Tema 2 AA

Stavăr Laurențiu-Cristian, Grupa 322CC

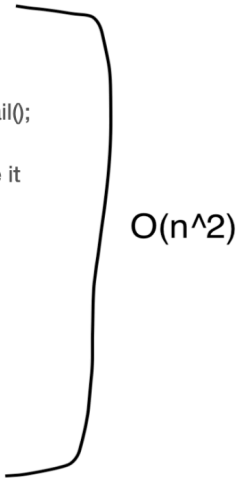
Problema 1

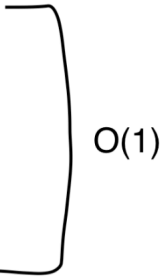
a)

```
void chess(int a[][], int n, int m) {
    // Search the starting point and count the number of empty cells(available cells)
    current_i = -1, current_j = -1, empty_cells = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++) {
            if (a[i][j] == 2) {
                current_i = i;
                current_j = j;
            } else if (a[i][j] == 1) {
                empty_cells++;
            }
        }

    // Define the possible moves as "how much I should add/subtract to/from
    // the current position indexes in order to make a move"
    possible_moves =
            [[1,1], [-1, 1], [1, -1], [-1, -1], [2, 2], [-2, 2], [2, -2], [-2, -2],
            [1, 2], [-1, 2], [1, -2], [-1, -2], [2, 1], [-2, 1], [2, -1], [-2, -1]];
```

O(n*m) = O(n^2)

```
// Initialize the list of visited cells and add the starting point to it
visited = [];
visited.append([current_i, current_j]);

// While the current cell is not the ending point, make a move
while (a[current_i][current_j] != 3) {
    // If the current cell is empty or if it was already visited then fail
    if (a[current_i][current_j] == 0 || visited.contains([current_i, current_j])) fail();

    // Choose a random move from the possible moves and actually make it
    next_move = choice(possible_moves);
    current_i += next_move[0];
    current_j += next_move[1];

    // If the new position is out of the board then fail
    if (current_i < 0 || current_i >= n || current_j < 0 || current_j >= m) fail();

    // Add the new cell to the list of visited cells
    visited.append([current_i, current_j]);

}

// Now the current cell is the ending point
// If all the empty cells were visited ONCE then success, otherwise fail
// In order to succeed, the number of visited cells should be equal to the
// number of empty cells plus the starting and ending points
if (visited.size == empty_cells + 2)
    success();
else
    fail();
}
```





Complexitate: O(n^2), unde n = m = dimensiunea tablei de sah

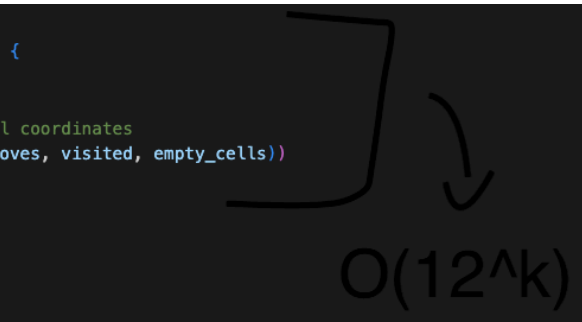
b)

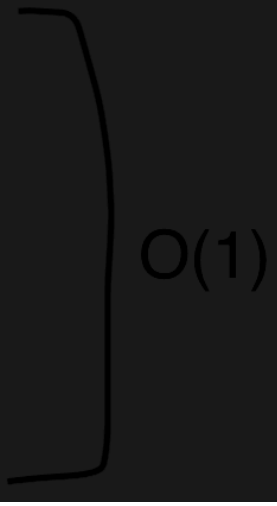
```c
#include <stdio.h>

int count_visited(int visited[][8]) {
    int count = 0;
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (visited[i][j] == 1)
                count++;
        }
    }

    return count;
}

int solve_recursive(int a[][8], int current_i, int current_j, int possible_moves[][2],
                    int visited[][8], int empty_cells) {

    // Check if the current cell is the end cell
    if (a[current_i][current_j] == 3) {
        // Check if all the empty cells were visited ONCE (plus the starting cell)
        if (count_visited(visited) == empty_cells + 1) {
            return 1;
        }
        else
            return 0;
    }

    // Check if the current cell is empty
    if (visited[current_i][current_j] == 1)
        return 0;

    // Add the current cell to the visited list
    visited[current_i][current_j] = 1;

    // For each possible move
    for (int i = 0; i < 16; i++) {
        // Compute the new cell coordinates
        int new_i = current_i + possible_moves[i][0];
        int new_j = current_j + possible_moves[i][1];
```

