CS32: Introduction to Computer Science II

Discussion Week 10

Yichao (Joey)

June 5, 2020

Credit: Carey Nachenberg, Junheng Hao

Announcements

- **Final exam is scheduled on June 6 (tomorrow)!**

Outline Today

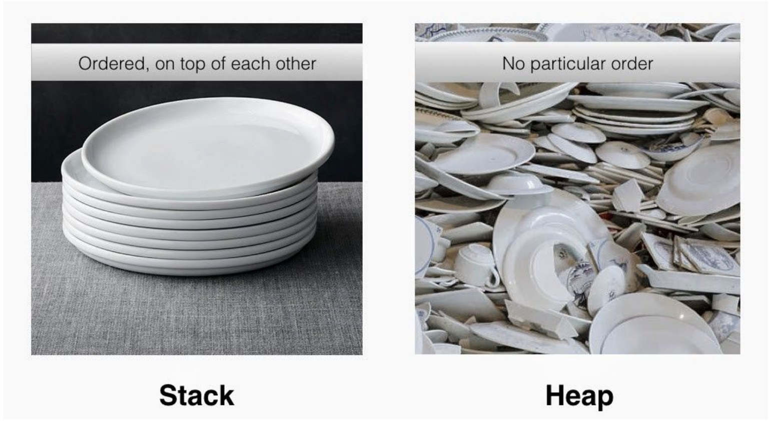
UCLA Samueli Computer Science

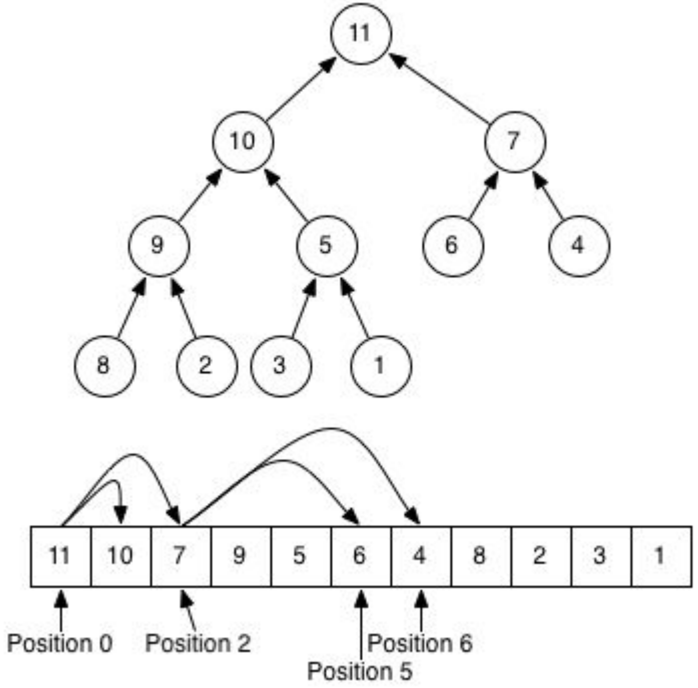
- Heap
- Priority queue
- Graph (not in final exam)
- Final exam review
- Q&A

