ממן 14

שאלה 1

אנו מחפשים מסלול בעל מחיר מזערי מהשכבה הימנית ביותר בשריג (i=1) לשכבה השמאלית ביותר בשריג (i=n).

הפתרון יכול להתחיל מכל מיקום k כאשר $1 \le k \le n$, כלומר להתחיל מהקדקוד 1,k. באותו אופן הוא יכול להסתיים בכל מיקום j כאשר $1 \le j \le n$, כלומר להסתיים בקדקוד n,j.

יהי O פתרון אופטימלי שפותר את הבעיה ותהי j השורה (המיקום) שבה מסתיים המסלול בעל המחיר המזערי מ-i=1 עד i=n. בפתרון האופטימלי O אנו מסיימים בשורה ה-j-ית.

כיוון שהצעדים המותרים הם מהצורה (i+1,j+1), (i+1,j-1), (i+1,j) אז על מנת לשחזר את המסלול בעל המחיר המזערי מהקדקוד n,j נבחר את המסלול בעל המחיר המזערי מהקדקודים הבאים:

(n-1,j) , (n-1,j+1) , (n-1,j-1)

אלו שלושת הקדקודים <u>היחידים</u> שדרכם ניתן להגיע אל הקדקוד n,j, ולכן המסלול האופטימלי מהשכבה הראשונה אל n,j <u>חייב לעבור דרך אחד מהשלשה</u>. כיוון שאנו מחפשים מסלול בעל <u>מחיר מזערי</u> נבחר את הקדקוד בעל המחיר המינימלי להגעה מהשכבה הראשונה מבין הקדקודים לעיל, ונוסיף את משקל הקדקוד הנוכחי שאנו נמצאים בו.

נגדיר נוסחת נסיגה בהתאם לתיאור:

<u>לכל $1 < i \le n$:</u>

$$OPT(i,j) = \min(OPT(i-1,j) , OPT(i-1,j+1) , OPT(i-1,j-1)) + c[i,j]$$

(<u>הערה:</u> ייתכן מצב שבו האינדקס j עלול לחרוג מגודל $n_x n$, במקרים אלו נחזיר ∞ כדי לטפל במקרי הקצה)

<u>עבור i=1:</u>

$$OPT(1,j) = c[1,j]$$

OPT(i,j) משמעותו המסלול האופטימלי (בעל מחיר מזערי) מהשכבה הראשונה (i=1) אל הקדקוד בעל הקואורדינטות i,j.

<u>נכונות נוסחת הנסיגה:</u>

- נוסחת הנסיגה תמיד תעצור כי בכל אחת מהקריאות אנו מקטינים את i ב-1 ולכן בשלב מסוים נוסחת הנסיגה תגיע ל-i=1.
- כפי שהוסבר לעיל בכלב שלב OPT(i,j) יחזיק את המסלול בעל המחיר המזערי מהשכבה הראשונה אל הקדקוד i,j כיוון שאל הקדקוד i,j ניתן להגיע רק דרך שלושת הקדקודים הנתונים בנוסחת הנסיגה, ולכן אם נבחר את המסלול בעל המחיר המינימלי מבין שלושת המסלול <u>האפשריים</u> ברור שזהו המסלול בעל המחיר המזערי.
- כיוון שאנו לא יודעים באיזה שורה j מסתיים המסלול האופטימלי, על מנת למצוא את המסלול האופטימלי מ-i=1 עד i=n נאלץ לחשב את כל הערכים מבין OPT(j,n) כאשר $1 \le j \le n$, ולאחר מכן למצוא את המינימום מתוך n ערכים אלו.

נממש את האלגוריתם שתואר לעיל באמצעות תכנון דינמי:

<u>האלגוריתם:</u>

<u>נתון:</u> מערך c בעל המשקלים של הקדקודים.

1. לכל i מ-1 עד n בצע:
    1.1. לכל j מ-1 עד n בצע:
        1.1.1. אם i=1 אז
            1.1.1.1. $M[i,j] \leftarrow c[1,j]$
        1.1.2. אחרת
            1.1.2.1. $M[i,j] \leftarrow \min(M[i-1,j], M[i-1,j-1], M[i-1,j+1]) + c[i,k]$
            (<u>הערה:</u> ייתכן מצב שבו האינדקס j עלול לחרוג מגודל המערך הדו-ממדי, במקרים אלו נחזיר ∞ כדי לטפל במקרי הקצה)
2. מצא את המינימום מבין $M[n,j]$ כאשר $1 \leq j \leq n$ ושמור את האינדקס j במשתנה k.
3. הדפס את n,k
4. $i \leftarrow n-1$
5. כל עוד $i > 0$ בצע
    5.1. מצא את המינימלי מבין $(M[i,j], M[i,j-1], M[i,j+1])$ והצב את אינדקס השורה ב-k
    (<u>הערה:</u> גם פה יש להתייחס לגבולות המערך כמו ב-1.1.2.1)
    5.2. הדפס את i,k
    5.3. $i \leftarrow i-1$

