אלגוריתמים – מטלה 2

אלעזר פיין
מאור אופק

א. האלגוריתם שהורץ מצורף בקובץ C ‘q1a’.

	j	0	1	2	3	4	5	6	7	8	9
i		Y_i	1	0	1	1	0	1	1	1	0
0	X_i	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	**1**	1	1	1	1	1	1
2	1	0	1	1	2	**2**	2	2	2	2	2
3	0	0	1	2	2	**2**	**3**	**3**	**3**	3	3
4	0	0	1	2	2	2	**3**	3	**3**	3	4
5	1	0	1	2	3	3	3	**4**	4	**4**	4
6	0	0	1	2	3	3	4	**4**	4	4	**5**
7	1	0	1	2	3	4	4	5	**5**	5	**5**

	j	0	1	2	3	4	5	6	7	8	9
i		Y_i	1	0	1	1	0	1	1	1	0
0	X_i	0	0	0	0	0	0	0	0	0	0
1	1	0	↖	←	↖	↖	←	↖	↖	↖	←
2	1	0	↖	↑	↖	↖	←	↖	↖	↖	←
3	0	0	↑	↖	↑	↑	↖	←	←	←	↖
4	0	0	↑	↖	↑	↑	↖	↑	↑	↑	↖
5	1	0	↖	↑	↖	↖	↑	↖	↖	↖	↑
6	0	0	↑	↖	↑	↑	↖	↑	↑	↑	↖
7	1	0	↖	↑	↖	↖	↑	↖	↖	↖	↑

תמ"א ממסלול אדום: 11010
תמ"א ממסלול ירוק: 11011

ב. האלגוריתם (רשום בפייתון, בנוסף הקובץ המצורף ‘q1b.py’):

```python
def is_common(x: str, y: str, z: str) -> bool:
    x_length = len(x)
    y_length = len(y)
    z_length = len(z)
    i = found_in_x = found_in_y = 0

    while i < z_length:
        current_i = i
        for j in range(found_in_x, x_length):
            if i < z_length and x[j] == z[i]:
                found_in_x += 1
                i += 1
        i = current_i
        for j in range(found_in_y, y_length):
            if i < z_length and y[j] == z[i]:
                found_in_y += 1
                i += 1
        i += 1

    return found_in_x == z_length == found_in_y
```

ג. 1200122

ד. ניתן להסיק מהטבלה בכל פעם שהערך גדל בכיוון X אותו ערך נמצא גם בX, לכן:

x = 2120211221

1.

א. ניתן:

אנחנו יודעים לפי R(1,8) = 3 כי הפיצול הראשי הוא בין 1..3 ל 4..8.

נבדוק פיצול פנימי של 1..3:

M[1,3] = min { m[1,1] + m[2,3] + d0d1d2, m[1,2] + m[3,3] + d0d2d3}

ניתן לראות כי הפיצול המינימלי האפשרי היחיד הוא m[1,1] + m[1,2] משום ש T(1,2) > T(1,3).

נבדוק פיצול פנימי של 4..8 בצורה דומה כאשר בכל פיצול פנימי יהיה אך ורק פיצול מינימלי אפשרי יחיד ונקבל:

$$(M_1 \cdot (M_2 \cdot M_3)) \cdot ((((M_4 \cdot M_5) \cdot M_6) \cdot M_7) \cdot M_8)$$

ב.   ניתן:

T(i, i+1) = d(i-1)d(i)d(i+1)
d(0) = 6, d(4) = 30

d(1)d(2) = d(0)d(1)d(2) / d(0) = 1440 / 6 = 240
d(1)d(2)d(3) = 480 => **d(3)** = 480 / 240 = **2**
d(3)d(4)d(5) = 420 => **d(5)** = 420 / (2 * 30) = **7**
d(4)d(5)d(6) = 21000 => **d(6)** = 21000 / (30 * 7) = **100**
d(5)d(6)d(7) = 63000 => **d(7)** = 63000 / (100 * 7) = **90**
d(6)d(7)d(8) = 72000 => **d(8)** = 72000 / (100 * 90) = **8**
d(2)d(3)d(4) = 4800 => **d(2)** = 4800 / (2 * 30) = **80**
=> **d(1) =** 240 / 80 = **3**

