CSCI 343

1/19/2021

1/21

Remove duplicates:

Public class C{

//remove duplicates from list

//O(N^2)

Static List<Integer> removeDups(List<Integer> ints){

List<Integer> res = new LinkedList<>(); // eventual result

for (int i : ints){ //execute N times

if ( ! (res.contains(i))){ //Might take N steps

res.add(i);

}}

return res;

}

//Does not preserve order O(N) – fastest – probabilistic?? Can become o(nlogn)

Static List<Integer> removeDups2 (List<Integer> ints) {

Hashset<Integer> hs = new HashSet<>(ints);

Return new LinkedList<>(hs);

}

0(NlogN)

Static List <Integer> noDups3 (List<Integer> ints) {

Ints.sort(Integer :: compareTo); //O(N logN)

Int previous = ints.get(0);

List<Integer> res = new LinkedList <>();

res.add(previous);

for(int i=1; i<int.size(); i++ ){

int current = ints.get(i);

if (current != previous){

res.add(current);

Class CTest{

@Test

Void removeDups(){

List<Integer> ints = new ArrayList<>();

Collections.addAll(ints, elements:1,4,3,4,1,2,5);

List<Integer> noDupsInts = c.removeDups(ints);

List<Integer> noDupsInts = c.removeDups2(ints);

//System.Out.printf(“Orginal List = %s%n”, ints);

//System.Out.printf(“No dups list = %s%n”, noDupsInts);

List<Integer> expected = new ArrayList<>();

Collections.addall(expected. 1,4,3,2,5);

assertEquals(expected, noDupsInts);

//assertEquals(expected, noDupsInts2);

For (int I :noDupsInts2){

assertTrue(ints.contains(i));

//assertTrue(unique0cc(noDupsInts2));

1/26

Data structures to update, find, delete data.

Understand patterns of use of data optimize storage to allow find/insert/delete

Fixed size DS is not really useful -> grows

Plate example – LIFO/ STack -> top plate is O(1)

Queue customer example FIFO: whos next to service O(1)

Dynamically growing data structures -> size changes

Relax O(1) Requirement

Probabilistically (high prob)

Amortized O(1) – every month maintenance of data for 1 hour -> take delay and spread it around

Ephemeral vs persistent data structures

Ephemeral – once you update you cant recover from old state

Persistent – never lose previous version (keeps history)

Ephemeral:

Collection x = 0,2,3

x.insert(1)

print x = 1,0,2,3

persistent:

collection x = 0,2,3

collection x’ = x.insert(1)

print x = 0,2,3

print x’ =1,0,2,3

First DS:

Stack – linear access to one end, LIFO

Dynamically growing

Persistent

O(1) insert(push) and delete(pop)

Stack s = new stack (1);

Stack s’ = s.push(1);

Stack push (int x);

List- access other element O(n) where on stack you can only access top

Shape and stack file file