TAIO1

Krzysztof Kicun, Paweł Kasjaniuk

Październik 2020

Spis treści

1 Opis problemu

Niech dany będzie zbiór k klocków złożonych z n kwadratów o wymiarach 1x1. Klocki tworzone są w taki sposób, że budujące je segmenty muszą przylegać do siebie co najmniej jedną krawędzią - nie dopuszcza się ułożenia kwadratów gdzie jedynym punktem styku jest wspólny wierzchołek.

Należy znaleźć wszystkie takie ułożenia klocków na planszy (nieskończona siatka złożona z kwadratów 1x1) aby kwadrat zawierający wszystkie elementy miał minimalne pole. Klocki można dowolnie obracać w ramach płaszczyzny równoległej do planszy.

2 Opis algorytmu

Algorytm z powrotami. Rozwiązanie problemu opierać się będzie na próbie położenia wszystkich klocków na fragmencie planszy ograniczonym przez jak najmniejszy kwadrat zaczynając od kwadratu o boku $a = \lceil \sqrt{\text{suma pól klocków}} \rceil$. W przypadku kiedy nie uda się znaleźć żadnego ułożenia, w którym możliwe jest położenie wszystkich klocków ze zbioru to zwiększymy bok kwadratu.
Dla danego wymiaru kwadratu będziemy rekurencyjnie układać elementy na wszystkich dostępnych miejscach uwzględniając 4 możliwe obroty klocka. Obliczenia zakończymy sprawdzając wszystkie możliwe ułożenia klocków na pierwszym kwadracie, na którym udało się położyć wszystkie klocki.

Poniżej znajduję się przedstawienie algorytmu z użyciem języka naturalnego:
Dane wejściowe: lista klocków K

1. zainicjuj pusty zbiór rozwiązań R
2. zainicjuj początkową długość boku planszy $a = \lceil \sqrt{\text{suma pól klocków}} \rceil$
3. zainicjuj pustą, kwadratową planszę P o boku a
4. wywołaj funkcję $F(0)$
5. jeśli $R \neq \emptyset$ - znaleziono rozwiązanie
   (a) zwróć R
6. jeśli $R = \emptyset$ to:
   (a) $a = a + 1$ - zwiększ bok planszy
   (b) idź do punktu 3 - rozpocznij obliczenia na rozszerzonej planszy

---

funkcja F(int i):

1. zainicjuj licznik obrotów $r = 0$
2. zainicjuj pustą listę segmentów planszy L - zbiór poprawnych położeń klocka k_i
3. wybierz dowolny segment s klocka k_i
4. dla każdego segmentu planszy $q \in P$:

- Sprawdź czy da się położyć klocek k_i poprzez ułożenie s na q. Jeśli klocek k_i:
  - nie nachodzi na już ułożone klocki
  - nie wystaje poza planszę P

  to:
  - dodaj lokalizację q do L

5. dla każdego $l \in L$:
   (a) ułóż klocek k_i na P, tak by segment s leżał na l
   (b) jeśli nie ułożono jeszcze wszystkich klocków z K, to wywołaj funkcję $F(i+1)$
   (c) jeśli ułożono już wszystkie klocki z K to:
   - dodaj r do R

   (d) podnieś klocek k_i z planszy
6. jeśli $r < 3$ to:
   - obróć klocek k_i o $90°$ zgodnie z ruchem wskazówek zegara
   - $r = r + 1$
   - idź do punktu 2

3 Dowód poprawności

Dowód stopu:

1. Rozmiar planszy jest ograniczony od dołu przez $a = \lceil \sqrt{\text{suma pól klocków}} \rceil$ oraz od góry poprzez $a = kn$.
2. Liczba możliwych lokalizacji wraz z obrotami jest ograniczona od góry przez $4a^2$.
3. Liczba klocków jest skończoną liczbą naturalną - co gwarantuje zakończenie zejść rekurencyjnych.

Skończona liczba możliwych wymiarów planszy oraz lokalizacji klocków na danym etapie układania oraz ograniczona głębokość zejść rekurencyjnych dowodzi spełnialności warunku stopu.

