הגשה 2 - אלגוריתמים

שאלה 1

א. הקוד מצורף

	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	↖	↑	↖	↖	↑	↖	↖	↖	↑

11010

11011

ב. הקוד - מצורף Q1_b.c

ג. כן, התמ"א שהתקבל הוא - 1200122

ד. $X = 2120211221$, נקבל זאת על ידי כך שכל פעם כשבטלה הערך גדול לראשונה, נכניס את אותו מספר אל X

10	0	1	2	3	4	4	5	5	5	6	7	7	7	7	7
9	0	1	2	3	4	4	4	4	5	6	7	7	7	7	7

8	0	1	2	3	3	3	4	4	5	6	6	6	6	6	6
7	0	1	2	2	3	3	4	4	5	5	5	5	5	5	5
6	0	1	2	2	3	3	4	4	4	4	4	4	5	5	5
5	0	1	1	2	3	3	3	3	3	4	4	4	5	5	5
4	0	1	1	2	2	3	3	3	3	3	3	4	4	4	4
3	0	1	1	2	2	2	2	2	2	3	3	3	3	3	3
2	0	1	1	1	1	1	2	2	2	2	2	2	2	2	2
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***

$$\boldsymbol{Y = 1 \quad 1 \quad 2 \quad 2 \quad 0 \quad 1 \quad 0 \quad 1 \quad 2 \quad 2 \quad 0 \quad 2 \quad 0 \quad 0}$$

שאלה 2

$(8, d_0 = 6, d_1, d_2, d_3, d_4 = 30, d_5, \dots, d_8)$

א. ניתן למצוא את הסדר.

משום ש R(1,8)=3, ניתן לדעת את הסוגריים הראשוניים שנציב כדי לקבל פתרון מקסימלי. כלומר, צריכות לתחום שני קבוצות (1,3) ו (4,8)

$$M[1,3] = Min\left\{\begin{matrix} M[1,1] + M[2,3] + d_0 d_1 d_3 \\ M[1,2] + M[3,3] + d_0 d_2 d_3 \end{matrix}\right\}$$

8								0
7							0	72000
6						0	63000	68040
5					0	21000	81900	69720
4				0	420	1820	19820	21260
3			0	4800	1540	17820	34220	22540
2		0	480	660	942	2900	20840	21788
1	0	1440	**516**	876	1020	3536	21416	21872
	1	2	3	4	5	6	7	8

מינימלי ב M[1,1] ו M[2,1], נשאיר את הסימון

$$M[4,8] = Min\begin{Bmatrix} M[4,4] + M[5,8] + d_3 * d_4 * d_8 \\ M[4,5] + M[6,8] + d_3 * d_5 * d_8 \\ M[4,6] + M[7,8] + d_3 * d_6 * d_8 \\ M[4,7] + M[8,8] + d_3 * d_7 * d_8 \end{Bmatrix}$$

8								0
7							0	72000
6						0	63000	68040
5					0	21000	81900	69720
4				0	420	1820	19820	**21260**
3			0	4800	1540	17820	34220	22540
2		0	480	660	942	2900	20840	21788
1	0	1440	516	876	1020	3536	21416	21872
	1	2	3	4	5	6	7	8

מינימלי ב M[8,8] ו M[7,8], נשאיר את הסימון

$$M[4,7] = Min\begin{Bmatrix} M[4,4] + M[5,7] + d_3 * d_4 * d_7 \\ M[4,5] + M[6,7] + d_3 * d_5 * d_7 \\ M[4,6] + M[7,7] + d_3 * d_6 * d_7 \end{Bmatrix}$$

8								0
7							0	72000
6						0	63000	68040
5					0	21000	81900	69720
4				0	420	1820	**19820**	21260
3			0	4800	1540	17820	34220	22540
2		0	480	660	942	2900	20840	21788
1	0	1440	516	876	1020	3536	21416	21872
	1	2	3	4	5	6	7	8

מינימלי ב M[4,5] ו M[6,6]

הצבנו את כל הסוגריים ונקבל $((1,1)(2,3))((((4,5)(6,6))(7,7))(8,8))$

ב. ניתן לשחזר את ווקטור הנתונים על ידי יצירת מערכת משוואות של האלכסון השני (i, i-1)

