Zadanie Klocki – LOGIA 21 (2020/21), etap 3

Treść zadania

Ola bawi się klockami. Wszystkie klocki w zestawie są prostopadłościanami o takiej samej długości boku kwadratu w podstawie, ale mogą różnić się wysokością. Ola układa wieżę – ustawia klocek jeden na drugim, stykając je podstawami. Stara się ułożyć wieżę o określonej wysokości tak, aby użyć jak najmniej klocków. Pomóż Oli i napisz program, który dla danego zestawu klocków i podanej wysokości wieży policzy minimalną liczbę potrzebnych klocków lub stwierdzi, że wieży o podanej wysokości nie da się ułożyć.

Wejście:

W pierwszym wierszu wejścia znajduje się liczba całkowita **n** z zakresu od 1 do 20 określająca liczbę rodzajów klocków w zestawie.

Kolejne n wierszy wejścia zawiera po dwie liczby całkowite **h** i **m** określające odpowiednio wysokość klocka (**h**) i liczbę klocków (**m**) o wysokości **h** w zestawie, **1 ≤ h ≤ 500**, **1 ≤ m ≤ 10 000**. Wysokość klocka w kolejnym wierszu jest większa od wysokości klocka w wierszu poprzednim.

W ostatnim wierszu wejścia znajduje się jedna liczba całkowita z zakresu od 1 do 10 000 określająca wysokość wieży do ułożenia.

Wyjście:

Liczba całkowita dodatnia określająca minimalną liczbę klocków potrzebnych do zbudowania wieży o podanej wysokości lub liczba -1, jeśli nie da się ułożyć wieży o podanej wysokości.

	Przykład 1	*Przykład 2*	*Przykład 3*	*Przykład 4*
Wejście	**5** **1 7** **2 1** **3 1** **4 1** **5 1** **3**	**3** **1 5** **8 5** **12 5** **11**	**6** **2 3** **3 2** **7 1** **8 5** **13 4** **15 1** **27**	**2** **2 7** **4 1** **3**
Wyjście	**1**	**4**	**4**	**-1**

Omówienie rozwiązania

W zadaniu został ukryty jeden z klasycznych algorytmów – problem wydawania reszty. W tym przypadku sprowadza się on do wybrania z danego zbioru klocków o określonych wysokościach takiego układu, który pozwoli ułożyć żądaną wysokość wieży przy użyciu minimalnej liczby klocków. Przy rozwiązywaniu tego problemu można zastosować dwa podejścia – zachłanne i dynamiczne. Niestety algorytm zachłanny nie zawsze daje poprawne wyniki. Na przykład dla klocków o wysokości 5, 4, 4, 1, 1, 1 i wysokości wieży równej 8 otrzymujemy wynik 4 (5, 1, 1, 1). Poprawnym rozwiązaniem jest odpowiedź 2 (4, 4). W tym wypadku należało wdrożyć rozwiązanie dynamiczne, które choć trudniejsze

co do idei, zawsze daje poprawny wynik. Ze względu na prostotę rozwiązania zachłannego i to, że można za nie było otrzymać 40% punktów zostanie ono również omówione.

Algorytm zachłanny

Sposób postępowania polega na odejmowaniu od wysokości wieży największej dostępnej wysokości klocka nie przewyższającej aktualnej wysokości wieży. Czynności powtarzamy, dopóki wysokość wieży jest większa od zera i nie zabraknie klocków.

Na przykład dla klocków o wysokościach 5, 4, 4, 1, 1, 1 i wysokości wieży 8 wybierzemy:

wysokość klocka	pozostała wysokość wieży
5	8 – 5 = 3
1	3 – 1 = 2
1	2 – 1 = 1
1	1 – 1 = 0 koniec działania algorytmu

Ułożona została wieża z 4 klocków.

Gdy klocki skończą się i nie osiągniemy wysokości wieży **0**, wynikiem algorytmu jest **-1**.