Rozważamy wszystkie zakresy planszy, które mogą spełniać założenia o minimalnym rozmiarze kwadratu. Układanie elementu na każdym możliwym wolnym miejscu wraz z przejściem rekurencyjnym do pozostałych elementów gwarantuję nam sprawdzenia wszystkich możliwych ułożeń planszy. Algorytm ma więc naturę brute force. Załóżmy, że istnieje rozwiązanie brakujące lub lepsze od znalezionych rozwiązań. Ponieważ algorytm sprawdził wszystkie możliwe rozwiązania i nie otrzymał tego rozwiązania dochodzimy do sprzeczności.

4 Oszacowanie złożoności

k - ilość klocków do ułożenia
n - ilość segmentów w każdym klocku

4.1 Złożoność pamięciowa

Pesymistyczne przypadki: (największe możliwe zużycie pamięci)

- $4(nk)^2$ - maksymalna ilość lokalizacji do sprawdzenia podczas wywołania funkcji F
- k - maksymalna głębokość wywołań rekurencyjnych

Zatem złożoność pamięciowa jest $O((nk)^2k)$

4.2 Złożoność obliczeniowa

1. Ograniczenie dolne
   Ze względu na szacowanie dolne przyjmujemy, że rozwiązanie optymalne znajduje się na najmniejszej (pierwszej) rozważanej planszy. Liczba operacji sprawdzenia dostępnych lokalizacji dla i-tego klocka w przypadku algorytmu bez powrotów wynosiłaby:

   $$L_c = 4(a^2 - in)$$

   gdzie 4 wynika z liczby możliwych obrotów klocka, a $(a^2 - in)$ jest liczbą wolnych segmentów na planszy.
   Liczba operacji przy sprawdzeniu wszystkich ułożeń dla wszystkich klocków na planszy o wymiarze a wynosi:

   $$L_c = \prod_{i=0}^{k-1} 4(a^2 - in)$$

   Jednakże w przypadku algorytmu z powrotami w sytuacji kiedy nie można położyć i-tego klocka, dostępne położenia klocków $i+1, i+2, ..., k-1$ nie będą sprawdzane.

   Należy znaleźć minimalną ilość klocków, co do których będziemy mieli gwarancje istnienia dostępnej lokalizacji.
   Rozważmy podział planszy na kwadraty o wymiarach n x n. Oznaczmy przez $p = \lfloor \frac{a^2}{n^2} \rfloor$ liczbę podziałów planszy. W każdym z takich podziałów jesteśmy w stanie położyć klocek. Mamy więc gwarancję, że dla danego rozmiaru a aktualnie rozważanej planszy w każdej iteracji położymy co najmniej p klocków, a dla kolejnego będziemy szukać lokalizacji.

Ostatecznie liczba klocków, dla których będziemy szukać miejsca na planszy wynosi co najmniej $\lceil \frac{a^2}{n^2} \rceil$
Minimalna ilość operacji wynosi więc:

$$L_{min} = \prod_{i=0}^{\lceil \frac{a^2}{n^2} \rceil} 4(a^2 - in)$$

Ze względu na przyjęte szacowanie dolne $a = a_{min} = \sqrt{kn}$.
Obliczamy:

$$L_{min} = \prod_{i=0}^{\lceil \frac{k}{n} \rceil} 4(kn - in) = (4kn)^{\lceil \frac{k}{n} \rceil + 1} \prod_{i=0}^{\lceil \frac{k}{n} \rceil} (1 - \frac{i}{k})$$

Szacując z dołu otrzymujemy:

$$L_{min} > (4kn)^{\lceil \frac{k}{n} \rceil + 1} (1 - \frac{\lceil \frac{k}{n} \rceil}{k})^{\lceil \frac{k}{n} \rceil + 1}$$

Ostatecznie:

$$L_{min} > (4k(n-1))^{\lceil \frac{k}{n} \rceil + 1}$$

2. Ograniczenie górne
   - $nk - \lceil \sqrt{nk} \rceil$ - maksymalna ilość rozmiarów planszy do sprawdzenia
   - $4(nk)^2$ maksymalna ilość lokalizacji do sprawdzenia dla wywołania funkcji F na głębokości 1
   - $4((nk)^2 - kn)$ maksymalna ilosc lokalizacji do sprawdzenia dla ostatniego wywołania funkcji F, na głębokości k

