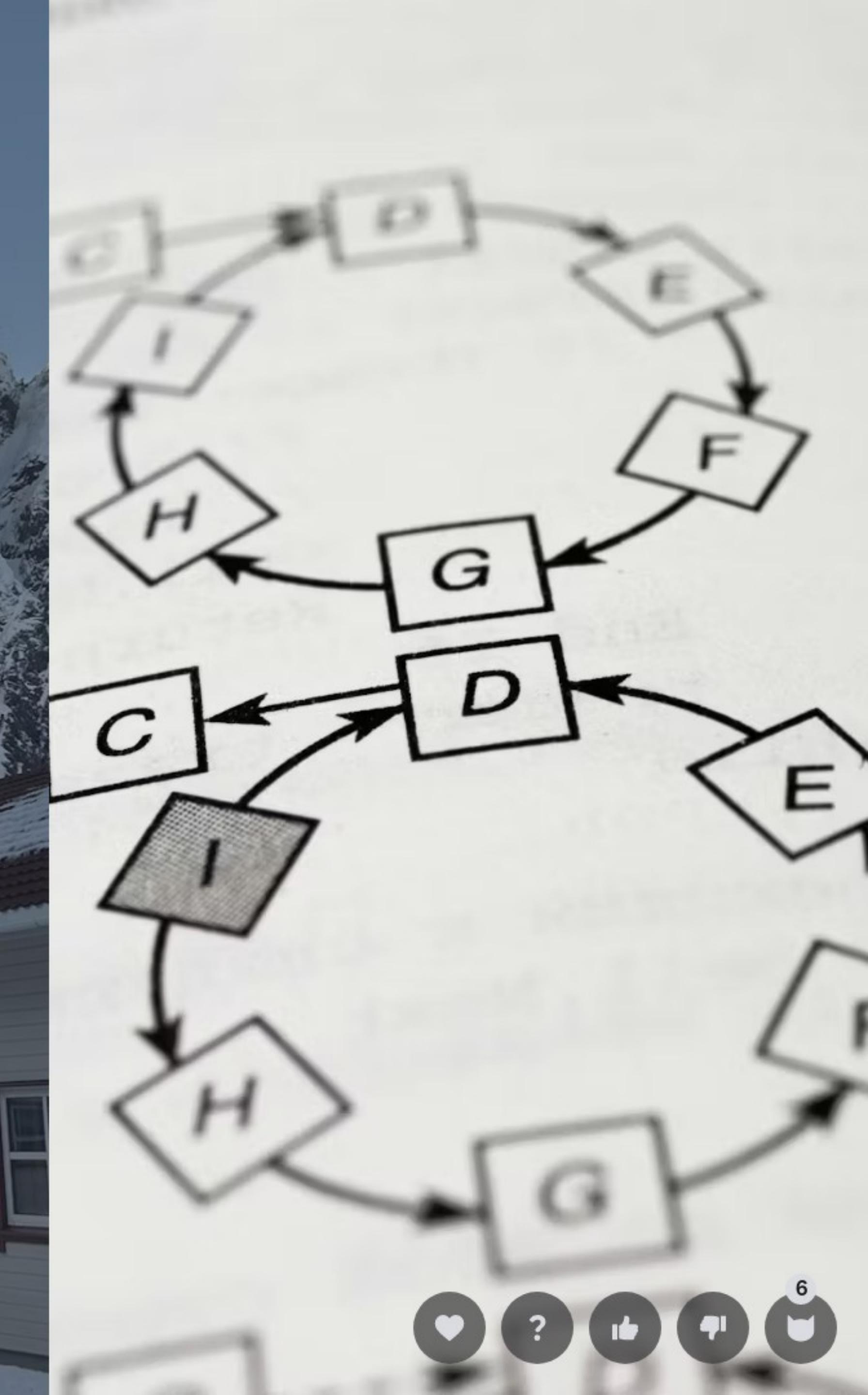




Algorytmy i Struktury Danych

Piękne miejsce, piękna historia, smutne zakończenie



Czy są pytania do dotychczasowego materiału?

