/"_________________.aux

סיכום דאסט

7 באפריל 2021

1 סיבוכיות של פונקציות:

- נאמר ש $f(n) = O(g(n))$ אם קיים c כך ש $f(n) \leq c \cdot g(n)$
- נאמר ש $f(n) = \Omega(g(n))$ אם קיים c כך ש $f(n) \geq c \cdot g(n)$
- נאמר ש $f(n) = \Theta(g(n))$ אם קיימים c_1 ו c_2 כך ש $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n)$

1.1 טענות:

- $O(O(f(n))) = O(f(n))$.
- $O(f(n) + g(n)) = O(f(n)) + O(g(n))$
- $O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$
- $O(log(n)) = O(log_2(n))$.
- $O\left(\sum_{i=1}^{k} a_i n^i\right) = O(n^k)$
- $n! = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$
- $n! = O(n^n)$
- $log(n!) = O(n \cdot log(n))$
- $lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow g(n) = \Omega(F(n))$
- אם $f(n) = O(g(n))$ וגם $f(n) \to \infty$ אז מתקיים $log\left(f(n)\right) = O\left(log\left(g(n)\right)\right)$
- $n^2 = O(2^n)$

2 חוקי לוגוריתמים וטורים:

2.1 לוגים:

- $log_a(n) = \frac{log_b(n)}{log_b(a)}$
- $log(a \cdot b) = log(a) + log(b)$
- $log\left(\frac{a}{b}\right) = log(a) - log(b)$
- עבור x מספר כלשהו מתקיים $x^{log_x(n)} = n$
- $log_x(x) = 1$ עבור x מספר כלשהו.

2.2 טורים:

- $\sum_{\mathrm{k}=0}^{\infty} \frac{1}{2k} = 1$
- $\sum_{\mathrm{k}=0}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2}$ עבור $|x| < 1$

3 סיבוכיות זמן ריצה - נוסחאות רקורסיביות

3.1 משפט האב:

לפעמים כדי להשתמש במשפט האב המורחב נצטרך לעשות מספר מניפולציות על הנוסחה הרקורסיבית. לדוגמה:

$$T(n) = 3T\left(n^{\frac{1}{3}}\right) + log(log\,(n)) \Rightarrow T(n) = T(2^m) = 3T\left(2^{\frac{m}{3}}\right) + log(m) \Rightarrow S(m) = 3S\left(\frac{m}{3}\right) + log\,(m)$$

בהתחלה הגדרנו $m = log(n)$
אחכ הגדרנו $s = 2^m$

3.2 עצי רקורסיה:

בעצי רקורסיה נצטרך לברר שלשה דברים:

1: מהו גובה העץ - $log_b n$.

2: מספר הקודקודים שיש בכל שלב בעץ a^k.

3: מה כמות העבודה על כל קודקוד $\left(\frac{n}{b^k}\right)^c$.

4: מהי כמות העבודה בשלב ה k : $\left(\frac{n}{b^k}\right)^c \cdot a^k$

5: לבסוף נצטרך לכפול את העבודה בכל קודקוד במספר הקודקודים שיש בעץ. $\sum_{k=0}^{log_b n}\left(\left(\frac{n}{b^k}\right)^c \cdot a^k\right)$

4 אלגוריתמי מיון :

- **הערה חשובה:** לא קיים אלגוריתם מיון באמצעות השוואה (שלא מניח הנחות מוקדמות על הקלט) שממיין בזמן שקטן מ $O(n \cdot log(n))$ - הוכחנו באמצעות עץ החלטה על גובה העץ.
- **הגדרה - מיון יציב:** מיון יציב הוא מיון שמכניס איברים למערך החדש באותו הסדר שבו הם הופיעו במערך המקורי.

4.1 מיון בועות $bubbleSort$:

מימוש: ממיין בעזרת זה שמפעפעים את המספר הגדול ימינה - כל פעם ש המספר מימין גדול מ x נחליף ביניהם עד ש x יגיע למקום שלו. באיטרציה הבאה האלגוריתם יעבור על $n-1$ איברים משום שכעת האיבר הגדול במערך נמצא בצד ימין.
סיבוכיות: $O(n^2)$

4.2 מיון מיזוג $mergeSort$:

מימוש: מיון שעובד בשיטה רקורסיבית - כל פעם מפצל את המערך לתתי מערכים עד שמגיע לגודל 1. ואחכ מתחיל לחבר אותם לפי הסדר.
סיבוכיות: $O(n \cdot log(n))$

4.3 מיון הכנסה $insertionSort$:

מימוש: נבחר את האיבר השני במערך. ונתייחס אל האיבר הראשון כאל מערך ממויין וכל פעם נצרף אליו את שאר האיברים בסדר ממויין.
סיבוכיות: $O(n^2)$

4.4 מיון מהיר $quickSort$:

מימוש: בעזרת אלגוריתם $partition$ נבחר את האיבר שימני במערך להיות איבר הציר $(pivot)$. כך שכל האיברים שגדולים ממנו ימוקמו בצד ימין של המערך. וכל האיברים שקטנים ממנו ימוקמו בצד שמאל של המערך. לבסוף נחליף את איבר הציר עם האיבר הראשון במערך האיברים הגדולים. כך נקבל שאיבר הציר נמצא באמצע וכל מי שמימינו גדול ממנו ומי שמשמאלו קטן ממנו.
נפעיל את האלגוריתם רקורסיבית על כל האיברים ולבסוף נקבל מערך ממויין.
נקודה חשובה: האלגוריתם עושה שימוש באלגוריתם $partition$ שבוחר את איבר הציר וממין את האיברים לפי איבר ציר
סיבוכיות: **במקרה הגרוע** - $O(n^2)$. **במקרה הממוצע** - $O(n \cdot log(n))$ כעומר העץ שנקבע לפי בחירת איבר הציר.
הערה: קיים אלגוריתם שבוחר את איבר הציר רנדומלית וזמן הריצה שלו הוא $O(n \cdot log(n))$

4.5 מיון מניה $countingSort$:

הנחה: האלגוריתם מניח שכל המספרים במערך נמצאים בטווח בין 1 ל k.
מימוש:

1: האלגוריתם יוצר מערך של אפסים בגודל k וכותב עבור כל אינדקס i כמה פעמים האיבר ה i נמצא במערך המקורי.

2: האלגוריתם יוצר מערך חדש של אפסים בגודל k ומעדכן בכל משבצת כמה פעמים יש במערך איברים שקטנים או שווים בגודלם לאיבר ה i.

3: לאחר מכן האלגוריתם עובר על המערך שיצר בסדר ההפוך. ומתחיל להכניס את מספר האיברים כפי שמופיע במערך הראשון. ואחכ הוא מוריד מהמערך השני את מספר האיברים שקטנים שווים מהאיבר ה i עבור כל מספר שמעודכן במערך הממויין.

סיבוכיות: האלגוריתם מניח שכל האיברים במערך נמצאים בטווח מסויים לשכן סיבוכיות זמן הריצה שווה ל $\Theta(n)$.

