Tablice jednowymiarowe – zadania maturalne

Źródło: Roman Synkevych, domena publiczna.

W e-materiale Tablice jednowymiarowe poznaliśmy podstawowe informacje dotyczące tablic jednowymiarowych.

Teraz skupimy się na rozwiązywaniu zadań, w których wykorzystane zostaną tablice jednowymiarowe.

Informacje o zastosowaniu tablic jednowymiarowych w wybranych językach programowania znajdziesz w e-materiałach:

- Tablice jednowymiarowe w języku C++,
- Tablice jednowymiarowe w języku Java,
- Tablice jednowymiarowe w języku Python.

Twoje cele

- Przeanalizujesz sposób rozwiązywania zadań wykorzystujących tablice jednowymiarowe.
- Rozwiążesz samodzielnie kilka zadań.
- Wykorzystasz w praktyce swoją wiedzę dotyczącą tablic jednowymiarowych.

Przeczytaj

Zadanie 1. Najdłuższy spójny podciąg niemalejący

Karol jest naukowcem w Republice Kernelowej. W ostatnim czasie przeprowadzał badania na temat stężenia pyłów przemysłowych w powietrzu. W tym celu w stolicy kraju zbudował stację pomiarową. Po kilku miesiącach pomiarów zestawił swoje wyniki w pliku `pomiary.txt`.

Plik `pomiary.txt` składa się ze 100 wierszy. W każdym z nich zapisano jedną liczbę naturalną.

Plik o rozmiarze 467.00 B w języku polskim

Pewnego dnia Karol został poproszony przez urzędników o podanie najdłuższego ciągu stężeń pyłów z dni, w których zanieczyszczenie bez przerwy utrzymywało się na tym samym poziomie lub rosło. Jeśli był więcej niż jeden taki okres, wszystkie powinny być wypisane w kolejnych liniach.

> **Przykład 1**
>
> Przykładowe dane:
>
> ```
> 5
> 27
> 23
> 20
> 4
> 26
> 24
> 12
> 9
> 1
> ```
>
> Wynik:
>
> ```
> 5 27
> 4 26
> ```

Napisz program, który znajdzie najdłuższy spójny podciąg niemalejący dla danych z pliku `pomiary.txt`, a wynik zapisze do pliku `powietrze.txt`.

Do oceny oddajesz:

- plik `powietrze.txt` z odpowiedzią (liczby wchodzące w skład najdłuższego spójnego niemalejącego podciągu liczb całkowitych dodatnich z pliku `pomiary.txt`; każdy podciąg w osobnej linii; linie powinny być oddzielone pojedynczym znakiem odstępu)
- plik(i) z komputerową realizacją zadania

> **Praca domowa**
>
> Przedstaw rozwiązanie zadania, pisząc program w języku C++, Java lub Python.

Rozwiązanie

> Rozwiązanie zadania przedstawimy w postaci pseudokodu, ponieważ na egzaminie maturalnym można korzystać z wybranego języka programowania: C++, Java lub Python.

Rozwiązanie podzielimy na dwa etapy. W pierwszej część określamy długość najdłuższego niemalejącego podciągu oraz indeksy tablicy, między którymi się on znajduje. Następnie, znając położenie podciągu, możemy przekopiować jego elementy do tablicy wynikowej. Na początku wczytajmy dane z zadania. Tablice indeksujemy od zera, zgodnie z zasadami indeksowania przyjętymi w językach programowania dostępnych na maturze:

```
1 pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"
```

W celu znalezienia położenia spójnego, niemalejącego podciągu definiujemy trzy dodatkowe zmienne: `maxDlugosc`, `obecnaDlugosc` oraz pustą tablicę `poczatkiPodciagow`, która będzie przechowywać indeksy początkowe wszystkich najdłuższych podciągów niemalejących.

```
1 pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"
2
3 maxDlugosc ← 1
4 obecnaDlugosc ← 1
5 poczatkiPodciagow[] // tablica liczb naturalnych
```