$M(0,2) = 6 * d_1 d_2 = 1440$

$M(2,3) = d_1 d_2 d_3 = 480$

$M(3,4) = d_2 d_3 * 30 = 4800 \rightarrow d_2 d_3 * 3 = 480 \rightarrow d_1 = 3$

$6 * 3 * d_2 = 1440 \rightarrow d_2 = 80$

$3 * 80 * d_3 = 480 \rightarrow d_3 = 2$

$M(4,5) = d_3 d_4 d_5 = 420$

$2 * 30 * d_5 = 420 \rightarrow d_5 = 7$

$M(5,6) = d_4 d_5 d_6 = 30 * 7 * d_6 = 21000 \rightarrow d_6 = 100$

$M(6,7) = d_5 d_6 d_7 = 7 * 100 * d_7 = 63000 \rightarrow d_7 = 90$

$M(7,8) = d_6 d_7 d_8 = 100 * 90 * d_8 = 72000 \rightarrow d_8 = 8$

$$Final\ Result\ \langle 6,3,80,2,30,7,100,90,8 \rangle$$

שאלה 3

קיבולת האונייה $W = 5.5$

Items	a_1	a_2	a_3	a_4	a_5
Weight	0.5	2	2.5	1	2
Value	8	16	28	14	15

א. פתרון אופטימלי (כאשר קיים פריט אחד מכל סוג)

	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 את המעבר על הטבלה בתא הגדול ביותר [5,5.5] - אין מקום לa_5 ולא יכנס.
2. נעבור לתא [4,5.5] - הפריט הרביעי יכנס - נחסיר את משקלו ונבדוק עבור פריטים $a_1 a_2 a_3$.
3. נעבור לתא [3,4.5] - הפריט השלישי יכנס - נחסיר את משקלו ונבדוק עבור $a_1 a_2$.

4. נעבור לתא [2,2] - הפריט השני יכנס - נחסיר את משקלו מהמשקל הכולל ונראה שלא נותר לנו משקל להעמסה, לכן מצאנו את כל האיברים לקבלת הפתרון האופטימלי.

האיברים שנכנסו הם a_2, a_3, a_4 וערכם הכולל הוא 58.

ב. הפתרון יגדל ב 14 מפני שיהיה מקום גם למוצרים $a_1 + a_2$, סה"כ הרווח יהיה 81

	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6	6.5	7	7.5	8
0	0	0	0	0	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	8Y	8Y	8Y	8Y	8Y
2	0	8N	8N	8N	16Y	24Y	24Y	24Y	24Y	24Y	24Y	24Y	24Y	24Y	24Y	24Y	24Y
3	0	8N	8N	8N	16N	28Y	36Y	36Y	36Y	44Y	52Y	52Y	52Y	52Y	52Y	52Y	52Y
4	0	8N	14Y	22Y	22Y	28N	36N	42Y	50Y	50Y	52N	58Y	66Y	66Y	66Y	66Y	66Y
5	0	8N	14N	22N	22N	28N	36N	42N	50N	50N	52N	58N	66N	66Y	67Y	73Y	81Y

ג.

Items	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	16	8	11.2	14	7.5

ניתן לראות ש a_1 הכי רווחי עבור משקל של 1, לכן אם אין הגבלה על כמות המוצרים נעדיף לקחת 11 פעמים את a_1, $W_{total} = 5.5, a_1 = 0.5 \to \frac{5.5}{0.5} = 11$

ד.

