CITS2200 Data Structures and Algorithms

Topic 6

Performance Analysis 1: Introduction

- Types of performance measurement
  - empirical
  - analytical
- An example of analytical analysis using Queue
- Introduction to growth rates

Reading: Weiss Chapter 5.

1. Types of Performance Measurement

Empirical measurement

We will see that the most efficient queue ADT to use depends on the program that uses it — which operations are used most often.

If we have access to the program(s), we may be able to measure the performance in those programs, on real data — called *evaluation in context*.

This is the "get yer hands dirty" approach. Run the system with real-world input and observe, or monitor (automatically), the results.

Can compare data structures on the same problems (same machine, same compiler, etc)

$\Rightarrow$ *benchmark* programs

- Useful if test input is close to expected input.
- Not much use if we are developing eg a library of modules for use in many different contexts

In some cases, it is not feasible to test a programme "in the field" (e.g. nuclear weapons systems). Here, we may construct a (computer) model of the system and evaluate performance with simulated data.

A computer program normally acts as its own model — run on simulated data (often generated using pseudo-random numbers).

However, a simplified model may be built or the program modified to fit the simulated data.

Advantages

- nondestructive
- cheap
- fast
- reproducible

Disadvantages

- only as good as the simulations
- can never be sure it matches reality

Analytical Measurement

Construct a mathematical or theoretical model — use theoretical techniques to estimate system performance.

Usually

- coarse estimates
- growth rates, complexity classes rather than 'actual' time
- *worst case* or *average case*

But. . . !

- *fundamental view of behaviour* — less susceptible to
  - speed of hardware, number of other processes running, etc
  - choice of data sets
  - unrepresentative examples, spurious responses
- gives a better understanding of the problems
  - why is it slow?
  - could it be improved?

We will concentrate on analytical analyses.

2. Example: A Basic Analysis of the Queue ADTs

As an example of comparison of ADT performance we consider different representations of queues using a crude time estimate

Simplifying assumptions:

- each high-level operation (arithmetic operation, Boolean operation, subscripting, assignment) takes 1 time unit
- conditional statement takes 1 time unit + time to evaluate Boolean expression + time taken by most time consuming alternative (*worst-case* assumption)
- field lookup ("dot" operation) takes 1 time unit
- method takes 1 (for the call) plus 1 for each argument (since each is an assignment)
- creating a new object (from a different class) takes T_c time units

2.1 Block Representation Queues (Without Wraparound)

```
public QueueBlock (int size) {                //2
  items = new Object[size];                      //1+Tc
  first = 0;                                     //1
  last = -1;                                     //1
}
```

$5 + T_c$ time units

```
public boolean isEmpty () {return first == last + 1;}
```

4 time units

```
public boolean isFull () {return last == items.length - 1;}
```

5 time units

```
public void enqueue (Object a) throws Overflow {       //2
  if (!isFull()) {                                   //7
    last++;                                          //1
    items[last] = a;                                 //2
  }
  else throw new Overflow("enqueuing to full queue");
}
```

12 time units

Exercise:

How many time units for each of the following...

```
public Object examine () throws Underflow {
  if (!isEmpty())  return items[first];
  else throw new Underflow("examining empty queue");
}

public Object dequeue() throws Underflow {
  if (!isEmpty()) {
    Object a = items[first];
    first++;
    return a;
  }
  else throw new Underflow("dequeuing from empty queue");
}
```


Summary for Block Implementation

isEmpty, enqueue, examine and dequeue are *constant time* operations

Queue is constant time *if* T_c is constant time

2.2 Recursive (Linked) Representation Queues

```
public QueueLinked () {
  first = null;
  last = null;
}
```

3 time units

```
public boolean isEmpty () {return first == null;}
```

3 time units

```
public void enqueue (Object a) {                    //2
  if (isEmpty()) {                                //4
    first = new Link(a,null);                   //1+Tc
    last = first;                                 //1
  }
  else {
    last.successor = new Link(a,null);          //2+Tc
    last = last.successor;                        //2
  }
}
```

$10 + T_c$ time units

```
public Object examine () throws Underflow {
  if (!isEmpty())  return first.item;
  else throw new Underflow("examining empty queue");
}
```

8 time units


```
public Object dequeue () throws Underflow {          //1
  if (!isEmpty()) {                                 //5
    Object c = first.item;                           //2
    first = first.successor;                        //2
    if (isEmpty())  last = null;                    //5
    return c;                                       //1
  }
  else throw new Underflow("dequeuing from empty queue");
}
```

16 time units

Summary for Linked Implementation

Again all are constant time, assuming T_c is.

Comparison...

	block	recursive
Queue	$5 + T_c$	3
isEmpty	4	3
enqueue	12	$10 + T_c$
examine	9	8
dequeue	11	16

...shows no clear winner, especially given

- estimates are very rough — many assumptions
- dependent on relative usage of operations in the programs calling the ADT — eg. is *isEmpty* used more or less than *dequeue*

We will generally not be interested in these "small" differences (eg 5 time units vs 3 time units) — given the assumptions made these are not very informative.

Rather we will be interested in *classifying* operations according to *rates of growth*...

3. Growth Rates

For comparative purposes, exact numbers are pretty irrelevant! It is the *rate of growth* that is important.

We will abstract away from inessential detail...

- ignore specific values of input and just consider the number of items, or "size" of input
- ignore precise duration of operations and consider the number of (specific) operations as an abstract measure of time
- ignore actual storage space occupied by data elements and consider the number of items stored as an abstract measure of space

4. Summary

Two main types of performance measurement — empirical and analytical.

We will concentrate on analytical:

- fundamental view of behaviour
- abstracts away from machine, data sets, etc
- helps in understanding data structures and their implementations

Rather than attempting 'fine grained' analysis that compares small differences, we will concentrate on a coarser (but more robust) analysis in terms of *rates of growth*.