Następnie przechodzimy po wszystkich elementach tablicy `pomiary` oprócz pierwszego. Jeśli element o wartości indeksu o jeden mniejszej od wartości indeksu analizowanego elementu jest mniejszy lub równy, inkrementujemy zmienną `obecnaDlugosc`.

```
1 pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"
2
3 maxDlugosc ← 1
```

```
obecnaDlugosc ← 1
poczatkiPodciagow[] // tablica liczb naturalnych

dla i = 1, 2, ..., 99 wykonuj:
    jeżeli pomiary[i] >= pomiary[i - 1]:
        obecnaDlugosc ← obecnaDlugosc + 1
```

Jeżeli element o wartości indeksu o jeden mniejszej od wartości indeksu analizowanego elementu jest większy, oznacza to koniec niemalejącego spójnego podciągu i rozpoczęcie zliczania na nowo — ustawiamy wartość zmiennej `obecnaDlugosc` na 1.

```
pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"

maxDlugosc ← 1
obecnaDlugosc ← 1
poczatkiPodciagow[] // tablica liczb naturalnych

dla i = 1, 2, ..., 99 wykonuj:
    jeżeli pomiary[i] >= pomiary[i - 1]:
        obecnaDlugosc ← obecnaDlugosc + 1
    w przeciwnym wypadku:
        obecnaDlugosc ← 1
```

Jeśli wartość zmiennej `obecnaDlugosc` jest większa od `maxDlugosc`, aktualizujemy wartość `maxDlugosc` oraz czyścimy zawartość tablicy `poczatkiPodciagow` poprzez przypisanie jej pustej tablicy, a następnie dodajemy do tablicy `poczatkiPodciagow` wartość `i - maxDlugosc + 1` jako początek najdłuższego niemalejącego podciągu (jesteśmy w stanie wyliczyć początek podciągu, znając obecny indeks tablicy oraz wartość `maxDlugosc`).

```
pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"

maxDlugosc ← 1
obecnaDlugosc ← 1
poczatkiPodciagow[] // tablica liczb naturalnych

dla i = 1, 2, ..., 99 wykonuj:
    jeżeli pomiary[i] >= pomiary[i - 1]:
        obecnaDlugosc ← obecnaDlugosc + 1
    w przeciwnym wypadku:
```

```
        obecnaDlugosc ← 1
    jeżeli obecnaDlugosc > maxDlugosc:
        maxDlugosc ← obecnaDlugosc
        poczatkiPodciagow ← []
        dodaj do tablicy poczatkiPodciagow wartość i - maxDlugosc
```

Jeśli długość obecnego podciągu niemalejącego zapisana w zmiennej `obecnaDlugosc` jest równa długości dotychczasowego najdłuższego podciągu zapisanego w zmiennej `maxDlugosc`, dodajemy wartość `i - maxDlugosc + 1` do tablicy `poczatkiPodciagow`.

```
pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"

maxDlugosc ← 1
obecnaDlugosc ← 1
poczatkiPodciagow[] // tablica liczb naturalnych

dla i = 1, 2, ..., 99 wykonuj:
    jeżeli pomiary[i] >= pomiary[i - 1]:
        obecnaDlugosc ← obecnaDlugosc + 1
    w przeciwnym wypadku:
        obecnaDlugosc ← 1
    jeżeli obecnaDlugosc > maxDlugosc:
        maxDlugosc ← obecnaDlugosc
        poczatkiPodciagow ← []
        dodaj do tablicy poczatkiPodciagow wartość i - maxDlugosc
    jeżeli obecnaDlugosc = maxDlugosc:
        dodaj do tablicy poczatkiPodciagow wartość i - maxDlugosc

```

W kolejnym kroku w pętli `dla` pobieramy początki podciągów z tablicy `poczatkiPodciagow` i wypisujemy elementy tablicy `pomiary` od indeksu `poczatekPodciagu` do `poczatekPodciagu + maxDlugosc - 1`.