Heap

Definition and properties

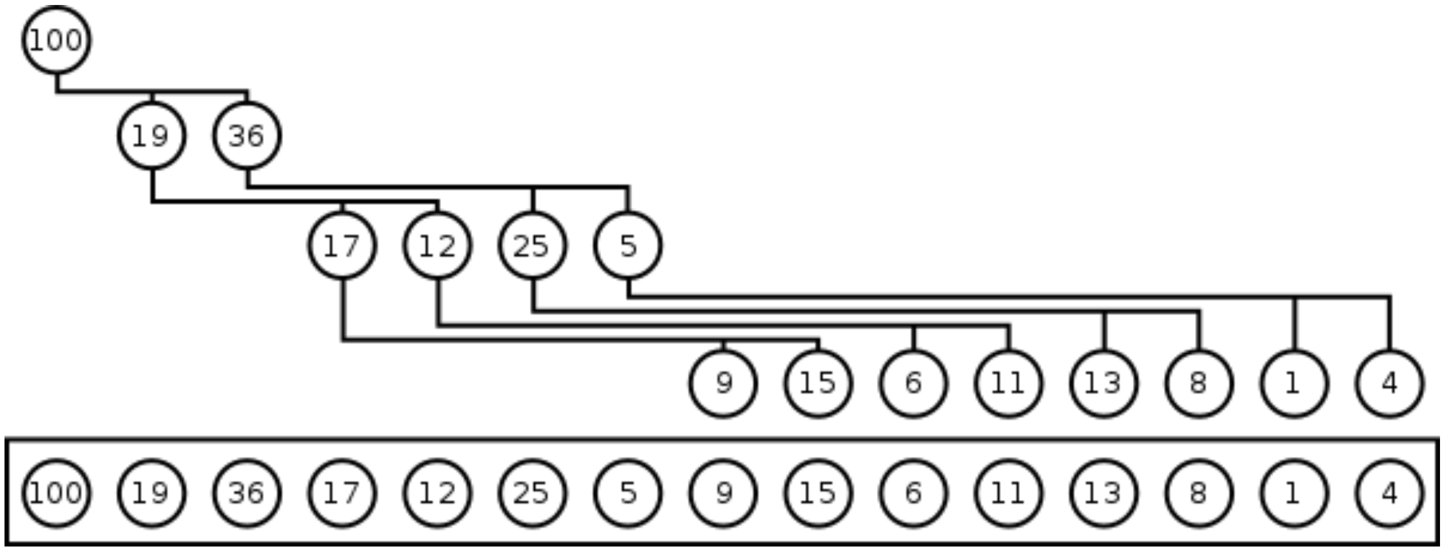
- About heap
  - Heap is considered as complete binary tree.
  - Every nodes carries a value greater than or equal to its children (for MaxHeap).
  - Often implemented as an array.
  - Body structure of priority queue.





Heap

Implementation by arrays



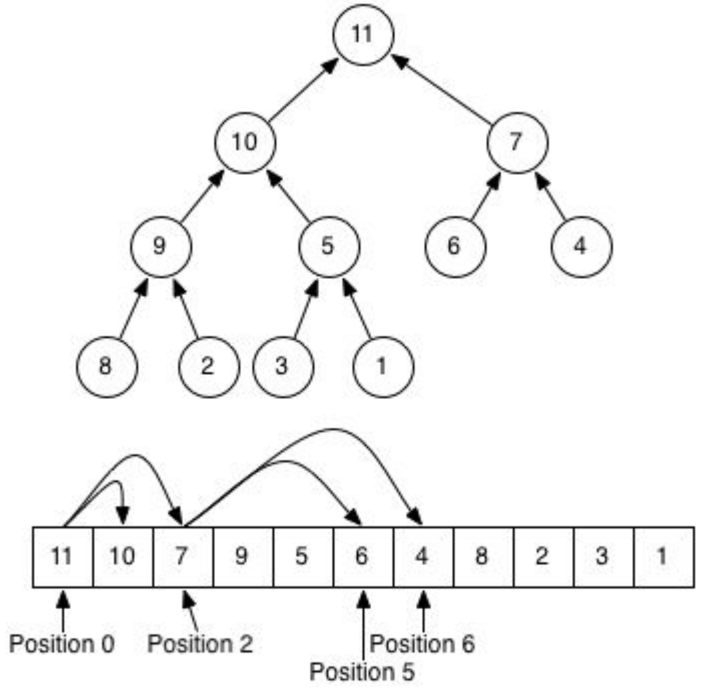
(Almost) full binary tree in an array implementation.

Heap

Standard operations

- Three operations of heaps
  - Find Max (search)
  - Insert Node (insert)
  - Delete Max (delete)

- How to implement `FindMax()` function of a heap?
  - Well, that is just too obvious!



