Data Structures and Algorithms
CS 213- Lecture 1

RK Shyamasundar

Broad Topics

- Introduction to data structures,
  - Abstract data types, analysis of algorithms.
- Creation and manipulation of data structures: arrays, matrices, sparse matrices, lists, stacks, queues, trees, heaps, hash tables, balanced trees, tries, graphs.
- Algorithms for sorting and searching,
- Depth-first and breadth-first search,
- Paradigms:
  - The greedy method.
  - Divide-and-conquer.
  - Dynamic programming.
  - Backtracking.
  - Branch-and-bound.

Text Books

- S. Sahni, Data Structures, Algorithms and Applications in C++,2nd edition, Universities Press,2005
- T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, Prentice-Hall India, 2001.

Organization

- Lectures – Tuesday, Friday: 330-500PM
- Quizes 3-- 30%
- Mid Term – 30%
- Final Exam – 40%
- Attendance Necessary
- Academic Honesty Policy:

http://www1.iitb.ac.in/newacadhome/rules.jsp

TAs

BS Radhika

Amit Goel

Sundaram Gupta

- MTP-1 --6 TAs??
- BTP – 3 TAs??

Tentative Agenda

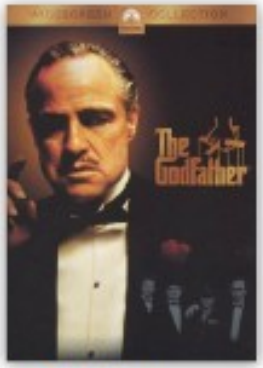
- Introduction
  - Role of Data structures
  - Concerns of program Construction
  - Analysis of Algorithms: Notations
  - Complexity Analysis forms
  - Recurrence Relations/Recursive Programs
- Merge Sort, asymptotic analysis and performance measurement of programs.
- Data representation methods and linear lists
- Arrays, matrices, sparse matrices
- Stacks.
- Queues.
- Hashing
- LZW compression.
- Binary trees, AVL Trees

- Priority queues.
- Tournament trees.
- Search trees.
- Graphs.
- The greedy method.
- Divide-and-conquer.
- Dynamic programming.
- Backtracking.
- Branch-and-bound.

What is it about?

- Data structures: Representation and manipulation of data.
- Programs manipulate data.
- Programs as data
- How do we manipulate Data?

Digital Data



Movies

Music

Photos

Protein Shapes

Maps

DNA

```
gatottttta tttaaaogat otottt atta gatotottat taggatoatg atoototgtg
gataagtgat tattoaoatg goagatoata taattaagga ggatogtttg ttgtgagtga
ooggtgatog tattgogtat aagotgggat otaaatggoa tgttatgoao agtoaotogg
oagaatoaag gttgttatgt ggatatotao tggttttaoo otgottttaa goatagttat
aoaoattogt togogogato tttgagotaa ttagagtaaa ttaatooaat otttgaoooa
```

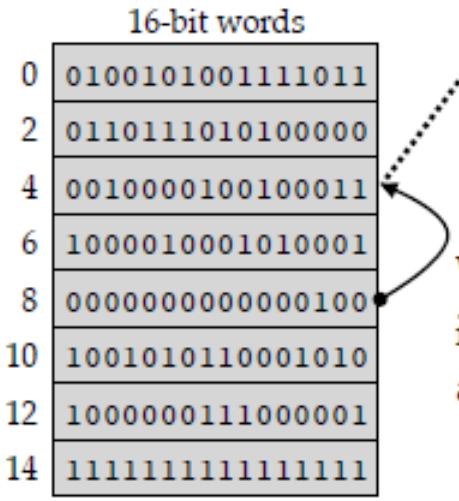
00101010010101010101001001001010100000100100101 00....

RAM = Symbols + Pointers

(for our purposes)

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	SP
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)

ASCII table: agreement for the meaning of bits



We may agree to interpret bits as an address (pointer)

Physically, RAM is a random accessible array of bits.

=> We can store and manipulate arbitrary symbols (like letters) and associations between them.

Digital Data Must Be ...

