Lab. 3. Implementacja kolejki priorytetowej

15-03-2016

1 Kolejka priorytetowa

Kolejka priorytetowa jest listą elementów, w której każdy element ma dołączony swój priorytet obsługi. Na zestawie priorytetów istnieje naturalny porządek, także można określić, który element z listy ma największy priorytet. Elementy dodawane są do listy w dowolnej (losowej) kolejności, jednakże usunięcie elementu z listy następuje zgodnie z wartością priorytetu – im wyższy tym wcześniej zostaje usunięty. Uwaga: w naszej implementacji będziemy przyjmować, iż wysoki priorytet oznacza niską wartość odpowiedniej zmiennej.

1.1 Kopiec binarny

Kopiec binarny (ang. *binary heap*) – tablicowa struktura danych reprezentująca drzewo binarne, którego wszystkie poziomy z wyjątkiem ostatniego muszą być pełne. W przypadku, gdy ostatni poziom drzewa nie jest pełny, liście ułożone są od lewej do prawej strony drzewa. Wyróżniamy dwa rodzaje kopców binarnych: kopce binarne typu *max* w których wartość danego węzła niebędącego korzeniem jest zawsze mniejsza niż wartość jego rodzica oraz kopce binarne typu *min* w których wartość danego węzła niebędącego korzeniem jest zawsze większa niż wartość jego rodzica.

1.1.1 Dodawanie nowych wierzchołków

Załóżmy, że kopiec składa się z n elementów, zaś elementy uporządkowane są od największych (warunek kopca brzmi więc: każdy element jest większy od swoich dzieci). Dodawany wierzchołek ma klucz równy k:

```
1. wstaw wierzchołek na pozycję n+1
2. zamieniaj pozycjami z rodzicem (przepychaj w górę) aż do przywrócenia
warunku kopca.
```

1.1.2 Usuwanie wierzchołka ze szczytu kopca

```
1. usuń wierzchołek ze szczytu kopca
2. przestaw ostatni wierzchołek z pozycji n na szczyt kopca; niech k
oznacza jego klucz
3. spychaj przestawiony wierzchołek w dół, zamieniając pozycjami z
mniejszymi z dzieci, aż do przywrócenia warunku kopca.
```

2 Implementacja kolejki priorytetowej (reprezentacja za pomocą kopca binarnego)

1. Należy dołączyć plik `lab3.h`.
2. Definiujemy klasę `PriorityQueue` która dziedziczy (public virtual) po `Container` i zawiera czysto wirtualne metody

```
virtual void Enqueue (Object&) = 0;
virtual Object& FindMin() const = 0;
virtual Object& DequeueMin () = 0;
```

3. Definiujemy klasę `BinaryHeap` która dziedziczy publicznie po `PriorityQueue` i zawiera:
   - prywatne `Array<Object*> array`,
   - konstruktor pobierający `unsigned int` i wywołujący konstruktordla zmiennej `array`
   - destruktor, który wywołuje metodę `Purge()`
   - metoda `Purge()` sprawdza, czy BinaryHeap jest właścicielem danych i jeśli tak to kasuje elementy ze zmiennej `array`.
   - metoda `Enqueue` zgodna z opisem j.w.
   - metoda `FindMin` zwraca minimalną wartość
   - metoda `DequeueMin` zgodna z opisem j.w.
   - jeśli trzeba to wszystkie metody wirtualne z klas bazowych muszą być zaimplementowane (nawet w trywialny sposób) aby dało się stworzyć obiekty (nie może to być klasa abstrakcyjna),
4. Główny program: tworzymy kolejkę, dodajemy do kolejki kilka elementów typu `Int`, następnie wyciągamy z je z kolejki.

```
void BinaryHeap::Enqueue (Object& object){
  if (count == array.Length ())
    throw domain_error ("priority queue is full");
  ++count;
  unsigned int i = count;
  while (i > 1 && *array [i / 2] > object){
    array [i] = array [i / 2];
    i /= 2;
  }
  array [i] = &object;
}

Object& BinaryHeap::DequeueMin (){
  if (count == 0)
    throw domain_error ("priority queue is empty");
  Object& result = *array [1];
  Object& last = *array [count];
  --count;
  unsigned int i = 1;
  while (2 * i < count + 1){
    unsigned int child = 2 * i;
    if (child + 1 < count + 1 && *array [child + 1] < *array [child])
      child += 1;
    if (last <= *array [child])
      break;
    array [i] = array [child];
    i = child;
  }
  array [i] = &last;
  return result;
}
```