Heap & Heapsort

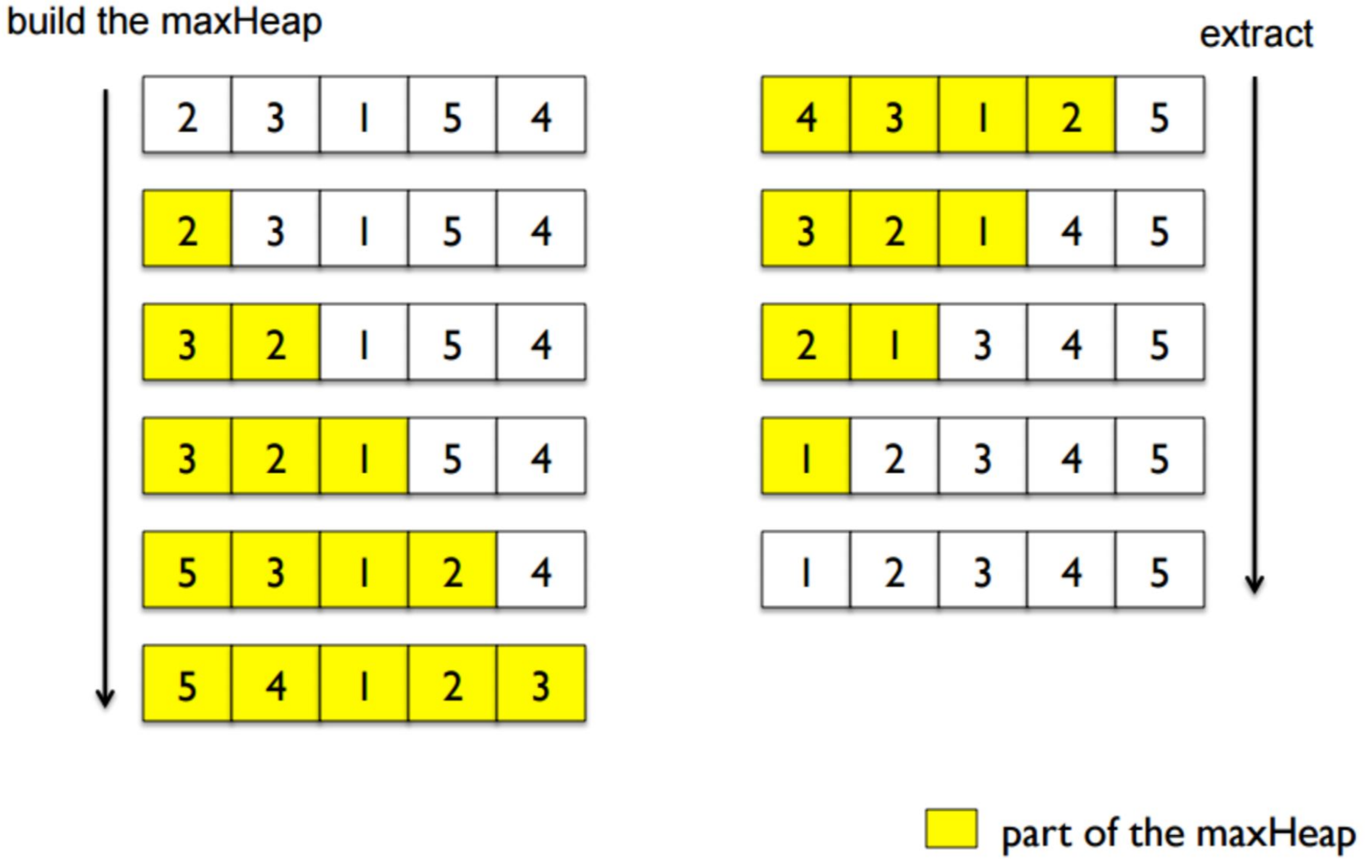
Complexity of heap operations

- Find Max → O(1)
- Insert → O(log *n*)
- Delete Max Node → O(log *n*)

6 5 3 1 8 7 2 4

- Bonus: How can you sort based on heap?
  - Insert all elements into a heap.
  - Extract the maximum element from the heap one by one.
- What is the complexity of heapsort?
  - $O(n \log n)$

In-place Heapsort (with an array)



Heapsort Problem 1

Find k largest numbers

UCLA Samueli Computer Science

- How can we efficiently find ***k*** **largest numbers** from n numbers? (n>>k, but k is not small)
    - Sort? → O(*n*log*n*)
    - Scan *k* times by linear search? → O(*nk*)

- Use heapsort
    - Only keep the largest
    - Whether to use MaxHeap or MinHeap?
    - Overall complexity?

