Lista 7

Zadanie 3

---

1 Problem

Należy zaimplementować kolejkę priorytetową na bazie Binary Search Tree.

2 Concept

Musimy odpowiednio zaimplementować wszystkie operacje, które można wykonać na kolejce priorytetowej:

1. `Insert`
2. `Minimum`
3. `ExtractMin`
4. `DecreaseKey`
5. `Union`
6. `Delete`

W celu osiągnięcia jak najlepszej złożoności obliczeniowej użyjemy tutaj drzew czerwono-czarnych, które są rozszerzeniem drzew BST.

3 Rozwiązanie

W przypadku operacji `Insert`, `Minimum`, `Delete` wykorzystujemy standardowe operacje o takiej samej nazwie, które już są w RB-Tree. Złożoność obliczeniowa tych procedur to $O(\lg n)$.

Operacja `ExtractMin` działa w taki sam sposób jak `Minimum`, przy czym dodatkowo wywoływana jest funkcja `Delete` na tym samym elemencie. Złożoność obliczeniowa wynosi $O(\lg n)$, ponieważ obie te operacje mają taką złożoność.

Operacja `DecreaseKey` najpierw usuwa zadany element i wstawia go ponownie ze zmniejszonym priorytetem. Złożoność obliczeniowa operacji `Insert` i `Delete` wynosi $O(\lg n)$ więc złożoność niniejszej operacji również wynosi $O(\lg n)$.

Operacja `Union`(A, B) najpierw wykonuje operację `inorder` na obu drzewach, następnie wykonuje operację `merge` która scala te dwie posortowane tablice w jedną, a następnie tworzy nowe drzewo przy pomocy poniższego algorytmu:

Algorithm 1 `sorted_array_to_RBT`

```
n = length(merged)
if n = 0 then:
    return
end if
middle ← ⌊n/2⌋
root ← merged[middle]
root.left ← sorted_array_to_RBT(merged[0...middle − 1])
root.right ← sorted_array_to_RBT(merged[middle + 1...n])
return root
```

Złożoność powyższego algorytmu wynosi $O(n)$, ponieważ wykonujemy n razy operacje o złożoności $O(1)$. Złożoność obliczeniowa obu pozostałych operacji `inorder` i `merge` wynosi $O(n)$. Zatem złożoność operacji `Union` również wynosi $O(n)$.

Porównanie złożoności obliczeniowych dla dwóch implementacji kolejki priorytetowej:

	Drzewo binarne (RB-Tree)	Kopiec binarny
`Insert`	$O(\lg n)$	$O(\lg n)$
`Minimum`	$O(\lg n)$	$O(1)$
`ExtractMin`	$O(\lg n)$	$O(\lg n)$
`DecreaseKey`	$O(\lg n)$	$O(\lg n)$
`Union`	$O(n)$	$O(n)$
`Delete`	$O(\lg n)$	$O(\lg n)$