# סיכום אלגוריתמים

**1 הפרד ומשול (Divide & Conquer)**

1.1 Merge-Sort
1.2 Quick-Sort
1.3 מכפלת מחרוזות בוליאניות
1.4 כפל מטריצות (אלגוריתם Strassen)
1.5 בעיית הסלקציה (בפרט מציאת חציון)

**2 אלגוריתמים חמדניים (Greedy Algorithms)**

2.1 Fractional Knapsack + סלקצית פעילויות (Activity Selection)
2.2 דחיסת טקסט - קוד Huffman
2.3 אלגוריתמים על גרפים
2.3.1 עץ פורש מינימלי (Minimum Spanning Tree)
2.3.1.1 אלגוריתם Kruskal
2.3.1.2 אלגוריתם Prim
2.3.2 מסלולים קצרים ביותר (Shortest Paths)
2.3.2.1 אלגוריתם Dijkstra (Single Source Shortest Paths עבור משקלות לא-שליליים)
2.3.2.2 אלגוריתם Bellman-Ford (Single Source Shortest Paths עבור משקלות כלליים)
2.3.3 רשתות זרימה
2.3.3.1 שיטת Ford-Fulkerson
2.3.3.2 אלגוריתם Edmonds-Karp
2.3.3.3 התאמה מקסימלית של גרף דו-חלקי (Maximum Bipartite Matching) בעזרת רשתות זרימה

**3 תכנות דינאמי (Dynamic Programming)**

3.1 מציאת מסלולים קצרים ביותר בגרף (All Pairs Shortest Paths)
3.1.1 אלגוריתם נאיבי
3.1.2 אלגוריתם Floyd-Warshall
3.2 מיקום סוגריים אופטימלי לכפל שרשרת מטריצות
3.3 אלגוריתמים על מחרוזות
3.3.1 תת-סדרה משותפת מקסימלית (Longest Common Subsequence)
3.3.2 Edit Distance

**4 שונות**

4.1 מכפלת פולינומים בייצוג לפי מקדמים (בעזרת FFT)
4.2 בעיית הסוכן הנוסע (Traveling Salesperson Problem)

# 1 הפרד ומשול (Divide & Conquer)

הרעיון הכללי מאחורי האלגוריתמים בקטגוריה זו הוא לחלק את הבעיה לתתי בעיות קטנות יותר, ובעזרת רקורסיה לפתור אותן, ואז לאחד את הפתרונות.

## Merge-Sort 1.1

**הרעיון הכללי:** מחלקים את המערך ל2 תתי מערכים בגודל זהה כל פעם, וכל תת מערך ממיינים באותה שיטה (ברקורסיה). תנאי העצירה הוא כאשר מגיעים לתת מערך בגודל 1 (הוא כבר ממוין), בחזרה ממזגים את 2 המערכים הממוינים שהתקבלו. המיון הוא יציב, אך לא מבוצע "in place" (דורש זיכרון נוסף בגודל הקלט!).

**סיבוכיות:** זמן המיון על קלט שלם הוא פעמיים מיון על חצי קלט ועוד זמן לינארי של השילוב:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\Rightarrow T(n) \in \Theta(n \cdot \log n)$$

**:Pseudo-Code**

```
MergeSort (array A, int start, int end)
{
        if (start<end)
        {
                breakPoint = (start+end)/2;
                MergeSort(A, start, breakPoint);
                MergeSort(A, breakPoint+1, end);
                Merge(A, start, breakPoint, end);
        }
}

Merge (array A, int start, int breakPoint, int end)
{
        array B[start..end];
        i = k = start;      j = breakPoint+1;
        while (i<=breakPoint && j<=end)
        {
                if (A[i] <= A[j])
                        B[k++] = A[i++];
                else B[k++] = A[j++];
        }
        while (i<=breakPoint)    { B[k++] = A[i++]; }
        while (j<=end)  { B[k++] = A[j++]; }
        for i=start to end           { A[i] = B[i]; }
}
```

## Quick-Sort 1.2

**הרעיון הכללי:** בוחרים ציר (pivot) וממיינים את המערך סביבו – כלומר מעבירים מה שגדול ממנו לימינו והשאר לשמאלו. מבצעים רקורסיבית את המיון על 2 חלקי המערך (מימין ומשמאל לציר). המיון מבוצע "in place" (ללא זיכרון נוסף), אך אינו תמיד יציב (תלוי בדרך היישום).

**סיבוכיות:** תלויה בשיטת בחירת הציר.

<u>בחירה אקראית:</u> זמן כל שלב (מיון סביב הציר) הוא $\Theta(n)$. מס' השלבים במקרה הגרוע ביותר הוא $\Theta(n)$ גם כן (אם בוחרים ציר קיצוני בכל פעם) כלומר סיבוכיות $\Theta(n^2)$ סה"כ. במקרה הממוצע מס' השלבים הוא $\Theta(\log n)$ ולכן הסיבוכיות תהיה סה"כ $\Theta(n \cdot \log n)$.

<u>בחירה עפ"י "שיטת החמישיות"</u> : (ע"ע בעיית הסלקציה, 1.5) מוודאת שמס' השלבים הוא $\Theta(\log n)$ ולכן הסיבוכיות תהיה תמיד $\Theta(n \cdot \log n)$.

<u>המיון נחשב למהיר יותר מאשר Merge-Sort למרות שהסיבוכיות זהה (יש הבדל בקבועים).</u>

**1.3 מכפלת מחרוזות בוליאניות**

**הרעיון הכללי:** נתונות 2 מחרוזות בוליאניות מאורך n כל אחת. בצורה ישירה חישוב מכפלה של המחרוזות יהיה בסיבוכיות $\Theta(n^2)$. נראה אלגוריתם שמשפר את זמן המכפלה ע"י חלוקה של המחרוזות לחצאי-מחרוזות בצורה רקורסיבית וביצוע החישוב בעזרת חצאי המחרוזות כל פעם ע"מ לצמצם את מס' הפעולות.

**הנוסחא:** נסתכל על המחרוזות בצורה

$$X = X_1 \cdot X_2$$

$$Y = Y_1 \cdot Y_2$$

נוכל לחסוך פעולות כפל ע"י שימוש ב3 מכפלות מסדר n/2. נגדיר:

$$A = X_1 \cdot Y_1$$

$$B = X_2 \cdot Y_2$$

$$C = (X_1 + X_2) \cdot (Y_1 + Y_2)$$ (זו עדיין מכפלה של שני איברים מסדר n/2!)

