Algorytm Huffmana w języku Java

Źródło: Michael Dziedzic, domena publiczna.

Poznaliśmy już podstawowe informacje dotyczące algorytmu Huffmana. W tym e-materiale zaimplementujemy go w języku Java.

Jeśli od razu chcesz zapoznać się z przykładowym rozwiązaniem konkretnego problemu i sprawdzić swoją wiedzę, przejdź do e-materiału Algorytm Huffmana – zadania maturalne.

Ciekawi cię, jak wygląda implementacja algorytmu Huffmana w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Algorytm Huffmana w języku C++,
- Algorytm Huffmana w języku Python.

Twoje cele

- Prześledzisz implementację algorytmu Huffmana w języku Java.
- Zaimplementujesz program w języku Java kodujący ciąg znaków za pomocą algorytmu Huffmana.
- Wykonasz ćwiczenia, wykorzystując w praktyce wiedzę dotyczącą omawianego zagadnienia.
- Wyjaśnisz, jak konstruować i przeszukiwać drzewa binarne.

Przeczytaj

Już wiesz

W roku 1952 David Huffman opracował prefiksową metodę kompresji bezstratnej.

Mówimy, że kodowanie jest prefiksowe, gdy żadne słowo kodowe nie jest początkiem innego słowa.

Słowem kodowym w kontekście kodowania Huffmana określamy ciągi bitów, które jednoznacznie interpretują jakiś znak. Uwaga – dla uproszczenia w implementacji będziemy kodować każdy znak osobno, bez wcześniejszego grupowania.
Problem 1

Załóżmy, że w alfabecie, którym się posługujemy, są trzy znaki zgrupowane w następującej książce kodowej:

A: 1
B: 01
C: 00

Mamy zakodować wiadomość: „ABBAAAC”.

Spróbuj samodzielnie zakodować wiadomość.

> **Problem 2**
>
> Załóżmy, że wiadomość „ABBAAAC” zakodujemy następującym kodem:
>
> A: 1
> B: 0
> C: 00
>
> Wiadomość ta przyjęłaby następującą postać: 10011100 – miałaby tym samym jedynie 8 bitów, czyli zaoszczędzilibyśmy więcej pamięci.
> Problem jednak polega na tym, że o ile zakodowana wiadomość jest krótsza, to nie da się jej jednoznacznie odkodować.
>
> Spróbuj zdekodować wiadomość na jak najwięcej różnych sposobów, aby przekonać się, że kodowanie z tego przykładu nie jest jednoznaczne.

> **Ważne!**
>
> Książka kodowa przedstawiona w Problemie 2 jest błędna, ponieważ słowo kodowe odpowiadające literze B jest prefiksem słowa kodowego odpowiadającego literze C i z tego powodu nie można jednoznacznie odkodować wiadomości po tym, gdy zostanie ona zakodowana.

Implementacja w języku Java

> **Problem 3**
>
> Zaimplementujmy w języku Java program kodujący podany ciąg znaków za pomocą algorytmu Huffmana. Następnie przetestujemy go dla ciągu znaków `"AABBCCDDDDDDDFFFEEER"`

Specyfikacja problemu:

Dane:

- `wiadomosc` – łańcuch znaków do zakodowania

Wynik:

- `zakodowanaWiadomosc` – łańcuch znaków; wiadomość zakodowana algorytmem Huffmana

Zaczynamy od zdefiniowania klasy `wierzcholek` zawierającej konstruktor, który później pomoże szybciej tworzyć nowe wierzchołki. Klasa będzie posiadała pola `znak`, `czestosc`, `lewy`, `prawy` przechowujące kolejno znak, jego częstość, lewe, a następnie prawe dziecko wierzchołka.

```
class Wierzcholek {
    char znak;
    int czestosc;
    Wierzcholek lewy = null;
    Wierzcholek prawy = null;
    Wierzcholek(char znak, int czestosc, Wierzcholek lewy, Wierzc
        this.znak = znak;
        this.czestosc = czestosc;
        this.lewy = lewy;
        this.prawy = prawy;
    }
}
```

Ponieważ w języku Java każdy symbol zapisywany jest w typie `char`, który ma długość 2 bajtów (wartości możliwe do zapisania mieszczą się w zakresie od 0000_{16} do $FFFF_{16}$), maksymalna liczba unikalnych symboli, jakie mogą pojawić się w tekście, wynosi 65536_{10}, co powoduje potencjalny problem z ich zliczeniem.

Można zrobić to na kilka sposobów. Moglibyśmy utworzyć tablicę ze zmiennymi typu `int` o długości 65536 i zapisać każde wystąpienie znaku na odpowiadającym mu indeksie. Metoda ta sprawiłaby jednak, że w trakcie kodowania wykorzystalibyśmy dodatkowe 262144 bajty.

Jesteśmy w stanie temu zaradzić, posługując się rozwiązaniem znanym z algorytmów sortujących. Przeszukując najpierw tekst i znajdując znaki o najmniejszej i największej wartości liczbowej, możemy zmniejszyć rozmiar tablicy, analogicznie jak w sortowaniu przez zliczanie czy w sortowaniu kubełkowym. Dokładne omówienie tych zagadnień znajdziesz w e-materiałach:

- Sortowanie przez zliczanie,
- Sortowanie kubełkowe.

Następnie zliczamy wystąpienia poszczególnych znaków i zapisujemy je do tablicy.

```
public static int[] zlicz(String wiadomosc) {
    int[] tabZliczajaca;
```

```
    char min = Character.MAX_VALUE;
    char max = Character.MIN_VALUE;
    for (int i = 0; i < wiadomosc.length(); i++) {
        if (min > wiadomosc.charAt(i)) min = wiadomosc.charAt(i);
        if (max < wiadomosc.charAt(i)) max = wiadomosc.charAt(i);
    }
    tabZliczajaca = new int[max - min + 1];
    for (int i = 0; i < wiadomosc.length(); i++) {
        tabZliczajaca[wiadomosc.charAt(i) - min]++;
    }
    return tabZliczajaca;
}
```

Stałe `Character.MAX_VALUE` i `Character.MIN_VALUE` dostępne są w klasie `Character`, której nie musimy importować, ponieważ jest dostępna w ramach pakietu `Java.lang`. Po zliczeniu częstości występujących w kodzie znaków przechodzimy do tworzenia wierzchołków.