תוספות: מיון מניה הוא מיון יציב .

4.6 מיון בסיס $radixSort$:

הנחה: ההנחה על הקלט היא - שכל איבר במערך מכיל לכל היותר d ספרות.

מימוש: האלגוריתם עובר על כל המיספרים וממיין אותם פי ספרת האחדות. לאחר מכן לפי ספרת העשרות והמאות וכו...

סיבוכיות: סיבוכיות זמן הריצה היא $\Theta(n)$

4.7 מיון דליים $bucketSort$:

הנחה: ההנחה על הקלט היא - שכל איבר במערך הוא מספר בין 0 ל 1 ושהאיברים מתפלגים אחיד על הקטע.

מימוש:

1: האלגוריתם מחלק את הקטע ל n דליים.

2: לאחר מכן הוא עובר על כל האיברים וממיין אותם לדלי המתאים לפי הספרה הראשונה ומכניס אותם לתוך הרשימה המתאימה באופן ממויין (כמו מיון הכנסה).

3: אחכ הוא משרשר את כל הדליים יחד.

סיבוכיות: סיבוכיות זמן הריצה **הממוצע** היא $\Theta(n)$

4.8 מיון ערימה :

מימוש: נבנה ערימה בזמן $O(n)$ נוציא כל פעם את האיבר המקסימלי מהערימה על ידי $extractMax$ ונעדכן את הערימה בזמן של $O(log(n))$ סהכ יש n קודקודים לכן

זמן הריצה: סהכ יש n קודקודים לכן זמן הריצה שווה ל $O(n \cdot log(n))$

5 טבלאות גיבוב:

אנו רוצים לסדר נתונים בטבלה שתאפשר לנו: חיפוש, הכנסה ומחיקה בזמן קבוע של $O(1)$. אנו רוצים גם לשמור על גודל טבלה פורפורציונלי למספר המפתחות.

סימונים:

U - קבוצת כל המפתחות הקיימים בעולם

k - קבוצת המפתחות אותם אנו צריכים לגבב. $|k| = n$

T - הטבלה. $|T| = m$

פונקציית גיבוב $h(k) : U \Rightarrow [0, m-1]$

מקדם עומס $factor - load$: יוגדר להיות $\alpha = \frac{n}{m}$ ומתקיים כי $0 \leq \alpha \leq 1$ כאשר m הם מספר המקומות בטבלה ו n זה מספר המפתחות הנוכחי שמשובצים בטבלה, לכן כאשר n גדל נצטרך להגדיל את הטבלה.

5.1 מיעון ישיר $direct\ address$:

אנו מגבבים את הערכים כל אחד לאינדקס אחר. לא מטפלים בהתנגשויות מפני שגודל הטבלה שווה מספר המפתחות.

פונקציית הגיבוב תיראה כך: $h(k) = k$

5.2 מיעון פתוח $open - addresing$:

אנו יוצרים פונקצית האש כך שאם תיווצר התנגשות אנו נעבור לאינדקס הבא שיהיה פנוי.

פונקציית הגיבוב תהיה מהצורה הבאה: $h(k,\text{i}) : U \times [0, m-1] \Rightarrow [0, m-2]$ כאשר הערך i מסמן את מספר הפעם שבה ניסינו לשבץ את הערך בטבלה. אחרי כל נסיון גיבוב כושל נעלה את i באחד.

הכנסה: נפעיל את הפונקציה עד שנמצא מקום פנוי.

חיפוש: נחפש עד שנמצא את הערך או עד שנגיע לתא שמסומן כמחוק.

מחיקה: נמחק את האיבר ונסמן את התא כמחוק.

טענה: בגיבוב פתוח - אם מתקיים שמקדם העומס קטן מ 1 כלומר - $\alpha < 1$ אזי מספר הבדיקות המקסימלי הוא:

בחיפוש מוצלח: $\frac{1}{1-\alpha}$

בחיפוש לא מוצלח: $\frac{1}{\alpha} \cdot ln\left(\frac{1}{1-\alpha}\right)$

וכאשר α קבוע: זמן החיפוש הינו $O(1)$

יש כמה שיטות כיצד לבחור את פונקציית הגיבוב שלנו:

5.2.1 גיבוב לינארי $linear - probing$:

אנו בוחרים פונקצית גיבוב ומקדמים כל פעם את i באחד. $h(k) = (h'(k) + i)\, mod(m)$

5.2.2 גיבוב ריבועי:

אנו בוחרים את הפונקציה כך : $h(k) = (h'(k) + c_1 i + c_2 i^2)\, mod(m)$

כאשר c_1 וc_2מייצגים מספרים קבועים, כך למעשה אנו בודקים כל פעם מקומות רחוקים יותר בטבלה.

5.2.3 גיבוב כפול $double-hashing$:

אנו מגבבים לפי הפונקציה, וכאשר אנו מגיעים להתנגשות אנו מפעילים פונקציית גיבוב נוספת.
פונקצית הגיבוב תיראה כך : $h(k,i)=(h_1(k)+i\cdot h_2(k))mod\,(m)$
כאשר $h_{1,}h_2$ הן פונקציות גיבוב. בנוסף $h_2(k)$ צריכה להחזיר **מספר ראשוני** אחרת זה לא יעבוד
וגם אנו צריכים שגודל הטבלה (m) תהיה חזקה של 2 .

5.3 מיעון סגור $close-addresing$:

טענה: בגיבוב סגור - אם מתקיים שמקדם העומס קטן מ 1 כלומר - $\alpha<1$ אזי מספר הבדיקות המקסימלי הוא:
בחיפוש מוצלח ולא מוצלח $\Theta(1+\alpha)$
וכאשר $\alpha=O(1)$: זמן החיפוש המוצע הינו $\Theta(1)$
כאשר$n=O(m)$ ו $\alpha=O\left(\frac{m}{n}\right)=1$: זמן החיפוש המוצע הינו $\Theta(1)$
אם נוצרת התנגשות אנו יוצרים רשימה מקושרת ומשרשרים את המפתחות יחד.

5.4 כיצד נבנה טבלאות גיבוב:

קיימות מספר שיטות כדי להרכיב פונקציית גיבוב טובה:

5.4.1 שיטת החילוק:

נבחר פונקצית גיבוב כך שתמיד נעשה לה מודולו לגודל הטבלה $h(k)=(k)\,mod\,(m)$
כיצד נבחר את m: נחלק את מבפר המפתחות ביוניברס במספר ההתנגדויות שאנו רוצים שיקרו, ונבחר את המספר הראשוני הקרוב לתוצאה. לדוגמה: $U=2000$ ואנו רוצים 3 התנגשויות אז $m=\frac{2000}{3}=701$
אנו נבחר m שאינו חזקה של 2 והוא מספר ראשוני.

