מרתון 2 - אלגוריתמים 2 - 10.6 (המשך)

[קישור להקלטה של המרתון]

אלגוריתם Best:

מטרה: מציאת קטע עם סכום מקסימאלי במערך.

נתון לנו מערך שיש בו גם מספרים שליליים ואנו רוצים למצוא רצף במערך עם הסכום הגדול ביותר.

לדוגמא: $[5, -6, 2, -1, 7, 3, -2, 4, 4, -20, 5, 8, 3]$

האלגוריתם:

עוברים פעם אחת על המערך וסוכמים:

בכל שלב שומרים את המקסימום עד כה.

אם הסכום יורד מתחת ל 0, מאפסים אותו ומתחילים את הרצף מחדש.

פסאודו-קוד:

```
Best(A)
    int s = 1, e = 1, sum = 0, max = -inf , t_s = 1
    for i = 1 to A.length
        sum = sum + A[i]
        if(sum > max)
            max = sum
            s = t_s
            e = i
        if(sum < 0)
            sum = 0
            t_s = i+1
    return (max, s, e)
```

סיבוכיות: $O(n)$ כאשר n הוא אורך המערך.

אלגוריתם Best מעגלי:

הרעיון להשתמש ב Best הרגיל.

לקחת את סכום כל המערך ולהוריד את הרצף הכי מינימאלי שאינו מעגלי ובכך לקבל את הרצף המקסימאלי המעגלי.

או שהרצף הוא לא מעגלי וניתן למצוא אותו כמו מקודם.

לדוגמא: $[5, -6, 2, -1, 7, 3, -2, 4, 4, -20, 5, 8, 3]$

ניקח את ה best הרגיל: $Best = (17, 3, 9)$.

וניקח את $B = [-5, 6, -2, 1, -7, -3, 2, -4, -4, 20, -5, -8, -3]$

מכאן: $sum(A) = 12$, $Best(B) = 20$

קוד:

```
cycleBest(A)
    sum = 0
    B = new Array[A.length]
    for i = 1 to A.length
        sum = sum + A[i]
        B[i] = -A[i]
```

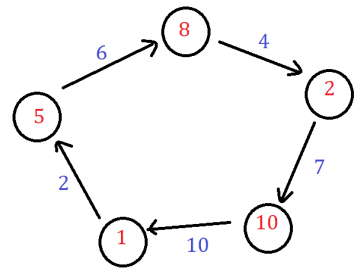
```
(b1, s1, e1) = Best(B)
(b2, s2, e2) = Best(A)
if(sum-(-b1) < b2) return (b2, s2, e2)
else return (sum+b1,e1+1,s1-1)
```

שימוש ב cycleBest - בעיית תחנת הדלק:

נתון מעגל שסביבו יש תחנות דלק, בכל תחנה i ניתן למלא $A[i]$ ליטר דלק. בין התחנה ה i לתחנה הבאה צורכים $B[i]$ דלק.

המטרה: למצוא האם ניתן להתחיל נסיעה מאחת התחנות עם 0 ליטר דלק בהתחלה ולסיים סיבוב שלם על המעגל. ואם כן - מאיזו תחנה.

דוגמא:



כאן לא ניתן להתחיל מאף נקודה כדי לסיים את הסיבוב.

אלגוריתם:

קלט: 2 מערכים:

$A[i]$ - כמה דלק ניתן למלא בתחנה i.

$B[i]$ - כמה דלק יש לבזבז מתחנה i לתחנה הבאה.

הרעיון:

- ניצור מערך חדש: $C[i] = A[i] - B[i]$.
- נפעיל $BestCycle$ על המערך C ונקבל נקודת התחלה אופטימאלית (הרצף שייתן את כמות הדלק הגבוהה ביותר בנסיעה באותו הרצף)
- נתחיל מהנקודה שחזרה (נקודת ההתחלה) ונבדוק האם ניתן לסיים את הסיבוב. אם כן - זו הנקודה. אם לא - אז לא ניתן.

סיבוכיות: $O(n)$ כאשר n - גודל 2 המערכים. (מספר תחנות הדלק).

אלגוריתם $Best$ במטריצה:

נתונה מטריצה $N \times M$ של מספרים. יש למצוא את תת המטריצה (מלבן) שסכום הערכים בה הוא הגדול ביותר.

