אלגוריתמים 2 – תרגול (שחר אנגל).

תרגול 1 + 2:

בעיית הבקבוקים:

תיאור הבעיה : נתונים שני בקבוקים בגדלים M,N שמלאים ב a,b ליטרים.

לדוגמא :אחד בגדול של 5 ליטרים והשני בגודל 3 ליטרים כאשר אנחנו נרצה למלא את בקבוק א' ב4 ליטרים בעזרת שלושה כללים:

1. למלא את הבקבוק עד הסוף.
2. לרוקן בקבוק עד הסוף.
3. למזוג מבקבוק אחד לשני.

כאשר בצורה כללית יש לנו 6 פעולות (!3 = כל הפרמוטציות).

כעת ניתן לתאר את הבעיה כגרף בעל קודקודים כאשר כל קודקוד מייצג מצב נתון למשל הקודקוד (1,2) ממנו יהיה ניתן להגיע ל6 קודקודים (מצבים שונים) ע"י הפעולות הנ"ל.

בצורה הזו נרכיב את הגרף עבור כל הקודקודים ונמצא את המסלול המתאים מהקודקוד שיבקשו אל הפתרון.

נייצג את הגרף ע"י מטריצת שכנויות , אך איך נדע אם קיים מסלול או לא ?

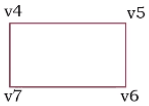
פתרון :

Floyd – Warshall (FFF) :

מטרת אלגוריתם זה הוא לבדוק האם קיים קודקוד בגרף שיכול לחבר לנו בין שתי קודקודים אחרים וליצור לנו מסלול.

כלומר ניקח את טבלת השכנויות ונבדוק בה האם קיים קודקוד המקשר בין שני קודקודים שאין בינהם קשר ישיר – ובכך נבין שיש לנו מסלול .

לדוגמא :



V5 מקשר לנו בין קודקודים V4 ו V6 .

מהלך האלגוריתם :

נעבור בשלושה לולאות for (שתי לולאות עבור מעבר על כל המטריצה ולולאה חיצונית שתיקח קודקוד (K) ותבדוק האם הוא מחבר בין i ל j) על המטריצה וניזהר לדרוס מסלול שכבר קיים לכן נוסיף תנאי שאם יש מסלול אז שידלג עליו.

פסואדו קוד :

```
for k from 1 to |V|
   for i from 1 to |V|
      for j from 1 to |V|
         if dist[i][j] > dist[i][k] + dist[k][j]
            dist[i][j] ← dist[i][k] + dist[k][j]
        end if
       end-for
   end-for
end-for
```

מסקנות :

1. כעת אם נרצה לדעת האם יש מסלול בין שני קודקודים נפעיל FW ואז ב O(1) ניגש לתא ונראה האם קיים שם 0 או 1.

2. בנוסף האם כל המטריצה מלאה באחדות אזי הגרף קשיר כי האחדות מייצגות את הקשר בין הקודקודים (נעבור רק על השורה הראשונה ונראה רק אם היא מלאה באחדות וזה יהיה זול יותר כלומר ב O(n)).

3. מציאת רכיבי קשירות – נייצר מערך עזר כמספר הקודקודים, נעבור על המערך וכאשר נראה 0 נבין שעוד לא הגענו לקודקוד הזה , ניגש למטריצת השכנויות נבדוק מי השכן שלו ונסמן במערך העזר, כעת נצהיר על counter שיהווה לנו את מס' רכיבי הקשירות ונעלה אותו בכל איטרציה עד שכל המערך ריק מאפסים כאשר ה counter שלנו הוא מס' רכיבי הקשירות.

4.מציאת מטריצת מסלולים – החזרת המסלול עצמו בstring ,נייצר מטריצת עזרת של מחרוזות ובכל תא נרשום איך הגענו אליו (איך הגענו מקודקוד לקודקוד) .

בעיית השוקולד :

תיאור הבעיה : נתון פס שוקולד בעל n קוביות , על כל חלוקה (שבירה של הפס) באינדקס K נשלם k(n-k) , כאשר המטרה היא לחלק את הפס לקוביות בודדות במינימום האפשרי.

פתרון הבעיה : לא משנה איך תחתוך את הפס אתה תגיע לתוצאה של n(n-1)/2 , כאשר ההוכחה היא באינדוקציה.

טבלה מסכמת עבור FW :

סוג הבעיה	סיבוכיות	תיאור
בעיית הבקבוקים	O(n^3)	שלושה לולאות for.
קיום מסלול	O(n^2)	מעבר על המטריצה.
בדיקת קשירות	O(n)	מעבר על השורה הראשונה.
מציאת מס' רכיבי קשירות	O(n^2)	מעבר על המטריצה.
מטריצת מסלולים	O(n^2)	ניצור מערך עזר של strings

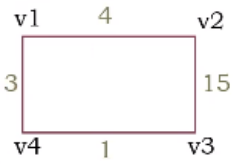
תרגול 3 :

משקלים עם צלעות :

תיאור הבעיה : כהמשך לפתרון FW שראינו בעבר כעת הצלעות הם בעלי משקל ואנו רוצים לדעת האם קיים קשר בין זוג קודקודים או לא ולדעת את העלות המינימלית של המסלול.

נחשב את מטריצת השכנויות – בהתחלה נסמן את האלכסון הראשי באפסים (כי אין לו משקל בינו לבין עצמו), במידה והוא שכן נכניס את המשקל שלו אחרת נכניס אין סוף.

לדוגמא :



	v1	v2	v3	v4
v1	0	4	∞	3
v2	4	0	15	∞
v3	∞	15	0	1
v4	3	∞	1	0

כעת נפעיל את FW כדי למצוא אם קיים מסלול בין שני קודקודים וגם נמצא את המסלול הקצר ביותר.

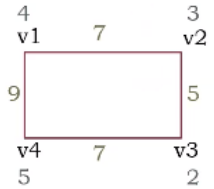
כיוון שFW לוקח את המינימום בין שני המסלולים הנתונים בתנאי אזי אלגוריתם זה ייתן לנו גם את המסלול הקצר ביותר.

כדי למצוא את מטריצת המסלולים נשתמש באותה השיטה מתרגול 2.

משקלים על הקודקודים:

תיאור הבעיה : כעת קשה יותר לייצג את המשקלים במטריצה שמייצגת קודקודים אז איך בכל זאת נמצא את המסלול הקצר ביותר ?

פתרון : נחבר את הערכים בין שני הקודקודים וזה יהיה המשקל של הצלע שבינהם שאותה נכניס למטריצה ,לדוגמא :



אבל כעת יש לנו בעיה , שאנחנו סופרים פעמיים קודקוד שמחבר בין שני קודקודים לדוגמא : V2 שמחבר בין V1 ל V3 נספר גם בחישוב צלע של v1,v2 וגם ב v2,v3 .

הפתרון לבעיה זו היא להוסיף את הקצוות ולחלק ב2.

נכניס את הערכים למטריצת שכנויות ונפעיל FW.

נסמן אינסוף – ע"י הוספת תנאי לפני ביצוע הFW שאם אף אחד מהערכים לא שווה למינוס 1 או MaxValue אזי תבצע את התנאי (החשש הוא שהוספה לאינסוף יהפוך למספר שלילי ואז FW ייקח את המספר החיובי בmin).