<u>נכונות האלגוריתם:</u>

נכונות האלגוריתם נובעת ישירות מנכונותה של נוסחת הנסיגה, האלגוריתם משתמש במערך דו-ממדי M בגודל $n_x n$ על מנת לאחסן את המסלול המזערי מהשכבה הראשונה אל הקדקוד i,j לכל זוג אינדקסים $M[i,j]$.

נוכיח כי בכל איטרציה הערכים במערך M קיימים:

לכל i,j אנו מחפשים את המינימום מבין (i-1,j-1) , (i-1,j+1) , (i-1,j) – כיוון שהלולאה החיצונית מקדמת את i מ-1 עד n, אזי כאשר נגיע לשורה i כלשהי תמיד יהיו כבר ערכים בשורה הקודמת לה, i-1, בכל אחד מהתאים $M[i-1,j], M[i-1,j+1], M[i-1,j-1]$.

לבסוף לאחר שחישבנו את כל הערכים נותר לנו עדיין למצוא את המסלול בעל המחיר המינימלי מבין כל הקדקודים בשכבה השמאלית ביותר (i=n). בסעיף 2 באלגוריתם אנו מוצאים את הערך הזה ומציבים אותו במשתנה k.

לאחר מכן אנו מדפיסים את המסלול בעל המחיר המזערי שמצאנו מ-n,k אל השכבה הימנית ביותר.
<u>הערה:</u> המסלול מודפס בסדר הפוך מהשכבה השמאלית ביותר אל הימנית ביותר, ניתן בקלות להפוך את הסדר.

<u>זמן ריצה:</u>

חלק 1 רץ על מערך דו-ממדי מסדר $n_x n$, ומבצע בו כמות עבודה קבועה לחישוב $M[i,j]$. לכן זמן הריצה של חלק זה הוא $O(n^2)$.

חלק 2 עובר על כל הערכים בשכבה השמאלית ביותר ומוצא את המינימום מביניהם, סה"כ $O(n)$.

חלקים 3-5 משחזרים את המסלול בעל המחיר המזערי ע"י מעבר על n ערכים וזמן קבוע של עבודה למציאת המינימום והדפסה, סה"כ $O(n)$.

לסיכום זמן הריצה של האלגוריתם הוא $O(n^2)$ כנדרש.

<u>שאלה 2</u>

נסמן את ערכו של הפתרון האופטימלי למציאת פלינדרום מרבי ב-OPT(i,j), OPT(i,j) יהיה שווה **לאורך** הפלינדרום המרבי במחרוזת הנתונה בין התווים i ל-j.

למשל בדוגמא הנתונה בשאלה עבור מחרוזת abcbea, אנו מחפשים את OPT(1,6) והוא יהיה 3, כי אורך הפלינדרום המרבי הוא 3 (אורכה של התת מחרוזת bcb).

יהי O פתרון אופטימלי עבור מחרוזת str באורך n. קיימים 2 אפשרויות עבור התו ה-n-י:

- התו הראשון והאחרון <u>שונים</u>, כלומר מתקיים $str[1] \neq str[n]$, ולכן יש להמשיך לבדוק את 2 תתי המחרוזות מהתו ה-1 ועד n-1, ומהתו ה-2 ועד n. משתי התתי המחרוזות יש לבחור את המקסימום וזה יהיה אורכו של הפלינדרום המרבי למחרוזת מהתו 1 ועד n.
  כלומר:

OPT(1,n) = max(OPT(1,n-1) , OPT(2,n))

ובאופן כללי:

OPT(i,j) = max(OPT(i,j-1) , OPT(i+1,j))

