Generic Programming

Project Abstract

**Team members:**

Anish Poddar – PES1201700269

Lamya Bhasin – PES1201701244

**Topic:**

Implementing generic sparse set with zero initialization.

**Project Overview:**

It is a project in which array values can be left uninitialized and then read during normal operations. The sleaziness of uninitialized data access is offset by performance improvements: some important operations change from linear to constant time.

Alfred Aho, John Hopcroft, and Jeffrey Ullman's 1974 book *The Design and Analysis of Computer Algorithms* hints at the trick in an exercise (Chapter 2, exercise 2.12):

“Develop a technique to initialize an entry of a matrix to zero the first time it is accessed, thereby eliminating the *O*(||*V*||2) time to initialize an adjacency matrix.”

Jon Bentley's 1986 book [*Programming Pearls*](http://www.cs.bell-labs.com/cm/cs/pearls/) expands on the exercise (Column 1, exercise 8; [exercise 9](http://www.cs.bell-labs.com/cm/cs/pearls/sec016.html) in the Second Edition):

“One problem with trading more space for less time is that initializing the space can itself take a great deal of time. We circumvent this problem by designing a technique to initialize an entry of a vector to zero the first time it is accessed. The scheme uses constant time for initialization and each vector access; we use extra space proportional to the size of the vector. Because this method reduces initialization time by using even more space, it should be considered only when space is cheap, time is dear, and the vector is sparse.”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Operation* |  | *Bit Vector* |  |  | *Sparse set* |
| is-member |  | *O*(1) |  |  | *O*(1) |
| add-member |  | *O*(1) |  |  | *O*(1) |
| clear-set |  | *O*(*m*) |  |  | *O*(1) |
| iterate |  | *O*(*m*) |  |  | *O*(*n*) |

The sparse set is as fast or faster than bit vectors for every operation

The only problem is the space cost: two words replace each bit. Still, there are times when the speed differences are enough to balance the added memory cost.

Another situation where sparse sets are the better choice is work queue-based graph traversal algorithms. Iteration over sparse sets visits elements in the order they were inserted, so that new entries inserted during the iteration will be visited later in the same iteration. In contrast, iteration over bit vectors visits elements in integer order, so that new elements inserted during traversal might be missed, requiring repeated iterations

In conclusion, the project will implement a generic data structure which behaves like a set- the sparse array will be implemented as a set of structures, where each structure contains the index of an array member and a pointer to it (this member will be of a generic type). We'll maintain this structure array in sorted order of index, so both insertion and searching will be of the order of the number of elements in the sparse array.

The sparse array will support insertion, deletion, search and update.