מרתון 1 - אלגוריתמים 2 - גיל לוי
[קישור להקלטה של המרתון]

ייצוג של גרפים במחשב:

1. רשימת סמיכויות: סיבוכיות מעבר: $O(|E| + |V|)$
מערך שהתא i מייצג את קודקוד i. והערך בתא הוא רשימה מקושרת של כל השכנים של קודוקד i (בגרף מכוון זה אומר שהחץ הוא מ i לשכן)
אם יש משקלים על הצלעות אז כל איבר ברשימה הוא מורכב מ Node שיש בו את האינדקס של הקודקוד השכן ואת משקל הצלע מ i לשכן.

2. מטריצת שכנויות: סיבוכיות מעבר: $O(|V|^2)$
מטריצה שהתא (i, j) מייצג האם יש צלע בין i ל j.
אם אין משקלים על הצלעות - זו מטריצה בוליאנית או (0/1)
אם יש משקלים על הצלעות - באלכסון יש תמיד 0 (כי קודקוד לא מחובר לעצמו). אם אין צלע אז המשקל הוא אינסוף ואם יש צלע אז המשקל מיוצג בתא (i,j).

מציאת מסלולים קצרים ביותר בגרף

מסלול קצר = הדרך להגיע מקודקוד אחד לאחר **במספר צלעות** מינימאלי. (אין משקל על הצלעות)
מסלול קל = הדרך להגיע מקודקוד אחד לאחר **בעלות כוללת** מינימאלית. (יש משקל על הצלעות)

- מסלול קל ביותר בין קודקוד לכל האחרים: Dijkstra
- מסלול קצר ביותר בין קודקוד לכל האחרים: BFS
- מסלול קצר/קל ביותר בין כולם לכולם: Floyd-Warshall

אלגוריתם פלויד וארשל:

קלט: מטריצת שכנויות.
הרעיון: כל קודקוד k משמש כמתווך בין כל 2 קודקודים: i,j.
פסאודו קוד: (כל המערכים מתחילים מאינדקס 1)

```
FW(G)
	D = new Matrix()
	P = new Matrix()
	for i=1 to |V|
		for j=1 to |V|
			D[i][j] = G[i][j]
			if(D[i][j]!=inf && D[i][j]!=0) P[i][j] = “->j"

	for k = 1 to |V|
		for i = 1 to |V|
			for j = 1 to |V|
				D[i][j] = min(D[i][j], D[i][k] + D[k][j])
				if(D[i][j]> D[i][k] + D[k][j]) P[i][j] = P[i][k] + P[k][j]
```

לאחר סיום האלגוריתם, בכל תא (i,j) ב D יהיה את אורך המסלול הקל ביותר בין i ל j.

הערה:
אלגוריתם פלויד ואשרל עובד נכון רק אם אין מעגלים שליליים בגרף.
איך מזהים מעגל שלילי? אם במהלך ריצת האלגוריתם, באלכסון יש מספר שלילי - זה אומר שיש מעגל שלילי.

סיבוכיות: $O(|V|^3)$.
הסבר:
נוכיח באינדוקציה על k שבשלב ה k אנו מוצאים את המסלול הקל ביותר בין כל 2 קודקודים שלא עובר בקודקודים שהאינדקס שלהם גדול ממש מ k.
בסיס: אתחול = אסור לעבור באמצע דרך אף קודקוד ולכן המסלול הקל ביותר הוא משקל הצלע (או אינסוף אם אין צלע).
צעד: נניח שהטענה נכונה עבור k ונוכיח עבור k+1.
לפי הנחת האינדוקציה, כל המסלולים בין i ל j חושבו נכון (הם הכי קלים) כך שהם לא עוברים באמצע דרך קודקודים עם אינדקס גבוה מ k.
בשלב ה k+1. המסלול הקל ביותר בין i ל j יכול לא לעבור דרך k+1 ואז הוא נשאר כמו מקודם או שדרך k+1 נקבל מסלול קל יותר ואז זה שווה להגיע מ i ל k+1 (ללא מעבר דרך קודקודים שהאינדקס שלהם גבוה מ k) ואז מ k+1 ל j (ללא מעבר דרך קודקודים שהאינדקס שלהם גבוה מ k) ואלו חושבו כבר לפי הנחת האינדוקציה.