⇨ d_vector = (6, 3, 80, 2, 30, 7, 100, 90, 8)

.2

א.   נפעיל את האלגוריתם לבעיית התרמיל עבור פריט אחד מכל סוג:

1. build tables T, S of size (n+1)X(W+1)
2. for j from 0 to W
   1. T[0,j] ← 0
3. for i from 0 to n
   1. T[i,0] ← 0
4. for i from 1 to n
   1. for j from 1 to W
      1. if $w_i > j$
         1. T[i,j] ← T[i-1,j]
         2. S[i,j] ← “n”
      2. else
         1. T[i,j] ← max (T[i-1,j- w_i]+ v_i , T[i-1,j])
         2. if T[i,j] = T[i-1,j- w_i]+ v_i
            1. S[i,j] ← “y”
         3. else
            1. S[i,w] ← “n”
5. return T[n,W]

נקבל טבלאות פתרון T,S אותן ניתן לאחד:

	0	**0.5**	**1**	**1.5**	**2**	**2.5**	**3**	**3.5**	**4**	**4.5**	**5**	**5.5**
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	8Y	8Y	8Y	8Y	8Y	8Y	8Y	8Y	8Y	8Y	8Y
2	0	8N	8N	8N	**16Y**	24Y	24Y	24Y	24Y	24Y	24Y	24Y
3	0	8N	8N	8N	16N	28Y	36Y	36Y	36Y	**44Y**	52Y	52Y
4	0	8N	14Y	22Y	22Y	28N	36N	42Y	50Y	50Y	52N	**58Y**
5	0	8N	14N	22N	22N	28N	46N	42N	50N	50N	52N	58N

1. נתחיל מהתא האחרון (5, 5.5) – לא נלקח לכן עוברים לפריט הבא (i-=1).
2. הפריט הבא בתא (4, 5.5) נלקח, נחסיר את משקלו, נסמן ונעבור לפריט הבא.
3. הפריט הבא בתא (3, 4.5) נלקח, נחסיר את משקלו, נסמן ונעבור לפריט הבא.
4. הפריט הבא בתא (2, 2) נלקח, נחסיר את משקלו, נסמן ונעבור לפריט הבא.
5. הגענו לתנאי עצירה – לא נותר משקל.
6. <u>פתרון אופטימלי סופי – העמסת פריטים 4, 3, 2 תמורת רווח של 58.</u>

ב. מכיוון שהמשקל החדש הוא בדיוק המשקל הכולל של כלל הפריטים הפתרון האופטימלי יהיה להעמיס את כולם... הרווח יהיה ערכם הכולל שהוא 8+16+28+14+15 = 81

ג.

	a_1	a_2	a_3	a_4	a_5
Weight	0.5	2	2.5	1	2
Value	8	16	28	14	15
Value Per 1 Weight Unit	16	8	11.2	14	7.5

הפתרון כאן טריוויאלי **ואין צורך** להשתמש באלגוריתם של בעיית תרמיל עבור אינסוף פריטים...
ניתן לראות כי פריט מס' 1 הוא בעל הרווח הגדול ביותר עבור יחידת משקל אחת, לכן הפתרון האופטימלי הוא למלא את המשקל איתו (a1 * 11 = a1 * 5.5 / 0.5) כלומר להעמיס 11 פעמים a1 תמורת רווח כולל של 11*16 = 176.