Zaprezentowany fragment kodu implementuje utworzenie wierzchołków dla tych znaków, które występują w kodzie przynajmniej raz. Korzystając z tablicy zwróconej przez funkcję `zlicz()`, tworzymy tablicę zawierającą nowo utworzone wierzchołki. Aby móc stwierdzić, jaki znak jest zakodowany, pod którym indeksem tablicy, musimy najpierw znaleźć w wiadomości znak o najniższym kodowaniu, a następnie do każdego indeksu z tablicy dodać jego wartość. W ten sposób po przekonwertowaniu na typ `char` otrzymamy odpowiedni znak.

```
String wiadomosc = "AABBCCDDDDDDDFFFEEER";
int[] tabZliczajaca = zlicz(wiadomosc);
char minZnak = Character.MAX_VALUE;
for (int i = 0; i < wiadomosc.length(); i++) {
    if (minZnak > wiadomosc.charAt(i))
        minZnak = wiadomosc.charAt(i);
}
Wierzcholek[] listaWierzcholkow = new Wierzcholek[tabZliczajaca.l
int dlugoscListy = 0;
for (int i = 0; i < listaWierzcholkow.length; i++) {
    if (tabZliczajaca[i] != 0) {
        listaWierzcholkow[dlugoscListy] = new Wierzcholek((char)
        dlugoscListy++;
    }
}
```

Następnie, zgodnie z kodowaniem Huffmana, wyszukujemy dwa wierzchołki o najmniejszej liczbie wystąpień i łączymy je w jeden o liczbie wystąpień równej sumie liczb wystąpień złączanych wierzchołków. Robimy tak do momentu, gdy zostanie nam jeden wierzchołek stanowiący korzeń całego drzewa binarnego.

Posortujmy na początku wszystkie wierzchołki według liczby wystąpień. Skorzystamy z algorytmu sortowania bąbelkowego (jeśli chcesz przypomnieć sobie wiadomości na ten temat, znajdziesz je w e-materiale Sortowanie bąbelkowe).

Podczas sortowania zawsze bierzemy dwa ostatnie wierzchołki z listy i łączymy je. Następnie wystarczy, że nowo powstały wierzchołek przesuniemy na odpowiednie miejsce w liście, a cała procedura będzie się powtarzać, aż zostanie jeden wierzchołek (czyli korzeń).

```
public static Wierzcholek sortowanie(Wierzcholek[] listaWierzchol
    int j = dlugoscListy;
    do {
        for (int i = 0; i < j - 1; i++) {
            if (listaWierzcholkow[i].czestosc < listaWierzcholkow
                Wierzcholek temp = listaWierzcholkow[i];
                listaWierzcholkow[i] = listaWierzcholkow[i + 1];
                listaWierzcholkow[i + 1] = temp;
            }
        }
        j--;
    } while (j > 1);

    int k = dlugoscListy;
    while (k > 1) {
        Wierzcholek pierwszyDoUsuniecia = listaWierzcholkow[k - 1
        Wierzcholek drugiDoUsuniecia = listaWierzcholkow[k - 2];
        Wierzcholek nowyWierzcholek = new Wierzcholek('\0',
                    pierwszyDoUsuniecia.czestosc + drugiDoUsuniec
        System.out.println("Usuwam i lacze w jeden następujace wi
        System.out.println("Wierzcholek o znaku: " + pierwszyDoUs
        System.out.println("Wierzcholek o znaku: " + drugiDoUsuni
        listaWierzcholkow[k - 2] = nowyWierzcholek;
        listaWierzcholkow[k - 1] = null;
        k--;
        // umieść nowy wierzchołek w odpowiednim miejscu
        for (int i = k - 1; i > 0; i--) {
            if (listaWierzcholkow[i - 1] == null || listaWierzcho
```

```
                Wierzcholek temp = listaWierzcholkow[i];
                listaWierzcholkow[i] = listaWierzcholkow[i - 1];
                listaWierzcholkow[i - 1] = temp;
            } else
                break;
        }
    }
    return listaWierzcholkow[0];
}
```

Funkcja `sortowanie` zwróci jeden wierzchołek będący korzeniem.

```
Wierzcholek korzen = sortowanie(listaWierzcholkow, dlugoscListy);
```

Napiszemy też funkcję pomocniczą, dzięki której będzie można odczytać, w jaki sposób kodujemy znaki – służy ona jedynie do sprawdzania poprawności drzewa. W tym celu przejdziemy rekurencyjnie po drzewie i wypiszemy wartości każdego ze znalezionych liści.

```
public static void wypiszKod(Wierzcholek doPrzeszukania, String d
    if (doPrzeszukania.lewy == null && doPrzeszukania.prawy == nu
        System.out.println("Znak: " + doPrzeszukania.znak +
                           " czestosc: " + doPrzeszukania.czesto
                           " slowo kodowe: " + dotychczasowyKod)
    }
    if (doPrzeszukania.lewy != null) {
        wypiszKod(doPrzeszukania.lewy, dotychczasowyKod + "0");
    }
    if (doPrzeszukania.prawy != null) {
        wypiszKod(doPrzeszukania.prawy, dotychczasowyKod + "1");
    }
}
```

Tworzymy dodatkową klasę `Slowo`, która będzie przechowywała słowo kodowe oraz kodowany znak pod tym słowem.

```
class Slowo {
    char znak;
    int czestosc;
    String slowoKodowe;
```

```
5
6       Slowo(char znak, int czestosc, String slowoKodowe) {
7           this.znak = znak;
8           this.czestosc = czestosc;
9           this.slowoKodowe = slowoKodowe;
10      }
11  }
```