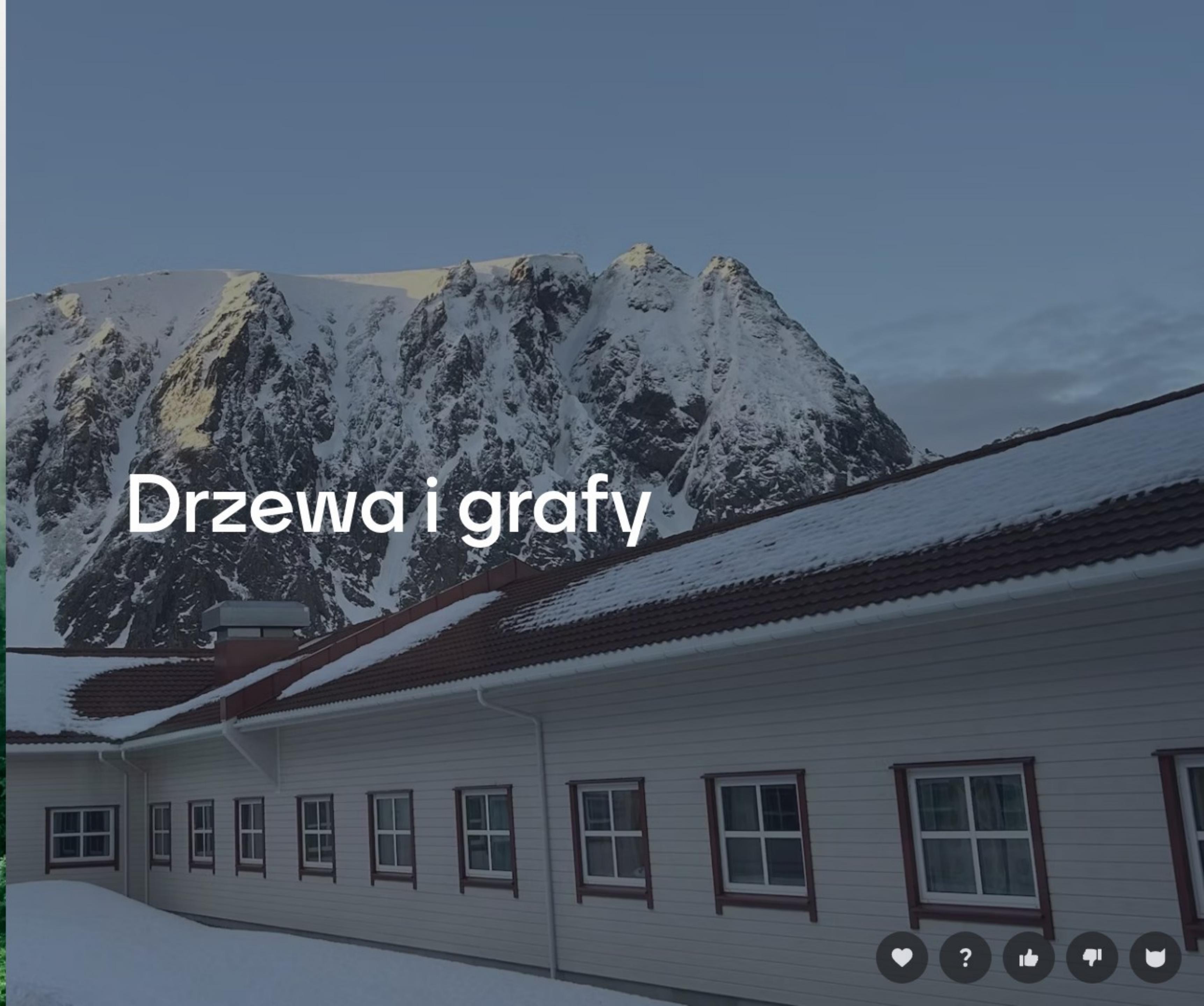
0

TAK

0

NIE



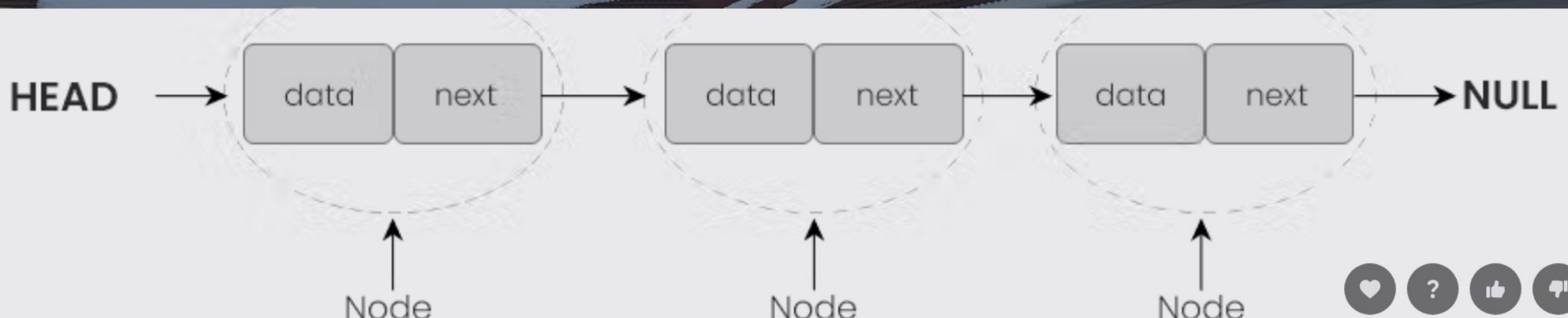


Drzewa i grafy



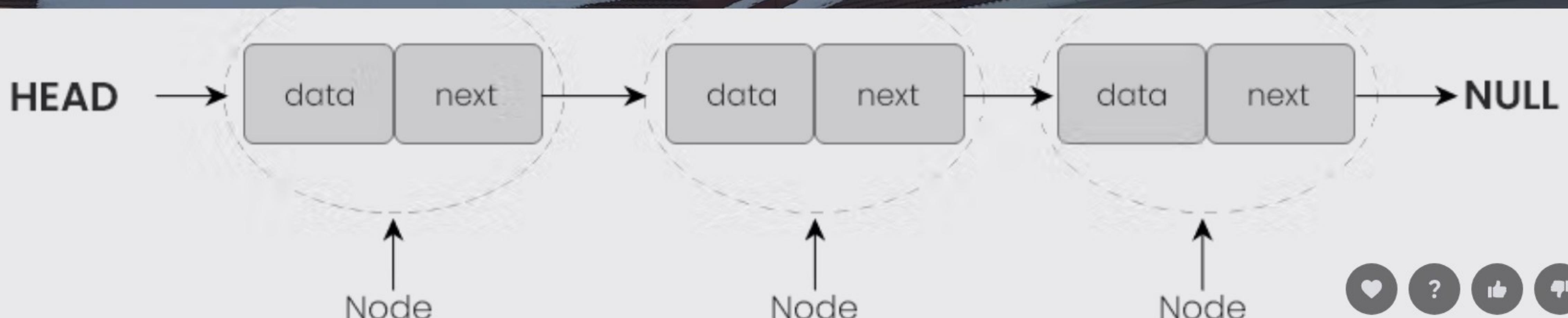
Listy wiązane (*linked list*)

- Każdy element jest węzłem listy
- Żeby dostać się do dalszych elementów, musimy przejść element po elemencie do przodu
- Możemy szybko usunąć elementy, "przepinając" poprzedni element na pierwszy element za nimi
- Możemy również szybko wpiąć nowe elementy do środka
- Jak to zaimplementować? (klasa vs krotka)



Przykład - napiszcie prostą listę wiązaną

1. Napiszcie klasę z dwoma polami *value* i *next*
2. *value* przechowuje wartość w węźle, a *next* przechowuje referencję na następny węzeł (lub **None**)
3. Metoda *get(i)* zwraca wartość *i*-tego elementu listy
4. Metoda *set(i, v)* ustawia wartość *v* pod indeksem *i*
5. Metoda *insert(i, v)* tworzy nowy węzeł pomiędzy dotychczasowymi węzłami *i-1* i *i* z wartością *v*
6. Metoda *remove(i)* usuwa *i*-ty węzeł z listy