$(nk - \lceil \sqrt{nk} \rceil) * 4(nk)^2 * 4((kn)^2 - n) * ... * 4((kn)^2 - kn) <$
$< (nk - \lceil \sqrt{nk} \rceil) * (4(kn)^2)^k < nk(4(nk)^2)^k$
Zatem złożoność obliczeniowa jest $O(nk(4(nk)^2)^k)$

5 Heurystyka

Algorymt heurystyczny znajduje tylko jedno ułożenie klocków, które niekoniecznie jest optymalne.
Wyjaśnienie: przez trójkę rozumiemy strukturę danych o trzech nazwanych polach.
Dane wejściowe: lista klocków K
Pseudokod:

1. zainicjuj pustą, kwadratową planszę P o boku $\lceil \sqrt{\text{suma pól klocków}} \rceil$
2. dowolną kolejnością ponumeruj klocki z K, od 0 do $(|K| - 1)$

3. ułóż klocek k_0 w górnym lewym rogu planszy
4. dla każego i, takiego że $0 < i < |K|$:
   (a) zainicjuj licznik obrotów $r = 0$
   (b) wybierz dowolny segment s klocka k_i
   (c) zainicjuj pustą listę trójek (segment; licznik obrotów; licznik sąsiadów) L
   (d) dla każdego segmentu $q \in P$:
      - jeśli można położyć k_i na P w taki sposób, że s leży na q i jednocześnie k_i nie nakłada się na już ułożone klocki na P oraz nie wystaje poza P to:
        – wyznacz liczbę krawędzi c którymi klocek k_i sąsiaduje z już ułożonymi klockami w takim ułożeniu
        – dodaj trójkę (q, r, c) do L
   (e) obróć klocek k_i o 90° zgodnie z ruchem wskazówek zegara
   (f) jeśli $r < 3$ to:
      - $r = r + 1$
      - idź do punktu (d)
   (g) Jeśli $L = \emptyset$ to rozszerz P o 1 pole w dół i o 1 pole w prawo i wróć do punktu (a)
   (h) Wybierz z L trójkę l o największej wartości "licznika sąsiadów" (jeśli jest wiele trójek w L o takiej samej wartości "licznika sąsiadów", to wybierz losową z nich)
   (i) obróć klocek k_i o 90° zgodnie z ruchem wskazówek zegara tyle razy, ile wskazuje na to "licznik obrotów" w trójce l
   (j) ułóż k_i na P tak, żeby segment s znajdował się na segmencie zapamiętanym w trójce l
5. zwróć P

