COMP 250 - Lecture 23

Linear data structures: "What to use when"

- Lists (single, double-link)
- Stacks
- Queues
- Arrays

"Collections" (Java) - Abstract List

List Interface (generic types)

List<T>

↑ generic type

Implementation:
- Linked List (single)
- Array List
- Vector

Linked List

Array List

add(e) or remove(e)

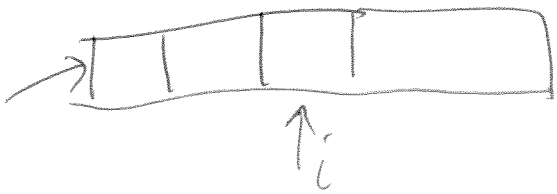
adding in n-th position

O(1)

Adding at front:
- shift content O(n)
- put the new element O(1)

= O(n)

Advantage of ArrayList: random access
Any element (index i) accessed in O(1)



start address + i * length of element

LinkedList: follow references => O(n)
Setting ith element to some value: same
Reaching max capacity of ArrayList:
↳ allocate more memory } -> O(n)
↳ copy content

ArrayList vs Vector

Both use array "under the hood"
Vector has some older methods
Single thread
2-4 "cores" common
Graphics processing units (GPUs)
Code -> split it into "Threads"
Threads execute in parallel (1 thread/processor maybe); fairly independent, but sometimes they need to interact with each other.

Synchronization may be required

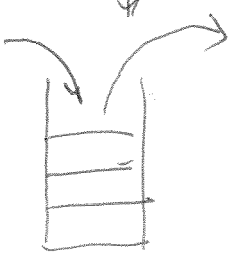
- Eg. multiple players accessing same resource
- "Spawn off" computation ⇒ returns some result
- Lock some resource for some time

ArrayList → synchronization is not automatic
Vector → synchronized automatically
Synchronization slows things down (a bit)

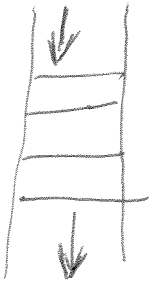
Thread is a class ⇒ run()

Locking - specifically gives control to one thread on shared structures
(also slows things down)

Stacks vs Queues



LIFO
(Last In First Out)
Execution stack
"Undo" features



First In First Out (& FIFO)
Multiple jobs that need to be processed "fairly"

what if First In First Out is not enough?