ד.   (בנוסף מצורף כקובץ C ”q3d.c“)

```c
#include <stdio.h>
#include <minmax.h>

void print_matrix(int *mat, int n, int w, int is_chars) {
    char *ch_mat = (char *) mat;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            if (is_chars)   printf("%3c", *ch_mat++);
            else            printf("%5d", *mat++);
        }
        puts("");
    }
    puts("");
}

void q3d(int N, const int *values, const int *weights, int W) {
    int T[N + 1][W + 1];
    char S[N + 1][W + 1];

    for (int i = 0; i < N + 1; i++) {
        for (int j = 0; j < W + 1; j++) {
            T[i][j] = 0;
            S[i][j] = '0';
        }
    }

    for (int i = 1; i < N + 1; i++) {
        for (int j = 1; j < W + 1; j++) {
            if (weights[i - 1] > j) {
                T[i][j] = T[i - 1][j];
                S[i][j] = 'n';
            } else {
                T[i][j] = max(T[i - 1][j - weights[i - 1]] + values[i - 1],
                              T[i - 1][j]);
                if (T[i][j] == T[i - 1][j - weights[i - 1]] + values[i - 1]) {
                    S[i][j] = 'y';
                } else {
                    S[i][j] = 'n';
                }
            }
        }
    }

    puts("T Table:");
    print_matrix((int *) T, N + 1, W + 1, 0);
    puts("S Table:");
    print_matrix((int *) S, N + 1, W + 1, 1);

    printf("Optimal Solution For The Original Problem (div by two): %d\n",
           T[N][W] / 2);
    printf("Optimal Solution: %d\n", T[N][W]);
    for (int n = N, w = W; n > 0; n--) {
        if (T[n][w] != T[n - 1][w]) {
            printf("\tItem #%d | Weight = %d | Value = %d\n",
                   n, weights[n - 1], values[n - 1]);
            w -= weights[n - 1];
        }
    }
}
```

ה. (בנוסף מצורף כקובץ C “q3e.c”)

```c
#include <stdio.h>
#include <minmax.h>

void print_matrix(int *mat, int n, int w, int is_chars) {
    char *ch_mat = (char *) mat;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            if (is_chars)   printf("%3c", *ch_mat++);
            else            printf("%5d", *mat++);
        }
        puts("");
    }
    puts("");
}

void print_x(int *X, int n) {
    printf("X Table:\nItem:\t");
    for (int i = 0; i < n; i++) printf("%5d", i + 1);
    printf("\nCount:\t");
    for (int i = 0; i < n; i++) printf("%5d", X[i]);
    puts("");
}

void q3e(int N, int *values, int *weights, int W) {
    int T[N + 1][W + 1];
    char S[N + 1][W + 1];
    int X[N];

    for (int i = 0; i < N + 1; i++) {
        for (int j = 0; j < W + 1; j++) {
            T[i][j] = 0;
            S[i][j] = '0';
        }
        X[i % N] = 0;
    }

    for (int i = 1; i < N + 1; i++) {
        for (int j = 1; j < W + 1; j++) {
            if (weights[i - 1] <= j) {
                T[i][j] = max(T[i - 1][j],
                              T[i][j - weights[i - 1]] + values[i - 1]);
                if (T[i][j] != T[i - 1][j]) {
                    X[i - 1]++;
                    S[i][j] = 'y';
                } else {
                    S[i][j] = 'n';
                }
            } else {
                T[i][j] = T[i - 1][j];
                S[i][j] = 'n';
            }
        }
    }
```

```
    puts("T Table:");
    print_matrix((int *) T, N + 1, W + 1, 0);
    puts("S Table:");
    print_matrix((int *) S, N + 1, W + 1, 1);
    print_x(X, N);

    printf("\nOptimal Solution: %d\n", T[N][W]);
    for (int n = N, w = W; n > 0; n--) {
        if (T[n][w] != T[n - 1][w]) {
            printf("\tItem #%d | Weight = %d | Value = %d\n", n,
                   weights[n - 1], values[n - 1]);
            w -= weights[n - 1];
        }
    }
}
```

3. **א+ב:**

זוהי בעיית הקצאת משאבים, להלן אלגוריתם לבניית טבלאות F, X לערכי f, x בהתאמה עבור כל שילוב, כאשר f(t,n) הוא הרווח המצטבר הכולל עד בניין מסוג t בכמות מקסימלית n ו x(t,n) הוא כמות הבניין הנוכחי בפתרון אופטימלי הכולל את הסוג שלו.