- התו הראשון והאחרון <u>שווים זה לזה</u>, או באופן כללי התו ה-i והתו ה-j שווים זה לזה. כלומר מתקיים str[i] = str[j]. המקרה הזה גם הוא מתחלק לתתי מקרים:
  - המחרוזת המלאה מ-i עד j תהיה פלינדרום מרבי אם"ם התת מחרוזת החל מהתו ה-i+1 ועד התו ה-j-1 גם היא פלינדרום מרבי. כיוון שאם התווים בקצוות שווים והתת מחרוזת באמצע איננה פלינדרום מרבי אז זהו לא פלינדרום מרבי (**חובה שהמחרוזת תהיה רציפה**).
    לפיכך יש לבדוק לפני כן ש- OPT(i+1,j-1) אכן מהווה פלינדרום מרבי.
    התת מחרוזת i+1,j-1 היא פלינדרום מרבי אם"ם OPT(i+1,j-1) שווה לאורכה של המחרוזת j-i+1 פחות 2 התווים i ו-j ששווים זה לזה. כלומר מתקיים השוויון הבא:
    OPT(i+1,j-1) = j – i + 1 – 2 = j – i – 1
    לפיכך אם התנאי לעיל מתקיים אז נגדיר:

OPT(i,j) = 2 + OPT(i+1,j-1)

  - <u>אחרת:</u> התת מחרוזת i+1,j-1 היא לא פלינדרום מרבי ולכן לא נכליל את השוויון של שני התווים i ו-j, ונמשיך לבדוק כמו במקרה שהם שונים זה מזה:

OPT(i,j) = max(OPT(i,j-1) , OPT(i+1,j))

<u>בנוסף נגדיר מקרי קצה:</u>

א. לכל i=j מתקיים OPT(i,j) = 1, כי מחרוזת באורך 1 היא פלינדרום מרבי.
ב. לכל j < i נגדיר OPT(i,j) = 0

<u>לסיכום מתקיים:</u>

לכל i=j:

OPT(i,j) = 1

לכל j < i:

OPT(i,j) = 0

אחרת:

if str[i] = str[j] **AND** OPT(i+1,j-1) = j – i – 1 **then**
    OPT(i,j) = 2 + OPT(i+1,j-1)
Else
    OPT(i,j) = max(OPT(i,j-1) , OPT(i+1,j))

כעת נממש את האלגוריתם שתואר לעיל באמצעות תכנון דינמי:

<u>האלגוריתם</u>:

<u>נתון</u>: מחרוזת str באורך n.

1. נאתחל את כל המשתנים שמקיימים j < i במערך הדו-ממדי 0 ← M[i,j]
2. לכל i מ-n עד 1 בצע:
   2.1. לכל j מ-i עד n בצע:
      2.1.1. אם i=j אז
         2.1.1.1. M[i,j] ← 1
      2.1.2. אחרת אם str[i] = str[j] וגם M[i+1,j-1] = j – i – 1 אז
         2.1.2.1. M[i,j] ← 2 + M[i+1,j-1]
      2.1.3. אחרת
         2.1.3.1. M[i,j] ← MAX(M[i+1,j] , M[i,j-1])
3. הדפס "הפלינדרום המרבי באורך M[1,n]"
4. j ← n , i ← 1
5. הגדר מערך תווים Result באורך M[1,n] ומונה k←1
6. כל עוד M[i,j] ≠ 0 בצע
   6.1. אם M[i+1,j] = M[i,j] אז
      6.1.1. i ← i+1
   6.2. אחרת אם M[i,j-1] = M[i,j] אז
      6.2.1. j ← j-1
   6.3. אחרת
      6.3.1. Result[k] ← M[i,j]
      6.3.2. Result[M[1,n]-k+1] ← M[i,j]
      6.3.3. i ← i+1
      6.3.4. j ← j-1
      6.3.5. k ← k+1
7. הדפס "הפלינדרום המרבי הוא Result"

<u>נכונות האלגוריתם:</u>

נכונותו של האלגוריתם נובעת מבניית הפתרון האופטימלי באמצעות נוסחת הנסיגה שתוארה לעיל.
האלגוריתם משתמש במערך דו-ממדי M בגודל $n_x n$ על מנת לאחסן את אורכו של הפלינדרום המרבי בתת המחרוזת i,j לכל זוג אינדקסים M[i,j] כאשר i≤j.

<u>נוכיח כי בכל איטרציה הערכים במערך M קיימים:</u>

אנו עוברים בלולאה החל מ-i=n בסדר <u>יורד</u> עד 1, וב-j=i בסדר <u>עולה</u> עד n. לפיכך בכל איטרציה של הלולאות המקוננות מתקיים תמיד i ≤ j.

כאשר i = j מציבים ערך 1 ב-M[i,j] כי מחרוזת באורך 1 היא פלינדרום.

כאשר i < j אז אנו משתמשים באחד משלושת מהתאים הבאים במערך:

M[i+1,j-1], M[i+1,j] , M[i,j-1]

הערכים M[i+1,j-1] , M[i+1,j] כבר קיימים כי אנו מטפלים ב-i בסדר <u>יורד</u>, ולכן אם נחזור לi+1 (שורה מתחת) כבר אתחלנו את כל הערכים בה באיטרציה הקודמת של הלולאה החיצונית.