ואז:

$$X \cdot Y = A + 2^{n/2}(C - A - B) + 2^n \cdot B$$ (המכפלה בחזקות של 2 נעשית ע"י פעולת shift בזמן קבוע).

**סיבוכיות:** זמן המכפלה של קלט שלם הוא 3 פעמים מכפלה של חצי קלט ועוד פעולות חיבור ו-shift בזמן לינארי:

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\Rightarrow T(n) \in \Theta(n^{\log_2 3}) \approx \Theta(n^{1.5849})$$

**1.4 כפל מטריצות (אלגוריתם Strassen)**

**הרעיון הכללי:** נתונות 2 מטריצות מסדר $n \times n$ כל אחת. בצורה ישירה חישוב מכפלת מטריצות יהיה בסיבוכיות $\Theta(n^3)$. גם כאן נשפר את הסיבוכיות ע"י חלוקה של המטריצות לרבעים בצורה רקורסיבית, וביצוע החישוב בעזרת רבעי המטריצות כל פעם ע"מ לצמצם את מס' הפעולות.

**הנוסחא:** נגדיר:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

כעת נגדיר 7 מטריצות ביניים:

$$M_1 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$M_2 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$M_3 = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})$$

$$M_4 = (A_{11} + A_{12}) \cdot B_{22}$$

$$M_5 = A_{11} \cdot (B_{12} - B_{22})$$

$$M_6 = A_{22} \cdot (B_{21} - B_{11})$$

$$M_7 = (A_{21} + A_{22}) \cdot B_{11}$$

ניתן לחשב את מכפלת המטריצות באמצעות 7 מטריצות הביניים שיצרנו:

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$C_{11} = M_1 + M_2 - M_4 + M_6$$

$$C_{12} = M_4 + M_5$$

$$C_{21} = M_6 + M_7$$

$$C_{22} = M_2 - M_3 + M_5 - M_7$$

**סיבוכיות:** זמן המכפלה של קלט שלם הוא 7 מכפלות על חצי קלט (המטריצות הקטנות הן מסדר $n/2\times n/2$) ועוד מס' קבוע של פעולות חיבור וחיסור בזמן לינארי:

$$T(n)=7T\left(\frac{n}{2}\right)+\Theta(n)$$
$$\Rightarrow T(n)\in\Theta(n^{\log_2 7})\approx\Theta(n^{2.81})$$

### 1.5 בעיית הסלקציה (בפרט מציאת חציון)

**הרעיון הכללי:** בהנתן סט $A$ של $n$ מספרים, יש למצוא את האיבר ה-$k$ בגודלו במקרה הכללי. אלגוריתם נאיבי (מיון הסט וגישה ישירה לאלמנט ה-$k$ בסט הממויין) יתבצע בסיבוכיות של $\Theta(n\cdot\log n)$ (זמן המיון). במקרים הפרטיים של $k=1$ או $k=n$ (מציאת מינימום או מקסימום) קל לראות שניתן למצוא את האיבר בזמן ליניארי. מה שנעשה במקרה הכללי ע"מ לשפר את הסיבוכיות הוא לבחור אלמנט ציר (pivot) ולבצע מיון סביבו (כמו איטרציה אחת של Quick-Sort) כך שהמערך יתחלק ל-3 חלקים: החלק של האיברים הקטנים מהציר, החלק של הציר עצמו והחלק של האיברים הגדולים מהציר. אחרי איטרציה אחת כזו הציר עצמו נמצא במקומו במערך, ולכן אם חיפשנו את האיבר במקום של הציר עצמו נחזיר את הציר וסיימנו. אחרת ניתן לבצע בצורה רקורסיבית על החלק הרלוונטי.

**בחירת ציר (pivot):** ע"מ להבטיח סיבוכיות טובה עלינו לוודא שכל ציר כזה מחלק את הסט לחלקים בגודל דומה (כלומר שבכל איטרציה אנחנו "נפטרים" מאחוז קבוע של איברים). לכן, בחירת הציר תיעשה באופן הבא:

(1) נחלק את A לקבוצות בנות 5 אלמנטים כל אחת. מס' הקבוצות הוא $m=\lceil n/5\rceil$.

(2) נמצא את החציון של כל קבוצה בצורה ישירה (brute-force). הזמן עבור כל קבוצה הוא קבוע, וסה"כ זמן שלב זה הוא $\Theta(n)$.

(3) נחשב את החציון של $m$ החציונים שמצאנו ע"י הפעלה רקורסיבית של האלגוריתם על קבוצת החציונים. הערך שיוחזר ישמש אותנו כציר.

ע"י שימוש באלגוריתם זה נקבל ציר שדרגתו $3n/10\le rank(pivot)\le 7n/10$, כלומר אנו נפטרים מיותר מרבע מהאיברים בכל איטרציה.

**סיבוכיות:** זמן מציאת האיבר המבוקש בסט השלם הוא זמן מציאת הציר + זמן המיון סביבו + זמן מציאת האיבר בתת-הסט המתאים:

$$T(n)=T\left(\frac{n}{5}\right)+T\left(\frac{3n}{4}\right)+n$$
$$\Rightarrow T(n)\in\Theta(n)$$

(הוכחה באינדוקציה).

# 2 אלגוריתמים חמדניים (Greedy Algorithms)

הרעיון הכללי מאחורי האלגוריתמים בקטגוריה זו הוא ביצוע סדרת החלטות שכל אחת מהן היא אופטימלית בפני עצמה, וביחד הן מביאות לפתרון אופטימלי גלובאלי.

### 2.1 Fractional Knapsack + סלקצית פעילויות (Activity Selection)

שני אלגוריתמים קצרים ופשוטים המדגימים את עקרון החמדנות:

**Fractional Knapsack – הבעיה:** גנב שודד חנות המכילה $n$ פריטים. כל פריט הוא בעל ערך $v_i$ ומשקל $w_i$. לשודד תרמיל המסוגל לשאת משקל $W$ לכל היותר. <u>ניתן לקחת גם חלק מפריט</u> (הגרסא המקורית שבה ניתן לקחת רק פריטים שלמים היא np-complete).

**הפתרון:** נסמן עבור כל פריט $i$ את הערך ליחידת משקל $p_i=v_i/w_i$, ונמיין את הפריטים לפי סדר זה. כעת לפי סדר יורד (מהגבוה לנמוך), אם ניתן לקחת את הפריט בשלמותו ניקח אותו, ואם לא ניקח עד כמה שאפשר.