5.4.2 שיטת החילוק:

1: נבחר $0<A<1$ ונכפול בה את המפתח k
2: נקח את החלק השברי של המכפלה . כלומר $b=Ak-\lfloor Ak\rfloor$ כאשר b הוא החלק השברי.
3: נקח את $\lfloor bm\rfloor$
ונקבל: $h(k)=\lfloor m(kA-\lfloor kA\rfloor)\rfloor$

5.5 גיבוב אוניברסלי:

הגדרה: נאמר שמשפחת פונקציות האש H היא אוניברסלית : אם הסיכוי להתנגשות בין שני ערכים קטן שווה מ $\frac{1}{m}$
משפט: אם H היא משפחה אוניברסלית אזי בגיבוב סגור (רשימות מקושרות):

- תוחלת אורכי הרישימות עבור מפתח k שלא נמצא בטבלה שווה ל α
- תוחלת אורכי הרשימות עבור מפתח שנמצא בטבלה שווה ל $1+\alpha$

משפט: אם H היא משפחה אוניברסלית אזי בגיבוב פתוח:

- תוחלת אורכי הרישימות עבור מפתח k שלא נמצא בטבלה שווה ל $\frac{1}{1-\alpha}$
- תוחלת אורכי הרשימות עבור מפתח שנמצא בטבלה שווה ל $\frac{1}{\alpha}\cdot ln\left(\frac{1}{1-\alpha}\right)$

5.5.1 כיצד נבנה משפחה אוניברסלית:

1: נבחר מספר ראשוני p שגדול מ m. ונגדיר את הקבוצה $Z_p=\{0,1,2,3,5.....p-1\}$ קבוצת כל המספרים הראשוניים עד p

2: נבחר שני מספרים ראשוניים $a,b\in Z_p$ ונגדיר לכל אחד מהם פונקצית גיבוב על ידי העתקה לינארית ושימוש במודולו. כך:

$$h_{a,b}(k)=((ak+b)mod\,(p))mod\,(m)$$

3: נקבל כי Hשווה ל:

$$H_{p,m}=h_{a,b}|a,b{\in}Z_p\ and\ a\neq 0$$

טענה: יהיו k,l שני מפתחות שונים. לכל שני מספרים $a,b\in Z_p$ הסיכוי ש k יגובב ל a ו l יגובב ל b יהיה $\frac{1}{p^2}$לכן המשפחה $H_{p,m}$אוניברלסית.

5.6 גיבוב מושלם:

הגדרה: כאשר קבוצת המפתחות **מוכרת לנו והיא סטטית ולא משתנה,** נוכל למצוא גיבוב מושלם כך שנאפשר חיפוש ב $O(1)$.
כיצד נממש: יש שתי אופציות מבחינת גודל הטבלה שנבחר:

1. **גיבוב מושלם במקום ריבועי:** נגריל פונקציות מהמשפחה H עד שנמצא פונקציה h שלא יוצרת התנגשויות כלל. כל הגרלה היא $O(1)$ לכן הזמן הוא $O(n)$
   נבחר את גודל הטבלה להיות $4n^2\leq m\leq 8n^2$ חכן המיקום הוא ריבועי.

2. **גיבוב מושלם במקום לינארי:** נבצע גיבוב באמצעות שרשור, וכאשר ניתקל בהתנגשות נגבב שוב בעזרת פונקצית גיבוב אחרת שתגיד לנו לאן ברשימה להכניס את הערך. את הגיבוב השני נבצע בעזרת גיבוב מושלם במקום ריבועי. כלומר - נגריל פונקצייה עד שנמצא פונקציה שלא יוצרת התנגשויות
   כלומר לכל תא בטבלה תהיה פונקצית גיבוב משנית שיחודית לתא הזה, שתגיד לנו לאן ברשימה לשלוח את הערך.

$$h(k)=((ak+b)mod\ p)mod\ m$$

$$h_i(k) = ((a_i k + b_i) mod\ p) mod\,(m_i)$$

משפט: אם נאחסן n מפתחות בטבלה בגודל $m = n^2$ בעזרת פונקציית גיבוב ששייכת למשפחה אוניברסלית . אזי הסיכוי להתנגשות אחת קטנה מ $\frac{1}{2}$.

הערה: הגיבובים נעשים בסיבוכיות **מקום** לינארי וריבועי אך בשניהם סיבוכיות הזמן היא לינארית.

6 ערימות:

6.1 ערימת מקסימום:

מוטיבציה: אנו רוצים ליצור מבנה נתונים שתומך בהוצאת מקסימום בזמן של $O(1)$. וחיפוש, מחיקה והגדלת ערך בזמן של $O(log(n))$. סיור הערימה מחדש יקח לנו $O(n)$ זמן.

כיצד נממש: באמצעות עץ בינארי כמעט מלא (השורה האחרונה יכולה להיות מלאה חלקית).

תכונות הערימה:

1: האיבר המקסימלי יהיה בשורש.

2: בכל תת עץ שורש תת העץ הוא המקסימלי בתת העץ.

3: נגדיר את הבנים כך: $left = 2i$, $right = 2i + 1, parent = \lfloor \frac{i}{2} \rfloor$

4: גובה העץ יהיה $\lfloor log(n) \rfloor$ (שורש העץ לא נספר בגובה)

5: בכל רמה יהיו 2^hקודקודים

6: מספר הקודקודים בעץ חסום על ידי $2^h \leq n \leq 2^{h+1} - 1$

שמירה על תכונות הערימה:

1: $maxHeapify(A, i)$ **:**

אלגוריתם שמסדר מחדש את הערימה והוא הבסיס לכל האלגוריתמים של $heap$ נתן לאלגוריתם אינדקס של קודקוד ונבדוק אותו מול שני הבנים שלו. אם אחד מהם גדול מהאב נחליף את האב עם הבן המקסימלי.

לכל היותר לאחר $log(n)$ איטרציות העץ יהיה מסודר כנדרש.

בכדי לסדר את העץ מחדש נשמש באלגוריתם $maxHeapify(A, i)$ כאשר המערך הוא A והאינדקס ממנו נתחיל לסדר הוא i.

סיבוכיות זמן ריצה של $maxHeapify(A, i)$**:** $O(log(n))$

2: **בניית ערימה** $buildMaxHeap$**:**

נבנה מהמערך עץ מאוזן ולאחר מכן נריץ את $maxHeapify(A, i)$ על כל אחד מהקודקודים מלבד העלים. כלומר נריץ את האלגוריתם על $[1.....\frac{n}{2}]$ מהשורש לכיוון העלים. (מכיוון שהחצי השני של המערך הינו עלים ואין צורך לסדר אותם).

סיבוכיות זמן : $O(n)$

3: **הוצאת המקסימום** $extractMax$**:**

נוציא את האיבר בשורש ונכניס במקומו עלה. לאחר מכן נפעיל את האלגוריתם $maxHeapify(A, 1)$ כדי לסדר את הערימה מחדש.

4: **העלאת ערך** $increaseKey(A, i, key)$**:**
אם נגדיל את המפתח תכונת הערימה עלולה להיהרס . לכן:
אם הערך החדש קטן יותר משל אביו - לא נשנה כלום.
אם הוא גדול משל אביו נשינה את הערך ונפעיל את $maxheapify$
5: **הוספת איבר חדש לערימה** $heapInsert(a, key)$**:**
נגדיר את הערך של המפתח להיות $-\infty$ ונכניס אותו לערימה בתור עלה.
אח"כ נקרא לאלגוריתם $increaseKey(A, i, key)$ עם הערך שאותו נרצה להכניס.
6: **מחיקת איבר:** נחליף את האיבר שאנו רוצים למחוק עם עלה. אחכ נמחק את העלה החדש ונריץ $maxHeapify$ מהעלה המוחלף.

