שלנו הוא ריק בעזרת הפונקציה המובנת בSTL - empty().

הפונקציה פועלת בסיבוכיות: **O(|V|)**.

```
bool connectivity(vector<vector<int>>&graph)
```

פונקציה זו מקבלת גרף כלשהו ובודקת האם הוא קשיר.

במידה והגרף אינו קשיר הפונקציה תחזיר `false`, אחרת הפונקציה תחזיר `true`.

ראשית כדי לייעל את תהליך הבדיקה, הפונקציה תבדוק אם קיים צומת בודד, במידה וכן הפונקציה תחזיר ישירות `false`. לאחר מכן, הפונקציה תריץ BFS על צומת מסוים ותבדוק האם מס' הצמתים שבוקרו שווה למס' הצמתים בגרף ובמידה וכן תחזיר `true` , אחרת תחזיר `false`.

הפונקציה פועלת בסיבוכיות: **O(|V|+|E|)** – סיבוכיות BFS.

```
int diameter(vector<vector<int>>&graph)
```

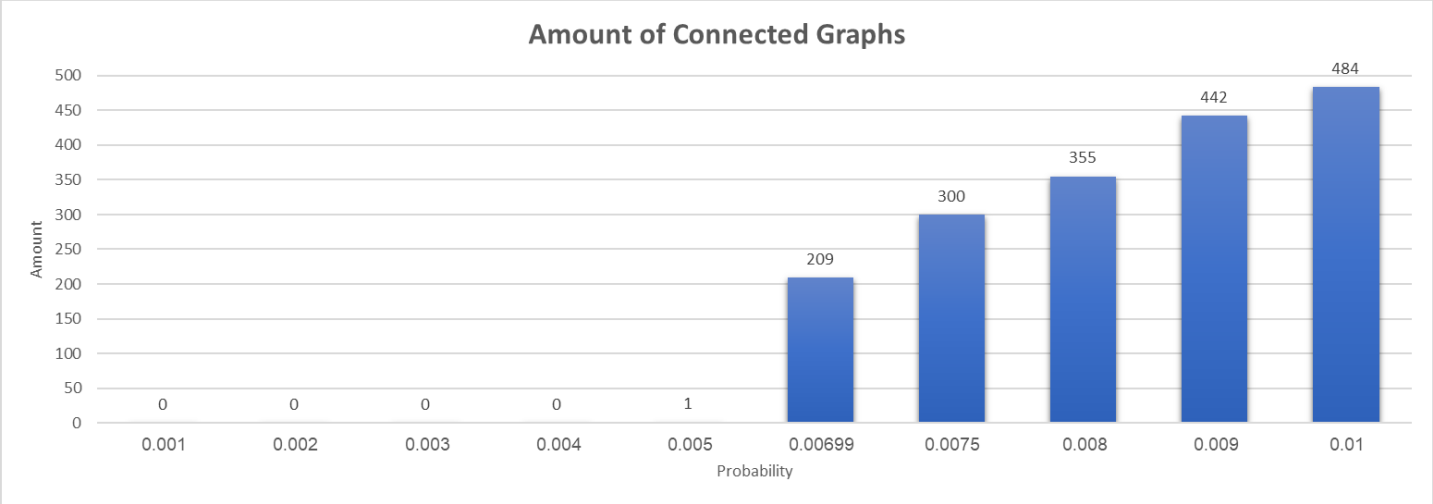
פונקציה זו מקבלת גרף כלשהו ומחזירה את הקוטר שלו.
בתחילת ההרצה של הפונקציה, יתבצעו כמה בדיקות של מקרים פרטיים:
ראשית, הפונקציה תבדוק האם הגרף קשיר, במידה ולא הפונקציה תחזיר קוטר בגודל |V|+1 (במקרה הפרטי של |V|=1000, יוחזר 1001) שמסמל קוטר ∞ (כי בגרף קשיר ולא מכוון אין אפשרות לקוטר שגדול ממס' הצמתים).
/* לא שומש בתוכנית (מייעל את תהליך חישוב הקוטר בגרף בעל מס' צמתים קטן יחסית)
שנית, הפונקציה תחשב את מס' הצלעות בגרף בעזרת הפונקציה – CalcNumOfEdges שתעזור לנו במקרים פרטיים אחרים על מנת לייעל את מציאת הקוטר. לאחר מכן, הפונקציה תבדוק אם הגרף הוא גרף שלם בעזרת הפונקציה – isCompleteGraph ובמידה והגרף הוא שלם יוחזר אוטומטית קוטר 1.
שלישית, הפונקציה תבדוק אם הגרף הוא עץ בעזרת הפונקציה – isTree ובמידה והוא עץ הפונקציה תחזיר את הקוטר בעזרת הפונקציה – calcTreeDiam.
כל אלו הם מקרים פרטיים שתפקידם למנוע מהפונקציה לרוץ על כל הצמתים ולהפעיל BFS על כל אחד מהם , דבר המייעל את תפקוד הפונקציה משמעותית.*/