תרגול 4 :

מעגלים :

מעגל שלילי בגרף לא מכוון – כאשר יש צלע אחת בעלת משקל שלילי אזי מתקיים מעגל שלילי כיוון שאני יכול להגיע מהקודקוד לעצמו.

מציאת הצלע השלילית היא o(m) כאשר M היא מספר הצלעות.

מעגל שלילי בגרף מכוון – קודם נתבונן שקיים מעגל – אבל איך נדע ללא ציור מתי יש מעגל שלילי ומתי אין ? אז קודם נפעיל FW שנותן לנו את כל המסלולים בגרף הקצרים ביותר, כאשר האלכסון הראשי מראה לנו את המרחק מהקודקוד לעצמו וכאשר באלכסון זה נמצא מספר שלילי נדע אם יש מעגל שלילי בגרף. (FW לא פותר לא הבעיה כמו שצריך (לא נותן את הערך המדויק) אך עדיין ניתן להסיק כי יש מעגלים שלילים).

דוגמא למעגל שלילי – לאחר שנפעיל FW כאשר נזהה באלכסון הראשי מספר שלילי נחזיר את המסלול מהקודקוד לעצמו (כלומר נחזיר מחרוזת שמייצגת את המסלול).

מציאת תת מערך עם סכום מקסימלי :

תיאור הבעיה – בהינתן מערך בעל n איברים מהו תת המערך שסכום אבריו הגדול ביותר ומהו תת המערך הקצר ביותר ? (כאשר תת המערך חייב להיות רצוף) .

פתרון – נחלק למקרים :

כאשר כל התאים חיוביים - אז כל המערך הוא הסכום והתשובה היא המערך עצמו.

כאשר יש מספרים שליליים – נתבונן האם משתלם לקחת את התא בעל הערך השלילי בכדי להגדיל את הסכום כלומר,(כלל אצבע) אם המספר השלילי משאיר אותנו בסכום חיובי שווה לי לצרף אותו כי הוא מגדיל לי את הסכום הכולל ואם לא נתחיל ספירה חדשה ,לדוגמא : 4 , 1 –, 2 שווה לי לצרף את 1- כדי לקבל את הסכום 5.

סדר פעולות :

1 . נייצג את המערך במטריצה, באלכסון הראשי וכל תת מערך מיוצג ע"י תא מסוים.

2. נמלא את המשולש העליון של המטריצה כאשר נסכום את התאים המדורגים וכך נקבל את כל תתי המערכים וכל התוצאות האפשריות.

-10	2	4	3	-3

-10	-8	-4	-1	-4
	2	6	9	6
		4	7	4
			3	0
				-3

לאחר המילוי נתבונן איפה נמצא המספר המקסימלי ביותר ומה הרצף שהרכיב אותו וברצף זה נבחר כתת המערך הטוב ביותר.

סיבוכיות האלגוריתם : O(n^2) (for בתור for).

תרגול 5 :

מציאת תת מערך עם סכום מקסימלי - Best :

פתרון בסיבוכיות של O(n) –

נבנה מערך עזר שיסכום לנו את הסכום עד אותו תא שאנו נמצאים עליו במערך המקורי וכאשר נגיע לסכום שלילי נתחיל לספור מחדש, כלומר מערך העזר מייצג לי את הסכום מהתחלה או מספירה מחודשת עקב סכימה שלילית.

התוצאה תיהיה המספר המקסימלי כאשר נקודת ההתחלה תיהיה תחילת מערך העזר או התא הבא אחרי המספר השלילי.

לדוגמא :

- דוגמת הרצה: נתון מערך:

1	7	3	-13	2	1	10	-2	1	-20

- נבנה מערך עזר:

1	8	11	-	2	3	13	11	12	-

מערך מעגלי -

מה נעשה כאשר המערך מעגלי והתוצאה שקיבלנו לאחר הפעלת Best היא לאו בהכרח הטובה ביותר ?

לדוגמא במערך הנתון 7,-9,2,1 לאחר הפעלת Best נקבל 7 ואילו התוצאה היא 10 כאשר המערך מעגלי.

פתרון-

א. מטריצה – נמלא אותה באותו אופן כמו בתרגול 4 אך הפעם נתבונן גם במשולש התחתון שמייצג את המעגליות וניקח משם את המספר הגדול ביותר.

מילוי משלוש תחתון – נבצע את המשלים של הסדרה כלומר ניתן להסתכל על סכום של סדרה בצורה בה נחסיר את האיבר שאנו לא נשתמש בו כדי לקבל את סכום המקסימלי ונחבר אותו עם הסכום הכולל ונקבל את הסכום המקסימלי.

סיבוכיות : O(n^2)

ב. מערך כפול – נשכפל את המערך ונפעיל Best על כל אופציה מעגלית של המערך הנתון וניקח את המקסימום מבין כולם כלומר אם נתון לי מערך בעל ארבעה אבירים יכול להיווצר מצב שלאחר השכפול הסכום יהיה מורכב מיותר מארבעה אברים וכדי למנוע את החריגה הזו נפעיל best רק על ארבעה תאים בכל פעם.

בתכנות נעשה מודלו ולא נצטרך לשכפל את המערך עצמו.

סיבוכיות : O(n^2)

ג. **הפתרון היעיל ביותר** O(n)– ניקח את הסכום הכולל ונפחית ממנו את הסכום המינימלי ובכך נמצא את הסכום המקסימלי לצורה המעגלית , נבצע חישוב של הצורה הסטנדרטית וניקח את המקסימום בינהם.

את הצורה המעגלית נמצא ע"י שהBest ימצא את הסכום המינימלי שאותו נחסיר כנ"ל או שנשנה את הנתונים במערך הנתון, כלומר נכפיל ב1- ונקבל את המערך ההופכי שלו ולאחר שנבצע Best רגיל נמצא את הסכום המינימלי ונחבר אותו יחד עם הסכום הכולל וזה יהיה הסכום המעגלי המקסימלי.

בעיית תחנות הדלק –

תיאור – נתון מסלול מעגלי עם תחנות דלק כאשר בכל תחנה ניתן למלא כמות ליטרים מסויימת כאשר כל מקטע בין שתי תחנות לוקח כמות ליטרים מסויימת.

האם ניתן להתחיל בתחנה מסויימת ולבצע סיבוב שלם כאשר אני עובר בכל התחנות בלי שנגמר הדלק ?

פתרון – אנו צריכים למצוא את התחנה הראשונה שבה אני אצא לדרך.

בנוסף נצטרך לוודא שלא נתקע בדרך ע"י זה שנבדוק שהסכום הדלק שנמלא גדול מהצריכה עצמה , אם הצריכה גדולה מהדלק אז אין פתרון לשאלה כלל.

נקבל את הקלט כ2 מערכים – צריכה ותחנות.

לאחר שנבצע את הבדיקה שסכום הדלק גדול מהצריכה ניצור מערך נוסף שמשמעות שלו זה חישוב של הדלק כאשר אתה מגיע לתחנה הבאה ולכן מתוצאות המערך נוכל לבחור את תחנת ההתחלה (הערך שלא יהיה שלילי).