ה.

```
#define MAX(a, b) (((a)>(b)) ? (a) : (b))

typedef struct Tables {
    int **T;
    char **S;
    int **X;
} Q3_Tables;

void freeTables(Q3_Tables* tables, int n) {
    for (int i = 0; i < n + 1; i++) {
        free(tables->T[i]);
    }
    free(tables->T);
    for (int i = 0; i < n + 1; i++) {
```

```c
        free(tables->S[i]);
    }
    free(tables->S);
}

void initTables(Q3_Tables *tables, int n, int W) {
    tables->T = calloc(n + 1, sizeof(int *));
    if (!tables->T) {
        printf("Error in alloc memory\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < n + 1; i++) {
        tables->T[i] = calloc(W + 1, sizeof(int));
        if (!tables->T[i]) {
            printf("Error in alloc memory\n");
            exit(EXIT_FAILURE);
        }
    }

    tables->S = calloc(n + 1, sizeof(char *));
    if (!tables->S) {
        printf("Error in alloc memory\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < n + 1; i++) {
        tables->S[i] = calloc(W + 1, sizeof(char));
        if (!tables->S[i]) {
            printf("Error in alloc memory\n");
            exit(EXIT_FAILURE);
        }
    }
}

void printS_Table(char **S, int n, int w) {
    printf("S table:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            if (S[i][j] == 0)
                printf(" %5d ", S[i][j]);
            else
                printf(" %5c ", S[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void printT_Table(int **T, int n, int w) {
    printf("T table:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            printf(" %5d ", T[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

```c
void Q3D(int items, int *values, int *weights, int W) {
    Q3_Tables* tables = malloc(sizeof(Q3_Tables));
    initTables(tables, items, W);

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

    printf("Max Result: %d\n", tables->T[items][W]);

    freeTables(tables, items);
    free(tables);
}
```

(קובץ ההרצה מצורף תחת השם *Q3_d*)

ו.

```c
#include "stdio.h"
#include "stdlib.h"

#define MAX(a, b) (((a)>(b)) ? (a) : (b))

typedef struct Tables {
    int **T;
    int **S;
    int *X;
} Q3_Tables;

void freeTables(Q3_Tables* tables, int n) {
    for (int i = 0; i < n + 1; i++) {
        free(tables->T[i]);
    }
    free(tables->T);
    for (int i = 0; i < n + 1; i++) {
        free(tables->S[i]);
    }
    free(tables->S);
}

void initTables(Q3_Tables *tables, int n, int W) {
```

```c
    tables->T = calloc(n + 1, sizeof(int *));
    if (!tables->T) {
        printf("Error in alloc memory\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < n + 1; i++) {
        tables->T[i] = calloc(W + 1, sizeof(int));
        if (!tables->T[i]) {
            printf("Error in alloc memory\n");
            exit(EXIT_FAILURE);
        }
    }

    tables->S = calloc(n + 1, sizeof(int *));
    if (!tables->S) {
        printf("Error in alloc memory\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < n + 1; i++) {
        tables->S[i] = calloc(W + 1, sizeof(int));
        if (!tables->S[i]) {
            printf("Error in alloc memory\n");
            exit(EXIT_FAILURE);
        }
    }
}

void print2D_Table(int **T, int n, int w, char* tableName) {
    printf("%s table:\n", tableName);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            printf(" %5d ", T[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void printX_Table(int *T, 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 ", T[i]);;
    printf("\n");
}

void Q3E(int items, int *values, int *weights, int W) {
    Q3_Tables tables;
    initTables(&tables, items, W);

    tables.X = calloc(items, sizeof(int *));
    if (!tables.X) {
        printf("Error in alloc memory\n");
        exit(EXIT_FAILURE);
    }
```

```
    for (int i = 1; i <= items; i++) {
        for (int j = 1; j <= W; j++) {
            if (weights[i - 1] <= j) {
                if (j - weights[i - 1] >= 0) {
                    tables.T[i][j] = MAX(tables.T[i -
1][j],tables.T[i][j - weights[i - 1]] + values[i - 1]);
                } else {
                    tables.T[i][j] = MAX(tables.T[i - 1][j],values[i -
1]);
                }


            }
            else tables.T[i][j] = tables.T[i - 1][j];


            if (tables.T[i][j] !=  tables.T[i - 1][j]) {
                tables.X[i - 1]++;
                tables.S[i][j] = 1 + tables.S[i][j - weights[i - 1]];
            }

