<u>פרוייקט אמינות</u>

<u>מגישים</u>: עידן בן משה 308118439, זוהר עזריהאב 201454899, מארק רבייב 310356621,

אלירן גבאי 203062831, עמיחי תורג'מן 203789649

<u>חלק 1</u>

רשמנו את הפרוייקט בשפת python. בנינו את הגרף שלנו בצורת מבנה נתונים של מילון. לכל צלע רשמנו את כל הקודקודים אשר הינם שכנים שלה בגרף, כמו רשימת שכינויות, ואיתחלנו כל ערך של ל-'O' בערך ברירת מחדל.

```
## A Dictionary that Describes to given Graph
Graph = {1:{2:'O', 5:'O', 8:'O'}, 2:{1:'O', 3:'O', 10:'O'}, 3:{2:'O', 4:'O', 12:'O'},
         4:{3:'O', 5:'O', 14:'O'}, 5:{1:'O', 4:'O', 6:'O'}, 6:{5:'O', 7:'O', 15:'O'},
         7:{6:'O', 8:'O', 17:'O'}, 8:{1:'O', 7:'O', 9:'O'}, 9:{8:'O', 10:'O', 18:'O'},
         10:{2:'O', 9:'O', 11:'O'}, 11:{10:'O', 12:'O', 19:'O'}, 12:{3:'O', 11:'O', 13:'O'},
         13:{12:'O', 14:'O', 20:'O'}, 14:{4:'O', 13:'O', 15:'O'}, 15:{6:'O', 14:'O', 16:'O'},
         16:{15:'O', 17:'O', 20:'O'}, 17:{7:'O', 16:'O', 18:'O'}, 18:{9:'O', 17:'O', 19:'O'},
         19:{11:'O', 18:'O', 20:'O'}, 20:{13:'O', 16:'O', 19:'O'}}
```

ניתן לראות כאן את פונקציית ה-main אשר מכילה במערך P את כל ההסתברויות של אמינות המערכת. מערך M מכיל את מספר האיטרציות שאנו רוצים להריץ לבדיקה.
שלושת הטרמינליים שלנו הם: קודקוד 10,14,20. ולבסוף את הקריאה לפונקציית MonteCarlo אשר מחשבת את אמינות המערכת עבור כל P.

```
def main():
    P = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99]
    M = [1000, 10000]
    T1 = 10
    T2 = 14
    T3 = 20
    print("Terminals: T1 = {}, T2 = {}, T3 = {}".format(T1,T2,T3))
    print('\tP\t|\tM=1000\t|\tM=10000')
    print('-'*50)
    MonteCarlo(M, P, Graph, T1, T2, T3)
```

ניתן לראות כאן את פונקציית MonteCarlo אשר בנינו. הפונקצייה מקבלת את המערך M אשר מכיל את מספר האיטרציות שנרוץ(1000, 10000) , מערך P אשר מכיל את כל ההסתברויות של המערכת, את הגרף שבנינו בצורת מילון והסברנו עליו מקודם, ואת שלושת הטרמינליים 10,14,20. אנו רצים בלולאה עבור כל P ובתוך יוצרים עוד לולאה שבה אנו רצים על המערך עבור כל M. מאתחלים את הדגימה r ל-0 ורצים תחילה עד 1000 ולאחר מכן עד 10000. מעתיקים את הגרף המקורי שלנו למשתנה copyOfGraph על מנת לא לשנות את גרף המקור אלא לרוץ על עותק ממנו, ובנוסף אנו יוצרים מילון בשם MC_graph אשר תפקידו זה לשמור את כל הצלעות אשר נמצאות במצב UP על מנת לבדוק אחר כך האם המערכת ב-UP. עבור כל צלע אנו מגרילים מספר רנדומלי מסוג float , אם ההסתברות שהגרלנו קטנה או שווה מההסתברות P אזי הצלע במצב UP אחרת DOWN. לאחר שרצנו על כל הצלעות בגרף , אנו בודקים האם קיימת דרך אשר מחברת את 3 הטרמינליים שלנו , אם כן מגדילים את המשתנה r ב-1 וכך הלאה ולבסוף מחלקים את r במספר האיטרציות שביצענו וכך מתקבלת ההסתברות שלנו.

```
def MonteCarlo(M, P, Graph, T1 ,T2 ,T3):
    for p in P:
        print('\t'+format(p, '.2f'), end='\t/\t')
        for m in M:
            r = 0
            for _ in range(m):
                copyOfGraph = copy.deepcopy(Graph)
                MC_graph = {}
                # iterate over all the edges and set their state
                for v1 in range(1, 21):
                    for v2 in copyOfGraph[v1]:
                        if copyOfGraph[v1][v2] == 'O':
                            # random number
                            x = random.uniform(0, 1)
                            if float(x) <= float(p):
                                copyOfGraph[v1][v2] = 'UP' #each edge is represented twice
                                copyOfGraph[v2][v1] = 'UP'
                                if v1 not in MC_graph:
                                    MC_graph[v1] = []
                                MC_graph[v1].append(v2)
                                if v2 not in MC_graph:
                                    MC_graph[v2] = []
                                MC_graph[v2].append(v1)
                            else:
                                copyOfGraph[v1][v2] = 'DOWN'
                                copyOfGraph[v2][v1] = 'DOWN'


                # check if one vertex is connected to the other two, if it is so the system is up
                if (searchPath(MC_graph, T3, T1) and searchPath(MC_graph, T3, T2))  or (
                    searchPath(MC_graph, T2, T1) and searchPath(MC_graph, T2, T3)) or (
                    searchPath(MC_graph, T1, T2) and searchPath(MC_graph, T1, T3)):
                        r = r + 1

            print(format(r/m, '.4f'), end='\t/\t')
            if m == 10000:
                print()
```