Następnie zapisujemy wartość elementu tablicy `pomiary[j]` do pliku `powietrze.txt`. W dalszej kolejności dodajemy nową linię do pliku, aby oddzielić od siebie podciągi.

```
pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"

maxDlugosc ← 1
obecnaDlugosc ← 1
```

```
poczatkiPodciagow[] // tablica liczb naturalnych

dla i = 1, 2, ..., 99 wykonuj:
    jeżeli pomiary[i] >= pomiary[i - 1]:
        obecnaDlugosc ← obecnaDlugosc + 1
    w przeciwnym wypadku:
        obecnaDlugosc ← 1
    jeżeli obecnaDlugosc > maxDlugosc:
        maxDlugosc ← obecnaDlugosc
        poczatkiPodciagow ← []
        dodaj do tablicy poczatkiPodciagow wartość i - maxDlugosc
    jeżeli obecnaDlugosc = maxDlugosc:
        dodaj do tablicy poczatkiPodciagow wartość i - maxDlugosc

dla poczatekPodciagu w poczatkiPodciagow wykonuj:
    dla j = poczatekPodciagu, poczatekPodciagu + 1, ..., poczatek
        zapisz pomiary[j] do pliku "powietrze.txt"
    zapisz nową linię do pliku "powietrze.txt"
```

Zamykamy pliki.

```
pomiary[0..99] ← wczytaj dane z pliku "pomiary.txt"

maxDlugosc ← 1
obecnaDlugosc ← 1
poczatkiPodciagow[] // tablica liczb naturalnych

dla i = 1, 2, ..., 99 wykonuj:
    jeżeli pomiary[i] >= pomiary[i - 1]:
        obecnaDlugosc ← obecnaDlugosc + 1
    w przeciwnym wypadku:
        obecnaDlugosc ← 1
    jeżeli obecnaDlugosc > maxDlugosc:
        maxDlugosc ← obecnaDlugosc
        poczatkiPodciagow ← []
        dodaj do tablicy poczatkiPodciagow wartość i - maxDlugosc
    jeżeli obecnaDlugosc = maxDlugosc:
        dodaj do tablicy poczatkiPodciagow wartość i - maxDlugosc

dla poczatekPodciagu w poczatkiPodciagow wykonuj:
    dla j = poczatekPodciagu, poczatekPodciagu + 1, ..., poczatek
```

```
        zapisz pomiary[j] do pliku "powietrze.txt"
    zapisz nową linię do pliku "powietrze.txt"

zamknij plik "pomiary.txt"
zamknij plik "powietrze.txt"
```

Odpowiedź do zadania

`powietrze.txt`

Plik o rozmiarze 56.00 B w języku polskim

Słownik

> **najdłuższy spójny podciąg niemalejący**
>
> fragment ciągu składający się z elementów ułożonych bezpośrednio po sobie, w kolejności niemalejącej, czyli w ten sposób, że każdy kolejny element jest równy poprzedniemu bądź większy od niego

Prezentacja multimedialna

Zadanie 2. Dostawa do magazynu

Adam i Kasia pracują w magazynie przeładunkowym w Faktorlandzie, odbierając dziennie 10 dostaw z całego świata o różnej liczbie ładunków. Zadaniem Adama jest rozładowywanie oraz zliczanie towarów, natomiast Kasia uzupełnia dokumentację. Na koniec każdego dnia Kasia zapisuje podsumowanie pracy w postaci 10 liczb naturalnych oddzielonych znakiem spacji, gdzie każdy element jest liczbą ładunków konkretnego towaru.

W kulturze mieszkańców Faktorlandu wyjątkowe znaczenie mają liczby pierwsze.
W związku z tym dyrektor magazynu poprosił o szczególny rodzaj informacji. Chciał wiedzieć, ile było takich dni, w których nie tylko liczba dostaw była liczbą pierwszą, ale również liczba ładunków w tych dostawach była liczbą pierwszą.

Dane zawierające podsumowania dostaw znajdują się w pliku `towary.txt`.

Plik `towary.txt` składa się z 50 wierszy. W każdym z nich znajduje się 10 liczb naturalnych oddzielonych od siebie znakiem spacji.

