Laboratorium 7: Zastosowanie tranzytywnego domknięcia do partycjonowania czasu

Wariant pętli 5

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

Zadanie 1.

Dla wskazanej pętli za pomocą kalkulatora ISCC znaleźć relację zależności, R, oraz przestrzeń iteracji, LD.

Relacja R:

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

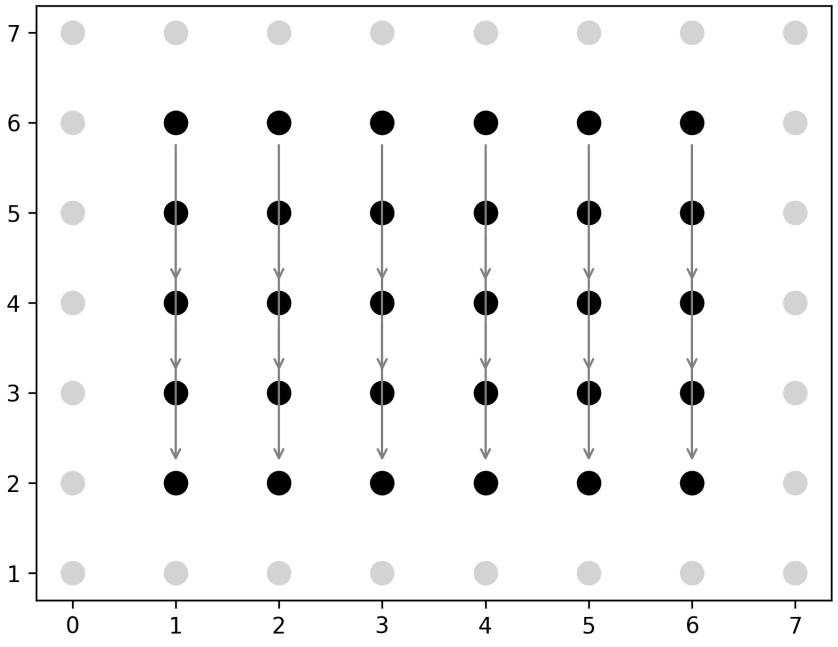
Przestrzeń iteracji LD (Loop Domain):

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

Zadanie 2.

Zrobić rysunek pokazujący zależności w przestrzeni 6 x 6. W tym celu trzeba zastosować operator scan (R*[n]->{:n=6}); który wygeneruje wszystkie zależności w przestrzeni 6 x 6.

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



Zadanie 3.

Obliczyć tranzytywne domknięcie relacji R, R^+

```
"R_PLUS"
([n] -> { [i, j] -> [i, j] :
      (j + j) mod 2 = 0 and
      0 < i < n and
      2 <= j <= -2 + n and
      j' >= 2 + j and
      4 <= j <= n
}, True)
```

Zadanie 4.
Obliczyć relację R^k.

```
"Rk"
([n] -> { [k] -> [[i, j] -> [i' = i, j' = 2k + j]] : k > 0 and 0 < i < n and 2 <= j <= n
- 2k }, True)
```

Zadanie 5.
Znaleźć zbiór UDS zawierający początki krańcowe

```
"UDS"
[n] -> { [i, j] : 0 < i < n and 2 <= j <= 3 and j <= n }
```

Zadanie 6.
Obliczyć zbiór S(k):= R^k(UDS) - (R^+ . R^k)(UDS).