5.1 Złożoność

k - ilość klocków do ułożenia
n - ilość segmentów w każdym klocku

5.1.1 Złożoność pamięciowa

- $(kn)^2$ - maksymalna wielkość planszy
- kn - ilość pól klocków

Złożonośc pamięciowa jest $O((kn)^2)$

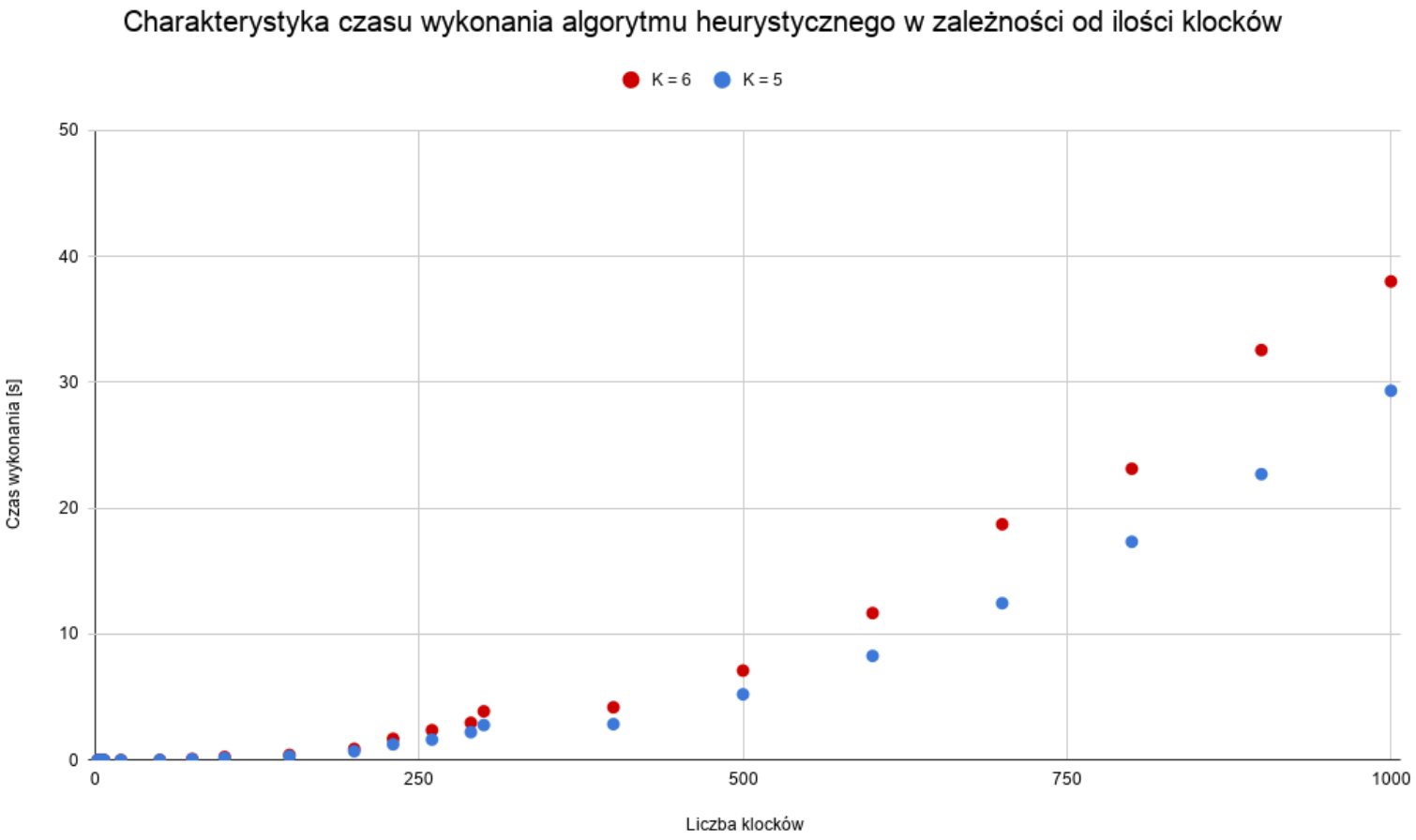
5.1.2 Złożoność obliczeniowa

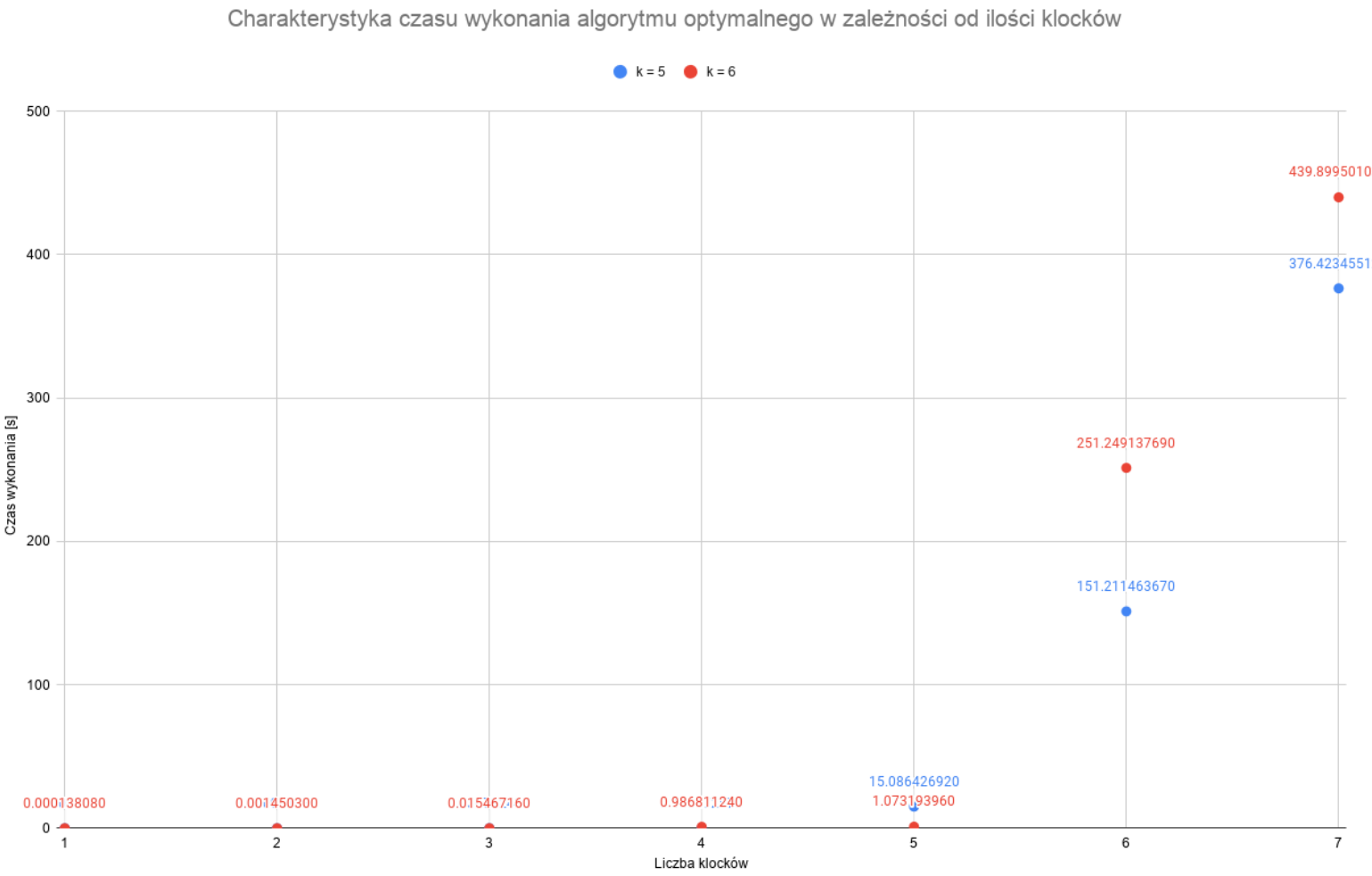
Szacowanie górne:

- $2nk + 2$ - maksymalna liczba lokalizacji do sprawdzenia
- $nk - \lceil\sqrt{nk}\rceil$ - maksymalna liczba rozszerzeń planszy

$(2nk + 2)(nk - \lceil\sqrt{nk}\rceil) < (2nk + 2)nk = 2(nk)^2 + 2nk$