פונקציית searchPath מקבלת את המשתנה MC_graph אשר הוא מילון שנכיל את כל הצלעות שנמצאות ב-UP , start הוא הטרמינל אשר ממנו אנו מתחילים את הניתוב, end הוא הטרמינל אשר אליו אנו רוצים להגיע, ו-path הוא מערך אשר שומר את כל הדרכים האפשריות להגיע מהטרמינל ההתחלתי לטרמינל הסופי. הפונקצייה היא פונקצייה רקורסיבית אשר מוצאת את כל הדרכים האפשריות להגיע מטרמינל אחד לשני.

```
def searchPath(graph, start, end, path=[]):
    path = path + [start]
    if start not in graph:
        return False
    if start == end:
        return True
    for i in graph[start]:
        if i not in path:
            if searchPath(graph, i, end, path):
                return True
    return False
```

לבסוף התוצאות מודפסות אבל כל ערך של P , למשך 1000 ו-10000 איטרציות ואלה הן התוצאות שהתקבלו:

```
Terminals: T1 = 10, T2 = 14, T3 = 20
```

P	M=1000	M=10000
0.10	0.0000	0.0000
0.20	0.0020	0.0011
0.30	0.0120	0.0123
0.40	0.0710	0.0646
0.50	0.2260	0.2165
0.60	0.5200	0.4963
0.70	0.7960	0.7842
0.80	0.9550	0.9501
0.90	0.9980	0.9960
0.95	1.0000	0.9995
0.99	1.0000	1.0000

חלק 2 A

הגרף הוא אותו הגרף מחלק 1 ללא שינוי.

```
## A Dictionary that Describes to given Graph
Graph = {1:{2:'O', 5:'O', 8:'O'}, 2:{1:'O', 3:'O', 10:'O'}, 3:{2:'O', 4:'O', 12:'O'},
         4:{3:'O', 5:'O', 14:'O'}, 5:{1:'O', 4:'O', 6:'O'}, 6:{5:'O', 7:'O', 15:'O'},
         7:{6:'O', 8:'O', 17:'O'}, 8:{1:'O', 7:'O', 9:'O'}, 9:{8:'O', 10:'O', 18:'O'},
         10:{2:'O', 9:'O', 11:'O'}, 11:{10:'O', 12:'O', 19:'O'}, 12:{3:'O', 11:'O', 13:'O'},
         13:{12:'O', 14:'O', 20:'O'}, 14:{4:'O', 13:'O', 15:'O'}, 15:{6:'O', 14:'O', 16:'O'},
         16:{15:'O', 17:'O', 20:'O'}, 17:{7:'O', 16:'O', 18:'O'}, 18:{9:'O', 17:'O', 19:'O'},
         19:{11:'O', 18:'O', 20:'O'}, 20:{13:'O', 16:'O', 19:'O'}}
```