Min Heap!
Pop out the `min` from heap and insert a number, if it's larger than `min`.

O(*n*log*k*)

Heapsort Problem 2

Find median from a streaming data

UCLA Samueli Computer Science

How to find the median of the streaming data?

That is, implementing the following program:

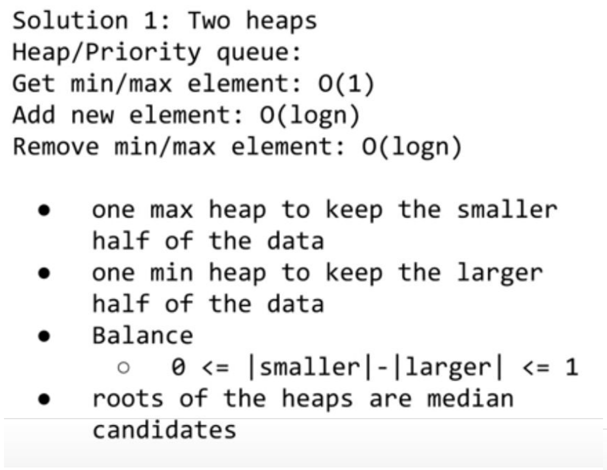
```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

Here you may use two heaps to store all the coming data → find median in O(1) time.