O(1)

O(1)

```
        // Chech if the new cell is inside the board
        if (new_i >= 0 && new_i < 8 && new_j >= 0 && new_j < 8) {
            // Check if the new cell is a valid move
            if (a[new_i][new_j] == 1 || a[new_i][new_j] == 3) {
                // Call the recursive function with the new cell coordinates
                if (solve_recursive(a, new_i, new_j, possible_moves, visited, empty_cells))
                    return 1;
            }
        }
    }

    // Remove the current cell from the visited list
    visited[current_i][current_j] = 0;

    // Return false if no solution was found
    return 0;
}

int chess_recursive(int a[][8]) {
    // Find the starting cell and
    int start_i = -1, start_j = -1, empty_cells = 0;
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (a[i][j] == 2) {
                start_i = i;
                start_j = j;
            } else if (a[i][j] == 1) {
                empty_cells++;
            }
        }
    }

    // If no starting cell was found, return false
    if (start_i == -1 || start_j == -1) {
        return 0;
    }

```





```
80      // Create the posible moves list
81      int possible_moves[12][2] = {
82              {1, 1},
83              {-1, 1},
84              {1, -1},
85              {-1, -1},
86              {1, 2},
87              {-1, 2},
88              {1, -2},
89              {-1, -2},
90              {2, 1},
91              {-2, 1},
92              {2, -1},
93              {-2, -1}
94      };
95
96      // Create the visited list and add the starting cell
97      int visited[][8] = {
98          {0, 0, 0, 0, 0, 0, 0, 0},
99          {0, 0, 0, 0, 0, 0, 0, 0},
100         {0, 0, 0, 0, 0, 0, 0, 0},
101         {0, 0, 0, 0, 0, 0, 0, 0},
102         {0, 0, 0, 0, 0, 0, 0, 0},
103         {0, 0, 0, 0, 0, 0, 0, 0},
104         {0, 0, 0, 0, 0, 0, 0, 0},
105         {0, 0, 0, 0, 0, 0, 0, 0}
106     };
107
108     // Call the recursive solver function and return its result
109     return solve_recursive(a, start_i, start_j, possible_moves, visited, empty_cells);
110 }
```

Am oferit și două exemple de inputuri și am rulat algoritmul pe aceste exemple pentru verificare.

```c
int main() {

    // Has solution
    int a[8][8] = {
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 2, 1, 1, 0, 0, 0, 0},
        {0, 1, 1, 0, 0, 0, 0, 0},
        {0, 1, 1, 1, 0, 0, 0, 0},
        {0, 1, 1, 1, 0, 0, 0, 0},
        {0, 3, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0}
    };

    // No solution
    int b[8][8] = {
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 2, 0, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 0, 0, 3, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0}
    };

    if (chess_recursive(a)) {
        printf("The chess problem has a solution\n");
    } else {
        printf("The chess problem has no solution\n");
    }

    return 0;
}
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
The chess problem has a solution
laur@LaurentiusMBP2 Problema1.2 % cd "/Users/laur/Desktop/Tema2_AA/Problema1/Problema1.2/" && gcc problema1.2.c -o problema1.2 && "/Users
1.2
ld: warning: ignoring duplicate libraries: '-lgcc'
The chess problem has a solution
laur@LaurentiusMBP2 Problema1.2 %
```

```c
int main() {

    // Has solution
    int a[8][8] = {
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 2, 1, 1, 0, 0, 0, 0},
        {0, 1, 1, 0, 0, 0, 0, 0},
        {0, 1, 1, 1, 0, 0, 0, 0},
        {0, 1, 1, 1, 0, 0, 0, 0},
        {0, 3, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0}
    };

    // No solution
    int b[8][8] = {
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 2, 0, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 0, 0, 3, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0}
    };

    if (chess_recursive(b)) {
        printf("The chess problem has a solution\n");
    } else {
        printf("The chess problem has no solution\n");
    }

    return 0;
}
```