הערך M[i,j-1] כבר קיים כי אנו מטפלים ב-j בסדר <u>עולה</u>, ולכן אם נחזור ל-j-1 (עמודה משמאל) כבר אתחלנו את הערך הזה באיטרציה הקודמת של הלולאה הפנימית.

לכל i > j ברור שישנם ערכים מהאתחול בשלב 1.

לבסוף לאחר שיש לנו את אורך הפלינדרום המרבי במחרוזת הנתונה str ב-M[1,n] <u>נשחזר אותו באמצעות צעדים 4-7 באלגוריתם:</u>

נאתחל מערך תווים Result שיחזיק את התוצאה ומונה k שיהיה אינדקס לתוצאה.

נאתחל את i להיות 1 ו-j להיות n ונרוץ בלולאה כל עוד $M[i,j] \neq 0$, כלומר יש פלינדרום מרבי באורך כלשהו.

אם M[i,j] **שווה לאחד התאים** M[i+1,j] או M[i,j-1] זאת אומרת שהוא לא גדל במהלך ריצת האלגוריתם, כלומר התווים str[i] ו-str[j] לא חלק מהפלינדרום המרבי, ולכן "נעקוב" אחר התא הגדול ביותר כדי לשחזר את הערך המתאים.

אם M[i,j] **לא שווה לאחד מהתאים** M[i+1,j] או M[i,j-1] זאת אומרת שהתווים str[i] ו-str[j] הם חלק מהפלינדרום המירבי, כי במהלך ריצת האלגוריתם הגדלנו את M[i,j] ב-2 (אורכם של 2 התווים הללו). לכן נציב ב-Result[k] = M[i,j] וגם Result[M[1,n]-k+1] = M[i,j] כדי לשמור על הערכים הללו בתחילתו וסופו של Result בהתאם לאינדקס k (ונקדם את כל האינדקסים בהתאמה).

בסופו של דבר נגיע לM[i,j] אשר שווה לאפס, הלולאה תסתיים ונדפיס את ערכו של הפלינדרום המרבי ששמרנו ב-Result.

<u>**זמן ריצה:**</u>

חלק 1 חסום ע"י גודלו של המערך הדו-ממדי ולכן חסום ע"י $O(n^2)$.

חלק 2 רץ על מערך דו-ממדי מסדר $n_x n$, ומבצע בו כמות עבודה קבועה לחישוב M[i,j]. לכן זמן הריצה של חלק זה הוא $O(n^2)$.

חלקים 3-7 מבצעים כמות עבודה קבועה ובנוסף משחזרת את הפלינדרום המרבי בזמן לינארי.

לסיכום זמן הריצה הוא $O(n^2)$.

שאלה 3

א. נתונות הנקודות $(x_1,y_1),\dots,(x_n,y_n)$.
נגדיר את הפולינומים q,r,s באופן הבא:

q(x) = $x_{j+1} - x$

r(x) = $x_i - x$

s(x) = $x_{j+1} - x_i$

נוכיח שמתקיים השוויון הנתון עם הפולינומים q,r,s שהגדרנו.

$$P_{i,j+1}(x) = \frac{(x_{j+1} - x)P_{i,j}(x) - (x_i - x)P_{i+1,j+1}(x)}{x_{j+1} - x_i}$$

על מנת להראות שמתקיים שוויון בין 2 הפולינומים ב-2 האגפים נראה כי יש להם אותם ערכים בכל הנקודות $x_i,\dots,x_{j+1}$. אם כל הערכים $y_i,\dots,y_{j+1}$ שווים עבור אותן נקודות בהתאמה אז הפולינומים שווים.

לפי הגדרה מתקיים עבור x_i,x_{j+1}:

$$P_{i,j+1}(x_i) = y_i \,,\, P_{i,j+1}(x_{j+1}) = y_{j+1}$$

נציב x_i באגף ימין של הפולינום.

$$\frac{(x_{j+1} - x_i)P_{i,j}(x_i) - (x_i - x_i)P_{i+1,j+1}(x_i)}{x_{j+1} - x_i} = \frac{(x_{j+1} - x_i)P_{i,j}(x_i)}{x_{j+1} - x_i} = P_{i,j}(x_i) = y_i$$

קיבלנו שמתקיים שוויון כנדרש.