6.2 ערימת מינימום:

ערימת מינימום: תומכת באותן הפעולות של ערימת מקסימום מלבד העובדה שהמינימום נמצא בשורש.

6.3 ערימת חציון:

ערימת חציון שומרת על החציון בשורש העץ ונממש אותה בעזרת שתי ערימות :
A ערימת מקסימום.
B וערימת מינימום.
ומתקיים שאיברי A קטנים שווים לאברי B.
מתקיים בנוסף $0 \leq |A| - |B| \leq 1$ כלומר A גדולה ב 1 מ B או שווה לה.

אלגוריתמים:
1: $getMedian$ **קבלת החציון:**
נחזיר את השורש של ערימת המינימום.
2: **הכנסת איבר** $insert(v)$**:**
מימוש:

1. אם אורכי הרשימות שווים:
   נבדוק אם v גדול שווה מאיברי A: אם כן - נכניס אותו לערימה B.
   אם לא - נעביר את השורש של A לערימה B. ואחכ נכניס את v לערימה A .

2. אם הרשימות שונות בגודלן:
   נבדוק אם v קטן מאיברי B :אם כן - נכניס אותו לערימה B.
   אם לא - נוציא את השורש של B לערימה A ואחכ נכניס את v לערימה B.

באופן כללי אם אורכי הרשימות שוות ואנו רוצים והוסיף איבר מבלי לפגוע בתכונה ש $A \geq B$ אז נצטרך להוציא את השורש מהערימה אליה אנו רוצים להכניס. ולהעביר אוו לערימה השניה.

7 טבלאות יאנג $YoungTableaux$:

טבלת יאנג היא מטריצה מגודל $n\times m$ כך שהשורות והעמודות מסודרות בסדר עולה. בתאים ריקים נכניס ∞.

הבחנות:

1. טבלת יאנג היא לא יחידה וניתן לבנות מרצף מספרים כמה טבלאות שונות.
2. אם $Y[1,1]=\infty$: הטבלה ריקה.
3. אם $Y[n,m]\neq\infty$ הטבלה מלאה.
4. **עתנאי שקול לטבלת יאנג :** עבור כל תא בטבלה: התאים משמאל למעלה - קטנים יותר. התאים מימין למטה - גדולים יותר.

אלגוריתמים:

1. $bubbleDown$: האלגוריתם מקבל את האיבר $Y[i,j]$ ובודק אותו מול האיבר שמימינו והאיבר שמתחתיו. ומחליף את האיבר הנוכחי בקטן מבן שלשתם. כך הוא מקדם את המינימלי כלפי מטה
   סיבוכיות זמן: $O(n+m)$
2. $Extractmin$: אלגוריתם להוצאת המינילי. נוציא את האיבר במקון $[1,1]$ מהטבלה ונשים במקומו ∞. אחכ נפעיל את האלגוריתם $bubbleDown$ כדי לסדר את הטבלה.
   סיבוכיות זמן: $O(n+m)$
3. $insert$: אלגוריתן להכנסת איבר חדש לטבלה: נכניס את האיבר למיקום $[n,m]$ ונפעיל עליו את האלגוריתם $bubblrUp$ שמעלה את המקסימלי מבין השמאלי והעליון ללמעלה.
   סיבוכיות זמן: $O(n+m)$
4. **אלגוריתם מיון:** נמיין מערך בגודל n^2 בזמן של n^3ע"י $extraxtMin$.

8 עצי חיפוש בינארים ועצים מאוזנים:

8.1 עצים בינארים:

מוטיבציה: אנו רוצים מבנה נתונים שתומך בחיפוש, מציאת מינ' ומקס', מציאת האיבר הבא ומציאת האיבר הקודם בזמן של $O(log(n))$ - גובה העץ.

טענה: גובה ממוצע של עץ בינארי רנדומלי (שהמפתחות הכנסו אליו באופן רנדומלי) הוא $O(log(n))$

תכונות העץ: לכל קודקוד יהיו 4 שדות: ערך. הורה. בן ימני. בן שמאלי.

בנוסף אנו נדרוש שהילד הימני יהיה גדול שווה מהשורש, והילד השמאלי קטן מהשורש.

1: **חיפוש בעץ** $treeSearch(x,k)$:

כאשר x מייצג את שורש העץ. ו k את המפתח שצריך לחפש.

נבדוק כל פעם אם המפתח קטן מהשורש: אם כן ־ נקרא לאלגוריתם שוב עם הילד השמאלי של העץ ואחרת נקרא לאלגורית ען=ם הילד הימני של העץ.

סיבוכיות זמן ריצה: $O(log(n))$ כגובה העץ.

2: **מציאת מקסימום** $treeMax(x)$:

נקח את השורש ונבדוק אם יש לו ילד ימני אם כן ־ נמשיך עד הסוף ובסוף נחזיר את הקודקוד אליו הגענו.

סיבוכיות זמן ריצה: $O(log(n))$ כגובה העץ.

3: **מציאת מינימום** $treeMin(x)$:

נקח את השורש ונבדוק אם יש לו ילד שמאלי אם כן ־ נמשיך עד הסוף ובסוף נחזיר את הקודקוד אליו הגענו.

סיבוכיות זמן ריצה: $O(log(n))$ כגובה העץ.

4: **מעבר על איברי העץ בסדר עולה** $inOrderTreeWalk(x)$:

כל עוד יש לעץ ילד שמאלי תגיע אליו ולבסוף תדפיס אותו. לאחר מכן תדפיס את האבא שלו ותלך לצד הימני של תת העץ.

כיצד האלגוריתם עובד: כל עוד השורש לא שווה ל $null$ תקרא לאלגוריתם בצורה רקורסיבית עם הליד השמאלי, תדפיס, ואחכ תקרא לאלגוריתם עם הילד הימני.

אחרת ־ אם השורש שווה ל $null$ תעשה $return$.

סיבוכיות זמן ריצה : $O(n)$.

5: **מציאת האיבר העוקב** $succesor$:

יש שלשה מקרים של החזרת האיבר העוקב:

- אם x הוא המקסימום ־ נחזיר $null$.
- אם ל x אין ילד ימני ־ אם x הוא ילד שמאלי של ההורה שלו : נחזיר את ההורה של x. אם x הוא הילד הימני של ההורה שלו : נעלה למעלה דרך כל הילדים הימניים, ולבסוף נדפיס את האב של הקודקוד האחרון שהגענו אליו.
- אם ל x יש ילד ימני ־ נפנה ימינה בעץ פעם אחת ונמצא אתת המינימום דרך צד שמאל. כלומר נפעיל את האלגוריתם למציאת מינימום על הילד הימני של x.

