Wstęp do Informatyki i Programowania

Laboratorium nr 6
17, 18 19 grudnia i 21, 22, 23 stycznia

Przeczytaj opis problemu 8 hetmanów (ang. 8 queens). Zauważ, że rozwiązanie problemu w postaci listy pozycji w kolejnych kolumnach szachownicy musi być permutacją wierszy (z warunku niebicia w poziomie i pionie).

Zadanie 1 (6 pkt)

Napisz w języku C program, który rozwiązuje problem n hetmanów następującym algorytmem:

> *Generujemy kolejne n elementowe permutacje (funkcja z listy na ćwiczeniach) i sprawdzamy następną funkcją, czy permutacja jest rozwiązaniem (czy nie ma bić po skosie).*

Program po podaniu n w linii poleceń powinien wypisać wszystkie znalezione rozwiązania oraz na końcu ich liczbę.

Przykładowa sesja powinna wyglądać następująco:

```
szmaragd:~/lab6/queens 3
Number of solutions: 0
szmaragd:~/lab6/queens 4
 2 4 1 3
 3 1 4 2
Number of solutions: 2
szmaragd:~/lab6/queens 6
 2 4 6 1 3 5
 3 6 2 5 1 4
 4 1 5 2 6 3
 5 3 1 6 4 2
Number of solutions: 4
```

Sprawdź liczbę rozwiązań dla n od 1 do 12.

Zadanie 2 (6 pkt)

Napisz w języku Ada program, który rozwiązuje problem n hetmanów następującym algorytmem:

> *Generujemy kolejne n elementowe permutacje (funkcja z listy na ćwiczeniach) i sprawdzamy następną funkcją, czy permutacja jest rozwiązaniem (czy nie ma bić po skosie).*

Program po podaniu n w linii poleceń powinien wypisać wszystkie znalezione rozwiązania oraz na końcu ich liczbę.

Przykładowa sesja powinna wyglądać następująco:

```
szmaragd:=~/lab6/queens 3
Number of solutions: 0
szmaragd:=~/lab6/queens 4
 2 4 1 3
 3 1 4 2
Number of solutions: 2
szmaragd:=~/lab6/queens 6
 2 4 6 1 3 5
 3 6 2 5 1 4
 4 1 5 2 6 3
 5 3 1 6 4 2
Number of solutions: 4
```

Sprawdź liczbę rozwiązań dla n od 1 do 12.

```
1: procedure HETMANI(n)
2:     position(1 : n) ← all 0
3:     bije_wiersz(1 : n) ← all false
4:     bije_przek1(2 : 2n) ← all false
5:     bije_przek2(−n + 1 : n − 1) ← all false
6:     procedure USTAW(i)
7:         for j from 1 to n do
8:             if ¬(bije_wiersz[j] ∨ bije_przek1[i + j] ∨ bije_przek2[i − j]) then
9:                 position[i] ← j
10:                bije_wiersz[j] ← bije_przek1[i + j] ← bije_przek2[i − j] ← true
11:                if i < n then
12:                    USTAW(i + 1)
13:                else
14:                    DRUKUJ ROZWIĄZANIE
15:                end if
16:                position[i] ← 0
17:                bije_wiersz[j] ← bije_przek1[i + j] ← bije_przek2[i − j] ← false
18:            end if
19:        end for
20:    end procedure
21:    USTAW(1)
22: end procedure
```

Rysunek 1: Pseudokod algorytmu z nawrotami dla problemu hetmanów.

Zadanie 3 (6 pkt)

Napisz w języku Python program, który rozwiązuje problem n hetmanów następującym algorytmem:

> *Generujemy kolejne n elementowe permutacje (funkcja z listy na ćwiczeniach) i sprawdzamy następną funkcją, czy permutacja jest rozwiązaniem (czy nie ma bić po skosie).*

Program po podaniu n w linii poleceń powinien wypisać wszystkie znalezione rozwiązania oraz na końcu ich liczbę.

Przykładowa sesja powinna wyglądać następująco:

```
szmaragd:~/lab6/python queens.py 3
Number of solutions: 0
szmaragd:~/lab6/python queens.py 4
 2 4 1 3
 3 1 4 2
Number of solutions: 2
szmaragd:~/lab6/python queens.py 6
 2 4 6 1 3 5
 3 6 2 5 1 4
 4 1 5 2 6 3
 5 3 1 6 4 2
Number of solutions: 4
```

Sprawdź liczbę rozwiązań dla n od 1 do 12.

