# Dokumentacja wstępna: Problem układania pudełek

## 1 Przedstawienie problemu

Dany jest zbiór pudełek, gdzie każde pudełko ma określoną szerokość oraz długość, które są reprezentowane jako para liczb naturalnych. Celem jest znalezienie najdłuższego ciągu pudełek, które można zapakować jedno w drugie. Pudełko A może zostać umieszczone w pudełku B, jeśli szerokość i długość A są nie większe niż szerokość i długość B. Pudełka można obracać o 90 stopni. Problem ten można rozwiązać za pomocą programowania dynamicznego.

## 2 Opis rozwiązania

Aby rozwiązać problem, zastosujemy następujące kroki:

1. Porządkujemy wymiary pudełek, tak aby dla każdego pudełka pierwsza współrzędna pary definiującej jego wymiar była nie mniejsza od drugiej (tzn. zakładamy, że szerokość jest zawsze nie mniejsza niż długość). Ten krok odpowiada ewentualnemu obrotowi pudełka o 90 stopni.

2. Sortujemy pudełka według szerokości rosnąco, a w przypadku równej szerokości – według długości malejąco. Jeżeli w ciągu $(w_i, l_i)_{i=1}^n$ niektóre pary się powtarzają, to powtórzenia możemy pominąć, ponieważ szukany podciąg wymiarów powinien być ściśle rosnący na obydwu współrzędnych.

3. Znajdujemy najdłuższy podciąg rosnący (LIS - Longest Increasing Subsequence) $(l_{i_j})_{j=1}^k$ długości pudełek. Do znalezienia LIS użyjemy programowania dynamicznego z binarnym wyszukiwaniem, co zapewni efektywność rozwiązania.

### 2.1 Dynamiczne wyszukiwanie LIS

Kluczową częścią rozwiązania problemu jest odnalezienie najdłuższego podciągu rosnącego $(l_{i_j})_{j=1}^k$, gdzie ciąg $(w_i, l_i)_{i=1}^n$ jest już odpowiednio posortowany, tzn. spełnia on warunek:

$$\forall_{1 \leq i < j \leq n} (w_i < w_j \text{ lub } (w_i = w_j \text{ oraz } l_i > l_j)). \tag{1}$$

W tym celu zastosujemy następujące kroki:

1. Tworzymy listę pustą *sub*, która w odpowiedni sposób będzie przechowywała indeksy niektórych wyrazów ciągu $(l_i)_{i=1}^n$.

2. Tworzymy $n$-elementową listę $parent = (p_i)_{i=1}^n$, gdzie $p_i = -1$ dla $i = 1, \ldots, n$. Listy *sub* oraz *parent* posłużą nam do odtworzenia LIS. Następnie postępujemy indukcyjnie:

3. W pierwszym kroku do listy *sub* dołączamy element $(w_1, l_1)$.

4. Załóżmy, że sprawdziliśmy pierwsze $k$ elementów ciągu $(l_i)_{i=1}^n$, gdzie $1 \leq k \leq n-1$. Załóżmy również, że do listy *sub* należy już $k'$ elementów. W $(k+1)$-tym kroku możliwe są dwa przypadki:

   (a) Ciąg $(l_{s_j})_{j=1}^{k'}$ zawiera tylko wyrazy mniejsze od $l_{k+1}$. Wówczas ciąg *sub* rozszerzamy o nowy element $k+1$, tzn. definiujemy $s_{k'+1} := k+1$.

   (b) W przeciwnym przypadku odnajdujemy najmniejszy możliwy wyraz $l_{s'_j}$ ciągu $(l_{s_j})_{j=1}^{k'}$, dla którego $l_{s_{j'}} \geq l_{k+1}$. Wówczas zastępujemy wyraz $s_{j'}$ w ciągu *sub* wyrazem $k+1$, tzn. definiujemy nową wartość $s_{j'} = k+1$. W tym wypadku długość ciągu *sub* nie ulega zmianie.