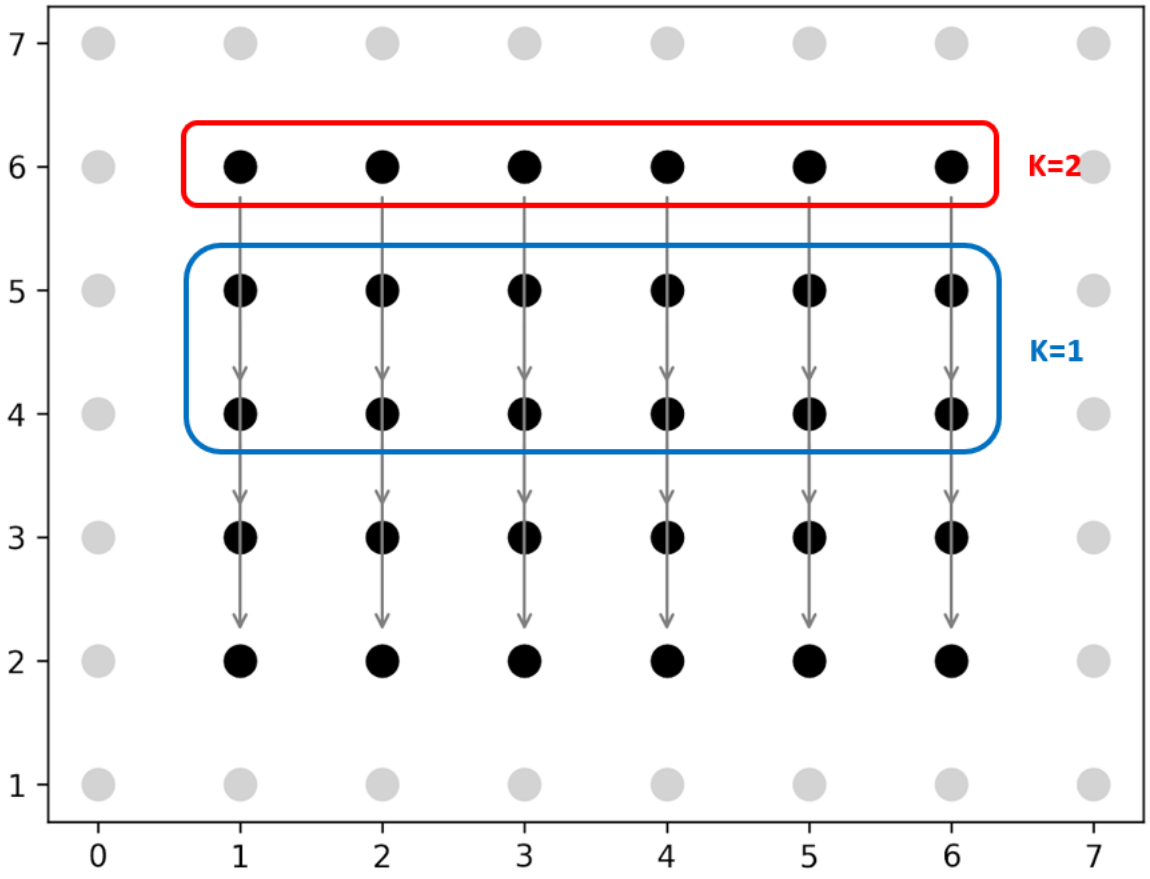
```
"Sk"
[n, k] -> { [i, j] : k > 0 and 0 < i < n and 2 + 2k <= j <= 3 + 2k and j <= n }
```

Zadanie 7.
Utworzyć relację CODE.

```
[n]-> { [i, j] -> [k, i, j] : 0 < i < n and j <= n and k > 0 and -3 + j <= 2k <= -2 + j }
```

Zadanie 8.
Stosując relację CODE za pomocą operatora scan znaleźć wszystkie partycje czasu dla przestrzeni 6x6 i nanieść uzyskane partycje na rysunku utworzonym w p.2.

```
[n] -> { [i = 5, j = 6] -> [k = 2, 5, 6] : n = 6; [i = 4, j = 6] -> [k = 2, 4, 6] : n =
6; [i = 3, j = 6] -> [k = 2, 3, 6] : n = 6; [i = 2, j = 6] -> [k = 2, 2, 6] : n = 6; [i =
1, j = 6] -> [k = 2, 1, 6] : n = 6; [i = 5, j = 5] -> [k = 1, 5, 5] : n = 6; [i = 4, j =
5] -> [k = 1, 4, 5] : n = 6; [i = 3, j = 5] -> [k = 1, 3, 5] : n = 6; [i = 2, j = 5] ->
[k = 1, 2, 5] : n = 6; [i = 1, j = 5] -> [k = 1, 1, 5] : n = 6; [i = 5, j = 4] -> [k = 1,
5, 4] : n = 6; [i = 4, j = 4] -> [k = 1, 4, 4] : n = 6; [i = 3, j = 4] -> [k = 1, 3, 4] :
n = 6; [i = 2, j = 4] -> [k = 1, 2, 4] : n = 6; [i = 1, j = 4] -> [k = 1, 1, 4] : n = 6 }
```



Zadanie 9.

Wygenerować pseudokod i przekonwertować go na kod kompilowany.