- Encoded (e.g. 01001001 <-> )

- Arranged
  - Stored in an orderly way in memory / disk

- Accessed
  - Insert new data
  - Remove old data
  - Find data matching some condition

} The focus of this class

- Processed
  - Algorithms: shortest path, minimum cut, FFT, ...

Data Structure Example Applications

1. How does Google quickly find web pages that contain a search term?
2. What's the fastest way to broadcast a message to a network of computers?
3. How can a subsequence of DNA be quickly found within the genome?
4. How does your operating system track which memory (disk or RAM) is free?
5. In the game Half-Life, how can the computer determine which parts of the scene are visible?

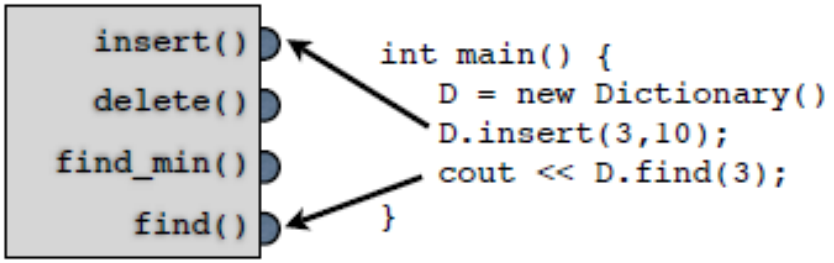
What is a Data Structure Anyway?

- It's an agreement about:
  - how to store a collection of objects in memory,
  - what operations we can perform on that data,
  - the algorithms for those operations, and
  - how time and space efficient those algorithms are.

- Ex. `vector` in C++:
  - Stores objects sequentially in memory
  - Can access, change, insert or delete objects
  - Algorithms for insert & delete will shift items as needed
  - Space: O(n), Access/change = O(1), Insert/delete = O(n)

Abstract Data Types (ADT)

```
class Dictionary {
   Dictionary();
   void insert(int x, int y);
   void delete(int x);
   ...
}
```



- Data storage & operations encapsulated by an ADT.
- ADT specifies permitted **operations** as well as **time** and **space** guarantees.
- User unconcerned with how it's implemented
  (but we are concerned with implementation in this class).
- ADT is a **concept** or **convention**:
  - not something that directly appears in your code
  - programming language may provide support for communicating ADT to users (e.g. classes in Java & C++)

Dictionary ADT

- Most basic and most useful ADT:
  - `insert(key, value)`
  - `delete(key, value)`
  - `value = find(key)`

- Many languages have it built in:

awk: `D["AAPL"] = 130` # associative array

perl: `my %D; $D["AAPL"] = 130;` # hash

python: `D = {}; D["AAPL"] = 130` # dictionary

C++:
```
map<string,string> D = new map<string, string>();
D["AAPL"] = 130;                                  // map
```

- **Insert**, **delete**, **find** each either O(log n) [C++] or expected constant [perl, python]

C++ STL

- STL (Standard Template Library) is a powerful set of C++ template classes to provides general-purpose templatized classes and functions that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks.