לדוגמא:

	2	10		5	
12		3		4	
	2	2		10	7
8	1				

		5	7	2	9

אלגוריתם:

- עוברים על כל רצף שורות (או עמודות - המינימאלי מביניהם) וסוכמים את הכל למערך אחד. (אם עוברים על השורות אז כל תא המערך החדש הוא סכום אותה עמודה בכל השורות (ולהיפך))
- מפעילים $Best$ רגיל על המערך הנוצר ומקבלים את הערכים - ושומרים את המקסימום.
- לבסוף מחזירים את הערך הגדול ביותר.

לדוגמא:

$[0,0]$ (שורה 0 בלבד) נקבל את המערך: $[-20, 2, 10, -1, 5, -3]$. הBest עבורו יחזיר: $(16, 1, 4)$.
הסכום הוא 16, מתחילים ממיקום 1 עד מיקום 4 (בעמודות).

$[0,1]$ (שורות 0+1) נקבל את המערך: $[-8, -2, 13, -11, 9, -4]$. הBest עבורו יחזיר: $(13, 2, 2)$.
$[0,2]$ (שורות 0+1+2) …
…
$[4,4]$ (שורה 4 בלבד)
לוקחים את המקסימום מכל התוצאות.

כדי לחשב את הסכומים עבור מערך העזר באופן יעיל. ניצור לפני תחילת האלגוריתם מטריצת עזר h שבתא $[i,j]$ יהיה שמור הסכום של עמודה j משורה 1 ועד שורה i.

כעת, כדי לחשב לדוגמא את המערך שסוכם את שורות 2+3. את המיקום i של מערך העזר יכיל את:
$arr[i] = h[3][i] - h[1][i]$.

לכן הסיבוכיות תהיה:
$O(nm)$ כדי ליצור את מטריצת h של הסכומים המצטברים.
$O(n^2(m+m))$ - עוברים על כל הקומבינציות של משורה x עד שורה y. סיבוכיות: $O(n^2)$.
ממלאים את מערך העזר בגדול m באמצעות מטריצת הסכומים (כמו שראינו) - $O(m)$
ואז מפעילים Best רגיל על אותו מערך. $O(m)$.

פסאודו - קוד: (כאן התחלנו מאינדקס 0)

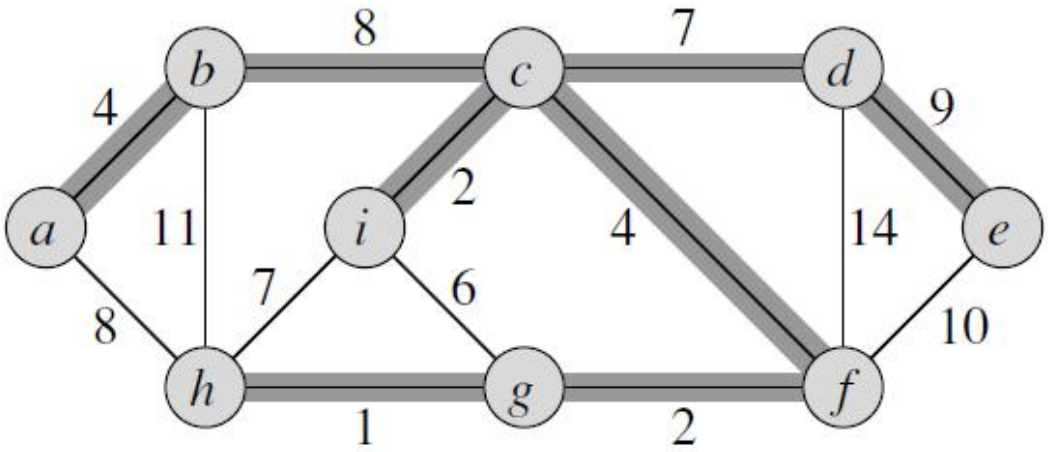
```
Best-Matrix(M)
    n - M-rows
    m - M-cols
    max = − ∞
    row_s, row_e, col_s, col_e
    H = new Matrix[n][m]
    for i = 0 to m-1
        H[0][i] = M[0][i]
    for i = 1 to n-1
        for j = 0 to m-1
            H[i][j] = H[i-1][j] + M[i][j]
    for i = 0 to n-1
```

```
        for j = i to n-1
                A = new Array[m]
                for k = 0 to m-1
                        if(i == 0) A[k] = H[j][k]
                        else A[k] = H[j][k] - H[i-1][k]
                (sum, s, e) = Best(A)
                if(sum > max)
                        max = sum
                        row_s = i , row_e = j, col_s = s, col_e = e
return (max, row_s, row_e, col_s, col_e)
```

עצים פורשים מינימאליים:

נתון לנו גרף שיש לו משקלים על הצלעות.

