COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

HSPC 2023

Solution Sketches

Problem	Title
P1	Relatively Prime Floating Mountains
P2	Tsireya's Number Play
P3	Interleaved Omangi Messages
P4	Woodsprites’ Patterns
P5	Unstable Floating Mountains
P6	Path-Breaker
P7	Akula Topple
P8	Hidden Pathways
P9	Starfish

P1 – Relatively Prime Floating Mountains

Problem: All pairs of elements of a given set relatively prime?

Solution: Two nested loops to iterate/test all pairs.

```
boolean allPairsRelativelyPrime(int[] d) {
   for (int i = 0; i < d.length; i++)
      for (int j = i+1; j < d.length; j++)
         if (gcd(d [i], d [j]) != 1) return false
   return true;
}
```

P2 – Tsireya's Number Play

Problem: Given a number and 2 digits, make the largest number

Solution: Iterate through the digits. Find first digit smaller than new digit and insert before it. Otherwise insert at end

Example: 987654321 and 7, 0: Answer: 9877654321<u>0</u>

Java notes:

- Convert integer into a string of digits: `Integer.toString(n)`
- Substring of first i symbols: `str.substring(0, i)`
- Remaining substring: `str.substring(i)`
- Insert new digit into string after index i:

```
str.substring(0, i) + Integer.toString(digit) + str.substring(i)
```

P2 – Tsireya's Number Play

Find first digit of n smaller than digit, and insert digit before it:

```
String str = Integer.toString(n) // convert n to string form
boolean inserted = false
for(int i = 0; i < str.length(); i++) {
    if( (str.charAt(i) - '0') < digit) { // i-th digit of number < new digit?
        str = str.substring(0, i) + Integer.toString(digit) + str.substring(i)
        inserted = true; break
    }
}
if (!inserted) str = str + Integer.toString(digit) // insert at end
```

Repeat for both digits

P3 – Interleaved Omangi Messages

Problem: Decrypt string of length n by “unshuffling” every n/k-th character

Solution: Let off = n/k be the offset. Nest two loops. The outer gives the starting character i=0...(off-1) and the inner generates offsets i+0, i+off, i+2·off, …

```
String result = ""
    int offset = s.length()/k // offset size
    for (int i = 0; i < offset; i++) { // start of each group
        for (int j = i; j < s.length(); j += offset)
            result += s.charAt(j) // take every offset character
    }
}
```

P4 – Woodsprites' Patterns

Problem: Compute $\sum_{i=1}^{n} \lfloor n/i \rfloor$ for **very large** n

Observation:

- Takes a long time if n is very large
- When i is large ($i > \sqrt{n}$), there are many repeats.
- For example, if $n = 100$, then $3 = \lfloor n/26 \rfloor = \lfloor n/27 \rfloor = \ldots = \lfloor n/33 \rfloor$
- Compute $\lfloor n/a \rfloor$ directly for small $a \leq \sqrt{n}$ and in groups for large a

How many repeats?

- $\lfloor n/a \rfloor = k \Rightarrow \frac{n}{k+1} < a \leq \frac{n}{k} \Rightarrow \left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{k+1} \right\rfloor$ repeats

P4 – Woodsprites' Patterns

Compute $k = \lfloor n/a \rfloor$ based on size of a

```
long count = 0
long sqrt = Math.sqrt(n)
for (long k = 1; k <= sqrt; k++) { // n/a for large a in groups
    long count1 = n/k - n/(k+1)
    count += count1 * k
}
for (long a = n/(sqrt+1); a >= 1; a--) { // n/a for small a
    count += n/a
}
return count
```

P5 – Unstable Floating Mountains

Problem: Determine whether a sequence comes from a subprime modular (1007) Fibonacci series, allowing skips of up to 9 items

Strategy:

- The first two elements determine the entire sequence, but we only have one (that is, $seq[0]$) since $seq[1]$ may have been skipped. So we need to try all possible choices
- To test a sequence, we try to match an entry against the first 10. For each that match, recursively check the remaining elements.
- Tests all sequences of the form $\mathrm{f}(1) = x$, $\mathrm{f}(2) = seq[0]$, for $0 \leq x < 1007$.
  - If none work, return null
  - If any work, take the one that minimizes the second element

P5 – Unstable Floating Mountains

Test whether a portion of a sequence (seq) matches the sequence f(1)=a, f(2)=b, …

```
boolean satisfied(int a, int b, int[] seq) { // does seq match Fibonacci with skips?
     if(seq.length == 0) return true // empty sequence trivially matches
     for(int i = 0; i < 10; i++) { // try all possible skips
          int next = largestFactors[(a + b) % 1007] // next element of Fibonacci series
          if (next == seq[0]) { // do with match the next item in Fibonacci?
               if(satisfied(b, next, seq[1..seq.length-1])) { // yes...recursively check remainder
                    return true
               }
          }
          a = b; b = next // didn't match - try next in Fibonacci series
     }
     return false
}
```

Precompute largest factors

P5 – Unstable Floating Mountains

Overall solution:

```
int first = seq[0] // fix f(1) to first in element of sequence
int lowestSecond = ∞
for(int zeroth = 0; zeroth < PRIME; zeroth++) { // try all possible 0th elements
    // 0th and 1st define the sequence – need to check that rest of sequence matches
    if (satisfied(zeroth, first, seq[1..seq.length])) { // do we match?
        int second = largestFactors[(zeroth + first) % PRIME] // yes..generate 2nd
        if (second < lowestSecond) // best 2nd seend so far?
            lowestSecond = second // save it
    }
}
// Generate the final answer from first and lowestSecond
```

Problem 6 (Path-Breaker)

