# Wykład 14

## Projektowanie procedur rekurencyjnych

Zadanie: znaleźć wszystkie położenia ośmiu hetmanów na 64 polowej szachownicy, tak by żaden hetman nie bił innego (żadne dwa hetmany nie mogą znajdować się w tej samej kolumnie, w tym samym wierszu i na tej samej przekątnej).
Rozwiązanie będzie ilustracją stopniowej konstrukcji metodą prób i błędów.
Stopniowa konstrukcja polega na tym, że znajdujemy reprezentację rozwiązania x postaci [x1,x2,...,xn] konstruując kolejno [x1], [x1,x2], [x1,x2,x3] itd. w ten sposób, że

1. każde przejście od [x1,...,xj] do [x1,...,xj,xj+1] jest prostsze niż obliczanie całego próbnego rozwiązania,
2. jeśli q jest predykatem charakteryzującym rozwiązanie, to musi zachodzić:
   $\forall j$ ((1<=j<=n) => (q(x)=>q([x1,...,xj])

Warunek 2 oznacza, że aby otrzymać pełne i poprawne rozwiązanie musimy uzupełnić rozwiązanie częściowe tak, by spełniało ono kryterium poprawności.

Jeżeli takie uzupełnienie nie jest w danym momencie możliwe, to odwołujemy pewne poprzednie uzupełnienia tego rozwiązania, czyli skracamy je do [x1,...,xi], gdzie i<j i próbujemy inne uzupełnienie. Takie powroty i próbowanie nowego rozwiązania nazywamy nawracaniem.

Szachownica jest podzielona na rzędy i kolumny ponumerowane od 0 do 7.
Hetmany umieszcza się w kolejnych rzędach poczynając od zerowego rzędu i dodając za każdym razem nowego bezpiecznego hetmana.

Algorytm może więc wyglądać następująco:

- umieszczamy hetmana w polu 0,0,
- w pierwszym rzędzie poruszamy się omijając wszystkie pola bite przez hetmana w rzędzie zerowym,
- dla każdego rozmieszczenia dwóch pierwszych hetmanów hetman 2 porusza się w 2 rzędzie opuszczając pola bite przez poprzednich hetmanów.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| X | | | | | | | |
| ? | ? | X | | | | | |
| ? | ? | ? | ? | X | | | |
| ? | X | | | | | | |
| ? | ? | ? | X | | | | |
| ? | ? | ? | ? | ? | ? | ? | ? |
| | | | | | | | |
| | | | | | | | |

0-hetman jest w 0-kolumnie, 1-2, 2-4, 3-1, 4-3,

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| | | | X | | | | | 0 |
| | X | | | | | | | 1 |
| | | | | | | X | | 2 |
| | | X | | | | | | 3 |
| | | | | | X | | | 4 |
| | | | | | | | X | 5 |
| | | | | X | | | | 6 |
| X | | | | | | | | 7 |

Dobre ustawienie: 0-3, 1-1, 2-6, 3-2, 4-5, 5-7, 6-4, 7-0.

Numeracja przekątnych:
Do dołu w lewo:  0,1,2,3,4,5,6,7,..,14
Do dołu w prawo: -7,-6,…,0,1,2,3,4,5,6,7.

Pole (x,y), gdzie x – numer wiersza, y – numer kolumny, znajduje się na przekątnej:

do dołu w prawo y-x,  do dołu w lewo  x+y

Czyszczenie tablicy;
h := 0;
repeat
    postawienie hetmana na polu (0,h);
    „utworzenie wszystkich rozmieszczeń z ustalonym hetmanem 0”;
    h:=h+1
until h=8
Instrukcję abstrakcyjną: „utworzenie wszystkich rozmieszczeń z ustalonym hetmanem 0” zapisujemy::
h1:=0;
repeat
    if pole(1,h1) jest bezpieczne then
    begin
        postawienie hetmana na polu (1,h1);
        utworzenie wszystkich rozmieszczeń z ustalonymi dwoma początkowymi hetmanami;
        usunięcie hetmana z pola (1,h1)
    end
    h1:=h1+1
until h1 = 8

Instrukcję „ Utworzenie wszystkich rozmieszczeń z ustalonymi dwoma początkowymi hetmanami” możemy również wykonać w podobny sposób.

Wstawiając te fragmenty jeden w drugi otrzymalibyśmy w ten sposób 8 zagnieżdżonych pętli repeat.

Taki program byłby poprawny, ale jego wadą byłoby to, że nie dałby się dostosowywać dla innych rozmiarów tablicy. Dobrze byłoby więc by wszystkie pętle były wyrażone za pomocą tego samego tekstu.

Wszystkie pętle poza skrajnymi będą traktowane tak samo. Natomiast pierwsza pętla jest wyróżniona z tego powodu, że nie sprawdzamy w niej, czy pole (0,h) jest bezpieczne gdyż wiemy, że tak jest.
Można jednak włączyć warunek

If pole(0,h) jest bezpieczne then ....