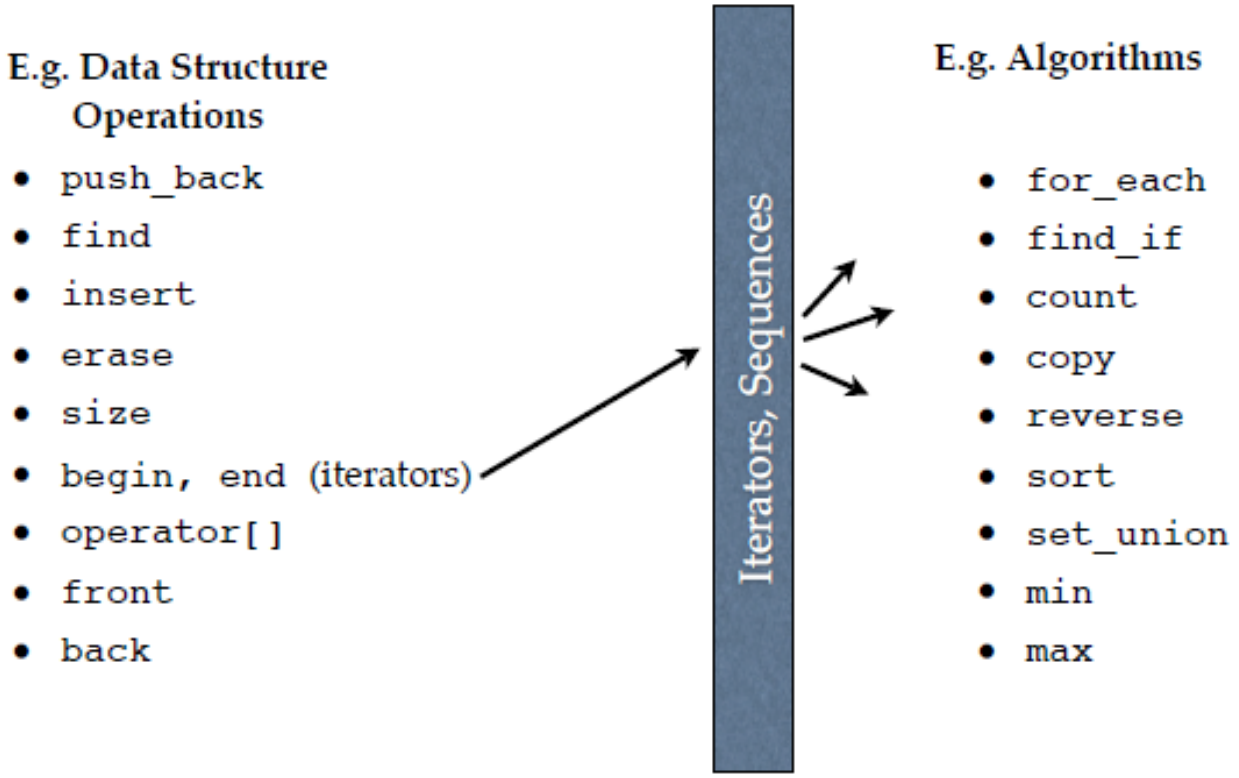
C++ STL

- Data structures = "containers"
- Interface specifies both operations & time guarantees

Container	Element Access	Insert / Delete	Iterator Patterns
`vector`	const	O(n)	`Random`
`list`	O(n)	const	`Bidirectional`
`stack`	const (limited)	O(n)	`Front`
`queue`	const (limited)	O(n)	`Front, Back`
`deque`	const	O(n), const @ ends	`Random`
`map`	O(log n)	O(log n)	`Bidirectional`
`set`	O(log n)	O(log n)	`Bidirectional`
`string`	const	O(n)	`Bidirectional`
`array`	const	O(n)	`Random`
`valarray`	const	O(n)	`Random`
`bitset`	const	O(n)	`Random`