Plik o rozmiarze 1.43 KB w języku polskim

> **Polecenie 1**
>
> Napisz program, który dla danych z pliku `towary.txt` obliczy, ile jest takich wierszy, w których liczba elementów będących liczbami pierwszymi również jest liczbą pierwszą. Wynik zapisz do pliku `pierwsze.txt`.

Do oceny oddajesz:

- plik `pierwsze.txt` z odpowiedzią (liczba naturalna będąca liczbą wierszy z pliku `towary.txt`, w których liczba elementów będących liczbami pierwszymi również jest liczbą pierwszą)
- plik(i) z komputerową realizacją zadania

> **Praca domowa**
>
> Przedstaw rozwiązanie zadania, pisząc program w języku C++, Java lub Python. Zadbaj o prawidłowe wczytanie danych z pliku tekstowego. Odpowiedź do zadania znajduje się pod prezentacją multimedialną.

Rozwiązanie

Rozwiązanie zadania przedstawimy w postaci pseudokodu, ponieważ na egzaminie maturalnym można korzystać z wybranego języka programowania: C++, Java lub Python.
Polecenie 2

Zapoznaj się z prezentacją przedstawiającą przykładowe rozwiązanie zadania.

1

Eratostenes z Cyreny (nowożytna rycina). Do wyszukania liczb pierwszych w zadanym przedziale wykorzystujemy tzw. sito Eratostenesa
Źródło: Wikimedia Commons, commons.wikimedia.org, dostęp 18.11.2022, domena publiczna.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Rozpoczynamy od zdefiniowania funkcji, która pozwoli określić, czy dana liczba jest liczbą pierwszą. Na początek sprawdzamy, czy liczba jest mniejsza od 2. Jeżeli tak, zwracamy fałsz.

```
1 funkcja
  czyPierwsza(liczba)
2     jeżeli liczba < 2
  zwróć fałsz
```

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Następnie sprawdzamy, czy argument nie jest równy 2 lub 3 – w tym przypadku zwracamy prawdę. Wszystkie kolejne liczby będące ich wielokrotnościami (a więc podzielne przez 2 lub 3) z definicji nie mogą być liczbami pierwszymi, zatem zwracamy dla nich fałsz. Operator `mod` oznacza resztę z dzielenia:

```
funkcja czyPierwsza(liczba)
    jeżeli liczba < 2 zwróć fałsz

    jeżeli liczba = 2 lub liczba = 3 zwróć prawdę

    jeżeli liczba mod 2 = 0 lub liczba mod 3 = 0 zwróć fałsz
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Poszukiwania ewentualnych podzielników możemy zawęzić od liczby 5 do pierwiastka kwadratowego z argumentu funkcji (jeśli jest on liczbą całkowitą, to jest też największym możliwym podzielnikiem argumentu funkcji). Używamy funkcji `pierwiastekKwadratowy()`, którą można zaimplementować w dowolnym języku programowania dostępnym na egzaminie maturalnym:

```
funkcja czyPierwsza(liczba)
```

```
    jeżeli liczba < 2
zwróć fałsz

    jeżeli liczba = 2 lub
liczba = 3 zwróć prawdę

    jeżeli liczba mod 2 =
0 lub liczba mod 3 = 0
zwróć fałsz

    podzielnik ← 5
    dopóki podzielnik <=
pierwiastekKwadratowy(licz
ba) wykonuj:
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Zauważmy, że liczby pierwsze większe od 3 różnią się między sobą na zmianę o 2 lub o 4 $(5 \rightarrow 7 \rightarrow 11 \rightarrow 13 \rightarrow 17 \rightarrow 19 \rightarrow 23)$. Wynika to z faktu, że wszystkie liczby pomiędzy nimi są wielokrotnościami 2 lub 3. Oczywiście nie wszystkie liczby z podanego ciągu będą liczbami pierwszymi (ponieważ od pewnego momentu znajdą się wśród nich wielokrotności liczb 5, 7 itd.), jednakże wśród liczb z tego ciągu znajdują się na pewno wszystkie liczby pierwsze.