**סיבוכיות:** זמן המיון $\Theta(n\cdot\log n)$ + זמן איסוף הפריטים $\Theta(n)$. סה"כ:

$$T(n)\in\Theta(n\cdot\log n)$$

**בחירת פעילויות (Activity Selection) – הבעיה:** נתון סט של $n$ פעילויות. כל פעילות מתחילה בזמן $s_i$ ומסתיימת בזמן $f_i$. קיום פעילות כלשהי תלויה בזמינות משאב יחיד – כלומר לא ניתן ש-2 פעילויות יתקיימו במקביל. דרוש לתזמן את הפעילויות השונות באופן <u>שמס' הפעילויות</u> שיתקיימו יהיה מקסימלי.

**הפתרון:** נמיין את הפעילויות לפי זמן הסיום, ובכל שלב נבחר את הפעילות שמסתיימת מוקדם ביותר אך לא מתנגשת עם פעילות שכבר בחרנו.

**סיבוכיות:** זמן המיון $\Theta(n\cdot\log n)$ + זמן בדיקה ועדכון הפעילויות $\Theta(n)$. סה"כ:

$$T(n)\in\Theta(n\cdot\log n)$$

### 2.2 דחיסת טקסט – קוד Huffman

**הרעיון הכללי:** ע"מ לצמצם בשטח האחסון, ניתן קוד לכל תו בטקסט. נבדוק את השכיחות של כל תו בטקסט, וכך ניתן קוד קצר יותר לתווים השכיחים יותר וקוד ארוך יותר לתווים הנדירים יותר. הביצוע נעשה בעזרת בנית עץ Prefix, שבו כל תו נמצא בעלה והקוד של התו הוא המסלול מהשורש ועד אליו כאשר כל פניה לבן שמאלי מסומנת 0 ופניה לבן ימני מסומנת 1.

**בניית העץ:** נשתמש ב-2 תורים. מתחילים כאשר לכל תו צומת משלו, ממיינים את הצמתים לפי סדר שכיחויות ומכניסים אותם לתור הראשון. בכל שלב בוחרים ומוציאים את 2 הצמתים הנמוכים ביותר מבין 2 הצמתים הראשונים בתור הראשון והצומת הראשון בתור השני (שבהתחלה עדיין ריק). יוצרים צומת חדש שיהיה האב של 2 הצמתים שבחרנו ושכיחותו היא סכום השכיחויות של 2 הבנים, ואותו מכניסים לתור השני. ממשיכים עד שנותרנו עם צומת אחד בלבד (אם בדרך התרוקן לנו התור הראשון מחליפים את התפקידים בין התורים – התור ה"חדש" ישמש לנו כתור הראשון והתור ה"מקורי ישמש בתור התור אליו נכניס את הצמתים החדשים שניצור, וחוזר חלילה).

**סיבוכיות:** אם יש לנו $n$ תווים שונים אזי מיון יתבצע בזמן $\Theta(n \cdot \log n)$. בכל שלב מציאת 2 צמתים מינמליים מתוך 3, יצירת צומת חדש והכנסתו לתור תיקח $\Theta(1)$. סה"כ מס' השלבים הוא $n-1$ כי בכל שלב מורידים 2 צמתים ומוסיפים צומת אחד לסה"כ הצמתים ב2 התורים, עד שנשאר צומת אחד. בסה"כ:

$$T(n) = \Theta(n \cdot \log n) + (n-1) \cdot \Theta(1)$$
$$\Rightarrow T(n) \in \Theta(n \cdot \log n)$$

### 2.3 אלגוריתמים על גרפים

#### 2.3.1 עץ פורש מינימלי (Minimum Spanning Tree)

**הבעיה:** בהנתן גרף קשיר, לא מכוון וממושקל, יש למצוא את תת הגרף בעל סכום המשקלות המינימלי המחבר בין כל הצמתים בגרף המקורי. מובן כי בגרף זה לא יתכנו מעגלים (סתירה למינימליות) ולכן הוא עץ.

##### 2.3.1.1 אלגוריתם Kruskal

**הרעיון הכללי:** נתחיל כאשר כל קודקוד בגרף נמצא בקבוצה (רכיב קשירות) משלו. נמיין את כל הקשתות בגרף בסדר יורד של משקלים, ובכל שלב נוסיף את הקשת המינימלית המחברת בין 2 קבוצות שונות, ונבצע איחוד של 2 הקבוצות. כאשר נותרה רק קבוצה אחת סיימנו.

**סיבוכיות:** יצירת קבוצה עבור כל צומת בגרף תיקח $\Theta(|V|)$. מיון הקשתות יתבצע בזמן $\Theta(|E| \cdot \log|E|)$. ביצוע סך כל פעולות האיחוד יקח $\Theta(|V| \cdot \log|V|)$ (אם מוסיפים את הקבוצה הקטנה לקבוצה הגדולה אז כל צומת יכול לעבור קבוצה מקסימום $\log|V|$ פעמים). סה"כ:

$$T(n) = \Theta(|V|) + \Theta(|E| \cdot \log|E|) + \Theta(|V| \cdot \log|V|)$$
$$\Rightarrow T(n) \in \Theta(|E| \cdot \log|E|)$$

**:Pseudo-Code**

```
MST_Kruskal (G(V,E,W))
{
        MST_Edges = Ø;
        for each u ∈ V
                Create_Set(u);
        Sort E in increasing order by w;
        for each (u,v) ∈ E (by sorting order)
                if (Find_Set(u) != Find_Set(v))
                {
                        Add (u,v) to MST_Edges;
                        Perform_Union(u,v);
                }
        return MST_Edges;
}
```

#### 2.3.1.2 אלגוריתם Prim

**הרעיון הכללי:** מתחילים מתת עץ של צומת בודד, ובכל שלב מוסיפים את הקשת המינימלית שמחברת צומת נוסף שאינו בתת העץ. כאשר תת העץ מכיל את כל הצמתים של העץ המקורי, סיימנו.
מאתחלים את הצומת ההתחלתי שבחרנו (אקראית) במרחק 0 ואת כל השאר באינסוף. בכל שלב בוחרים את הצומת המינימלי, ובודקים בכל שכניו האם הקשת ממנו אל השכן היא קטנה מהמרחק הנוכחי לשכן, ואם כן מעדכנים את מרחק השכן למשקל הקשת (<u>אין צורך לבצע בדיקה זו עבור שכנים שכבר נמצאים בתוך תת-העץ שלנו</u>). מוסיפים את הצומת שבחרנו לתת העץ, וחוזרים שוב, עד שכל הצמתים של העץ המקורי נמצאים בתת העץ.
**סיבוכיות:** $|V|$ שלבים (על כל צומת עוברים פעם אחת). בכל שלב צריך לבחור את הקודקוד שאליו מובילה הקשת הקלה ביותר, ולהוסיפה לעץ.
בחירת הקודקוד יכולה להתבצע ב-2 דרכים:

<u>שימוש בתור עדיפויות (ערימה):</u> בניית ערימה (בהתחלה) תתבצע בזמן $\Theta(|V|)$. הוצאת קודקוד מינימלי תתבצע בזמן $\Theta(\log|V|)$ (כולל תיקון הערימה), עבור $|V|$ קודקודים, סה"כ $\Theta(|V|\cdot\log|V|)$. עבור כל קודקוד שהוצאנו צריך לעדכן את המרחקים (Decrease Key) לכל הצמתים שיש ממנו קשתות אליהם – כל עדכון יתבצע בזמן $\Theta(\log|V|)$, ועבור כל קשת יהיה עדכון אחד, סה"כ $\Theta(|E|\cdot\log|V|)$.
סה"כ הסיבוכיות תהיה:

$$\Theta(|V|)+\Theta(|V|\cdot\log|V|)+\Theta(|E|\cdot\log|V|)$$
$$\Rightarrow \Theta(|E|\cdot\log|V|)$$

<u>שימוש במערך:</u> נשתמש במערך פשוט במקום בערימה. מציאת קודקוד מינימלי תתבצע בזמן $\Theta(|V|)$ (חיפוש סדרתי) עבור $|V|$ קודקודים, ועדכון מרחק לכל קודקוד (Decrease Key) יתבצע בזמן $\Theta(1)$ (כאמור יש עדכון אחד עבור כל קשת). סה"כ סיבוכיות:

$$|V|\cdot\Theta(|V|)+|E|\cdot\Theta(1)$$
$$\Rightarrow \Theta(|V|^2)$$

<u>**בגרף דליל כדאי להשתמש בערימה, ובגרף צפוף כדאי להשתמש במערך!**</u>

### 2.3.2 מסלולים קצרים ביותר (Shortest Paths)

בהקשר זה נטפל בבעיית Single Source Shortest Paths – מציאת המסלולים הקצרים ביותר מצומת התחלה <u>יחיד</u> לכל שאר צמתי הגרף.
מקרה פרטי הוא כאשר כל המשקלים שווים ל-1 ואז ניתן להשתמש ב-BFS מצומת המקור ע"מ למצוא את המרחק המינימלי לכל צומת.

#### 2.3.2.1 אלגוריתם Dijkstra (Single Source Shortest Paths) עבור משקלות לא-שליליים

**הרעיון הכללי:** דומה מאוד לאלגוריתם Prim. מאתחלים את צומת המקור במרחק 0, ואת שאר הצמתים באינסוף. בכל שלב בוחרים את הצומת $v$ בעל המרחק $\delta(v)$ המינימלי ובודקים את כל שכניו, כאשר אם $\delta(v)+w(v,u)<\delta(u)$, מעדכנים את מרחק השכן למרחק של הצומת שלנו + משקל הקשת לשכן. הצומת שבחרנו הגיע למרחק המינימלי שלו ולכן <u>אין צורך לבדוק מרחק אליו בשלבים הבאים</u>. חוזרים שוב, עד שעברנו על כל הצמתים. למעשה ההבדל היחיד מאלגוריתם Prim הוא קריטריון המרחק לצומת (ב-Prim זוהי הקשת המינימלית שמובילה אליו, וכאן זהו המסלול המינימלי שמוביל אליו).
<u>**סיבוכיות: זהה לסיבוכיות של אלגוריתם Prim!**</u>

#### 2.3.2.2 אלגוריתם Bellman-Ford (Single Source Shortest Paths) עבור משקלות כלליים

**הרעיון הכללי:** מאתחלים את המרחק של צומת המקור ל-0, ואת המרחק של שאר הצמתים לאינסוף. עוברים $|V|-1$ פעמים על כל הקשתות ועבור כל קשת (v,u) בודקים האם $\delta(v)+w(v,u)<\delta(u)$, ואם כן מעדכנים את $\delta(u)=\delta(v)+w(v,u)$. כאשר סיימנו לעשות את הלולאה עוברים פעם נוספת ובודקים אם עדיין יש שיפור, אם כן אזי מחזירים הודעה שיש מעגל שלילי בגרף (הסיבה: אם אין מעגל שלילי בגרף, המסלול המינימלי בין שני צמתים יכול להיות בעל $|V|-1$ קשתות לכל היותר, ולכן זהו גם מס' השיפורים המקסימלי). אם אין שיפור, אז עבור כל צומת $v$ בגרף $\delta(v)$ הוא המרחק המינימלי מצומת המקור לצומת $v$.

**הסיבוכיות:** מעבר על כל הקשתות בגרף $|V|$ פעמים (כאשר הבדיקות לוקחות זמן קבוע):

$$\Theta(|V|\cdot|E|)$$

**2.3.3 רשתות זרימה**

רשת זרימה היא גרף מכוון בו זרימה מתבצעת על פני הקשתות. לכל קשת יש קיבולת (capacity) המציינת את הזרימה המקסימלית האפשרית דרך אותה הקשת. המטרה היא למצוא את הזרימה המקסימלית שניתן לדחוס מצומת מקור s (source) לצומת בור t (sink).

**2.3.3.1 שיטת Ford-Fulkerson**

**הרעיון הכללי:** מתחילים עם זרימה 0, ובכל איטרציה מגדילים את הזרימה ע"י מציאת מסלול שיפור (augmenting path) שלאורכו ניתן לדחוס עוד כמות זרימה, עד שאין אפשרות להוסיף עוד זרימה ברשת. זוהי שיטה כללית, שלא אומרת כיצד לבחור מסלול שיפור (נניח שהוא נבחר אקראית כרגע).