W kolejnym kroku implementujemy funkcję tworzącą tablicę słów kodowych. W tym celu przejdziemy rekurencyjnie po drzewie Huffmana i wartość każdego liścia zapiszemy w obiekcie klasy `Słowo`, który umieścimy w tablicy.

```
1  static int obecnyElementTablicySlow = 0;
2
3  private static Slowo[] zapiszKod(Wierzcholek doPrzeszukania, Stri
4      if (doPrzeszukania.lewy == null && doPrzeszukania.prawy == nu
5          listaSlow[obecnyElementTablicySlow] = new Slowo(doPrzeszu
6          obecnyElementTablicySlow++;
7      }
8      if (doPrzeszukania.lewy != null) {
9          zapiszKod(doPrzeszukania.lewy, dotychczasowyKod + "0", li
10     }
11     if (doPrzeszukania.prawy != null) {
12         zapiszKod(doPrzeszukania.prawy, dotychczasowyKod + "1", l
13     }
14     return listaSlow;
15 }
```

Implementujemy odnajdywanie odpowiedniego słowa kodowego dla odpowiedniego znaku. Iterując po książce słów, szukamy odpowiedniego znaku, a następnie go zwracamy.

```
1  static String zakodujZnak(char znak, Slowo[] ksiazkaSlow) {
2      for (int i = 0; i < ksiazkaSlow.length; i++) {
3          if (ksiazkaSlow[i].znak == znak)
4              return ksiazkaSlow[i].slowoKodowe;
5      }
6      return "";// jeśli korzystamy z odpowiedniej funkcji kodowej,
7  }
```

Gdy mamy już zaimplementowane kodowanie znaku, pora na wdrożenie kodowania całej wiadomości z użyciem gotowej książki kodowej. Dla każdego znaku wywołujemy metodę `zakodujZnak()`, a następnie wszystkie zakodowane znaki łączymy w jedną zakodowaną wiadomość.

```
static String zakodujWiadomosc(String wiadomosc, Slowo[] ksiazkaS
    String zakodowanaWiadomosc = "";
    for (int i = 0; i < wiadomosc.length(); i++) {
        zakodowanaWiadomosc += zakodujZnak(wiadomosc.charAt(i), k
    }
    return zakodowanaWiadomosc;
}
```

Funkcja `main` połączy poszczególne omówione powyżej fragmenty kodu. Poza tym dodajemy wywołanie funkcji tworzącej drzewo i kodującej przykładową wiadomość.

```
public static void main(String[] args) {
    String wiadomosc = "AABBCCDDDDDDDFFFEEER";
    int[] tabZliczajaca = zlicz(wiadomosc);
    char minZnak = Character.MAX_VALUE;
    for (int i = 0; i < wiadomosc.length(); i++) {
        if (minZnak > wiadomosc.charAt(i))
            minZnak = wiadomosc.charAt(i);
    }
    Wierzcholek[] listaWierzcholkow = new Wierzcholek[tabZliczaja
    int dlugoscListy = 0;
    for (int i = 0; i < listaWierzcholkow.length; i++) {
        if (tabZliczajaca[i] != 0) {
            listaWierzcholkow[dlugoscListy] = new Wierzcholek((ch
            dlugoscListy++;
        }
    }
    Wierzcholek korzen = sortowanie(listaWierzcholkow, dlugoscLis
    // w przypadku, gdy w tekście znajduje się tylko jeden unikal
    // obsłużyć ten wyjątek, ponieważ drzewo się nie zbuduje
    if (dlugoscListy == 1)
        korzen = new Wierzcholek('\0', 1, null, listaWierzcholkow
    Slowo[] ksiazkaSlow = new Slowo[dlugoscListy];
    wypiszKod(korzen, "");
    // zapisujemy kodowanie w książce słów
```

```
25      ksiazkaSlow = zapiszKod(korzen, "", ksiazkaSlow);
26      // kodujemy wiadomość
27      String wiadomoscZakodowana = zakodujWiadomosc(wiadomosc, ksia
28      System.out.println("Wiadomosc do zakodowania: ");
29      System.out.println(wiadomosc);
30      System.out.println("Wiadomosc zakodowana: ");
31      System.out.println(wiadomoscZakodowana);
32  }
```