Gorliwość a leniwość

- Czasami musimy coś znaleźć w ogromnym zbiorze danych
- Powiedzmy, mamy listę miast w kraju i szukamy jakiegokolwiek miasta, w którym działa Krzysztof Jarzyna
- Możliwe, że musimy najpierw wykonać ciężkie operacje zanim stwierdzimy, czy to ten element
- Musimy znaleźć wszelkie biznesy w mieście i musimy sprawdzić, do kogo należą
- Możemy wykonać te operacje na wszystkich elementach a następnie je przeszukać
- Co, jeśli bardzo szybko natrafimy na nasz element?
- Porównajmy operacje leniwe i gorliwe





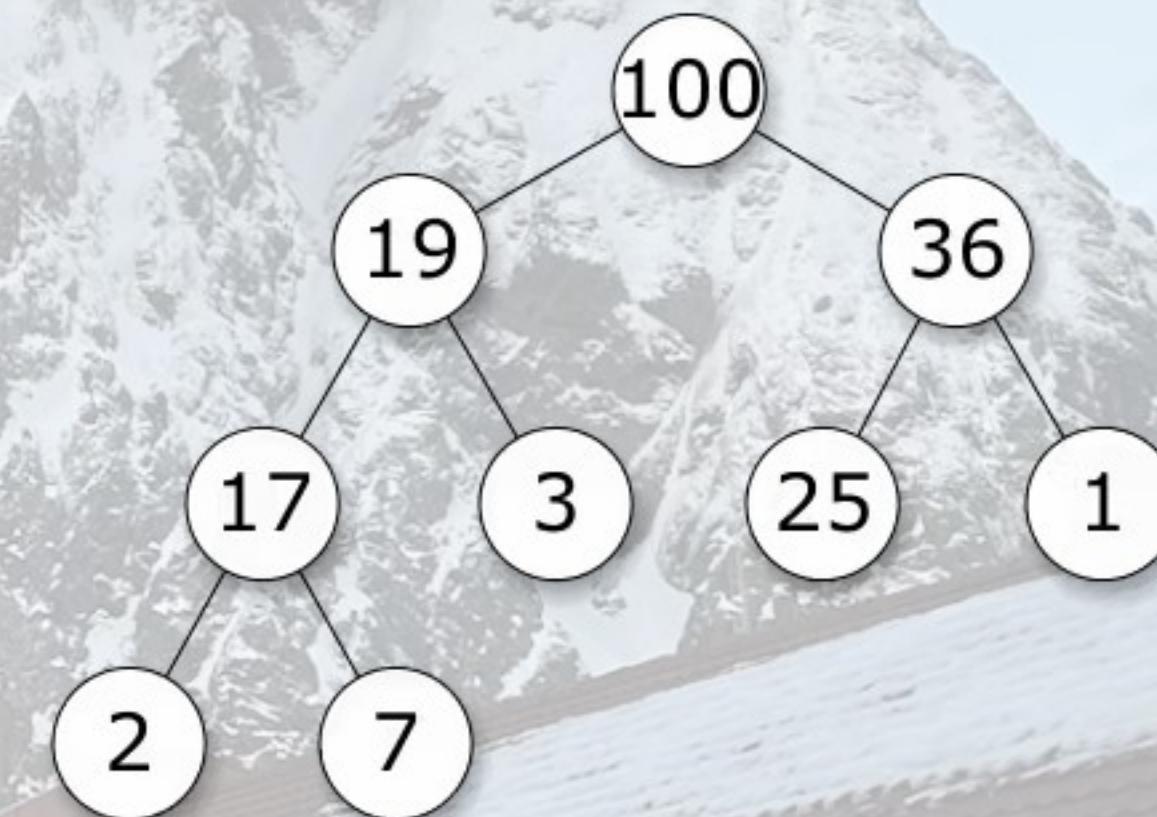
Leniwe sortowanie

- Posortowaliśmy wielką listę
- Potrzebujemy dodać do listy kolejny element
- Chcemy, żeby lista była dalej posortowana
- Rozwiązaniem są kopce (*heap*)