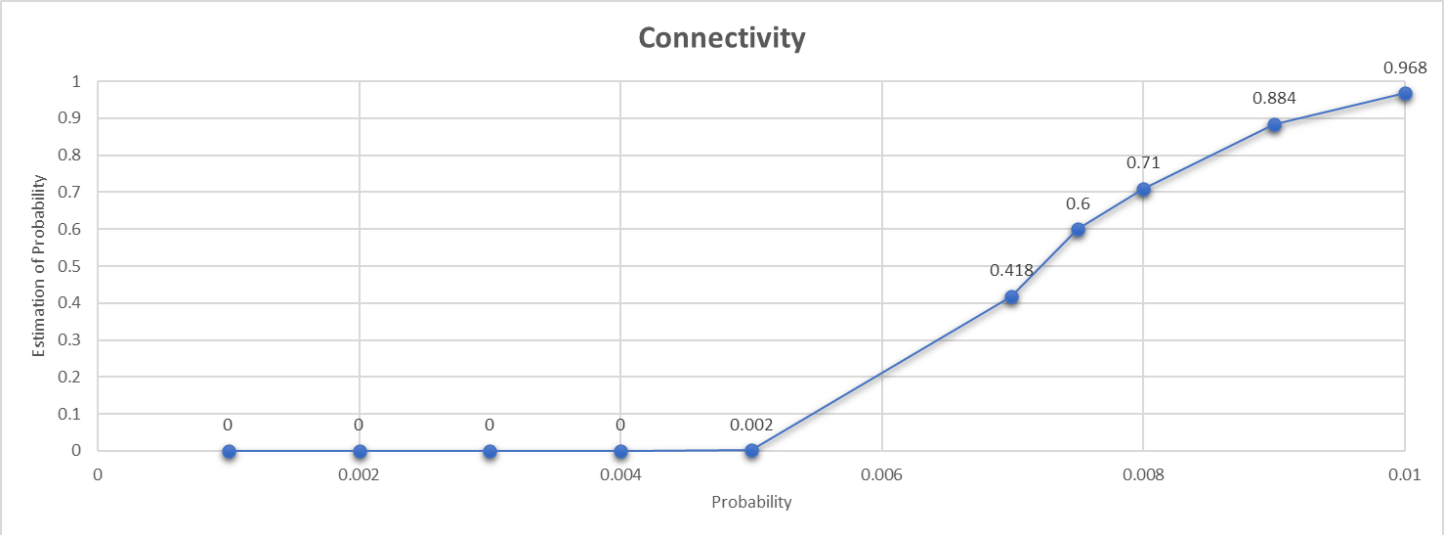
לבסוף, במידה והגרף לא עונה על אף אחד מהמקרים הפרטיים שבדקנו(לא שומשו), הפונקציה תריץ BFS על כל אחד מהצמתים ותבדוק מהו המרחק הגדול ביותר בכל הרצה ותחזיר את הגדול ביותר – הקוטר.
הפונקציה פועלת בסיבוכיות: **O(|V|*(|E|+|V|))** - סיבוכיות |V|*BFS.

<u>תוצאות התוכנית וסימולציות:</u>

בתוכנית נתבקשנו להגריל 500 גרפים אקראיים בעלי 1000 צמתים ולבדוק 3 תכונות.
את תוצאות התוכנית שמרנו בתוך קבצי CSV נפרדים, שאותם פתחנו בExcel וערכנו אותם על מנת להציגן באופן ויזואלי.
בקובץ הZIP שכולל בתוכו את כל קבצי המטלה ישנה תיקייה בשם Pure CSV Files אשר כוללת את הקבצים שקיבלנו מהרצת התוכנית ללא עריכה כלל.
בנוסף, ישנה תיקייה נוספת XLSX Files אשר כוללת את קבצי הCSV הערוכים ואשר מציגים את התוצאות בעזרת גרפים וטבלאות. חשוב לציין, שישנו קובץ אחד בשם: All_In_One שמכיל את כל התוצאות בקובץ אחד למען הנוחות.
כעת, נציג את התוצאות ונסביר על כל אחת מהתכונות (בעמוד הבא):