שימושים:
- בעיית הבקבוקים:

נתונים לנו 2 בקבוקים: הראשון מכיל a ליטר והשני b ליטר.
פעולות מותרות על הבקבוקים:
1. מילוי אחד הבקבוקים עד הסוף.
2. ריקון של אחד הבקבוקים לגמרי.
3. העברה מבקבוק אחד לבקבוק שני, עד שהשני מתמלא או שהראשון נגמר (כלומר, אסור לשפוך יותר ממה שהבקבוק השני יכול להכיל)

הבעיה: האם ניתן להגיע למצב בו בבקבוק הראשון יש בדיוק x ליטר של מים ובבקבוק השני בדיוק y ליטר של מים. $(x \leq a,\ y \leq b)$.

ניתן להמיר את הבעיה לגרף שבו כל קודקוד מתאר מצב (x, y) של 2 הבקבוקים.
כלומר: יהיו בגרף: $(a + 1) \cdot (b + 1)$ קודקודים.
תהיה צלע מכוונת בין 2 קודקודים אם ניתן להגיע מהראשון לשני ע"י אחת הפעולות המותרות.

המטרה למצוא מסלול בין המצב ההתחלתי $(0, 0)$ למצב המבוקש (x, y).
אם יש מסלול - אז ניתן לבצע את הפעולות שמביאות אותנו למצב הדרוש וגם נוכל למצוא את המסלול הקצר ביותר שעושה זאת.

שאלה יצירתית 1
כתבו אלגוריתם למציאת מסלול קל ביותר בין כל 2 קודקודים כאשר המשקלים הם רק על הקודקודים.

פתרון:
קלט: גרף G עם מטריצת שכנויות של 0 או 1 (על הצלעות אין משקלים) ומערך W של משקלי הקודקודים.
ממירים למשקלים על הצלעות:
לכל צלע בין i ל j נכניס למיקום (i, j) במטריצה את הערך: $w[i] + w[j]$.

לאחר מכן, נריץ את אלגוריתם פלויד וארשל על הגרף שהתקבל.

כעת נקבל משקלים שגויים: כל קודקוד שהיה באמצע נספר פעמיים. כלומר כל קודקודי המסלול נספרו פעמיים חוץ מקודקודי הקצה.
לכן נעבור על מטריצת התוצאה ונתקן את תא (i, j) באופן הבא: $G[i][j] = (D[i][j] + w[i] + w[j])/2$.

שאלה יצירתית 2
כתבו אלגוריתם למציאת מסלול קל ביותר בין כל 2 קודקודים כאשר המשקלים הם גם על הקודקודים וגם על הצלעות.

פתרון:
קלט: גרף G עם מטריצת שכנויות של משקלים ומערך W של משקלי הקודקודים.
ממירים הכל למשקלים על הצלעות:
לכל צלע בין i ל j נכניס למיקום (i, j) במטריצה את הערך: $D[i][j] = 2 \cdot G[i][j] + w[i] + w[j]$.
לאחר מכן, נריץ את אלגוריתם פלויד וארשל על הגרף שהתקבל.

כעת נקבל משקלים שגויים: כל קודקוד שהיה באמצע נספר פעמיים וגם כל הצלעות חושבו פעמיים. כלומר הכל נספר פעמיים חוץ מקודקודי הקצה.
לכן נעבור על מטריצת התוצאה ונתקן את תא (i, j) באופן הבא: $G[i][j] = (D[i][j] + w[i] + w[j])/2$.

אלגוריתם BFS
סריקה לרוחב.
מעבר על כל קודקודי הגרף החל מקודקוד מקור (התחלה) s.
שיטת המעבר - קודם עוברים על הקרובים ביותר ל s ולאט לאט מתרחקים.
לכן אלגוריתם זה מוצא מסלול קצר ביותר בין s לכל שאר הגרף.