Zatem złożoność obliczeniowa algorytmu heurystycznego jest $O((nk)^2)$

6 Wyniki wydajnościowe




Charakterystyka czasu wykonania algorytmu optymalnego w zależności od ilości klocków
k = 5
k = 6
500
400
300
200
100
0
Czas wykonania [s]
439.8995010
376.4234551
251.249137690
151.211463670
15.086426920
1.073193960
0.986811240
0.015467160
0.001450300
0.000138080
1
2
3
4
5
6
7
Liczba klocków

7 Historia zmian dokumentacji

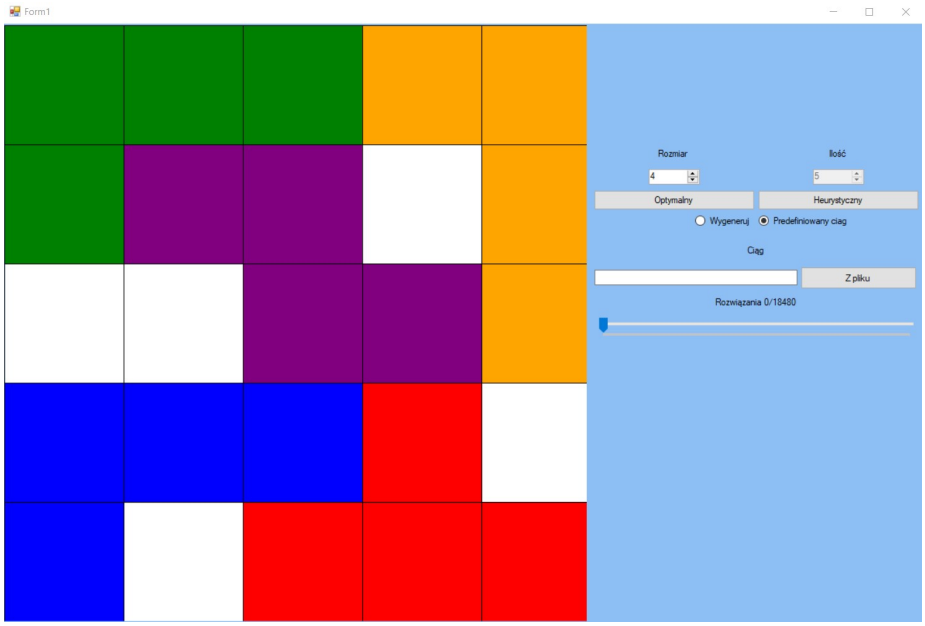
Algorytm optymalny po położeniu wszystkich klocków zawsze dodaje rozwiązanie. Poprzednio sprawdzał czy takie rozwiązanie już istnieje.