משתמשים באלגוריתם סטנדרטי של בעיית הקצאת משאבים כלומר עוברים סוג סוג (בכל סוג עוברים על כל הכמויות) ופותרים בהסתמכות על התוצאות של השלב הקודם כלומר, בכל תא בוחרים את המקסימום האפשרי מהערך של התא, כמות קודמת בשילוב עם סוגים אחרים, או סוג קודם באותה כמות.
לאחר שמילאנו את טבלאות F, X עושים שחזור, הערך האופטימלי יהיה בF[K][N] ואת הכמות מכל סוג נשחזר מטבלה X לפי האלגוריתם הבא:

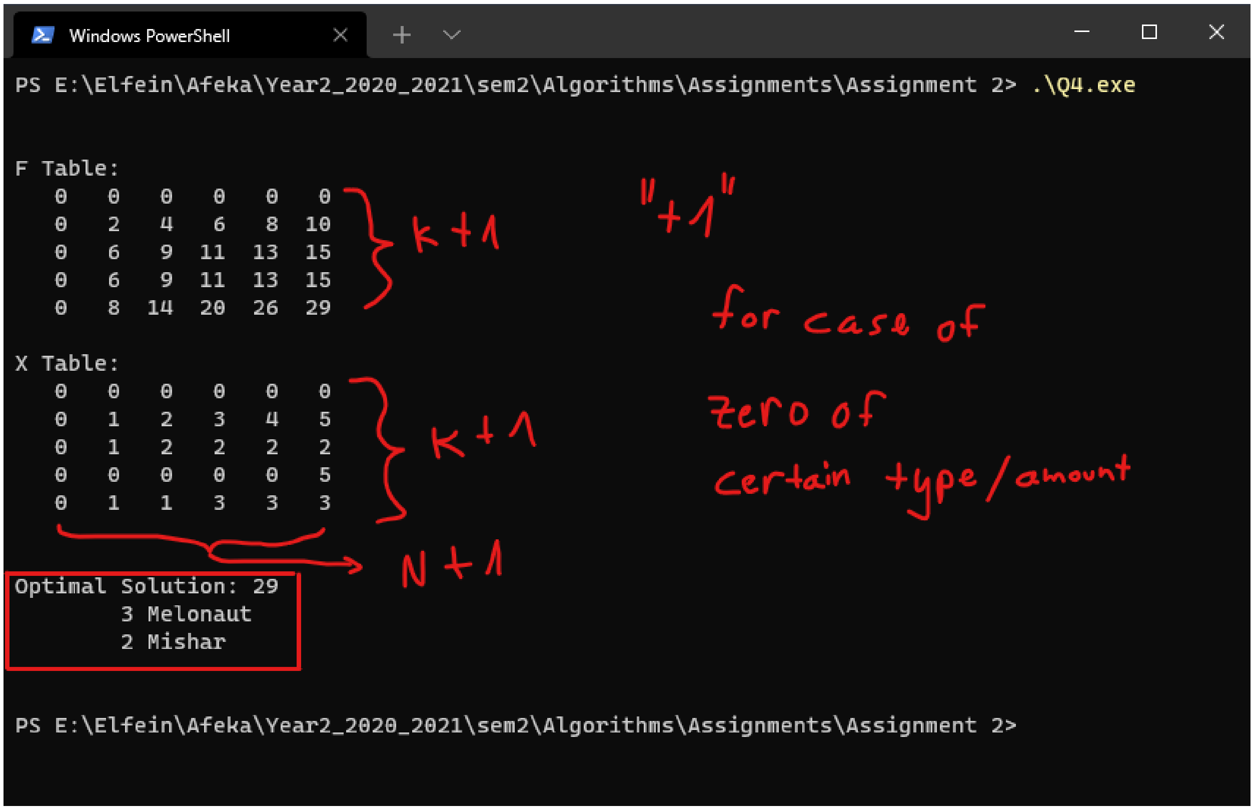
```
for (int k = K, t = k - 1, n = N, b_num; k > 0 && n > 0;
     k -= b_num, n -= b_num, t -= (b_num - 1)) {
    b_num = X[k][n];
    printf("\t%d %s\n", b_num, types + t * STR_LEN);
}
```

