Other Basic Algorithm – I – 2021-2022

**Basic Array Operation**

**public** **class** MyArray {

**public** **static** Object[] **copyOf**(Object[] a, **int** newLen) {

Object[] b = **new** Object[newLen];

**int** minLen = Math.*min*(a.length, newLen);

**for** (**int** i = 0; i < minLen; i++) {

b[i] = a[i];

}

**return** b;

}

**public** **static** Object[] **copyOfRange**(Object[] a, **int** from, **int** to) {

**int** actualTo = Math.*min*(to, a.length);

**int** len = actualTo - from;

Object[] b = **new** Object[len];

**for** (**int** i = from, j=0; i < actualTo; i++, j++) {

b[j] = a[i];

}

**return** b;

}

**private** **static** Object[] **addByIndex**(Object[] a, Object key, **int** index) {

Object[] b = **new** Object[a.length + 1];

**for** (**int** i = 0; i < index; i++) { // Copy 0 to index

b[i] = a[i];

}

b[index] = key; // put at index

**for** (**int** i = index + 1; i < b.length; i++) { // copy from index+1 to last

b[i] = a[i - 1];

}

**return** b;

}

**public** **static** Object[] **addFirst**(Object[] data, Object x) {

**return** *addByIndex*(data, x, 0);

}

**public** **static** Object[] **addLast**(Object[] data, Object x) {

**return** *addByIndex*(data, x, data.length);

}

**public** **static** **void** print(Object[] a) {

System.***out***.println("---------BEGIN----------");

**for** (Object obj : a) System.***out***.print(obj + "\t");

System.***out***.println("\n---------END----------");

}

}

**Delete From Array**

**public class** DeleteFromArray {  
  
 **public static** Object[] copyOfRange(Object[] a, **int** from, **int** to) {  
 **int** actualTo = Math.*min*(to, a.**length**);  
 **int** len = actualTo - from;  
 Object[] b = **new** Object[len];  
 **for** (**int** i = from, j = 0; i < actualTo; i++, j++) {  
 b[j] = a[i];  
 }  
 **return** b;  
 }  
  
 **public static** Object[] deleteFast1(Object[] a) {  
 **return** *copyOfRange*(a,0, a.**length**);  
 }  
  
 **public static** Object[] deleteLast1(Object[] a) {  
 **return** *copyOfRange*(a, a.**length**-1, a.**length**);  
 }  
  
 **public static** Object[] **removeByIndex**(Object[] a, **int** index) {  
 Object[] b = **new** Object[a.**length** -1];  
 **for**(**int** i = 0 , j = 0; i < a.**length**; i++) {  
 **if**( i != index) b[j++] = a[i];  
 }  
 **return** b;  
 }  
  
*// OR* **public static int**[] **removeByIndex2**(**int**[] a, **int** index) {  
 **int**[] b = **new int**[a.**length** -1];  
 **int** j = 0;  
 **for**(**int** i = 0; i < a.**length**; i++) {  
 **if**( i != index) b[j++] = a[i];  
 }  
 **return** b;  
 }  
  
 **public static** Object[] deleteFirst(Object[] a) {  
 Object[] b= *removeByIndex*(a, 0);  
 **return** b;  
 }  
  
 **public static** Object[] deleteLast(Object[] a) {  
 Object[] b = *removeByIndex*(a, a.**length**-1);  
 **return** b;  
 }

**Least Recently Used – LRU**

**import** java.util.\*;

**public** **class** LeastRecentUsed1 {

**private** LinkedHashSet<Integer> set;

**private** Deque<Integer> cache;

**private** **int** capacity;

**public** LeastRecentUsed1(**int** capacity) {

**this**.capacity = capacity;

set = **new** LinkedHashSet<>(capacity);

cache = **new** LinkedList<>();

}

**public** **void** look(**int** key) {

//check if it is already present in cache (deque)

**if** (set.contains(key)) {

//remove it from cache so that it can be added at front of the cache

cache.remove(key);

} **else** {

//not already in cache, adding it

**if** (cache.size() == capacity) {

//remove the last one from cache

**int** last = cache.removeLast();

set.remove(last);

}

}

//add it to cache

cache.push(key);

set.add(key);

System.***out***.println("Looking for key: " + key);

printCache();

}

**public** **void** printCache() {

System.***out***.print("Cache: ");

Iterator<Integer> iterator = cache.iterator();

**while** (iterator.hasNext()) {

System.***out***.print(iterator.next() + " ");

}

System.***out***.println();

}

**public** **static** **void** main(String[] args) {

**int** capacity = 4;

LeastRecentUsed1 lru = **new** LeastRecentUsed1(capacity);

lru.look(1);

lru.look(2);

lru.look(1);

lru.look(3);

lru.look(4);

lru.look(3);

lru.look(5);

lru.look(4);

lru.look(6);

}

}

**URL Shortener**

**public** **class** URLShortener {

**private** **final** String chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

**private** **final** **int** charBase = 62; //chars.length();

**public** String numToString(**int** num) {

StringBuilder sb = **new** StringBuilder();

**while** (num > 0) {

sb.append(chars.charAt(num % charBase));

num = num / charBase;

}

**return** sb.reverse().toString();

}

**public** **int** stringToNum(String str) {

**int** num = 0;

**for** (**int** i = 0; i < str.length(); i++)

num = num \* charBase + chars.indexOf(str.charAt(i));

**return** num;

}

**public** **static** **void** main(String[] args) **throws** MalformedURLException {

String urlStr = "https://www.geeksforgeeks.org/how-to-design-a-tiny-url-or-url-shortener/";

URL url = **new** URL(urlStr);

System.***out***.println("protocol = " + url.getProtocol());

System.***out***.println("host = " + url.getHost());

System.***out***.println("path = " + url.getPath());

String urlPath = url.getPath();

**int** num = urlPath.length();

URLShortener us = **new** URLShortener();

String shorStr = us.numToString(num);

System.***out***.println("Tiny url: " + shorStr);

System.***out***.println("Id: " + us.stringToNum(shorStr));

String uniqueID = UUID.*randomUUID*().toString();

System.***out***.println("IniqueID--->" + uniqueID);

}

}