דפי נוסחאות למבחן בתכנון וניתוח אלגוריתמים

סימפלקס:

חישוב שורת המחירים המתוקנים

$$C'_j = \sum_{i=1}^{m} a_{ij} C_{B_i} - C_j$$

חישוב עמודת החסמים העליונים

$$\frac{b_i}{a_{ik}}$$

חישוב המקדמים המתוקנים

$$a'_{ij} = a_{ij} - \frac{a_{ik} \cdot a_{rj}}{a_{rk}}$$

כללים שראינו בסימפלכס בעבור הדוגמה שלהלן:

$$\text{Min } \{Z = 20x_1 + 12x_2\}$$

S.t.

$$\begin{array}{lllllll}
2x_1 & & + s_1 & & & = & 8 \\
2x_1 & + 4x_2 & & + s_2 & & = & 14 \\
x_1 & + 4x_2 & & & + s_3 & = & 12
\end{array}$$

$$x_j \geq 0 \qquad j = 1, 2$$

$$s_j \geq 0 \qquad j = 1, 2, 3$$

הטבלה האחרונה בעבור הבעיה הנתונה היא:

		Z	x_1	x_2	s_1	s_2	s_3			b	$\frac{b_i}{a_{ik}}$
	Z	1	0	0	7	3	0			98	
1	x_1	0	1	0	0.5	0	0			4	
2	x_2	0	0	1	–0.25	0.25	0			1.5	
3	s_3	0	0	0	0.5	–1	1			2	
	א	יוצ׳									

מטריצת הבסיס המתאימה לבסיס $\{x_1, x_2, s_3\}$ **של הטבלה האחרונה היא:**

$$B = \begin{pmatrix} 2 & 0 & 0 \\ 2 & 4 & 0 \\ 1 & 4 & 1 \end{pmatrix}$$

המטריצה ההופכית של מטריצה זו היא:

$$B^{-1} = \begin{pmatrix} 0.5 & 0 & 0 \\ -0.25 & 0.25 & 0 \\ 0.5 & -1 & 1 \end{pmatrix}$$

והיא נמצאת מתחת למשתני החוסר בשורות האילוצים.

ו- $x_B = B^{-1} \cdot b$ **ובעבור בעיה דואלית** $\underline{Y^T} = \underline{c_B} \cdot B^{-1}$

<u>משפט הדואליות החלש</u>

נתונות בעיה פרימאלית ודואלית:

<u>פרימאלית</u> (P)	<u>דואלית</u> (D)
$Max\{Z = \underline{C}^T \underline{X}\}$	$Min\{V = \underline{b}^T \underline{Y}\}$
$A\underline{X} \le b$	$A^T Y \ge C$
$\underline{X} \ge 0$	$\underline{Y} \ge 0$

לכל פתרון אפשרי x של (P) ולכל פתרון אפשרי y של (D) מתקיים:

$$c^T x \le b^T y$$

<u>משפט הדואליות החזק</u>

נתונות בעיה פרימאלית ודואלית:

<u>פרימאלית</u> (P)	<u>דואלית</u> (D)
$Max\{Z = \underline{C}^T \underline{X}\}$	$Min\{V = \underline{b}^T \underline{Y}\}$
$A\underline{X} \le b$	$A^T Y \ge C$
$\underline{X} \ge 0$	$\underline{Y} \ge 0$

עבור פתרון אפשרי x של (P) ופתרון אפשרי y של (D) מתקיים כי: $c^T x = b^T y$ אם ורק אם x ו- y פתרונות אופטימליים לבעיותיהם.

בלת הסימפלכס הראשונה נראית כך :

ערך פונק' המטרה		**ערכי מקדמי c**
0	**0**	$-C^T$
b	I	A

טבלת הסימפלכס האחרונה תראה כך:

ערך פונק' המטרה		**ערכי מקדמי c**
$C_B^T B^{-1} b$	$C_B^T B^{-1}$	$-C^T + C_B^T B^{-1} A$
$B^{-1} b$	B^{-1}	$B^{-1} A$
ערכי המשתנים הבסיסיים		

הגדרות על גרפים:

בגרף לא מכוון $\sum_{v \in V} d(v) = 2|E|$

בגרף מכוון $\sum_{v \in V} d_{in}(v) = \sum_{v \in V} d_{out}(v) = |E|$

גרף מלא – גרף בו בין כל זוג צמתים קיימת קשת.