5. O ile wyraz $k+1$ nie znalazł się na pierwszym miejscu listy *sub*, to zmieniamy $(k+1)$-ty wyraz listy *parent* na poprzedni wyraz ciągu *sub* (tzn. wyraz $p_{k+1}$ ma teraz wartość $s_{k'}$ w przypadku (a), oraz wartość $s_{j'-1}$ w przypadku (b)).

6. Po sprawdzeniu $n$-tego wyrazu ciągu $(l_i)_{i=1}^n$ przystępujemy do odtworzenia LIS. W tym celu tworzymy listę pustą *lis*, do której będziemy dołączać odpowiednie wyrazy ciągu $(w_i, l_i)_{i=1}^n$.

7. Do listy *lis* dołączamy wyraz $(w_k, l_k)$, gdzie $k$ jest ostatnim wyrazem znajdującym się w liście *sub*.

8. W następnym kroku dołączamy do *lis* poprzednika elementu $(w_k, l_k)$ odczytanego z listy *parent*, tzn. wyraz $(w_{p_k}, l_{p_k})$. W analogiczny sposób odnajdujemy poprzednika elementu $(w_{p_k}, l_{p_k})$.

9. Procedurę dołączania poprzedników do *lis* kontynuujemy tak długo, aż dojdziemy do pewnego wyrazu $k_0$ listy *sub*, dla którego $p_{k_0} = -1$, tzn. do indeksu wyrazu ciągu $(w_i, l_i)_{i=1}^n$, który nie posiada poprzednika.

10. Lista *lis* z zamienioną kolejnością elementów (zapisana od końca) jest szukanym LIS.

Pseudokod powyższej procedury przedstawia Algorytm 1.

*Uwaga* 1. Zauważmy, że w każdej iteracji wyrazy listy *sub* są definiowane w taki sposób, że ciąg $(a_s)_{s \in sub}$ jest ciągiem niemalejącym. Dzięki temu krok 4 naszej procedury możemy wykonać w czasie logarytmicznym, o ile zastosujemy wyszukiwanie binarne.

# 3 Analiza poprawności i złożoności

## 3.1 Poprawność

Po zakończeniu działania programu odnaleziony ciąg będzie spełniał nasze założenia:

- Sortowanie według szerokości rosnąco, a następnie według długości malejąco zapewnia, otrzymany LIS będzie ciągiem ściśle rosnącym na obydwu współrzędnych.
- Algorytm LIS gwarantuje znalezienie najdłuższego podciągu pudełek, które można zapakować jedno w drugie.

Pozostaje uzasadnić, że algorytm LIS rzeczywiście zwraca najdłuższy podciąg rosnący.

**Algorithm 1** Najdłuższy podciąg rosnący (LIS)

**Require:** Lista $(w_i, l_i)_{i=1}^{n}$ spełniająca warunek (1)
**Ensure:** Najdłuższy rosnący podciąg (LIS)

1: sub $\leftarrow$ [] ▷ Lista przechowująca indeksy LIS
2: parent $\leftarrow [-1] \times n$ ▷ Tablica indeksów poprzedników
3: **for** $k \leftarrow 1$ to $n$ **do**
4: $x \leftarrow l_k$
5: **if** $\forall_{s \in sub} l_s < x$ **then**
6: append(sub, $k$)
7: $j' \leftarrow$ length(sub $- 1$) ▷ Indeks elementu $k$ w liście sub
8: **else**
9: $j' \leftarrow \arg\min_j(\{l_{s_j} : l_{s_j} \geq x, s_j \in \text{sub}\})$
10: sub$[j'] \leftarrow k$ ▷ Zamiana elementu na bardziej optymalny
11: **end if**
12: **if** $j' > 1$ **then** ▷ Przypisanie poprzednika wyrazowi $(w_k, l_k)$
13: parent$[k] \leftarrow$ sub$[j' - 1]$
14: **end if**
15: **end for**
16: ▷ Odtwarzanie LIS
17: lis $\leftarrow$ [] ▷ Najdłuższy podciąg rosnący wymiarów pudełek
18: $k \leftarrow$ sub[length(sub) $- 1$] ▷ Ostatni element listy sub
19: **while** $k \neq -1$ **do**
20: append(lis, $(w_k, l_k)$)
21: $k \leftarrow$ parent$[k]$
22: **end while**
23: **return Reverse**(lis) ▷ Lista lis zapisana od końca