Teraz możemy przetestować działanie kodowania.

```
 1  public class Main {
 2
 3      public static int[] zlicz(String wiadomosc) {
 4          int[] tabZliczajaca;
 5          char min = Character.MAX_VALUE;
 6          char max = Character.MIN_VALUE;
 7          for (int i = 0; i < wiadomosc.length(); i++) {
 8              if (min > wiadomosc.charAt(i))
 9                  min = wiadomosc.charAt(i);
10              if (max < wiadomosc.charAt(i))
11                  max = wiadomosc.charAt(i);
12          }
13          tabZliczajaca = new int[max - min + 1];
14          for (int i = 0; i < wiadomosc.length(); i++) {
15              tabZliczajaca[wiadomosc.charAt(i) - min]++;
16          }
17          return tabZliczajaca;
18      }
19
20      public static Wierzcholek sortowanie(Wierzcholek[] listaWier
21          int j = dlugoscListy;
22          do {
23              for (int i = 0; i < j - 1; i++) {
24                  if (listaWierzcholkow[i].czestosc < listaWierzch
25                      Wierzcholek temp = listaWierzcholkow[i];
26                      listaWierzcholkow[i] = listaWierzcholkow[i +
27                      listaWierzcholkow[i + 1] = temp;
28                  }
29              }
30              j--;
```

```
        } while (j > 1);

        int k = dlugoscListy;
        while (k > 1) {
            // wierzchołki o najmniejszej częstości znajdują się
            Wierzcholek pierwszyDoUsuniecia = listaWierzcholkow[
            Wierzcholek drugiDoUsuniecia = listaWierzcholkow[k -
            Wierzcholek nowyWierzcholek = new Wierzcholek('\0',
                    pierwszyDoUsuniecia.czestosc + drugiDoUsunie
            System.out.println("Usuwam i lacze w jeden następuja
            System.out.println("Wierzcholek o znaku: " + pierwsz
            System.out.println("Wierzcholek o znaku: " + drugiDo
            listaWierzcholkow[k - 2] = nowyWierzcholek;
            listaWierzcholkow[k - 1] = null;
            k--;
            // umieść nowy wierzchołek w odpowiednim miejscu
            for (int i = k - 1; i > 0; i--) {
                if (listaWierzcholkow[i - 1] == null
                        || listaWierzcholkow[i].czestosc > lista
                    Wierzcholek temp = listaWierzcholkow[i];
                    listaWierzcholkow[i] = listaWierzcholkow[i -
                    listaWierzcholkow[i - 1] = temp;
                } else
                    break;
            }
        }
        return listaWierzcholkow[0];
    }

    public static void wypiszKod(Wierzcholek doPrzeszukania, Str
        if (doPrzeszukania.lewy == null && doPrzeszukania.prawy
            System.out.println("Znak: " + doPrzeszukania.znak +
                    " czestosc: " + doPrzeszukania.czestosc +
                    " slowo kodowe: " + dotychczasowyKod);
        }
        if (doPrzeszukania.lewy != null) {
            wypiszKod(doPrzeszukania.lewy, dotychczasowyKod + "0
        }
        if (doPrzeszukania.prawy != null) {
            wypiszKod(doPrzeszukania.prawy, dotychczasowyKod + "
        }
    }
```

```
73
74      static int obecnyElementTablicySlow = 0;
75
76      private static Slowo[] zapiszKod(Wierzcholek doPrzeszukania,
77          if (doPrzeszukania.lewy == null && doPrzeszukania.prawy
78              listaSlow[obecnyElementTablicySlow] = new Slowo(doPr
79                      dotychczasowyKod);
80              obecnyElementTablicySlow++;
81          }
82          if (doPrzeszukania.lewy != null) {
83              zapiszKod(doPrzeszukania.lewy, dotychczasowyKod + "0
84          }
85          if (doPrzeszukania.prawy != null) {
86              zapiszKod(doPrzeszukania.prawy, dotychczasowyKod + "
87          }
88          return listaSlow;
89      }
90
91      static String zakodujZnak(char znak, Slowo[] ksiazkaSlow) {
92          for (int i = 0; i < ksiazkaSlow.length; i++) {
93              if (ksiazkaSlow[i].znak == znak)
94                  return ksiazkaSlow[i].slowoKodowe;
95          }
96          return "";// jeśli korzystamy z odpowiedniej funkcji kod
97      }
98
99      static String zakodujWiadomosc(String wiadomosc, Slowo[] ksi
100         String zakodowanaWiadomosc = "";
101         for (int i = 0; i < wiadomosc.length(); i++) {
102             zakodowanaWiadomosc += zakodujZnak(wiadomosc.charAt(
103         }
104         return zakodowanaWiadomosc;
105     }
106
107     public static void main(String[] args) {
108         String wiadomosc = "AABBCCDDDDDDDFFFEEER";
109         int[] tabZliczajaca = zlicz(wiadomosc);
110         char minZnak = Character.MAX_VALUE;
111         for (int i = 0; i < wiadomosc.length(); i++) {
112             if (minZnak > wiadomosc.charAt(i))
113                 minZnak = wiadomosc.charAt(i);
114         }
```

```
115         Wierzcholek[] listaWierzcholkow = new Wierzcholek[tabZli
116         int dlugoscListy = 0;
117         for (int i = 0; i < listaWierzcholkow.length; i++) {
118             if (tabZliczajaca[i] != 0) {
119                 listaWierzcholkow[dlugoscListy] = new Wierzchole
120                 dlugoscListy++;
121             }
122         }
123         Wierzcholek korzen = sortowanie(listaWierzcholkow, dlugo
124         // w przypadku, gdy w tekście znajduje się tylko jeden u
125         // obsłużyć ten wyjątek, ponieważ drzewo się nie zbuduje
126         if (dlugoscListy == 1)
127             korzen = new Wierzcholek('\0', 1, null, listaWierzch
128         Slowo[] ksiazkaSlow = new Slowo[dlugoscListy];
129         wypiszKod(korzen, "");
130         ksiazkaSlow = zapiszKod(korzen, "", ksiazkaSlow);
131         String wiadomoscZakodowana = zakodujWiadomosc(wiadomosc,
132         System.out.println("Wiadomosc do zakodowania: ");
133         System.out.println(wiadomosc);
134         System.out.println("Wiadomosc zakodowana: ");
135         System.out.println(wiadomoscZakodowana);
136     }
137 }
138
139 class Wierzcholek {
140     char znak;
141     int czestosc;
142     Wierzcholek lewy = null;
143     Wierzcholek prawy = null;
144
145     Wierzcholek(char znak, int czestosc, Wierzcholek lewy, Wierz
146         this.znak = znak;
147         this.czestosc = czestosc;
148         this.lewy = lewy;
149         this.prawy = prawy;
150     }
151 }
152
153 class Slowo {
154     char znak;
155     int czestosc;
156     String slowoKodowe;
```

```
157
158     Slowo(char znak, int czestosc, String slowoKodowe) {
159         this.znak = znak;
160         this.czestosc = czestosc;
161         this.slowoKodowe = slowoKodowe;
162     }
163 }
```

Program wypisuje:

```
1  Usuwam i lacze w jeden następujace wierzcholki:
2  Wierzcholek o znaku: R 1
3  Wierzcholek o znaku: C 2
4  Usuwam i lacze w jeden następujace wierzcholki:
5  Wierzcholek o znaku: B 2
6  Wierzcholek o znaku: A 2
7  Usuwam i lacze w jeden następujace wierzcholki:
8  Wierzcholek o znaku:  3
9  Wierzcholek o znaku: F 3
10 Usuwam i lacze w jeden następujace wierzcholki:
11 Wierzcholek o znaku: E 3
12 Wierzcholek o znaku:  4
13 Usuwam i lacze w jeden następujace wierzcholki:
14 Wierzcholek o znaku:  6
15 Wierzcholek o znaku:  7
16 Usuwam i lacze w jeden następujace wierzcholki:
17 Wierzcholek o znaku: D 7
18 Wierzcholek o znaku:  13
19 Znak: D czestosc: 7 slowo kodowe: 0
20 Znak: R czestosc: 1 slowo kodowe: 1000
21 Znak: C czestosc: 2 slowo kodowe: 1001
22 Znak: F czestosc: 3 slowo kodowe: 101
23 Znak: E czestosc: 3 slowo kodowe: 110
24 Znak: B czestosc: 2 slowo kodowe: 1110
25 Znak: A czestosc: 2 slowo kodowe: 1111
26 Wiadomosc do zakodowania:
27 AABBCCDDDDDDDFFFEEER
28 Wiadomosc zakodowana:
29 11111111111011101001100100000001011011011101101101000
```

> **Problem 4**
>
> W kolejnym kroku implementujemy odkodowywanie na podstawie utworzonego drzewa. Odkodowanie wiadomości na podstawie książki kodowej jest możliwe, ale nieefektywne. Informacje na temat algorytmów zachłannych, stosowanych przy przeszukiwaniu wszystkich możliwości, znajdziesz w e-materiale Algorytmy zachłanne.

