13/1/08

אלגוריתמים - שיעור 11

מאחר שבשיעור האחרון קצת מיהרנו, נחזור בקצרה על חלקו האחרון.

משפט Hall

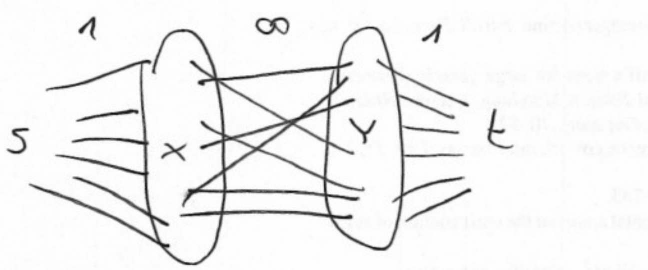
נתון גרף דו"צ $V = X \cup Y$, $E \subseteq X \times Y$, $|X|=|Y|=n$.
לכל $A \subseteq X$ נסמן ב-$\Gamma(A)$ את קב' השכנים ב-Y. אז קיים בגרף זיווג מושלם אם ורק אם לכל A: $|\Gamma(A)| \geq |A|$.

הוכחה

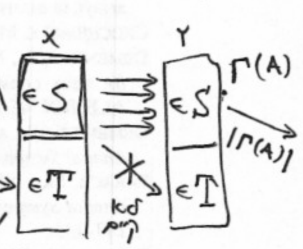
נניח שמתקיים התנאי $|\Gamma(A)| \geq |A|$ לכל A, ונניח בשלילה שאין זיווג מושלם. כלומר, ערך הזרימה המקסימלית F ברשת המתאימה $< n$.
נסתכל על רשת הזרימה המתאימה המקבלת שקיבולי קשתות $E = \infty$.
נסתכל בשלב שבו אין כבר מסלול משפר ברשת השיורית.

S = קב' הצמתים הנגישים מ-s ברשת השיורית.

T = המשלים.



$s \in S$, $t \in T$



נראה כי $A = S \cap X$ לא עונה לתנאי שהנחנו.

ראשית, נבחין שאין קשת מ-A ל-$Y \cap T$, לפי הגדרות S, T.
לכן, כל הקשתות מ-A הן ל-$Y \cap S$. כמו כן, כל איבר מ-$Y \cap S$ שכן של איבר מ-A (מדוע?). לכן $\Gamma(A) = S \cap Y$.

לפי ההוכחה של משפט ה-MFMC, $c(S,T) = F < n$.
מצד שני, $c(S,T) = n - |A| + |\Gamma(A)|$. מכאן:

$$n - |A| + |\Gamma(A)| < n \Rightarrow |\Gamma(A)| < |A|$$

בסתירה להנחה. □

והנה סיימנו את נושא בעיות הזרימה, ואנו יכולים לעבור לנושא חדש ומלהיב!

תכנות דינאמי Dynamic Programming

(הערה סמנטית: בהקשר זה, "תכנות" איננו כתיבת תוכנה, וגם המילה "דינאמי" קצת מטעה. כדי להבין מה זה, ניתן הגדרה לא-מדוייקת, ואחריה מספר דוגמאות)

תכנות דינאמי הוא תחליף לגישה רקורסיבית לפתרון בעיות, כאשר הפתרון הרקורסיבי יקר מדי, כי הוא נתקל שוב ושוב באותן תת-בעיות (מבלי להרגיש) והוא שוב ושוב פותר אותן.

דוגמה: חישוב מספרי פיבונאצ'י ("ספרים וסרטים נכתבו על הסדרה המסכנה הזו").

ניתן לחשב את הסדרה ברקורסיה כך:

```
proc F(n):
  if n≤1 then
    return 1
  else
    return F(n-1)+F(n-2)
```

זמן החישוב הנדרש $T(n)$ הוא:

$$T(n) = \begin{cases} 1 & n \le 1 \\ 1 + T(n-1) + T(n-2) & \text{else} \end{cases}$$

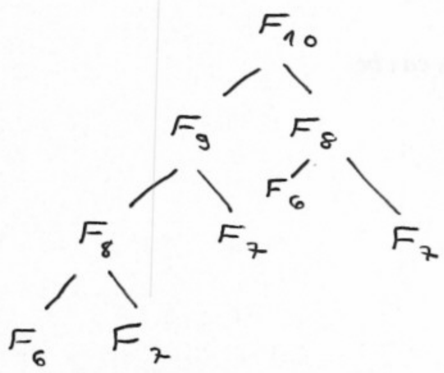
לכן: $T(n) \ge F_n \sim \left(\frac{1+\sqrt{5}}{2}\right)^n$

כלומר, הסיבוכיות אקספוננציאלית, שזה איום ונורא!