**Twierdzenie 1.** *Niech $(a_i)_{i=1}^n$ będzie skończonym ciągiem liczbowym. Wówczas algorytm LIS zwraca najdłuższy podciąg rosnący $(a_{i_j})_{j=1}^k$ ciągu $(a_i)_{i=1}^n$.*

*Dowód.* Tezy dowiedziemy poprzez indukcję po długości $n$ ciągu $(a_i)_{i=1}^n$. Dla $n = 1$ teza jest oczywista, ponieważ otrzymamy podciąg długości 1.

Załóżmy, że teza zachodzi dla ciągów długości mniejszej od $n$, gdzie $n \geq 2$. Wówczas w $n$-tym kroku algorytmu LIS za pomocą list $sub = (s_i)_{i=1}^{n'}$ oraz $parent = (p_i)_{i=1}^n$ utworzonych w poprzednich iteracjach możemy odtworzyć najdłuższy podciąg rosnący ciągu $(a_i)_{i=1}^{n-1}$, którego długość oznaczymy przez $n'$. Załóżmy dodatkowo, że ciąg $sub$ ma następującą własność $(*)$: dla każdego $1 \leq j \leq n'$ wyraz $a_{s_j}$ jest ostatnim wyrazem pewnego podciągu rosnącego $(a_{i_k})_{k=1}^{s_j}$ długości $j$ i nie istnieje podciąg rosnący ciągu $(a_i)_{i=1}^{n-1}$, którego ostatni wyraz jest mniejszy od wartości $a_{s_j}$.

W $n$-tej iteracji pętli opisanej w liniach 3–15 pseudokodu Algorytmu 1 możliwe są następujące przypadki:

(i) Wyraz $a_n$ jest większy od wszystkich wyrazów podciągu $(a_{s_i})_{i=1}^{n'}$. Wówczas listę $sub$ przedłużamy o indeks $n$, zatem $n' + 1$-ty wyraz listy jest indeksem ostatniego wyrazu najdłuższego podciągu rosnącego długości $n' + 1$.

(ii) W przeciwnym wypadku elementem $n$ zastępujemy pewien wyraz $s_{j'}$ listy $sub$, gdzie z założenia indukcyjnego $a_{s_{j'}}$ jest ostatnim wyrazem pewnego podciągu rosnącego długości $j'$ ciągu $(a_i)_{i=1}^{n-1}$, nie istnieje podciąg długości $j'$ o ostatnim wyrazie mniejszym od $a_{s_{j'}}$ oraz $a_{s_{j'}} \geq a_n$. Nowa wartość $s_{j'}$ jest zatem indeksem najmniejszego możliwego wyrazu podciągu rosnącego długości $j'$ ciągu $(a_i)_{i=1}^n$.

W obydwu przypadkach lista $sub$ wciąż spełnia założenie indukcyjne $(*)$.

Jeżeli zachodzi przypadek (i), to algorytm LIS zwróci najdłuższy możliwy podciąg rosnący, który ma długość $n' + 1$. Ponieważ z założenia indukcyjnego odnaleziony w $n - 1$-tej iteracji najdłuższy podciąg rosnący miał tylko $n'$ wyrazów, to teza o poprawności rozwiązania dla ciągu długości $n$ jest spełniona.

