

מבני נתונים ואלגוריתמים – תרגול #5

אלגוריתמי מיון

מיונים שאינם מיוני השוואה:

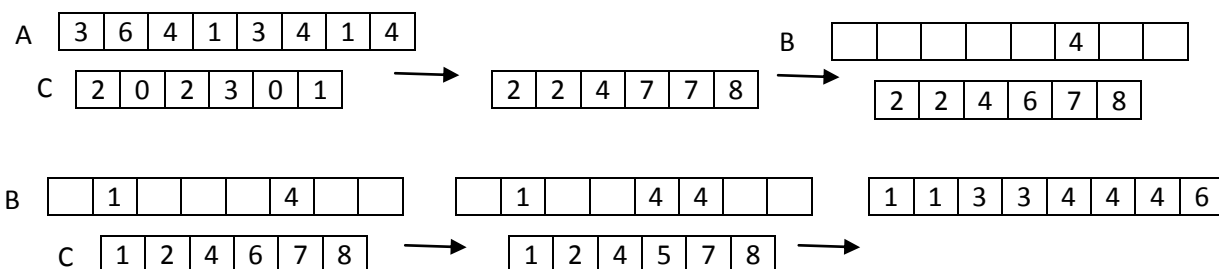
- (1) *Counting sort*: מניחים שכל איברי הקלט הם מספרים בתחום מ-1 עד k , עבור k שלם כלשהו. כאשר $k=O(n)$, הסיבוכיות היא $O(n)$. הרעיון – סופרים כמה ערכים במערך הקלט קטנים או שווים לכל ערך מ-1 עד k , ואז מכניסים את האיברי הקלט למערך פלט ישר למקום הנכון. המיון יציב.
- (2) *Bucket sort*: גם כאן יש הנחה על הקלט – איברי הקלט מגיעים מהתפלגות אחידה בקטע מסוים (לדוג' - $(0,1)$). מחלקים את הקטע ל- n דליים ומפזרים את הערכים בכל דלי. בגלל שהם מתגעים מהתפלגות אחידה – מצפים שהחלוקה תהיה בערך שווה בין הדליים. ממיינים את המספרים בכל דלי ועוברים על כל הדליים כדי לקבל את הפלט. תוחלת זמן הריצה היא $O(n)$.
- (3) *Radix sort*: ממיינים לפי הספרה הכי פחות משמעותית (*Least significant digit*) (באמצעות מיון יציב כגון *counting sort*) ואז לפי הספרה הלפני האחרונה וכך הלאה. סיבוכיות – $O(\log n)$. יש גם *Most significant digit* (הפוך).

לפניכם פסאודו קוד של *Counting sort*:

Counting sort (A, B, k) :

```
// A – input array (size n), B- output array (size n), k – range of values (1-k). C – array of size k
for i=1 to k
    C[i] = 0;
for j=1 to length(A)
    C[A[j]] = C[A[j]]+1;
// C[i] contains the number of elements equal to i
for i=2 to k
    C[i] = C[i] + C[i-1];
// C[i] contains the number of elements less than or equal to i
for j=length(A) downto 1
    B[C[A[j]]] = A[j];
    C[A[j]] = C[A[j]] – 1
// reduce by 1 for equal values (the next equal value will be inserted in the previous slot)
```

דוגמאת ריצה:



תרגיל:

ניח שמשנים את לולאת ה-for האחרונה באופן הבא: for j=1 to length(A). האם לאחר השינוי האלגוריתם עדיין עובד? האם הוא יציב?

פיתרון: האלגוריתם עדיין עובד אך לא יציב, מכיוון שכאשר יש 2 ערכים זהים, והכנסו כבר את הראשון, הבא יוכנס לפניו. לכן אם הלולאה פועלת בסדר עולה במערך איבר "מאוחר" יותר יוכנס לפני איבר "מוקדם" יותר.

תרגיל:

בעיית הבחירה – Select:

נתונה רשימה L באורך n. רוצים למצוא את האיבר ה-i בגודלו.

פיתרון 1:

- מיין את האיברים ב- $O(n \log n)$
- החזר את האיבר ה-i.

סיבוכיות: $O(n \log n)$

פיתרון 2 – הסתברותי:

- בוחרים איבר $k \in L$ באקראי.
- מחלקים את L ל-2 חלקים – האיברים שגדולים מ-k (L_2) ואיברים שקטנים מ-k.
- אם $len(L_1) > i$ מחזירים $Select(L_1, i)$.
- אם $len(L_1) = i$ מחזירים את k.
- אחרת מחזירים $Select(L_2, k - len(L_1) - 1)$.

ניתוח סיבוכיות:

במקרה הטוב: כל פעם שנבחר איבר ציר זה יהיה החציון ואז נקטין את המערך פי 2:

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

לפי משפט המאסטר מקבלים ש- $T(n) = \theta(n)$.

