Data Structures Library

Submitted as Mini Project for DAA: Design and Analysis of Algorithms
Semester IV

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering

Mehul Thakral

PES1201701122

Skanda VC

PES1201700987

Under Guidance of

Prof. Shruti Kaivalya

Assistant Professor

Jan 2019- May 2019

Department of Computer Science and Engineering

PES UNIVERSITY

Outer Ring Rd, Banashankari 3rd Stage, Banashankari, Bengaluru, Karnataka 560085

www.pes.edu

Abstract

A high performance generic Library of Popular Data Structures for C implemented using macros for handling input of various data types and thus providing basic functionalities associated with each data structure implemented in algorithmically superior ways.Data structures provided involve namely vector, doubly linked list, stack, queue, heap, self-balancing tree(AVL tree), dictionary, matrix.User of the library can use a particular data structure by simply including the header file for the corresponding structure and usage of functions understood by going through a simple documentation.

High Level Algorithm

- **Vector:** Functions implemented include initializing a new vector(with variable size input), referencing/assigning an element, pushing(with variable size input), popping an element and freeing the entire structure.
- **Doubly Linked List:** Functions implemented include initializing a new node, inserting a new node at head/tail/middle,deleting node from any position,printing DLL in forward/backward direction.
- **Stack:** Functions implemented include initializing a new node of stack, checking is stack empty, pushing a new element, popping a new element, reading element at top.
- **Queue:** Functions implemented include creating a new queue, creating a new node, enqueueing an element, dequeuing an element.
- **Dictionary** : Functions implemented include creating a new dictionary, inserting into a dictionary, inserting into dictionary, search a dictionary and deleting from a dictionary.
- **Heap :** Functions implemented include creating a new min or max heap, inserting into heap, deleting max/min element from heap

- **Matrix :** Functions implemented include functions for creating a new matrix, inserting elements dynamically rowise, multiply matrices, gaussian elimination, add and subtract 2 matrices, find transpose and inverse of matrices.
- **AVL Tree :** Functions implemented include creating a new avl tree, inserting into AVL tree, deleting from AVL tree and level order traversal.

Test Results

```c
#include "vector.h"

typedef char* string;

typedef struct {
    int x, y;
} Tuple;
      skandavc18, 8 days ago • pushing all the code
int main(void)
{
    printf("Output for String:\n");
    qvec(string) *sv = qvec_new(string, "Who", "are", "you?");
    qvec_print(sv);
    qvec_at(sv, 2) = "we?";
    qvec_print(sv);
    qvec_free(sv);

    printf("Output for int:\n");
    qvec(int) *iv = qvec_new(int, 1, 2, 3, 4);
    qvec_print(iv);
    printf("%d\n", qvec_pop(iv));
    qvec_free(iv);

    printf("Output for Double:\n");
    qvec(double) *dv = qvec_new(double, 1, 2, 3, 4);
    qvec_print(dv);
    printf("%lf\n", qvec_pop(dv));
    qvec_free(dv);

    printf("Output for Tuple:\n");
    qvec(Tuple) *tv = qvec_new(Tuple, { .x = 0, .y = 1 }, { 4, 2 }, { 5, 4 });
    printf("%d\n", qvec_at(tv, 1).x);
    printf("%d\n", qvec_at(tv, 2).x);
    qvec_free(tv);
}
```

Fig 1: Showing typical usage of library particularly vector for different core data types(string[char*] , int, double) and even for user defined data type(tuple).

Outputs obtained by using for different data types for each data structure:

- **Vector:**

```
mehul@mehul-HP-250-G5-Notebook-PC:~/DS_lib$ gcc vector.c
mehul@mehul-HP-250-G5-Notebook-PC:~/DS_lib$ ./a.out
Output for String:
[Who, are, you?]
[Who, are, we?]

Output for int:
[1, 2, 3, 4]
4

Output for Double:
[1.000000, 2.000000, 3.000000, 4.000000]
4.000000

Output for Tuple:
4
5
```

Fig 2: Showing output of library usage particularly vector for different core data types(string[char*] , int, double) and even for user defined data type(tuple).