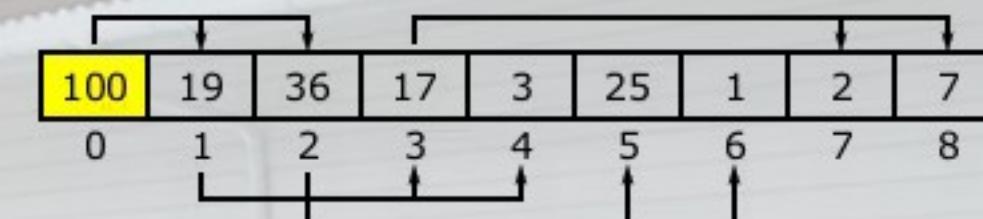
Kolejka priorytetowa

- Stwórz strukturę drzewiastą, gdzie każdy węzeł może mieć do dwóch dzieci
- W korzeniu (na szczycie) przechowujemy minimum (maximum, jeśli odwrócimy działanie)
- Każdy węzeł ma pod sobą większe lub równe elementy
- Wiersze zapełniamy od lewej do prawej
- Gdy dodajemy element na kopiec, dodajemy go w ostatnim wierszu i przesuwamy do góry, aż znajdzie się na swoim miejscu
- Gdy usuwamy element, wstawiamy ostatni element na szczyt i przesuwamy go w dół