8 Readme

8.1 Instrukcja uruchomienia

1. Umieść pliki *TAIO.exe*, *Algorithm.dll* w jednym folderze
2. Uruchom program klikając dwukrotnie na *TAIO.exe*

8.2 Opis interfejsu użytkownika



Program działa w dwóch głównych trybach:

1. Układanie klocków generowanych losowo (tryb ten jest aktywny dla zakreślonego checkboxa *Wygeneruj*)
2. Układanie klocków w oparciu o podany ciąg (tryb ten jest aktywny dla zakreślonego checkboxa *Predefiniowany ciąg*)

Dla pierwszego trybu wybieramy w okienkach *Rozmiar* i *Ilość* rozmiar i ilość generowanych losowo klocków, a następnie, w zależności od tego jaki algorytm chcemy uruchomić, klikamy przycisk *Optymalny* lub *Heurystyczny*. Algorytm heurystyczny zwróci jedno ułożenie klocków, zostanie ono wyświetlone na ekranie. Algorytm optymalny znajdzie wszystkie rozwiązania i wyświetli pierwsze

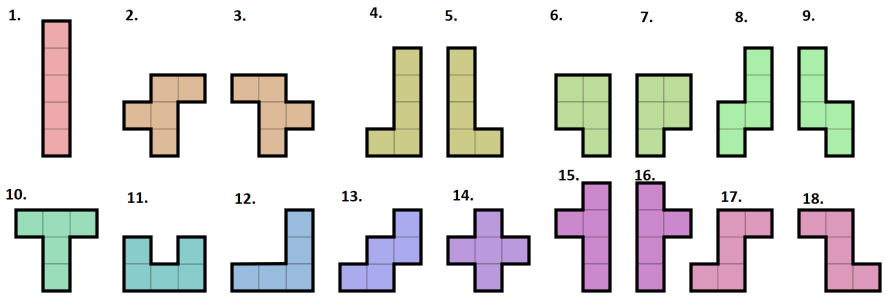
z nich, dodatkowo suwakiem po prawej stronie będzie możliwe przewijanie wyświetlanych rozwiązań.
Dla drugiego trybu pojawia się po prawej stronie pole tekstowe podpisane *Ciąg*.

- Jeżeli wpiszemy w nim jedną liczbę, to będzie to liczba klocków w zadaniu. Rozmiar zostanie wzięty z pola *Rozmiar* powyżej. Aby uruchomić obliczenia trzeba wcisnąć przycisk *Optymalny* lub *Heurystyczny*, w zależności od wybranego algorytmu.
- Jeżeli wpiszemy ciąg liczb całkowitych oddzielonych spacjami, a rozmiar klocka będzie ustawiony na 5 lub 6, to ciąg ten będzie definiował ilość i kształt klocków w zadaniu. Przykładowo: *1 0 1 1* oznacza, że ułożone zostaną 3 klocki (w zależności od pola *Rozmiar* o rozmiarze 5 lub 6) o indeksach 1, 3 i 4 (indeksy klocków zdefiniowane poniżej). Aby uruchomić obliczenia trzeba wcisnąć przycisk *Optymalny* lub *Heurystyczny*, w zależności od wybranego algorytmu.