- **Doubly linked list:**

```
Insert nodes at tail: hi, hello, how

Your updated linked list in FORWARD ORDER:
hi hello how

Insert node: ? at middle. Position: 2

Your updated linked list in FORWARD ORDER:
hi ? hello how

Delete item of position number 44

Your updated linked list in FORWARD ORDER:
hi ? hello

Insert nodes at front: r

Your updated linked list in FORWARD ORDER:
r hi ? hello

Insert nodes at tail: what

Your updated linked list in FORWARD ORDER:
r hi ? hello what

Insert nodes at front: r

Your updated linked list in FORWARD ORDER:
r r hi ? hello what

Delete first node of list

Your updated linked list in FORWARD ORDER:
r hi ? hello what

Insert nodes at tail: u, doing, these, days

Your updated linked list in FORWARD ORDER:
r hi ? hello what u doing these days


Your full linked list in REVERSE ORDER:
days these doing u what hello ? hi
```

Fig 3: Showing output of library usage particularly doubly linked list for different core data types(string[char*] , int, double).

- **Stack:**

```
mehul@mehul-HP-250-G5-Notebook-PC:~/DS_lib$ ./a.out
Output for Int:
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top element is 20

Output for Double:
10.000000 pushed to stack
20.000000 pushed to stack
30.000000 pushed to stack
30.000000 popped from stack
Top element is 20.000000

Output for string:
Hello pushed to stack
Hi pushed to stack
Bye pushed to stack
Bye popped from stack
Top element is Hi
```

Fig 4: Showing output of library usage particularly stack for different core data types(string[char*] , int, double).

- **Queue:**

```
mehul@mehul-HP-250-G5-Notebook-PC:~/DS_lib$ gcc queue.c
mehul@mehul-HP-250-G5-Notebook-PC:~/DS_lib$ ./a.out
Output for int:
Dequeued item is 30

Output for double:
Dequeued item is 30.000000

Output for string:
Dequeued item is r
```

Fig 5: Showing output of library usage particularly queue for different core data types(string[char*] , int, double).