גרף קשיר – גרף בו כל צומת נגיש מכל צומת אחר.

רכיב קשירות – תת-גרף קשיר ולא ניתן להרחבה תוך שמירת קשירותו.

עץ – גרף קשיר וחסר מעגלים פשוטים.

מסלול פשוט – מסלול העובר בכל קשת פעם אחת לכל היותר.

מסלול אלמנטרי – מסלול העובר בכל צומת פעם אחת לכל היותר.

עלה – צומת בעל דרגה 1 בדיוק.

```
BFS(G, s)
for each vertex u ∈ V [G] - {s}
   do color[u] ← WHITE
      d[u] ← ∞
      π[u] ← NIL
color[s] ← GRAY
d[s] ← 0
π[s] ← NIL
Q ← Ø
ENQUEUE(Q, s)
while Q ≠ Ø
   do u ← DEQUEUE(Q)
      for each v ∈ Adj[u]
         do if color[v] = WHITE
               then color[v] ← GRAY
                    d[v] ← d[u] + 1
                    π[v] ← u
                    ENQUEUE(Q, v)
      color[u] ← BLACK
```

```
DFS(G)
for each vertex u ∈ V
   do color[u] ← WHITE
      π[u] ← NIL
time ← 0
for each vertex u ∈ V
   do if color[u] = WHITE
         then DFS-VISIT(u)
```

```
DFS-VISIT(u)
color[u] ← GRAY
d[u] ← time ← time + 1
for each v ∈ Adj[u]
   do if color[v] = WHITE
         then π[v] ← u
              DFS-VISIT(v)
color[u] ← BLACK
f[u] ← time ← time + 1
```

להלן השגרה המשוכתבת DFS_Visit(u)

1. $color[u] \leftarrow$ אפור
2. $time \leftarrow time+1$
3. $d[u] \leftarrow time$
4. עבור כל קודקוד v שהינו קודקוד סמוך ל- u
   בצע :
   בדוק את הצבע של הקודקוד v :
   4.1 אם color[v] הוא צבע לבן
   אז בצע : 4.1.1 סמן את הקשת (u,v) כ"קשת עץ" (tree_edge).
   4.1.2 $p[v] \leftarrow u$
   4.1.3 DFS_Visit(v)
   4.2 אם color[v] הוא צבע אפור
   אז בצע : סמן את הקשת (u,v) כ"קשת אחורית " (back_edge)
   .
   4.3 אם color[v] הוא צבע שחור
   אז בצע :
   אם $d[v] > d[u]$
   אז בצע :סמן את הקשת (u,v) כ"קשת קדימה"
   אחרת בצע :סמן את הקשת (u,v) כ"קשת חוצה"
5. $color[u] \leftarrow$ שחור
6. $time \leftarrow time+1$
7. $f[u] \leftarrow time$

<u>אלגוריתם למציאת רכיבי קשירות חזקה בגרפים מכוונים</u>

Strong Connected Component (SCC)

צעד 1. מריצים DFS(G) ויוצרים רשימת קדקודים (L) אשר ממוינת בסדר יורד לפי זמני סיום הטיפול בהם.

צעד 2. הופכים את הגרף G=(V,E) ומקבלים $G^T=(V,E^T)$

כאשר $E^T = \{ (U,V) \mid (V,U) \in E \}$,כלומר הופכים את קשתות הגרף .

צעד 3 . מריצים $DFS(G^T)$, כך שהלולאה המרכזית של DFS עוברת על קודקודי הגרף לפי הסדר , כפי שנקבע בצעד 1 ברשימה L .

