CS 91 USACO

Bronze Division

Unit 3: Problem Solving Using Algorithms

LECTURE 14: GREEDY ALGORITHMS – PROBLEM SOLVING

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- Practice Problem: barn1
- Practice Problem: crypt1

Practice: Barn Repair (barn1)

SECTION 1

Problem Statement

- It was a dark and stormy night that ripped the roof and gates off the stalls that hold Farmer John's cows. Happily, many of the cows were on vacation, so the barn was not completely full.
- The cows spend the night in stalls that are arranged adjacent to each other in a long line. Some stalls have cows in them; some do not. All stalls are the same width.

Problem Statement

- Farmer John must quickly erect new boards in front of the stalls, since the doors were lost. His new lumber supplier will supply him boards of any length he wishes, but the supplier can only deliver a small number of total boards. Farmer John wishes to minimize the total length of the boards he must purchase.

Problem Statement

- Given M (1 <= M <= 50), the maximum number of boards that can be purchased; S (1 <= S <= 200), the total number of stalls; C (1 <= C <= S) the number of cows in the stalls, and the C occupied stall numbers (1 <= stall_number <= S), calculate the minimum number of stalls that must be blocked in order to block all the stalls that have cows in them.
- Print your answer as the total number of stalls blocked.

INPUT FORMAT (barn1.in):

- Line 1: M, S, and C (space separated)
- Lines 2-C+1: Each line contains one integer, the number of an occupied stall.

SAMPLE INPUT:

```
4 50 18
3
4
6
8
14
15
16
17
21
25
26
27
30
31
40
41
42
43
```

OUTPUT FORMAT (barn1.out):

- A single line with one integer that represents the total number of stalls blocked.

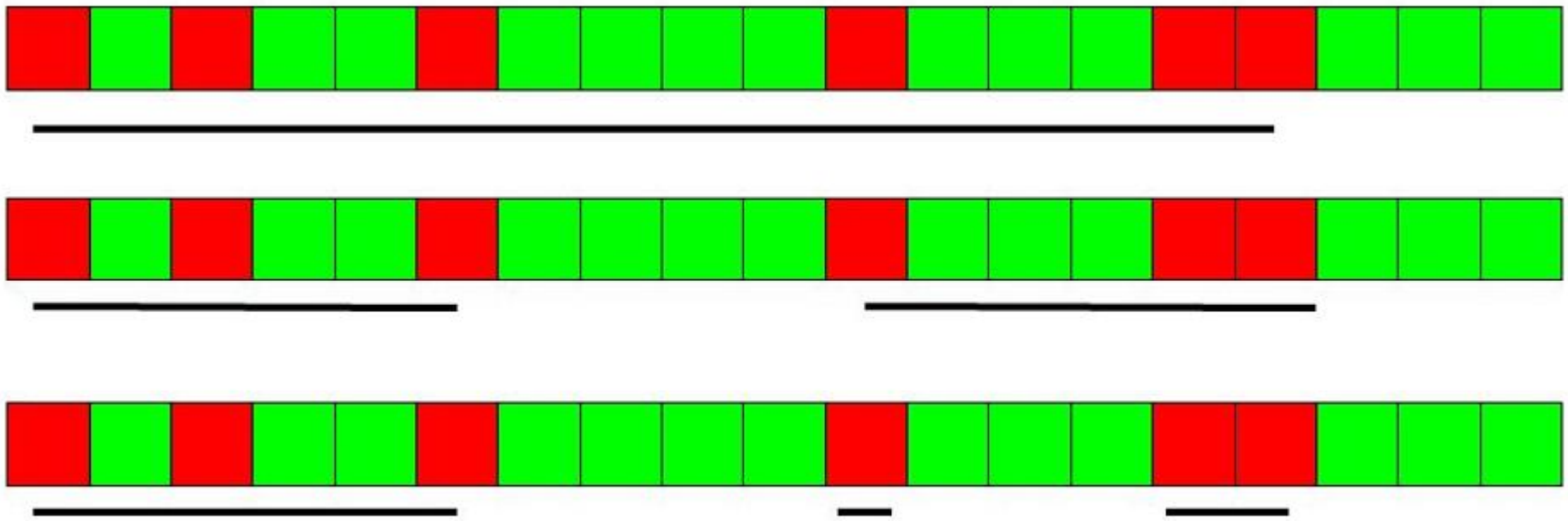
SAMPLE OUTPUT:

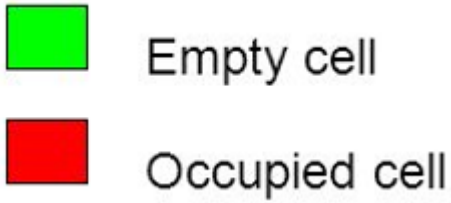
```
25

[One minimum
arrangement is one
board covering
stalls 3-8, one
covering 14-21, one
covering 25-31, and
one covering 40-43.]
```

Nature of the Problem

- Barns with many stalls for cows.
- Cows' stalls with cows are not always connected.
- Covers all stalls with cows.
- Then, connect the stalls with minimum boards

Nature of the Problem





Nature of the Problem

- Read the stall numbers with cows.
- Sort the stall numbers.
- Convert the stall numbers into arrayList of connected stalls.
- Convert the connected stall list to gap list.
- Sort the gap list by the size.
- Take smallest gaps out to make the connected stall list shorter until the number of connected stall numbers are less than or quals the maximum board allowed.