המטרה: למצוא תת גרף שהוא עץ (קשיר ללא מעגלים) שסכום משקלי הצלעות שעליו הוא מינימאלי.



בגרף הנ"ל, העץ הפורש המינימאלי הוא הצלעות המסומנות.

ישנם 2 אלגוריתמים עיקריים (+1 נוסף לקורס) - כולם חמדניים.

אלגוריתם 1: **קרוסקל**.

- מיין את הצלעות מהצלע עם המשקל הנמוך ביותר לגבוה ביותר.
- עבור על הצלעות מהנמוך לגבוה וכל צלע שלא סוגרת מעגל עם מה שכבר לקחנו לעץ לפני כן - הוסף אותה לעץ.
- סיים כאשר לקחת בדיוק $n - 1$ צלעות (n הוא כמות הקודקודים).

סיבוכיות: $O(|E|log|E|)$ עבור המיון.

מעבר על הצלעות מהקטנה לגדולה: $O(|E|)$.

- בכל מעבר צריך לבדוק שלא סוגרים מעגל.

שומרים את הקודקודים שהם באותו רכיב קשירות בתוך מבנה נתונים $union - Find$ (איחוד קבוצות זרות) סיבוכית פעולות שימוש במבנה (union/find) היא $O(log|V|)$.

- סה"כ: $O(|E|log|V|)$.

לכן הסיבוכיות הכוללת: $O(|E|log|E|)$.

הערה: פונקציית union מאחדת ומחזירה האם הצלחנו לאחד (כלומר שהם לא באותה קבוצה)

פסאודו קוד:

```
Kruskal(G=(V,E))
        DisjointSets d = new DisjointSets(|V|)
        Tree T = Φ
        sort(|E|) // by weight
        for i = 1 to |E|
                e = E[i]
                if(d.union(e.v1, e.v2)) T.add(e)
                if(|T| == n-1) return T
        return T
```

אלגוריתם 2: **פרים**

- מגדירים תור עדיפויות ומערך $visit[|V|]$ עבור הקודקודים.
- מתחילים מקודקוד שרירותי s על הגרף. (s נכנס לתור עם עדיפות 0)
- מכניסים לתור העדיפויות את כל השכנים שלו עם העדיפות של משקל הצלע שמגיעה אליהם מ s.
- כל עוד התור לא ריק, שולפים את הקודקוד עם העדיפות המינימאלית, מוסיפים את הצלע לעץ (בין הקודקוד הנשלף לאבא שלו) ומכניסים לתור את כל השכנים שלא סיימנו איתם עם עדיפות של משקל הצלע שמגיעה אליהם מאותו קודקוד (אם הם כבר בתור אז רק מעדכנים את העדיפות למינימאלי)

פסאודו קוד:

```
Prim(G=(V,E))
        PriorityQueue q
        Tree T = Φ
        visit = new Array[|V|] // init to False
        pred = new Array[|V|] // init null
        d = new Array[|V|] // init ∞
        d[1] = 0
        q.add((1,0))
        while(q not empty)
                v = q.extractMin()
                if(pred[v] ≠ null) T.add((v,pred[v]))
                for each u in G[v] // שכנים
                        if(!visit[u] and d[u] > E[v][u].weight)
                                d[u] = E[v][u].weight
                                pred[u] = v
                                if(q.contains(u)) q.decreaseKey(u, d[u])
                                else q.add((u,d[u]))
                visit[v] = True
```

```
return T
```

סיבוכיות: אתחול מערכים: $O(|V|)$.
לולאה ראשית עוברת על כל קודקוד פעם אחת ואז על כל השכנים שלו: $O(|E|)$.
- בכל איטרציה, מכניסים ומעדכנים בתור העדיפויות (במקרה הגרוע) - $O(log|V|)$.

