Zestaw 6

C#, biblioteki .NET

14 kwietnia 2004

Streszczenie

Rozwiązanie zadań w tym zestawie polega na napisaniu programów w języku C#. Do uzyskania 10+1 punktów.

1. Zaimplementować kolekcję `Set` działającą jak zbiór, odrzucający duplikaty elementów.
   [**1p**]
2. Przetestować w praktyce składanie enumeratorów opakowujących. Ściślej, uzupełnić szkic kodu ze str. 211 z podręcznika, tak aby kod zadziałał zgodnie z sugestią.
   [**2p**]
3. Napisać klasę `TArray`, która będzie pewną specjalną implementacją tablicy elementów.
   Wewnątrz obiektu klasy dane powinny być przechowywane na drzewie, w którym każdy węzeł ma 10 synów, oznaczonych indeksami od 0 do 9. Aby dostać się do elementu o indeksie $i = \sum_{j=0}^{k} 10^j * i_j$ przechodzimy drzewo, na poziomie j przechodząc do syna i_j.
   Na przykład chcąc uzyskać dostęp do elementu o indeksie 7 wybieramy 7 syna korzenia drzewa i odczytujemy zapamiętany w nim element. Aby uzyskać dostęp do elementu o indeksie 145 przechodzimy kolejnoprzez 5-ego, 4-ego i 1-ego syna kolejnych węzłów począwszy od korzenia.
   Odpowiednie gałęzie drzewa powinny być budowane tylko wtedy, kiedy do tablicy dodawny jest element o odpowiednim indeksie. Na przykład dodanie do tablicy elementu o indeksie 10000000000 powinno spowodować powstanie tylko jednej długiej gałęzi od 0-ego syna korzenia, przez dziesięciu kolejnych synów kolejnych węzłów.
   Bezpośrednio po zainicjowaniu korzeń drzewa powinien mieć tylko 10 pustych referencji na kolejnych synów.
   Dzięki takiej konstrukcji użytkownik będzie mógł dodać na przykład element o indeksie 1 i element o indeksie 10000, a w drzewie będą przechowane tylko te 2 elementy (plus oczywiście puste referencje na pozostałe elementy w kolejnych węzłach). Tablica nie będzie więc (jak zwykła tablica liniowa) zużywać miejsca na wszystkie brakujące elementy między 1 a 10000.
   Tego rodzaju tablice są dostępne w niektórych językach programowania.
   Zdefiniować odpowiedni indekser, tak aby do elementów tablicy można było odwoływać się w "zwykły" sposób, na przykład:

```
TArray a = new TArray();
a[17]      = 5;
a[1000000] = 176;
```

   Zdefiniować odpowiedni enumerator, tak aby elementy tablicy można było przeglądać w "zwykły" sposób, na przykład:

```
TArray a = new TArray();
a[17]      = 5;
```

```
a[1000000] = 176;
foreach ( int i in a )
...
```

Porównać wydajność (tworzenie, przeglądanie):

- TArray
- ArrayList
- zwykłych tablic

[**3p**]

4. Zaprojektować kolekcję `BiArrayList`, która oprócz standardowego enumeratora dostępnego za pomocą metody `GetEnumerator`, będzie zwracać *enumerator dwukierunkowy* za pomocą funkcji `GetBidirectionalEnumerator`.

   Enumerator dwukierunkowy powinien oprócz trzech elementów interfejsu *IEnumerable* udostępniać metodę `MovePrev`, która będzie cofać enumerację do poprzedniego elementu względem bieżącego.

   [**2p**]

5. Wykorzystać refleksje do realizacji dynamicznego łączenia modułów. Ściślej, zaprojektować jakiś interfejs i jego deklarację umieścić w module.

   Następnie przygotować dwa moduły, z których każdy zawierałby implementację interefejsu.

   W module głównym pozwolić użytkownikowi wybrać jedną z tych dwóch bibliotek (przez wskazanie nazwy), a następnie dynamicznie załadować bibliotekę przez refleksje, wykreować obiekt i wywołać jego metody oraz przechwycić wyniki.

   Jeden dodatkowy punkt za wersję modułu głównego, która sama przeskanuje biblioteki w bieżącym folderze i sprawdzi, które z nich nadają się do dynamicznego łączenia (w których z nich znajdują się definicje klas implementujących wskazany interfejs).

   [**2p+1**]