נציב x_{j+1} באגף ימין של הפולינום.

$$\frac{(x_{j+1} - x_{j+1})P_{i,j}(x_{j+1}) - (x_i - x_{j+1})P_{i+1,j+1}(x_{j+1})}{x_{j+1} - x_i} = \frac{-(x_i - x_{j+1})P_{i+1,j+1}(x_{j+1})}{x_{j+1} - x_i} = \frac{(x_{j+1} - x_i)P_{i+1,j+1}(x_{j+1})}{x_{j+1} - x_i} = = P_{i+1,j+1}(x_{j+1}) = y_{j+1}$$

קיבלנו שמתקיים שוויון כנדרש.

כעת לכל k כאשר i<k<j+1 נראה שמתקיים שוויון ע"י הצבת x_k:

$$\frac{(x_{j+1} - x_k)P_{i,j}(x_k) - (x_i - x_k)P_{i+1,j+1}(x_k)}{x_{j+1} - x_i} = \frac{(x_{j+1} - x_k)y_k - (x_i - x_k)y_k}{x_{j+1} - x_i} = \frac{(x_{j+1} - x_k - x_i + x_k)y_k}{x_{j+1} - x_i} = \frac{(x_{j+1} - x_i)y_k}{x_{j+1} - x_i} = y_k$$

קיבלנו שמתקיים שוויון כנדרש, $y_k = (x_k)P_{i,j+1}$ וזה מה שרצינו להוכיח.

ב. נגדיר אלגוריתם תכנון דינמי לבעיית האינטרפולציה.

בנוסף נגדיר כי לכל $y_i = (x_i)P_{i,i}$. כיוון שזהו פולינום ממעלה 0 שעובר דרך הנקודה (x_i,y_i) כנדרש.

<u>האלגוריתם:</u>

<u>נתון:</u> 2 מערכים של הנקודות X ו-Y אשר $X[i] = x_i$ ו-$Y[i] = y_i$ כאשר $1 \le i \le n$.

1. לכל i מ-n עד 1 בצע:
   1.1. לכל j מ-i-1 עד n-1 בצע:
   1.1.1. אם i=j+1 אז
   1.1.1.1. $M[i,j+1] \leftarrow Y[i]$
   1.1.2. אחרת
   1.1.2.1. $M[i,j+1] \leftarrow \frac{(X[j+1]-x)M[i,j] - (X[i]-x)M[i+1,j+1]}{X[j+1]-X[i]}$
2. הדפס את הפולינום M[1,n]

<u>נכונות האלגוריתם:</u>

נכונותו של האלגוריתם נובעת מנכונותה של נוסחת הנסיגה שהוכחנו בסעיף א'.

האלגוריתם משתמש במערך דו-ממדי M בגודל $n_x n$, שלכל זוג אינדקסים i,j במערך הדו-ממדי M מתקיים ש-**M[i,j] הוא פולינום ממעלה i-j שעובר דרך הנקודות $(x_i,y_i),\ldots,(x_j,y_j)$**.

כיוון שהאינדקסים בנוסחת הנסיגה מתייחסים ל-$P_{i,j+1}$ אז בלולאה הפנימית j רץ החל מ-i-1 (במקום מ-i) ועד ל-n-1 (במקום עד n) ובתוך הלולאה מתייחסים ל-j+1 במקום j כך שהאינדקסים יסתדרו בכל שלב.

<u>נוכיח כי בכל איטרציה הערכים במערך M קיימים:</u>

אנו עוברים בלולאה החל מ-i=n בסדר <u>יורד</u> עד 1, וב-j=i-1 בסדר <u>עולה</u> עד n-1. לפיכך בכל איטרציה של הלולאות המקוננות מתקיים תמיד $i \le j+1$.

כאשר $i = j+1$ מציבים ערך Y[i] ב-M[i,j+1] כיוון שזהו פולינום ממעלה 0 שעובר דרך הנקודה (x_i,y_i) לפי ההגדרה.

כאשר $i < j+1$ אז אנו משתמשים בשני התאים במערך:

M[i,j], M[i+1,j+1]

הערך M[i+1,j+1] כבר קיים כי אנו מטפלים ב-i בסדר <u>יורד</u>, ולכן אם נחזור לi+1 (שורה מתחת) כבר אתחלנו את כל הערכים בה החל מ-j=i-1 של האיטרציה הקודמת.

הערך M[i,j] כבר קיים כי אנו מטפלים ב-j בסדר <u>עולה</u>, ולכן אם נחזור ל-j מ-j+1 (עמודה שמאלה) כבר אתחלנו את הערך הזה באיטרציה הקודמת של הלולאה הפנימית.