סיבוכיות זמן ריצה: $O(log(n))$ כגובה העץ.

6: **הכנסת ערך חדש לעץ** $treeInsert(T, z)$:

נבצע חיפוש בעץ עד שנגיע למקום מתאים לערך שנרצה להוסיף ונוסיף אותו ־ תמיד נכניס אותו כעלה ונגדיר את הילדים שלו להיות $null$.

סיבוכיות זמן ריצה: $O(log(n))$ כגובה העץ.

7: **מחיקת ערך** $treeDelete(T, z)$:

מחיקת ערך זוהי פעולה מסובכת יותר מהכנסת ערך. יש שלשה מקרים:

- לקודקוד z אין ילדים ־ נמחק את הערך ונעדכן את האב להיות $null$.
- לקודקוד z יש ילד יחיד ־ נמחק את z ונעדכן את הילד להיות הילד של $z.parent$.
- לקודקוד z יש שני ילדים ־ נחליף את z באיבר העוקב שלו $(succesor)$ ונוציא את העוקב שהוא בעל ילד 1 לכל היותר .

האלגוריתם של מחיקת ערך משתמש באלגוריתם $transplant$ שהוא אלגוריתם להחלפת שני תתי עצים.
טענה: אם ל z יש שני ילדים, לעוקב שלו יש לכל היותר ילד אחד.
סיבוכיות זמן ריצה: במקרה 1,2 - הזמן קבוע $O(1)$ במקרה השלישי - $O(log(n))$ כגובה העץ.
8: **החלפת שני תתי עצים** $transplant(T,u,v)$:
אלגוריתם שמקבל עץ ושני קודקודים ומבצע את ההחלפה ביניהם.

8.2 עצים מאוזנים $AVL-trees$:

הגדרה: עץ מאוזן הינו עץ ששומר על כך שהמרחק בין כל שני תתי עצים מקיים $|1| \leq left-right$.
מקדם איזון: בנוסף לשדות שיש לכל קודקוד בעץ בינארי,כאן אנו נוסיף לעץ שדה ששומר עבור כל קודקוד את גובה העץ החל מהקודקוד הזה ועד לעלה האחרון - $higth-diff$ או בקיצור hd.
מוטיבציה: מכיוון שאנו מבצעים על עצים פעולות שיכולות להוציא אותם מאיזון ולגרום לגובה העץ להיות גדול מ $log(n)$. אנו רוצים עץ שהגובה שלו יהיה תמיד $log(n)$.
איזון עץ מחדש $rotation$:
אם x הוא קודקוד הזקוק לאיזון מחדש, יש 4 מקרים:

- הכנסה לתת עץ שמאלי של הילד השמאלי של x - רוטציה ימינה.
- הכנסה לתת עץ ימני של הילד השמאלי של x - רוטציה שמאלה ואח"כ רוטציה ימינה.
- הכנסה לתת עץ שמאלי של הילד הימני של x - רוטציה ימינה ואח"כ רוטציה שמאלה.
- הכנסה לתת עץ ימני של הילד הימני של x - רוטציה שמאלה.

באופן כללי - נתפוס את הקודקוד שיש בו את ההפרה ו "נמשוך" אותו כלפי מטה ונחבר את הבן הימני שלו בתור הבן השמאלי של הקודקוד (לא יודע איך לכתוב את זה).
סיבוכיות זמן ריצה של רוטציה: $O(1)$.

כיצד נטפל בהפרות:
תמיד נבדוק את ההפרה בקודקוד האחרון שבו ההפרה מתרחשת ועליו נבצע את הרוטציה.

1. RR: גורם האיזון בשורש הוא -2. גורם האיזון בבן הימני הוא -1. **נבצע:** רוטצית L על השורש.
2. LL: גורם האיזון בשורש הוא 2. גורם האיזון בבן הימני הוא 1. **נבצע:** רוטצית R על השורש.
3. LR: גורם האיזון בשורש הוא 2. גורם האיזון בבן הימני הוא -1. **נבצע:** רוטצית L על הבן השמאלי ורוטצית R על השורש
4. RL: גורם האיזון בשורש הוא -2. גורם האיזון בבן הימני הוא 1. **נבצע:** רוטצית R על הבן השמאלי ורוטצית L על השורש

9 גרפים:

הגדרה: גרף הוא זוג כך $G(V,E)$ כאשר V מייצג את הקודקודים ו E את הצלעות.
גודל הגרף: $|G|=|V|+|E|$
ייצוג גרף: יש שתי דרכים לייצג גרף: (אם מספר קשתות מועט נעדיף לייצג על ידי רשימה).
1: **מטריצת שכנויות** - כל שני קודקודים שיש בניהם מסלול נסמן בתא המתאים כ 1 ואם אין מסלול התא יסומן כ 0 .
2: **רשימת שכנויות:** לכל קודקוד $v \in V$ יש רשימה עם כל שכניו.

9.1 גרף לא מכוון:

סכום הדרגות: בגרף לא מכוון סכום הצלעות שווה ל $\sum_{i=1}^{n} d(v_i) = 2\cdot|E|$

9.2 גרף מכוון:

גרף מכוון הוא גרף שיכול להכיל צלע מקודקוד אל עצמו. בנוסף הצלעות מכוונות ואם יש צלע מ v ל u לא בהכרח שיש צלע הפוכה.
סכום הדרגות: בגרף מכוון סכום הצלעות שווה ל $\sum_{i=1}^{n} d_{in}(v_i) = \sum_{i=1}^{n} d_{out}(v_i) = |E|$

9.3 גרף ממושקל:

סגרף ממושקל יש לכל צלע מחיר.
יכולות להיות צלעות בעלי משקל שלילי.
אם בין שני קודקודים אין צלע - המרחק בניהם יוגדר להיות ∞ .

9.4 הגדרות כלליות וטענות על גרפים:

1. **גרף קשיר:** נאמר שגרף G הוא קשיר אם קיים מסלול בין כל שני קודקודים בגרף.
2. **קשיר היטב:** (גרף מכוון קשיר) נאמר שגרף G הוא קשיר היטב אם קיים מסלול מכוון בין כל שני קודקודים בגרף.
3. קשיר למחצה: נאמר שהגרף קשיר למחצה אם לכל זוג קודקודים או שיש מסלול מ $v-u$ או מסלול $u-v$.
4. **תת גרף :** תת קבוצה $G'(E',V')$ של קודקודים וצלעות של הגרף G המחוברים בניהם.
5. **רכיבי קשירות:** תת הקבוצה הקשירה הגדולה ביותר.
6. **מספר הצלעות:** בגרף יש לכל היותר $|E|=O(|V|^2)$. ובגרף קשיר $|E|\geq|V|-1$ (אם מתקיים שוויון אזי G הוא עץ)
7. **קליקה:** גרף יקרא קליקה אם כל קודקוד מחובר לשאר הקודקודים בגרף. כלומר - $E|=|V|^2$
8. **גרף פלנארי:** אף צלע א חוצה צלע אחרת. ומתקיים $|E|=O(|V|)$

9. **למה:** עבור כל צלע $(u,v)\in E$ מתקיים $\delta(s,v)\leq\delta(s,u)+1$

10. **מסלול פשוט:** הוא מסלול שלא עובר באותו קודקוד פעמיים.