פונקציית ה-main מכילה מערך של הזמנים אשר אנו בודקים בהם את אמינות הרשת, מילון alive אשר מכיל את כל הזמנים, ותפקידו הוא לשמור בכל זמן את אורך החיים של המערכת. M=10000 הוא מספר האיטרציות אאשר נבצע, שלושת הטרמינליים שלנו 10,14,20 ופונקציית MonteCarlo אשר תחשב לנו בכל זמן את אורך החיים שלה(אמינות המערכת).

```
def main():
    times = [0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95,1]
    alive={0:[],0.05:[],0.1:[],0.15:[],0.2:[],0.25:[],0.3:[],0.35:[],0.4:[],0.45:[],0.5:[],0.55:[],0.6:[],0.65:[],0.7:[],0.75:[],0.8:[],0.85:
    M = 10000
    T1 = 10
    T2 = 14
    T3 = 20
    print("Terminals: T1 = {}, T2 = {}, T3 = {}".format(T1,T2,T3))
    MonteCarlo(M, alive, times, Graph, T1, T2, T3)
```

פונקציית MonteCarlo מקבלת את כל מה שהסברנו בדף הקודם, רצה בלולאה עבור כל אחד מהזמנים, עבור כל זמן מבצעים 10000 איטרציות מרחב דגימה, יוצרים מילון MC_graph אשר יכיל את כל הצלעות שהזמן שהן תקינות יותר גדול מהזמן אשר אנו בודקים, copyOfGrpah הוא העתק של הגרף שלנו עם רשימת השכינויות. אנו מגרילים ערך רנדומלי float מ-0 עד 1 עבור כל צלע ומחשבים ת אורך חייה של כל צלע באמצעות הנוסחה של התפלגות מעריכית. על מנת לחשב את אורך חייה של המערכת השתמשנו בהתפלגות Burtin Pittel ע"י כך שחישבנו קבוצת נתק בין כל שלושת הטרמינליים. לבסוף אם אורך חייה של הרשת גדול מהזמן שאנו רצים עליו אנו מוסיפים למילון alive את אמינות המערכת בזמן שר בדקנו. את כל הנתונים ששמרנו במילון alive אנו שולחים לפונקציה tableprint אשר תדפיס לנו גרף של אמינות המערכת בכל זמן שנבדק.

```python
def MonteCarlo(M, alive, times, Graph, T1 ,T2 ,T3):
    for t in times:
        for _ in range(M):
            MC_graph = {}
            copyOfGraph = copy.deepcopy(Graph)
            # iterate over all the edges and set their state
            for v1 in range(1, 21):
                for v2 in copyOfGraph[v1]:
                    if copyOfGraph[v1][v2] == '0':
                        # random number
                        x = random.uniform(0,1)
                        copyOfGraph[v1][v2] = -1 * math.log(x)
                        copyOfGraph[v2][v1] = -1 * math.log(x)
                        if float(copyOfGraph[v1][v2]) > float(t):
                            if v1 not in MC_graph:
                                MC_graph[v1] = []
                            MC_graph[v1].append(v2)
                            if v2 not in MC_graph:
                                MC_graph[v2] = []
                            MC_graph[v2].append(v1)
            system_life = 0
            # check if one vertex is connected to the other two, if it is so the system is up
            if (searchPath(MC_graph, T3, T1) and searchPath(MC_graph, T3, T2)):
                system_life = 1 - (1 - math.e**((-t**3) * (copyOfGraph[20][13] * copyOfGraph[20][16] * copyOfGraph[20][19])))
            elif (searchPath(MC_graph, T2, T1) and searchPath(MC_graph, T2, T3)):
                system_life = 1 - (1 - math.e**((-t**3) * (copyOfGraph[14][4] * copyOfGraph[14][13] * copyOfGraph[14][15])))
            elif (searchPath(MC_graph, T1, T2) and searchPath(MC_graph, T1, T3)):
                system_life = 1 - (1 - math.e**((-t**3) * (copyOfGraph[10][2] * copyOfGraph[10][9] * copyOfGraph[10][11])))

            if (float(system_life) > float(t)):
                alive[t].append(system_life)

    tableprint(alive, M)
```

אותה פונקציית searchPath מחלק 1.