סה"כ כל האלגוריתם הכולל (בנוסף קובץ C מצורף "q4.c"):

```
#include <stdio.h>

# define STR_LEN 9

void print_matrix(int *mat, int n, int w) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            printf("%4d", *mat++);
        }
        puts("");
    }
    puts("");
}

int get_val(const int *arr, int row, int col, int row_size) {
    return *(arr + row * row_size + col);
}
```

```c
void q4(int *values, int K, int N, char *types) {
    int F[K + 1][N + 1];
    int X[K + 1][N + 1];

    for (int i = 0; i < K + 1; i++) {
        for (int j = 0; j < N + 1; j++) {
            F[i][j] = 0;
            X[i][j] = 0;
        }
    }

    int cell_value, mixed_value, last_total;

    for (int t = 1; t < K + 1; t++) {        // t for type
        for (int n = 1; n < N + 1; n++) {    // n of number
            last_total = mixed_value = 0;

            // current cell_value of this cell
            cell_value = get_val(values, t - 1, n, N + 1);
            if (t > 1) {
                if (n > 1) {
                    /*
                     * total cell_value of prev num of current type
                     *  + total cell_value remaining last type
                     */
                    mixed_value = get_val(values, t - 1, X[t][n - 1], N + 1)
                                  + F[t - 1][n - X[t][n - 1]];
                }
                // current total cell value for same number
                 // of buildings of all last types
                last_total = F[t - 1][n];
            }
            if (last_total > cell_value && last_total >= mixed_value) {
                F[t][n] = last_total;
                X[t][n] = 0;
            } else if (cell_value >= mixed_value && cell_value >= last_total) {
                F[t][n] = cell_value;
                X[t][n] = n;
            } else if (mixed_value > cell_value && mixed_value > last_total) {
                F[t][n] = mixed_value;
                X[t][n] = X[t][n - 1];
            }
        }
    }

    puts("\nF Table:");
    print_matrix((int *) F, K + 1, N + 1);
    puts("X Table:");
    print_matrix((int *) X, K + 1, N + 1);

    printf("\nOptimal Solution: %d\n", F[K][N]);
    for (int k = K, t = k - 1, n = N, b_num; k > 0 && n > 0;
         k -= b_num, n -= b_num, t -= (b_num - 1)) {
        b_num = X[k][n];
        printf("\t%d %s\n", b_num, types + t * STR_LEN);
    }
}
```

output של האלגוריתם + אלגוריתם שחזור:



```
PS E:\Elfein\Afeka\Year2_2020_2021\sem2\Algorithms\Assignments\Assignment 2> .\Q4.exe

F Table:
   0   0   0   0   0   0
   0   2   4   6   8  10
   0   6   9  11  13  15
   0   6   9  11  13  15
   0   8  14  20  26  29

X Table:
   0   0   0   0   0   0
   0   1   2   3   4   5
   0   1   2   2   2   2
   0   0   0   0   0   5
   0   1   1   3   3   3

Optional Solution: 29
        3 Melonaut
        2 Mishar

PS E:\Elfein\Afeka\Year2_2020_2021\sem2\Algorithms\Assignments\Assignment 2>
```


קיבלנו פתרון אופטימלי:

3 מלונאות, 2 מסחר עבור רווח כולל של 29.

.4

א.

1. נכניס את הקופסאות לרשימה
2. נמיין את הרשימה בסדר יורד לפי h
3. נבדוק עבור כל הקופסאות האם הגובה של הקופסא הנוכחית(i) גדול מהאורך של הקופסא הבאה(j):
   3.1. אם כן: i=j, j++
   3.2. אחרת: נוציא את המקום הבא(j) מהרשימה ו j++

ב. מיון: O(nlgn)

בדיקה: O(n)

סה"כ: O(nlgn)

ג.

נתון כי :

לכל קופסה מתקיים $h_i \le w_i \le d_i$.

תכניס קופסה b_i לתוך קופסה b_j רק כאשר מתקיים כי $d_i \le h_j$

נבין מהנתון כי: כדי שקופסא i תיכנס לתוך קופסא j כל צלעות i צריכות להיות קטנות מגובה של קופסא i

ולכן יהיה ניתן ל"שים" את הקופסא איך שנרצה בתוך הקופסא שתתאפשר(אורך/רוחב).