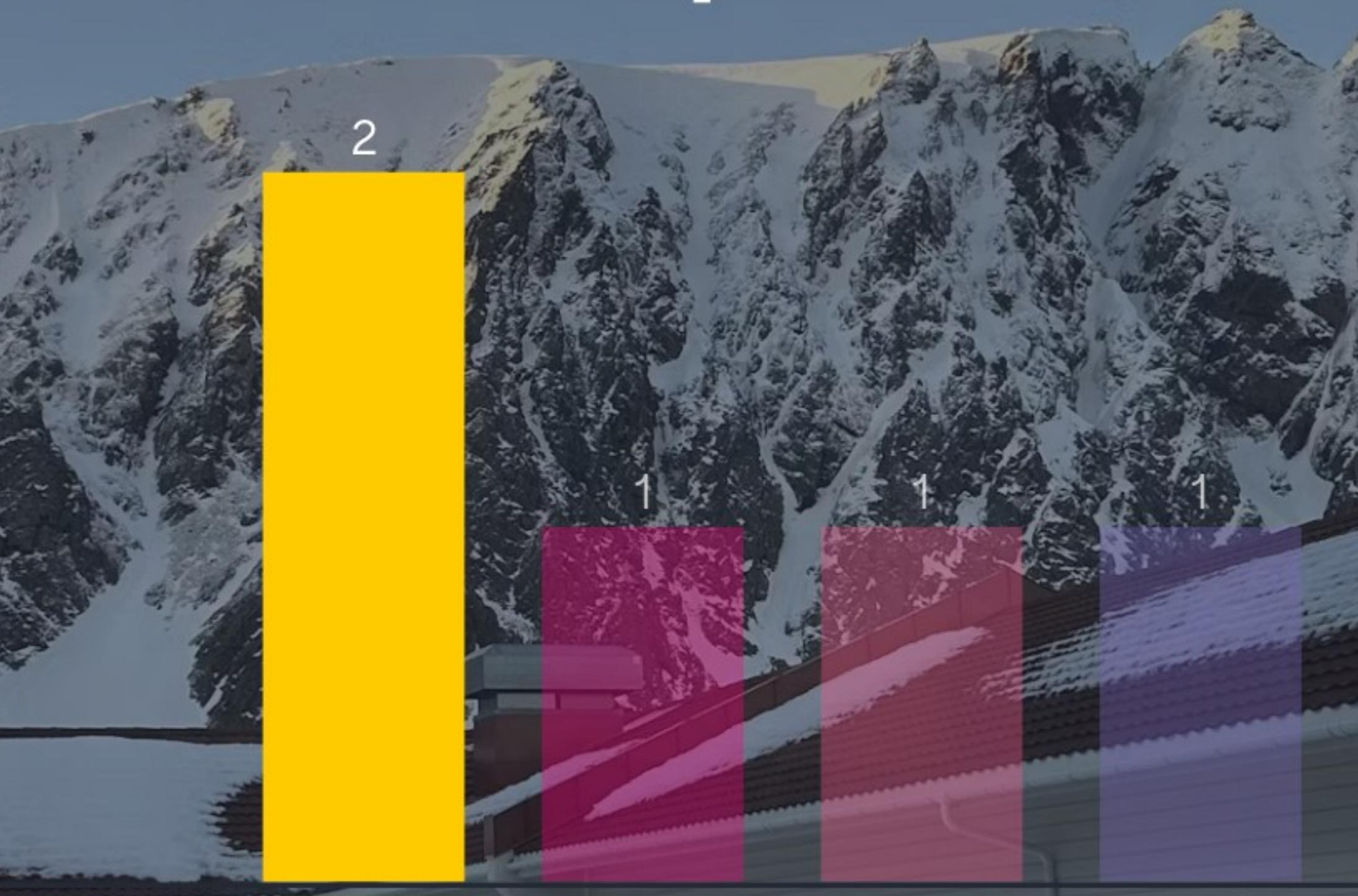
Tree representation



Array representation



Jaką wysokość ma kopiec?



Sortowanie przez kopcowanie (*heapsort*)

W Pythonie mamy dostępny pakiet `heapq`, który pozwala na łatwe tworzenie kopków (*heapify*), dodawanie nowego elementu (*heappush*) i usuwanie minimum (*heappop*). Tworząc kopiec z elementów listy a następnie ściągając kolejne minima uzyskujemy posortowaną listę.

The background of the slide features a wide-angle photograph of a majestic mountain range. The mountains are covered in patches of white snow, particularly on their upper ridges and peaks. The sky above is a clear, pale blue. In the foreground, the dark, textured roofs of several wooden houses are visible, partially obscured by snow. The overall scene is one of a cold, serene winter landscape.

Inne rodzaje kopców

- Min-max kopce
- Kopce dwumianowe
- Kopce Fibonacciego

Jaka jest złożoność heapsort?

$\Theta(n)$ $\Theta(n \log n)$ $O(n \log n)$ $\Theta(n^2)$ $O(n^2)$



Drzewa przeszukiwania binarnego (BST)

- Co, jeśli chcemy szybko znajdować elementy na samosortującej się strukturze?
- Zamiast dwóch większych, niech lewe dziecko będzie mniejsze a prawe większe
- Dodawanie i szukanie przebiega tak samo
- Porównaj wartość z korzeniem
- Jeśli jest równa, znaleźliśmy element
- Jeśli mniejsza, schodzimy w lewe poddrzewo
- Jeśli większa, schodzimy w prawe