כדי לצבור את כמות הדלק הגדולה ביותר נצטרך למצוא את תת המערך עם הסכום המקסימלי וכך נדע מאיזה אינדקס להתחיל כלומר נבצע את ה best שלמדנו והתוצאה שהוא ייתן זו תיהיה נקודת ההתחלה.

פסואדו קוד best :

```
Best(A)
        int s = 1, e = 1, sum = 0, max = -inf , t_s = 1
        for i = 1 to A.length
                sum = sum + A[i]
                if(sum > max)
                        max = sum
                        s = t_s
                        e = i
                if(sum < 0)
                        sum = 0
                        t_s = i+1
        return (max, s, e)
```

תרגול 6:

Dikjstra , Dikjstra דו כיוונית.

תיאור ומהלך האלגוריתם – אלגוריתם חמדן שמטרתו היא לחשב את המסלול הקצר ביותר מקודקוד אחד לשאר הקודקודים.

לוקח את התוצאה הטובה ביותר בכל מצב נתון כלומר בכל צומת האלגוריתם יבחר ללכת לצלע בלעת המשקל הנמוך ביותר.

נאתחל את כל הקודקודים לאינסוף ואת קודקוד ההתחלה ל0.

נכניס את קודקוד ההתחלה ואת השכנים שלו לקוד ונעדכן את המרחק מקודקוד ההתחלה לכל שכן ונוציא את קודקוד ההתחלה.

נבחר את הקודקוד המינימלי בתור ונעדכן את המרחק ממנו אל השכנים שלו ונכניס אותם לתור.

נמשיך כך עד שהתור יתרוקן כלומר כל קודקוד יעד קיבל ערך.

שחזור המסלול – נשמור את קודקוד האב ונשאל מי הבא שלך בכל פעם עד שנגיע להתחלה.

פסואדו קוד –

```
Dijkstra(G, v0):
for each v in V:
    d[v] = ∞
    f[v] = null
d[0] = 0
Queue q ← V
while q not empty:
    u ← q.dequeue() //שליפה של הקודקוד המינימלי
    for each w in N(u):
        if d[w] > d[u]+Mat[u,w]
            d[w] = d[u]+Mat[u,w]
            f[w] = u
return f,d
```

כאשר נרצה לקבל מסלול קודקוד ספציפי נשנה את התנאי ונחכה שהוא ייצא מהמחסנית ונעצור את האלגוריתם.

Dikjstra דו כיוונית – באלגוריתם זה נתקדם גם מנקודת ההתחלה וגם מנקודת הסיום בו זמנית.

כלומר הרעיון הוא לגלות כל פעם את השכנים וליצור הפעם שני מעגלים שמבטאים את השכנים של שני נקודות ההתחלה בניגוד לדייקסטרה רגילה שבה אני יוצר מעיין שכבות סביב נקודת התחלה אחת.

שיטה זו חוסכת לי את זמן הריצה כמעט בחצי רק במקרה שיש שני מעבדים בצורה מקבילה.

מהלך האלגוריתם –

נבנה גרף רוורס כלומר נהפוך את כיווני הצלעות.

נריץ את האלגוריתם דייקסטרה לשני נקודות ההתחלה כלומר פעם אחד בצורה רגילה על הגרף הרגיל ופעם שניה בגרף הרוורס, נעשה שלב פה ושלב שם בצורה מקבילה כך שנגיע עד אמצע הדרך וזו תיהיה נקודת העצירה (המסלול הקצר ביותר לא חייב לעבור דרך נקודת האמצע).

תנאי העצירה יהיה קודקוד שיצא גם מהתור של הגרף הרגיל וגם בגרף רוורס.

לבסוף נחשב את המסלול הקצר ביותר בין שני הנקודות.

תרגול 7:

מציאת תת מטריצה בעלת סכום מקסימלי :

תיאור הבעיה : מטריצה בגודל m*n מהי תת המטריצה שסכום אבריה הוא הגדול ביותר ? (כמו best רק במטריצה ולא מערך),המטריצה חייבת להיות רצף של תאים.

פתרון :

1. חיפוש שלם – חיפוש שלם עובר על כל תתי המטריצות ומחשבת את הסכום שלהם

חסרון – זמן ריצה מאוד גדול.

נגדיר מלבן – נגדיר ארבעה משתנים כאשר כל זוג מגדיר את פינות המלבן, כלומר שתי משתנים יגדירו את הפינה השמאלית העליונה ושניים יגדירו את הפינה הימנית התחתונה.

לכן נצטרך ארבעה לולאות for רק כדי להגדיר את המלבן.

כעת נוסיף עוד שני לולאות פור כדי לעבור על כל תתי המטריצות ולסכום אותם.

סה"כ : 6 לולאות for כלומר : **O(n^6)**.

פסאודו קוד :

```
for i=1 to n
    for j=1 to m
        for k=i to n
            for l=j to m
                for x=i to k
                    for y=j to l
                        נחשב סכום
                        נעדכן מקסימום ואינדקסים
```

2. מערך עזר – נרצה לצמצם את כמות הלולאות לכן פה נשתמש ב best לעזרתנו, כלומר נייצר את כל תתי שורות ואת כל תתי עמודות ל best וכך נדע את התשובה.

למעשה ניקח שתי שורות מהמטריצה המקורית ונצמצם אותם לשורה אחת ע"י פעולת סכום – על השורה שנקבל נעשה best והתאים שנקבל כתוצאה הם יהיו התת מטריצה.

כך נעשה על כל תתי השורות האפשריות – נכניס אותם למערך עזר ואז נפעיל עליהם best .

סיבוכיות : **O(n^4)** - ארבעה לולאות for . שני לולאות עבור ייצור כל תתי השורות ועוד שתי לולאות עבור סכימה של כל תתי המטרציות.

פסואדו קוד:

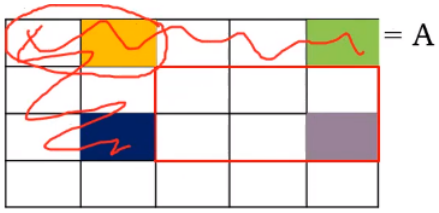
```
for i=1 to n
  for j=i to n
  איפוס של מערך העזר
    for k=i to j
      for l=1 to m
        arr[l] += Mat[k,l]
    Best(arr)
    נעדכן מקסימום ואינדקסים
```

3.מטריצת עזר – בפתרון זה נשתמש בתכנות דינאמי כלומר כאשר נקבל סכומים חלקיים ממטריצת העזר אז נוכל בעזרתם לחשב את הסכום הכולל.

כלומר נייצר מטריצת עזר שהערך בכל תא יהיה הסכום של המטריצה מנקודת ההתחלה ועד התא הנוכחי שאנו נמצאים בו עכשיו, וכך נוכל לשלוף סכום ב O(1) .

כעת ממטריצת העזר נשלוף את הסכום הכולל (כאשר הוא נמצא תמיד בקצה של השורה או בתא הימני התחתון).