Mając listę uporządkowanych niemalejąco wysokości klocków (**klocki[0..n-1]**) oraz uwzględniając powyższe założenia, algorytm wygląda tak:

```
1.   ile ← 0
2.   i = n - 1
3.   dopóki i ≥ 0 i wysokość wieży > 0
4.       jeśli wysokość wieży >= klocki[i]
5.           wysokość wieży  ← wysokość wieży - klocki[i]
6.           ile ← ile + 1
7.       i ← i - 1
8.   jeśli wysokość wieży > 0
9.       wynik -1
10.  wynik ile
```

Komentarz: listę uporządkowanych niemalejąco wysokości klocków można w łatwy sposób utworzyć wczytując dane (są one już wstępnie posortowane względem wysokości klocków).

Programowanie dynamiczne

Algorytm polega na rozpatrywaniu kolejnych wysokości klocków i obliczeniu ich minimalnej liczby niezbędnej do ułożenia wież o wysokościach od 0 do k. Przy analizowaniu kolejnej wysokości klocka wykorzystywane są informacje wcześniej uzyskane. W przypadku, gdy uzyskamy mniejszą wartość – poprawiamy wynik.

W pierwszym kroku algorytmu zakładamy, że do ułożenia wież o wysokościach od 1 do k potrzebujemy nieskończenie wiele klocków. W tym przypadku może to być dowolna liczba większa od 10000 (maksymalna wysokość wieży) – np. 20000. Na przykład dla wysokości wieży równej 8 lista przechowująca informacje o liczbie klocków dla każdej potencjalnej wysokości wieży od 0 do 8 będzie wyglądać tak:

wysokość wieży	0	1	2	3	4	5	6	7	8
liczba klocków	0	20000	20000	20000	20000	20000	20000	20000	20000

Dla każdego rozpatrywanego klocka powyższa lista będzie aktualizowana. Należy sprawdzić, czy dodając dla danej wysokości wieży kolejny klocek, uzyska się liczbę klocków mniejszą, niż dotychczas wyznaczoną.

Prześledźmy przykład dla klocków o wysokościach 5, 4, 4, 1, 1, 1. Rozpatrujemy wyłącznie wysokości wież większe lub równe od wysokości analizowanego klocka. Listę analizujemy zawsze od strony prawej do lewej, aby dla danego klocka uwzględnić wyłącznie te wysokości, które jeszcze nie zostały w danym kroku zaktualizowane. Dla klocka o wysokości 5 będą analizowane kolejno wieże o wysokościach 8, 7, 6, 5. Dla wysokości 8, 7 i 6 nie jesteśmy w stanie polepszyć wyniku (20000 + 1 > 20000). Jedynie do wieży o wysokości równej 0 opłaca się dodać klocek o wysokości 5, wtedy w komórce tabeli odpowiadającej wysokości 5 będzie można umieścić mniejszą wartość (0 + 1 < 20000). Tabelę należy uzupełnić dla pozostałych klocków stosując powyższą zasadę. Po aktualizacji dla wszystkich klocków tabela będzie wyglądać tak:

0	1	2	3	4	5	6	7	8	wysokości klocków
0	20000	20000	20000	20000	20000	20000	20000	20000	start
0	20000	20000	20000	20000	**0 + 1**	20000	20000	20000	5
0	20000	20000	20000	**0 + 1**	1	20000	20000	20000	4
0	20000	20000	20000	**1**	1	20000	20000	**1 + 1**	4
0	**0 + 1**	20000	20000	1	**1**	**1 + 1**	20000	2	1
0	**1**	**1 + 1**	20000	1	1	**2**	**2 + 1**	2	1
0	1	**2**	**2 + 1**	1	1	2	3	**2**	1

Wszystkie zamiany zostały zaznaczone kolorem szarym, kremowym kolorem zaznaczono wartość przez aktualizacją. Po rozpatrzeniu wszystkich klocków w polu oznaczony żółtym kolorem znajduje się poszukiwana informacja (minimalna liczba klocków). Jeśli to pole zawiera wartość 20000 – oznacza to, że wieży o podanej wysokości nie da się ułożyć.

Co ciekawe, nie ma znaczenia od jakiego klocka zaczynamy analizę (kolejność może być dowolna).

