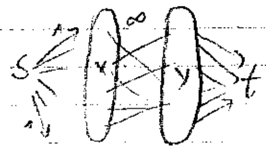
13.1.09 1

אלגוריתמים - שיעור 11

משפט Hall

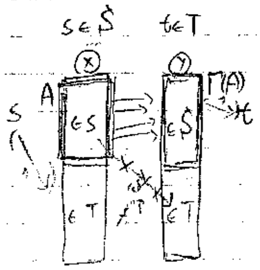
נתון גרף דו צדדי $V=X\cup Y$, $E\subseteq X\times Y$, $n=|X|=|Y|$.
לכל $A\subseteq X$ נסמן ב-$\Gamma(A)$ את קב' שכניה ב-Y. אז: קיים בגרף זיווג מושלם $\Leftrightarrow$ לכל A, $|\Gamma(A)|\geq|A|$

נניח שהתנאי מתקיים $|\Gamma(A)|\geq|A|$ לכל A ונניח בשלילה שאין זיווג מושלם. כלומר ערך הזרימה המקס' F ברשת המתאימה $< n$.



נסתכל על רשת הזרימה (המתאימה להגדרה) שקיבולי הקשתות ב-$E=\infty$
נסתכל בחתך שבו אין יותר מסלול משפר ברשת השיורית.

S היא קבוצת הצמתים הנגישים מ-s ברשת השיורית. ו-T הוא המשלים.



לפי ההוכחה של משפט MFMC, $C(S,T)\underset{\text{לפי MFMC}}{=}F\underset{\text{לפי הנחה}}{<}n$

$\Leftarrow$ $\underbrace{n-|A|+|\Gamma(A)|}_{\text{כמות הקשתות שחתך}}<n$ $\Leftarrow$ $|\Gamma(A)|<|A|$ וזו סתירה.

נמצאות ברשת השיורית, כלומר נמצא דרכן זרימה מלאה.

$A=S\cap X$

תכנות דינאמי Dynamic Programing

<u>הבהרה:</u> מקור השם אינו בהקשר של תכנות במדעי המחשב אלא לבעיות פתר-היסטוריות שבדבר ע"י מילוי של טבלה. את המושג נסביר ע"י שימוש של דוגמאות. לכן דוגמא היא מספר הטבל [illegible] שיטה ומבחן החוצה.

תחליף לגישה רקורסיבית לפתרון בעיה כאשר הפתרון הרקורסיבי יקר מדי, כי הוא נתקל שוב ושוב באותן תתי-בעיות (מבלי להרגיש) והוא שוב ושוב פותר אותן. [השיטה הזו מזהה את תתי הבעיות הללו ופותרת אותן אחת לאחת, כך שכל תת בעיה נפתרת פעם אחת בלבד].

<u>דוגמא ראשונה:</u> חישוב מספר פיבונאצ'י

<u>תזכורת:</u> הגדרה רקורסיבית של הסדרה: $F_0=F_1=1$, $F_n=F_{n-1}+F_{n-2}$

רקורסיה "בנאלית"

```
Proc F(n)
   if n≤1 then return 1;
   else return F(n-1)+F(n-2);
```

לרקורסיה זו זמן ריצה אקספוננציאלי: נחשב את זמן הריצה של קלט n - $T(n)$.

$$T(n)=\begin{cases}1 & n=0,1\\ 1+T(n-1)+T(n-2) & n\geq 2\end{cases} \Rightarrow T(n)\geq F_n\sim\left(\frac{1+\sqrt{5}}{2}\right)^n$$

F_{10}
F_9 F_8
F_8 F_7 F_7 F_6

בשיטה הרקורסיבית, אנו מחשבים את האיברים שוב ושוב בדרך של. בשיטה חליפית, נוכל לחשב תחילה את כל המספרים הדרושים לנו החל מהמספר הראשון עד F_n ולאחר מכן נשתמש בהם.

$F_2=F_0+F_1=2$
$F_3=F_2+F_1=3$
$F_4=\ldots$

בדרך כזו, נשפר את הסיבוכיות להיות לינארית. ניתן לתכנת פתרון כזה בשימוש overhead בתכנות הזו.