```
class MedianFinder {
public:
  MedianFinder() {
    // construction
  }

  void addNum(int num) {
    // add new integers from stream
  }

  double findMedian() {
    // return the median
  }
private:
    // define your private data member(s)
};
```

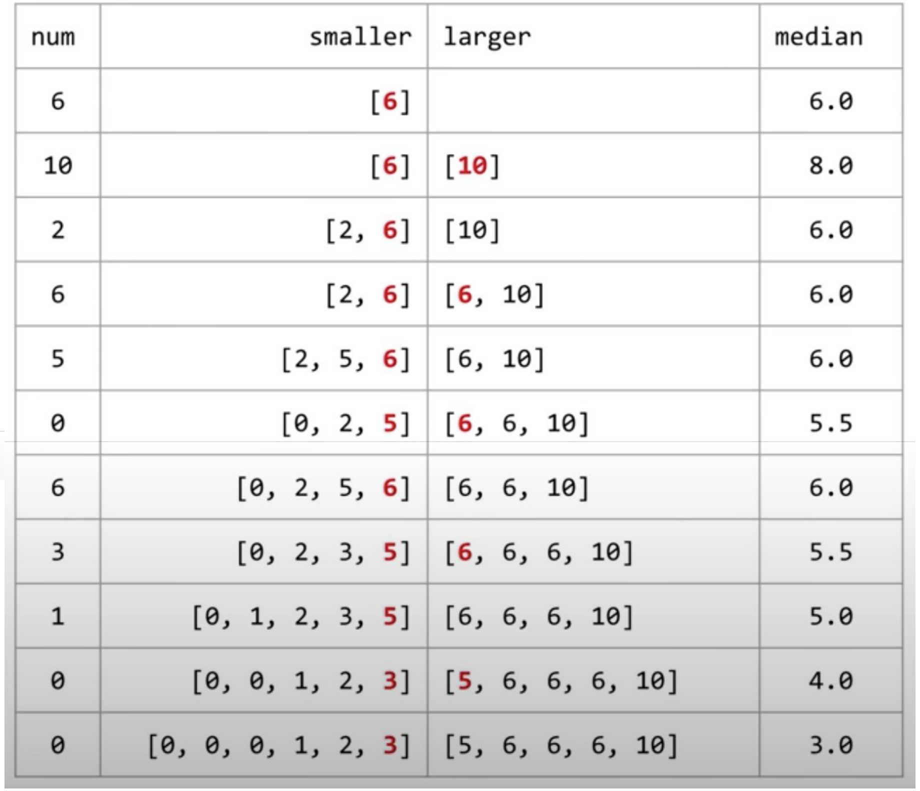
Heapsort Problem 2

Find median from a streaming data



Solution 1: Two heaps
Heap/Priority queue:
Get min/max element: O(1)
Add new element: O(logn)
Remove min/max element: O(logn)

- one max heap to keep the smaller half of the data
- one min heap to keep the larger half of the data
- Balance
  - 0 <= |smaller|-|larger| <= 1
- roots of the heaps are median candidates



num	smaller	larger	median
6	[6]		6.0
10	[6]	[10]	8.0
2	[2, 6]	[10]	6.0
6	[2, 6]	[6, 10]	6.0
5	[2, 5, 6]	[6, 10]	6.0
0	[0, 2, 5]	[6, 6, 10]	5.5
6	[0, 2, 5, 6]	[6, 6, 10]	6.0
3	[0, 2, 3, 5]	[6, 6, 6, 10]	5.5
1	[0, 1, 2, 3, 5]	[6, 6, 6, 10]	5.0
0	[0, 0, 1, 2, 3]	[5, 6, 6, 6, 10]	4.0
0	[0, 0, 0, 1, 2, 3]	[5, 6, 6, 6, 10]	3.0

Heapsort Problem 3

Merge *k* linked list

UCLA Samueli Computer Science

- How to merge k sorted linked lists? Each list has n nodes.
- Solution 1: Brute forth $\rightarrow O(nk^2)$
  - Keep linear searching the k heads and fetching the smallest until all lists are empty
- Solution 2: Use MinHeap $\rightarrow O(nk \log k)$
  - Insert the head of each list to the heap.
  - Each time we pop-out a node from the heap and append it to the result list, insert the next node of that node from its list to the heap (until heap is empty)
- Solution 3: Merge sort $\rightarrow O(nk \log k)$
  - Merge each pair of sorted lists. k sorted lists become k/2 sorted lists.
  - Repeat the list merge from k/2 to k/4… (until everything is merged into 1 list.)

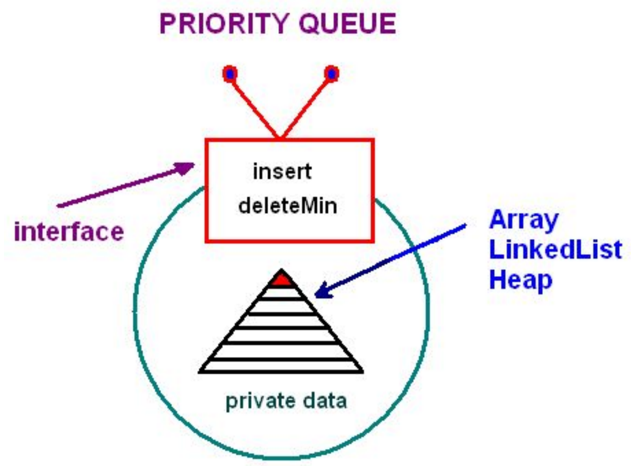
Priority Queue

Defininations & Properties

UCLA Samueli Computer Science

- Abstract data type (providing interface)
- Heap-based Priority Queue over other implementations:

Implementations	**Insert**	**DeleteMin**	**FindMin**
Ordered array	O(n)	O(1)	O(1)
Ordered list	O(n)	O(1)	O(1)
Unordered array	O(1)	O(n)	O(n)
Unordered list	O(1)	O(n)	O(n)
Binary Heap	**O(log n)**	**O(log n)**	**O(1)**



Credit to: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html

Priority Queue

Definations & Properties

std::**priority_queue**