Specyfikacja problemu:

Dane:

- `zakodowanaWiadomosc` – łańcuch znaków do odkodowania
- `korzen` – korzeń drzewa binarnego użytego do zakodowania wiadomości

Wynik:

- `zdekodowanaWiadomosc` – łańcuch znaków

Aby odkodować wiadomość, skorzystamy z drzewa binarnego, którego używaliśmy do kodowania. Zagłębiamy się rekurencyjnie w drzewo aż do odnalezienia liścia zawierającego znak. Powtarzamy tę procedurę do momentu, w którym skończy się łańcuch znaków:

```
static String odkodujWiadomosc(String wiadomosc, Wierzcholek korz
    Wierzcholek doPrzeszukania = korzen;
    String wiadomoscOdkodowana = "";
    for (int i = 0; i < wiadomosc.length(); i++) {
        if (wiadomosc.charAt(i) == '0')
            doPrzeszukania = doPrzeszukania.lewy;
        else
            doPrzeszukania = doPrzeszukania.prawy;
        if (doPrzeszukania.lewy == null || doPrzeszukania.prawy =
            wiadomoscOdkodowana += doPrzeszukania.znak;
            doPrzeszukania = korzen;
        }
    }
    return wiadomoscOdkodowana;
}
```

Aby przetestować działanie funkcji odkodowującej, potrzebujemy najpierw stworzyć drzewo binarne. Użyjemy w tym celu implementacji z poprzedniego zadania i połączymy kodowanie z odkodowywaniem.

```java
public class Main {

    public static int[] zlicz(String wiadomosc) {
        int[] tabZliczajaca;
        char min = Character.MAX_VALUE;
        char max = Character.MIN_VALUE;
        for (int i = 0; i < wiadomosc.length(); i++) {
            if (min > wiadomosc.charAt(i))
                min = wiadomosc.charAt(i);
            if (max < wiadomosc.charAt(i))
                max = wiadomosc.charAt(i);
        }
        tabZliczajaca = new int[max - min + 1];
        for (int i = 0; i < wiadomosc.length(); i++) {
            tabZliczajaca[wiadomosc.charAt(i) - min]++;
        }
        return tabZliczajaca;
    }

    public static Wierzcholek sortowanie(Wierzcholek[] listaWier

        int j = dlugoscListy;
        do {
            for (int i = 0; i < j - 1; i++) {
                if (listaWierzcholkow[i].czestosc < listaWierzch
                    Wierzcholek temp = listaWierzcholkow[i];
                    listaWierzcholkow[i] = listaWierzcholkow[i +
                    listaWierzcholkow[i + 1] = temp;
                }
            }
            j--;
        } while (j > 1);

        int k = dlugoscListy;
        while (k > 1) {
            // wierzchołki o najmniejszej częstości znajdują się
            Wierzcholek pierwszyDoUsuniecia = listaWierzcholkow[
            Wierzcholek drugiDoUsuniecia = listaWierzcholkow[k -
            Wierzcholek nowyWierzcholek = new Wierzcholek('\0',
                    pierwszyDoUsuniecia.czestosc + drugiDoUsunie
            System.out.println("Usuwam i lacze w jeden następuja
```

```
42              System.out.println("Wierzcholek o znaku: " + pierwsz
43              System.out.println("Wierzcholek o znaku: " + drugiDo
44              listaWierzcholkow[k - 2] = nowyWierzcholek;
45              listaWierzcholkow[k - 1] = null;
46              k--;
47              // umieść nowy wierzchołek w odpowiednim miejscu
48              for (int i = k - 1; i > 0; i--) {
49                  if (listaWierzcholkow[i - 1] == null
50                          || listaWierzcholkow[i].czestosc > lista
51                      Wierzcholek temp = listaWierzcholkow[i];
52                      listaWierzcholkow[i] = listaWierzcholkow[i -
53                      listaWierzcholkow[i - 1] = temp;
54                  } else break;
55              }
56          }
57          return listaWierzcholkow[0];
58      }
59
60      public static void wypiszKod(Wierzcholek doPrzeszukania, Str
61          if (doPrzeszukania.lewy == null && doPrzeszukania.prawy
62              System.out.println("Znak: " + doPrzeszukania.znak +
63                      " czestosc: " + doPrzeszukania.czestosc +
64                      " slowo kodowe: " + dotychczasowyKod);
65          }
66          if (doPrzeszukania.lewy != null) {
67              wypiszKod(doPrzeszukania.lewy, dotychczasowyKod + "0
68          }
69          if (doPrzeszukania.prawy != null) {
70              wypiszKod(doPrzeszukania.prawy, dotychczasowyKod + "
71          }
72      }
73
74      static int obecnyElementTablicySlow = 0;
75
76      private static Slowo[] zapiszKod(Wierzcholek doPrzeszukania,
77          if (doPrzeszukania.lewy == null && doPrzeszukania.prawy
78              listaSlow[obecnyElementTablicySlow] = new Slowo(doPr
79                      dotychczasowyKod);
80              obecnyElementTablicySlow++;
81          }
82          if (doPrzeszukania.lewy != null) {
83              zapiszKod(doPrzeszukania.lewy, dotychczasowyKod + "0
```

```
84          }
85          if (doPrzeszukania.prawy != null) {
86              zapiszKod(doPrzeszukania.prawy, dotychczasowyKod + "
87          }
88          return listaSlow;
89      }
90
91      static String zakodujZnak(char znak, Slowo[] ksiazkaSlow) {
92          for (int i = 0; i < ksiazkaSlow.length; i++) {
93              if (ksiazkaSlow[i].znak == znak)
94                  return ksiazkaSlow[i].slowoKodowe;
95          }
96          return "";// jeśli korzystamy z odpowiedniej funkcji kod
97      }
98
99      static String zakodujWiadomosc(String wiadomosc, Slowo[] ksi
100         String zakodowanaWiadomosc = "";
101         for (int i = 0; i < wiadomosc.length(); i++) {
102             zakodowanaWiadomosc += zakodujZnak(wiadomosc.charAt(
103         }
104         return zakodowanaWiadomosc;
105     }
106
107     static String odkodujWiadomosc(String wiadomosc, Wierzcholek
108         Wierzcholek doPrzeszukania = korzen;
109         String wiadomoscOdkodowana = "";
110         for (int i = 0; i < wiadomosc.length(); i++) {
111             if (wiadomosc.charAt(i) == '0')
112                 doPrzeszukania = doPrzeszukania.lewy;
113             else
114                 doPrzeszukania = doPrzeszukania.prawy;
115             if (doPrzeszukania.lewy == null || doPrzeszukania.pr
116                 wiadomoscOdkodowana += doPrzeszukania.znak;
117                 doPrzeszukania = korzen;
118             }
119         }
120         return wiadomoscOdkodowana;
121     }
122
123     public static void main(String[] args) {
124         String wiadomosc = "AABBCCDDDDDDDFFFEEER";
125         int[] tabZliczajaca = zlicz(wiadomosc);
```

```
126         char minZnak = Character.MAX_VALUE;
127         for (int i = 0; i < wiadomosc.length(); i++) {
128             if (minZnak > wiadomosc.charAt(i))
129                 minZnak = wiadomosc.charAt(i);
130         }
131         Wierzcholek[] listaWierzcholkow = new Wierzcholek[tabZli
132         int dlugoscListy = 0;
133         for (int i = 0; i < listaWierzcholkow.length; i++) {
134             if (tabZliczajaca[i] != 0) {
135                 listaWierzcholkow[dlugoscListy] = new Wierzchole
136                 dlugoscListy++;
137             }
138         }
139         Wierzcholek korzen = sortowanie(listaWierzcholkow, dlugo
140         // w przypadku, gdy w tekście znajduje się tylko jeden u
141         // obsłużyć ten wyjątek, ponieważ drzewo się nie zbuduje
142         if (dlugoscListy == 1)
143             korzen = new Wierzcholek('\0', 1, null, listaWierzch
144         Slowo[] ksiazkaSlow = new Slowo[dlugoscListy];
145         wypiszKod(korzen, "");
146         ksiazkaSlow = zapiszKod(korzen, "", ksiazkaSlow);
147         String wiadomoscZakodowana = zakodujWiadomosc(wiadomosc,
148         String wiadomoscOdkodowana = odkodujWiadomosc(wiadomoscZ
149         System.out.println("Wiadomosc do zakodowania: ");
150         System.out.println(wiadomosc);
151         System.out.println("Wiadomosc zakodowana: ");
152         System.out.println(wiadomoscZakodowana);
153         System.out.println("Wiadomosc odkodowana: ");
154         System.out.println(wiadomoscOdkodowana);
155         System.out.println("Oryginalna wiadomosc zajmuje w pamię
156                 + " bajtow, a po zakodowaniu zajmowalaby jedynie
157                 + (wiadomosc.length() + (wiadomosc.length() % 8
158     }
159 }
160
161 class Wierzcholek {
162     char znak;
163     int czestosc;
164     Wierzcholek lewy = null;
165     Wierzcholek prawy = null;
166
167     Wierzcholek(char znak, int czestosc, Wierzcholek lewy, Wierz
```

```
        this.znak = znak;
        this.czestosc = czestosc;
        this.lewy = lewy;
        this.prawy = prawy;
    }
}

class Slowo {
    char znak;
    int czestosc;
    String slowoKodowe;

    Slowo(char znak, int czestosc, String slowoKodowe) {
        this.znak = znak;
        this.czestosc = czestosc;
        this.slowoKodowe = slowoKodowe;
    }
}
```