סיבוכיות כוללת: $O(|V| + |E|log|V|)$
סה"כ: $O(|E|log|V|)$.

אלגוריתם 3: **בורובקה**
נתחיל מרכיבי קשירות (כמו union-find) בקרוסקל.
בכל שלב נמצא את הצלע הכי נמוכה שמחוברת לרכיב הקשירות ולא סוגרת מעגל.
כל עוד לא סיימנו, מצא את הצלעות הנמוכות לכל רכיב (המחברות בין 2 רכיבים שונים) והוסף אותן לעץ.

פסאודו קוד:

```
Boruvka(G=(V,E))
    DisjointSets d = new DisjointSets(|V|)
    Tree T = Φ
    isFinnished = False
    while(!isFinnished)
        cheapest = new Array[|V|] // init to null
        for i = 1 to |E|
            e = E[i]
            g1 = d.find(e.v1) , g2 = d.find(e.v2)
            if(g1 ≠ g2)
                if(e.weight < cheapest[g1].weight) cheapest[g1] = e
                if(e.weight < cheapest[g2].weight) cheapest[g2] = e
        isFinnished = True
        for i = 1 to |V|
            if(cheapest[i] ≠ null)
                T.add(cheapest[i])
                d.union(cheapest[i].v1, cheapest[i].v2)
                isFinnished  = False
```

סיבוכיות: ניתן להגיע ע"י מימוש יעיל ל $O(|E|log|V|)$ כי בכל שלב עוברים על כל הצלעות ולאחר כל שלב, כמות הרכיבים מצטמצמת בפי 2 פחות (ואולי גם יותר).

אלגוריתם קידוד האפמן:
נתון מערך של תווים שלכל תו יש תדירות מופעים בטקסט.
המטרה לקודד כל תו לרצף ביטים כך שהרצף הכולל יהיה קצר ביותר (לחסוך במקום)

לדוגמא: טקסט: $aabcbbdcaabaa$
אז: התדירויות הן: $\#_a = 6,\ \#_b = 4,\ \#_c = 2,\ \#_d = 1$.
אם נבחר את הקידוד: $a = 0,\ b = 101,\ c = 100,\ d = 11$.

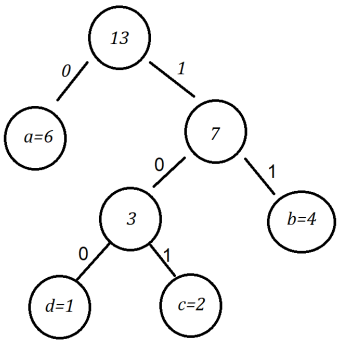
אז קידוד הטקסט כולו יהיה: 0010110010110111110000010100.
השימוש הוא בעיקר לכיווץ קבצים.

הרעיון הכללי הוא לתת קידוד כמה שיותר קצר לתו שמופיע כמה שיותר הרבה.
לשם כך יש את קידוד האפמן. (שהוא אופטימאלי).

אלגוריתם:

- הכנס את כל התדירויות לערימת מינימום.
- בכל שלב, שלוף את 2 הערכים הקטנים ביותר. בנה מהם עץ בינארי ש 2 הערכים הם בנים ימני ושמאלי ולהם אב אחד משותף שערכו הוא סכום 2 הערכים. הכנס את האבא לערימה במקומם.
- חזור על התהליך עד שנשאר רק איבר אחד בערימה והוא שורש העץ.

דוגמא:



סיבוכיות: $O(nlog(n))$.

ניתן ליעל את הסיבוכיות אם נתון מערך ממויין לפי התדירויות אז במקום להשתמש בערימה, נשתמש ב 2 תורים רגילים: נכניס את כולם לתור הראשון ובכל שלב, נשלוף את ה 2 הנמוכים ביותר (שהם בראשי התורים), נמזג אותם לאבא משותף אחד כמו מקודם ונכניס אותו לתור השני.
לאורך כל האלגוריתם, 2 התורים ממויינים ולכן 2 הקטנים ביותר יהיו 2 הראשונים בתור הראשון או 2 הראשונים בתור השני או אחד מהראשון ואחד מהשני.