משתמשים **בתור** כדי שמי שייכנס ראשון (השכנים הקרובים) - יצא ראשון.
קלט: רשימת סמיכויות.

פסאודו קוד:
```
BFS(G,s)
	dist = new Array[|V|] - המרחק מההתחלה
	pred = new Array[|V|] - הקודקוד הקודם במסלול מההתחלה עד אלי
	color = new Array[|V|] - הצבע של כל קודקוד שמסמן האם ביקרנו בו או לא
	for i = 1 to |V|
		color[i] = white
	dist[s] = 0 , pred[s] = null , color[s] = grey
	Queue q = new Queue
	q.enqueue(s)
	while(q is not empty)
		v = q.dequeue()
		for each u in G[v]
			if(color[u] == white)
```

```
                color[u] = grey
                dist[u] = dist[v] + 1
                pred[u] = v
                q.enqueue(u)
        color[v] = black
```

לאחר סיום האלגוריתם, המערך dist יכיל בתא i את המרחק בצלעות הקצר ביותר בין s ל i.
המערך pred יכיל בתא i את הקודקוד הקודם במסלול הקצר ביותר בין s ל i.
המערך color יהיה שחור לכל מי שיש מסלול מ s אליו.

סיבוכיות: עוברים על כל קודקוד לכל היותר פעם אחת ועל כל צלע לכל היותר פעמיים (אם הגרף לא מכוון) ולכן: $O(|E| + |V|)$.

שאלות יצירתית 1:
כתבו אלגוריתם המקבל גרף קשיר ומחזיר האם הגרף הוא דו צדדי.

פתרון:
נבצע BFS כאשר את הראשון נצבע בצבע אחד ואז את השכנים שלו נצבע בצבע השני לסירוגין. אם הצלחנו לסיים את הצביעה, נחזיר שהגרף אכן דו צדדי. אם במהלך הסריקה נגלה קודקוד צבוע בצבע הזהה לשכן שלו, נחזיר שהגרף הוא לא דו צדדי.

קוד:

```
isBipartite(G)
    color = new Array[|V|] - הצבע של כל קודקוד שמסמן האם ביקרנו בו או לא
    for i = 1 to |V|
        color[i] = white
    color[1] = red
    Queue q = new Queue
    q.enqueue(1)
    while(q is not empty)
        v = q.dequeue()
        for each u in G[v]
            if(color[u] == white)
                if(color[v] == red) color[u] = blue
                else color[u] = red
                q.enqueue(u)
            else if(color[u] == color[v]) return false
    return true
```

שאלה יצירתית 2:

כתבו אלגוריתם המקבל גרף קשיר ומחזיר האם יש מעגל בגרף.

פתרון:
נריץ BFS כרגיל ואם במהלך הסריקה נגלה שכן שהוא לא לבן אבל גם לא ה pred שלי - זה אומר שיש מעגל ונחזיר true. אם נסיים את כל הלולאה - נחזיר false.

קוד:

```
isCycle(G)
	pred = new Array[|V|] - הקודקוד הקודם במסלול מההתחלה עד אלי
	color = new Array[|V|] - הצבע של כל קודקוד שמסמן האם ביקרנו בו או לא
	for i = 1 to |V|
		color[i] = white
	pred[1] = null , color[1] = grey
	Queue q = new Queue
	q.enqueue(1)
	while(q is not empty)
		v = q.dequeue()
		for each u in G[v]
			if(color[u] == white)
				color[u] = grey
				pred[u] = v
				q.enqueue(u)
			else if(pred[v] != u) return true
	return false
```

שימוש נפוץ: מציאת רכיבי קשירות:
מעבר על כל הקודקודים ובכל פעם שמוצאים מישהו לבן, מריצים שוב BFS החל ממנו ומלקטים את כל הקודקודים החדשים לרכיב נוסף.