→ priority of each job

~~FIFO~~ → highest priority goes first

Priority Queue

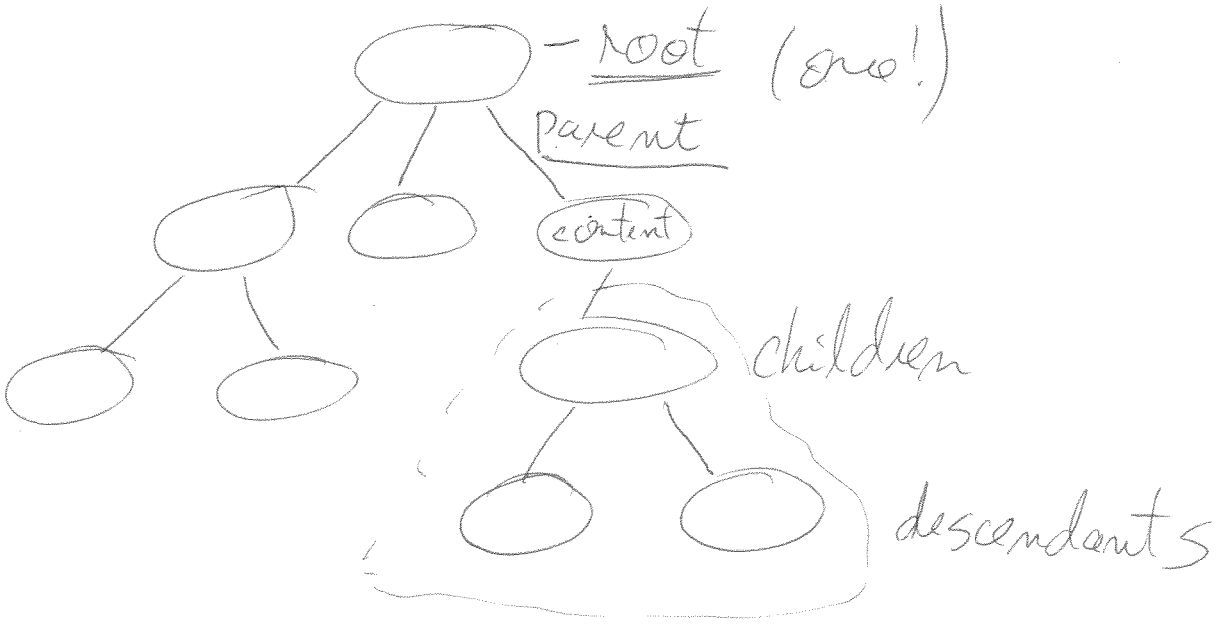
| Content | Priority (int) |

may want it to change with time

- Sorted array or list (sorted in order of priority)
  Inserting a new element is expensive (need to do it in right position)
  If priorities change, queue may need to be rearranged

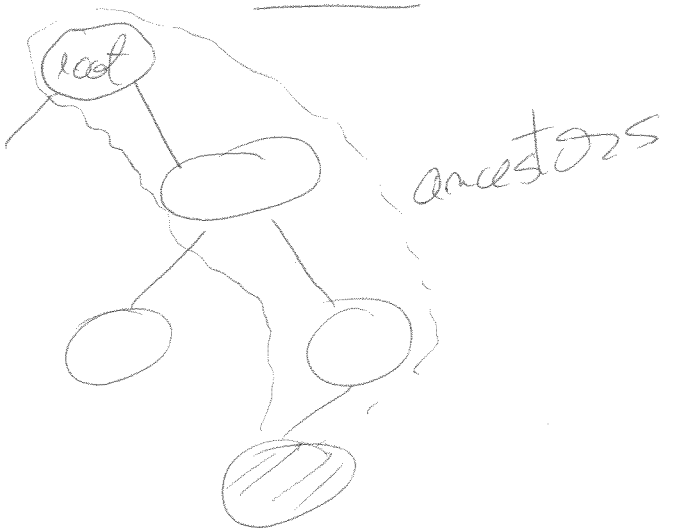
→ Heap: better data structure for priority queues (makes insert faster)

Trees: "non-linear" data structure



In general: each node in the tree has

- 1 parent
- 0 to ... children
- Descendents: all nodes below
- Ancestors: all nodes <u>above</u> (until the root)