Program podaje na wyjście następujące wiadomości:

```
Usuwam i lacze w jeden następujace wierzcholki:
Wierzcholek o znaku: R 1
Wierzcholek o znaku: C 2
Usuwam i lacze w jeden następujace wierzcholki:
Wierzcholek o znaku: B 2
Wierzcholek o znaku: A 2
Usuwam i lacze w jeden następujace wierzcholki:
Wierzcholek o znaku:  3
Wierzcholek o znaku: F 3
Usuwam i lacze w jeden następujace wierzcholki:
Wierzcholek o znaku: E 3
Wierzcholek o znaku:  4
Usuwam i lacze w jeden następujace wierzcholki:
Wierzcholek o znaku:  6
Wierzcholek o znaku:  7
Usuwam i lacze w jeden następujace wierzcholki:
Wierzcholek o znaku: D 7
Wierzcholek o znaku:  13
Znak: D czestosc: 7 slowo kodowe: 0
Znak: R czestosc: 1 slowo kodowe: 1000
```

```
Znak: C czestosc: 2 slowo kodowe: 1001
Znak: F czestosc: 3 slowo kodowe: 101
Znak: E czestosc: 3 slowo kodowe: 110
Znak: B czestosc: 2 slowo kodowe: 1110
Znak: A czestosc: 2 slowo kodowe: 1111
Wiadomosc do zakodowania:
AABBCCDDDDDDDFFFEEER
Wiadomosc zakodowana:
11111111111011101001100100000001011011011101101101000
Wiadomosc odkodowana:
AABBCCDDDDDDDFFFEEER
Oryginalna wiadomosc zajmuje w pamięci: 40 bajtow, a po zakodowan
```

> **Już wiesz**
>
> Podsumujmy najważniejsze informacje:
>
> - algorytm Huffmana generuje kod zero-jedynkowy;
> - kod każdego znaku nie jest początkowym fragmentem kodu innego znaku;
> - generowany kod jest tzw. kodem prefiksowym (żaden kod nie jest prefiksem innej litery), który pozwala na jednoznaczne dekodowanie zapisanego znaku;
> - kod jest tworzony w taki sposób, aby średnia długość kodu znaku była najkrótsza.