במקרה הגרוע: בכל שלב נקטין את המערך רק ב-1:

$$T(n) = \theta(n) + T(n - 1) \rightarrow \theta(n^2)$$

במקרה הממוצע: נבחר בכל פעם איבר ציר אקראי (לכל איבר יש סיכוי $\frac{1}{n}$ להיבחר) ונחסום מלמעלה ע"י כך שתמיד נבחרת תת-הרשימה הארוכה ביותר:

$$T(n) \leq \frac{1}{n} \left(\sum_{k=1}^n T(\max(k-1, n-k)) \right) + O(n)$$

• $O(n)$ זמן עבודה עבור החלוקה ל-2 רשימות.

אורך תת הרשימה הארוכה ביותר עבור כל k שנבחר מתפלג בין $[\frac{n}{2}, n - 1]$ לכן נקבל:

$$T(n) \leq \frac{1}{n} \left(2 \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} T(k) \right) + O(n)$$

- אם n זוגי, אז כל אורך של קטע הכי ארוך מופיע פעמיים (לדוגמא אם $n=6$ אז כאשר $k=2$ וגם כאשר $k=5$ אורך הקטע הארוך יותר הוא 4). אם n אי-זוגי, כולם יופיעו פעמיים פרט לאחד מהם.

נראה ש- $T(n) = O(n)$ באינדוקציה:

הוכחה: צריך למצוא c כך שמתקיים $T(n) \leq cn$. נניח שזה מתקיים עבור c קבוע מסוים המקיים את תנאי ההתחלה של הנוסחה. נשתמש בהנחת האינדוקציה ונקבל:

$$\begin{aligned} T(n) &\leq \frac{2}{n} \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} ck + dn \\ &= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k \right) + dn \\ &= \frac{2c}{n} \left(\frac{1}{2}(n-1)n - \frac{1}{2}(\lceil \frac{n}{2} \rceil - 1)\lceil \frac{n}{2} \rceil \right) + dn \\ &\leq c(n-1) - \frac{c}{n}(\lceil \frac{n}{2} \rceil - 1)\lceil \frac{n}{2} \rceil + dn \\ &= c \left(\frac{3}{4}n - \frac{1}{2} \right) + dn \\ &\leq cn \end{aligned}$$

נבחר $c \geq 4d$ כך ש- $\frac{3}{4}n - \frac{1}{2}$ יהיה דומיננטי יותר מאשר $O(n)$.

גרפים

הגדרות:

גרף – זוג $G = (E, V)$ כאשר E קבוצת קשתות ו- V קבוצת קודקודים.
 בגרף מכוון – $E \subseteq \{(u, v) | u, v \in V\}$
 בגרף לא מכוון – $E \subseteq \{\{u, v\} | u, v \in V\}$ (אין לולאות עצמיות).
 גרף ממושקל: $G = (E, V, W)$ כאשר $W: E \rightarrow \mathbb{R}$.

ייצוג של גרפים:

רשימת שכנויות – טוב לגרפים דלילים. מערך של קודקודים ולכל אחד יש מערך של מצביעים לקודקודים אליהם יש קשת.
 מטריצת שכנויות – טוב לגרפים צפופים.

ברשימת שכנויות:

- בגרף מכוון – סכום אורכי הרשימות הוא $|E|$. דוגמא – (u, v) מיוצג ברשימה של u בלבד.

- בגרף לא מכוון – סכום אורי הרשימות הוא $2|E|$. לדוגמא – (u,v) מיוצג ברשימה של u וגם של v .
- בגרף ממושקל – המשקל של קשת מאוחסן עם קודקוד v ברשימה של u .
- חיסרון – יש צורך לעבור על כל הרשימה כדי לחפש האם קשת קיימת.

במטריצת שכנויות:

- הקודקודים ממוספרים בצורה שרירותית $1,2,\dots, |V|$. המטריצה A שמימדיה $|V| \times |V|$ מייצגת את גרף G וערכיה הם:

$$a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$$
- נגדיר את A^T להיות המטריצה המשוחלפת של A . בגרף לא מכוון $A = A^T$.
- בגרף לא מכוון – אפשר לשמור רק חצי מטריצה (חוסך מקום).
- ייצוג בגרף ממושקל:

$$a_{ij} = \begin{cases} w(i,j) & (i,j) \in E \\ 0 \text{ or } \infty & (i,j) \notin E \end{cases}$$

מעבר על גרפים

חיפוש לרוחב: BFS

בהינתן $G = (E, V)$ s -י (קודקוד מקור), BFS בוחן את הקשתות כדי לגלות כל קודקוד שניתן להגיע אליו מ- s . הוא מחשב מרחק (מספר קשתות מינמלי) מ- s לכל קודקוד ובונה "עץ רוחב" ששורשו s . "רוחב" – האלגוריתם מגלה את כל הקודקודים שבמרחק k מ- s לפני שהוא מגלה את הקודקודים במרחק $k+1$ מ- s .