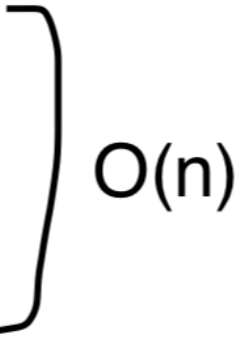
```
PROBLEMS  OUTPUT  TERMINAL  PORTS

TERMINAL

laur@LaurentiusMBP2 Problema1.2 % cd "/Users/laur/Desktop/Tema2_AA/Problema1/Problema1.2/" && gcc problema1.2.c -o problema1.2 &&
1.2
ld: warning: ignoring duplicate libraries: '-lgcc'
The chess problem has no solution
laur@LaurentiusMBP2 Problema1.2 %
```

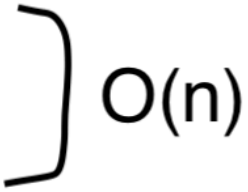
Complexitate: O(12^k). Menționez că am considerat, pentru acest subpunct, că tabla de șah poate fi doar de dimensiuni 8x8 deci astfel parcurgerea întregii table de șah este făcută în timp constant O(8x8) = O(1)

Problema 2

a)

```
TSP(V, E) {
    // Generate
    road = [];
    for (i = 1:V.size) {
        X = choice(V);

        if (road.contains(X))
            fail();
        else
            road.append(X);
    }

    // Verify
    for (i = 0:V.size - 2) {
        if (!E.contains(road[i], road[i+1]))
            fail();
    }

    if (!E.contains(road[V.size - 1], road[0]))



        fail();

    success();
}
```





Complexitate: O(n)

b)

```c
#define MAX 1001

typedef struct {
    int n;
    int matrix[MAX][MAX];
} Graph;

int TSP(Graph *g) {
    int cycle[MAX], current = 1;

    // Find a cycle of length 3
    for (int i = 1; i <= g->n; i++)
        for (int j = 1; j <= g->n; j++)
            for (int k = 1; k <= g->n; k++)
                if (g->matrix[i][j] && g->matrix[j][k] && g->matrix[k][i]) {
                    cycle[current++] = i;
                    cycle[current++] = j;
                    cycle[current++] = k;
                    goto skip;
                }

skip:

    // Find the rest of the cycle
    for (int i = 1; i <= g->n; i++)
        if (!cycle_contains(cycle, i)) {
            for (int j = 1; j <= current; j++)
                if (g->matrix[i][cycle[j]] && g->matrix[cycle[(j + 1) % g->n]][i]) {
                    // Add i to cycle
                    for (int k = current; k > j; k--)
                        cycle[k] = cycle[k - 1];
                    cycle[j] = i;
                    current++;
                    break;
                }
        }

    // Check if the cycle is hamiltonian
    if (current == g->n + 1)
        return 1;

    return 0;
}
```

O(n^3)

O(n^3)

Complexitate: O(n^3)

Problema 3

Stoian Leonția

③ Fie un termen de forma $X_1 \vee \bar{X}_2 \vee X_3$, acesta poate fi rescris drept $X_1 + (\overbrace{1-X_2}^{X_2'}) + X_3 \geq 1$. Se obține astfel o inegalitate pe care o putem transpune în egalități adăugând încă două variabile $a, b \in \{0, 1\}$, astfel încât să putem aplica inegalitatea. Fie n = numărul variabilelor, m = numărul termenilor

Fie termenii lui E - $X_1, \ldots, X_n$
- $a_1, \ldots, a_m$
- $b_1, \ldots, b_m$

```
- 3_SAT(E, n, m)
    E = [];
  - for i = 1:n
    E.add("E.x(i)*E.x(i) - E.x(i) = 0");
    int K[3];
  - for i = 1, m
    - for j = 1,3
      - if E.x[j] ≥ 0 // Considerăm că variabilele negate sunt returnate drept numere negative
          K[j] = E.x[j];
        else
          K[j] = 1 + E.x[j];
    E.add("K[0] + K[1] + K[2] + a[i] + b[i] = 3");
    E.add("a[i]^2 - a[i] = 0");
    E.add("b[i]^2 - b[i] = 0");
  Evaluate(E); // Eval(E)
```