לדוגמא :



הריבוע הסגול מסמן את הסכום של המטריצה מתא (0,0) ועד אליו , וכאשר נרצה לחשב את הסכום של תת המטריצה המסומנת בריבוע המסומן באדום דק נצטרך לחשב את תתי המטריצות המתחילות בתא (0,0) ועד הריבוע הירוק, כך גם לריבוע הכחול ולהוסיף את הריבוע הצהוב (כי הורדנו אותו פעמים).

כלומר : סכום תת מטריצה המסומנת באדום תהיה שווה : סגול – ירוק - כחול + צהוב.

כלומר ממטריצת העזר שבה נמצאים כל הסכומים אני שולף במהירות את הסכומים שאני צריך ומחשב בפשטות.

מהלך האלגוריתם -

נמלא את מטריצת העזר ההתחלתית ע"י הנוסחאות הבאות : כאשר A היא המטריצה המקורית וH היא מטריצת העזר.

לשורה ולעמודה הראשונה : H[0,j] = H[0,j-1]+A[0,j]

לשאר המטריצה : H[i,j] =A[i,j]+H[I,j-1]+H[i-1,j]-H[i-1,j-1]

לאחר בניית המטריצה נעבור עליה ונחשב את הסכומים של כל תתי המטריצות האפשריות :

עבור הדוגמא הנ"ל נעשה :

- for i=1 to n
  - for j=1 to m
    - for k=i to n
      - for t=j to m
        - sum = H[k,t] – H[k,j-1] – H[i-1,t] + H[i-1,j-1]
        - נעדכן מקסימום ואינדקסים

סיבוכיות : **O(n^4)**

4. **הפתרון היעיל ביותר – super best :**

בכדי להוריד את הסיבוכיות בכל אחד מהמקרים הנ"ל השתמשנו במערך או מטריצת עזר בכדי לשמור סכומי ביניים ובנוסף השתמשנו בbest.

אנו נשכלל את שני הכלים האלו בכדי להגיע לסיבוכיות של O(n^3) .

כעת במקום לאפס את מערך העזר בכל פעם שאנחנו עושים best נשתמש במערך הקיים ונוסיף רק את העמודה/השורה החדשה ונסכום אותה יחד עם העמודה הישנה, ושוב נפעיל best ונמצא את כל תתי המטריצות הקשורות לתאים אלו.

כלומר,כיוון שאנחנו מחשבים סכום אין סיבה לאפס את המערך עזר שאיתו אנו עושים את best ניתן תמיד להוסיף לו את האיטרציה הבאה ובכך אנחנו חוסכים לולאת for בה אנו היינו מאפסים את מערך העזר בכל פעם ומורידים את הסיבוכיות.

פסואודו קוד :

- for i=1 to m
  - איפוס של מערך העזר
  - for j=i to m
    - for k=1 to n
      - arr[k] += Mat[k,j]
    - Best(arr)
    - נעדכן מקסימום ואינדקסים

סיבוכיות :סדר גודל : **O(n^3)** , הסיבוכיות האמיתית היא **O(n*m^2) .**

פסואדו קוד super best :

```
cycleBest(A)
        sum = 0
        B = new Array[A.length]
        for i = 1 to A.length
                sum = sum + A[i]
                B[i] = -A[i]
        (b1, s1, e1) = Best(B)
        (b2, s2, e2) = Best(A)
        if(sum-(-b1) < b2) return (b2, s2, e2)
        else return (sum+b1,(e1+1)%A.length,(s1-1)%A.length)
```

תרגול 8:

BFS- מציאת המרחק הקצר ביותר בגרף.

תיאור האלגוריתם - בניגוד לDFS שעובד לעומק BFS עובד לרוחב, בצורת שכבות, בכל פעם אני מגלה שכבה חדשה מקודקוד ההתחלה.

חיפוש לרוחב עובד על הקשתות של קודקוד המקור וכך ממשיך עד היעד, בכל פעם נספור את הקשתות ומספר הקשתות מקודקוד המקור לשאר הקודקודים יהיה המרחק הקצר ביותר.

לכן קודקוד במרחק K יתגלה לפני קודקוד במרחק K+1.

האלגוריתם זה יכול לעבוד על גרף משוקלל או לא.

מהלך האלגוריתם –

האלגוריתם מחלק את כל הקודקודים לשלושה צבעים (ניתן גם בשתי צבעים) :

לבן – טרם התגלה.

אפור – התגלה ולא טופל.

שחור – טופל.

כל קודקוד שמתגלה הופך להיות האבא של שכניו ותמיד יש אבא אחד.

את הקודקוד הזה נכניס לתור ותמיד נשלוף את השכנים של הקודקוד הזה ועליהם נעבוד ונעדכן את המרחקים.

כל קודקוד ייכנס לתור פעם אחת וכל עוד התור לא ריק לא עברנו על כל הגרף.

פסאודו קוד –

```
- BFS(G,s):
- for each vertex in V(G)\{s}
    - color[u] = WHITE
    - d[u] = ∞
    - f[u] = null
- color[s] = GRAY
- d[s] = 0
- f[s] = null
- Queue q
- Enqueue(q, s)
- while q not empty:
    - u = dequeue(q)
    - for each v in N(u):
        - if color[v] = WHITE:
            - d[v] = d[u]+1
            - f[v] = u
            - Enqueue(q, v)
    - color[u] = BLACK
```

סיבוכיות האלגוריתם – O(|E+V|) - עוברים על כל הקודקודים והקשתות.

מסקנות-

1. אם על הקודקודים שחורים או אם יש ערכים לכל הקודקודים אזי הגרף קשיר.

2. במידה והגרף לא קשיר נוכל לדעת כמה רכבי קשירות יש כאשר נפעיל את האלגוריתם על הקודקודים שלא הגענו אליהם (הלבנים) וכך נעשה בכל פעם עד שכולם יצבעו בשחור וכל איטרציה זהו מספר רכבי הקשירות.

3. כדי לדעת את הקודקודים המרכיבים את רכבי הקשירות ניצור בכל איטריצה מערך שישמור לנו את הקודקודים.

מציאת קוטר של גרף

הגדרה – קוטר של גרף הינו המרחק המקסימלי בין שני קודקודים בגרף.

אלגוריתם א' למציאת הקוטר – נשתמש בBFS למציאת המרחק בין כל הקודקודים ונחזיר את המרחק הגדול ביותר.

פסאודו קוד –

```
diam = 0
for each s in V:
    call BFS(G,s)
    max = find max value in d
    if max > diam:
        diam = max
return diam
```

סיבוכיות – O(|V|*|V+E|) - עוברים על כל הקודקודים ועבור כל קודקוד מפעילים BFS .

אלגוריתם ב' למציאת הקוטר – נצמצם את השימוש בBFS ע"י גיאומטריה במישור כלומר קוטר במעגל חייב לעבור במרכז המעגל ולכן נעתיק את הרעיון הזה גם לגרף.

מהלך האלגוריתם – ניקח נקודה שרירותית מהגרף ונבצע עליה BFS.

נבחר מהמערך את הקודקוד עם הערך המקסימלי ביותר ועל קודקוד זה נפעיל BFS וכך בוודאות נקבל את המרחק הגדול ביותר.

כלומר במקום לעבור קודקוד השתמשנו רק פעמיים בBFS.