<u>מיון טופולוגי - TP_Sort (Graph G)</u>

1. קרא לשגרה DFS(G) . השגרה DFS מחשבת את ה- f[v] , המציין את מועד סיום הטיפול בקדקוד v , עבור כל קדקוד v .
   כאשר הטיפול בקדקוד v מסתיים אז נשים אותו בחזית הרשימה המייצגת את המיון הטופולוגי.
2. הדפס את הרשימה המייצגת את המיון הטופולוגי.

```
INITIALIZE-SINGLE-SOURCE(G, s)
    for each vertex v ∈ V
        do d[u] ← ∞
            π[v] ← NIL
    d[s] ← 0
```

```
RELAX(u, v, w)
    if d[v] > d[u] + w(u, v)
        then d[v] ← d[u] + w(u, v)
                π[v] ← u
```

```
DIJKSTRA(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    S ← Ø
    Q ← V
    while Q ≠ Ø
        do u ← EXTRACT-MIN(Q)
            S ← S ∪ {u}
            for each vertex v ∈ Adj[u]
                do RELAX(u, v, w)
```

```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i ← 1 to |V| - 1
        do for each edge (u, v) ∈ E
                do RELAX(u, v, w)
    for each edge (u, v) ∈ E
        do if d[v] > d[u] + w(u, v)
                then return FALSE
    return TRUE
```

```
MST-KRUSKAL(G, w)
A ← Ø
for each vertex v ∈ V
   do MAKE-SET(v)
sort E into nondecreasing order by weight w
for each edge (u, v) ∈ E, by nondecreasing weight order
   do if FIND-SET(u) ≠ FIND-SET(v)
         then A ← A ∪ {(u, v)}
               UNION(u, v)
return A
```

```
MST-PRIM(G, w, r)
for each u ∈ V
   do key[u] ← ∞
      π[u] ← NIL
key[r] ← 0
Q ← V
while Q ≠ Ø
   do u ← EXTRACT-MIN(Q)
      for each v ∈ Adj[u]
         do if v ∈ Q and w(u, v) < key[v]
               then π[v] ← u
                     key[v] ← w(u, v)
```

<u>אוילר והמילטון</u>

- מסלול אוילר הוא מסלול בגרף (מכוון/לא מכוון) העובר דרך כל קשת בגרף בדיוק פעם אחת
- מעגל אוילר – הוא מעגל בגרף מכוון/לא מכוון העובר דרך כל קשת בגרף בדיוק פעם אחת
- **<u>משפט</u>**

נתון גרף G ובו בדיוק 2k קדקודים בעלי דרגה אי זוגית, k>=1 ומספר טבעי.
אפשר לצייר את G ב- k משיכות קולמוס ואי אפשר לצייר את G בפחות מ- k משיכות קולמוס.

- <u>מסלול / מעגל המילטון - Hamiltonian circuit</u>

מעגל (מסלול) המילטון מוגדר כמעגל (מסלול) פשוט העובר דרך כל קדקודי גרף נתון פעם אחת בדיוק.

- **<u>משפט:</u>**

נתון גרף פשוט וקשיר המילטוני,

אזי מספר הרכיבים ב-G* המתקבל מגרף- G על ידי השמטת k>0 קדקודים (וכל הקשתות הנוגעות בהם) אינו עולה על k . (**התנאי ההכרחי** לגרף המילטוני).

- **<u>משפט דירק</u>** (Dirac)

נתון גרף פשוט G בעל n>=3 קדקודים . אם הדרגה של כל קדקוד היא לפחות n/2

אז הגרף G המילטוני. (התנאי הוא ***תנאי מספיק*** ע"מ ש-G יהיה המילטוני אך אין הוא הכרחי).

אלגוריתם למציאת מסלול (מעגל) אוילר

<u>קלט:</u> גרף אוילרי לא מכוון $G(V,E)$, וצמת קצה של המסלול (המעגל) האוילרי ב-G: a.

<u>פלט:</u> רשימה מקושרת של קשתות P המתארת מסלול (מעגל) אוילר בגרף.

<u>מבני הנתונים:</u>

1. מערך של צמתים, ולכל צומת, v:
   - רשימת הקשתות הפוגעות בצומת, בסוף הרשימה NIL.
   - סימון האם הצומת "חדש" או "ישן". בתחילה כל הצמתים "חדשים".
   - מחוון $N(v)$ המצביע על הקשת הראשונה שעוד לא נסרקה ברשימת הקשתות של v. בתחילה מצביע על הקשת הראשונה ברשימה (או על NIL אם הרשימה ריקה).
   - מחוון $E(v)$ המצביע על הקשת במסלול שדרכה v התגלה לראשונה. בתחילה $E(v)$ אינו מוגדר.
2. מערך של קשתות, ולכל קשת, e:
   - שני צמתי הקצה של e.
   - דגל הקובע האם הקשת "פנויה" או "תפוסה". בתחילה כל הקשתות "פנויות".
3. שתי רשימות מקושרות של קשתות, P,Q, לאחסון מסלולים. בהתחלה P,Q ריקות.
4. תור L של צמתים "ישנים". בהתחלה L ריק. (כל צמת יכנס ל-L רק פעם אחת, כאשר יתגלה. L יכול להיות גם מחסנית או כל מאגר אחר התומך ב"כניסה" ו"שליפה").