סיבוכיות: $O(n)$.

פסאודו קוד לשני + מיון אם צריך:

```
HuffmanCode(A) // A.f - תדירות , A.c - התו עצמו
        Sort(A) // by A.f
        n = |A|
        Queue q1, q2
        for i = 1 to n
                q1.enqueue(new Node(A[i]))
        while(q1.size() + q2.size()  > 1)
                x = getMin(q1,q2)
```

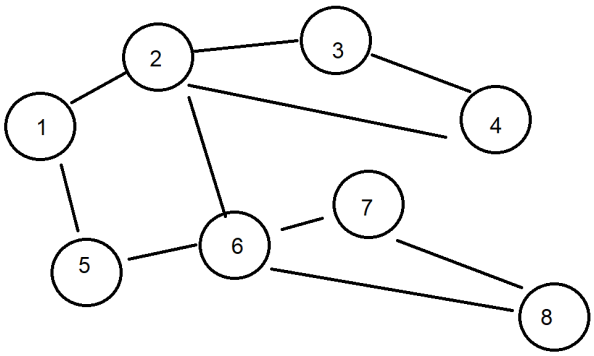
```
            y = getMin(q1,q2)
            z = new Node(x.data.f + y.data. f)
            z.left = x , z.right = y
            q2.enqueue(z)
        if(q1 empty) return q2.dequeue()
        else return q1.dequeue()

getMin(Queue q1, Queue q2)
        if(q1 empty) return q2.dequeue()
        else if(q2 empty) return q1.dequeue()
        else if(q1.head().data.f < q2.head().data.f) return q1.dequeue()
        else return q2.dequeue()
```

מעגל אוילר:

מעגל בגרף שעובר בכל הצלעות בגרף, בכל צלע בדיוק פעם אחת.
משפט: יש מעגל אוילר בגרף אם ורק אם כל הדרגות הן זוגיות.
משפט: יש מסלול אוילר בגרף אם ורק אם יש בדיוק 0 או 2 קודקודים מדרגה אי זוגית.

המטרה: בהינתן הגרף, להחליט הם יש מסלול/מעגל ואז למצוא אותו.
לדוגמא:



מעגל אוילר: $1 - 5 - 6 - 8 - 7 - 6 - 2 - 4 - 3 - 2 - 1$.

האלגוריתם:

- בדיקה האם יש מעגל: עוברים על כל קודקוד ובודקים האם דרגתו זוגית, סופרים את אלו עם הדרגה שהיא אי זוגית. אם יש לנו 0 אי זוגיים - יש מעגל, אם יש 2 אי זוגיים - יש מסלול ואחרת אין מעגל ואין מסלול.
- מגדירים מחסנית ורשימה המייצגת את המסלול עצמו.
  אם זה מסלול ולא מעגל, מתחילים מקודקוד מדרגה אי זוגית. ואם זה מעגל אז בוחרים קודקוד שרירותית. מכניסים את קודקוד ההתחלה למחסנית.
- אם יש לו שכן, מכניסים את השכן למחסנית, מוחקים את הצלע וממשיכים עם השכן. עד שנתקעים (אין יותר שכנים).
- ברגע שנתקעים, מוציאים את ראש המחסנית ומוסיפים למסלול. וממשיכים עם הקודקוד הבא במחסנית. עד שמתרוקנת המחסנית.

פסאודו קוד:

```
EuilerPathCycle(G=(V,E))
        Stack s
        List path
        count = 0
        start = 1
        for each v in V
                if(G[v].size() % 2 == 1)
                        count++
                        start = v
        if(count > 2) return “No path and no cycle”
        s.push(start)
        while(s not empty)
                if(G[s.top()].size() > 0)
                        u = G[v].getFirst()
                        s.push(u)
                        G.remove(v,u)
                else
                        path.add(s.pop())
        return path
```

סיבוכיות: $O(|V| + |E|)$ ולכן: $O(|E|)$. כי בכל איטרציה יורדת צלע ולכן יהיו לכל היותר $O(|E|)$ איטרציות כאשר בכל איטרציה זה $O(1)$ (הערה: זה כולל מימוש נכון של מחיקת הצלע בין u ל v ב 2 הכיוונים)