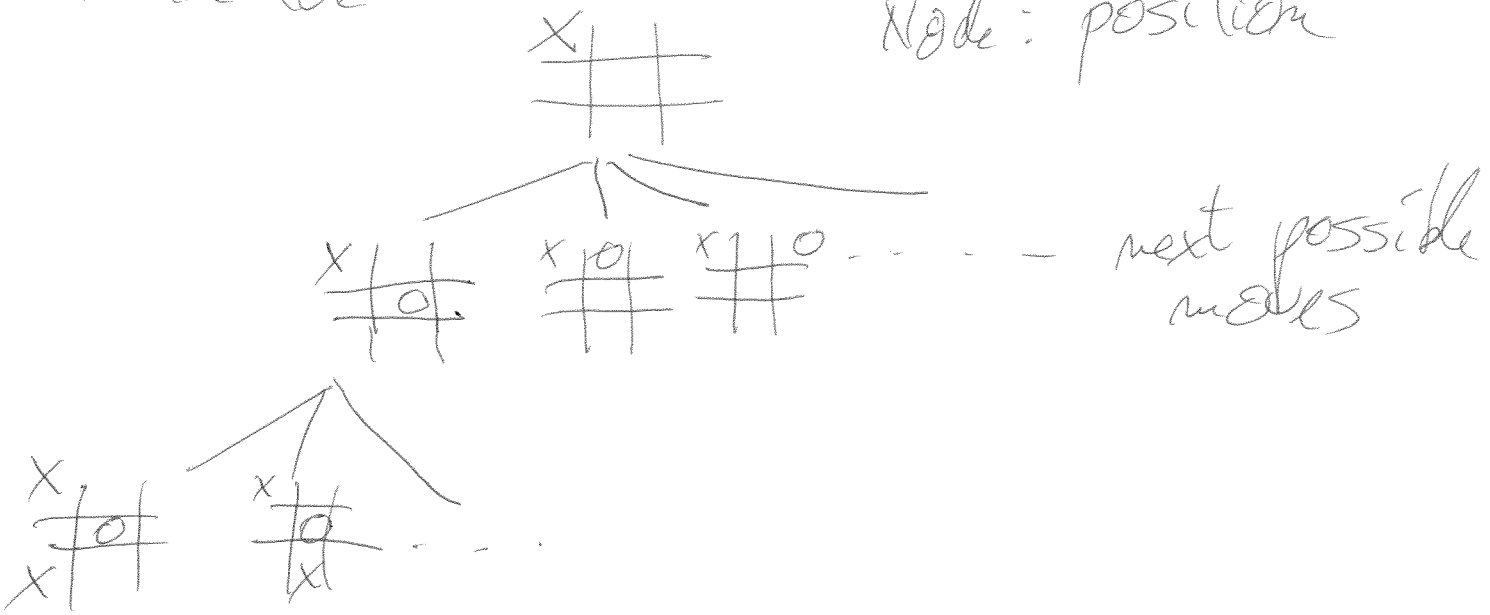


In general trees: can have as many children as wanted

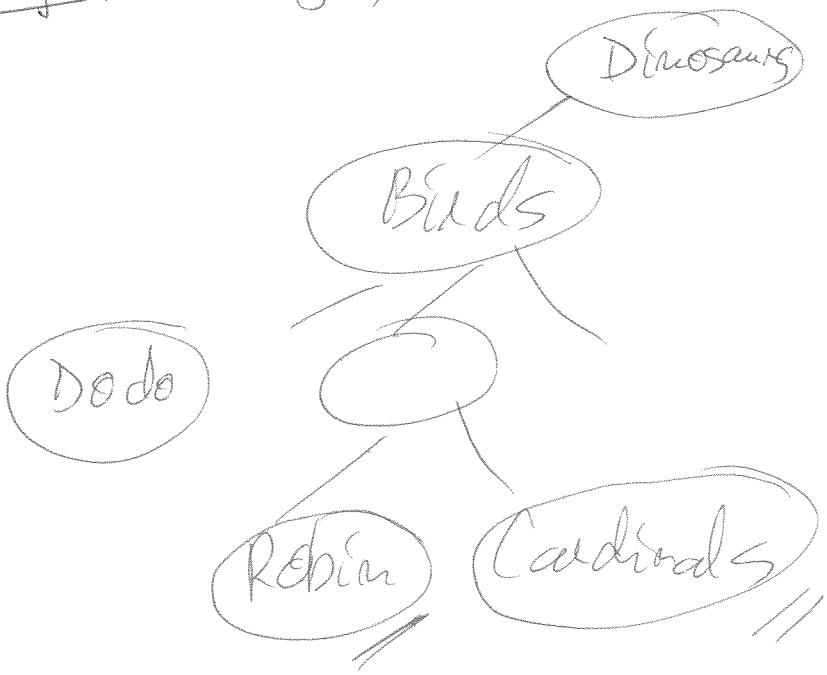
Binary trees: up to 2 children per node

E.g. Game tree

Tic tac toe



Eg. Philogeny tree



Eg: Expression tree (2+3)*5