פסאודו קוד –

```
select s from V
call BFS(G,s)
u = find the vertex with max value in d
call BFS(G,u)
return max value in d
```

סיבוכיות – **O(|V+E|)**

תרגול 9 :

אלגוריתם שריפה

הגדרות ומושגים:

עץ – גרף קשיר ללא מעגלים , גרף קשיר עם n-1 צלעות , גרף קשיר עם n-1 צלעות ללא מעגלים.

מרכז - הנקודה ממנה יוצא הרדיוס.

רדיוס – קטע המחבר בין מרכז המעגל לשפת המעגל.

קוטר – הקטע הארוך ביותר בין 2 קצוות המעגל, חייב לעבור במרכז המעגל.

כעת נעביר את המושגים הללו לעולם העצים ונתעניין בעיקר בערך המספרי שלהם.

קוטר – מרחק מקסימלי בין שני קודקודים (נספור את הצלעות), נמצא אותו פעמיים ע"י הפעלת BFS פעמיים (תרגול 8).

רדיוס –במידה והקוטר אי זוגי אזי זה יהיה המרחק הגדול ביותר מהמרכז לשפת המעגל או במידה והקוטר זוגי אזי חצי מהקוטר.

מרכז המעגל – אמצע הקוטר (יכול להיות שתי קודקודים במקרה שהקוטר אי זוגי).

סיבוכיות מציאת הקוטר – O(|V+E|) .

תיאור האלגוריתם – העיקרון המלווה של אלגוריתם זה הוא הורד עלים מהעץ,

וברגע שמורידים אנו עדיין נשאים עם עץ כיוון שלא נגענו בצלעות ועדיין אנו עונים על ההגדרה.

כלומר נשרוף כל פעם עלים עד שנגיע למרכז המורכב מקודקוד אחד או שני קודקודים וכך נוכל לדעת מה הקוטר ומה הרדיוס.

השריפה תתבצע בצורת שכבות כלומר בכל איטרציה נשרוף רק את העלים החיצוניים עד שנגיע למרכז אחד או שניים (זה תנאי העצירה).

אם המרכז מורכב מקודקוד אחד אזי הקוטר זוגי.

אם המרכז מורכב משני קודקודים אזי הקוטר אי זוגי.

הרדיוס יהיה כמות השרפות במקרה שהקוטר זוגי, ובמקרה שהקוטר אי זוגי הוא יהיה כמות השריפות +1 .

והקוטר יהיה פעמיים כמות השריפות במקרה שהקוטר זוגי, ובמקרה שהקוטר אי זוגי הוא יהיה פעמיים כמות השריפות פחות אחת.

לסיכום –

מרכז מקודקוד אחד - קוטר זוגי , רדיוס שווה לכמות השריפות ,קוטר שווה פעמיים כמות השריפות.

מרכז משני קודקודים – קוטר אי זוגי ,רדיוס שווה לכמות השריפות +1 ,קוטר שווה פעמיים כמות השריפות-1.

מהלך האלגוריתם -

נקבל את הגרף כקלט של מערך שכנויות (מערך שבו לכל קודקוד יש עוד מערך שמייצג את השכנים שלו).

נזהה את העלים מרשימת השכנויות כאשר לקודקודים שיש שכן אחד הם עלים, את אותם העלים נשלוף ונשים במערך עזר (כאשר כל תא במערך העזר ייצג לנו גם את דרגות הקודקודים).

כעת נשמור את כל העלים בתור ובכך "נשרוף" אותם.

לאחר השריפה נעדכן את מערך העזר ונוריד את הדרגות כאשר בכל פעם אני לוקח מהמערך את הדרגה 1 שהיא מייצגת עלה.

נחזור על התהליך הזה עד שנגיע למרכז (האחרון בתור).

סיבוכיות : O(|V|) .

פסואדו קוד :

```
FindCenters(G)
        L = new List // רשימת עלים
        for each v in G
                if(G[v].size == 1)
                        L.add(v)
        G' = G
        n = |V|
        while(n > 2)
                temp = new List
                for each v in L
                        n--
                        u = G'[v].get(1) // פונים לשכן היחיד של העלה
                        G'[u].remove(v) // מחיקת העלה מתוך רשימת השכנים של השכן שלו
                        if(G'[u].size == 1)
                                temp.add(u)
                L = temp
        return L
```

תרגול 10 :

בנית עץ מרשימת דרגות :

תיאור הבעיה : בהינתן רצף מספרים מסוים האם הוא יכול לבטא רשימת דרגות של עץ ?

הגדרות :

כמות הצלעות = |E|=|V|-1 – כמות הצלעות שווה לקמות הקודקודים פחות 1.

סכום הדרגות : סכום הדרגות שווה לפעמים סכום הצלעות.

מסקנה : כמות הדרגות שיש בגרף יהיה שווה ל 2(|V|-1) (הצבה פשוטה של |E|).

כעת נשמש במסקנה כדי לדעת אם רשימה היא דרגות של עץ או לא.

לדוגמא :

1,1,1,1,2,2,3,4

- נסכום את האיברים: 1+1+1+1+2+2+3+4 = 15
- נסיק מסקנה לגבי כמות הקודקודים:

$$2\,(|V| - 1) = 15$$
$$|V| - 1 = 7.5$$
$$|V| = 8.5$$

כאשר מספר הקודקודים הוא 8.5 ולכן זה לא יכול להיות עץ.

ולכן כאשר הסכום הוא זוגי הרשימה יכולה לתאר עץ.

כעת יש מצב בו אנו מקבלים לפי החישוב סכום שהוא גדול יותר ממספר הקודקודים הנתונים ברשימה.

ולכן למסקנה , לאחר שנבדוק שהסכום שקיבלנו הוא מספר שלם נבדוק גם שהמספר שווה לכמות הקודקודים שקיבלנו לפי הרשימה.

לדוגמא :

- בואו נראה דוגמא נוספת:

1,1,1,1,2,2,3,3

- נסכום את האיברים: 1+1+1+1+2+2+3+3 = 14
- נסיק מסקנה לגבי כמות הקודקודים:

$$2\,(|V| - 1) = 14$$
$$|V| - 1 = 7$$
$$|V| = 8$$

- וזה אכן מתקיים!

רעיון האלגוריתם : בכל עץ יש לי לפחות שני עלים, לכן ננסה לחבר במידה ואפשר עלה לקודקוד שהדרגה שלו גדולה מ1 (כדי שיישארו עלים) כלומר ניצור עץ קשיר.

לאחר החיבור נוריד את דרגות הקודקודים שחיברנו ונמשיך כך עד שכל הערכים במערך יהיו 0 וכך נבנה את העץ.

(נמיין את המערך לפני).

פסואדו קוד :

```
BuildTreeFromDegreesArray(deg[]):
N = deg.size()
sum = 0
tree[N] = null
for i=1 to N:
    sum += deg[i]
if sum/2 + 1 != N:
    print(Not a tree degrees array)
    return
deg[] = sort(deg[])
j = first index that deg[j]>1
for i=1 to N-2:
    tree[i] = j
    deg[j] = deg[j]-1
    if deg[j] == 1
        j = j+1
tree[N-1] = N
return tree
```

זה רץ עד n-2 כיוון ששני הקודקודים האחרונים בטוח יתחברו ביחד ואז בשורה הלפני האחרונה אתן לשורש את הערך null.