Input: $seed$, $N = 10$, $K = 2$

We generate a permutation, P, using the seed

$$P = [1, 3, 0, 2, 5, 4, 6, 9, 8, 7]$$

Place defense systems on 0 and 7

Output: 7 habitats can be saved

```
PriorityQueue<Integer> PQ = new PriorityQueue<>(Collections.reverseOrder());

for (int i = 0; i < N; ++i) {
  if (!visited[i]) {
    visited[i] = true;
    int cur = P[i];
    int length = 1;
    while (cur != i) {
      visited[cur] = true;
      cur = P[cur];
      ++length;
    }
    PQ.add(length);
  }
}

int num_saved = 0;
for (int j = 0; j < K && PQ.size() > 0; ++j) {
  num_saved += PQ.poll();
}
```

4 cycles in the permutation above

Problem is to compute the length of each cycle in the permutation

Greedy solution of placing defense units on the largest cycles is optimal

P7 – Akula Topple

Problem: A game with 2 players, 5 objects [A B C D E], assignment of objects to players (e.g., 1 ⟵ {A,C,D}, 2 ⟵ {A,B,E}), and each player has 6 actions. Determine the winner.

Solution:

- Classical “complete information” game.
- Game state:
  - char[5] indicating the current positions of objects
  - int[3] indicating the remaining moves by player 1 (1-up, 2-up, or Topple)
  - int[3] indicating the remaining moves by player 2
  - Brute force: $5!\,(3^3)(3^3) = 87{,}480$ possibilities
- Each game state is winning either for player-1 or player-2

P7 - Akula Topple

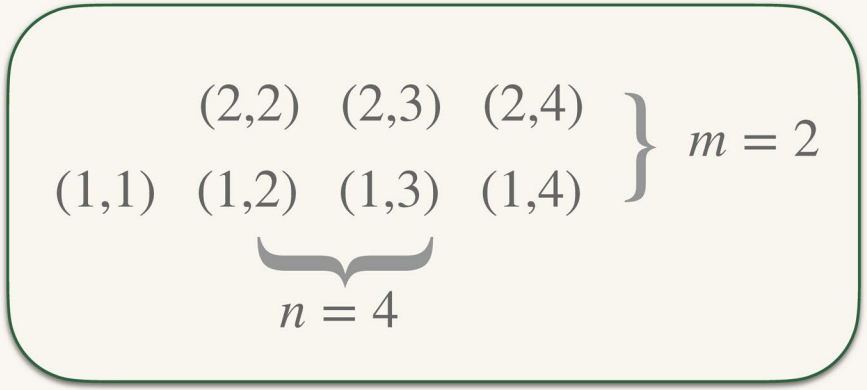
Solution:

- Each game state is winning either for player-1 or player-2
- There are 12 plays in total (6 apiece for players 1 and 2)
- Simulate the game tree, using the set of winning/losing configurations conditioned upon the number of plays remaining in the game.
- Use dynamic programming to “memoize” the results, avoiding the exponential blowup of states

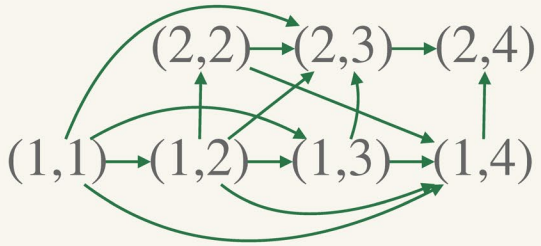
Problem 8 (Hidden Pathways)

Had a small interpretation issue; fixed during clarifiactions in the contest

Input: $n = 4, m = 2$



Should have had $(2,2) \to (1,4)$!



Edge between $(1,3)$ and $(2,3)$ since

$gcd(1+3, 2+3) = gcd(4,5) = 1$

8	6	2	1	1	num_paths
1	1	2	2	1	counts
2	3	4	5	6	values

(1,1) (1,2) (2,2) (1,4) (2,4)
(1,3) (2,3)

num_paths[i] represents paths from one of the (potentially multiple) nodes with sum = i

```
for (int k=m+n; k>1; --k) {
  for (int i = k+1; i <= m+n; ++i) {
    if (gcd(k, i) == 1) {
      num_paths[k] += counts[i] * num_paths[i];
    }
  }
}
```

P9 – Starfish

Problem: Given points X on the line, capacity (cap), length (len), and gap size (gap), compute disjoint intervals that cover a maximum number of points of X, such that no interval has more than cap points, no interval is longer than len, and no two intervals are closer than gap

Solution: Based on dynamic programming

- Precomputed helper indices: Suppose an interval ends at X[i]
  - cap[i] = i-cap+1 // earliest start to interval contains at most cap points
  - len[i] = $\min_{1 \le j \le i} (x[i] - x[j] \le len)$ // earliest start so interval of length at most len
- Suppose interval starts at X[i]
  - gap[i] = $\max_{0 \le j < i} (x[i] - x[j] \le gap)$ // latest end for an interval for gap bound

P9 – Starfish

- Define: $M[i]$ = largest number of points covered by valid intervals ending at $X[i]$
- How to compute $M[i]$?
  - Smallest starting index s for the interval is $\max(len[i], cap[i])$
    $$\Rightarrow \max(len[i], cap[i]) \leq s \leq i$$
  - Largest ending index t for prior interval is $gap[s]$
    $$\Rightarrow 0 \leq t \leq gap[s]$$
  - This covers $(i - s + 1)$ starfish plus prior starfish up to X[t]

$$M[i] = \max_{\max(len[i],cap[i]) \leq s \leq i} \; \max_{0 \leq t \leq gap[s]} ((i - s + 1) + M[t])$$