Dla klocków rozpatrywanych w kolejności 1, 1, 1, 4, 4, 5 otrzymamy ten sam wynik końcowy:

0	1	2	3	4	5	6	7	8	wysokości klocków
0	20000	20000	20000	20000	20000	20000	20000	20000	start
0	**0 + 1**	20000	20000	20000	20000	20000	20000	20000	1
0	**1**	**1 + 1**	20000	20000	20000	20000	20000	20000	1
0	1	**2**	**2 + 1**	20000	20000	20000	20000	20000	1
0	**1**	**2**	**3**	**0 + 1**	**1 + 1**	**2 + 1**	**3 + 1**	20000	4
0	1	2	3	**1**	2	3	4	**1 + 1**	4
0	**1**	**2**	3	1	1	**1 + 1**	**2 + 1**	**2**	5

Zapis algorytmu dla listy wysokości klocków (klocki[0..n-1]):

```
1.  ile_klockow[0] ← 0
2.  dla każdego i od 1 do wysokość wieży
3.      ile_klockow[i] ← 20000
4.  dla każdego i od 0 do n - 1
5.      dla każdego j od wysokość wieży do klocki[i]
6.          ile_klockow[j] ← min(ile_klockow[j],
                                 ile_klockow[j - klocki[i]] + 1)
7.  jeśli ile_klockow[wysokość wieży] = 20000
8.  wynik -1
9.  wynik ile_klockow[wysokość wieży]
```

Rozwiązanie w języku Python

Wczytywanie danych połączone z tworzeniem listy klocków

```
1.  n = int(input())
2.  klocki = []
3.  for i in range(n):
4.      x, y = input().split()
5.      for j in range(int(y)):
6.          klocki.append(int(x))
7.  wys = int(input())
8.  print(policz(klocki, wys))
```

Algorytm zachłanny – nie daje poprawnych odpowiedzi dla wszystkich testów

```
9.  def policz(klocki, wys):
10.     i = len(klocki) - 1
11.     ile = 0
12.     while i >= 0 and wys > 0:
13.         if wys >= klocki[i]:
14.             wys = wys - klocki[i]
15.             ile = ile + 1
16.         i = i - 1
17.     if wys > 0:
18.         return -1
19.     return ile
```

Programowanie dynamiczne

```
9.  def policz(klocki, wys):
10.     ile_klockow = [20000 for i in range(wys + 1)]
11.     ile_klockow[0] = 0
12.     for ele in klocki:
13.         for i in range(wys, ele - 1, -1):
14.             ile_klockow[i] = min(ile_klockow[i], ile_klockow[i - ele] + 1)
15.     if ile_klockow[wys] == 20000:
16.         return -1
17.     return ile_klockow[wys]
```

Testy

Testy zostały podzielone na 5 grup. Testy dla 1 i 2 grupy dają takie same wyniki dla algorytmu zachłannego i programowania dynamicznego. W pierwszej grupie (do 30 klocków maksymalnie 3 rodzajów) jeden wynik jest dodatni a drugi -1. W drugiej grupie (do 150 klocków maksymalnie 15 rodzajów) oba wyniki są dodatnie. Pozostałe test dają błędne wyniki dla algorytmu zachłannego (co najmniej jeden z testów w grupie daje błędny wynik dla tego algorytmu). Dane do testów grup 3-5 różnią się wielkością.

Grupa testów	**Wejście**	**Wynik**
I	3 1 10 2 10 5 10 34	8
	4 14 1 21 2 50 2 67 1 28	-1
II	6 1 4 4 3 5 2 10 5 40 2 50 8 85	3
	10 1 2 4 2 5 2 10 12 20 5 50 2 60 34 70 23 80 5 99 6 1000	13

III	9 1 10 2 10 5 10 10 10 20 10 50 10 100 10 200 10 500 10 3758	12
	15 1 10 2 10 3 10 4 10 5 10 10 10 20 10 30 10 40 10 50 10 100 10 200 10 300 10 400 10 500 10 9876	27
IV	3 1 3 4 5 5 4 8	2
	3 1 3 4 5 5 4 44	-1

V	15 1 1 2 1 3 1 10 10 40 10 50 10 100 100 150 100 200 100 250 100 300 100 350 100 400 100 450 100 500 100 9981	23
	20 2 100 4 100 6 100 8 100 10 100 12 100 14 100 16 100 18 100 20 100 22 100 24 100 26 100 28 100 30 100 32 100 34 100 36 100 38 100 40 100 321	-1