```
template <class T, class Container = vector<T>,
  class Compare = less<typename Container::value_type> > class priority_queue;
```

Template parameters

`T`
Type of the elements.

Aliased as member type priority_queue::value_type.

`Container`
Type of the internal *underlying container* object where the elements are stored.

Its value_type shall be T.

Aliased as member type priority_queue::container_type.

`Compare`
A binary predicate that takes two elements (of type T) as arguments and returns a bool.

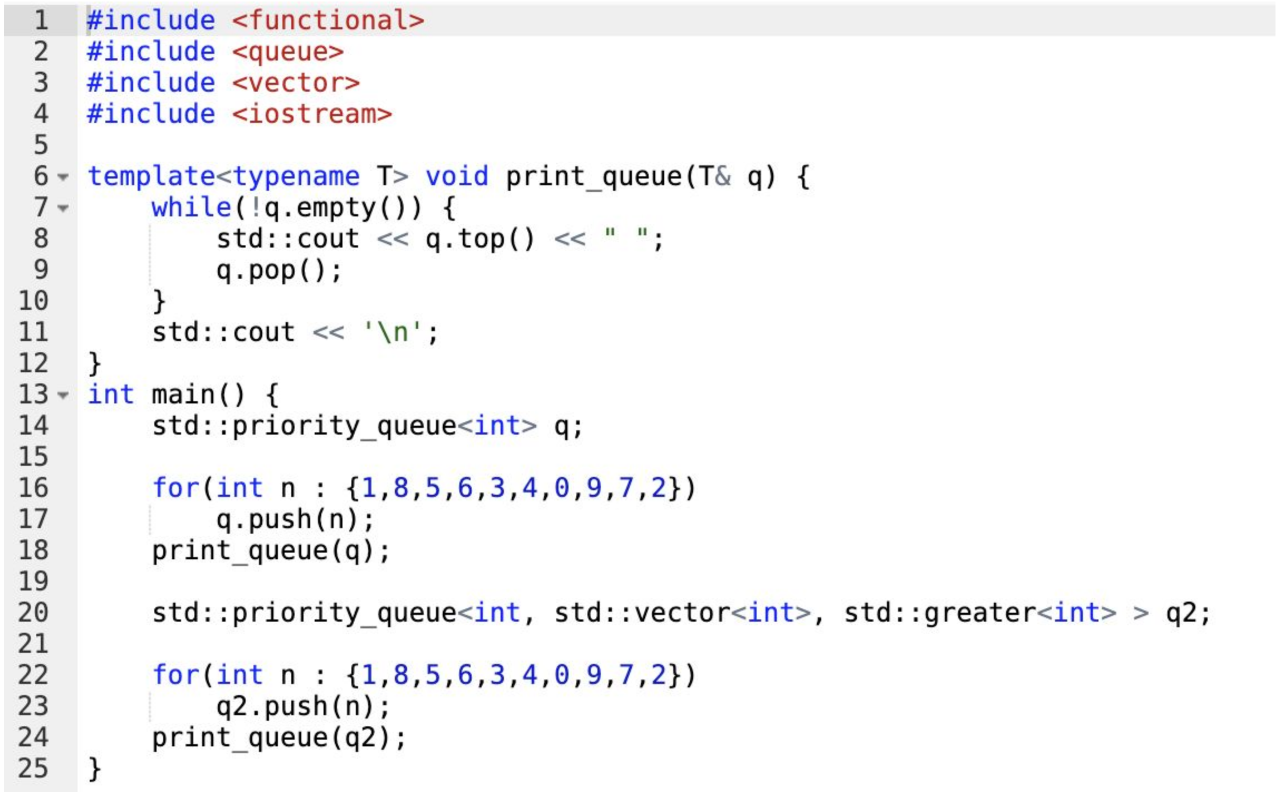
The priority_queue uses this function to maintain the elements sorted in a way that preserves *heap properties* (i.e., that the element popped is the last according to this *strict weak ordering*).

This can be a function pointer or a function object, and defaults to less<T>, which returns the same as applying the *less-than operator* (a<b).

Priority Queue

Example

UCLA Samueli Computer Science



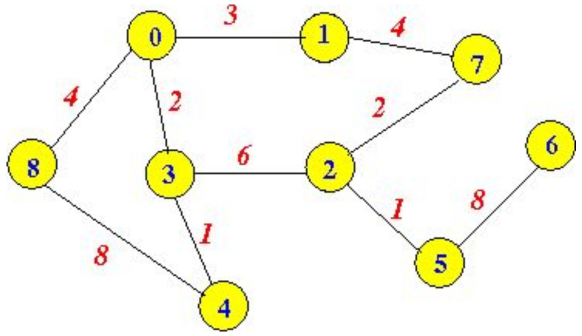
```cpp
#include <functional>
#include <queue>
#include <vector>
#include <iostream>

template<typename T> void print_queue(T& q) {
    while(!q.empty()) {
        std::cout << q.top() << " ";
        q.pop();
    }
    std::cout << '\n';
}
int main() {
    std::priority_queue<int> q;

    for(int n : {1,8,5,6,3,4,0,9,7,2})
        q.push(n);
    print_queue(q);

    std::priority_queue<int, std::vector<int>, std::greater<int> > q2;

    for(int n : {1,8,5,6,3,4,0,9,7,2})
        q2.push(n);
    print_queue(q2);
}
```