13.1.09 (2)

אלגוריתמים - שיעור 11

כפל מטריצות יעיל

נתונה סדרה של מטריצות $A_1, A_2, A_3, \dots, A_n$ לאו דווקא מטריצות ריבועיות, אך המטריצות בגודל מתאים כך שניתן לחשב את כפל המטריצות $A_1 \cdot A_2 \cdot A_3 \cdots A_n$. כלומר, נניח ש-$A_i$ היא מטריצה מגודל $\underset{\text{שורות}}{P_{i-1}} \times \underset{\text{עמודות}}{P_i}$. P_0 ו-P_n הם פרמטרים שאינם תלויים בסדר. תוצאת המכפלה של המטריצות, תהיה מטריצה מסדר $P_0 \times P_n$.

הרעיון נובע מכך שכפל מטריצות הוא אסוציאטיבי, מוסכם בדרך להתייחס לכפל של $A_1 \cdot A_2 \cdots A_n$ בצורה הבאה: $(((A_1 \cdot A_2) \cdot A_3) \cdots) A_n)$. אך ניתן לכפול אותן גם בסדר $(A_1 A_2)(A_3 A_4) \dots)$.

נזכר בנוסחה שלכפל מטריצות $C_{ij} = \sum_{k=1}^{q} A_{ik} B_{kj}$ $\quad 1 \le i \le p,\ 1 \le j \le r$ לכן סדר גודל של ההכפלה הוא $O(pqr)$.

<u>דוגמה:</u> נתון A_1, A_2, A_3 בגדלים $10\times100,\ 100\times5,\ 5\times50$ בהתאמה. כלומר $P_0=10,\ P_1=100,\ P_2=5,\ P_3=50$ (סה"כ 6 תלויות)

בהתחשב בסדר הכפלה:

$(A_1 A_2) A_3$ $\quad (A_1A_2): 10\times5$

$10\times100\times5 + 10\times5\times50$

~ 7500

$A_1 (A_2 A_3)$ $\quad (A_2A_3): 100\times50$

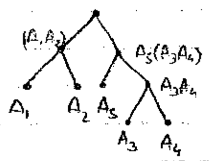
$100\times5\times50$

$10\times100\times50$

~ 75000

<u>המטרה:</u> למצוא סדר הכפלה עם מספר מינימלי של פעולות. כשמחיר כפל מידה של מטריצה $p\times q$ במטריצה $q\times r$ הוא pqr.

ייצוג של סדר הסוגריים, ניתן לבצע באמצעות עץ. לדוגמה, כפל של המטריצות A_1, A_2, A_3, A_4, A_5 בסדר $(A_1A_2)((A_3A_4)A_5)$ ניתן להציג כעץ ←



עץ ההכפלה של $(A_1A_2)((A_3A_4)A_5)$

(*) כדי להגיע לפתרון האופטימלי, נתחיל תחילה עם ההכפלה החיצונית, לאחר החלטה זו, נצטרך לפתור כל תת הכפלה בצורה אופטימלית בדרך דומה.

נסמן ב- $C[i,j]$ את המחיר המינימלי של כפל המטריצות $A_i \cdot A_{i+1} \cdots A_j$

$$C[i,j] = \min_{i\le k<j} \{ C[i,k] + C[k+1,j] + P_{i-1} P_k P_j \};\quad C[i,i]=0$$

$$\overbrace{\underset{\substack{P_{i-1}\\ \text{שורות}}}{A_i} \cdot A_{i+1} \cdot \dots \cdot \underset{\substack{P_k\\ \text{עמודות}}}{A_k}}^{P_{i-1}\times P_k}\ \Big|\ \overbrace{\underset{\substack{P_k\\ \text{שורות}}}{A_{k+1}} \cdots \underset{\substack{P_j\\ \text{טורים}}}{A_j}}^{P_k\times P_j}$$

(*) בעצם הגדרנו פרוצדורה הפועלת בצורה רקורסיבית "נאיבית" לחישוב סדר הכפלת המטריצות. נחשב את הפעולות של המימוש הנאיבי:

$T[i,j]$ וסימון להיות הפעולות של הביצוע הרקורסיבי. $\quad T[i,i]=1$

$$T^*(j-i) = T[i,j] = \sum_{k=i}^{j-1} \left(T[i,k] + T[k+1,j] + 1 \right)$$

נחשב את הפעולות כפונקציה של המרחק בין i ו-j:

$$T^*(0)=1 \ ;\quad T^*(m) = 2\left[T^*(n-1) + T^*(n-2) + \dots + T^*(0)\right] + m$$

13.1.09 (3)

אלגוריתמים - שיעור 11

<u>לדוגמא:</u>

$$T^*(4)= T^*(0)+T^*(3)+1 + T^*(1)+T^*(2)+1 + T^*(2)+T^*(1)+1 + T^*(3)+T^*(0)+1$$

<u>שיטת התכנות הדינאמי:</u>

(א) גישה זו טובה שיש תתי חישובים שאנו מחשבים שוב ושוב.

בסה"כ יש $\binom{n}{2}+n=O(n^2)$ תתי בעיות ונחשב כל אחת מהן פעם אחת בלבד.
פתרון $C[i,j]$ עם הפרש $l=j-i$ תלוי בתתי בעיות עם הפרש $< l$.
ולכן, נחשב בסדר עולה של $j-i$, החל מ- $j-i=0$ עד $j-i=n-1$.

(ב) נוכל לעשות זאת, לדוגמא, בעזרת האלגוריתם הבא:

```
for i=1 to n do
    C[i,i] ← 0      (0 הפרש)
for l=1 to n-1 do   (1 הפרש)
    for i=1 to n-l
        j ← i+l
        C[i,j] ← ∞
        for k=i to j-1 do
            t ← C[i,k] + C[k,j] + p_{i-1} p_k p_j
            if t < C[i,j] then
                C[i,j] ← t
                π[i,j] ← k
```

זמן ריצת האלגוריתם הוא $O(n^3)$.

פתרון רקורסיבי הוא top-down. תכנות דינאמי הוא bottom-up.
(א) כלומר, התכנות הרקורסיבי מתחיל בבעיה גדולה ומפרק אותה לבעיות קטנות יותר, והתכנות הדינאמי מתחיל בבעיות קטנות ומתקדם עד לפתרון הבעיה הגדולה.

<u>דוגמא שלישית:</u> תת סדרה משותפת מקסימלית
Longest Common Sequance (LCS)

(א) רוצים למצוא את תת הסדרה המשותפת המקסימלית בין הקלטים, לא בהכרח ברצף אך באותו סדר. לדוגמא:

$X = (A\ B\ C\ B\ D\ A\ B)$ $\quad m=|X|$

$Y = (B\ D\ C\ A\ B\ A)$ $\quad n=|Y|$

תתי הסדרות המשותפות המקסימליות הן BCBA, BDAB, BCAB.
לנו מספיק רק אחת מהן.

רוצים למצוא תת סדרה המופיעה גם ב-X וגם ב-Y, באותו סדר אך לאו דווקא ברצף עם מספר מקסימלי של איברים.

<u>אבחנה:</u> נסתכל ב-X_m וב-Y_n (האיברים האחרונים). נניח ש $X_m = Y_n$
(א) במקרה כזה, "כדאי" לוותר על האיבר האחרון. לכן, יש למצוא תת סדרה עד האיבר האחרון ואליה להוסיף בוודאות את איבר זה.

$X=(\underbrace{\ldots\ldots}) A \quad ; \quad Y=(\underbrace{\ldots\ldots}) A$

נמצא תת סדרה מקסימלית לה נוסיף בסוף A.

אם $X_m \neq Y_n$, נסמן כ- $X^{(i)}$ את הרישא (Prefix) של i התווים הראשונים של X ($X = X^{(m)}$) וכנ"ל עבור Y.

כדי לחשב את LCS נגדיר את האלגוריתם הבא:

```
LCS[X,Y]:
    if X_m = Y_m then return LCS[X^(m-1), Y^(n-1)] || (X_m)
    else
        return the longest of LCS(X^(m-1), Y), LCS(X, Y^(n-1))
```

שרשור

<u>תנאי עצירה:</u> $X=\emptyset$ או $Y=\emptyset$ (כלומר $m=0$ או $n=0$). במקרה כזה נחזיר את תת הסדרה הריקה.