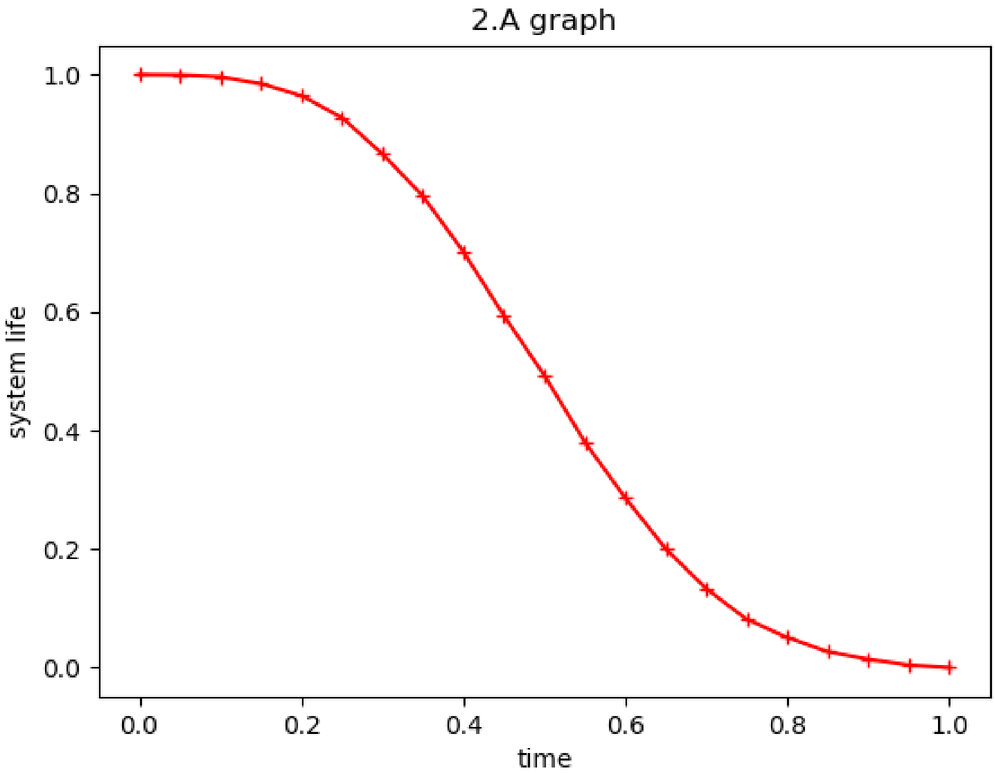
```python
def searchPath(graph, start, end, path=[]):
    path = path + [start]
    if start not in graph:
        return False
    if start == end:
        return True
    for i in graph[start]:
        if i not in path:
            if searchPath(graph, i, end, path):
                return True
    return False
```

פוקנציית tableprint מקבלת את המילון alive אשר מכיל עבור כל זמן שנבדק את אמינות המערכת, ו-M שהוא מספר האיטרציות שרצנו 10000. אנו רצים בלולאה עבור כל זמן וסופרים את מספר אמינויות הרשת שהתקבלו, ומחלקים במרחב הדגימה 10000 על מנת לקבל את אמינות המערכת בכל זמן שנבדק. ומדפיסים את התוצאות בגרף שיובא בהמשך.

```
def tableprint(alive, M):
    print ('\t{}\t|\t{}'.format('time','system life'))
    print('-' * 40)
    for i,j in alive.items():
        R=len(j)/M
        print ('\t{}\t \t{}'.format(i,R))
    time=[i for i,j in alive.items()]
    system_life=[len(j)/M for i,j in alive.items()]
    plt.plot(time,system_life,color='red',marker="+")
    plt.title("2.A graph")
    plt.xlabel("time")
    plt.ylabel("system life")
    plt.show()
```

גרף התוצאות אשר התקבל ותוצאות אמינות המערכת לפי זמן:

```
Terminals: T1 = 10, T2 = 14, T3 = 20
        time    |        system life
----------------------------------------
        0                1.0
        0.05             0.9995
        0.1              0.9966
        0.15             0.985
        0.2              0.9647
        0.25             0.9278
        0.3              0.8665
        0.35             0.7953
        0.4              0.7014
        0.45             0.5936
        0.5              0.4926
        0.55             0.38
        0.6              0.2858
        0.65             0.1998
        0.7              0.1326
        0.75             0.081
        0.8              0.0506
        0.85             0.0265
        0.9              0.0139
        0.95             0.0043
        1                0.0
```