Projektując pętlę najbardziej wewnętrzną, powinniśmy uwzględnić drukowanie układu po otrzymaniu bezpiecznego zestawu hetmanów. Możemy zastąpić instrukcję abstrakcyjną „utworzenie ....” instrukcją:

```
if szachownica zapełniona
 then drukowanie rozmieszczenia
 else utworzenie wszystkich rozmieszczeń
        przez uzupełnienie poprzedniego.
```

Warunek szachownica zapełniona można skonstruować wprowadzając zmienną globalną n, jest ona zapełniona gdy n=8. Każda pętla ma swoją zmienną lokalną h.

```
procedure utworzenie;
var h:integer;
begin
    h:=0;
    repeat
        if pole(n,h) jest bezpieczne then
        begin
            umieszczenie hetmana na polu (n,h);
            n:=n+1;
            if n=8  then drukowanie rozmieszczenia
                    else utworzenie;
            n:=n-1;
            usunięcie hetmana z pola (n,h)
        end;
        h:=h+1
    until h=8
end
```

Szukany algorytm redukuje się do instrukcji:

```
    wyczyszczenie szachownicy;
    n:=0;
    utworzenie;
```

Należy określić reprezentację rozmieszczeń hetmanów na szachownicy: var X : array[0..7] of integer;
X[i] = j tzn. i-ty hetman jest w j-tej kolumnie.

Wprowadzamy dodatkowo zmienne boolowskie Kol, Lewo i Prawo:

- Kol[k]=true gdy w k-tej kolumnie nie ma żadnego hetmana,
- Lewo[k]=true, na k-tej przekątnej biegnącej w lewo w dół nie ma żadnego hetmana,
- Prawo[k]=true, na k-tej przekątnej biegnącej w prawo w dół nie ma żadnego hetmana.

Tablice te należy zdefiniować następująco:

```
var  Kol: array[0..7] of boolean;
     Lewo:array[0..14] of boolean;
     Prawo: array[-7..7] of boolean;
```

Warunek pole(n,h) jest bezpieczne jest postaci;

Kol[h] and Lewo[n+h] and Prawo[h-n].

Instrukcja umieszczenie hetmana w polu (n,h) uściśla się następująco:

```
X[n]:=h; Kol[h]:=false;
Lewo[n+h]:=false; Prawo[h-n]:=false
```

Instrukcję usunięcie hetmana z pola (n,h) uściśla się następująco:

```
Kol[h]:=true; Lewo[n+h]:=true; Prawo[h-n]:=true
```

```
program hetmany;
var  n,k:integer; X: array[0..7] of integer;
       Kol: array[0..7] of boolean;
       Lewo: array[0..14] of boolean;
       Prawo: array[-7..7] of boolean;
procedure utworzenie;
var h:integer;
begin
       h:=0;
       repeat
             if {pole(n,h) jest bezpieczne} Kol[h] and
                Lewo[n+h] and Prawo[h-n] then
              begin {umieszczenie hetmana w polu (n,h)}
                   X[n]:=h;
                   Kol[h]:=false;
                   Lewo[n+h]:=false;
                   Prawo[h-n]:=false;
                   n:=n+1
                   If {szachownica zapełniona} n=8 then
                   begin {drukowanie rozmieszczenia}
                         for k:=0 to 7 do write(X[k]); writeln
                   end else utworzenie;
                   n:=n-1 {usunięcie hetmana z pola (n,h)}
                   Kol[h]:=true;
                   Lewo[n+h]:=true;
                   Prawo[h-n]:=true
             end;
             h:=h+1;
       until h=8
end;
```

```
begin {program główny}
{wyczyszczenie tablicy}
      n:=0;
      for k:=0 to 7 do
            Kol[k]:=true;
      for k:=0 to 14 do
      begin
            Lewo[k]:=true;
            Prawo[k-7]:=true
      end;
      utworzenie  end.
```

**Problem: Wieże Hanoi***

**Wieże Hanoi** – problem polegający na odbudowaniu, z zachowaniem kształtu, wieży z krążków o różnych średnicach, przy czym podczas przekładania wolno się posługiwać buforem (reprezentowanym w tym przypadku przez dodatkowy słupek), jednak przy ogólnym założeniu, że nie wolno kłaść krążka o większej średnicy na mniejszy ani przekładać kilku krążków jednocześnie.

Jest to przykład zadania, którego złożoność obliczeniowa wzrasta niezwykle szybko wraz ze zwiększaniem liczby elementów wieży.

* Materiał zaczerpnięty z Wikipedii.

Zagadka Wież Hanoi stała się znana w XIX wieku dzięki matematykowi Édouard Lucasowi, który proponował zagadkę dla 8 krążków.

Tybetańska legenda głosi, że mnisi w świątyni Brahmy rozwiązują tę łamigłówkę dla 64 złotych krążków. Legenda mówi, że gdy mnisi zakończą zadanie, nastąpi koniec świata.

