[CS302-Data Structures] Homework 5: Heap ADT using STL

Instructor: Kostas Alexis
Teaching Assistants: Tung Dang, Mustafa Solmaz
Fall 2019 Semester

Section 1. Heaps using STL

A Heap data structure can be efficiently implemented in a range using the C++ Standard Template Library (STL). STL is a massive software library for C++ that provides four components, namely a) algorithms, b) containers, c) functions, and d) iterators. STL provides a set of common classes for C++, such as containers and associative arrays that can be used with any built-in type and with any user-defined type that supports certain elementary operations (e.g., copying and assignment). STL algorithms are independent of containers, which then significantly reduces the complexity of the library.

STL achieves its results through the use of templates. This approach provides compile-time polymorphism that is often more efficient than traditional run-time polymorphism. Modern C++ compilers are tuned to minimize abstraction penalties arising from heavy use of STL.

Using the STL you can build and use a Heap efficiently. The whole process relies on the use of certain subsets of the library. There are certain operations to focus on, such as:

make_heap(): Rearranges the elements in the range [first,last) in such a way that they form a heap.
Detailed reference: https://tinyurl.com/3ykcorl

push_heap(): Given a heap in the range [first,last-1), this function extends the range considered a heap to [first,last) by placing the value in (last-1) into its corresponding location within it.
Detailed reference: https://tinyurl.com/ydbdq6wm

pop_heap(): Rearranges the elements in the heap range [first,last) in such a way that the part considered a heap is shortened by one: The element with the highest value is moved to (last-1).
Detailed reference: https://tinyurl.com/ycfcb2cr

sort_heap(): Sorts the elements in the heap range [first,last) into ascending order.
Detailed reference: https://tinyurl.com/yd763ywt

is_heap(): Returns true if the range [first,last) forms a heap, as if constructed with make_heap.
Detailed reference: https://tinyurl.com/y848x8wv

is_heap_until(): Returns an iterator to the first element in the range [first,last) which is not in a valid position if the range is considered a heap (as if constructed with make_heap).
Detailed reference: https://tinyurl.com/yaol9jau

An example use of STL to make a heap is shown below:

```
#include<iostream>
#include<algorithm> // for heap operations
using namespace std;
int main()
{

    // Initializing a vector
    vector<int> v1 = {10, 30, 100, 40, 20};

    // Converting vector into a heap
    // using make_heap()
    make_heap(v1.begin(), v1.end());

    // Displaying the maximum element of heap
    // using front()
    cout << "The maximum element of heap is : ";
    cout << v1.front() << endl;

    return 0;
}
```

Similarly, you may sort such a heap through:

```
#include<iostream>
#include<algorithm> // for heap operations
using namespace std;
int main()
{

    // Initializing a vector
    vector<int> v1 = {10, 30, 100, 40, 20};

    // Converting vector into a heap
    // using make_heap()
    make_heap(v1.begin(), v1.end());
```

```
    // Displaying the maximum element of heap
    // using front()
    cout << "The maximum element of heap is : ";
    cout << v1.front() << endl;

    // using push_back() to enter element
    // in vector
    v1.push_back(50);

    // using push_heap() to reorder elements
    push_heap(v1.begin(), v1.end());

    // Displaying the maximum element of heap
    // using front()
    cout << "The maximum element of heap after push is : ";
    cout << v1.front() << endl;

    // using pop_heap() to delete maximum element
    pop_heap(v1.begin(), v1.end());
    v1.pop_back();

    // Displaying the maximum element of heap
    // using front()
    cout << "The maximum element of heap after pop is : ";
    cout << v1.front() << endl;

    return 0;
}
```

Section 2. Exercise on using STL for Heaps

In this exercise, your goal is to familiarize yourself with respect to how to use the STL to implement and operate on a heap data structure.

Exercise 1. Using STL for Heaps

Utilize STL to

1. Make a heap consisting of 100 random integers.
2. Add a new value that is the mean of the random values you created in the previous step. Floor the value if needed.
3. Delete the maximum element of the heap and
4. Sort the heap.

Deliver code, a TXT file with the output of the terminal and a short report.