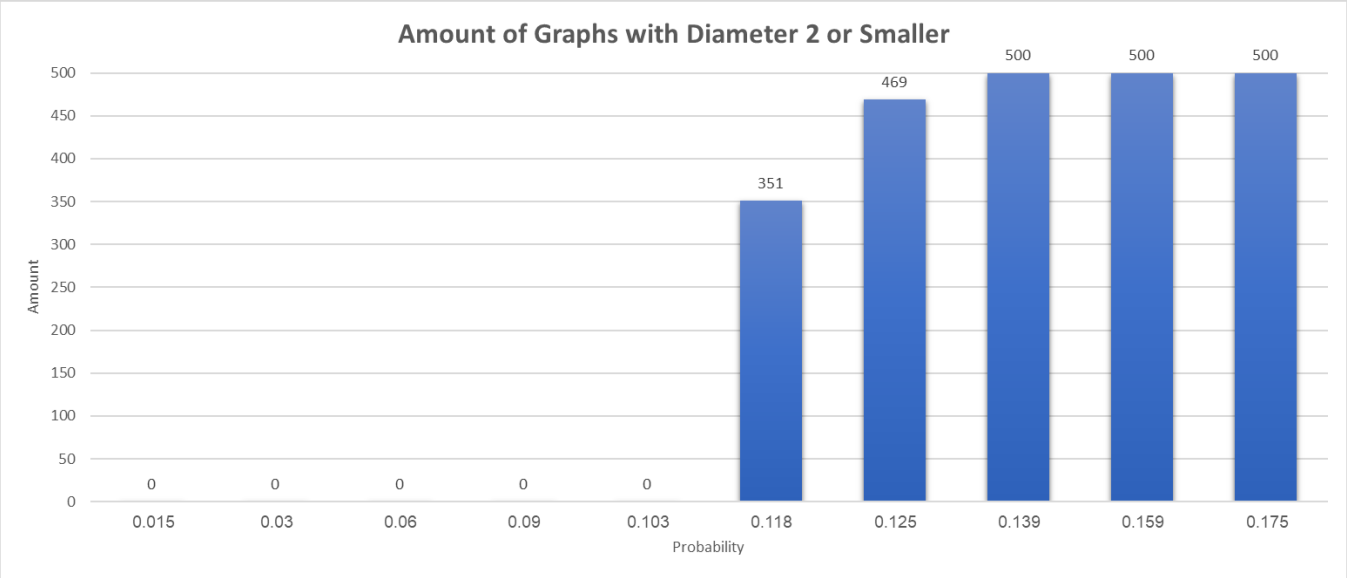
Connectivity	p<$Threshold1$					p>$Threshold1$					$Threshold1$=	0.006907755
P	0.001	0.002	0.003	0.004	0.005	0.00699	0.0075	0.008	0.009	0.01		
Amount of connected graphs	0	0	0	0	1	209	300	355	442	484		
Estimation of probability	0	0	0	0	0.002	0.418	0.6	0.71	0.884	0.968		