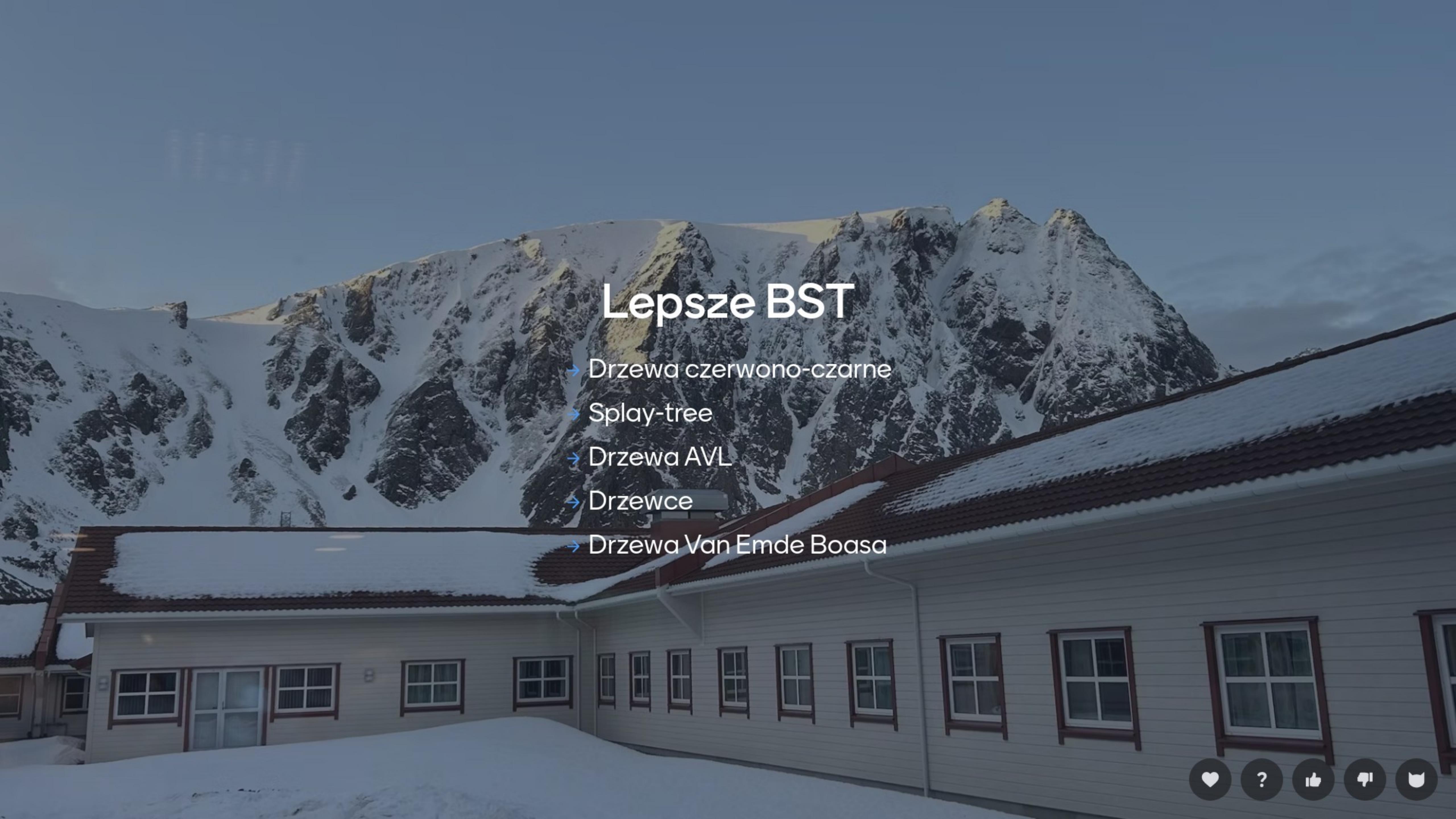


THE CODE

Jaką głębokość ma takie drzewo przeszukiwania binarnego?

5

$\Theta(1)$ \times $O(1)$ \times $\Theta(\log n)$ \times $O(\log n)$ \times $\Theta(n)$ \times $O(n)$ \checkmark

The background of the slide features a wide-angle photograph of a majestic, snow-covered mountain range under a clear blue sky. In the foreground, the dark brown roof and white-painted wooden exterior of a long building are visible, partially obscured by snow.

Lepsze BST

- Drzewa czerwono-czarne
- Splay-tree
- Drzewa AVL
- Drzewce
- Drzewa Van Emde Boasa



