Praca domowa 01 – plate

Michał Szczygieł

Wstęp

Zadanie składa się z 2 plików:

- Main.java
- Plate.java
-

Klasa Main

Klasa ta odpowiedzialna jest za właściwe wczytanie pliku, którego nazwa zostaje podana jako argument programu. Na podstawie danych wczytanych z pliku program przygotowywuje on dwie macierze, których rozmiary są ustalone poprzez dwie pierwsze wartości (n i m) i uzupełnione kosztami cięć w odpowiedniej zależności ($x_{1,}x_{2,}\ldots,x_{m-1}$, $y_{1,}y_{2,}\ldots,y_{n-1}$). Macierze te potem zostaję przekazane do konstruktora klasy Plate.

Pola:

`private ArrayList<Integer> arrayX` – Kolekcja przechowywująca dane z wagami dla cięć pionowych

`private ArrayList<Integer> arrayY` – Kolekcja przechowywująca dane z wagami dla cięć poziomych

Metody:

`private boolean readFile(File fileName) throws FileNotFoundException` – Metoda otwiera plik i zczytuje wszystkie poprawne dane do kolekcji znaków (nieujemne liczby całkowite), gdzie przekazuje je do metody `fillData()` w celu wypełnienia kolekcji `arrayX` i `arrayY.`

`private boolean fillData(ArrayList<String> data)` – Metoda wypełniająca `arrayX` i `arrayY.`

`public ArrayList<Integer> getArrayX()` – Getter dla kolekcji przechowywującej dane z wagami dla cięć pionowych.
`public ArrayList<Integer> getArrayY()` – Getter dla kolekcji przechowywującej dane z wagami dla cięć poziomych.

`public void setArrayX` – Setter dla kolekcji przechowywującej dane z wagami dla cięć pionowych.

`public void setArrayY` – Setter dla kolekcji przechowywującej dane z wagami dla cięć poziomych.

Klasa Plate

Klasa Plate oblicza koszt cięcia niejednorodnej tafli z danych otrzymanych od klasy Main. Posiada ona konstruktor, który ustawia kolekcje dla cięć pionowych i poziomych.

Pola:

`private ArrayList<Integer> arrayX` – Kolekcja przechowywująca dane z wagami dla cięć pionowych

`private ArrayList<Integer> arrayY` – Kolekcja przechowywująca dane z wagami dla cięć poziomych

Metody:

`public Plate(ArrayList<Integer> arrayX, ArrayList<Integer> arrayY)` – Konstruktor ustawiający kolekcje dla cięć pionowych i poziomych.

`public String cutCost()` – Metoda obliczająca koszt cięcia niejednorodnej tafli i zwracająca wynik w postaci Stringa „Koszt cięcia : x”, gdzie x jest kosztem cięcia tafli.

`public ArrayList<Integer> getArrayX()` – Getter dla kolekcji przechowywującej dane z wagami dla cięć pionowych.

`public ArrayList<Integer> getArrayY()` – Getter dla kolekcji przechowywującej dane z wagami dla cięć poziomych.

`public void setArrayX` – Setter dla kolekcji przechowywującej dane z wagami dla cięć pionowych.

`public void setArrayY` – Setter dla kolekcji przechowywującej dane z wagami dla cięć poziomych.

Realizacja algorytmu cięcia tafli

Sposób rozwiązania problemu cięcia niejednorodnej tafli, jak najmniejszym kosztem jest realizowany w następujący sposób. Otrzymaną kolekcję cięć pionowych i poziomych, powinna być posortowana rosnąco, w celu poukładnia kosztów cięć, gdyż rozpoczynając cięcia od najbardziej kosztownych do najmniej kosztownych otrzymujemy najbardziej optymalny wynik. Programowa realizacja problemu polega na zliczaniu ilości powstałych elementów na wskutek kolejnych cięć. Największa waga * ilość elementów do przecięcia jest sumowana do akumulatora. Element o największej wadze zostaje usunięty z kolekcji i zostaje powtórzony ten krok, aż do zdjęcia wszystkich elementów. Wynik koncowy zostaje wydrukowywany na ekranie. Złożoność obliczeniowa tego algorytmu jest liniowa.

Dyskusja

Należało by sobie postawić pytanie, czy powyższe rozwiązanie zawsze jest najelpsze. Co w przypadku tafli wielkości rzędu kilku set tysięcy na kilka set tysięcy. Czy mogą powstać jakieś pewne anomalie dla powyższego problemu. Więc nasuwa się rozwiązanie aby sprawdzić wszystkie możliwe kombinacje cięć i wybrać najtańszą spośród wszystkich. Rozwiązanie na pewno nie jest optymalne programowo, gdyż złożność tego algorytmu wynosiła by n!. Biąc nawet mała liczbę możliwych cięć jak 3x4, to możliwa ilość wszystkich kombinacji wynosi 5040, a nie wpominając już o 100000x100000 ...