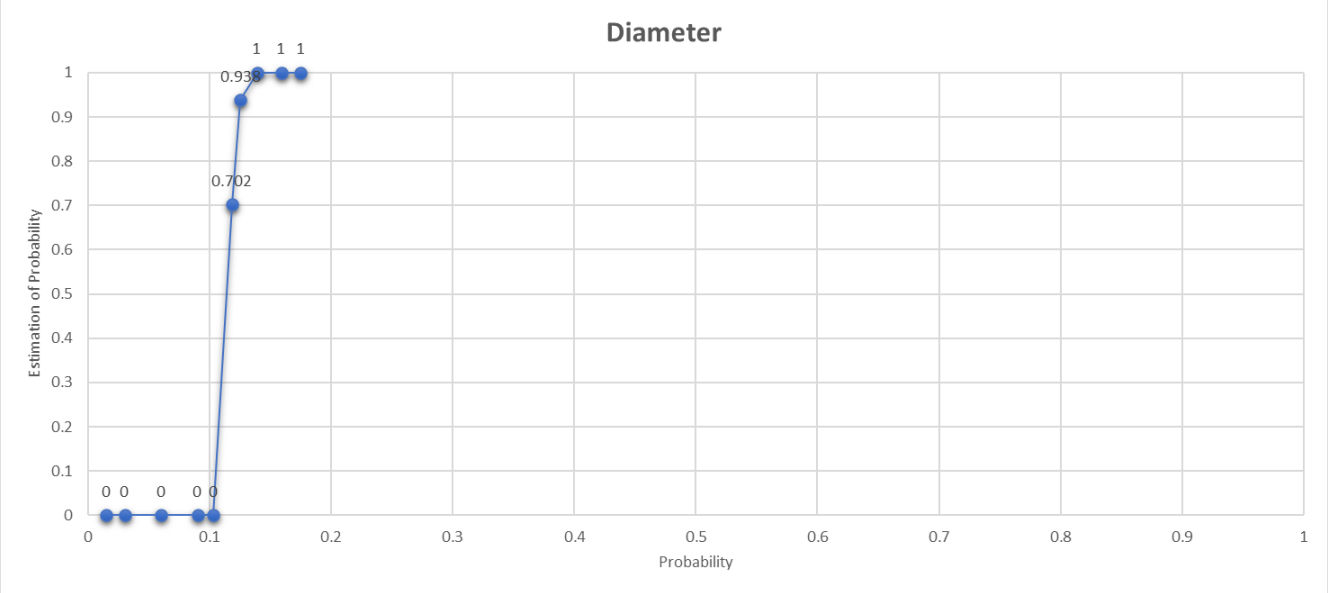




כפי שניתן לראות בתוצאות הנ"ל בחרנו בערכים בסביבת ה$Threshold1$ על מנת להדגים את התוצאות על הצד הטוב ביותר. בגרפים הנ"ל בכל אחת מההסתברויות שקטנות מה$Threshold1$ קיבלנו הסתברות אפסית, כלומר הגרף לא קשיר בהסתברות גבוהה מאוד, כפי שציפינו לקבל.
ככל שההסתברות גדלה חלה מגמת עליה במס' הגרפים הקשירים מתוך ה500 שנבדקו, דבר העולה בקנה אחד עם הרשום במטלה . (אם Threshold1< p אזי הגרף קשיר בהסתברות גבוהה.)

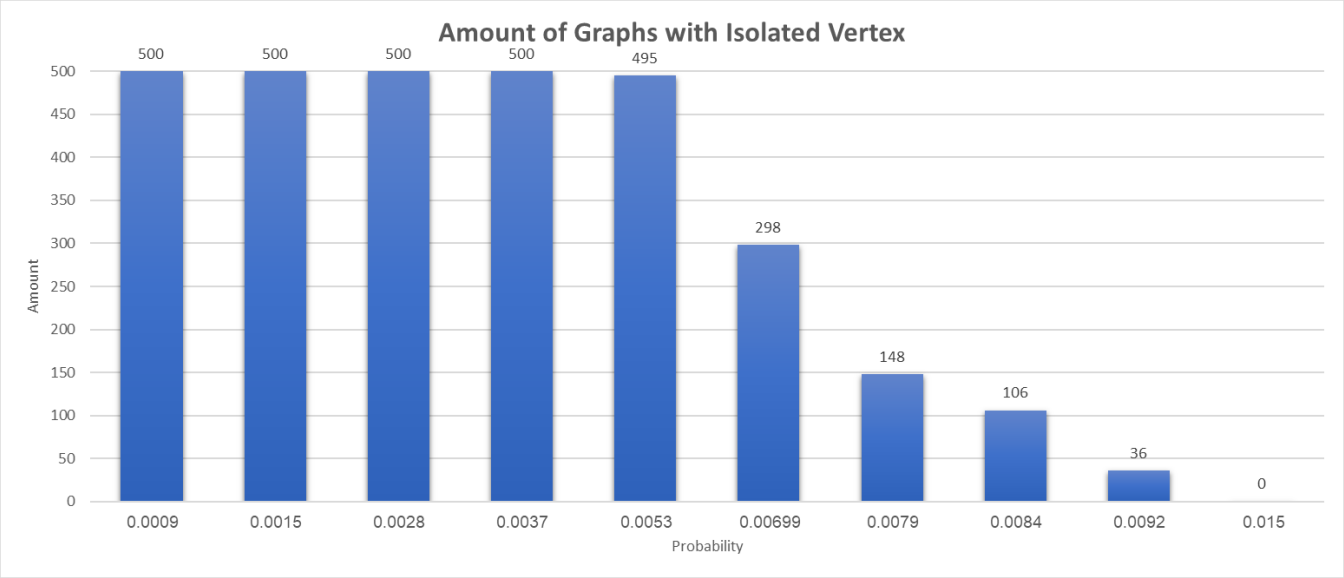
Diameter	p<$Threshold2$					p>$Threshold2$					$Threshold2$=	0.1175394
P	0.015	0.03	0.06	0.09	0.103	0.118	0.125	0.139	0.159	0.175		
Amount of graphs with diameter 2 or smaller	0	0	0	0	0	351	469	500	500	500		
Estimation of probability	0	0	0	0	0	0.702	0.938	1	1	1		