במקום זה, אפשר פשוט לחשב את $F_0 \dots F_n$ תוך הסתמכות על תוצאות שכבר חושבו!!

$F_0 = 1,\ F_1 = 1,\ F_2 = F_1 + F_0 = 2,\ F_3 = F_2 + F_1 = 3, \dots$

כך, הסיבוכיות תהיה לינארית!



על החישוב של הרקורסיה. שימו לב כמה פעמים אנו מחשבים את F_6, F_7.

בואו נראה דוגמה נוספת, מעט יותר מייגעת:

כפל מטריצות יעיל

נתונה סדרה של מטריצות, לאו-דווקא ריבועיות:

אנחנו רוצים לחשב את $A_1A_2\cdot\ldots\cdot A_n$ — $A_1, A_2, \ldots, A_n$

נתונה סדרת מספרים $p_0, p_1, \ldots, p_n$ (הנחה: גדלי המטריצות "מסתדרים"), כך שגודל A_i הוא $p_{i-1}\times p_i$.

שימו לב, שבגלל האסוציאטיביות של כפל מטריצות, יש דרכים רבות לבצע את הכפל:

$$((((A_1\cdot A_2)A_3)A_4)A_5) = (A_1A_2)(A_3(A_4A_5))$$

נשים לב, שהכפלת מטריצה בגודל $p\times q$ במטריצה בגודל $q\times r$, דורשת זמן $O(pqr)$.

כיצד נכפיל את הסדרה הבאה?

$$\underset{10\times 100}{A_1} \cdot \underset{100\times 5}{A_2} \cdot \underset{5\times 50}{A_3}$$

נבחן כמה עולה כל דרך:

$$\underbrace{(A_1A_2)}_{10\times 5}A_3$$

$10\times 100\times 5$
$+\ 10\times 5\times 50$
$= 7500$

$$A_1\overbrace{(A_2A_3)}^{100\times 50}$$

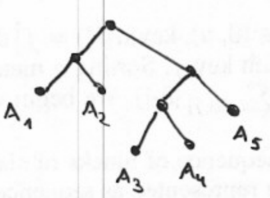
$100\times 5\times 50$
$+\ 10\times 100\times 50$
$= 75000$

מסקנה: כדאי שנתכנן באופן יעיל את סדר הגדלים שבו אנו מכפילים.

המטרה: למצוא סדר הכפלה עם מספר מינימלי של פעולות, כשמחיר כפל בודד של מטריצת $p\times q$ במטריצת $q\times r$ הוא בדיוק pqr.

ניתן לכתוב כל סדר הכפלה כעץ. למשל:

$(A_1A_2)((A_3A_4)A_5) \longrightarrow$



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

אזי נקבל את הרקורסיה:

$$C[i,j] = \min_{i\le k<j} C[i,k] + C[k+1,j] + p_{i-1}p_kp_j$$

$$C[i,i] = 0$$

יהא $T[i,j]$ עלות הביצוע הרקורסיבי:

$$T[i,i] = 1$$

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

נשים שהנוסחה האחרונה תלויה למעשה רק בהפרש j-i, לכן נסמן מעתה $T^*(j-i) = T[i,j]$. אזי מתקיים:

$$T^*(0) = 1$$

$$T^*(m) = 2\left[T^*(m-1) + T^*(m-2) + \ldots + T^*(0)\right] + m$$

למשל:

$$\begin{aligned} T^*(4) = \ & T^*(0) + T^*(3) + 1 \\ & + T^*(1) + T^*(2) + 1 \\ & + T^*(2) + T^*(1) + 1 \\ & + T^*(3) + T^*(0) + 1 \end{aligned}$$

ניתן להראות כי $T^*(m) \sim 3^m$.

נשים לב שיש בסה"כ $\binom{n}{2} + n = O(n^2)$ תת-בעיות מהצורה $C[i,j]$, ונחשב כ"א פעם אחת. $C[i,j]$ עם $j-i = l$ תלויה אך ורק בתת-בעיות עם הפרש $< l$. לכן נחשבן בסדר עולה, החל מ- $j-i=0$ עד $j-i=n-1$.

```
for i = 1 to n do
  C[i,i] ← 0
for l = 1 to n-1 do
  for i = 1 to n-l do
    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
```

(הפעולה האחרונה נועדה כדי שנדע באיזה סדר לכפול את המטריצות, ולא רק את העלות המינימלית).

זמן הריצה של האלגוריתם: $O(n^3)$.

"פתרון רקורסיבי הוא top-down,
תכנות דינאמי הוא bottom-up."

אפשר לכתוב את האלגוריתם באופן רקורסיבי, תוך שימוש ב-lookup table שבו נבדוק בכל פעם האם כבר פתרנו את התת-בעיה הנדרשת.