לא נגיע למצב ש- $j+1 < i$ כי j תמיד מאותחל להיות i-1 בכל סבב של הלולאה הפנימית והוא גדל בכל איטרציה.

בסיום ריצת האלגוריתם התוצאה מאוחסנת ב-$M[1,n]$, הלוא הוא $P_{1,n}$ – פולינום האינטרפולציה של הנקודות הנתונות $(x_1,y_1),\ldots,(x_n,y_n)$.

<u>זמן ריצה:</u>

חלק 1 של האלגוריתם רץ על לולאה מקוננת בגודל n, לכן חסום ע"י $O(n^2)$.

פנים הלולאה מבצע עבודה בסדר גודל של $O(1)$ מלבד חלק 1.2.1.1 אשר מבצע מכפלה, חיסור, חילוק פולינומים פשוטים מדרגה 0 או 1 עם פולינומים במערך M – נניח לצורך פשטות (כפי שנאמר בתרגיל) שפעולות אריתמטיות על מספרים הם פעולות אלמנטריות. מספר פעולות הכפל, חיסור וחילוק הינו קבוע ולכן נאמר שגם זמן ריצה זה הוא $O(1)$.

<u>לסיכום זמן הריצה של האלגוריתם הוא $O(n^2)$.</u>

ג. ראשית נציב את חמשת הערכים הנתונים על מנת לקבל את הנקודות הנתונות לאלגוריתם $(x_1,y_1),\ldots,(x_n,y_n)$.

$P(-2) = 46$
$P(-1) = 2$
$P(0) = 0$
$P(1) = 10$
$P(2) = 98$

קיבלנו את סדרת הנקודות:

$((-2,46),(-1,2),(0,0),(1,10),(2,98))$

נריץ את האלגוריתם מסעיף ב' ונשרטט את המערך הדו-ממדי M ונראה כיצד מקבלים את $p(x)$ הנתון ב-$M[1,n]$.

נתאר את מצבו של המערך הדו-ממדי M בסיומה של כל ריצת הלולאה החיצונית.

<u>לאחר i=5:</u>

X				
X	**X**			
X	**X**	**X**		
X	**X**	**X**	**X**	**98**

<u>לאחר i=4:</u>

X				
X	**X**			
X	**X**	**X**	**10**	**88x-78**
X	**X**	**X**	**X**	**98**

<u>לאחר i=3:</u>

X				
X	X	0	10x	$39x^2$-29x
X	X	X	10	88x-78
X	X	X	X	98

לאחר i=2:

X	2	-2x	$6x^2$+4x	$11x^3+6x^2$-7x
X	X	0	10x	$39x^2$-29x
X	X	X	10	88x-78
X	X	X	X	98

לאחר i=1:

46	-44x-42	$21x^2$+19x	$-5x^3+6x^2$+9x	$x + 2x^2 + 3x^3 + 4x^4$
X	2	-2x	$6x^2$+4x	$11x^3+6x^2$-7x
X	X	0	10x	$39x^2$-29x
X	X	X	10	88x-78
X	X	X	X	98

בסיומו של האלגוריתם קיבלנו את הפולינום p(x) ב-M[1,5] כפי שציפינו.

שאלה 4

א. האלגוריתם מחשב את המסלול בעל המשקל המינימלי מהצומת r∈V אל כל שאר הצמתים בגרף G. אם קיים מסלול מ-r אל v∈V אז A[v] יחזיק את המשקל המינימלי מ-r אל v, אחרת אם לא קיים מסלול כזה A[v] = ∞.
נוכיח באינדוקציה על מספר האיטרציות של הלולאה <u>החיצונית</u>.

<u>טענה:</u> **בסיום** כל איטרציה k, A[v] יכיל את המשקל **המינימלי** מבין כל המסלולים מ-r ל-v **שנסרקו עד כה** בכל האיטרציות. אם לא קיים מסלול כזה אז A[v]= ∞.

<u>עבור k=0:</u>
כל הצמתים מאותחלים ל-∞ =A[v] מלבד A[r] אשר שווה לאפס, כי המסלול בעל המשקל המינימלי מהצומת לעצמו הוא אפס.

<u>נניח את נכונותה של הטענה ל-k<n ונוכיח אותה עבור k=n:</u>
יהי v∈V צומת בגרף G. נבדוק את המצב של A[v] בסיום האיטרציה ה-n-ית.
ראשית נניח כי **קיים** מסלול מ-r אל v שטרם נסרק ב-n-1 האיטרציות הקודמות. לפי ההנחה צריך להוכיח ש-A[v] הוא המשקל המינימלי מבין המסלולים שנסרקו בסיום n האיטרציות.

