Justin Ciocoi

Oct. 5, 2023

CSCI 373 Class Notes

Advanced Data Structures and Algorithms

Circularly Linked Lists

- It basically uses the same node structure as a singly linked list
- However, the list is circular, which means the last node of the list points back at the first node
- If a circularly linked list contains **only one** element, the node will point to *itself*
- Travel across a circularly linked list can only be done unidirectionally
- A cursor node is defined to remember the current node that we are in
- We also define a front and back node, although we don't need to implement them in code
    - Since its a circle, the program doesn't really care which is the *front* or *back* so we use this only for our understanding as programmers
- **Functions in a circularly linked list (API)**
    - `back()`
        - Return element referenced by the cursor
        - Empty list returns error
    - `front()`
        - Return element immediately after the cursor
        - Empty list returns error
    - `advance()`
        - Advance cursor to next node in list
    - `add(e)`

        - Insert a new node immediately after the cursor
        - If the list is empty, the node becomes the cursor and its next pointer points to itself
    - `remove()`
        - Remove the node immediately after the cursor
        - If cursor is the only node, then it is removed and the cursor is set to `NULL`
    - `add()` and `remove()` operate on the circularly linked list in a *Last-in First-out* stack data structure where code such as:

    -
      ```
      add(x);
      remove();
      add(y);
      remove();
      add(n);
      add(m);
      remove();
      remove();
      ```
    - will result in no change in the original circularly linked list
- **Interface of the node in a circularly linked list**

    -
      ```
      typedef string Elem;
      class cNode
      {
          Elem elem;
          cNode* next;

          friend class CircleList;
      };
      ```
- **Interface of the circularly linked list**

-
  ```
  class CircleList
  {
      public:
          CircleList();                        //sets cursor to null
          ~CircleList();                       //remove nodes one by one
          bool empty() const;
          const Elem& front() const;
          const Elem& back() const;
          void advance();
          void add(const Elem& e);
          void remove();

      private:
          cNode** cursor;
  };
  ```