9.5 עצים:

הגדרה: עץ הוא גרף G המקיים: 1: קשיר. 2: ללא מעגלים. 3: יש לו $|E|=|V|-1$ אם מתקיימים שניים מהתנאים אזי G הוא עץ.

9.6 מפרקים וגשרים:

הגדרה - מפרק: נאמר שקודקוד v הוא מפרק. אם הסרת הקודקוד מהגרף תשאיר את הגרף לא קשיר.
הגדרה - גשר: נאמר ש e היא גשר. אם הסרת הצלע מהגרף תשאיר אותו לא קשיר.
טענה - קודקוד v הוא מפרק אם:

1. הוא שורש ביער ה DFS. יש לו לפחות שני בנים.

2. הוא לא שורש ביער ה DFS אך לאף אחד מהצאצאים של u לא יכול להתחבר אחורה יותר מ u. כלומר אן לאף אחד מהצאצאים צלע אחורה לאב קדמון של u.

טענה - צלע e היא גשר אם:

1. היא לא נמצאת על מעגל פשוט ב G.

אלגוריתם למציאת מפרקים וגשרים:
האלגוריתם בודק האם הבן קשור לאב קדום כך - אם v בן של u נבדוק האם האב הקודם ביותר של הבן v קטן ממש מהקודם ביותר של אב u. ואז u אינו מפרק והצלע אינה גשר.

9.7 בעיות גרפים ואלגוריתמים:

9.7.1 מציאת מסלול בין שני קודקודים בגרף:

יש שני אלגוריתמים למציאת מסלולים בגרף:

1. BFS **חיפוש לרוחב:** נסמן את כל הקודקודים כ $not-visited$. את הקודקודים שאנו מטפלים בהם נסמן כ $current$ ואת הקודקודים שביקרנו בהם נסמן כ $visited$.
נתחיל מהקודקוד s ונסמן אותו כ 0 אחכ נעבור על שכניו ונסמן אותם ב 1. כשנסיים לעבור על השכנים נעבור על השכנים שלהם וכן הלאה עד שנמצא את t.
לכל קודקוד נשמור שלש שדות: 1: האם הוא $visited, current, notvisited$. 2: הקודם של הקודקוד הנוכחי. 3: מרחק מקודקוד המוצא - s
כיצד נממש את האלגוריתם: נסמן את כל המרחקים כ ∞ ואת הקודמים כ $null$ וכ $notvisited$ נכניס את כל הקודקודים הנוכחיים למבנה נתונים של תור Q. ונעבור עליהם אחד אחד . נעדכן את המרחק והקודם של כל קודקוד

ולאחר שנסיים עם קודקוד נוציא אותו מהתור. ולבסוף נחזיר את המסלול
האלגוריתם מחזיר את המסלולים עם המרחקים הקצרים ביותר מקודקוד s לכל קודקוד.
סיבוכיות זמן ריצה: מכיוון שלא חוזרים על כל קודקוד פעמיים זמן הריצה הוא $O(|V|+|E|)$ צריך לשים לב שישנם מקרים שמתקיים $|E|=|V|^2$

2. DFS **חיפוש לעומק:**

בעזרת האלגוריתם ניתן לתור מספר בעיות: 1: מיון טופולוגי. 2: מציאת רכיבי קשירות. 3: עצים פורשים מינימלים.
כיצד האלגוריתם עובד: האלגוריתם עובד בשיטת באק טראקינג. נרד עד שניתקע, וכשניתקע נחזור אחורה.
מימוש: באותו האופן של BFS נשמור שדות של : 1: $visited, current, notvisited$ ..2: הקודם של הקודקוד הנוכחי. 3: מרחק מקודקוד המוצא - s. **בנוסף נוסיף עוד שני שדות** 4: זמן הכניסה לקודקוד $v.d\backslash pre$. 5: זמן היציאה מהקודקוד $v.f\backslash post$.
זמני הכניסה והיציאה חסומים על ידי $1\leq v.d<v.f\leq 2|V|$
נתחיל מקודקוד המקור ונרד לעומק . אחכ נלך לקודקוד הבא שסומן כ $notvisited$ עד שנסיים לעבור על כל הגרף אלגוריתם יחזיר מסלולים לכל הקודקודים - לא בהכרח המסלול הקצר ביותר.
סיבוכיות זמן ריצה: מכיוון שלא חוזרים על כל קודקוד פעמיים זמן הריצה הוא $O(|V|+|E|)$ צריך לשים לב שישנם מקרים שמתקיים $|E|=|V|^2$
ערך ההחזרה: האלגוריתם יחזיר להו את המסלול **ויער DFS**
חוק הסוגריים: משפט הסוגריים אומר שיש שלשה מצבים ביער ה DFS:
1: $(a(b,b)a)$ זאת אומרת ש b הוא בן של a.
2:$(a,a)(b,b)$ הקודקודים אינם שכנים.
מסקנה: מתקיים שקודקוד v צאצא של קודקוד u אם $u.d<v.d<v.f<u.f$
יער DFS בגרף מכוון: כשנעבור כל גרף מכוון נצטרך להבדיל בין הצלעות . יהיו לנו 4 סוגים של צלעות:
1: צלעות עץ - צלע רגילה בעץ
2: צלע אחורה - צלעות המחברות בין u לאב שלו.
3:צלע קדימה - צלע המחברת בין v לצאצא שלו. ואינן קשתות בעץ
4: גשר \ צלע חוצה - מקשרות קודקודים שאין בניהן יחס אב בן.
משפט: ביער DFS בגרף מכוון כל הקשתות הן קשתות אחורה או קשתות עץ. והוא חסר מעגלים אמ"מ ביער אין אף צלע אחורה.

9.7.2 מיון טופולוגי $topologicalsort(G)$:

מיון טופולוגי הוא מיון **לגרף מכוון ללא מעגלים** שמסדר את הגרף רק עם צלעות קדימה, כך שנעבור על הקודקודים לפי הסדר ולא נפספס אף קודקוד. באופן פורמלי - אם G מכיל צלע (u,v) אז u מופיע לפני v בגרף המוחזר מהאלגוריתם.
כיצד האלגוריתם עובד: נריץ על הגרף את האלגוריתם DFS ונבדוק עבור כל קודקוד האם הוא $visited$ אם כן - נוסיף אותו לתחילת הרשימה.
בעיקרון - הוא מסדר את הקודקוקים בסדר יורד, כך שערך ה $post$ הגבוה נמצא בראש הרשימה.
טענה: גרף מכוון G אינו מכיל מעגלים אמ"מ לאחר ריצת DFS על הגרף אין קשתות אחורה.

