Implementacja zbioru

Definicja:

Zbiór (set) -nieuporządkowana grupa elementów.

Na nasze potrzeby, dla uproszczenia, możemy założyć, że ma on przechowywać liczby naturalne (jeśli obiekty mają być innego typu, to łatwo przypisać do nich liczby naturalne).

Metody:

- Insert - dodaje element do zbioru
- Withdraw – usuwa element ze zbioru
- isInSet - sprawdza czy dany element jest we zbiorze
- suma – zwraca sumę dwóch zbiorów (elementy nie duplikują się, jeśli są w obu sumowanych zbiorach)
- część wspólna – zwraca część wspólną dwóch zbiorów (elementy które są jednocześnie w jednym i w drugim zbiorze)
- różnica (A-B) -usuwa ze zbioru A elementy, które są także w B.
- równość – sprawdza czy oba zbiory zawierają dokładnie te same elementy
- zawieranie - sprawdza, czy zbiór B jest podzbiorem zbioru A (tzn. czy wszystkie elementy B są elementami A)

Sumę, różnicę, część wspólną, równość zbiorów oraz ich zawieranie proszę zaimplementować jako przeciążone operatory: + , - , * , == , <= i >=

Dodatkowo proszę u implementację metod:

- getSize - sprawdza rozmiar
- clearSet – czyści zbiór
- printSet – wyświetla elementy

Sposoby implementacji i ich wpływ na wydajność

Implementacja za pomocą tablicy.

Mamy tablicę o określonej długości przechowującą wartości bool. W takiej implementacji mamy ograniczenie na maksymalną wartość liczbową przechowywaną w zbiorze. Jeśli dany element (liczba) jest w zbiorze to w tablicy dla danego indeksu jest true, jeśli nie ma to false.

Zalety:

-prosta implementacja

-bardzo szybkie sumowanie, liczenie części wspólnej (czas rzędu O(N), gdzie N to długość tablicy)

-dodawanie elementu, usuwanie elementu lub sprawdzanie czy element jest w zbiorze w czasie O(1).

Wady:

-Ograniczona liczba elementów w zbiorze.

-Duże wymagania odnośnie pamięci (szczególnie widoczne gdy mamy stosunkowo mało elementów, ale o dużych indeksach).

Implementacja za pomocą listy

-kolejne elementy dodawane są do listy (w kolejności ich dodawania).

Zalety

-prosta implementacja

-lista jest rozszerzalna, zatem zajmuje w pamięci tyle miejsca ile jest elementów w zbiorze

-nie ma ograniczenia na liczbę elementów (w przeciwieństwie do implementacji jako tablica)

Wady

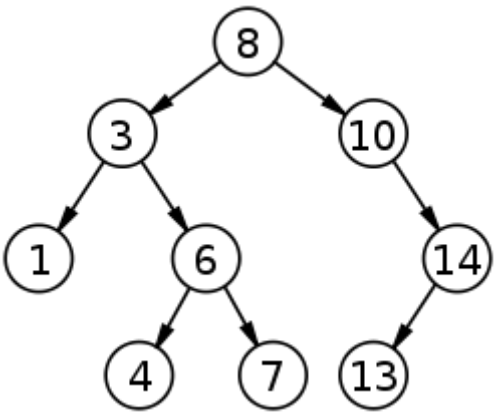
-stosunkowo niska szybkość działania:

-sprawdzanie czy element należy do zbioru O(N)

-suma, część wspólna $O(N^2)$

Dla chętnych - implementacja za pomocą drzewa binarnego poszukiwań (BST).

Działa to tak, każdy element drzewa (oprócz korzenia), ma co najwyżej dwóch potomków: lewego lub prawego (element jest „rodzicem" potomków). Lewy potomek jest zawsze mniejszy od rodzica a prawy większy od rodzica. Dzięki temu sprawdzanie, czy dany element należy do drzewa o N elementach w większości przypadków trzeba wykonać jedynie log(N) sprawdzeń (a nie N sprawdzeń jak dla listy). Ale trzeba mieć na uwadze, że w szczególnie złym przypadku (np. dodawanie do drzewa liczb „po kolei") drzewo może się jednak zredukować do listy.



Wizualizacja binarnego drzewa poszukiwań (za Wikipedig)

Zalety

-mała zajętość pamięci

-stosunkowo szybka : sprawdzanie czy element należy do zbioru O(log(N)), suma, część wspólna

O(N log(N))

Wady:

-Dosyć skomplikowana implementacja.

-Dodawanie w czasie log(N)

Literatura:

https://eduinf.waw.pl/inf/alg/001_search/0121d.php

Generalnie to co będziemy robić na ćwiczeniach, to nie jest jakiś „rocket science”. Omawiane algorytmy są dostępne w Internecie. Co jakiś czas będę jednak podawać linki do stron www, gdzie dobrze omówiono dane zagadnienie (nawet, jeśli przy okazji podano przykład implementacji). Bardzo jednak zachęcam, aby próbowali Państwo tworzyć własne implementacje, a nie korzystali z gotowych, gdyż samodzielne tworzenie algorytmów bardzo rozwija (wymaga przemyślenia tematu).