Na podstawie tego możemy zwiększać zmienną `podzielnik` w każdej iteracji pętli o 6 i sprawdzać, czy jej wartość lub jej wartość powiększona o 2 jest podzielnikiem argumentu funkcji. Jeśli warunek ten jest spełniony, zwracamy fałsz:

```
funkcja
czyPierwsza(liczba)
```

```
    jeżeli liczba < 2
zwróć fałsz

    jeżeli liczba = 2 lub
liczba = 3 zwróć prawdę

    jeżeli liczba mod 2 =
0 lub liczba mod 3 = 0
zwróć fałsz

    podzielnik ← 5
    dopóki podzielnik <=
pierwiastekKwadratowy(licz
ba) wykonuj:
        jeżeli liczba mod
podzielnik = 0 lub liczba
mod (podzielnik + 2) = 0
zwróć fałsz
        podzielnik ←
podzielnik + 6
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

5

Jeśli działanie pętli nie zostanie przerwane, oznacza to, że liczba nie ma żadnych podzielników (oprócz 1 i siebie samej). Jest zatem liczbą pierwszą i możemy zwrócić prawdę:

```
funkcja
czyPierwsza(liczba)
    jeżeli liczba < 2
zwróć fałsz

    jeżeli liczba = 2 lub
liczba = 3 zwróć prawdę

    jeżeli liczba mod 2 =
0 lub liczba mod 3 = 0
```

```
    zwróć fałsz
7
8       podzielnik ← 5
9       dopóki podzielnik <=
    pierwiastekKwadratowy(licz
    ba) wykonuj:
10          jeżeli liczba mod
    podzielnik = 0 lub liczba
    mod (podzielnik + 2) = 0
    zwróć fałsz
11          podzielnik ←
    podzielnik + 6
12      zwróć prawdę
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Przystępujemy do właściwej części zadania. Dla każdego wiersza z pliku sprawdzamy, ile jego elementów jest liczbami pierwszymi. Przed wykonaniem sprawdzenia zerujemy zmienną `licznikDostaw`, która będzie przechowywać tę informację oraz wczytujemy dane z pliku. Pamiętamy o tym, że tablice indeksujemy, rozpoczynając od zera.

```
1  licznikTablic ← 0
2
3  dla i = 0, 1, ..., 49
   wykonuj:
4      licznikDostaw ← 0
5      towary[0..9] ← wczytaj
   linię z pliku "towary.txt"
6      dla j = 0, 1, ..., 9
   wykonuj:
```

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Jeżeli element tablicy jest liczbą pierwszą, inkrementujemy zmienną `licznikDostaw`. Po przejściu całej tablicy sprawdzamy, czy zmienna `licznikDostaw` również jest liczbą pierwszą – jeśli tak, zwiększamy zmienną `licznikTablic`:

```
licznikTablic ← 0

dla i = 0, 1, ..., 49
wykonuj:
    licznikDostaw ← 0
    towary[0..9] ← wczytaj linię z pliku "towary.txt"
    dla j = 0, 1, ..., 9
wykonuj:
        jeżeli czyPierwsza(towary[j]) = prawda:
            licznikDostaw ← licznikDostaw + 1
    jeżeli czyPierwsza(licznikDostaw) = prawda:
        licznikTablic ← licznikTablic + 1
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Po zakończeniu zewnętrznej pętli zapisujemy wartość zmiennej `licznikTablic` do wynikowego pliku:

```
licznikTablic ← 0

```

```
3  dla i = 0, 1, ..., 49
   wykonuj:
4      licznikDostaw ← 0
5      towary[0..9] ← wczytaj
   linię z pliku "towary.txt"
6      dla j = 0, 1, ..., 9
   wykonuj:
7          jeżeli
   czyPierwsza(towary[j]) =
   prawda:
8              licznikDostaw
   ← licznikDostaw + 1
9      jeżeli
   czyPierwsza(licznikDostaw)
   = prawda:
10         licznikTablic ←
   licznikTablic + 1
11
12 zapisz licznikTablic do
   pliku "pierwsze.txt"
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