יהי P מסלול מ-r אל v שלא נסרק בn-1 האיטרציות הראשונות של הלולאה החיצונית.
נסמן את P באופן הבא: P = r->v1->...->u->v, **ו-** u∈V כך שמתקיים e=(u,v)∈E.
לפי **הנחת האינדוקציה** מתקיים ש- A[u] מכיל את המשקל המינימלי מבין כל המסלולים מ-r ל-u שנסרקו עד כה בכל האיטרציות (כי אם אנו סורקים את P באיטרציה ה-n-ית, אז בהכרח A[u] כבר נסרק באיטרציות קודמות, אחרת הוא היה עדיין שווה ל-∞ ולא היינו סורקים את המסלול P באיטרציה הנוכחית).

לפיכך באיטרציה ה-n-ית נגיע לאי שוויון הבא A[v] > A[u] + c(e). אם האי-שוויון מתקיים אז נובע מכך שהמסלול P הוא בעל משקל נמוך יותר מהמשקל שנמצא כעת ב-A[v], ולפיכך **נבצע השמה ל-A[v] את המשקל הנמוך ביותר של המסלול שידוע לנו עד כה**, הלוא הוא A[u] + c(e). אם האי-שוויון לא מתקיים אז נובע מכך שקיים מסלול אחר מ-r אל v שכבר סרקנו באיטרציות קודמות, שמשקלו נמוך יותר מ-A[u] + c(e) ולכן A[v] ימשיך לשמור בערך זה.

לפיכך קיבלנו שבסיום האיטרציה ה-n-ית, A[v] מחזיק במשקל המינימלי מבין כל המסלולים מ-r ל-v שנסרקו עד כה – כנדרש.

נניח כעת **שלא קיים** מסלול מ-r ל-v ונוכיח שבסיום האיטרציה ה-n-ית מתקיים ∞= A[v].

נניח בשלילה ש-∞≠A[v]. לפי הנחת השלילה קיים u∈V כך שקיימת קשת (u,v)∈E ומתקיים האי שוויון A[v] > A[u] + c(e). האי שוויון מתקיים אם"ם ∞≠A[u], כלומר קיים מסלול באורך A[u] מ-r אל u. אם קיים מסלול מ-r אל u וקיימת קשת (u,v) אז בפרט קיים מסלול מ-r ל-v וזאת סתירה להנחת השלילה. לפיכך אם לא קיים מסלול מ-r אל v אז A[v] = ∞.

ב. יהי B(n) המספר המרבי של האיטרציות שמתבצעות בלולאה החיצונית על גרפים בעלי n קדקודים.

לפני שנחשב את B(n) נוכיח טענת עזר שתעזור לנו בחישוב:

טענת עזר: צומת v שאורך מסלולו מ-r הוא k יסרק **לכל היותר** באיטרציה ה-k-ית (כלומר A[v]≠ ∞ בסיום האיטרציה).
הוכחה: נוכיח באינדוקציה.
בסיס האינדוקציה:

- נראה שמתקיים עבור k=1:

באתחול A[v]= ∞ לכל v≠r. לכן באיטרציה הראשונה תמיד יתקיים האי שוויון A[v] > A[u] + c(e) כאשר u∈V צומת שכן של r ומתקיים e=(u,v)∈E, ותתבצע ההשמה אל A[v] בערך של A[u] + c(e) כנדרש.

- נניח את נכונות הטענה עבור k<n ונוכיח עבור k=n:

יהי v צומת שאורך מסלולו מ-r הוא n. בפרט קיים u במסלול זה אשר אורך מסלולו מ-r הוא n-1, כך שמתקיים A[u]≠ -∞ לפי הנחת האינדוקציה.
באיטרציה ה-n-ית האי-שוויון A[v] > A[u] + c(e) יתקיים, כי A[v]= ∞ ו-A[u]≠ ∞ ולכן תתבצע ההשמה A[v] ← A[u] + c(e).
זה מה שנדרשנו להוכיח.