        }
    }

    print2D_Table(tables.T, items + 1, W + 1, "T");
    print2D_Table(tables.S, items + 1, W + 1, "S");
    printX_Table(tables.X, items);

    printf("\nMax Result: %d\n", tables.T[items][W]);

    while (items != 0) {
        if (tables.T[items][W] != tables.T[items - 1][W]) {
            printf("\tPackage %d with W = %d and value = %d\n", items,
weights[items - 1], values[items - 1]);
            W -= weights[items - 1];
        }
        items--;
    }

    freeTables(&tables, items);
}

int main() {
    // replace values and weights with dynamic arrays and then free
them
    int values[] = {16, 32, 56, 28, 30};
    int weights[] = {1, 4, 5, 2, 4};
    int W = 11;
    int items = sizeof(values) / sizeof(values[0]);
    Q3E(items, values, weights, W);
    return 1;
}
```

(קובץ ההרצה מצורף תחת השם *Q3_e*)

שאלה 4

מספר מבנים	מגורים	מסחר	משרדים	מלונאות
0	0	0	0	0
1	2	6	1	8
2	4	9	1	12
3	6	9	2	20
4	8	10	3	16
5	10	11	15	12

```
typedef struct Tables {
    int **T;
    int **X;
} Q4_Tables;

void freeTablesQ4(Q4_Tables *tables, int n) {
    // frees all dynamically allocated tables
    for (int i = 0; i < n + 1; i++) {
        free(tables->T[i]);
    }
    free(tables->T);
    for (int i = 0; i < n + 1; i++) {
        free(tables->X[i]);
    }
    free(tables->X);
}

void initTablesQ4(Q4_Tables *tables, int n, int W) {
    // allocating all tables
    tables->T = calloc(n + 1, sizeof(int *));
    if (!tables->T) {
        printf("Error in alloc memory\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < n + 1; i++) {
        tables->T[i] = calloc(W, sizeof(int));
        if (!tables->T[i]) {
            printf("Error in alloc memory\n");
            exit(EXIT_FAILURE);
        }
    }

    tables->X = calloc(n + 1, sizeof(int *));
    if (!tables->X) {
        printf("Error in alloc memory\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < n + 1; i++) {
        tables->X[i] = calloc(W, sizeof(int));
        if (!tables->X[i]) {
            printf("Error in alloc memory\n");
```

```
            exit(EXIT_FAILURE);
        }
    }
}

void printX_TableQ4(int **X, int n, int w) {
    // printing S table
    printf("X table:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            printf(" %5d ", X[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void printT_TableQ4(int **T, int n, int w) {
    // printing T table
    printf("T table:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < w; j++) {
            printf(" %5d ", T[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void Q4(int values[4][6], int K, int N, char typesNames[4][10]) {
    Q4_Tables tables;
    initTablesQ4(&tables, K, N);
    int x, y, z;
    bool zeroBuildingTypes, jBuildingTypes, singleBuildingType;

    for (int i = 1; i < K + 1; i++) {
        for (int j = 1; j < N; j++) {
            y = 0;
            z = 0;
            x = values[i - 1][j];
            if (i > 1) {
                if (j > 1) {
                    y = values[i - 1][tables.X[i][j - 1]] + tables.T[i
- 1][j - tables.X[i][j - 1]];
                }
                z = tables.T[i - 1][j];
            }
            zeroBuildingTypes = z > x && z >= y;
            jBuildingTypes = y > x && y > z;
            singleBuildingType = x >= y && x >= z;
            if (zeroBuildingTypes) {
                tables.T[i][j] = z;
                tables.X[i][j] = 0;
            }
            if (jBuildingTypes) {
                tables.T[i][j] = y;
                tables.X[i][j] = tables.X[i][j - 1];
```

```
            }
            if (singleBuildingType) {
                tables.T[i][j] = x;
                tables.X[i][j] = j;
            }
        }
    }
    printT_TableQ4(tables.T, K + 1, N);
    printX_TableQ4(tables.X, K + 1, N);

    int i = K;
    int m = i - 1;
    int j = N-1;
    int amount;
    while(i > 0 && j > 0) {
        amount = tables.X[i][j];
        printf("%d %s\n", amount, typesNames[m]);
        i -= amount;
        j -= amount;
        m = m - amount + 1;
    }