Practice: Prime Cryptarithm (crypt1)

SECTION 2

Problem Statement

- A cryptarithm is usually presented as a pencil-and-paper task in which the solver is required to substitute a digit for each of the asterisks (or, often, letters) in the manual evaluation of an arithmetic term or expression so that the consistent application of the digits results in a proper expression. A classic example is this cryptarithm, shown with its unique solution:

Problem Statement

```
   SEND          9567        S->9  E->5  N->6  D->7
+ MORE         + 1085        M->1  O->0  R->8
-------       --------
 MONEY         10652        Y->2
```

Problem Statement

- The following cryptarithm is a multiplication problem that can be solved by substituting digits from a specified set of N digits into the positions marked with *. Since the asterisks are generic, any digit from the input set can be used for any of the asterisks; any digit may be duplicated as many times as desired.

Problem Statement

- Consider using the set {2,3,5,7} for the cryptarithm below:

```
    * * *
x     * *
 -------
   * * *    <-- partial product 1 -- MUST BE 3 DIGITS LONG
 * * *      <-- partial product 2 -- MUST BE 3 DIGITS LONG
 -------
 * * * *
```

Problem Statement

- Digits can appear only in places marked by `*'. Of course, leading zeroes are not allowed.
- The partial products must be three digits long, even though the general case (see below) might have four digit partial products.

Problem Statement

- In USA, children are taught to perform multidigit multiplication as described here. Consider multiplying a three digit number whose digits are 'a', 'b', and 'c' by a two digit number whose digits are 'd' and 'e':
- [Note that this diagram shows far more digits in its results than the required diagram above which has three digit partial products.].

Problem Statement

```
        a b c     <-- number 'abc'
      x   d e     <-- number 'de'; the 'x' means 'multiply'
    -----------
p1    * * * *     <-- product of e * abc; first star might be 0 (absent)
p2   * * * *      <-- product of d * abc; first star might be 0 (absent)
    -----------
     * * * * *    <-- sum of p1 and p2 (e*abc + 10*d*abc) == de*abc
```

Problem Statement

- Note that the 'partial products' are as taught in USA schools. The first partial product is the product of the final digit of the second number and the top number. The second partial product is the product of the first digit of the second number and the top number.
- Write a program that will find all solutions to the first cryptarithm above (with three digit partial-products) for any subset of supplied non-zero single-digits. Note that the multiplicands, partial products, and answers must all conform to the cryptarithm's framework.

INPUT FORMAT (crypt1.in):

- Line 1: N, the number of digits that will be used
- Line 2: N space separated non-zero digits with which to solve the cryptarithm

SAMPLE INPUT:

```
5
2 3 4 6 8
```

OUTPUT FORMAT (crypt1.out):

A single line with the total number of solutions. Here is the one and only solution for the sample input:

```
     2 2 2
   x   2 2
   -------
     4 4 4
   4 4 4
 ---------
   4 8 8 4
```

SAMPLE OUTPUT:

```
1
```

Generation of All Combinations

- A number set {1, 2, 3}, how can we generate all numbers that can be generated using these numbers as digits for 3 digits:

  111, 112, 113, 121, 122, 123, 131, 132, 133,
  211, 212, 213, 221, 222, 223, 231, 232, 233,
  311, 312, 313, 321, 322, 323, 331, 332, 333.

- Totally, 3^3 numbers. For a set of N numbers should generate N^3 numbers.

Generation of All Combinations

For each pattern, analyze its validity.

Total number of patterns

For each pattern, create an array of index to access the digits

```
int NN = 1;
for (int i=0; i<5; i++) NN *= N;

for (int i=0; i<NN; i++){
    int[] xx = getCode(i, N);
    //System.out.println(Arrays.toString(xx));
    String multiplicant="";
    for (int j=0; j<3; j++) multiplicant += (new Integer(x[xx[j]])).toString();
    int mm = Integer.parseInt(multiplicant);
    if ((new Integer(mm)).toString().length()<3) continue;
}
```

Index Code Generation

Repetitive Modulo Assignment for Each Index

```
public static int[] getCode(int x, int N){
        Integer xx = new Integer(x);
        String xxx = xx.toString();
        int[] xxxx = new int[5];
        for (int i=0; i<xxxx.length; i++) xxxx[i]=0;
        int i=0;
        while (x>0){
            xxxx[i++] = x % N;
            x /= N;
        }
        return xxxx;
    }
```

Implementation of Rejection Rules

Rejection by Illegal Number Length :

```
if ((new Integer(mm)).toString().length()<3) continue;
```

Rejection by Illegal Digits :

```
if (!allIn((new Integer(temp10)).toString(), xstr)) continue;
```

```
public static boolean allIn(String num, String setStr){
    for (int i=0; i<num.length(); i++)
        if (setStr.indexOf(num.substring(i, i+1))<0) return false;
    return true;
  }
```

Nature of the Problem

- Generation of all possible combinations.
- Convert the combinations to two numbers.
- Verify the partial products and final products by
  - Rule 1: check the number of digits
  - Rule 2: are all digits in the combination valid?
- Tally the total number of valid combinations.