רוצים עוד בעיה שעדיף לפתור בתכנות דינאמי? בבקשה!

Longest Common Subsequence — תת-סדרה משותפת מקסימלית

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

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

(BCAB) היא תת-סדרה באורך 4.

גם (BDAB) ו- (BCBA) הן סדרות כאלה (אתם יכולים לבדוק).

<u>המטרה:</u> למצוא תת-סדרה <u>אחת</u>, לאו דווקא רצופה, שמופיעה גם ב-X וגם ב-Y באותו סדר, עם מספר מקסימלי של איברים.

נסתכל ב-x_m ו-y_n האיברים האחרונים בסדרות.

נניח $x_m = y_n$, מספיק למצוא את התת-סדרה המשותפת המקסימלית מתוך הסדרות שעשינו להן "קפד-זנבו", ונשרשר לה את x_m.

לעומת זאת, אם $x_m \neq y_n$ ברור כי ה-LCS לא מסתיימת בתו האחרון ב-X וגם ב-Y. לכן נפצל את הבעיה לשני תת-בעיות, כשבכל פעם נעשה "קפד-זנבה" של סדרה אחרת.

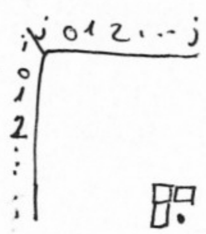
נסמן ב-$X^{(i)}$ את הרישא (prefix) של i האיברים הראשונים של X, וכנ"ל ל-Y.

```
LCS[X,Y]:
  if x_m = y_n then                          שרשור
     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)]
```

תנאי עצירה: $X = \emptyset$ או $Y = \emptyset$, אז נחזיר תת סדרה ריקה.

(אתם מוזמנים לעשות דוגמת הרצה על הסדרות שנתונות בתחילת הדף).

מספר התת-בעיות מהצורה $LCS[X^{(i)}, Y^{(j)}]$ (שניתן לכתוב גם בקיצור $LCS[i,j]$) היא $(m+1)(n+1)$. אם נבנה טבלה של ערכי $LCS[i,j]$ ונחשב אותה שורה אחרי שורה משמאל לימין, החישוב של כל תא בטבלה ייקח זמן קבוע.



הערך שמסומן בנקודה, תלוי בערכים שמסומנים בריבוע.

פסאודו-קוד:

```
for j = 0 to n do
  LCS[0,j] ← ∅
for i = 1 to m do
  LCS[i,0] ← ∅
  for j = 1 to n do
    if x_i = y_j then
      LCS[i,j] ← LCS[i-1, j-1] || (x_i)
    else
      LCS[i,j] ← Longest{LCS[i,j-1], LCS[i-1,j]}
```

במקום לחשב את LCS מפורשות, נחשב רק את אורכה וגם מצביע לאבר האחרון שלה, וכן למקום הקודם בטבלה שבעזרתו היא נבנתה.

זמן הריצה: $O(mn)$

טוב, כמה דוגמאות אפשר כבר לתת? לפחות עוד אחת:

נתונים מספרים טבעיים $a_1, a_2, \dots, a_n, b$ $a_i > 0,\ b \ge 0$

אנו רוצים לפתור את המשוואה $\sum_{i=1}^{n} a_i x_i = b$ כאשר לכל i, $x_i \in \{0,1\}$.

למעשה, אנחנו רוצים לקבוע האם יש פתרון, ואם אפשר, אפילו למצוא את הפתרונות.

אם מחפשים פתרון שזמן הריצה שלו תלוי ב-n בלבד ולא בגודל המספרים a_i, b, לא ידוע פתרון פולינומיאלי.

בעיה זו (Subset-Sum) היא למעשה NP-שלמה.

אבל! נניח שנתון $M = \sum_{i=1}^{n} a_i$. אנחנו רוצים למצוא אלגוריתם "יעיל" שזמן הריצה שלו תלוי ב-M, n.

קל לראות כי $b \le M$.

נסתכל על תת-בעיות עם 2 פרמטרים k, b: $\sum_{i=1}^{k} a_i x_i = b$ $x_i \in \{0,1\}$.

נסמן ב- $C[k,b]$ את מס' הפתרונות עם הפרמטרים k, b.

מהו $C[k,b]$? אם $x_i = 0$, $C[k-1, b]$.

אחרת, $x_i = 1$, $C[k-1, b-a_k]$.

לכן, אם $b \ge a_k$ אז $C[k,b] := C[k-1,b]$.

אחרת, $C[k,b] := C[k-1,b] + C[k-1, b-a_k]$

$C[k,0] := 1$ (הפתרון $x_i = 0$ לכל