Scala Interview Handbook — Batch 10

Generated: 2025-09-13 02:45:31Z (UTC)

Scala Theory & Cheatsheet

```
SCALA THEORY & CHEATSHEET (Quick Ref)
-------------------------------------
- Syntax: vals vs vars; methods; case classes; pattern matching.
- Collections: immutable List/Vector/Map/Set; mutable variants.
- Functional: map/flatMap/filter/fold; Options/Eithers; for-comprehensions.
- Performance: prefer immutable + structural sharing; use arrays for tight loops.
- Testing: ScalaTest AnyFunSuite; assertions; property-based (optional).
```

091. SerializeGraphAdjlist

Problem Overview & Strategy

SerializeGraphAdjlist — Detailed Explanation Approach: BFS/DFS for traversal & cycle detection; Kahn for topological order; Dijkstra with a min-heap. Correctness: Standard graph invariants (visited sets, indegrees, relaxation) guarantee optimality. Complexity: O(V+E) for BFS/DFS/Topo; O((V+E) log V) for Dijkstra.

Scala Solution

```
package problems
object Problem091SerializeGraphAdjlist {
  def serialize(g: Map[Int,List[Int]]): String =
    g.toList.sortBy(_._0).map{case (k,vs)=> s"$k:" + vs.sorted.mkString(",")}.mkString(";")
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem091SerializeGraphAdjlist

class Problem091SerializeGraphAdjlistSpec extends AnyFunSuite {
  test("serialize graph") { val s=Problem091SerializeGraphAdjlist.serialize(Map(1->List(2,3),2->Nil)); assert(s.contain
}
```

092. DeserializeGraphAdjlist

Problem Overview & Strategy

DeserializeGraphAdjlist — Detailed Explanation Approach: BFS/DFS for traversal & cycle detection; Kahn for topological order; Dijkstra with a min-heap. Correctness: Standard graph invariants (visited sets, indegrees, relaxation) guarantee optimality. Complexity: O(V+E) for BFS/DFS/Topo; O((V+E) log V) for Dijkstra.

Scala Solution

```
package problems
object Problem092DeserializeGraphAdjlist {
  def deserialize(s:String): Map[Int,List[Int]] = {
    if (s.trim.isEmpty) return Map.empty
    s.split(";").map { part =>
      val kv = part.split(":"); val k = kv(0).toInt
      val vs = if (kv.length==1 || kv(1).isEmpty) Nil else kv(1).split(",").toList.map(_.toInt)
      k -> vs
    }.toMap
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem092DeserializeGraphAdjlist

class Problem092DeserializeGraphAdjlistSpec extends AnyFunSuite {
  test("deserialize graph") { val g=Problem092DeserializeGraphAdjlist.deserialize("1:2,3;2:"); assert(g(1)==List(2,3))
}
```

093. PalindromePartitioning

Problem Overview & Strategy

PalindromePartitioning — Detailed Explanation Approach: Expand-around-center for every index (odd/even centers). Correctness: The longest palindrome must expand from some center; scanning all centers guarantees finding it. Complexity: O(n^2) time, O(1) space. Edge cases: empty string, all identical chars, multiple optimal answers.

Scala Solution

```
package problems
object Problem093PalindromePartitioning {
  def partition(s:String): List[List[String]] = {
    val res = scala.collection.mutable.ArrayBuffer[List[String]]()
    def isPal(a:Int,b:Int): Boolean = s.substring(a,b+1) == s.substring(a,b+1).reverse
    def bt(i:Int, cur: List[String]): Unit = {
      if (i==s.length) res += cur
      else {
        var j=i
        while (j<s.length) {
          if (isPal(i,j)) bt(j+1, cur :+ s.substring(i,j+1))
          j+=1
        }
      }
    }
    bt(0, Nil); res.toList
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem093PalindromePartitioning

class Problem093PalindromePartitioningSpec extends AnyFunSuite {
  test("pal partitions") { val out=Problem093PalindromePartitioning.partition("aab"); assert(out.contains(List("aa","b"
}
```

094. MinWindowSubString

Problem Overview & Strategy

MinWindowSubString — Detailed Explanation Approach: Use hash maps / frequency arrays and linear scans. Correctness: Frequencies capture necessary character counts; single pass maintains invariants. Complexity: O(n) time, O(Σ) space.