האלגוריתם:

- $Color$ – מערך בגודל $|V|$ המאחסן את צבע הקודקוד
- Π – מערך שמאחסן קודקוד קודם של קודקוד מסוים. לדוגמא - $\Pi[u]=v$ אומר שבמהלך הסריקה, v נמצא לפני u .
- d – מערך מרחקים מ- s .
- Q – תור של קודקודים אפורים.

BFS(E,V, s):

```

for each  $u \in V - \{s\}$ 
     $color[u] = white$ 
     $d[u] = \infty$ 
     $\Pi[u] = null$ 
 $color[s] = gray$ 
 $d[s] = 0$ 
 $\Pi[s] = null$ 
 $Q.Enqueue(s)$ 
while  $Q.IsEmpty \neq false$ 
     $u = Q.Top$ 
    for each  $v \in Adj[u]$ 
        if  $color[v] == white$ 
             $color[v] = gray$ 
             $d[v] = 0$ 
             $\Pi[v] = u$ 
             $Q.Enqueue(v)$ 
     $Q.Dequeue$ 
     $Color[u] = black$ 

```

סיבוכיות:-

- הוצאה הכנסה לתור – $O(1)$. כל קודקוד נכנס לכל היותר פעם אחת לתור – $O(|V|)$.
- רשימת שכנויות נסרקת פעם אחת לכל היותר – $O(|E|)$.

$$O(|V| + |E|) \leftarrow$$

חיפוש לעומק: DFS

מתחילים מקודקוד אקראי. נבדקות כל הקשתות היוצאות מן הקודקוד שהוא אחרון הקודקודים שהתגלו שעדיין יש לו קשתות שיוצאות ממנו ועדיין לא התגלו. לאחר שנבדקו כל הקשתות היוצאות מ- v , האלגוריתם "נסוג" וממשיך בבדיקת קשתות שיוצאות מהקודקוד שממנו התגלה v . (הרעיון דומה ל- $preorder$ בעצים).

האלגוריתם:

- $Color$ – מערך בגודל $|V|$ המאחסן את צבע הקודקוד. לבן- קודקוד שלא ביקרו בו, אפור – קודקוד שביקרו בו אך לא סיימנו לסרוק את הבנים שלו, שחור – סיימנו לסרוק את הבנים שלו.
- II – מערך שמאחסן קודקוד קודם של קודקוד מסוים. לדוגמא - $II[u]=v$ אומר שבמהלך הסריקה, v נמצא לפני u .
- d – זמן המציין מתי הגענו לקודקוד מסוים בפעם הראשונה.
- f – זמן סיום מעבר על כל הבנים של קודקוד מסוים.

DFS(V,E):

```
for each  $u \in V$ 
     $color[u] = white$ 
     $II[u] = null$ 
 $time = 0$ 
for each  $u \in V$ 
    if  $color[u] == white$ 
        DFS-visit( $u$ )
```

DFS-visit(u):

```
 $color[u] = gray$ 
 $time++$ 
 $d[u] = time$ 
for each  $v \in Adj[u]$ 
    if  $color[v] == white$ 
         $II[v] = u$ 
        DFS-visit( $v$ )
 $color[u] = black$ 
 $f[u] = time++$ 
```

סיבוכיות: $O(|V|+|E|)$

עצים פורשים מינמליים – Minimum spanning tree

עץ – גרף קשיר ללא מעגלים, כלומר יש מסלול מכל קודקוד לכל קודקוד.

הגדרה – יהי גרף $G = (V, E)$ פונקציית משקל על הקשתות $W: E \rightarrow \mathbb{R}$. עץ פורש הוא עץ (או תת-גרף) $T = (V, E')$ כאשר $E' \subseteq E$. משקל $T: w(T) = \sum_{e \in E'} w(e)$.

עץ פורש מינימלי – עץ פורש עם המשקל הנמוך ביותר מבין כל העצים הפורשים של הגרף.

הגדרה – בעיית הסוכן הנוסע – TSP.

נתון גרף מלא (יש קשת בין כל זוג קודקודים) ממושקל. רוצים למצוא מסלול $v_1, v_2, \dots, v_n, v_1$ שעובר בכל קודקוד בדיוק פעם אחת ומשקלו הנמוך ביותר.
הנחה – G מקיים את אי-שוויון המשולש $w(u, v) + w(v, k) \geq w(u, k)$.

