Laboratorium 6: Zastosowanie tranzytywnego domknięcia do znalezienia równoległości pozbawionej synchronizacji

Wariant pętli 5

```
for (int i = 2; i <= n; i++)
    for (int j = 1; j <= n; j++)
        a[i][j] = a[i-2][j-1];
```

Zadanie 1.

Dla wskazanej pętli za pomocą kalkulatora ISCC znaleźć relację zależności R, przestrzeń iteracji LD, oraz zrobić rysunek grafu zależności w przestrzeni 6 x 6.

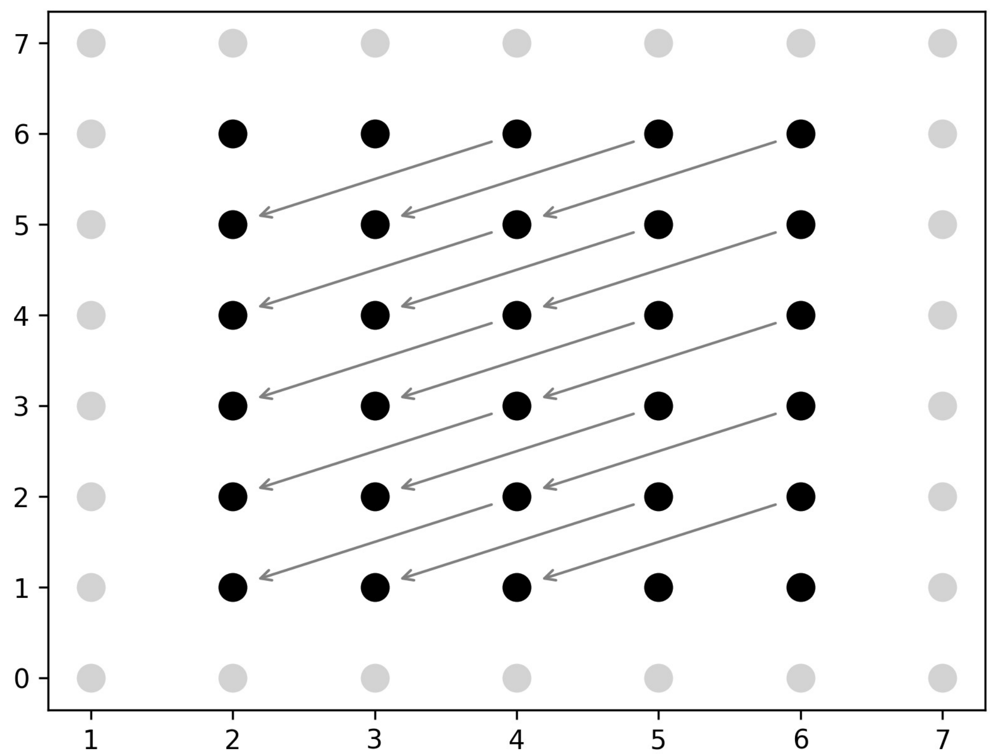
Relacja R:

```
[n] -> { [i, j] -> [i' = 2 + i, j' = 1 + j] : 2 <= i <= -2 + n and 0 < j < n }
```

Przestrzeń iteracji LD (Loop Domain):

```
[n] -> { [i, j] : 2 <= i <= n and 0 < j <= n }
```

Rysunek grafu w przestrzeni 6x6:



Zadanie 3.

Znaleźć początki krańcowe reprezentowane przez zbiór UDS.

"UDS"

[n] -> { [i, j] : 2 <= i <= 3 and i <= -2 + n and 0 < j < n; [i, j = 1] : n >= 2 and 4 <= i <= -2 + n }

Zadanie 4.

Obliczyć relację R_USC.

```
"R_USC"
[n, i1, i2] -> {  }
```

Zadanie 5.
Określić jaka jest topologia grafu zależności

Jest to topologia grafu - łańcuch, ponieważ relacja *R_USC* jest pusta.

Zadanie 6.
Znaleźć punkty reprezentatywne niezależnych fragmentów grafu, czyli zbiór REPR.