9

Zamykamy pliki.

```
1  licznikTablic ← 0
2
3  dla i = 0, 1, ..., 49
   wykonuj:
4      licznikDostaw ← 0
5      towary[0..9] ← wczytaj
   linię z pliku "towary.txt"
6      dla j = 0, 1, ..., 9
   wykonuj:
7          jeżeli
   czyPierwsza(towary[j]) =
   prawda:
8              licznikDostaw
   ← licznikDostaw + 1
```

```
    jeżeli
czyPierwsza(licznikDostaw)
= prawda:
        licznikTablic ←
licznikTablic + 1

zapisz licznikTablic do
pliku "pierwsze.txt"

zamknij plik "towary.txt"
zamknij plik
"pierwsze.txt"
```

10

Rozwiązując zadania maturalne, nie zapomnij o właściwym opisaniu plików wynikowych.
Źródło: Pixabay, pixabay.com, dostęp 18.11.2022, CC 0.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PAiVsY5nn

Cały kod prezentuje się następująco:

```
funkcja
czyPierwsza(liczba)
    jeżeli liczba < 2
zwróć fałsz

    jeżeli liczba = 2 lub
liczba = 3 zwróć prawdę

```

```
    jeżeli liczba mod 2 = 0 lub liczba mod 3 = 0 zwróć fałsz

    podzielnik ← 5
    dopóki podzielnik <= pierwiastekKwadratowy(liczba) wykonuj:
        jeżeli liczba mod podzielnik = 0 lub liczba mod (podzielnik + 2) = 0 zwróć fałsz
        podzielnik ← podzielnik + 6
    zwróć prawdę

licznikTablic ← 0

dla i = 0, 1, ..., 49 wykonuj:
    licznikDostaw ← 0
    towary[0..9] ← wczytaj linię z pliku "towary.txt"
    dla j = 0, 1, ..., 9 wykonuj:
        jeżeli czyPierwsza(towary[j]) = prawda:
            licznikDostaw ← licznikDostaw + 1
    jeżeli czyPierwsza(licznikDostaw) = prawda:
        licznikTablic ← licznikTablic + 1

zapisz licznikTablic do pliku "pierwsze.txt"

zamknij plik "towary.txt"
zamknij plik "pierwsze.txt"
```

Odpowiedź do zadania:

`pierwsze.txt`

Plik o rozmiarze 2.00 B w języku polskim

> **Polecenie 3**
>
> Dodaj do swojego programu komentarze tak, aby program był zrozumiały nawet dla osoby, która nie potrafi programować.
>
> ```
> 
> ```

Sprawdź się

Zadanie 3. Giełda bajtocka

Ania jest analitykiem giełdowym. W ostatnim miesiącu zbierała dane dotyczące wzrostów i spadków wartości wskaźnika na lokalnej giełdzie w Bajtocji. W celach statystycznych zamierza sprawdzić, jaki najlepszy możliwy wynik giełda osiągnęła w ciągu następujących po sobie dni. By to zrobić, musi znaleźć w całym badanym okresie ciąg następujących po sobie dni, w których suma wyników była największa. Ania bierze pod uwagę to, że wśród dni może być taki, w którym wartość wskaźnika była ujemna oraz to, że na giełdzie może mieć miejsce bessa, czyli wszystkie wskaźniki będą ujemne. W takiej sytuacji program powinien wypisać jednoelementowy podciąg składający się z największej liczby ujemnej.

Dane zapisane są w pliku `giełda.txt`. Plik składa się ze 100 wierszy. Każdy wiersz zawiera liczbę rzeczywistą oznaczającą wzrost lub spadek wartości wskaźnika giełdy w danym dniu miesiąca:

Plik o rozmiarze 602.00 B w języku polskim

Napisz program, który dla danych z pliku `giełda.txt` znajdzie spójny podciąg o maksymalnej sumie, a jego sumę zaokrągloną do dwóch miejsc po przecinku zapisze w pliku `wyniki.txt`.