Słownik

> **bezstratna kompresja danych**
>
> metoda zmniejszania objętości danych, w której informacja po skompresowaniu jest dekompresowalna do postaci początkowej
> **drzewo binarne**
>
> rodzaj drzewa (struktury danych), w którym każdy wierzchołek ma tylko dwóch potomków (lewego i prawego) oraz tylko jeden początek, czyli korzeń; wierzchołki bez potomków określane są jako liście
> **książka kodowa**
>
> zbiór słów kodowych, który służy do kodowania wiadomości; może służyć również do dekodowania, jednak w algorytmie Huffmana unika się tego, gdyż jest to nieefektywne
> **stratna kompresja danych**
>
> metoda zmniejszania objętości danych, w wyniku której traci się część informacji (dane po zdekompresowaniu nie mają takiej samej postaci, mogą być zniekształcone)

Symulacja interaktywna

Polecenie 1

Zapoznaj się z symulacją interaktywną. Przedstawia ona przykładowy proces tworzenia grafu kodującego za pomocą algorytmu Huffmana. Czy potrafisz zakodować wiadomość zgodnie z symulacją?

Symulacja 1

Zasób interaktywny dostępny pod adresem https://zpe.gov.pl/a/DPqjRZojm

Polecenie 2

Wpisując do symulacji odpowiednie dane, czyli znaki do zakodowania i ich częstość, utwórz drzewo binarne. Następnie za jego pomocą zakoduj wiadomość: `"Jestem tekstem testowym"`. Podczas kodowania uwzględnij spacje.

Polecenie 3

Zaimplementuj w języku Java algorytm kodowania Huffmana dla ciągu znaków. Podczas łączenia wierzchołków upewnij się, że lewy syn będzie węzłem z najmniejszą możliwą częstością, a prawy następnym w kolejności. Podczas porównywania węzłów z tą samą częstością priorytet powinien mieć ten, który podczas porównywania przetrzymywanych znaków będzie miał mniejszą wartość. Łącząc dwa węzły, zapisz w wierzchołku wynikowym znak mający niższą wartość. Przetestuj swoje rozwiązanie dla podanego ciągu znaków:

```
wiadomosc = "Jestem tekstem testowym"
```

Zakoduj również spacje. Swoje rozwiązanie porównaj z symulacją. Jeśli pomimo poprawnej konstrukcji drzewa otrzymasz inne kodowanie, zastanów się, czemu tak się stało.

Specyfikacja:

Dane:

- `wiadomosc` – łańcuch znaków do zakodowania

Wynik:

- `zakodowanaWiadomosc` – zakodowany łańcuch znaków

Twoje zadania

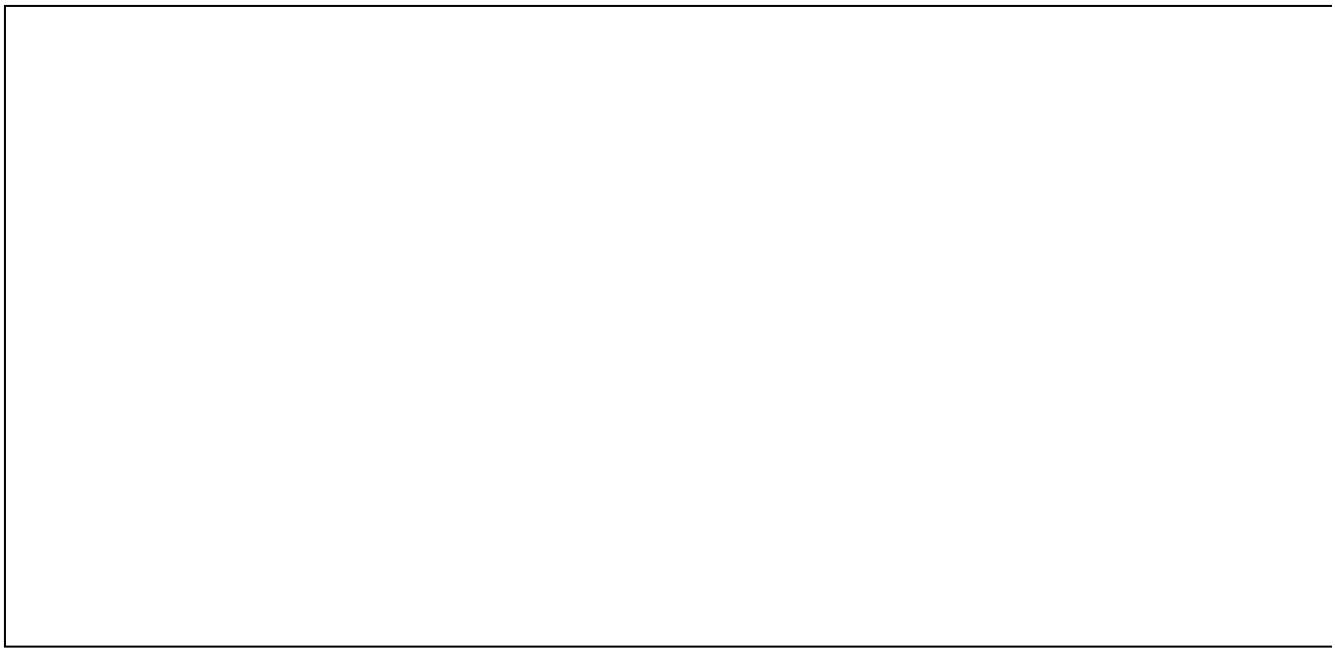
1. Program zakodowuje ciąg znaków `wiadomosc`.

```
public class Main {
    public static void main(String[] args) {

    String wiadomosc = "Jestem tekstem testowym";
    // tu wpisz swój kod

    }
}
```

```
1
```



Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program zliczający wystąpienia każdego znaku znajdującego się w łańcuchu znaków `wiadomosc`. Program powinien wypisywać wyniki zgodnie z następującym schematem:

```
Znak:Liczba wystąpień znaku
```

Znaki powinny być wypisane w kolejności rosnącej według ich kodowania w języku Java. Na przykład znak A powinien zostać wypisany przed znakiem B. Nie powinny zostać wypisane znaki, które nie były zawarte w łańcuchu znaków `wiadomosc`.