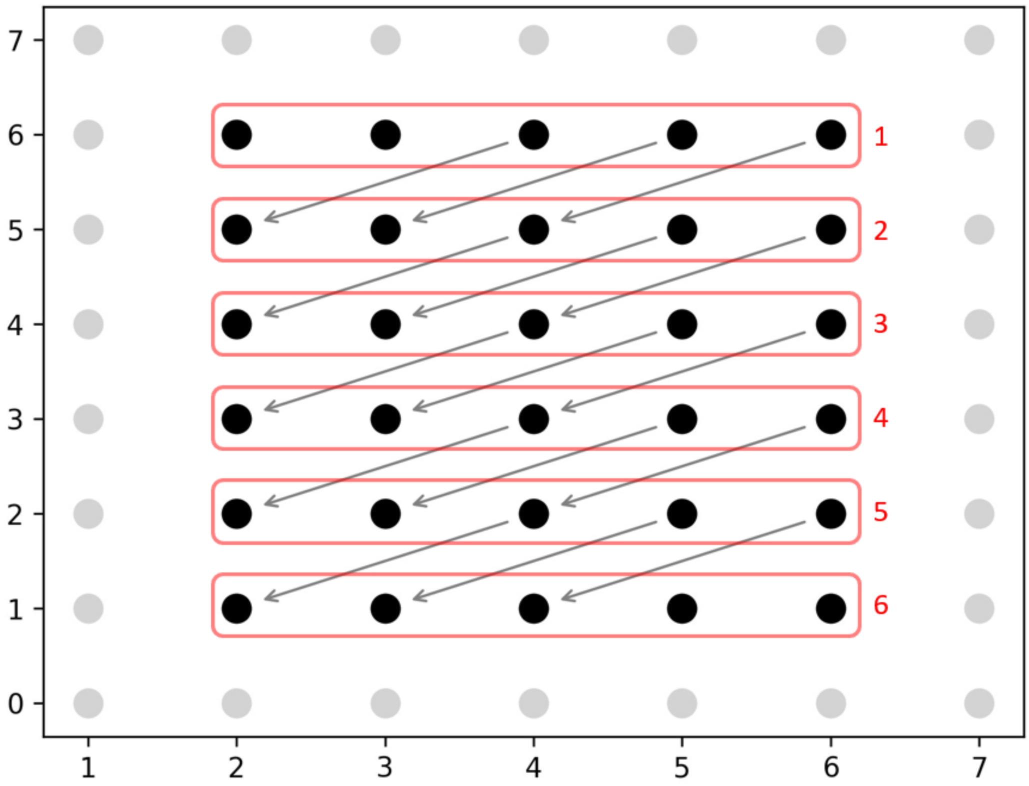
```
"REPR"
[n] -> { [i, j] : 2 <= i <= 3 and i <= n and 0 < j <= n and ((i <= -2 + n and j < n) or i >= -1 + n); [i, j = n] : 2 <= i <= 3 and i <= -2 + n; [i, j = 1] : 4 <= i <= n and (i <= -2 + n or i >= -1 + n) }
```

Zadanie 7.
Znaleźć zbiór, SLICES, zawierający wszystkie iteracje należące do niezależnego fragmentu z danym punktem reprezentatywnym, I.

```
"SLICES"
[n] -> { [i1, i2 = 1, i1, i1'] : 0 < i1 <= n and 2 <= i1' <= n; [i1, i2 = 1, i1, i1' = 1] : n >= 2 and 0 < i1 <= n; [i1 = 1, i2 = 1, 1, i1' = 1] : n = 1 }
```

Zadanie 8.
Stosując zbiór SLICES za pomocą operatora scan znaleźć wszystkie niezależne fragmenty kodu i zaznaczyć je na rysunku utworzonym w p. 2 (rysunek z zależnościami) w przestrzeni 6x6 (12x12).

```
[n] -> { [i1 = 6, i2 = 1, 6, i1' = 6] : n = 6; [i1 = 5, i2 = 1, 5, i1' = 6] : n = 6; [i1 = 4, i2 = 1, 4, i1' = 6] : n = 6; [i1 = 3, i2 = 1, 3, i1' = 6] : n = 6; [i1 = 2, i2 = 1, 2, i1' = 6] : n = 6; [i1 = 1, i2 = 1, 1, i1' = 6] : n = 6; [i1 = 6, i2 = 1, 6, i1' = 5] : n = 6; [i1 = 5, i2 = 1, 5, i1' = 5] : n = 6; [i1 = 4, i2 = 1, 4, i1' = 5] : n = 6; [i1 = 3, i2 = 1, 3, i1' = 5] : n = 6; [i1 = 2, i2 = 1, 2, i1' = 5] : n = 6; [i1 = 1, i2 = 1, 1, i1' = 5] : n = 6; [i1 = 6, i2 = 1, 6, i1' = 4] : n = 6; [i1 = 5, i2 = 1, 5, i1' = 4] : n = 6; [i1 = 4, i2 = 1, 4, i1' = 4] : n = 6; [i1 = 3, i2 = 1, 3, i1' = 4] : n = 6; [i1 = 2, i2 = 1, 2, i1' = 4] : n = 6; [i1 = 1, i2 = 1, 1, i1' = 4] : n = 6; [i1 = 6, i2 = 1, 6, i1' = 3] : n = 6; [i1 = 5, i2 = 1, 5, i1' = 3] : n = 6; [i1 = 4, i2 = 1, 4, i1' = 3] : n = 6; [i1 = 3, i2 = 1, 3, i1' = 3] : n = 6; [i1 = 2, i2 = 1, 2, i1' = 3] : n = 6; [i1 = 1, i2 = 1, 1, i1' = 3] : n = 6; [i1 = 6, i2 = 1, 6, i1' = 2] : n = 6; [i1 = 5, i2 = 1, 5, i1' = 2] : n = 6; [i1 = 4, i2 = 1, 4, i1' = 2] : n = 6; [i1 = 3, i2 = 1, 3, i1' = 2] : n = 6; [i1 = 2, i2 = 1, 2, i1' = 2] : n = 6; [i1 = 1, i2 = 1, 1, i1' = 2] : n = 6; [i1 = 6, i2 = 1, 6, i1' = 1] : n = 6; [i1 = 5, i2 = 1, 5, i1' = 1] : n = 6; [i1 = 4, i2 = 1, 4, i1' = 1] : n = 6; [i1 = 3, i2 = 1, 3, i1' = 1] : n = 6; [i1 = 2, i2 = 1, 2, i1' = 1] : n = 6; [i1 = 1, i2 = 1, 1, i1' = 1] : n = 6 }
```