<u>האלגוריתם:</u> (ליצירת מסלול אוילרי החל מ-a.)

(1) • סמן את a "ישן" והכנס אותו ל-L.

(2) • הגדר את $E(a)$ כסמל ההתחלה של P.

(3) • <u>בצע</u> סרוק (a, P).

(4) • <u>כל עוד</u> L אינו ריק <u>בצע</u> :

(5) ∘ הוצא צומת u מ-L.

(6) ∘ <u>בצע</u> סרוק (u, P'). (P' תכיל "טיול" מ-u ל-u. יתכן ש P' ריקה מקשתות.)

(7) ∘ צרף את P' לתוך P החל מ-$E(u)$.

<u>נהל סרוק (x, Q):</u>

(למציאת מסלול מקסימלי (שאי אפשר להמשיך בו) המורכב רק מקשתות חדשות, החל מצומת x; בסיום הנהל Q יכיל את רשימת קשתות המסלול.)

(1) • אתחל את הרשימה Q לריקה

(2) • $y \leftarrow x$

(3) • <u>כל עוד</u> $N(y)$ מצביע על קשת (ולא על NIL), <u>בצע</u> :

(4) ∘ <u>אם</u> $N(y)$ היא קשת "תפוסה" <u>אזי</u>

(5) ⋆ קדם את $N(y)$ להצביע על הקשת הבאה ברשימת הקשתות של y.

(6) ∘ <u>אחרת</u> <u>בצע</u> :

(7) ⋆ $e \leftarrow N(y)$

(8) ⋆ סמן את e "תפוסה".

(9) ⋆ הוסף את e לסוף הרשימה Q.

(10) ⋆ בעזרת טבלת הקשתות מצא את צומת הקצה השני של e, נניח z.

(11) ⋆ <u>אם</u> z צומת "חדש" <u>אזי</u>

(12) ◇ $E(z) \leftarrow e$

(13) ◇ הכנס את z ל-L וסמן ש-z צמת "ישן".

(14) ⋆ $y \leftarrow z$.

תכנות דינמי – כפל מטריצות

```
MATRIX-CHAIN-ORDER(P)
n ← length[p] - 1
for i ← 1 to n do m[i, i] ← 0
for L ← 2 to n
    do for i ← 1 to n - L + 1
        do j ← i + L-1
           m[i, j] ←
           for k ← i to j - 1
               do q= m[i, k] + m[k + 1, j] +
                  if q < m[i, j]
                     then m[i, j] ← q
                          s[i, j] ← k
return m and s
```

בניית פתרון אופטימלי

```
MATRIX-CHAIN-MULTIPLY(A, s, i ,j)
if j >i
   then X ← MATRIX-CHAIN-MULTIPLY(A, s, i, s[i, j])
        Y ← MATRIX-CHAIN-MULTIPLY(A, s, s[i, j] + 1, j)
        return MATRIX-MULTIPLY(X, Y)
else return A[i]
```

אלגוריתם למציאת האורך של תמ"א

LCS-LENGTH (X, Y)

```
m ← length [X]
n ← length [Y]
for i ← 1 to m  do
   c[i, 0] ← 0
for j ← 0 to n  do
   c[0, j] ← 0
for i ← 1 to m  do
   for j ← 1 to n  do
      if  (xi = yj)  then
         c[i, j] ← c[i-1, j-1] + 1
         b[i, j] ← "        "
      else if  (c[i-1, j] ≥ c[i, j-1])  then
         c[i, j] ← c[i-1, j]
         b[i, j] ← "↑"
      else
         c[i, j] ← c[i, j-1]
         b[i, j] ← "←"
return  c and b
```

בניית תמ"א:

```
PRINT-LCS (b, X, i, j)
  if  (i = 0) or (j =0)  then  return;
  if  (b[i, j] = "     ")  then
        PRINT-LCS (b, X, i-1, j-1)
        print Xi
  else
        if  (b[i, j] = " ↑ ")  then
              PRINT-LCS (b, X, i-1, j)
        else  PRINT-LCS (b, X, i, j-1)
```