2.A graph
system life
time
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

חלק B 2

בחלק זה השתמשנו בשתי פונקציות ה-MonteCarlo של הרשת הסטטית מחלק 1 ושל הרשת הדינמית מחלק A 2 , כל שאר הפונקציות והגרף לא השתנו, חוץ מפונקציית ה-main שלהלן:
את כל החלקים של חלק B 2 הרצנו ב-main אחד אשר מחולק לחלקים:
חלק 1 מחשב את אמינות המערכת הסטטית כאשר אמינות כל צלע שווה ל-$e^{-0.5}$ ומציג את אמינות המערכת.
חלק 2 משווה בין המערכת הדינאמית מסעיף A 2 למערכת הסטטית $e^{-0.5}$ בחמשת הזמנים הבאים: 0.1, 0.3, 0.5, 0.7, 0.9.
חלק 3 מחשב את הטעות היחסית של כל מערכת מחלק 2 בכל אחד מחמשת הזמנים ומשווה ביניהם.

```
def main():
    M = 10000
    T1 = 10
    T2 = 14
    T3 = 20
    print("Terminals: T1 = {}, T2 = {}, T3 = {}".format(T1,T2,T3))
    "-------------Part 1-------------"
    print("\n\t\tpart 1")
    print('-'*40)
    MonteCarloStatic(M, Graph, T1, T2, T3,"part1")
    "-------------Part 2-------------"
    print("\n\t\tpart 2")
    print('-'*40)
    times = [0.1, 0.3, 0.5, 0.7, 0.9]
    alive = {0.1:[], 0.3:[], 0.5:[], 0.7:[], 0.9:[]}
    print('\n\ttime\t|\tstatic\t|\tdynamic')
    print('-'*50)
    dynamicRe = {}
    staticRe = {}
    for t in times:
        dynamic = float(MonteCarloDynamic(M, alive, t, Graph, T1, T2, T3))
        dynamicRe[t] = []
        dynamicRe[t].append((math.sqrt(1 - dynamic))/(math.sqrt(dynamic) * math.sqrt(M)))
        static = float(MonteCarloStatic(M, Graph, T1, T2, T3,""))
        staticRe[t] = []
        staticRe[t].append((math.sqrt(1 - static))/(math.sqrt(static) * math.sqrt(M)))
        print("\t{}\t|\t{}\t|\t{}".format(t, static, dynamic, ".1f", ".4f", ".4f"))
    "-------------Part 3-------------"
    print("\n\t\tpart 3")
    print('-'*40)
    print('\n\ttime\t|\t\tstatic\t\t|\t\tdynamic')
    print('-'*80)
    for t in times:
        print("\t{}\t|\t{}\t|\t{}".format(t, staticRe[t][0], dynamicRe[t][0], ".1f", ".4f", ".4f"))
```

התוצאות של כל שלושת הסעיפים מוצגות להלן:

```
Terminals: T1 = 10, T2 = 14, T3 = 20

                part 1
----------------------------------------
R(t) = 0.5098

                part 2
----------------------------------------

        time    |       static  |       dynamic
--------------------------------------------------
        0.1     |       0.521   |       0.9971
        0.3     |       0.5209  |       0.8677
        0.5     |       0.516   |       0.4924
        0.7     |       0.513   |       0.1357
        0.9     |       0.5149  |       0.016

                part 3
----------------------------------------

        time    |               static                  |               dynamic
--------------------------------------------------------------------------------
        0.1     |       0.009588460755226073            |       0.0005392990320716163
        0.3     |       0.009590382012481328            |       0.003904766991116254
        0.5     |       0.009684959969581863            |       0.01015317296229482
        0.7     |       0.00974329379004566             |       0.025237267596856492
        0.9     |       0.009706310753343614            |       0.07842193570679061
```

לפי התוצאות ניתן לראות כי השיטה של מערכת סטטית שלא תלויה בזמן עדיפה על השיטה של מערכת אשר תלוי בזמן, הטעות היחסית די קבועה במערכת סטטית ונעה באזור ה-0.0095 ל-0.0098 לעומת מערכת שתלויה בזמן שהטעות היחסית בה נעה בין 0.0005 ל-0.07 שזה טווח די גדול של טעות יחסית.