Interview Questions –

1. Based on an array char[] order which defines a custom order of characters, find if the list of strings is sorted increasingly by the custom order
   1. String order[] = {‘c’,’t’,’a’} indicates ‘c’ < ‘t’ < ‘a’. Based on the order, String “cat” < “act” since ‘c’ < ‘a’.
   2. If two string share a prefix, sort by the first pair of different chars at the same index. “tat” < “taa”.
   3. If a string is a prefix of another string, the shorter string comes first. “aa” < “aac”

e.g. input

order[] = {‘c’, ‘b’, ‘a’}

list[] = {“cc”, “ccc”, “cb”, “bbbbbbb”, “bc”, “ab”, “aaaaaa”}

output = true

e.g. input

order[] = {‘c’, ‘b’, ‘a’}

list[] = {“ca”, “c”, “cb”}

output = false

since the correct sorted order is {“c”, “cb”, “ca”}

public class CustomOrder{

public bool isInOrder(char[] order, String[] list){

/fill the logic here

}

}

1. Design and implement the TTL (Time To Live) caching with LRU (least recently used) as the eviction strategy  
   Your cache will support three functions:
   1. *get(key)* – get the value of the key. If the key does not exist, return null. Also, if the key expired return null
   2. *put(key, value, ttl)* – set or update the value for the given key along with ttl. When the cache reaches its capacity, we need to evict the least recently used record.
   3. *cleanUp()* – clears all the entries from the cache that have expired

public class LRUCacheWithTTL {

public LRUCacheWithTTL(int capacity)

{

//fill this

}

public String get(String key){

//fill this

}

public void put(String key, String value, int ttlInSecs)

{

//fill this

}

public void cleanup(){

//fill this

}

}

1. Determine the min cost of flight from start to end if allowed at most k transfers.

Say, we have the below flights available:

|  |  |  |
| --- | --- | --- |
| **From** | **To** | **Cost** |
| Miami | Atlanta | 100 |
| Miami | Dallas | 300 |
| Miami | Chicago | 600 |
| Atlanta | Dallas | 100 |
| Atlanta | Chicago | 400 |
| Dallas | Chicago | 200 |
| Boston | Dallas | 300 |
| Dallas | Seattle | 500 |
| … | … | … |
| … | … | … |

So, from Miami to Chicago the lowest cost route will be as below:

|  |  |  |
| --- | --- | --- |
| **Max Transfers Allowed** | **Path** | **Cost** |
| 0 | Miami-> Chicago | 600 |
| 1 | Miami -> Atlanta -> Chicago  or  Miami -> Dallas -> Chicago | 100+400=500  or  300+200=500 |
| 2 | Miami -> Atlanta -> Dallas -> Chicago | 100+100+200 = 400 |

public Path[] getAllPaths(Flight[] flights, String from, String to, int maxAllowedTransfers)

{

//fill the logic

}

Where Flight class is:

public class Flight

{

String fromCity,

String toCity;

int cost;

}

Where Path class is:

public class Path

{

Flight[] flights;

int totalCost;

}

1. Say we have two tables as shown below

Item Table:

|  |  |  |
| --- | --- | --- |
| ItemId (PK) | Name | Description |
| 1 | Item-1 | This is item-1. Available in Red color only |
| 2 | Item-2 | This is item-2. Available in Blue color only |
| 3 | Item-3 | This is item-3. Available in both Red and Blue colors |

Item Attribute Table:

|  |  |  |
| --- | --- | --- |
| ItemID(FK) | Attribute | Value |
| 1 | Color | Red |
| 1 | Size | S |
| 1 | Size | M |
| 2 | Color | Blue |
| 2 | Size | X |
| 3 | Color | Red |
| 3 | Color | Blue |
| 3 | Size | XL |

* 1. Write a query to return items (columns – ItemId, Description) that are available in Red color

//fill the query

* 1. Write a query to return items (columns – ItemId, Description) that are available in Red color ONLY (i.e. not available in any other color)

//fill the query

* 1. Write a query to return items (columns – ItemId, Description) that are available in colors other than Red.

//fill the query