משפט: האלגוריתם מיון טופולוגי מחזיר גרף מכוון חסר מעגלים.
סיבוכיות זמן ריצה: מכיוון שאנו מריצים את DFS זמן הריצה הוא $O(|V|+|E|)$ צריך לשים לב שישנם מקרים שמתקיים $|E|=|V|^2$

9.7.3 גרף רכיבי קשירות scc:

האלגוריתם עובר על הגרף ומחזיר לנו גרף של רכיבי הקשירות ללא מעגלים.
הגדרה - גרף משוחלף : הגרף G^T נקרא הגרף המשוחלף של G וניצור אותו על ידי זה שנכוון כל צלע ב G לכיוון ההפוך.
טענה: גרף רכיבי הקשירות החזקה על G ועל G^T שווים.
כיצד האלגוריתם עובד: נריץ על הגרף את האלגוריתם DFS ואחכ נשחלף את הגרף ונריץ על הגרף המשוחלף את האלגוריתם DFS שוב לפי סדר ערכי ה $post$ **מהגבוה לנמוך**.
סיבוכיות זמן ריצה: אנו מריצים על הגרף את DFS פעמיים, ניתן לממש בעזרת איחוד קבוצות כך שזמן הריצה הוא $O(|V|+|E|\cdot\alpha(|V|))$ זהו זמן כמעט לינארי . צריך לשים לב שישנם מקרים שמתקיים $|E|=|V|^2$.

9.7.4 עצים פורשים מינימלים:

בהינתן גרף **קשיר,ממושקל ולא מכוון** נרצה למצוא את העץ הפורש המינימלי שמזיר מסלול מינימלי בין כל קדקודי העץ.
המשקל בין כל שני קודקודים שאין בניהם צלע שווה ל ∞.
נישם לב כי לכל גרף יכולים להיות כמה עצים פורשים מינימלים השונים בבחירת הצלעות אך שווים במשלק.
הגדרה - אלגוריתם חמדן: אלגריתם חמדן הוא אלגוריתם שמוסיף בכל פעם את הצלע שמשקלה הנמוך ביותר מבלי לחשוב על ההשפעות העתידיות.
סימונים:

1. נסמן ב V את כל קודקודי הגרף.
2. נסמן ב U את כל הקודקודים שאנו בוחרים להוסיף ל T - העץ הפורש.
3. נסמן ב $V-u$ את כל הקודקודים שטרפ הוספנו ל T .
4. **חתך:** חתך זהו החלק המחבר בין U ל $V-U$ ונסמנו ב C.
5. נאמר שהחתך מכבד את קבוצת הצלעות $A\subseteq E$ אמ"מ אין צלע ב a שחוצה את C.

יש שני אלגוריתמים לפתרון הבעיה:
1 - $kruskal$: אלגוריתם חמדן שבוחר בכל פעם את הצלע הנמוכה ביותר (ניתן לממש באמצעות ערימת מינימום) ובודק שהיא לא סוגרת מעגל, אם לא - נוסיף אותה בתור צלע של העץ.
מימוש: ניצור סט ונוסיף אליו את כל קודקודי הגרף, בכל פעם נוציא את הצלע המינימלית ונבדוק בעזרת $Union-Find$ שהיא לא סוגרת מעגל. לבסוף נחזיר את העץ הפורש.
משפט: אם נוסיף לעץ צלע מהחתך אזי היא תשמור על תכונות העץ ולא תיצור מעגלים - בעזרת משפט זה מוכיחים את נכונות האלגוריתם.
סיבוכיות זמן ריצה: אם נממש בעזרת איחוד קבוצות זמן הריצה יהיה $O(|E|\cdot log(|V|))$

2 ־ $prim$: נוסיף צלע רק אם היא המשך של העץ הקיים

מימוש: לכל קודקוד נגדיר את המשקל להיות אינסוף ואת הקודם להיות $null$. נוסיף את כל הקודקודים לתור ונבדוק כל עוד התור לא ריק, נוציא קודקוד ונמשיך דרך הצלע שמשקלה נמוך ביותר ונגיע כך לקודקוד הבא.

סיבוכיות זמן ריצה: אנו בונים ערימה ומוציאם כל פעם את השכן המינימלי לכן זמן הריצה הוא $O(|E| \cdot log(|V|))$

למת החתך: אם e היא צלע בחתך אזי היא חלק מעץ פורש מינימלי ־ בעזרת הלמה מוכיחים את נכונות האלגוריתם.

9.7.5 המסלולים הקצרים ביותר $shortedPathes$:

תכונות והגדרות :

1. **הגדרה ־ מעגל שלילי:** גרף עם מעגל שלילי יוגדר בתור משקל עם $-\infty$.
2. **תכונה:** המסלול הקצר ביותר בגרף בין שני קודקודים הוא חסר מעגלים **חיוביים**.
3. **תכונה:** תיתי מסלולים של המסלול הקצר ביותר ־ הם גם הקצרים ביותר.
4. **תכונת החסם העליון:** לכל $v \in V$ מתקיים כי $v.dist \geq \delta(s,v)$. כ"כ $v.dist = \delta(s,v)$ לא משתנה לעולם. כלומר ־ תמיד המרחק הקצר ביותר נשמר ורק קט ולעולם אינינו גדל.
5. **מונוטניות יורדת:** בסוף התהליך יתקיים $v.dist = \delta(s,v)$
6. **אי שוויון המשולש:** עבור כל צלע מתקיים $\delta(s,v) \leq \delta(s,u) + w(u,v)$.
7. **תכונת הקודם של תת הגרף:** אם $v.dist = \delta(s,v)$ אזי לכל קודקוד מתקיים כי עץ הקודמים הוא עץ רוחב של המסלולים הקצרים ביותר ששורשו הוא s .

האלגוריתם $Relaxe$: למעשה הוא המנוע של שני האלגוריתמים הבאים. הוא בודק אם יש מסלול במשקל נמוך יותר יותר בין שני קודקודים על ידי מעבר בקודקוד נוסף, ואם כן ־ הוא מעדכן את המרחק ואת הקודם.

יש שני אלגוריתמים לפתרון הבעיה: שני האלגוריתמים הינם חמדנים.

1. $Dijkstra$: **הערה:** האלגוריתם מניח כי הגרף שמתקבל הוא **חסר מעגלים שליליים.**
   מימוש: נממש בעזרת תור ונבדוק כל עוד התור לא ריק, נאתל את המרחקים להיות אינסוף ואת הקודמים היות $null$. נתחיל מהקודקוד s ונסתכל על כל שכניו. וכל פעם נמשיך דרך הצלע שמשקלה הוא המינימלי ביותר כלומר נריץ על כל קודקוד את האלגוריתם $relaxe$.
   סיבוכיות זמן ריצה: אם נממש בעזרת ערימת מינימוםזמן הריצה יהיה $O(|E| \cdot log(|V|))$
2. $Bellman-ford$: **הערה:** אם יש מעגל שלילי בגרף, האלגוריתם יחזיר לנו $false$.
   כיצד האלגוריתם עובד: אנו יודעים כי מספר הצלעות המקסימלי במסלול ללא מעגלים הוא $|V|-1$ לכן מספיק לעשות $V-1$ איטרציות ובאיטרציה הבאה ־ V נבדוק האם המסלול קטן. אם כן ז"א שיש בגרף מעגל שלילי.
   מימוש: ננפעיל את האלגוריתם $Relaxe$ $V-1$ פעמים ונריץ את הבדיקה הנ"ל .
   סיבוכיות זמן ריצה: אנו רצים על כל צלע V פעמים לכן זמן הריצה הוא $O(|E| \cdot |V|)$