אלגוריתם דייקסטרה:
המטרה: מציאת **מסלול קל ביותר** בין קודקוד אחד לכל השאר. (גרף עם משקלים)
הסריקה תהיה "דומה" ל BFS רק שמקום תור, נשתמש בתור עדיפויות (ערימה)

פסאודו קוד:

```
Dijkstra(G,s)
	visited = new Array[|V|]
	dist = new Array[|V|]
	pred = new Array[|V|]
	for i = 1 to |V|
		visited[i] = false
		pred[i] = null
		dist[i] = inf
```

```
dist[s] = 0 , visited[s] = true
PriorityQueue q = new PriorityQueue // compare by dist[i]
q.enqueue(s)
while(q is not empty)
        v = q.extractMin()
        for each u in G[v]
                if(!visited[u])
                        if(dist[u] > dist[v] + G[v][u].w) pred[u] = v
                        dist[u] = min(dist[u], dist[v] + G[v][u].w)
                        if(q.contains(u)) q.decreasekey(dist[u])
                        else q.enqueue(u)
        visited[v] = true
```

לאחר האלגוריתם dist מכיל בתא i את העלות הקלה ביותר בין s לקודקוד i
המערך pred מכיל את הקודקוד הקודם ל i במסלול הטוב ביותר בין s ל i
המערך visited הוא true עבור כל מי שניתן להגיע אליו מ s

סיבוכיות: $O(|V| + |E| \cdot log|V|)$.
כי עוברים על כל קודקוד וצלע לכל היותר פעם אחת ועבור כל שכן מעדכנים את התור (במקרה הגרוע) ועדכון זה עולה: $log|V|$.

האלגוריתם לא עובד טוב אם יש משקלים שליליים.

אלגוריתם שריפת עלים
נתון לנו עץ (גרף קשיר ללא מעגלים) ואנו רוצים למצוא את המרכז שלו.
מרכז = "הקודקוד שקרוב לכולם"

הרעיון, לקחת את העלים ולמחוק אותם מהעץ, נקבל עץ עם עלים חדשים, באופן איטרטיבי נמחק גם אותם וכך הלאה עד שנישאר עם קודקוד אחד או 2 קודקודים ואלו יהיו המרכזים.

פסאודו קוד:

```
FindCenters(G)
        L = new List
        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
```

סיבוכיות: בכל עץ יש לפחות 2 עלים ולכן בכל איטרציה יורדים לפחות 2 קודקודים.
כל קודקוד נכנס לרשימת העלים פעם אחת בדיוק ולכן: $O(|V|) = O(|E|)$ כי בעץ $|E| = |V| - 1$.

אלגוריתם Best:
מטרה: מציאת קטע עם סכום מקסימאלי במערך.
נתון לנו מערך שיש בו גם מספרים שליליים ואנו רוצים למצוא רצף במערך עם הסכום הגדול ביותר.
לדוגמא: $[5, -\ 6, 2, -\ 1, 7, 3, -\ 2, 4, 4, -\ 20, 5, 8, 3]$

האלגוריתם:
עוברים פעם אחת על המערך וסוכמים:
בכל שלב שומרים את המקסימום עד כה.
אם הסכום יורד מתחת ל 0, מאפסים אותו ומתחילים את הרצף מחדש.

פסאודו-קוד:

```
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)
```

סיבוכיות: $O(n)$ כאשר n הוא אורך המערך.
אלגוריתם Best מעגלי:
הרעיון להשתמש ב Best הרגיל.
לקחת את סכום כל המערך ולהוריד את הרצף הכי מינימאלי שאינו מעגלי ובכך לקבל את הרצף המקסימאלי המעגלי.

או שהרצף הוא לא מעגלי וניתן למצוא אותו כמו מקודם.
לדוגמא: $[5,-6,2,-1,7,3,-2,4,4,-20,5,8,3]$
ניקח את ה best הרגיל: $Best=(17,3,9)$.
וניקח את $B=[-5,6,-2,1,-7,-3,2,-4,-4,20,-5,-8,-3]$
מכאן: $sum(A)=12$, $Best(B)=20$
קוד:

```
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,s1-1)
```