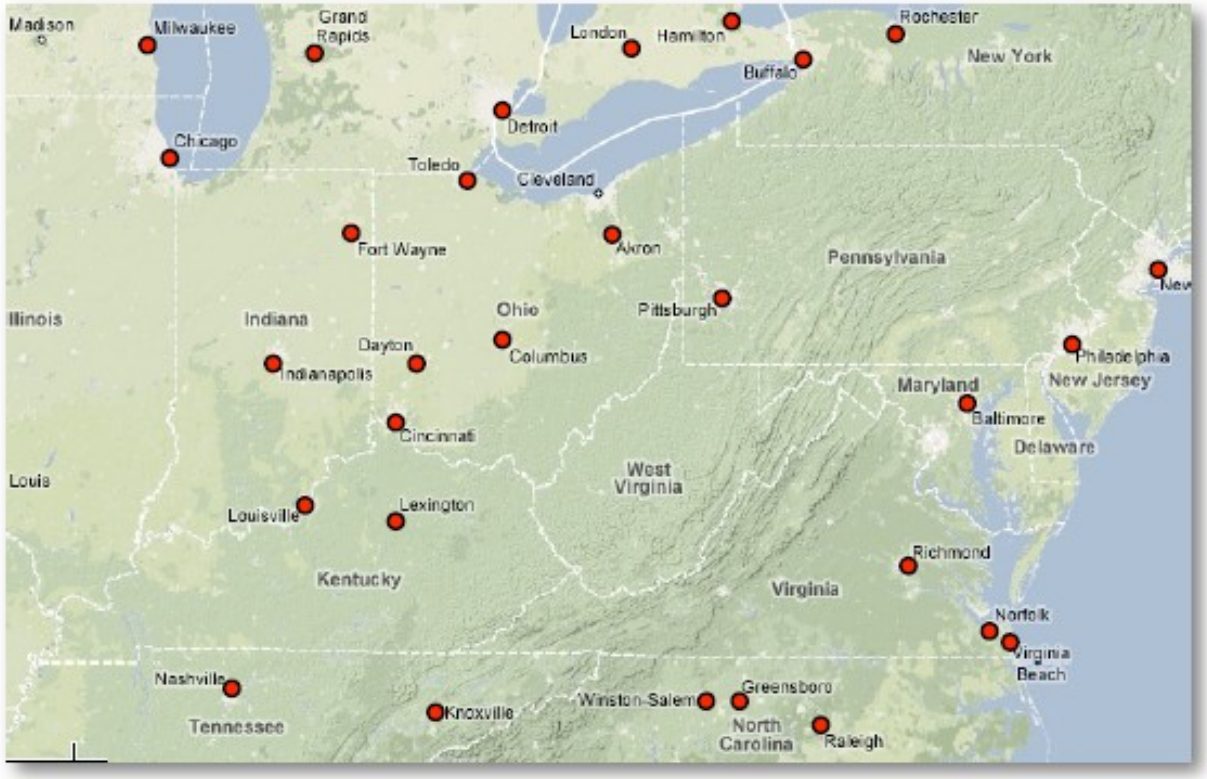
Some STL Operations

- Select operations to be *orthogonal*: they don't significantly duplicate each other's functionality.
- Choose operations to be useful building blocks.



Consider Google Maps

You want to store data about cities (location, elevation, population)...



What kind of operations should your data structure(s) support?

Operations to support these Operations

- Finding addresses on map?
  - Lookup city by name...
- Mobile iPhone user?
  - Find nearest point to me...
- Car GPS system?
  - Calculate shortest-path between cities...
  - Show cities within a given window...
- Political revolution?
  - Insert, delete, rename cities

Data Organizing Principles

- **Ordering**:
  - Put keys into some order so that we know something about where each key is are relative to the other keys.
  - Phone books are easier to search because they are alphabetized.
- **Linking**:
  - Add pointers to each record so that we can find related records quickly.
  - E.g. The index in the back of book provides links from words to the pages on which they appear.
- **Partitioning**:
  - Divide the records into 2 or more groups, each group sharing a particular property.
  - E.g. Multi-volume encyclopedias (Aa-Be, W-Z)
  - E.g. Folders on your hard drive

Ordering

Pheasant,	10
Grouse,	89
Quail,	55
Pelican,	3
Partridge,	32
Duck,	18
Woodpecker,	50
Robin,	89
Cardinal,	102
Eagle,	43
Chicken,	7
Pigeon,	201
Swan,	57
Loon,	213
Turkey,	99
Albatross,	0
Ptarmigan,	22
Finch,	38
Bluejay,	24
Heron,	70
Egret,	88
Goose,	67

Sequential Search – O(n)

Albatross,	0	
Bluejay,	24	
Cardinal,	102	
Chicken,	7	
Duck,	18	(2)
Eagle,	43	
Egret,	88	(3)
Finch,	38	
Goose,	67	(4)
Grouse,	89	
Heron,	70	(1)
Loon,	213	
Partridge,	32	
Pelican,	3	
Pheasant,	10	
Pigeon,	201	
Ptarmigan,	22	
Quail,	55	
Robin,	89	
Swan,	57	
Turkey,	99	
Woodpecker,	50	

Search for "Goose"

Binary Search
O(log n)

Every step discards half the remaining entries:

$n/2^k = 1$

$2^k = n$

$k = \log n$

Paradigms

- Algorithms + Data Structures = Programs
  - Imperative Programs

- Logics + Control = Programs
  - Logic Programs – Declarative Specifications
    - Example Finding a transitive closure of a

Broad Coverage

- Algorithm design methods needed to develop programs that do the data manipulation.