```
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9
```

Graph

Introduction, BFS & DFS

- Terms
  - Nodes, edges
  - Adjacency matrix, adjacency list
- Complexity to store graph, add/remove vertex, add/remove edge
- Graph algorithms
  - BFS: Breadth First Search Graph traversal algorithm
  - DFS: Depth First Search Graph traversal algorithm [Link]
  - Dijkstra's Algorithm: Compute the minimum cost paths from a node (e.g., node 1) to all other node in the graph [Link]
  - Prim's Algorithm: finding Minimum cost Spanning Tree [Link]
  - More in CS180



Final Review

- http://getacollegelife.tumblr.com/post/70756494466/my-buddy-rachel-fangs-cs31-and-cs32-cheat
- Final exam practice
  - http://web.cs.ucla.edu/classes/spring19/cs32/Sampleproblems/ChangFinalPractice.pdf
  - http://web.cs.ucla.edu/classes/spring19/cs32/Sampleproblems/ChoiFinalPractice.pdf

Credit: Angelina Poole's personal page. https://sites.google.com/g.ucla.edu/angelinapoole

Final Review: Topics in CS32

You really have learned a lot!

UCLA Samueli Computer Science

- Modern features about C++
  - Resource management
  - Inheritance and polymorphism
  - Templates, Iterators, STL containers
- Data Structures
  - Array, Vector
  - Linked List
  - Stack, Queue
  - Tree, Heap, Graphs
  - Hash Table
- Algorithms and complexity analysis
  - Recursion
  - Big-O
  - Sorting

David’s Reminders: Part 1

- Classes containing members of a class type -- order of construction and destruction, initializer lists.
- Destructors, copy constructors, and assignment operators.
- Base/derived classes and inheritance of members
- Virtual functions, overriding member function implementations, pure virtual functions and abstract base classes, virtual
- Construction order and destruction order
- Recursion

David's Reminders: Part 2

- Various sorting algorithms.
- Preorder traversal and postorder, traversal and then introduced binary trees and more tree algorithms especially using recursion.
- Data structures and corresponding complexity (those Big-Os)
- Clear on open hash tables, load factor, and so on.
    - They should almost *never* assume that collisions are impossible. Even if you have a hash table with 10000 buckets and are storing only 100 items, you may well have 2 of those keys that happen to end up in the same bucket. It's a common beginner mistake to write code that assumes that if a bucket is not empty, you've found the item and the first item in the bucket is it.

David’s Reminders: Not in CS32 final

UCLA Samueli Computer Science

- Multiple inheritance, private inheritance, protected members in inheritance
- Details of AVL trees, 2-3 tree or red-black trees, or more advanced trees.
- Specific hash function (FNV-1)
- Graphs
- Accurate names of member function of STL containers.

“The important thing to know is imply that there exist algorithms that keep the trees more or less balanced so that the average and worst case insert, delete and lookup performance is **O(log *n*)**.” -- David

Problem Checklist

after midterm 2 ...

1. Infix to postfix conversion, postfix to infix conversion, postfix expression evaluation
2. Pre-order, in-order and post-order tree (BST) traversal (can be combined with recursion)
3. Different sorting algorithms, detailed steps, variations and complexity, especially heapsort
4. Tree recursion problem
5. Real-world applications with different data structures (with all what you have learned combined)
    - How items are stored? Are the items sorted / ordered?
    - Complexity of operations (insertion, deletion and search)
    - If possible, how can the hash table be applied in the problem with efficient implementation and less collision?

Helpful Resources

- Previous TA Mark Edmonds’s CS32 worksheet (Spring 2016): [Link]
- UCLA CS Practice Problems [Link]

Thank you and good luck!

UCLA Samueli Computer Science