<u>דוגמאת הרצה:</u> (נקצר ונרשום את אורך תת הסדרה הנלקחת מ-X ומ-Y).

$LCS[7,6] <$ $LCS[7,5]$ — $LCS[6,4] \| B$ ----

$LCS[6,6]$ — $LCS[5,5] \| A <$ ----

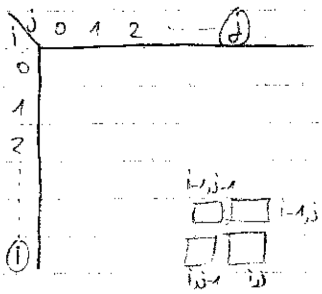
$LCS[4,5] - LCS[3,4] \| BA <$ $LCS[3,3]-(*)$

$(*) - LCS[2,2] \| CBA <$ ----

$LCS[2,1] - \underbrace{LCS[1,0]}_{\text{ריק}} \| BCBA$

/* גם הפעם נשים לב שמספר תתי הבעיות קטן מאוד שפתרנו בפועל. */

מספר תתי הבעיות הוא mn $(m+1)(n+1)$.



אם כן נגדיר טבלה של ערכי $LCS[i,j]$

/* אנו מחשבים את התא או באלכסון פנימה ↖ או גם שמאלה וגם כלפי מעלה ↑←. האלכסון כאשר התו האחרון זהה, והשני כאשר התווים אינם זהים. כדי לחשב את הטבלה בצורה של הערכים והיו צמודים כשנזדקק להם, נחשב שורה אחר שורה משמאל לימין */

```
        for j=0 to n do
            LCS[0,j] <- Ø
(שורה)  for i=0 to m do
            LCS[i,0] <- Ø
            for j=1 to n do
                if X_i = Y_j then
                    LCS[i,j] <- LCS[i-1,j-1] || (X_i)     <- (* כאן אנו מוסיפים שרשור ללא באלכסון ואת התו הנוכחי; *)
                else
                    LCS[i,j] <- max { LCS[i,j-1], LCS[i-1,j] }
```

במקום לחשב את LCS מפורש נחשב רק את אורכה (מספרים שלמים) ואם מצביעים לאיבר האחרון שלה וכן למקום הקודם בטבלה שממנו הגענו.

בשלב זו, זמן ריצת האלגוריתם הוא $O(mn)$.

13.1.09

5

אלגוריתמים - שיעור 11

<u>דוגמה:</u>

נתונים מספרים טבעיים $a_1, a_2, \dots, a_n, b$, $0<a_i$, $b\geq 0$
ורוצים למצוא $\sum_{i=1}^{n} a_i x_i = b$ כאשר x_i הוא 0 או 1.

הבעיה יכולה להיות: "האם יש פתרון?" או "מספר הפתרונות האפשריים".

זו בעיה קשה במובן הבא: אם מחפשים פתרון שזמן הריצה שלו תלוי ב-n בלבד ולא בגודל המספרים a_i ו-b, אז לא ידוע פתרון בזמן פולינומיאלי. כלומר זו בעיה NP-שלמה.

הפתרון הבא נותן $M=\sum_{i=1}^{n} a_i$, ונראה למצוא אלג' ועוד שזמן הריצה שלו תלוי ב-n וב-M.

נניח ש $b \leq M$ כי אחרת אין פתרון. ננסח את הפתרון ע"י תכנות דינאמי באופן הבא:

(סתם) לכל תת בע"ת עם 2 פרמטרים k,b: $\sum_{i=1}^{k} a_i x_i = b$
כאשר $k=1,\dots,n$ $b=0,\dots,M-1$ $(x_i \in \{0,1\})$.

אנו רוצים לפתור בעית k,b באמצעות בע"ת $k-1,?$. כלומר:

$$(\text{= כמה פתרונות קיימים})\ C[k,b] := \underbrace{C[k-1,b]}_{x_k=0 \text{ אם}} + \underbrace{C[k-1,b-a_k]}_{\substack{x_k=1 \text{ אם (ורק) } \\ b\geq a_k}} \quad ; \quad C[k,0]=\underset{\substack{\text{הפתרון} \\ x_1=x_2=\dots=x_k=0}}{1}$$

פתרון זה יביא בזמן $O(mn)$.