Przypuśćmy zatem, że zachodzi przypadek (ii), ale istnieje podciąg rosnący ciągu $(a_i)_{i=1}^n$ długości $n'+1$ (którego ostatnim wyrazem jest $a_n$). Wówczas istnieje podciąg rosnący długości $n'$ ciągu $(a_i)_{i=1}^{n-1}$, którego ostatni wyraz jest mniejszy niż $a_n$. Oznaczmy ten wyraz jako $a_l$. Ponieważ zachodzi przypadek (ii), to $a_{s_{n'}} \geq a_n$ na mocy Uwagi 1, co razem daje $a_l < a_n \leq a_{s_{n'}}$. Otrzymujemy sprzeczność, ponieważ z założenia indukcyjnego wyraz $a_{s_{n'}}$ jest najmniejszym możliwym wyrazem podciągu rosnącego $(a_i)_{i=1}^{n-1}$ długości $n'$.

Na mocy prawa indukcji algorytm LIS zwróci najdłuższy podciąg rosnący ciągu $(a_i)i = 1^n$. □

## 3.2 Złożoność czasowa

Uporządkowanie współrzędnych i posortowanie ciągu $(w_i, l_i)_{i=1}^n$ możemy wykonać w $O(n \log n)$ krokach, jeżeli użyjemy algorytmu sortującego o złożoności $O(n \log n)$, np. *merge sort*.

Przyjrzyjmy się złożoności czasowej Algorytmu 1.

1. Pętlę opisaną w liniach 3–15 pseudokodu wykonujemy $n$ razy.

2. W każdej iteracji tej pętli odnajdujemy liczbę $j'$ w czasie logarytmicznym przy pomocy wyszukiwania binarnego.

3. Liczba pozostałych operacji wykonywanych w pojedynczej iteracji nie zależą od wartości $n$ i $k$. Zatem $k$-ta iteracja ma złożoność czasową nieprzekraczającą $a \log k$, gdzie $a$ jest pewną stałą liczbą.

4. Odtwarzanie LIS (linie 17–24) wykonamy w $cn$ krokach, ponieważ pętla **While** opisaną w liniach 20–22 zostanie wykonana maksymalnie $n$ razy. Pozostałe linie wykonamy w maksymalnie $d$ krokach, gdzie $d$ jest stałą liczbą.

Zatem całkowita złożoność czasowa Algorytmu 1 wynosi

$$\sum_{k=1}^{n} a \log k + cn + d = O(n \log n).$$

Cały program będzie miał więc złożoność obliczeniową równą $O(n \log n)$.

# 4 Opis wejścia/wyjścia

## 4.1 Format pliku wejściowego

Plik wejściowy będzie miał format .csv, którego pierwszym wierszem będą nagłówki `w,l`, a następne $n$ wierszy będzie zawierało wymiary pudełek – szerokość i długość oddzieloną przecinkiem, gdzie $1 \leq n \leq 100000$. Maksymalna długość i szerokość pudełka będzie wynosiła $10^9$.

## 4.2 Przykładowe wejście

```
w,l
5,1
6,7
6,2
2,3
3,4
```

## 4.3 Format pliku wyjściowego

Program będzie tworzył dwa pliki wyjściowe:

1. Plik `lis.csv` będzie miał analogiczną strukturę, co plik wejściowy. Kolejne wiersze będą zawierały wyrazy najdłuższego podciągu wymiarów pudełek odnalezionego przez nasz algorytm.

2. Plik `dlugosc.txt` będzie zawierał pojedynczą liczbę – długość odnalezionego podciągu.

## 4.4 Przykładowe wyjście

Plik `lis.csv`:

```
w,l
5,1
6,2
7,6
```

Plik `dlugosc.txt`:

```
3
```

W celu wizualizacji wyników można również utworzyć wykres, który naniesie na siatkę uporządkowane wymiary wszystkich pudełek (pierwsza współrzędna większa od drugiej) oraz wyróżni punkty, które znalazły się w najdłuższym podciągu zwróconym przez program.

Rysunek 1: Przykładowy wykres zwrócony przez program