Zadanie 9.
Wygenerować pseudokod.

```
for (int c0 = 2; c0 <= n; c0 += 1)
    for (int c3 = 1; c3 <= n; c3 += 1)
        (c0, 1, c0, c3);
```

Zadanie 10.
Przetransformować pseudokod na kod kompilowany w OpenMP.

```
for (int c0 = 2; c0 <= n; c0 += 1)
    for (int c3 = 1; c3 <= n; c3 += 1)
        a[c0][c3] = a[c0-2][c3-1];
```

Zadanie 11.
Zastosować program opracowany w (p.7, L2) do sprawdzenia poprawności kodu docelowego w przestrzeni 6x6 (12x12).

```
# gcc -fopenmp 2-joined.c && ./a.out
Initial code result:
00 01 02 03 04 05 06
00 01 02 03 04 05 06
00 00 01 02 03 04 05
00 00 01 02 03 04 05
00 00 00 01 02 03 04
00 00 00 01 02 03 04
```

```
00 00 00 00 01 02 03

Generated code result:
00 01 02 03 04 05 06
00 01 02 03 04 05 06
00 00 01 02 03 04 05
00 00 01 02 03 04 05
00 00 00 01 02 03 04
00 00 00 01 02 03 04
00 00 00 00 01 02 03

Results are identical.
```

Załączniki.

Skrypt implementujący zadania.

```
#krok 1 relacja zaleznosci
R:=[n] -> { [i, j] -> [i' = 2 + i, j' = 1 + j] : 2 <= i <= -2 + n and 0 < j < n };
print "R"; R;

# przestrzen iteracji
LD:=[n]->{ [i, j] : 2 <= i <= n and 0 < j <= n };
print "LD"; LD;

#krok 3 - poczatki krancowe
UDS:= domain R - range R;
print "UDS"; UDS;

#krok 4 - tworzenie relacji R_USC wedlug wzoru
## R_USC :=  { [e] -> [e’] | e, e’ in UDS & e > e’ &  e’ in (R+ R+^-1)(e) }

#4.1 tworzymy relacja  R1:={[e]->[e']}, gdzie  e=(i1,i2) e'=i1',i2'
R1:={[i1,i2]->[i1',i2']};
print "R1"; R1;

## 4.2 twozymy relacje; R2, implementujaca warunek: domain i range R1 nalezy do UDS
R2:=(R1*UDS)->*UDS;
print "R2"; R2;
scan (R2* [n]->{:n=2});

##4.3   tworzymy  relacje  m,  dla  ktorej   elementy  dziedziny  sa  leksykograficznie  mniejsze  niz
elementy  zakresu: e<e'
S:={[i1,i2]};
m:=S<<S;
print "m"; m;
R3:=R2*m;
print "R3"; R3;
scan (R3* [n]->{:n=2});

##4.4 implementujemy warunek e' in (R+ + R+^-1)(e)
RR:=R^+ + (R^+)^-1;
print "RR"; RR;
```

```
e:=[i1,i2]->{[i1,i2]}*UDS;
SET:=RR(e);
print "SET"; SET;
scan (SET* [n]->{:n=2});

##4.5
R_USC:=R3->*SET;
print "R_USC"; R_USC;

#krok 6 tworzenie zbioru zawierajacego punkty reprezentatywne niezaleznych fragmentow kodu
REPR:=UDS +(LD -(domain R +range R));
print "REPR"; REPR;

#krok 7 - obliczenie zbioru SLICES
## obliczenie pozytywnego tranzytywnego domkniecia realcji R
R_PLUS:=R^+;
print "R_PLUS"; R_PLUS;

#obliczenie tranzytywnego domkniecia, R_STAR
R_STAR:=R_PLUS + {[i1,i2]->[i1,i2]};

## tworzymy sparametrzyzowany punkt reprezentatywny, I
I:=[i1,i2]->{[i1,i2]}*REPR;

## obliczamy wynik zastosowania R_STAR do punktu  reprezentatywnego I, S
S:=R_STAR(I);
print "S"; S;
scan (S*[n]->{:n=2});

## krok 7 - Tworzenie zbioru SLICES
SLICES:=
[n] -> { [i1,i2,i1, i1'] : i2 = 1 and n >= 2 and 0 < i1 <= n and 2 <= i1' <= n;
    [i1,i2,1, 1] : i2 = 1 and n = 1 and i1 = 1;
    [i1,i2, i1, 1] : i2 = 1 and n >= 2 and 0 < i1 <= n
};
print "SLICES"; SLICES;

##krok 8
scan (SLICES*[n]->{:n=6});

## krok 9 - wygenerowanie pseudokodu
codegen (identity SLICES);
```