עבור קשת $(u,v)$ בעלת קיבולת $c(u,v)$ וזרימה $f(u,v)$ נגדיר קיבולת שיורית (residual capacity) ע"י $c_f(u,v)=c(u,v)-f(u,v)$, כלומר כמה יחידות זרימה נוספות ניתן להזרים מ-$u$ ל-$v$ דרך הקשת. נבנה רשת שיורית (residual capacity) $G_f(V,E_f)$ כך ש-$(u,v)\in E_f \Leftrightarrow c_f(u,v)>0$. נתחיל עם רשת שבה זרימה 0, ניצור עבורה רשת שיורית, ונחפש ברשת השיורית מסלול מ-s ל-t. אם מצאנו מסלול כזה, נעדכן את רשת הזרימה המקורית בזרימה המקסימלית שניתן להעביר במסלול שמצאנו, נעדכן את הרשת השיורית בהתאם לזרימה החדשה וחוזר חלילה, עד שלא קיים מסלול מ-s ל-t ברשת השיורית.

**סיבוכיות:** מציאת מסלול שיפור אפשרית למשל ע"י שימוש ב-BFS או DFS בזמן $\Theta(|V|+|E|)$. בנית הרשת השיורית של הלולאה ועדכון הזרימה מתבצעות בזמן של $\Theta(|E|)$. אם נניח שהזרימה היא בערכים שלמים (או רציונליים) אזי בכל שלב יש שיפור של יחידה אחת לפחות, ולכן מס' השלבים הוא במקרה הגרוע ביותר $\Theta(\tilde{f})$ כאשר $\tilde{f}$ היא הזרימה המקסימלית האפשרית ברשת. סה"כ:

$$\Theta(|E|\cdot\tilde{f})$$

**2.3.3.2 אלגוריתם Edmonds-Karp**

**הרעיון הכללי:** שימוש יעיל בשיטת Ford-Fulkerson ע"י מציאת מסלול השיפור בעזרת BFS וכתוצאה מכך מציאת מסלול השיפור בעל מס' הקשתות המינימלי בכל שלב.

**סיבוכיות:** בכל מסלול שיפור יש לפחות קשת אחת "קריטית" שבה מזרימים זרימה בכל ערך הקיבולת השיורית שלה (כי אם אין קשת כזו ניתן למעשה להגדיל עוד את הזרימה באותו המסלול). כל קשת יכולה להיות קשת קריטית $O(|V|)$ פעמים לכל היותר (הוכחה ארוכה), ולכן מס' השלבים המקסימלי הוא מס' הקשתות כפול מס' הפעמים שכל קשת יכולה להיות קריטית כלומר $\Theta(|E|\cdot|V|)$. הזמן של כל שלב זהה ל-Ford-Fulkerson. סה"כ הסיבוכיות:

$$\Theta(|E|^2\cdot|V|)$$

**2.3.3.3 התאמה מקסימלית של גרף דו חלקי (Maximum Bipartite Matching) בעזרת רשתות זרימה**

**הבעיה:** התאמה בגרף לא מכוון היא קבוצת קשתות $M\subseteq E$ כך שלכל צומת $v\in V$ יש רק קשת אחת ש"נוגעת" (incident on) ב-$v$. התאמה מקסימלית היא ההתאמה בעלת מס' הקשתות הרב ביותר. אנחנו נתמקד בגרף דו חלקי, כלומר ניתן לחלק את כל הצמתים ל-2 קבוצות : L ו-R, כך שכל קשת מחברת בין צומת ב-L לצומת ב-R.

**הרעיון הכללי:** מיפוי הבעיה לבעית זרימה, כך שלכל התאמה תתאים זרימה מסויימת ולהפך. עבור הגרף הנתון $G(V,E)$ נגדיר רשת זרימה $G'(V',E')$ כך ש-$V'=V\cup\{s,t\}$. בנוסף, נגדיר $E'=\{(s,u):u\in L\}\cup\{(u,v):u\in L\wedge v\in R\}\cup\{(v,t):v\in R\}$ ונקבע את הקיבולות של כל הקשתות ב-$E'$ ל-1. כעת נשתמש בשיטת Ford-Fulkerson למציאת רשת זרימה מקסימלית, והזרימה ברשת זו תהיה ההתאמה המקסימלית שחיפשנו.

**סיבוכיות:** עוצמת התאמה כלשהי בגרף היא לכל היותר $\Theta(|V|)$, ומאחר וכל קשת בהתאמה מזרימה 1 ברשת, אזי זוהי גם גם הזרימה המקסימלית ברשת, ולפי הסיבוכיות של שיטת Ford-Fulkerson סה"כ הסיבוכיות היא:

$$\Theta(|V|\cdot|E|)$$

# 3 תכנות דינאמי (Dynamic Programming)

הרעיון הכללי מאחורי האלגוריתמים בקטגוריה זו הוא פירוק הבעיה למס' תת-בעיות קטנות יותר, פתירת כל תת-בעיה באופן אופטימלי, וחישוב bottom-up בעזרת טבלה (תוך הסתמכות על פתרונות תת-הבעיות). כך נמנעים מחישובים שנעשים יותר מפעם אחת (מה שנעשה הרבה באלגוריתמים בשיטת "הפרד ומשול").

**3.1 מציאת מסלולים קצרים ביותר בגרף (All Pairs Shortest Paths)**

בבעיה זו אנו רוצים למצוא את המרחק המינימלי בין כל 2 צמתים בגרף ממושקל כלשהו.

### 3.1.1 אלגוריתם נאיבי

**הרעיון הכללי:** מקבלים את הגרף ביצוג של מטריצת סמיכויות $W_{n\times n}$ שבה כל אלמנט שווה למשקל הקשת המתאימה, ומחזירים מטריצה $D_{n\times n}$ כך שכל אלמנט $d_{ij}$ יהיה שווה למרחק המקסימלי בין קודקוד *i* לקודקוד *j*. נסמן ב- $d_{ij}^{(m)}$ את עלות המסלול הקצר ביותר מ-*i* ל-*j* המכיל לכל היותר *m* קשתות.

ברור ש- $D^{(1)}$ היא מטריצת המשקלים שקיבלנו בקלט. נחשב בצורה אינדוקטיבית את $D^{(m)}$ מתוך $D^{(m-1)}$:

<u>מקרה 1:</u> אם המסלול הקצר ביותר בין *i* ל-*j* הוא בעל פחות מ-*m* קשתות, אזי $d_{ij}^{(m)}=d_{ij}^{(m-1)}$.