Przetestuj swój program dla podanego ciągu znaków:

```
wiadomosc = "AAAA121345122 - Zlicz mnie"
```

Dla przedstawionych danych program powinien wypisać:

```
 :3
-:1
1:3
2:3
3:1
4:1
5:1
A:4
Z:1
c:1
e:1
i:2
l:1
m:1
n:1
z:1
```

Specyfikacja:

Dane:

- `wiadomosc` – łańcuch znaków, w którym należy zliczyć wystąpienia poszczególnych znaków

Wynik:

- `(char) (i + min) + ":" + zliczoneWystapienia[i]` – wyniki zliczenia zgodnie ze schematem: `Znak:LiczbaWystapien`

Twoje zadania

1. Program zlicza liczbę wystąpień poszczególnych znaków w łańcuchu `wiadomosc`, a następnie wypisuje rezultaty zgodnie ze schematem podanym w zadaniu. Pamiętaj, że w łańcuchu występują również spacje, które także należy zliczyć.

Ćwiczenie 2

Napisz program, który zakoduje łańcuch znaków `wiadomosc`, korzystając z podanej książki kodowej. Program powinien wypisać zakodowaną wiadomość w postaci ciągu zer i jedynek. Swój program przetestuj dla podanego ciągu znaków:

```
wiadomosc = "AAABCDEFGHAABDDEFFG - Zakoduj mnie proszę"
```

Specyfikacja:

Dane:

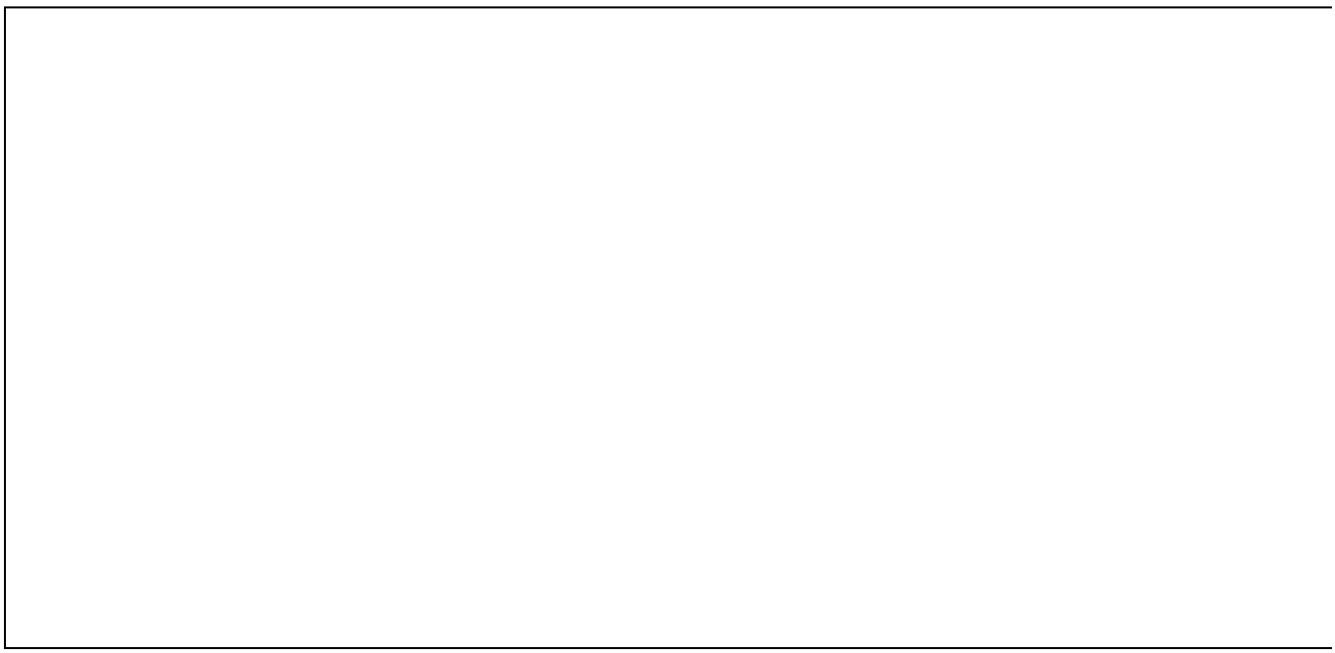
- `wiadomosc` – łańcuch znaków do zakodowania

Wynik:

- `zakodowanaWiadomosc` – zakodowany łańcuch znaków

Twoje zadania

1. Poprawnie zakoduj łańcuch znaków `wiadomosc`, a następnie wypisz jej zakodowaną postać.



Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Algorytm Huffmana w języku Java

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres podstawowy i 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.

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi, w tym: znajomość zasad działania urządzeń cyfrowych i sieci komputerowych oraz wykonywania obliczeń i programów.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

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

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

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

Zakres podstawowy. Uczeń:

1) projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);

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

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi.

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

2) dokonuje kompresji informacji, objaśnia różnice między kompresją stratną i bezstratną tekstów, obrazów, dźwięków, filmów;

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):

- Prześledzisz implementację algorytmu Huffmana w języku Java.
- Zaimplementujesz program w języku Java kodujący ciąg znaków za pomocą algorytmu Huffmana.
- Wykonasz ćwiczenia, wykorzystując w praktyce wiedzę dotyczącą omawianego zagadnienia.
- Wyjaśnisz, jak konstruować i przeszukiwać drzewa binarne.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

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

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 Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Algorytm Huffmana w języku Java". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj". Na forum klasy uczniowie analizują problemy 1-3. Następnie pracując indywidualnie, uczniowie powtarzają zaprezentowaną implementację algorytmu na swoich komputerach.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Symulacja interaktywna". Uczniowie indywidualnie wykonują polecenie 1. Omawiają je na forum klasy. W kolejnym kroku w parach wykonują polecenie 2 i 3, a po zakończeniu pracy porównują swoje rozwiązania z inną grupą. Nauczyciel weryfikuje poprawność rozwiązań, w razie potrzeby wyjaśnia niezrozumiałe kwestie.

3. **Ćwiczenie umiejętności.** Uczniowie, pracując w parach, wykonują ćwiczenie nr 1 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność pisanych kodów, porównuje je i omawia wraz z uczniami. Wskazuje najbardziej efektywne rozwiązanie.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia z programowania w Javie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 2 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Treści w sekcji „Symulacja interaktywna" można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.