E este rezolvabilă $\Leftrightarrow$ Sistemul de ecuații are o soluție

- "$\Rightarrow$": E este rezolvabilă $\Rightarrow$ Sistemul de ecuații are o soluție

  $\hookrightarrow \exists x_1, x_2, \dots, x_m \in \{0,1\}$ care satisfac $E \Rightarrow$

  $\Rightarrow$ ele satisfac toate ecuațiile de forma $x_i^2 - x_i = 0$

Pentru fiecare ecuație de forma $x[0] + x[1] + x[2] + a[i] + b[i] = 3$, unde $x[0], x[1], x[2]$ sunt literalii unui termen de aici $\Rightarrow$ $x[0] \vee x[1] \vee x[2] = 1$

$\Leftrightarrow x[0] + x[1] + x[2] \geq 1$ și $x[0] + x[1] + x[2] \geq 3$

Cum se pot atribui valori lui $a[i]$ și $b[i] \Rightarrow$ se asigură că ecuația este satisfăcută

Astfel, sistemul de ecuații este satisfăcut.

- "$\Leftarrow$": Sistemul de ecuații are o soluție $\Rightarrow$ E rezolvabilă

Sistemul de ecuații are soluții $\Rightarrow$ [illegible]

$\Rightarrow x_1, \dots, x_m; a_1, \dots, a_m; b_1, \dots, b_m$ are un set de valori ce satisfac E

Cum avem ecuațiile de forma $x_i^2 - x_i = 0$ pentru $i = \overline{1,m} \Rightarrow$ Setul de valori este $\{0,1\}$, iar cum $x_i + \bar{x}_i = 1 \Rightarrow$ Pentru fiecare $i \in [1,m]$, fie un literal de forma $x_0 \vee x_1 \vee x_2$, obț din ecuația corespunzătoare $x_0 + x_1 + x_2 \geq 1 \Leftrightarrow x_0 \vee x_1 \vee x_2 = 1$

$\Rightarrow$ E este rezolvabilă

- Din "$\Rightarrow$" și "$\Leftarrow$" $\Rightarrow$ problema 3_SAT se poate reduce la problema ecuații

Sistemul devine:

$$\begin{cases} x_i + x_j + x_k + a_i + b_i = 3 \rightarrow m \text{ ecuații} \\ x_i^2 - x_i = 0 \rightarrow \text{pentru } i = 1:m \\ x_i + x_i' = 1 \rightarrow \text{pentru } i = 1:m \\ a_i^2 + a_i = 0 \rightarrow \text{pentru } i = 1:m \\ b_i^2 - b_i = 0 \rightarrow \text{pentru } i = 1:m \end{cases}$$

Astfel, sunt $3m+2m$ ecuații cu $2m+2m$ necunoscute $(x_1, \dots, x_m)(x_1', \dots, x_m')$ $(a_1, \dots, a_m)(b_1, \dots, b_m)$

Dacă formula din 3_SAT este satisfiabilă $\Rightarrow x_i \in \{0,1\} \Rightarrow x_i^2 - x_i = 0$ este îndeplinită pentru $i=1:m \Rightarrow x_i' = 1 - x_i$

$x_i \vee x_j \vee x_k = 1$ pentru toate ecuațiile $\Rightarrow x_i + x_j + x_k \geq 1 \Rightarrow a_i \neq b_i$ pt că sunt diferite de 0 sau 1 $\Rightarrow$ Sistemul are soluții

Dacă sistemul are soluții atunci $x_i \in \{0,1\}$; $x_i + x_i' = 1$; $a_i, b_i \in \{0,1\}$

$\Rightarrow x_i + x_j + x_k \geq 1 \Rightarrow x_i \vee x_j \vee x_k = 1$ deci se poate alege cel puțin o valoare pt fiecare ecuație pentru cel puțin o variabilă pentru 3_SAT