אז: לאחר שנסדר את כל הקופסאות בסדר יורד לפי h נבדוק אם אכן

הגובה של קופסא הנוכחית גדול מהאורך של הקופסא הבאה: hi>dj

כלומר להכניס את הקופסא בעלת הגובה המקסימלי שאפשר להכניס בכל פעם.

ד. נריץ:

-נכניס את כל הקופסאות למערך

-נמיין את המערך לפי h מהגדול לקטן (סדר יורד)

i	h	d	w
1	12	15	15
2	9	17	10
3	8	12	12
4	5	20	5
5	1	7	3

i=1,j=2;

ונעבור על המערך ונשאל:

האם h[1]>=d[2]

התשובה היא לא- לכן נוציא את קופסא 2 ונקדם את j

i=1,j=3

האם h[1]>=d[3]

התשובה היא כן לכן i=j , j++

I=3,j=4

האם h[3]>=d[4]

התשובה היא לא- לכן נוציא את קופסא 4 ונקדם את j

I=3,j=5

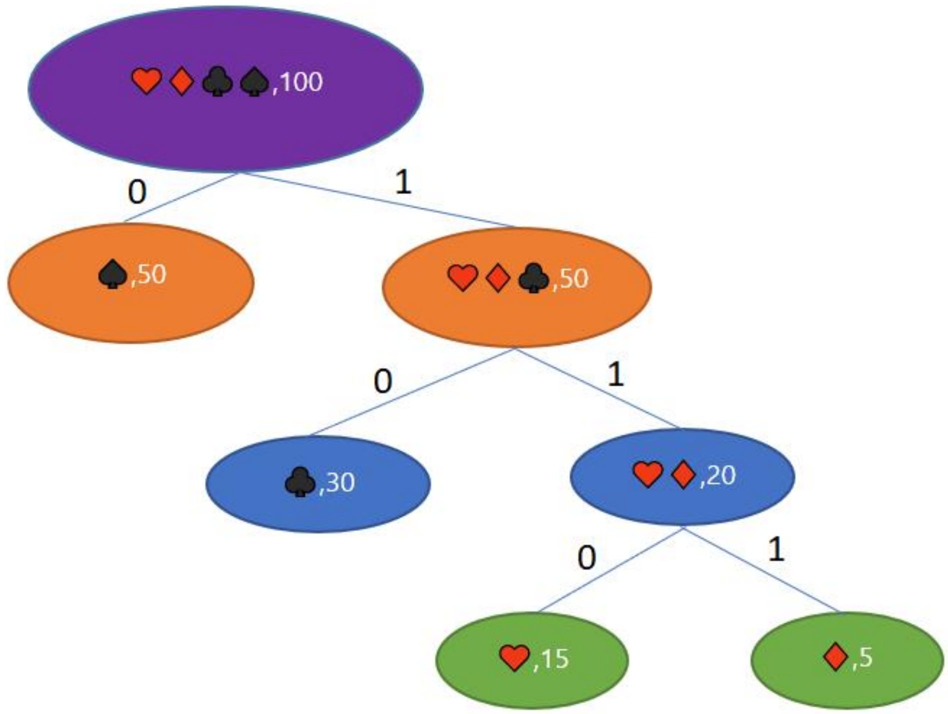
האם h[3]>=d[5]

התשובה היא כן והגענו לסוף המערך

לכן הוצאנו את קופסאות 2 ו 4, שבמקור היו 3 ו 4.

תוצאה סופית (לפי האינדקסים בטבלה המקורית):
2->1->5
ובחוץ קופאות 3,4

.5
א.



ב.

B = 1*50+30*2+15*3+5*3= 170 bits

ג.

♦ = 111

♥ = 110

♣ = 10

♠ = 0

ד. הוספת אפס משמאל שומרת על הייחודיות של כל סימן.

♦ = 0111

♥ = 0110

♣ = 0010

♠ = 0000

0111011000100000

NEW SIZE = B = (5+15+30+50)*4 = 400 bits

ה.

קידוד בינארי סטנדרטי עבור 4 אופציות...

♦ = 00

♥ = 01

♣ = 10

♠ = 11

B = (5+15+30+50) *2 = 200 bits