<u>מקרה 2:</u> אם המסלול הקצר ביותר מכיל בדיוק *m* קשתות אזי ניתן לפרק את המסלול למסלול מינימלי בן *m*-1 קשתות מ-*i* ל-*k* כלשהו + הקשת (*k*,*j*), ואז $d_{ij}^{(m)}=d_{ik}^{(m-1)}+w_{kj}$.

<u>בסך הכל:</u>

$$d_{ij}^{(m)}=\min\left(d_{ij}^{(m-1)},\min_{1\le k\le n}\left(d_{ik}^{(m-1)}+w_{kj}\right)\right)$$

כעת מה שנותר הוא למלא את הטבלה לפי האינדוקציה מלמטה למעלה עד לשלב ה-*n*-1 (לא יתכן שמסלול קצר ביותר בגרף בעל n קודקודים יעבור דרך יותר מאשר *n*-1 קשתות).

**סיבוכיות:** הטבלה היא בגודל $n\times n$ ובכל שלב ממלאים אותה מחדש – סה"כ חישוב של $\Theta(n^3)$ תאים. חישוב כל תא מתבצע בזמן $O(n)$ (כי מחשבים את הגודל עבור כל *k* אפשרי). סה"כ זמן הריצה:

$$T(n)\in\Theta(n^4)$$

### 3.1.2 אלגוריתם Floyd-Warshall

**הרעיון הכללי:** דומה לאלגוריתם הנאיבי שראינו. בשונה ממנו, נגדיר את $d_{ij}^{(m)}$ כאורך המסלול הקצר ביותר מ-*i* ל-*j* כך שצמתי הביניים במסלול חייבים להשתייך לסט $\{1,2,\ldots,m\}$.

<u>מקרה 1:</u> אם המסלול הקצר ביותר בין *i* ל-*j* אינו עובר בצומת *m*, אזי $d_{ij}^{(m)}=d_{ij}^{(m-1)}$.

<u>מקרה 2:</u> אם המסלול הקצר ביותר עובר בצומת *m* אזי ניתן לפרק את המסלול ל-2 מסלולים מינימליים מ-*m* ל-*i* ומ-*m* ל-*j*. כמו"כ כל אחד מהמסלולים חייב לעבור בצמתים מהסט $\{1,2,\ldots,m-1\}$ בלבד, ואז $d_{ij}^{(m)}=d_{im}^{(m-1)}+d_{mj}^{(m-1)}$.

<u>בסך הכל:</u>

$$d_{ij}^{(m)}=\begin{cases} w_{ij} & m=0 \\ \min\left(d_{ij}^{(m-1)},d_{im}^{(m-1)}+d_{mj}^{(m-1)}\right) & o/w \end{cases}$$

שוב, מה שנותר הוא רק למלא את הטבלה לפי האינדוקציה ($\Theta(n)$ שלבים).

**סיבוכיות:** מס' התאים שממלאים זהה לאלגוריתם הנאיבי - $\Theta(n^3)$, אך בניגוד אליו עדכון כל תא לוקח זמן קבוע, ולכן סה"כ הסיבוכיות:

$$T(n)\in\Theta(n^3)$$

### 3.2 מיקום סוגריים אופטימלי לכפל שרשרת מטריצות

**הבעיה:** כפל מטריצות הוא אסוציאטיבי, כלומר ניתן לשנות את סדר הקדימויות של פעולות הכפל בינן לבין עצמן מבלי שהתוצאה תשתנה. מספר הפעולות, לעומת זאת, כן משתנה (בהנחה שלא כל המטריצות מאותו הסדר), והבעיה היא כיצד ניתן למצוא את מיקום הסוגריים במכפלה של *n* מטריצות, שבו מס' הפעולות יהיה מינימלי.

**הרעיון הכללי:** נפתור את הבעיה עבור תת-שרשראות ונבנה טבלה *A* בה $a_{ij}$ ייצג את מס' הפעולות המינימלי במכפלה $M_i\cdot M_{i+1}\cdot\ldots\cdot M_j$.

נבנה את הטבלה בצורה אינדוקטיבית:

<u>אם *i=j*:</u> אזי הסדרה כוללת מטריצה יחידה ועלות המכפלה היא 0.

<u>אחרת:</u> עבור כל ערך $i\le j<k$ הזמן האופטימלי לחישוב $M_i\cdot M_{i+1}\cdot\ldots\cdot M_k$ הוא $a_{ik}$, הזמן האופטימלי לחישוב $M_{k+1}\cdot M_{k+2}\cdot\ldots\cdot M_j$ הוא $a_{kj}$ ומכיוון ש- $M_i\cdot M_{i+1}\cdot\ldots\cdot M_k$ היא מסדר $p_{i-1}\times p_k$ ו- $M_{k+1}\cdot M_{k+2}\cdot\ldots\cdot M_j$ היא מסדר $p_k\times p_j$ אזי מכפלתן כרוכה ב- $p_{i-1}\cdot p_k\cdot p_j$ פעולות כפל, ובסה"כ:

$$A_{ij}=\min_{i\le k<j}\left(A_{ik}+A_{k+1j}+p_{i-1}\cdot p_k\cdot p_j\right)$$

**סיבוכיות:** זמן מילוי הטבלה, כאשר גודל הטבלה הוא $n \times n$, וזמן חישוב כל תא הוא $O(n)$ (בדיקה עבור כל $k$ אפשרי), סה"כ:

$$T(n) \in \Theta(n^3)$$

## 3.3 אלגוריתמים על מחרוזות

### 3.3.1 תת-סדרה משותפת מקסימלית (Longest Common Subsequence)

**הבעיה:** בהנתן סדרת תווים $X = \langle x_1, x_2, \ldots, x_n \rangle$, נאמר כי $Z = \langle z_1, z_2, \ldots, z_k \rangle$ תת סדרה של $X$ אם קיימת סדרה עולה של $k$ אינדקסים כך ש- $Z = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k} \rangle$. במקרה שלנו, נתונות שתי סדרות $X$ ו-$Y$ ואנו רוצים למצוא את תת-הסדרה הארוכה ביותר המשותפת לשתיהן.

**הרעיון הכללי:** כרגיל בתכנות דינאמי נבנה טבלה $m \times n$ (בגודל אורכי המחרוזות) של פתרונות לתתי-בעיות בעזרת אינדוקציה עד שנגיע לפתרון הבעיה השלמה:

בסיס: $a_{i0} = a_{0j} = 0$.

מקרה 1: אם התווים האחרונים זהים $x_i = y_j$ ברור שהם יכללו בתת-הסדרה המשותפת ולכן $a_{ij} = a_{i-1j-1} + 1$.