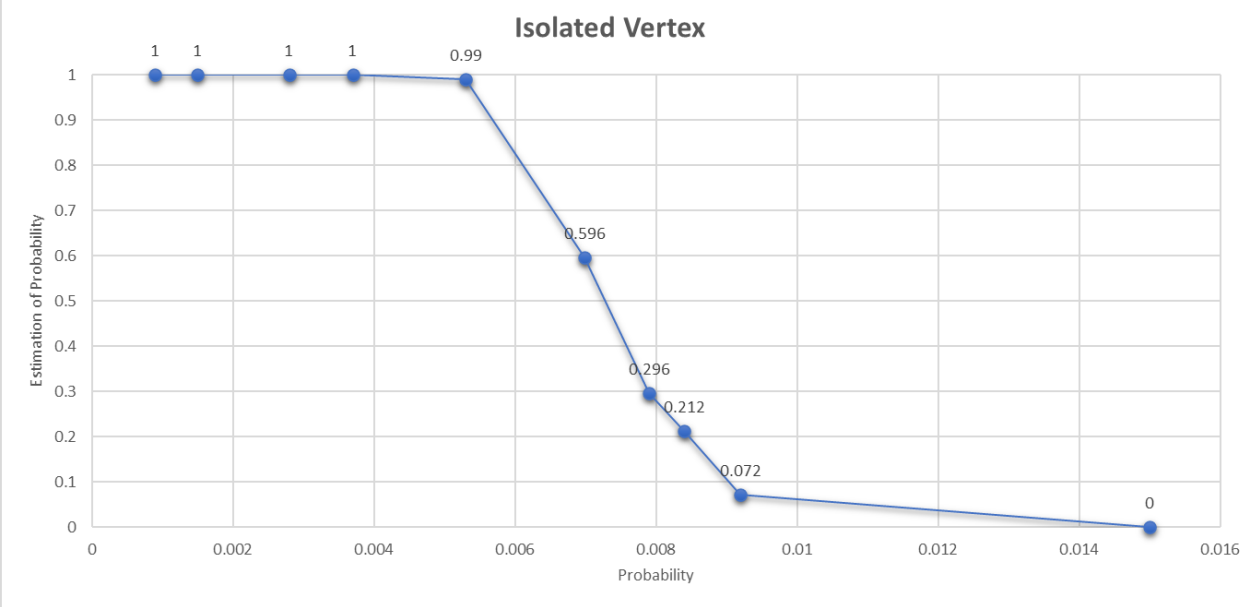




כפי שניתן לראות בתוצאות הנ"ל בחרנו בערכים בסביבת ה$Threshold2$.
בגרפים הנ"ל בכל אחת מההסתברויות שקטנות מה$Threshold2$ קיבלנו הסתברות אפסית, כלומר קוטר הגרף שהתקבל גדול מ-2 בהסתברות גבוהה מאוד, כפי שציפינו לקבל.
ככל שההסתברות גדלה חלה מגמת עליה חדה מאוד במס' הגרפים שקוטרם קטן או שווה ל-2 מתוך ה500 שנבדקו, דבר העולה בקנה אחד עם הרשום במטלה.

Is Isolated	p<$Threshold3$					p>$Threshold3$					$Threshold3$=	0.006907755
P	0.0009	0.0015	0.0028	0.0037	0.0053	0.00699	0.0079	0.0084	0.0092	0.015		
Amount of graphs with isolated vertex	500	500	500	500	495	298	148	106	36	0		
Estimation of probability	1	1	1	1	0.99	0.596	0.296	0.212	0.072	0		





כפי שניתן לראות בתוצאות הנ"ל בחרנו בערכים בסביבת ה$Threshold3$.
בגרפים הנ"ל בכל אחת מההסתברויות שקטנות מה$Threshold3$ קיבלנו הסתברות גבוהה מאוד (1 או קרובה מאוד ל1), כלומר הסבירות שיהיה צומת מבודד בגרף שהתקבל גדולה מאוד.
ככל שההסתברות גדלה חלה מגמת ירידה במס' הגרפים שקיים בהם צומת בודד מתוך ה500 שנבדקו, דבר העולה בקנה אחד עם הרשום במטלה.