9.7.6 מסלולים קצרים ביותר בין כל זוג קודקודים:

יש כמה דרכים לפתור את הבעיה, זמן הריצה הוא גדול לכן נרצה להקטין אותו כמה שיותר.
כיצד האלגוריתם יעבוד: האלגוריתם יקבל את הגרף בייצוג מטריציוני - מטריצת משקלים עם משקל כל צלע ואם אין צלע המשקל יהיה ∞. ויעשה פעולות על המטריצה המייצגת.
האלגוריתם פועל בשיטת כפל מטריצות - הוא מסתכל על **המטריצה הנוכחית מול מטריצת המשקלים** ובודק היכן ניתן לשפר את המסלול. כלומר - האם מסלול ארוך יותר דרך צלע אחרת תחזיר לנו מסלול במשקל קצר יותר.
ערך החזרה: האלגוריתם יחזיר שתי טריצות כך שכל שורה במטריצה מייצגת קודקוד יחיד . מטריצה אחת תייצג את המרחקים הקצרים ביותר. ומטריצה שניה תייצג את גרף הקודמים.
הנחה: האלגוריתם מסתמך על כך שתת מסלול של מסלול קצר ביותר הוא גם קצר ביותר.
מימוש: תחילה נאתחל את מטריצת המסלולים הקצרים ומטריצת הקודמים. ואחכ נחיל לבדוק את כל המסלולים הקצרים בעלי צלע אחת. באיטרציה הבאה נבדוק האם ניתן לשפר את המסלול בעזרת הוספת צלע ונבדוק את אורכי המסלולים בגודל 2 צלעות . כך בכל איטרציה נוסיף צלע עד שנגיע ל $|E|$.
שיפור האלגוריתם: ניתן לשפר את האלגוריתם אם במקום לחשב את המטריצה בשלב ה n מול מטריצת המשקלים, **נחשב את המטריצה בשלב** $\frac{n}{2}$ **מול עצמה פעמיים** (כמו בכפל חזקות שמקיים $a^n = a^{\frac{n}{2}} \cdot a^{\frac{n}{2}}$) וכך נחסוך ולולאה אחת באלגוריתם תרוץ בזמן של $log|V|$ פעמים במקום $|V|$ פעמים.
סיבוכיות זמן ריצה: אנו רצים בשלש לולאות ולולאה של כפל מטריצות לכן זמן הריצה הוא $O(|V|^3 \cdot log|V|)$

נרצה אלגוריתם שרץ בפחות זמן, לכן נציג את האלגוריתם הבא:
אלגוריתם פלוייד וורשל $Floyd - Warshal - algorithem$ **:**
הנחה: האלגוריתם מניח שאין מסלולים שליים בגרף.
בניגוד לאגוריתם שהראינו מקודם שרץ על **צלעות**, האלגוריתם הבא רץ על **קודקודים**. באתו האופן של האלגוריתם הקודם הוא עובד עם מטריצת משקלים ומחזיר מטריצת קודמים ומטריצת מסלולים קצרים עבור כל קודקוד.
סיבוכיות זמן ריצה: אנו רצים על שלש לולאות לכן זמן הריצה הוא $O(|V|^3)$

10 איחוד קבוצות זרות $Union - Find$:

נרצה מבני נתונים שיתמוך בפעולות הבאות: 1: יצירת סט. 2: איחוד קבוצות. 3: מציאת הקבוצה שהאיבר x שייך אליה.
מתי השתמשנו בזה: 1: בגרפים לחיפוש עץ פורש מינימלי בדקנו אם הצלע סוגרת מעגל באלגוריתם של $kruskal$. 2: במציאת רכיבי קשירות חזקה, בדקנו רכיבי קשירות לפי קבוצות.
הגדרות: נגדיר את מספר האלמנטים להיות n. ואת מספר הפעולות של איחוד ומציאת סט נגדיר בתור m. נקבל תמיד ש $m > n$
מימוש: יש שתי דרכים שניתן לממש איתן איחוד קבוצות:

1. **שימוש ברשימות מקושרות:** נייצג כל קבוצה באמצעות רשימה מקושרת, ונשמור בכל רשימה שתי שדות: 1 - $head$. 2 - $tail$. ראש הרשימה יוגדר בתור נציג הרשימה.

   בנוסף כל תא ברשימה יכיל שלש שדות: 1 - ערך הקודקוד. 2 - מתביע לאיבר הבא. 2 - מצביע לראש הרשימה.

סיבוכיות זמן:

יצירת סט: $O(1)$.

מציאת סט: נעבור על הרשימה וכשנמצא את האיבר נלך לראש הרשימה ע"י המצביע, ונחזיר את הראש. $O(n)$.

איחוד קבוצות: נחבר את הזנב של הרשימה הארוכה לראש של הרשימה הקצרה, כך שראש הרשימה יהיה ראש הרשימה הארוכה. ונעדכן את כל המצביעי של הרשימה הקצרה להצביע לראש הרשימה הארוכה. הזמן הממוצע יהיה $O(n)$

סיבוכיות זמן הריצה הכללי: $O(m + n \cdot log(n))$

2. **שימוש בעצים:** נייצג כל קבוצה בעזרת עץ. כל קודקוד בעץ יצביע על האב שלו, והשורש יצביע על עצמו.

סיבוכיות זמן:

יצירת סט: $O(1)$.

מציאת סט: נעבור על העץ עד שנמצא את המצביע ולבסוף נחזור אחורה לשורש ונחזיר אותו $O(log(n))$ כגובה העץ.

איחוד קבוצות: נאחד את העץ הקטן עם העץ הגדול. וכך למעשה גובה העץ יגדל לפי העץ שהוספו לו בכל פעם.

שיטה נוספת שנשתמש בה בכי להוריד את זמן הריצה נראת - **כיווץ מסלולים:** בשיטה זו אנו מחברים את העץ הנוסף ישירות לשורש העץ. לכן כל פעם שנחפש איבר מסויים. בדרך חזרה למעלה במעלה העץ נבצע כיווץ מסלולים ונאחד את כל תתי העצים לשורש וכך למעשה נקטין את גובה העץ.

זמן הריצה הכולל הוא: $O(m \cdot \alpha(n))$ כאשר $\alpha(n)$ מייצגת את הפונקציה ההופכית לפונקציית אקרמן ושווה ל $\alpha(n) < log^*(n) = log(log(log...(n)...))$.

הערה: בשיטה זו - זמן הריצה של מציאת רכיבי קשירות חזקה הוא $O(|V| + |E| \cdot \alpha(|V|))$ זהו למעשה זמן כמעט לינארי

וזמן הריצה של קרוסקל הוא $O(|V| + |E| \cdot log(|V|))$.