- Study of data structures and algorithms: Crux of Computer Science.

Programs

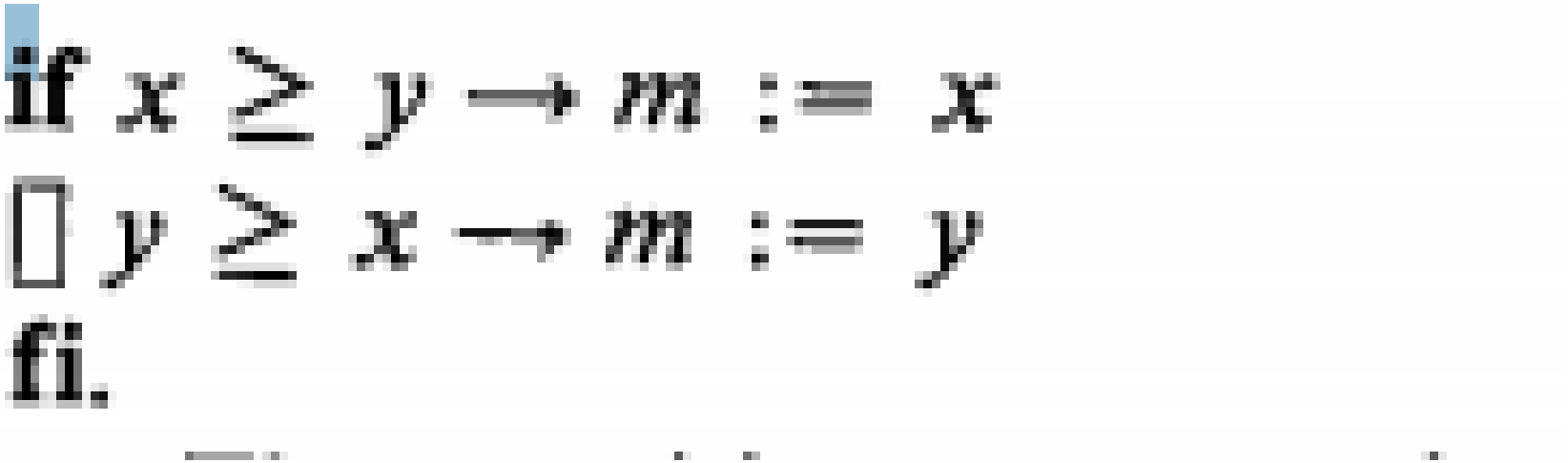
Two Concerns

- Correctness
- Efficiency

Writing an convincing program

- Three arrays
- F[i], G[j], H[k] monotonically decreasing
- Given that there exists a i,j, k such that F[i] = G[j] = H[k], write program computing the points of common intersection

A Simple Example



Permutation

$q1, q2, q3, q4 := Q1, Q2, Q3, Q4;$
do $q1 > q2 \rightarrow q1, q2 := q2, q1$
$\square \quad q2 > q3 \rightarrow q2, q3 := q3, q2$
$\square \quad q3 > q4 \rightarrow q3, q4 := q4, q3$
od.

Another Example

$k := 0;\ j := 1;$
do $j \neq n \rightarrow$ **if** $f(j) \leq f(k) \rightarrow j := j + 1$
$\square\ f(j) \geq f(k) \rightarrow k := j;\ j := j + 1$
fi
od.

GCD

$x := X; y := Y;$

do $x > y \rightarrow x := x - y$

□ $y > x \rightarrow y := y - x$

od.

GCD (Classical versions)

$x := X; y := Y;$ (version A)
while $x \neq y$ **do if** $x > y$ **then** $x := x - y$
else $y := y - x$ **fi od**

and

$x := X; y := Y;$ (version B)
while $x \neq y$ **do while** $x > y$ **do** $x := x - y$ **od**;
while $y > x$ **do** $y := y - x$ **od**
od.

What about computation of LCM?

Ackerman Function

- A(m,n) = n+1 if m=0
- = A (m-1,1), m > 0 and n=0
- = A (m-1, A(m, n-1)), m > 0 n > 0
-

Tower of Hanoi