מקרה 2: אם התווים האחרונים שונים $x_i \neq y_j$ אז יש שתי אפשרויות: או שאורך תת-הסדרה הוא $a_{ij} = a_{i-1j}$, או שאורך תת הסדרה הוא $a_{ij} = a_{ij-1}$ (המקסימלי מביניהם), כלומר $a_{ij} = \max(a_{i-1j}, a_{ij-1})$.

מה שנותר הוא למלא את הטבלה ואז אורך תת הסדרה המקסימלית יהיה $a_{mn}$. כמובן שהטבלה שלנו נותנת רק את האורך המקסימלי, אבל ניתן בקלות לשחזר מתוכה (או לבנות במקביל טבלת עזר שתאפשר לשחזר) את תת-הסדרה המקסימלית עצמה.

**סיבוכיות:** זמן בניית הטבלה, סה"כ $\Theta(m \cdot n)$ תאים, וכל תא מחושב בזמן קבוע. סה"כ:

$$T(n) \in \Theta(m \cdot n)$$

### 3.3.2 Edit Distance

**הבעיה:** בהנתן טקסט תווים $T = t_1 t_2 \ldots t_n$ ותבנית (pattern) $P = p_1 p_2 \ldots p_m$ יש למצוא לכל מקום $k$ בטקסט את מס' השגיאות המינימלי שעבורו התבנית מתאימה לטקסט שמסתיים במקום ה-$k$. נגדיר 3 סוגי שגיאות:

Mismatch: כאשר יש התאמה מלאה (כולל אינדקסים!) למעט אות אחת, נאמר שיש שגיאה אחת מסוג mismatch. לדוגמא:

T = abcde
P = abade

Insertion: כאשר היתה יכולה להיות התאמה, אך הוכנסה אות נוספת לטקסט, נאמר שיש שגיאה אחת מסוג insertion. לדוגמא:

T = abcde
P = acde

כאן אפשר לומר שיש 3 שגיאות mismatch או שגיאת insertion אחת.

Deletion: כאשר היתה יכולה להיות התאמה, אך הוסרה אות אחת מהטקסט, נאמר שיש שגיאה אחת מסוג deletion. לדוגמא:

T = acde
P = abcde

שוב, כאן ניתן לומר שיש 3 שגיאות mismatch או שגיאת deletion אחת.

**הרעיון הכללי:** כרגיל בתכנות דינאמי נבנה טבלה $m \times n$ (בגודל אורכי המחרוזות) של פתרונות לתתי-בעיות בעזרת אינדוקציה עד שנגיע לפתרון הבעיה השלמה. האיבר ה- $a_{ij}$ יהיה שווה למס' המינימלי של פעולות edit כך ש- $p_1 p_2 \ldots p_i$ תתאים לטקסט המסתיים במקום ה-$j$:

בסיס: אם $i = 0$ מדובר בעצם בתבנית ריקה ולכן $a_{0j} = 0$ (אין צורך בשום פעולות עריכה כדי להתאים את התבנית). אם $j = 0$, אז הטקסט הוא ריק, כלומר ניתן לחשוב שבוצעו $i$ פעולות מחיקה על הטקסט ולכן $a_{i0} = i$.

מקרה 1: אם התווים זהים $p_i = t_j$ אזי לא נדרשת שום פעולת עריכה ולכן $a_{ij} = a_{i-1j-1}$.

מקרה 2: אם התווים שונים $p_i \neq t_j$ אזי ניקח את המינימום מפעולות העריכה: או mismatch ואז $a_{ij} = a_{i-1j-1} + 1$, או insertion ואז $a_{ij} = a_{ij-1} + 1$, או deletion ואז $a_{ij} = a_{i-1j} + 1$. בסה"כ:

$$a_{ij} = \min(a_{i-1j-1}, a_{i-1j}, a_{ij-1}) + 1$$

לאחר שסיימנו למלא את הטבלה, השורה האחרונה היא למעשה כל ההתאמות של התבנית המלאה הטקסט שמסתיים בכל מקום $j$. כמו באלגוריתם הקודם, קל לשחזר (או לבנות במקביל טבלת עזר שתאפשר לשחזר) את המסלול ואת פעולות ה-edit שבוצעו.

**סיבוכיות:** זמן בניית הטבלה, סה"כ $\Theta(m \cdot n)$ תאים, וכל תא מחושב בזמן קבוע. סה"כ:

$$T(n) \in \Theta(m \cdot n)$$

# 4 שונות

## 4.1 מכפלת פולינומים בייצוג עפ"י מקדמים (בעזרת FFT)

**הבעיה:** ישנן 2 דרכים לייצג פולינום מדרגה $n$ – ע"י מקדמים ( $P(x)=\sum_{j=0}^{n-1} a_j \cdot x^j$ ) או ע"י ערכים נקודתיים ( $P(x)=\{(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)\}$ ). חיבור 2 פולינומים לפי 2 השיטות מתבצע בזמן $\Theta(n)$, אבל מכפלת 2 פולינומים מתבצעת בזמן $\Theta(n^2)$ בייצוג לפי מקדמים, בעוד שבייצוג לפי ערכים נקודתיים היא מתבצעת גם כן בזמן $\Theta(n)$. אנו מעוניינים לשפר את הסיבוכיות של מכפלת פולינומים לפי מקדמים (מאחר ולרוב נוח יותר לעבוד בייצוג לפי מקדמים).

**הרעיון הכללי:** מאחר והמכפלה בייצוג לפי ערכים נקודתיים זולה יותר, נרצה להעביר את הפולינום לצורה זו, לבצע את המכפלה בזמן $\Theta(n)$, ולחזור חזרה לייצוג לפי מקדמים. ע"מ לבצע את המעברים נשתמש בהתמרת פורייה הבדידה (Discrete Fourier Transform), שעליה נבצע שיפור כך שתהיה מהירה יותר (Fast Fourier Transform). זאת ע"י שימוש בשורשי היחידה המרוכבים (קומפלקסיים) וע"י שימוש בטכניקת הפרד ומשול.