    printf("\nMax Result: %d\n", tables.T[K][N - 1]);
    freeTablesQ4(&tables, K);
}
```

(קובץ ההרצה מצורף תחת השם Q4)

א. *values* - טבלה המכילה את הנתון כאשר השורות מייצגות את **מספר הסוגים השונים של המגורים(*K*)** כאשר הסוג הראשון הוא טיפוס "מגורים" לאחר מכן "מסחר" וכו ...
מספר הטור מייצג את **מספר המבנים שמתוכם עלינו לבחור (*N*)**
T – טבלת הסכומים האפשריים עבורה בגודל *(K+1)xN* כאשר הרווח המקסימלי נמצא בתא *[K,N-1]*
X - טבלה בה שומרים לכל *[i,j]* את מספר המופעים של אותו טיפוס בניין שבעזרתו הגענו לרווח מקסימלי ב *T[K,N-1]*
(הטבלאות *T* ו *X* באותו גודל)
typesNames - מערך שמות כל סוגי המגורים. המערך בגודל *K*

<u>Restrictions</u>

Values: $0 \le i \le K-1$ *and* $0 \le j \le N-1$
T and X: $0 \le i \le K$ *and* $0 \le j \le N-1$ *and **must be initialize 0 for each cell before starting the algorithm***

<u>Algorithm</u>

Zero Building Types Picked -> T[i,j] = T[i-1,j] and X[I,j] = 0
Single Building Type Picked -> T[i,j] = values[i-1,j] and X[I,j] = j
Multiple Building Type Picked -> T[i,j] = values[i-1, X[i,j-1] + T[i-1, X[i,j-1]]

<u>שיחזור פתרון</u>

i = K
j = N – 1