Scala Solution

```
package problems
object Problem094MinWindowSubString {
  def minWindow(s:String, t:String): String = Problem094MinWindowSubstring.minWindow(s,t)
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem094MinWindowSubString

class Problem094MinWindowSubStringSpec extends AnyFunSuite {
  test("alias min window") { assert(Problem094MinWindowSubString.minWindow("ADOBECODEBANC","ABC")=="BANC") }
}
```

094. MinWindowSubstring

Problem Overview & Strategy

MinWindowSubstring — Detailed Explanation Approach: Use hash maps / frequency arrays and linear scans. Correctness: Frequencies capture necessary character counts; single pass maintains invariants. Complexity: O(n) time, O(Σ) space.

Scala Solution

```
package problems

object Problem094MinWindowSubstring {
  def minWindow(s:String, t:String): String = {
    if (t.isEmpty) return ""
    val need = Array.fill(128)(0)
    var required = 0
    t.foreach { c => if (need(c)==0) required += 1; need(c) += 1 }
    val have = Array.fill(128)(0)
    var formed = 0
    var l=0; var bestLen = Int.MaxValue; var bestL=0
    var r=0; while (r<s.length) {
      val c = s(r); have(c) += 1
      if (have(c)==need(c) && need(c)>0) formed += 1
      while (formed==required) {
        if (r-l+1 < bestLen) { bestLen = r-l+1; bestL = l }
        val lc = s(l); have(lc) -= 1
        if (have(lc) < need(lc) && need(lc)>0) formed -= 1
        l += 1
      }
      r += 1
    }
    if (bestLen==Int.MaxValue) "" else s.substring(bestL, bestL+bestLen)
  }
}
```

ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem094MinWindowSubstring

class Problem094MinWindowSubstringSpec extends AnyFunSuite {
  test("min window") {
    assert(Problem094MinWindowSubstring.minWindow("ADOBECODEBANC","ABC") === "BANC")
  }
}
```

095. MaxRectangleHistogram

Problem Overview & Strategy

MaxRectangleHistogram — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

Scala Solution

```
package problems
import scala.collection.mutable

object Problem095MaxRectangleHistogram {
  def largestRectangleArea(h:Array[Int]): Int = {
    val st = new mutable.ArrayDeque[Int]()
    var max = 0; var i=0
    while (i<=h.length) {
      val cur = if (i==h.length) 0 else h(i)
      while (st.nonEmpty && cur < h(st.last)) {
        val height = h(st.removeLast())
        val left = if (st.isEmpty) -1 else st.last
        val width = i - left - 1
        max = math.max(max, height * width)
      }
      st.append(i); i+=1
    }
    max
  }
}
```

ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem095MaxRectangleHistogram

class Problem095MaxRectangleHistogramSpec extends AnyFunSuite {
  test("largest rectangle") {
    assert(Problem095MaxRectangleHistogram.largestRectangleArea(Array(2,1,5,6,2,3)) === 10)
  }
}
```

096. SlidingWindowMax

Problem Overview & Strategy

SlidingWindowMax — Detailed Explanation Approach: Maintain a window with counts/deque; expand and contract to satisfy constraints. Correctness: Each index enters/leaves window at most once, preserving minimality when contracted. Complexity: O(n) time, O(Σ) space.

Scala Solution

```
package problems
import scala.collection.mutable

object Problem096SlidingWindowMax {
  def maxSliding(a:Array[Int], k:Int): Array[Int] = {
    if (a.isEmpty || k==0) return Array()
    val dq = new mutable.ArrayDeque[Int]()
    val res = Array.ofDim[Int](a.length - k + 1)
    var i=0
    while (i<a.length) {
      while (dq.nonEmpty && dq.head <= i-k) dq.removeHead()
      while (dq.nonEmpty && a(dq.last) <= a(i)) dq.removeLast()
      dq.append(i)
      if (i>=k-1) res(i-k+1) = a(dq.head)
      i+=1
    }
    res
  }
}
```

ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem096SlidingWindowMax

class Problem096SlidingWindowMaxSpec extends AnyFunSuite {
  test("sliding window max") {
    assert(Problem096SlidingWindowMax.maxSliding(Array(1,3,-1,-3,5,3,6,7),3).toList == List(3,3,5,5,6,7))
  }
}
```

097. DesignHashmapSimple

Problem Overview & Strategy

DesignHashmapSimple — Detailed Explanation Approach: Idiomatic Scala solution with clear invariants and tested behavior. Correctness: Proven by invariant reasoning and unit tests. Complexity: See code comments.

Scala Solution

```
package problems
object Problem097DesignHashmapSimple {
  class MyHashMap[K,V](cap:Int=1024) {
    private case class Node(k:K, var v:V, var next:Node)
    private val buckets = new Array[Node](cap)
    private def idx(k:K) = (k.hashCode & 0x7fffffff) % cap
    def put(k:K, v:V): Unit = {
      val i = idx(k)
      var n = buckets(i)
      while (n!=null) { if (n.k==k) { n.v=v; return }; n=n.next }
      val newN = Node(k,v,buckets(i)); buckets(i)=newN
    }
    def get(k:K): Option[V] = {
      var n = buckets(idx(k)); while (n!=null) { if (n.k==k) return Some(n.v); n=n.next }; None
    }
    def remove(k:K): Unit = {
      val i=idx(k)
      var n=buckets(i); var prev:Node=null
      while (n!=null) { if (n.k==k) { if (prev==null) buckets(i)=n.next else prev.next=n.next; return }; prev=n; n=n.ne
    }
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem097DesignHashmapSimple

class Problem097DesignHashmapSimpleSpec extends AnyFunSuite {
  test("hashmap") { val m=new Problem097DesignHashmapSimple.MyHashMap[Int,Int](); m.put(1,10); assert(m.get(1).contains
}
```

098. DesignCacheTtl

Problem Overview & Strategy

DesignCacheTtl — Detailed Explanation Approach: Window and token-bucket algorithms for rate limiting; exponential backoff for retries; blocking queues for producer/consumer. Correctness: Counters/tokens enforce limits; backoff reduces contention. Complexity: Amortized O(1) per event.

Scala Solution

```
package problems
object Problem098DesignCacheTtl {
  class TTLCache[K,V] {
    private val store = scala.collection.mutable.HashMap[K,(V,Long)]()
    def put(k:K, v:V, ttlMillis:Long): Unit = store(k)=(v, System.currentTimeMillis()+ttlMillis)
    def get(k:K): Option[V] = store.get(k).flatMap{case (v,exp) => if (System.currentTimeMillis()<=exp) Some(v) else {
    def size: Int = store.size
  }
}
```

ScalaTest

```
package tests

import org.scalatest.funsuite.AnyFunSuite
import problems.Problem098DesignCacheTtl

class Problem098DesignCacheTtlSpec extends AnyFunSuite {
  test("ttl cache") { val c=new Problem098DesignCacheTtl.TTLCache[Int,Int](); c.put(1,2,50); assert(c.get(1).contains(2
}
```

099. UrlValidation

Problem Overview & Strategy

UrlValidation — Detailed Explanation Approach: Use precompiled regex patterns with sane anchors and character classes. Correctness: Patterns encode structural rules; not a full RFC check but covers common cases. Complexity: O(n) matching.

Scala Solution

```
package problems
import java.util.regex.Pattern

object Problem099UrlValidation {
  private val P = Pattern.compile("^(https?://)?([\\w.-]+)(:[0-9]+)?(/.*)?$")
  def isValid(url:String): Boolean = url!=null && P.matcher(url).matches()
}
```

ScalaTest

```
package tests
import org.scalatest.funsuite.AnyFunSuite
import problems.Problem099UrlValidation

class Problem099UrlValidationSpec extends AnyFunSuite {
  test("url validation") {
    assert(Problem099UrlValidation.isValid("https://example.com"))
    assert(!Problem099UrlValidation.isValid("ht!tp://bad"))
  }
}
```