**הנוסחא:** תוצאת מכפלה של 2 פולינומים מדרגה $n/2$, $A(x)=\sum_{j=0}^{n/2-1} a_j \cdot x^j$ ו- $B(x)=\sum_{j=0}^{n/2-1} b_j \cdot x^j$ היא פולינום מדרגה $n$. מאחר ואנו מבצעים את המכפלה בייצוג לפי ערכים נקודתיים, אנו זקוקים ל-$n$ נקודות של כל פולינום (למרות שכאמור במצב רגיל מספיקות $n/2$ נקודות).
הנקודות שנבחר יהיו שורשי היחידה מסדר $n$ - $w_n^0, w_n^1, \dots, w_n^{n-1}$ הניתנים לחישוב ע"י:

$$w_n^k = e^{\frac{2\pi \cdot i}{n} \cdot k}$$

(יש לשים לב ש-$i$ מייצג את השורש של 1- ולא אינדקס)

חישוב $n$ נקודות בפולינום מדרגה $n/2$ מתבצע ע"י הנוסחא:

$$y_k = P(w_n^k) = \sum_{j=0}^{n/2-1} a_j \cdot w_n^{jk}$$

עבור כל ערך. סיבוכיות זמן החישוב של וקטור הערכים עבור כל פולינום היא עדיין $\Theta(n^2)$, ולכן נשפר את החישוב ע"י שימוש בטכניקת "הפרד ומשול". נפריד את הפולינום למקדמים הזוגיים והאי-זוגיים, כך ש:

$$P^{even}(x) = a_0 + a_2 \cdot x + a_4 \cdot x^2 + \dots + a_{n/4-2} \cdot x^{n/4-1}$$

$$P^{odd}(x) = a_1 + a_3 \cdot x + a_5 \cdot x^2 + \dots + a_{n/4-2} \cdot x^{n/4-1}$$

$$P(x) = P^{even}(x^2) + x \cdot P^{odd}(x^2)$$

מאחר ו- $n/2$ שורשי היחידה הראשונים מסדר $n$ כאשר מעלים אותם בריבוע מהווים את $n/2$ שורשי היחידה מסדר $n/2$, בעצם מיפינו את הבעיה ל-2 תת בעיות בחצי הגודל, כאשר זמן חיבור תת הבעיות הוא לינארי. כל תת-בעיה ניתן גם כן לפצל ברקורסיה, וכן הלאה.
לאחר שהעברנו את 2 הפולינומים לייצוג לפי ערכים נקודתיים, נבצע את המכפלה נקודה מול נקודה (כאמור, בזמן לינארי). מה שנותר הוא רק להחזיר את פולינום התוצאה שקיבלנו בצורת ערכים נקודתיים, לייצוג לפי מקדמים. נבנה פולינום עזר $d(x)$ המוגדר:

$$d(x) = \sum_{j=0}^{n-1} c(w_n^{-j}) \cdot x^j$$

כאשר $c(w_n^{-j})$ הוא הערך של הפולינום שחישבנו בנקודה $w_n^{-j}$ (למעשה עוברים על שורשי היחידה מהסוף להתחלה). בעזרת פולינום העזר שבנינו, נחשב את מקדמי פולינום התוצאה $C(x)=\sum_{j=0}^{n/2-1} c_j \cdot x^j$ ע"י הנוסחא:

$$c_j = \frac{d(w_n^j)}{n}$$

שוב עלינו לחשב $n$ פולינומים מסדר $n$ (זמן brute-force $\Theta(n^2)$), ולכן נשתמש שוב בפתרון "הפרד ומשול" שראינו לחישוב ערכי פולינום בשורשי היחידה, ואז מתקבל פולינום המכפלה בייצוג לפי מקדמים, כמתבקש.

**סיבוכיות:** זמן חישוב ערכי הפולינומים הנתונים בשורשי היחידה בעזרת הטכניקה שראינו הוא רקורסיה מהצורה: $T(n)=2T\left(\frac{n}{2}\right)+\Theta(n)$ (עבור כל פולינום) שזמן הריצה שלה הוא $\Theta(n \cdot \log n)$. חישוב מכפלה בייצוג לפי ערכים נקודתיים נעשה כאמור בזמן $\Theta(n)$. בניית פולינום

העזר מתבצעת גם היא בסיבוכיות $\Theta(n)$, וחישוב ערכי פולינום העזר ע"מ לקבל את מקדמי פולינום התוצאה נעשית שוב לפי הרקורסיה בזמן $\Theta(n\cdot\log n)$. סה"כ:

$$T(n) = 2\cdot\Theta(n\cdot\log n) + \Theta(n) + \Theta(n) + \Theta(n\cdot\log n)$$
$$\Rightarrow T(n) \in \Theta(n\cdot\log n)$$

### 4.2 בעיית הסוכן הנוסע (Traveling Salesperson Problem)

**הבעיה:** בהנתן גרף שלם, לא מכוון ובעל משקלות לא-שליליים, יש למצוא את המעגל בעל העלות הנמוכה ביותר העובר בכל הצמתים. מניחים מתקיים אי-שוויון המשולש, כלומר עבור כל 3 צמתים $u,v,w$ מתקיים ש- $c(u,w) \le c(u,v) + c(v,w)$ (כאשר $c(u,v)$ היא עלות הקשת $(u,v)$).

**הרעיון הכללי:** פתרון מלא הוא בעיית np-complete. נציג אלגוריתם הערכה (approximation algorithm) המחזיר מסלול בעלות קטנה או שווה לפי 2 מהעלות האופטימלית הקיימת (אותה לא ניתן אף לחשב בזמן פולינומיאלי). המסלול האופטימלי הוא מעגלי, ולכן קל לראות שע"י הורדת קשת אחת מהמסלול האופטימלי נקבל עץ פורש (לאו דווקא מינימלי). אם כך, נמצא עץ פורש מינימלי, והמסלול שלנו יהיה מעבר preorder על העץ (כאשר מבצעים "קיצורי דרך" – כלומר לא חוזרים למעלה בעץ אם סיימנו תת-עץ אלא "קופצים" ישר לצומת הבא שיש לבקר בו. זה אפשרי מכיוון שהגרף הוא מלא!) כאשר מהצומת האחרון במסלול נחזור לצומת המקור. הנחת אי-שוויון המשולש מבטיחה שאורך המסלול לא יעלה על פי 2 מאורך המסלול האופטימלי.

**סיבוכיות:** קל לראות שהפעולה המשמעותית היא מציאת העץ הפורש המינימלי. מאחר והגרף צפוף ניתן לעשות זאת ע"י אלגוריתם Prim (ביצוג ע"י מערך!) וזמן הריצה יהיה:

$$\Theta(|V|^2)$$