סכום המטריצה הגדול ביותר :

תיאור הבעיה : נתונה מטריצה שמלאה ב1 ומינוס 1 בצורה רנדומלית.

אנו רוצים להפוך את המטריצה לבעלת הסכום הגדול ביותר.

מותר לי לקחת כל שורה או עמודה ולהכפיל במינוס 1 .

התהליך הוא סופי – כלומר לאחר מספר סופי של איטרציות אנחנו נגיע בהכרח לסכום הגדול ביותר.

כלומר יש לנו חסם עליון כי לא יכול להיות שהסכום יהיה יותר גדול ממספר התאים במטריצה (כי במקסימום המטריצה יכולה להיות מלאה ב1).

פסואדו קוד :

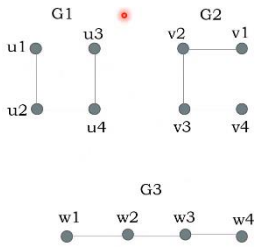
```
BestMatrix(M)
    n := M.rows
    m := M.cols
    max = −∞
    row_s, row_e, col_s, col_e
    H = new Matrix[n][m]
    for j = 0 to m-1 //אתחול שורה ראשונה
        H[0][j] = M[0][j]
    for i = 1 to n-1
        for j = 0 to m-1
            H[i][j] = H[i-1][j] + M[i][j]
    for i = 0 to n-1
        for t = i to n-1
            A = new Array[m]
            for j = 0 to m-1
                if(i == 0) A[j] = H[t][j]
                else A[j] = H[t][j] - H[i-1][j]
            (sum, s, e) = Best(A)
            if(sum > max)
                max = sum
                row_s = i , row_e = t, col_s = s, col_e = e
    return (max, row_s, row_e, col_s, col_e)
```

תרגול 11 :

עצים איזומרפים :

הגדרה לגרפים איזומרפיים : גרפים G ו H יהיו איזומורפיים אם קיימת פונקציה מ f:V(G)->V(H) שהיא חח"ע ועל כל שעבור כל שני קודקודים מתוך V(G) מספר הקשתות המקשרות בינהם יהיה זהה למספר הקשתות המקשרות בפונקציה.

לדוגמא :



כלומר נחפש את אותה ההתנהגות אך לא בהכרח אותה צורה.

מציאת איזומרפיות :

כאשר מגדירים לנו קודקוד ספציפי שהוא השורש נחפש מקביליות בהתנהגות של שני העצים כלומר אם לעץ מספר 1 יש שלושה בנים וגם עץ 2 מתנהג אותו דבר אזי הם איזומרפיים כדי לוודא שהם איזומרפיים בוודאות נפעיל איזה מיון על העצים ואם נקבל את אותה התוצאה אזי הם איזומרפים.

מהלך האלגוריתם :

עצים עם שורש - נסרוק את הגרף לעומק ונשמור את הפעולות שלנו כאשר בכל פעם שנרד נרשום 0 ובכל פעם שנעלה נרשום 1 כלומר נקבל מחרוזת שמאפיינת לנו את ההתנהגות של העץ וכאשר נקבל מעץ אחר את אותה המחרוזת אזי הם איזומרפיים.

כעת לא בהכרח נקבל את אותם המחרוזות למרות שהעצים כן איזומרפיים אזי ניתן יהיה לסמן עבור כל עלה 01 ואז לכל אבא אני יכול לפתוח אותו ב0 ולסיים אותו ב1 ובאמצע לשים את הבנים שלו בצורת מיון ואז נגיע לאותם מחרוזות.

עצים ללא שורש – נבחר בכל פעם קודקוד אחר שיהיה השורש ונפעיל עליו את האלגוריתם הנ"ל או שנמצא את המרכז (אלגוריתם שריפה) שהוא יכול להיות השורש הפוטנציאלי ועליו נפעיל את האלגוריתם.

סדר פעולות (פסואדו קוד) :

- **עצים איזומורפיים- עצים עם שורש**
- אלגוריתם ליצירת מחרוזת המתארת עץ מושרש:
  1. נסרוק את הגרף לעומק
  2. כשנגיע לעלה, נגדיר את המחרוזת שלו כ-01
  3. כאשר נגדיר את המחרוזת של האבא (נגדיר אותה רק אחרי שנסיים עם כל המחרוזות של הבנים שלו) נתחיל אותה ב-0, נסיים אותה ב-1 והאמצע יהיה שרשור ממויין של מחרוזות הבנים.

תרגול 12 :

קידוד הופמן :

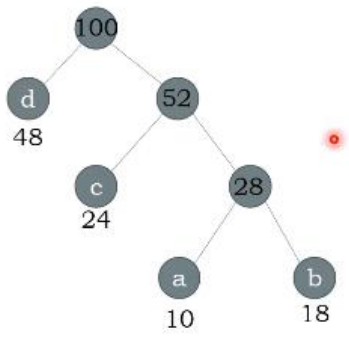
מטרת הקידוד היא לקודד טקסט במינימום ביטים כדי שנוכל לפענח אותו בחזרה בצורה מדוייקת כאשר אנו רוצים שהקידוד לא יתפוס הרבה מקום בזיכרון.

לדוגמא : נסווג את האות עם המופעים הכי רבים בשפה (מה שאנחנו מקבלים כקלט) כזאת עם תפיסת הזיכרון הקטנה ביותר.

כלומר אנחנו מקבלים כקלט שפה יחד עם אחוז מופע לכל שפה.

הקידוד יתבצע בעץ על פי השיעורים האחרונים כאשר האותיות בעלי האחוזים הגבוהים ביותר יהיו כמה שיותר קרובים לשורש העץ , כאשר האבא שלהם יהיה סכום האחוזים של שניהם.

- לדוגמא: a=10%, b=18%, c=24%, d=48%
- נרצה שככל שהתו מופיע יותר פעמים הוא יהיה קרוב לשורש העץ, וככל שהוא מופיע פחות הוא יהיה קרוב לעלים.
- בוא נבנה את העץ:



כעת נוכל לכתוב את הקידוד ובצורה בה הבן הימני יהיה 0 ואילו השמאלי 1.

ואז לדוגמא האות a=001 ואילו b=000 1 =d ,c=11 כאשר כל מילה שניצור מארבעת התווים האלה נוכל לפענח אחורה.

כעת נחשב את הזיכרון : ניקח את האחוז של האות ונכפיל אותה בביטים שלה כלומר a מורכבת משלושה ביטים והיא מופיעה 10% אזי a=3*10 ,כך נעשה לכל האותיות.

לכן בדוגמא שלנו : 180= 48*1+24*2 + 18*3 +10*3 , כאשר ה180 נחלק ב100 ונראה כמה כל תא תפס לי בזיכרון (1.8) בממוצע.

מימוש העץ :

דרך א' – סטטית – ברגע שיש 4 תווים אני יודע מה יהיה גודל העץ (מספר הקודקודים= 2n-1),וכאשר אני יודע את מספר הקודקודים אני יכול להיעזר במטריצה שתראה לנו את הקשר בין הבנים.