```
m = i - 1
While i > 0 and j > 0 do
        Amount = X[i,j]
        Print("Amount" + "typesNames[m]")
        i -= Amount
        j -= Amount
        m = m – Amount + 1
```

ב.

T table:

0	0	0	0	0	0	0
1	0	2	4	6	8	10
2	0	6	9	11	13	15
3	0	6	9	11	13	15
4	0	8	14	20	26	29
	0	**1**	**2**	**3**	**4**	**5**

S table:

0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	1	2	2	2	2
3	0	0	0	0	0	5
4	0	1	1	3	3	3

Max Result: 29

לפי אלגוריתם שיחזור פתרון נראה כי נקבל 3 מלונאות ו 2 מסחר.

שאלה 5

א. האלגוריתם

נגדיר מבנה

$w, h, d = box\ dimensions$

next = pointer to next node

1. נכניס את כל הקופסאות למבני של רשימה מקושרת
2. נמיין את הרשימה בסדר יורד לפי h
3. נשמור מצביעים לקופסה הראשונה (הגדולה ביותר לפי h) p_curr, ולקופסה אחריו p_next.
4. מהקופסה הראשונה (p_curr) ברשימה ועד האחרונה נבדוק:
   4.1. אם הקופסה הנוכחית (p_curr) קטנה מהקופסה הבאה (p_next)
      4.1.1. אם כן - נוציא את p_next מהרשימה, ונקדם אותו לקופסה הבאה
      4.1.2. אחרת - נוציא את הקופסה הראשונה ברשימה, נכניס את p_next אל p_curr ונקדם את שני המצביעים

ב. מיון האיברים - $O(nlogn)$
הלולאה שעוברת על כל האיברים - $O(n)$
לכן נקבל זמן ריצה של $O(nlogn)$

ג. נתון כי עבור כל קופסה j מתקיים כי $h_j \leq w_j \leq d_j$. לכן אם נרצה להכניס את קופסה i לתוך קופסה j הפונקציה תבדוק האם $d_i \leq h_j$, כלומר תבדוק שאכן מתקיים $h_i \leq w_i \leq d_i \leq h_j \leq w_i \leq d_i$.
ניקח קופסה i בעלת גובה מקסימלי, מכן עבור כל קופסה נותרת j המקיימת $d_j \leq h_i$ מתקיים כי $w_i \leq d_j \leq w_i \leq d_i$, לכן כל קופסה כזאת ניתן להכניס ל i.

על ידי בחירת h_i מקסימלי, ניתן להבטיח כי מספר הקופסאות המקיימות $d_j \leq h_i$ יהיה מקסימלי בכל איטרציה.

ד. הרצת האלגוריתם

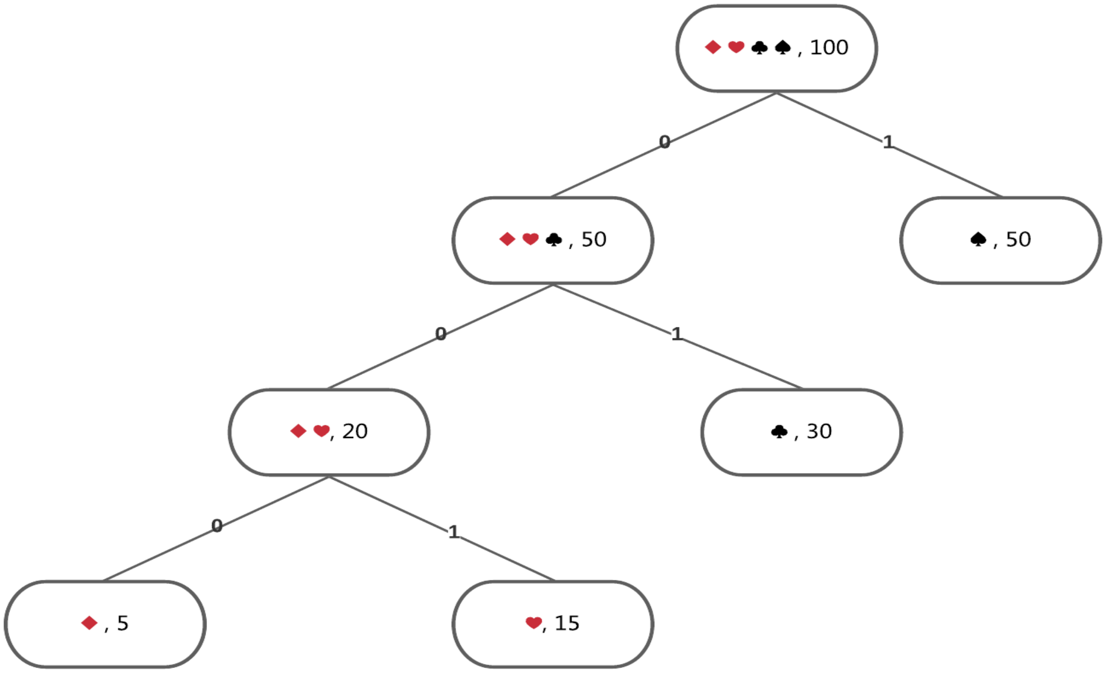
1. נבחר את הקופסה בעלת הגובה המקסימלי - קופסה 5.
2. קופסאות 3 ו-4 ניתן לשלול מכיוון שהאורך שלהם גדול מהגובה של קופסה 5, $d_3 = 17, d_4 = 20 > h_5 = 12$.
3. נבחר בקופסה 1 מכיוון שאורכה שווה לגובה של קופסה 5, $d_1 = h_5 = 12$.
4. נותרה רק קופסה 2 שמקיימת את התנאי - אורכה קטן מגובהה של קופסה 1, $d_2 = 7 < 8 = h_1$, לכן נכניס אותה.

לכן הסדר שנקבל הוא: קופסה 2 בתוך קופסה 1 שבתוך קופסה 5.

קופסאות 3 ו-4 יישארו בחוץ

שאלה 6

א.



ב. $B = \sum_{c=1}^{4} f(c) d_T(i) = 5 * 3 + 15 * 3 + 30 * 2 + 50 * 1 = 170\ bits$

ג.

♥ = 001

♦ = 000

♣ = 01

♠ = 1

ד.

♥ = 0010

♦ = 0000

♣ = 0100

♠ = 1000

$B = (30 + 50 + 15 + 5) * 4 = 400\ bits$

ה.

♥ = 00

♦ = 11

♣ = 01

♠ = 10

$B = \sum_{c=1}^{4} f(c) d_T(i) = 5 * 2 + 15 * 2 + 30 * 2 + 50 * 2 = 200 \text{ bits}$