כעת נחזור לחישוב B(n):

טענה: מספר האיטרציות המרבי (B(n)) הוא n, כאשר לכל היותר n-1 איטרציות מבצעות שינוי והאיטרציה האחרונה לא מבצעת שינוי ויוצאת מהלולאה החיצונית.
הוכחה: נניח בשלילה ש-B(n) = n+1, כלומר באיטרציה ה-n-ית מתבצע שינוי בערך A[v] כלשהו.
יהי v∈V צומת שערך A[v] שלו מתעדכן באיטרציה ה-n-ית.
אם A[v] התעדכן במהלך האיטרציה, אז בהכרח קיים מסלול בין r ל-v (לפי סעיף א'). כיוון ש-G הוא גרף בעל n צמתים, המסלול הפשוט הארוך ביותר בין r ל-v הוא באורך n-1 (אחרת היה במסלול מעגל וניתן להשמיט את הצמתים הללו).
לפיכך לפי טענת העזר – A[v]≠ ∞ לכל היותר לאחר n-1 איטרציות.

נסיק מכך שבמהלך האיטרציה ה-n-ית הערך של A[v] התעדכן בגלל <u>מסלול חדש</u> P שטרם נסרק באיטרציה הקודמת (ה-n-1). אמנם אם קיים מסלול שכזה אז בפרט קיים צומת u∈V שעבורו A[u]= ∞ עד האיטרציה ה-n-1, אחרת היינו כבר סורקים את המסלול הזה קודם.

אם u נסרק רק באיטרציה ה-n-1, אזי לפי טענת העזר המסלול מ-r ל-u הוא **בהכרח** n-1 (אחרת היינו מקבלים סתירה לכך ש-A[u]= ∞). אך אם אורך המסלול מ-r ל-u הוא באורך n-1, ו-u הוא חלק מהמסלול P מ-r ל-v, אז בהכרח אורך המסלול גדול מ-n-1, כלומר יש בו מעגל, ולפיכך ניתן להשמיט את הצמתים הפנימיים בו. מה שגורר לכך שאורך המסלול יהיה קטן מ-n-1 וגם זאת סתירה לטענת העזר.

בכל אחד מהמקרים לעיל הגענו לסתירה ולכן מספר האיטרציות המרבי הוא לכל היותר n-1 + איטרציה אחת ללא שינוי לסיום האלגוריתם, לסיכום $B(n) = n$.

נגדיר כעת סדרת גרפים G_n עליהם מתבצעות בדיוק $B(n)$ איטרציות.

$V = \{v_1,v_2,v_3,\ldots,v_{n-1},v_n\}$

$E = \{ (v_1,v_n), (v_2,v_1), (v_3,v_2), \ldots, (v_{n-1},v_{n-2}), (v_n,v_{n-1}) \} = \{ (v_i,v_{i-1}) \mid 1 < i \leq n\} \cup \{v_1,v_n\}$

נגדיר $r = v_n$.

כאשר עוברים על הקשתות בסדר לקסיקוגרפי, באיטרציה הראשונה מעדכנים אך ורק את $A[v_{n-1}]$, באיטרציה השנייה אך ורק את $A[v_{n-2}]$, וכך בכל איטרציה i מעדכנים את $A[v_{n-i}]$, עד שבאיטרציה ה-n-1 מעדכנים את $A[v_1,v_n]$.

באיטרציה ה-n לא מתבצע שינוי והאלגוריתם מסתיים.

ניתן להציג סדרת גרפים G_n בכך שנגדיר לכל הקשתות ב-G משקל בגודל n.

ג. נגדיר סדרת גרפים G'_n כך שנכנסים ללולאה החיצונית רק פעמיים ומתקיים $|E(G'_n)| = |E(G_n)|$

$V' = \{v_1,v_2,v_3,\ldots,v_{n-1},v_n\}$

$E' = \{ (v_1,v_2), (v_2,v_3), (v_3,v_4), \ldots, (v_{n-2},v_{n-1}), (v_{n-1},v_n) \} = \{ (v_i,v_{i+1}) \mid 1 \leq i < n\} \cup \{v_n,v_1\}$

נגדיר $r = v_1$.

כאשר עוברים על הקשתות בסדר לקסיקוגרפי, באיטרציה הראשונה מעדכנים את כל הצמתים בערכי $A[v]$ המתאימים להם. בלולאה **הפנימית** תחילה עוברים על (v_1,v_2) ומעדכנים את $A[v_2]$, לאחר מכן עוברים ל-(v_2,v_3) ומעדכנים את $A[v_3]$, וכך הלאה לכל i ולכל קשת $(v_i,v_{i+1}) \in E'$ מעדכנים את $A[v_{i+1}]$ לכל $1<i<n$.

באיטרציה השנייה לא מתבצע שינוי והאלגוריתם מסתיים.

ניתן להציג סדרת גרפים G'n בכך שנגדיר לכל הקשתות ב-'G משקל בגודל n.

מספר הקשתות זהה ומתקיים $|E(G'_n)| = |E(G_n)|$ כנדרש.