Zakładając, że wykonują 1 ruch na sekundę, ułożenie wieży zajmie $2^{64}-1$ = 18 446 744 073 709 551 615 (blisko 18 i pół tryliona) sekund, czyli około 584 542 miliardów lat.

Dla porównania: Wszechświat ma około 13,7 mld lat.

**Wieże Hanoi** można rozwiązać za pomocą algorytmu rekurencyjnego lub iteracyjnego.

- Oznaczmy kolejne słupki literami A, B i C.
- Niech *n* będzie liczbą krążków, które chcemy przenieść ze słupka A na słupek C posługując się słupkiem B jako buforem.

**Rozwiązanie rekurencyjne:**

Algorytm rekurencyjny składa się z następujących kroków:

1. przenieś (rekurencyjnie) *n*-1 krążków ze słupka A na słupek B posługując się słupkiem C,
2. przenieś jeden krążek ze słupka A na słupek C,
3. przenieś (rekurencyjnie) *n*-1 krążków ze słupka B na słupek C posługując się słupkiem A

Implementacja w C++

```
#include <iostream>
using namespace std;

void hanoi(int n, char A, char B, char C) {
// przekłada n krążków z A korzystając z B na C
  if (n > 0) {
    hanoi(n-1, A, C, B);
    cout << A << " -> " << C << endl;
    hanoi(n-1, B, A, C);
  }
}


int main(int argc, char *argv[]) {
  hanoi(3, 'A', 'B', 'C');
  return 0;
}
```

## Rozwiązanie iteracyjne

Algorytm iteracyjny składa się z następujących kroków:

1. przenieś najmniejszy krążek na kolejny (*) słupek.
2. wykonaj jedyny możliwy do wykonania ruch, nie zmieniając położenia krążka najmniejszego
3. powtarzaj punkty 1 i 2, aż do odpowiedniego ułożenia wszystkich krążków.

(*) Kolejny słupek wyznaczamy w zależności od liczby krążków. Jeśli liczba krążków jest parzysta, kolejnym słupkiem jest ten po prawej stronie (gdy dojdziemy do słupka C w następnym ruchu używamy słupka A).

Jeśli liczba krążków jest nieparzysta, kolejnym słupkiem jest ten po lewej stronie (gdy dojdziemy do słupka A w następnym ruchu używamy słupka C)

Równanie określające liczbę ruchów potrzebnych do rozwiązania problemu wież Hanoi dla *n* krążków:

$$L(n) = L(n-1) + 1 + L(n-1)$$

**Dowód**

Łatwo pokazać, że $L(n) \leq L(n-1) + 1 + L(n-1)$:

- w pierwszym kroku przekładamy *n-1* krążków na jeden słupek (bez straty ogólności załóżmy, że jest to słupek nr 3) - wymaga to co najmniej *L(n-1)* ruchów
- przekładamy *n-ty* krążek na drugi słupek - wymaga to jednego ruchu
- przekładamy pozostałe krążki ze słupka *3.* na *n-ty* krążek leżący na drugim słupku - wymaga to co najmniej *L(n-1)* ruchów.

Aby wykazać, że $L(n) \geq L(n-1) + 1 + L(n-1)$ przeprowadźmy następujące rozumowanie:

Aby móc ruszyć *n-ty* krążek, trzeba najpierw zdjąć wszystkie leżące na nim krążki, tak by po ich zdjęciu, jeden z słupków pozostał wolny (aby na jego "dno" mógł trafić *n-ty* krążek).

A więc ze słupka *1* przekładamy krążki $1, 2, \cdots n-1$ na słupek *3*. Ponieważ aż do momentu gdy na drążku *1* pozostanie tylko *n-ty* krążek nie ma znaczenia czy rzeczywiście się on tam znajduje, a więc do tego momentu sytuacja upraszcza się do rozwiązania problemu wież Hanoi dla *n-1* krążków (którego minimalna liczba ruchów wynosi *L(n-1)*). Na przełożenie krążka *n-tego* potrzeba co najmniej jeden ruch. Po jego przełożeniu znów potrzeba przełożyć krążki $1, 2, \cdots n-1$- jest to oczywiście znów sytuacja *n-1* krążków (wymagająca co najmniej L(n-1) ruchów).

Powyższe równanie rekurencyjne można przekształcić do postaci nie korzystającej z rekursji:

$$L(n) = 2 \cdot L(n-1) + 1$$
$$L(n) + 1 = 2 \cdot L(n-1) + 1 + 1 = 2 \cdot (L(n-1) + 1)$$

Niech $L'(n) = L(n) + 1$

Wtedy

$$2 \cdot (L(n-1) + 1) = 2 \cdot L'(n-1)$$
$$L'(n) = 2 \cdot L'(n-1)$$

jest to równanie określające ciąg geometryczny o ilorazie równym *2* takie, że

$$\begin{aligned} L(1) &= 1 \\ L'(1) &= 2 \\ L'(2) &= 4 \\ &\vdots \\ L'(n) &= 2^n \end{aligned}$$

Po powrocie do *L*(*n*) otrzymujemy

$$L(n) = 2^n - 1$$