

0 1 2 3 4 5 . 6  
 3 4 1 4 1 1  
 0 0 0 2 7  
 0  
 wynik 2 3

3 0 1 6 2 3 4 | 9 5 6

## Wstęp do programowania\*

I kolokwium

2022/2023

**Uwaga:** każde zadanie proszę rozwiązać na osobnej, podpisanej kartce.  
Należy podać złożoność czasową i pamięciową rozwiązań.

1. Ciąg nazwiemy schodkowym, jeśli jest ciągiem kolejnych liczb naturalnych. Mamy dany ciąg  $n$  liczb naturalnych z zakresu od 0 do  $n$ , który chcemy podzielić na minimalną liczbę (niekoniecznie spójnych) podciągów schodkowych. Na przykład, ciąg  $(3, 0, 1, 6, 2, 3, 4, 4, 5, 6)$  można podzielić na minimalnie 3 podciągi schodkowe, np.:  $(0, 1, 2, 3, 4)$  ( $3, 4, 5, 6$ ) i  $(6)$ .

Napisz funkcję `int schodki(int n, int t[])`, która dla danej tablicy  $n$  liczb naturalnych (z zakresu  $\{0, \dots, n\}$ ) wyznaczy podział tablicy  $t$  na minimalną liczbę podciągów schodkowych i da w wyniku liczbę tych podciągów.

2. Rozważamy macierz o wymiarach  $2^k \times 2^k$ , dla  $k \in \mathcal{N}$ , wypełnioną liczbami od 0 do  $2^{2k} - 1$  w pewnej kolejności. Taką macierz nazwiemy *cieniowaniem*, jeśli:

- Dla  $k = 0$ , jednoelementowa macierz zawierająca 0 jest cieniowaniem.
- Dla  $k > 0$ , dzielimy macierz na ćwiartki wielkości  $2^{k-1} \times 2^{k-1}$  i wstawiamy kolejne liczby na przemian do ćwiartek: górnej lewej, dolnej prawej, górnej prawej i dolnej lewej. W każdej ćwiartce liczby wstawiamy tak jak w cieniowaniu  $2^{k-1} \times 2^{k-1}$ .

Dla  $k = 1, 2, 3$ , cieniowania wyglądają tak:

	0	1	
0	0	2	
1	3	1	

  

	0	1	2	3	
0	0	8	2	10	
1	12	4	14	6	
2	3	11	1	9	
3	15	7	13	5	

	0	1	2	3	4	5	6	7
0	0	32	8	40	2	34	10	42
1	48	16	56	24	50	18	58	26
2	12	44	4	36	14	46	6	38
3	60	28	52	20	62	30	54	22
4	3	35	11	43	1	33	9	41
5	51	19	59	27	49	17	57	25
6	15	47	7	39	13	45	5	37
7	63	31	55	23	61	29	53	21

Napisz funkcję `int **cieniowanie(int k)` która zwraca cieniowanie rozmiaru  $2^k \times 2^k$  (jako tablicę wskaźników do tablic liczb).

Wskazówka: Taką tablicę możesz utworzyć w następujący sposób:

```

int **t = (int**)malloc(sizeof(int*) * ...);
for (int i = 0; i < ...; i++)
  t[i] = (int*)malloc(sizeof(int) * ...);
  
```

a do jej komórek można się odwoływać: `t[i][j]`.