Zadanie 4 (6 pkt)

Przeanalizuj algorytm z nawrotami z Rysunku 1 i zaimplementuj go w języku C.

Napisz program, który po podaniu n w linii poleceń wypisuje wszystkie znalezione rozwiązania oraz na końcu ich liczbę.

Sprawdź liczbę rozwiązań dla n od 1 do 12. Który algorytm jest szybszy.

Zadanie 5 (6 pkt)

Przeanalizuj algorytm z nawrotami z Rysunku 1 i zaimplementuj go w języku Ada.

Napisz program, który po podaniu n w linii poleceń wypisuje wszystkie znalezione rozwiązania oraz na końcu ich liczbę.

Sprawdź liczbę rozwiązań dla n od 1 do 12. Który algorytm jest szybszy.

Zadanie 6 (6 pkt)

Przeanalizuj algorytm z nawrotami z Rysunku 1 i zaimplementuj go w języku Python.

Napisz program, który po podaniu n w linii poleceń wypisuje wszystkie znalezione rozwiązania oraz na końcu ich liczbę.

Sprawdź liczbę rozwiązań dla n od 1 do 12. Który algorytm jest szybszy.

Gra Mastermind W grze łamie się ukryty kod złożony z sekwencji czterech cyfr wybranych spośród sześciu (od 1 do 6, cyfry mogą powtarzać się w sekwencji).

Gra dwóch graczy. Jeden układa kod z czterech cyfr (nazywać będziemy go koderem) a drugi stara się go odgadnąć (nazywać będziemy go dekoderem).

Dekoder podaje sekwencję czterech cyfr i dostaje od kodera w odpowiedzi informację ile cyfr jest poprawnych i na swoich miejscach, a ile poprawnych ale na złych miejscach.

Celem jest napisanie programu, który będzie łamał ukryty kod (grał jako dekoder). Zakładamy, że osoba uruchamiająca program jest koderem i zapisała sobie na kartce kod złożony z czterech cyfr. Program cyklicznie drukuje swoją propozycję kodu (cztery cyfry z zakresu od 1 do 6) i czeka na wprowadzenie przez kodera liczby cyfr na swoich miejscach i liczby cyfr dobrych ale nie na swoich miejscach.

Program nie musi łamać kodu minimalną liczbą pytań. Wystarczy, że będzie to robił w kilku pytaniach (maksymalnie ośmiu).

W poniższym przykładzie program znalazł poprawny kod po zadaniu pięciu pytań:

```
$ ./mastermind
1: 1 1 1 1 ?
Na swoim miejscu: 1
Nie na swoim miejscu: 0
2: 1 2 2 2 ?
Na swoim miejscu: 0
Nie na swoim miejscu: 1
3: 3 1 3 3 ?
Na swoim miejscu: 1
Nie na swoim miejscu: 1
4: 3 4 1 4 ?
Na swoim miejscu: 2
Nie na swoim miejscu: 2
5: 3 4 4 1 ?
Na swoim miejscu: 4
Nie na swoim miejscu: 0
Wygrałem.
```

W tym przykładzie program wykrył po pięciu pytaniach, że koder oszukał go:

```
$ ./mastermind
1: 1 1 1 1 ?
Na swoim miejscu: 1
```

```
Nie na swoim miejscu: 0
2: 1 2 2 2 ?
Na swoim miejscu: 1
Nie na swoim miejscu: 0
3: 1 3 3 3 ?
Na swoim miejscu: 1
Nie na swoim miejscu: 0
4: 1 4 4 4 ?
Na swoim miejscu: 1
Nie na swoim miejscu: 0
5: 1 5 5 5?
Na swoim miejscu: 0
Nie na swoim miejscu: 1
Oszukujesz!
```

> **Wskazówka:** Rozważ następujący algorytm: Wygeneruj wszystkie możliwe kody (warianc je z powtórzeniami), a następnie wypisuj pierwszy dostępny i na podstawie odpowiedzi kodera eliminuj te które nie pasują.

Zadanie 7 (6 pkt)
Napisz program `mastermind` w języku C. Czy może on być uogólniony do wszystkich 10 cyfr?

Zadanie 8 (6 pkt)
Napisz program `mastermind` w języku Ada. Czy może on być uogólniony do wszystkich 10 cyfr?

Zadanie 9 (6 pkt)
Napisz program `mastermind` w języku Python. Czy może on być uogólniony do wszystkich 10 cyfr?