Data structures and algorithms

Ravi Varadarajan

Primary goals of the course

- Learn the commonly used data structures and algorithms.

  --These form a programmer's basic "toolkit".

- Understand how to measure the cost of a data structure or program in terms of use of computing resources

  --- These measures enable one to judge the merits of a variety of data structure options.

- Reinforce the concept that cost-benefit tradeoffs exist for every data structure and use them wisely in choice of data structures to solve a problem efficiently.

Course expectations

- Course builds on basic Java programming skills

- At the end of the course, you should be able to handle

  (a) theoretical analysis of program efficiency

  (b) practical application of data structures in writing efficient Java programs.

About me…

- Did my PhD in CS at UPenn
- Did teaching and research at UF. Supervised a few PhD and Master's theses.
- Undertook major software projects in industry (C++, Java, Scala, Python)
- Like to read, listen to music, watch mystery shows, avid tennis fan.

Terminology

- **Problem** – Task to be performed i.e. produce outputs given some inputs (e.g. sorting)

  Can be considered as mathematical function F(x) where x is input

- **Problem size** – Size of inputs/outputs (e.g. # of elements to be sorted)
- **Algorithm** – A method to solve the problem in question. Step-by-step procedure ("recipe") to solve the problem.

  -- Many possible algorithms to solve same problem

- **Program** – Algorithm implementation in a programming language (e.g. Java, C++)
- **Choice of algorithm** – Depends on resource constraints of a computer (e.g. CPU, Memory) and how big the problem size is expected to grow.

Need for data structures

- Data structures organize data ⇒ more efficient programs and programming.
- Choice of data structure or algorithm can make the difference between a program running in seconds or days.
- An efficient algorithm minimizes its requirements of Space and Time
- The cost of a solution is quantified in terms of resources that the solution consumes.
- Programmer efficiency – Not having to reinvent the wheel by use of known data structures!!

Selecting a data structure

- Analyze the algorithm to determine basic data operations that must be supported
- Quantify the resource constraints (time, space) for these operations
- Select a proper data structure that meets these requirements
- Cost-benefit tradeoffs include factors such as time, space and programming effort
- No data structure is best in all situations.
- Requires careful analysis of problem and algorithm characteristics in terms of time/space requirements.

Example

- Online order fulfillment system for a hardware company
- **Requirements** :

1. **Item search** : Users should be able to search for a part using key words.
2. **Persistent shopping cart** : Users should be able to maintain shopping carts where they add items to be purchases in a single order. This shopping cart should exist even if they log out.

Need to store this on disk and should be able to retrieve it in real time

Example (contd.)

- **Requirements** (contd.)

3. **Check out orders**: If users have credit/debit card info stored, should be able to retrieve this info for check out process.

Need to store this info on disk and should be able to retrieve it in real time

4. **Order search** : Users should be able search for their orders to check for status.

All the four operations need to have a user response time < 5 seconds.

Number of users ~ 100000

Abstract data type (ADT)

- Represents a data structure as possible set of values and the allowable operations that are permitted on these data values.
- Examples :

(a) Integer data type in Java. Contains all integer values between Integer.MIN_VALUE (-2^{31}) and Integer.MAX_VALUE($2^{31} - 1$). Allowable operations include +, -, *, /, %, <, <=, >, >=, == etc.

(b) String data type in Java. Contains all strings of Unicode (16-bit) characters, including empty string " ". Allowable operations include + (concatenation), substring(), indexOf(),replace(), equals() etc.

ADT implementation

- ADT is an interface that specifies only the inputs and output for each allowable operation but does not specify how it is implemented.
- Data structure is implementation of ADT
- It encapsulates data and hides operational details.
- Integer, String data types in Java have only one implementation.
- Some ADTs (e.g. list) have many possible implementations; choice of an implementation is guided by algorithm time/space requirements.