Przeszukiwanie

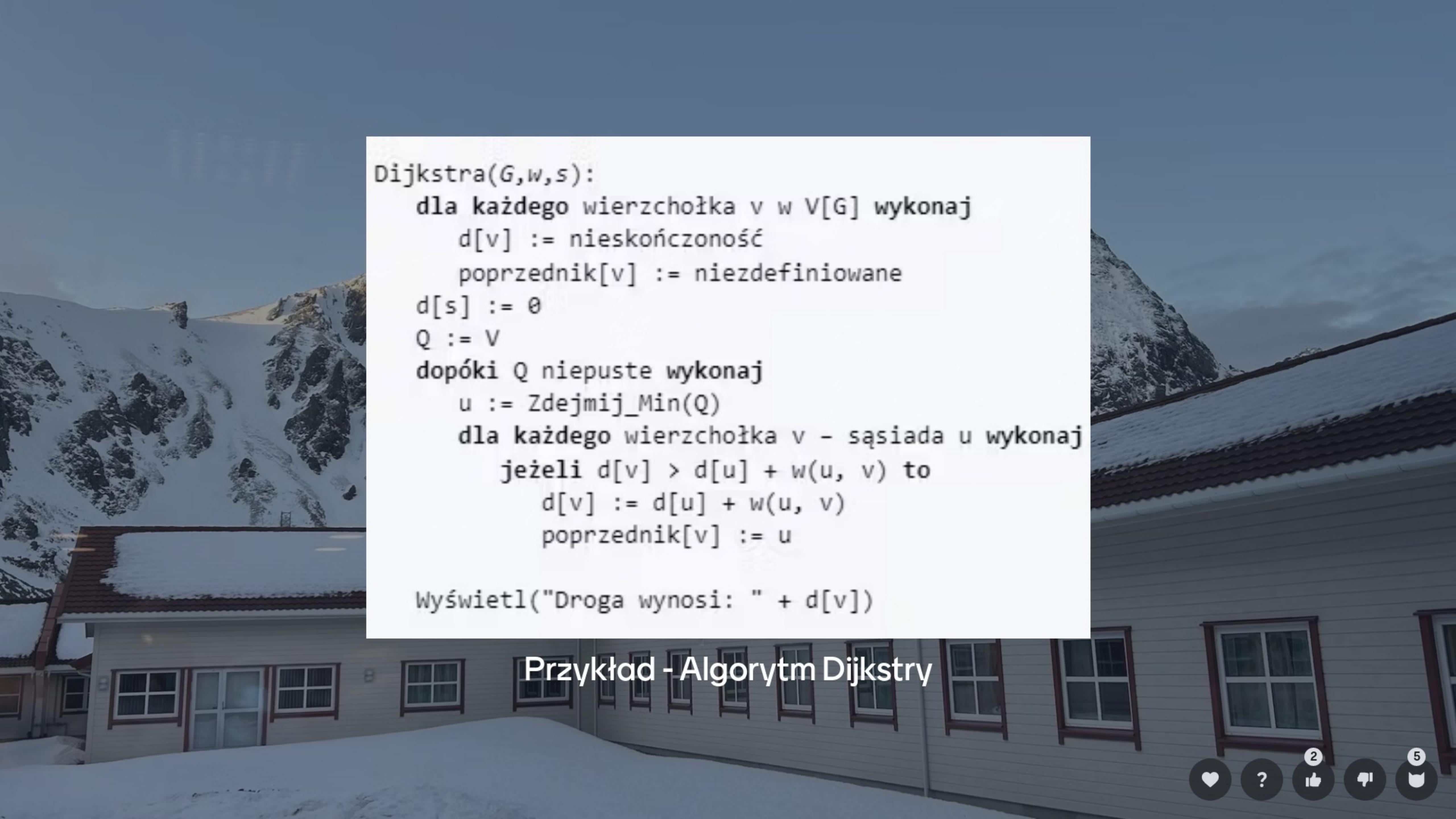
- Założmy, że mamy system plików - foldery z folderami
- Musimy znaleźć wszystkie pliki .csv w tym systemie
- Jak powinniśmy przechodzić po nim?
- Przechodźmy poziom po poziomie - przeszukiwanie wszerz (BFS)
- Przechodźmy najpierw w dół dla każdego węzła - przeszukiwanie wgłąb (DFS)



THE CODE

Przykład - wypisz wszystkie pliki w folderze

1. Zaimportujmy `listdir` i `path` z pakietu `os`
2. `listdir(dir)` zwraca listę nazw wszystkich plików i folderów w `dir`
3. `path.isdir(dir)` sprawdza, czy `dir` jest adresem folderu
4. `path.isfile(dir)` sprawdza, czy `dir` jest adresem pliku
5. Korzystając z tych trzech funkcji, wypisz wszystkie pliki w folderze



Dijkstra(G, w, s):
dla każdego wierzchołka v w $V[G]$ wykonaj
 $d[v] := \infty$
 poprzednik $[v] := \text{niezdefiniowane}$
 $d[s] := 0$
 $Q := V$
dopóki Q niepuste wykonaj
 $u := \text{Zdejmij_Min}(Q)$
 dla każdego wierzchołka v - sąsiada u wykonaj
 jeżeli $d[v] > d[u] + w(u, v)$ to
 $d[v] := d[u] + w(u, v)$
 poprzednik $[v] := u$

Wyświetl("Droga wynosi: " + $d[v]$)

Przykład -Algorytm Dijkstry

Minimalne Drzewo Rozpinające

- Mamy zbiór punktów z połączeniami
- Każde połączenie ma pewien koszt
- Jak znaleźć najtańszą sieć, która połączy wszystkie punkty?
- Typowe rozwiązania to algorytmy Prima, Kruskala i Borůvki

Jak się czujecie z całością tego kursu?