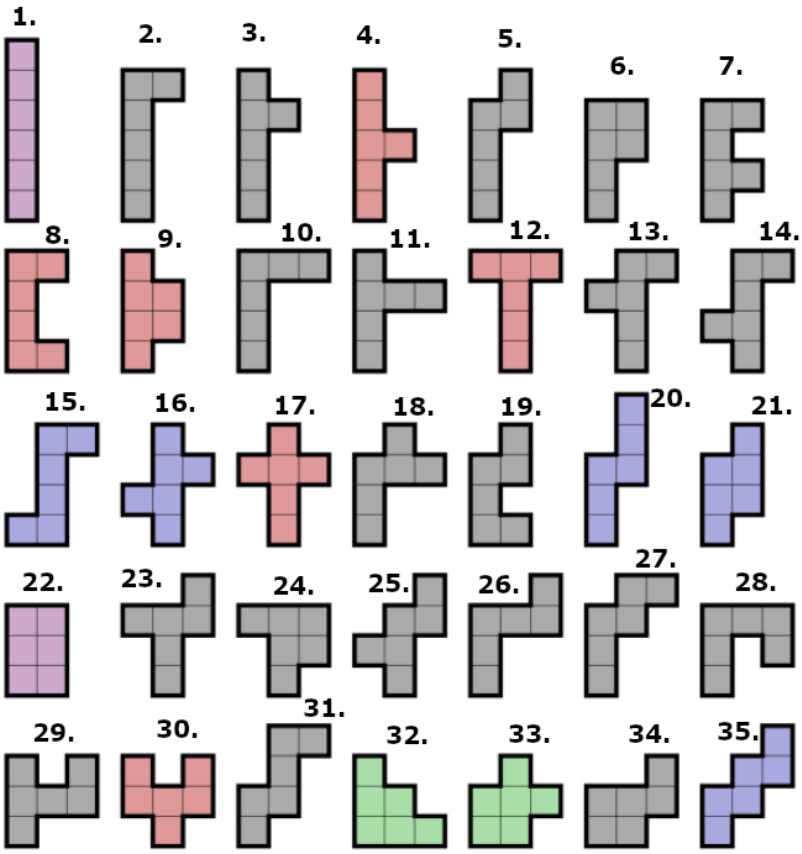
Dodatkowo, w drugim trybie jest możliwość podania na wejściu programu pliku .txt o odpowiednim formacie (format pliku wejściowego opisany poniżej), zawierającego klocki do ułożenia. Plik może zawierać wiele zadań do obliczenia, wtedy zadania są obliczane kolejno, każde po wciśnięciu przycisku *Następne zadanie*. Wraz z projektem zawarte są przykładowe pliki wejściowe w folderze *przyklady*.

8.3 Indeksy predefiniowanych klocków

8.3.1 Pentomino - klocki o rozmiarze 5



8.3.2 Hexoomino - klocki o rozmiarze 6



8.4 Format pliku wejściowego

1. W pierwszej linii wielkość klocków (na przykład 6)
2. W drugiej linii typ algorytmu, dozwolone wartości to:
   - hk - algorytm heurystyczny
   - ok - algorytm optymalny
3. W trzeciej linii znajduje się ciąg liczb oddzielonych spacjami, lub jedna liczba:
   - jedna liczba oznacza ilość losowych klocków w zadaniu
   - ciąg liczb lista ilości kolejnych klocków dla których uruchomiamy nasz algorytm (ignorujemy zera końcowe) np. 2 5 1 1 0 0 5 3 oznacza, że

powinniśmy wziąć 2 klocki o numerze 1, 5 klocków o numerze 2, 1 klocek o numerze 3 itd. Ciąg można podać jednynie dla klocków o rozmiarze 5 lub 6. Indeksy klocków są przedstawione powyżej.

Plik może zawierać wiele takich sekwencji, wtedy zadania są obliczane kolejno, każde po wciśnięciu przycisku *Następne zadanie*. Aby uruchomić algorytm jedynie dla klocków o indeksie 1, musimy dodać na koniec ciągu nieznaczące zero. Na przykład dla 4 klocków o indeksie 1 trzeba podać wartość w trzeciej linii: 4 0. Wraz z projektem zawarte są przykładowe pliki wejściowe.