CSE 373: Queues

Chapter 3

Definition

Queue:
Queue Operations

Main Operations:

void enqueue(Object);
Object dequeue();
Object front(); // or getFront();
bool isEmpty();

Other Operations:
- normal creation/deletion operations
- again, generally no iteration operations

Queue Example

Queue Q;
int frontval, newval;

Q.enqueue(1);
Q.enqueue(1);
for (i=2; i<n; i++) {
    frontval = Q.dequeue();
    newval = frontval + Q.front();
    Q.enqueue(newval);
}
List-based Queue Implementation

- As with Stacks, Queues are a specialized List
  - `enqueue()` = `insert()` at a specific end of the list
  - `dequeue()` = `remove()` from the opposite end
- Thus, Lists could be used to implement the Queue ADT
  - Similar advantages and disadvantages as the Stack case

Array-Based Queue Implementation

Naive approach:
- `enqueue()` = insert at end of array
- `dequeue()` = delete from front of array

Running Time:
- `enqueue()`:
- `dequeue()`:

How could we improve this?
Link-Based Queue Implementation

What are the challenges to making a link-based
enqueue() and dequeue() efficient?

Evaluating Queue Implementations

Operations:
- enqueue()
- dequeue()
- front()
- isEmpty()

Space:

Other:
Applications of Queues

Anything where “fairness” (FIFO) is required
- operating systems: printer queues, storing user input, servers, scheduling processes
- compilers (and in general): worklists
- graphics: queue of things to render
- applications: list of recently used files
- real-life: lines at fast-food restaurants, “waiting for next available operator” lists
- searching: “breadth-first” searches

Introduction to Templates

The point:
- Lists, Stacks, and Queues are examples of ADTs that can store an arbitrary data type
  (e.g., List of integers, List of doubles, List of strings)
- The implementation of these ADTs’ operations is independent of the data type
  (e.g., insert() / delete() didn’t care which type of List)
- Templates support this separation of operation implementation and base type
Using Templates

declaration:  
template <class Object>
    class List {
    private:
        ListNode<Object> *head;
        ListNode<Object> *tail;
    };

use:  
    List<int> myIntList;
    List<double> myDbList;
    List<string> myStrList;

Compiling Templates

```cpp
#include "list.h"

List<int> myIntList;
List<double> myDbList;

void main() {
    ...
}
```

My Project

```cpp
My Project

main.cpp

(list.cpp)
```

UW, Autumn 1999  CSE 373 – Data Structures and Algorithms  Bind Chamberlain