Pseudokod

```
for (int c0 = 1; c0 < floord(n, 2); c0 += 1)
  for (int c1 = 1; c1 < n; c1 += 1)
    for (int c2 = 2 * c0 + 2; c2 <= min(n, 2 * c0 + 3); c2 += 1)
      (c1, c2);
```

Kod kompilowalny

```
for (int c0 = 1; c0 < floord(n, 2); c0 += 1)
  #pragma openmp parallel for
  for (int c1 = 1; c1 <= n; c1 += 1)
    for (int c2 = 2 * c0; c2 <= min(n, 2 * c0 + 3); c2 += 1)
        a[c1][c0] = a[c1][c0-2];
```

Zadanie 10.

Obliczyć zbiór, IND, zawierający niezależne iteracje pętli.

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

Pierwsza krotka w IND jest tylko dla przykładu gdy n jest w przedziale od 2 do . Ze względu na to, że w ćwiczeniu jest używane n=6, to można byłoby ominąć tę krotkę.

Zadanie 11.
Jeśli zbiór IND nie jest pusty, to wygenerować pseudokod i kod kompilowany.

Zbiór IND, nie jest pusty, więc można utworzyć relację CODE_IND i na jego podstawie wygenerować pseudokod.

Pseudokod

```
"CODE for IND"
for (int c1 = 3; c1 <= 6; c1 += 1)
  (2, c1);
```

Pierwsza pętla wykona się tylko dla n przedziale od 2 do 4. W przypadku tego zadania, gdzie n = 6, ta pętla nie zostanie wykonana.

Kod kompilowany

```
#pragma openmp parallel for
for (int c1 = 3; c1 <= 6; c1 += 1)
   a[2][c1] = a[2][c1-2];
```

Zadanie 12.
Zastosować program porównujący wyniki obliczeń (zadanie 7, L2) do sprawdzania poprawności kodu docelowego w przestrzeni 6x6

Połączony kod kompilowalny

```
   for (int c0 = 1; c0 < floord(n, 2); c0 += 1)
        #pragma openmp parallel for
        for (int c1 = 1; c1 <= n; c1 += 1)
            for (int c2 = 2 * c0; c2 <= min(n, 2 * c0 + 3); c2 += 1) {
                aGenerated[c1][c2] = aGenerated[c1][c2-2];
                printf("[c1 = %d, c2 = %d]   <=   [c1 = %d, c2 = %d]\n", c1, c2, c1,
c2-2);
            }


    #pragma openmp parallel for
    for (int c1 = 3; c1 <= 6; c1 += 1) {
        aGenerated[2][c1] = aGenerated[2][c1-2];
        printf("[c1 = %d, c2 = %d]   <=   [c1 = %d, c2 = %d]\n", 2, c1, 2, c1-2);
    }


Initial code result:
00 01 02 03 04 05 06
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
```

```
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00

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

Załączniki.

Skrypt implementujący zadania.

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

print "scan (R*[n]->{:n=6})";
scan (R*[n]->{:n=6});

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

##krok 3: Tranzytywne domkniecie realacji R:
R_PLUS:=R^+;
print "R_PLUS"; R_PLUS;

##krok 4: Obliczenie R^k:
Rk:=pow R;
print "Rk"; Rk;

Rk:=[n, k] -> { [i, j] -> [i' = i, j' = 2k + j] :
    k > 0 and
    0 < i <= n and
    2 <= j <= n - 2k
};

#krok 5: obliczenie zbioru UDS:
UDS:= domain R - range R;
print "UDS"; UDS;

# krok 6: Obliczenie zbioru Sk
Sk:=Rk(UDS) - (R_PLUS . Rk)(UDS);
print "Sk"; Sk;

## krok 7: Tworzenie relacji CODE w oparciu o zbior Sk
CODE:=[n] -> { [i, k] -> [k, i, j] :
    k > 0 and
    0 < i < n and
    2 + 2k <= j <= 3 + 2k and
    j <= n
```

```
};
print "CODE"; CODE;
scan (CODE*[n]->{:n=6});

##krok 8 generowanie pseudokodu  i kodu kompilowalnego
print "CODE"; codegen CODE;

##krok 9 obliczenie zbioru IND
IND:= LD - (domain R + range R);
print "IND"; IND;
CODE_IND:= [n] -> { [i, j] -> [i = n, j] : n > 0 and 2 <= j <= n };

print "CODE for IND";
codegen CODE_IND;
```