תרגיל:

נתון G כנ"ל. יהי W_{OPT} המשקל של המסלול הטוב ביותר ב-TSP. מצאו אלגוריתם פולינומיאלי שמוצא מסלול המבקר בכל צומת פעם אחת בלבד עם משקל $W < 2 * W_{OPT}$. (אלגוריתם קירוב של TSP עד 2).

פיתרון:

- OPT הוא עץ פורש (עוברים בכל קודקוד פעם אחת) + קשת נוספת שסוגרת את המעגל. כלומר מתקיים $W_{MST} < W_{OPT}$.
- הוכחה – ניקח את OPT ונוריד ממנו קשת e (שסוגרת מעגל):
 $w_{MST} \leq w(OPT - e) = w_{OPT} - w(e)$
 מכיוון שעבור כל קשת מתקיים $w(e) > 0$ אזי $w_{MST} < w_{OPT}$.
- נמצא MST באמצעות $Prim$.
- נעבור על ה- MST באמצעות BFS . יוצרים מסלול שהוא סדר הקודקודים במעבר (הלוך וחזור). עוברים על כל קשת פעמיים. משקל המסלול כעת הוא $W = 2w_{MST} < 2w_{OPT}$.
- נשמיט קודקודים כפולים.
- כשמורידים קודקוד, לוקחים במקום זה את הקשת בין הקודקוד שלפני לזה שאחריו. בגלל אי-שוויון המשלוש – מובטח שאנחנו לא מגדילים את המסלול –
 $w(u, v) + w(v, k) \geq w(u, k)$
 נשמיט את v ונישאר עם:
 $W - w(u, v) - w(v, k) + w(u, k)$

דוגמא:

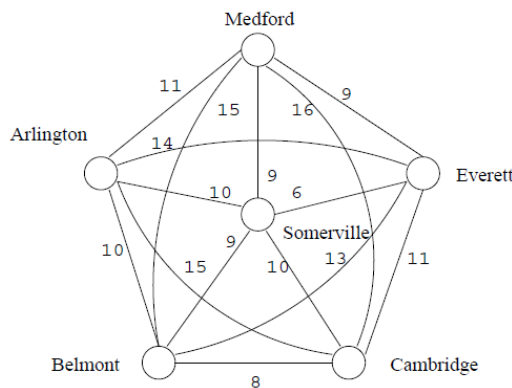


Figure 1: Example input to the TSP

מציאת MST:

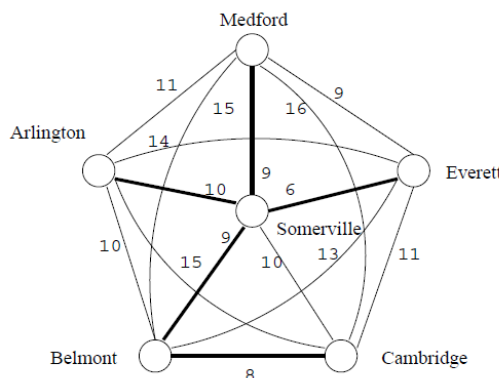


Figure 2: Minimum spanning tree

מעבר על העץ באמצעות BFS (הלוך וחזור):

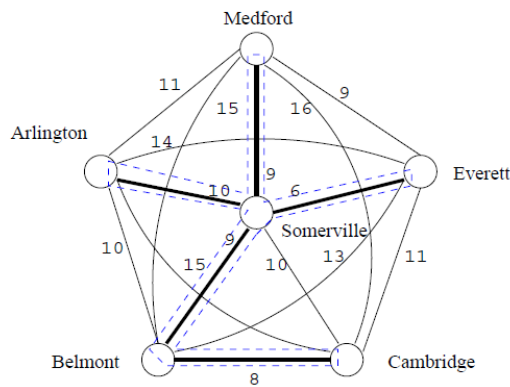


Figure 3: Cycle from MST shown in dashed blue

נקבל את המסלול: S M S E S B C B S A

נשמיט קודקודים כפולים (כמובן שבגלל שזה מעגל נוסף את הקודקוד הראשון) S M E B C A S:

הפיתרון (מסלול ללא קודקודים כפולים):

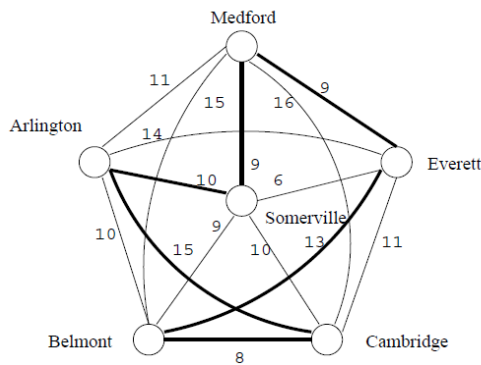


Figure 4: Solution with weight less than $2 * OPT$