- **Dictionary :**

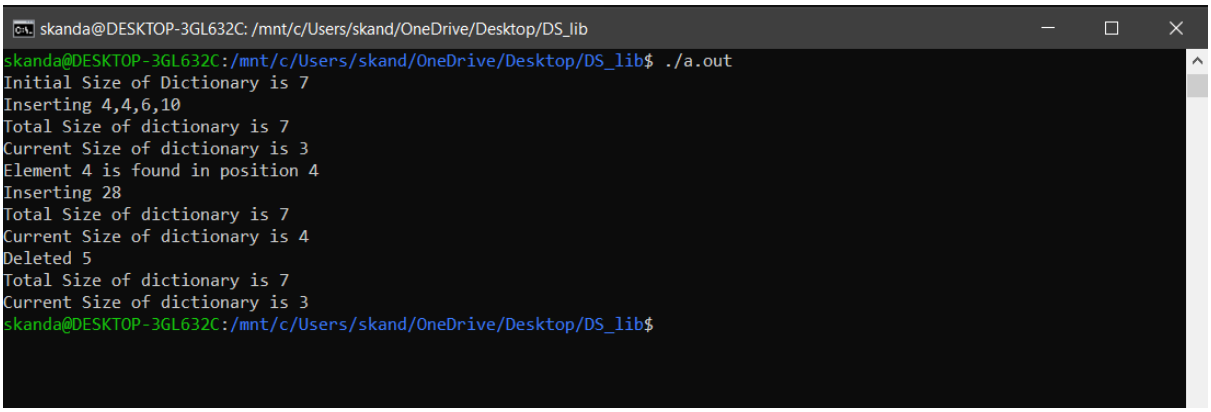

Fig 6: Showing output of library usage particularly for Dictionary of int type

- **Heap :**

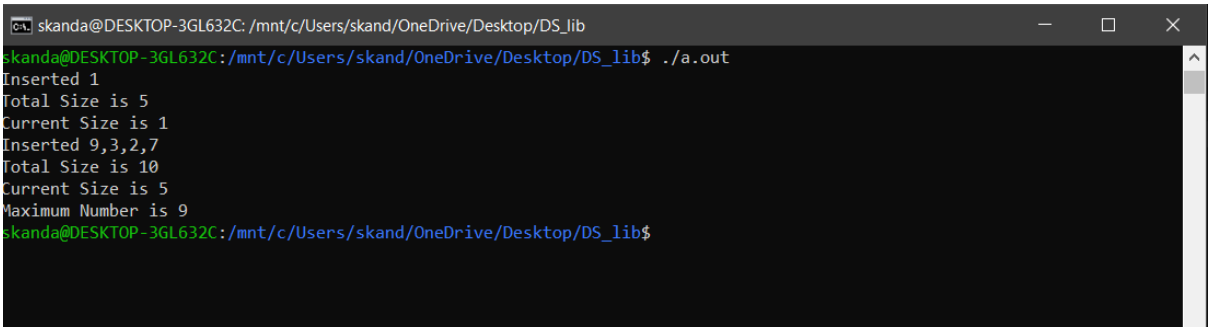

Fig 7: Showing output of library usage particularly for Max Heap of int type

- **Matrix :**

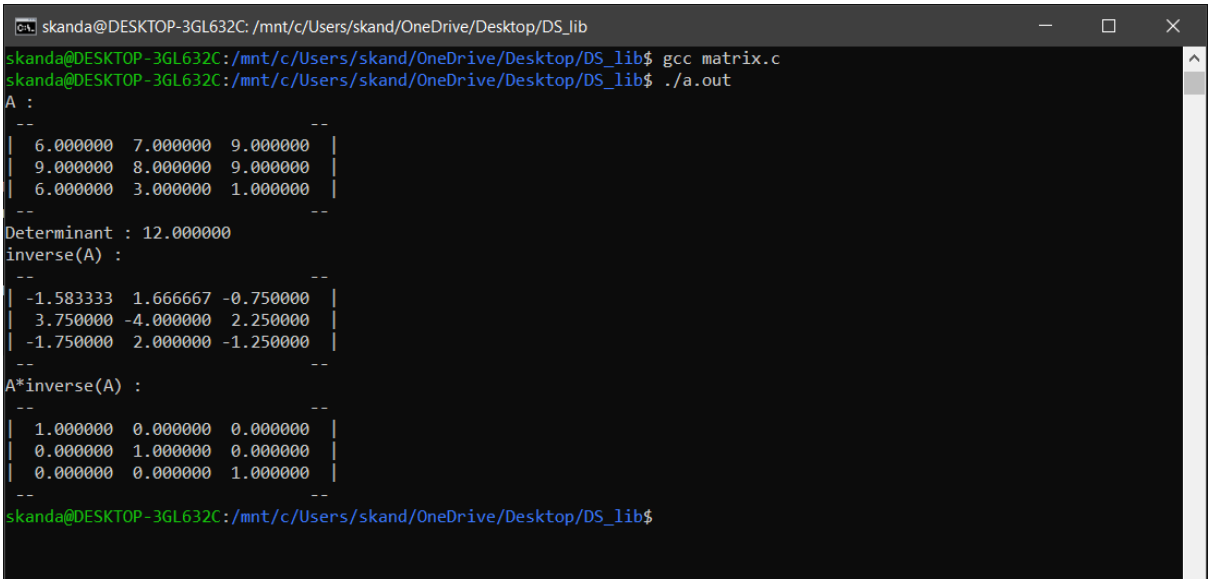

Fig 8: Showing output of library usage particularly for matrix of double type

- **AVL Tree :**

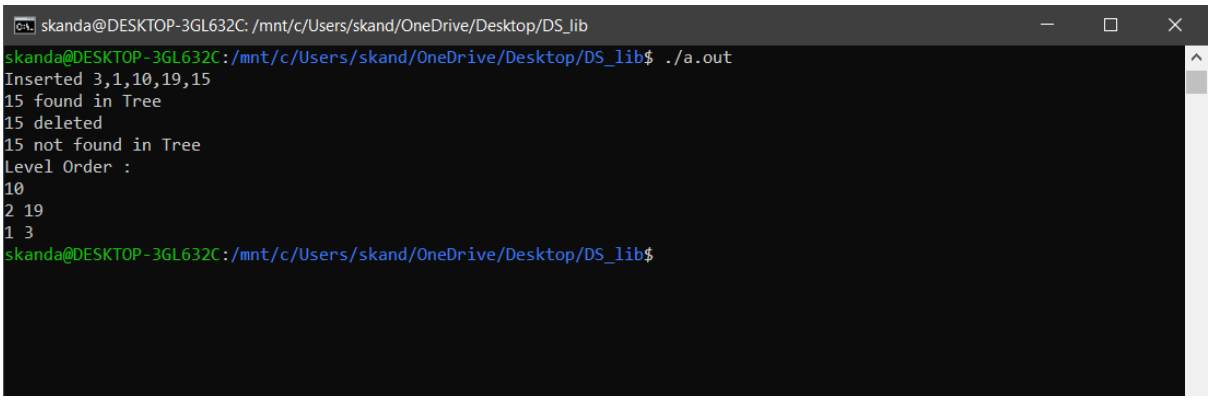

Fig 9: Showing output of library usage particularly for tree of int type