הסיבוכיות היא O(n^2) .

לדוגמא :

בן שמאלי 1	בן ימני 0	אבא			
		5	10	a	**1**
		5	18	b	**2**
		6	24	c	**3**
		7	48	d	**4**
1	2	6	28		**5**
3	5	7	52		**6**
4	6		100		**7**

- המטריצה ריקה כיוון שלכל העלים אין בנים ולשורש אין אבא.
- נשחזר את הקידוד ע"י זה שנשאל כל אבא מי הילד שלו והאם אותו הילד הוא בן ימני או שמאלי עד שנגיע לשורש.

דרך ב' – ערמת מינימום – אנו רואים שבדרך א' אנחנו כל הזמן מחפשים את המינימום כעת נממש תור עדיפויות שיחסוך לנו את זמן הריצה הזה (כיוון שבערמת מינימום השורש הוא האיבר המינימלי אזי כל פעם נשלוף את השורש).

פסואדו קוד :

```
Huffman(C):
n = |C|
Q = C
for i=1 to n-1:
    z = allocate Node()
    x = z.left = Extract_Min(Q)
    y = z.right = Extract_Min(Q)
    f(z) = f(x) + f(y)
    insert(Q,z)
return Extract_Min(Q)
```

כאשר f מייצג את האחוזים.

כאשר z זה הקודקוד שמחזיק את המידע.

<u>סיבוכיות :</u> O(nlogn).

דרך ג' – שימוש בשתי תורים – נניח ונותנים לי את התווים ממויינים , בצורה זו נשתמש ב2 תורים.

ניקח את תור הראשון ונכניס אליו את כל הערכים הנתונים לי בצורה ממויינת כעת אני יודע שהמינימום שלי הוא הראשון בתור וכאשר אשלוף אותו המינימום הבא שיהיה בתור יהיה גם כן הערך הנמוך ביותר כאשר כל זה עולה לנו O(1).

ניקח את שניהם נחבר אותם ונכניס לתור השני, כעת נשלוף את המינימום מבין שני התורים נחבר אותם ונכניס לתור השני ,בצורה זו נמשיך עד שיתרוקן התור הראשון.

כל הפעולות יהיו בO(1).

כמות האיטרציות תיהיה n-1 ולכן **הסיבוכיות תיהיה ב O(n).**

מרתון 13 :

מעגל אויילר :

הגדרה : גרף G לא מכוון מסלול יקרא מסלול אוילר אם הוא עובר בכל הצלעות בG כך שכל צלע מופיעה בו פעם אחת בלבד כלומר אני עובר על כל צלעות בגרף וקודקוד הסיום שלי הוא שונה מקודקוד ההתחלה.

מעגל אוילר – הוא מסלול אוילר סגור כלומר מסלול שבו קודקוד ההתחלה והסיום שווים (למשל מגן דויד).

גרף אוילנריאני – גרף שמכיל מעגל אוילר.

זיהוי אויילר בגרפים :

משפט 1 – יש בגרף G **מעגל** אוילר אמ"מ G קשיר וכל דרגות הגרף זוגיות.

כלומר כל הדרגות של כל הקודקודים זוגיות.

משפט 2 – יש בגרף G **מסלול** אוילר אמ"מ G קשיר וקיימים בדיוק 2 קודקודים בעלי דרגות אי זוגיות.

מהלך האלגוריתם :

1. נבחר קודקוד מסוים ונחיל ממנו מסלול עד שהוא סוגר מעגל בלי שנחזור על צלע פעמיים.
2. אם עברתי על כל הקודקודים נדלג לשלב הבא אחרת נוריד את המעגל ונחזור לסעיף 1.
3. אם עברנו על כל הקודקודים ויש רק מעגל אחד סיימנו. אחרת, נחיל מסלול ממעגל מסוים וכאשר נגיע למעגל חדש נסגור קודם את המעגל החדש ואז נחזור למעגל הקודם.

אלגוריתם נוסף למציאת אוילר :

1. נבחר קודקוד ונתחיל ממנו מסלול.
2. אם סגרנו מעגל נדפיס את הקודקודים בחזרה עד שיהיה אפשר להמשיך בדרך אחרת כלומר, בניגוד לפעם קודמת לא נוריד את המעגל אלא נבין שאם סגרתי מעגל ולא עברתי על כל הקודקודים אזי יש לנו בעיה וננסה להמשיך בדרך אחרת (קודקוד אחר) וממנו ננסה ליצור מעגל נוסף כך עד שנעבור על כל הצלעות.

פסואדו קוד :

```
- EulerCycle(G):
- stack S = null, stack C = null
- S.push(v0)
- while S is not empty:
  - u = s.top()
  - if deg(u)=0:
    - S.pop()
    - C.push(u)
  - else
    - v = v ϵ N(u)
    - S.push(v)
    - E = E – {u,v}
- return C
```

כאשר S הוא המחסנית שאיתה אני עובר על המסלול ומחסנית C בה אני מכניס את מה שאני מוחק.

V0 זה הקודקוד הנבחר.

V = הצלע הבאה שניתן לגשת אליה ,השכן.

E = מחיקת הצלע בה השתמשתי.

עץ פורש מינמלי – פרים :

נתון גרף עם משקלים על הצלעות אנו נצטרך לבחור את הצלעות המינימליות ביותר כבר שסכום העץ הפורש יהיה מינימלי.

אלגוריתם זה חמדני ומתאים לגרף ממושקל ולא מכוון.

מהלך האלגוריתם :

1. נבחר קודקוד באקראי.

2. בכל צעד נוסיף לעץ את הצלעות בעלות המשקל המינימלי היוצאות מקודקוד העץ ולא סוגרות מעגל (כי אנחנו רוצים עץ).

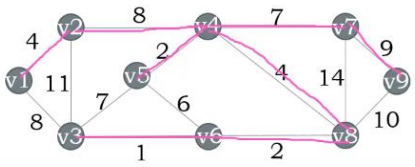
3. נוודא שלא סגרנו מעגל כאשר נרצה להוסיף צלע לעץ נוודא שרק אחד מהקודקודים השייכים לצלע קיים בעץ.

נשמור את העץ במערך אבות כאשר נשתמש גם במערך של משקלים כדי לדעת את המשקלים המינימלים.

לדוגמא:

	father	key	visited
v1	null	0	V
v2	v1	4	V
v3	v6	1	V
v4	v2	8	V
v5	v4	2	V
v6	v8	2	V
v7	v4	7	V
v8	v4	4	V
v9	v7	9	V

- לדוגמא:



פסואדו קוד :

```
    Prim(G, root):
o(n) for each v in V(G):
        visited[v] = false
        key[v] = inf
        father(v) = NULL
     key[root] = 0
o(n) Q = V(G) // Q = min heap
     while (Q is not empty) // or for i=1 to N-1
o(logn)  u = Extract_Min(Q)
         for each v in N(u)
             if(visited[v] == false and key[v] > weigh
                 key[v] = weight(u,v)
                 father[v] = u
o(logn)          decreaseKey(Q,v, weight(u,v))
         visited[u] = true
```

הroot זה הקודקוד שממנו נתחיל.

לאחר מכן הלולאה הראשונה היא עבור האתחולים.