> **Ważne!**
>
> Poprzez **spójny podciąg o maksymalnej sumie** rozumiemy taki podciąg, który jest złożony z elementów kolejno następujących po sobie w ciągu bazowym, a którego zsumowane elementy dają największy możliwy wynik.

> **Przykład 1**
>
> Spójny podciąg o maksymalnej sumie wcale nie musi być podciągiem, który składa się ze wszystkich elementów danego ciągu.

> Dla przykładowej tablicy dane:
>
> ```
> dane[0..9] = {2.51, -1.3, 5.23, -7.1, -2.4,
>     3.1, -1, 2.5, 1.5, 0.4}
> ```
>
> Suma wszystkich elementów wynosi `3.44`. Jednak już na pierwszy rzut oka możemy stwierdzić, że **spójnym podciągiem o większej sumie** byłby podciąg jednoelementowy `5.23`.
>
> Tymczasem suma maksymalnego spójnego podciągu`{3.1, -1, 2.5, 1.5, 0.4}` wynosi `6.5`.

Do oceny oddajesz:

Dane:

- plik `wyniki.txt` z odpowiedzią (liczbą rzeczywistą będącą sumą maksymalnego spójnego podciągu liczb z pliku `giełda.txt` zaokrągloną do dwóch miejsc po przecinku)
- plik(i) z komputerową realizacją zadania (kodem programu)

> **Praca domowa**
>
> Przedstaw rozwiązanie zadania, pisząc program w języku C++, Java lub Python. Zadbaj o prawidłowe wczytanie danych z pliku tekstowego. Rozwiązanie zadania znajduje się pod sekcją ćwiczeń.

Pokaż ćwiczenia: 

> **Język C++**

> **Ćwiczenie 1** 

Java

Ćwiczenie 2

Python

Ćwiczenie 3

Odpowiedź do zadania:

`wyniki.txt`

Plik o rozmiarze 5.00 B w języku polskim

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Tablice jednowymiarowe – zadania maturalne

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) w zależności od problemu rozwiązuje go, stosując metodę wstępującą lub zstępującą;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) wykorzystuje znane sobie algorytmy przy rozwiązywaniu i programowaniu rozwiązań następujących problemów:

c) znajdowania w ciągu podciągów o różnorodnych własnościach, np. najdłuższego spójnego podciągu niemalejącego, spójnego podciągu o największej sumie,

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz sposób rozwiązywania zadań wykorzystujących tablice jednowymiarowe.
- Rozwiążesz samodzielnie kilka zadań.
- Wykorzystasz w praktyce swoją wiedzę dotyczącą tablic jednowymiarowych.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;

- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji);
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Tablice jednowymiarowe – zadania maturalne". Uczniowie zapoznają się z multimedium w sekcji „Prezentacja multimedialna".

Faza wstępna:

1. Nauczyciel wprowadza uczniów szczegółowo w temat lekcji i jej cele. Może posłużyć się wyświetloną na tablicy zawartością sekcji „Wprowadzenie".

Faza realizacyjna:

1. Uczniowie analizują przykład z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązanie na swoim komputerze.
2. **Praca z multimedium.** Uczniowie w zespołach dwuosobowych zapoznają się z treścią polecenia nr 1: „Przeanalizuj podany sposób rozwiązania poniższego zadania" z sekcji „Prezentacja multimedialna" i wspólnie analizują kolejne kroki rozwiązania postawionego problemu.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1-6 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Sprawdź się".

Praca domowa:

1. Uczniowie implementują algorytm z polecenia 1 z sekcji „Prezentacja multimedialna" w wybranym przez siebie języku programowania.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla języka C++.

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Multimedium w sekcji „Prezentacja multimedialna" można potraktować jako zadanie domowe dotyczące analizy problemu zawartego w temacie „Tablice jednowymiarowe – zadania maturalne".