Q שווה לערמת מינימום (כניסה ויציאה ממנה היא בסיבוכיות של logn).

לבסוף נחזיר את מערך האבות (return father).

סיבוכיות : O(|E|*log(|V|)).

אלגוריתמים ששחר לא נגעה בהם :

Kruskal :

תיאור הבעיה :נתון גרף עם משקלים על הצלעות אנו נצטרך לבחור את הצלעות המינימליות ביותר כבר שסכום העץ הפורש יהיה מינימלי (קשיר אך ללא מעגלים).

אלגוריתם זה הוא אלגוריתם חמדן.

מהלך האלגוריתם :

1. נמיין את הצלעות מהצלע הקטנה לגדולה (ע"פ משקלים).

2. נעבור על הצלעות מהקטנה לגדולה וכל צלע שלא סוגרת מעגל עם מה שלקחנו לעץ לפני כן – נוסיף לעץ הפורש, אך אם היא סוגרת מעגל – דלג עליה.

3. סיים כאשר נלקחו n-1 צלעות כאשר n הוא מספר הקודקודים.

נשמור את הקודקודים שהם באותו רכיב קשירות בתוך מבנה נתונים שנקרא union -find שמטרתו לאחד בין קבוצות זרות.

עלות השימוש במבנה נתונים זה היא O(log|V|).

הסיבה שעץ זה הוא עץ פורש מינימלי זה כיוון שאני ממיין ולוקח צלעות מהנמוך לגבוהה וכאשר נעצור אז כל הצלעות בעלי המשקל הנמוך ביותר בידנו.

סיבוכיות : O(|E|log(|E|)) .

פסאודו קוד :

```
Kruskal(G=(V,E))
        DisjointSets d = new DisjointSets(|V|)
        Tree T = Φ
        sort(|E|) // by weight
        for i = 1 to |E|
                e = E[i]
                if(d.union(e.v1, e.v2)) T.add(e)
                if(|T| == n-1) return T
        return T
```

d שווה למבנה הנתונים union – find .

T שווה לעץ שאותו נחזיר.

Sort עבור המיון של הצלעות.

If הראשון אם ניתן לאחד בין הצלעות תוסיף לעץ ואם לא תדלג.

If השני כאשר גודל העץ מגיע ל n-1 תעצור.

קורסקל הפוך – במקום להתחיל מהצלע בעלת המשקל הנמוך למשקל הגבוהה נתחיל מהיקר אל הזול ונוריד צלעות כאשר נשים לב שבהורדת הצלעות אנו לא פוגעים בקשירות הגרף.

בורובקה boruvkas :

רעיון האלגוריתם הוא לשלב בין קרוסקל לפרים יחד, כלומר ניקח את הצלע הכי טובה שמתאימה לכל רכיב קשירות (union find) .

כאשר כל עוד לא סיימנו (לא הגענו ל-n1) מצא את הצלעות הנמוכות (המתאימות) לכל רכיב והוסף אותו לעץ כל עוד הן לא סוגרות מעגל.

פסאודו קוד :

```
Boruvka(G=(V,E))
        DisjointSets d = new DisjointSets(|V|)
        Tree T = Φ
        isFinnished = False
        while(!isFinnished)
                cheapest = new Array[|V|] // init to null
                for i = 1 to |E|
                        e = E[i]
                        g1 = d.find(e.v1) , g2 = d.find(e.v2)
                        if(g1 ≠ g2)
                                if(e.weight < cheapest[g1].weight) cheapest[g1] = e
                                if(e.weight < cheapest[g2].weight) cheapest[g2] = e
                isFinnished = True
                for i = 1 to |V|
                        if(cheapest[i] ≠ null)
                                T.add(cheapest[i])
                                d.union(cheapest[i].v1, cheapest[i].v2)
                                isFinnished  = False
```

d שווה למבנה הנתונים find – union .

T שווה לעץ שאותו נחזיר.

Isfinnished – דגל בוליאני שיהיה התנאי ללולאת הוויל.

Cheapest מערך בגודל הצלעות שמחזיק את הצלע הכי זולה.

ה for הראשון כדי לעבור על כל הקודקודים כאשר g 1 הוא ראש הקבוצה לקודקוד של e1 ובהתאמה גם g 2.

אם הם בקבוצות שונות אז ה if הראשון והשני אומר שאם הצלע שאנחנו עליו נמצאים קטן יותר מהצלע של הראש קבוצה אזי תיקח את הצלע.

ה for השני עובר על מערך הcheapest ובודק אם הוא שונה מnull ניתן להוסיף אותו לעץ הפורש ותאחד בין קודקודי הקצה.

סיבוכיות : O(|E|log(|V|)

DFS – אלגוריתם חיפוש לעומק:

מטרת האלגוריתם היא לעבור על הגרף או לחפש בו.

מתחיל את החיפוש מקודקוד רנדומלי ומתקדם לאורך הגרף עד שהוא נתקע, כאשר הוא נתקע הוא חוזר על עקבותיו עד שהוא יכול לבחור קודקוד ממנו הוא יכול להמשיך את הסריקה על הגרף, כלומר קודקוד שהוא לא ביקר בו.

פסאודו קוד :

```
DFS(G):
        for each v in V(G)
                color[v] = WHITE
                prev[v] = NULL
                dist[v] = ∞
        for each v in V(G)
                if color[v] = WHITE
                        DFS_Rec(G,v)

DFS_Rec(G,v):
        color[v] = GRAY
        for each u in v.neighbors
                if color[u] = WHITE
                        prev[u] = v
                        DFS_Rec(G,u)
```

סיבוכיות : O(|V|+|E|)

טבלת סיבוכיות :

בעיה		סיבוכיות	הערות
FW	בעיית הבקבוקים	O(n^3)	שלושה לולאות for (FW).
	קיום מסלול	O(n^2)	מעבר על המטריצה.
	בדיקת קשירות	O(n)	מעבר על השורה הראשונה.
	מציאת מס' רכיבי קשירות	O(n^2)	מעבר על המטריצה.
	מטריצת מסלולים	O(n^2)	ניצור מערך עזר של strings.
Best	מציאת תת מערך עם סכום מקסימלי	O(n)	
	תחנת דלק	O(n)	
Super best	תת מטריצה בעלת סכום מקסימלי	O(n^3)/O(n*m^2)	
	סכום מטריצה הגדול ביותר	O(n^3)	מבוסס על best
סריקה	BFS	O(|E|+|V|)	
	DFS	O(|E|+|V|)	
	Dikjstra	O(|E|+|V|*log|V|)	רגילה ודו כיוונית
עצים	מציאת קוטר של גרף	O(|E|+|V|)	על בסיס BFS
	אלגוריתם שריפה (fire)	O(|V|)	משמש גם למציאת קוטר של גרף
	בנית עץ מרשימת דרגות	O(nlogn)	
עץ פורש מינימלי	פרים	O(|E|*log|V|)	
	קרוסקל	O(|E|*log|E|)	
	בורובקה	O(|E|*log|V|)	
קידוד הופמן	סטטית	O(n^2)	
	ערימת מינימום	O(nlogn)	
	שתי תורים	O(1)	נשתמש רק כאשר הערכים ממויינים