Kattis Practice #6 Recap

Competitive Programming Club

Problem A - Coast Length

Pad one pixel (0) around the grid edge to make it a (n+2)*(m+2) grid.

Color the outer rim of this grid (aka flood fill) with bfs

(dfs is easier to write but recursion can cause stack overflow)

Check for every cell in the grid, if it’s land (1), and is next to the colored pixel, we add 1 to the coast length

Problem B - Button Bashing Summary

- Reach the goal time with the fewest number of button presses and wait time
- Able to subtract or add time
- When you reach goal time, the wait time is 0
- If it is impossible to reach the goal time, minimize the wait time

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and t ($1 \leq n \leq 16$ and $0 \leq t \leq 3\,600$): the number of buttons available to change the cooking time, and the desired cooking time in seconds, respectively.
- one line with n space-separated integers b_i ($-3\,600 \leq b_i \leq 3\,600$): the number of seconds added to the cooking time when button i is pressed.

Commentary

- No clear brute force route
- Need a search algorithm
- Algorithm with negative edges + edge relaxation
- Dijkstra? Bellman-ford?
- Setup
  - Array of buttons
  - Queue of nodes
  - Array of nodes
- Steps
  - Read in array of buttons
  - Add the start state of 0 button presses for 0 time to the queue
  - Update the array of nodes as we go
  - Stop when we reach the exact time or stop at the end

This solution produces WA

```
int testCases = sc.nextInt();
for (int a = 0; a < testCases; a++) {
    int buttons = sc.nextInt();
    int goalTime = sc.nextInt();
    int[] buttonsArray = sc.nextIntArray(buttons);

    int[] vertices = new int[3601];
    vertices[0] = 0;
    for(int b = 1; b < 3601; b++) {
        vertices[b] = Integer.MAX_VALUE;
    }
    int storeTime = Integer.MAX_VALUE;
    int storePresses = Integer.MAX_VALUE;;
    LinkedList<Integer> queue = new LinkedList<>();
    if(goalTime == 0) {
        storeTime = 0;
        storePresses = 0;
    }else{
        queue.add(0);
    }
    while(queue.size() > 0) {
        int currentTime = queue.poll();
        int presses = vertices[currentTime];
        for(int edge : buttonsArray) {
            int next = currentTime + edge;
            if(next == goalTime) {
                storeTime = goalTime;
                storePresses = presses+1;
                queue.clear();
                break;
            }
            if(next > goalTime && next < storeTime) {
                storeTime = next;
                storePresses = presses+1;
            }
            next = Math.max(0, Math.min(3600, next));
            if(vertices[next] > vertices[currentTime] + 1) {
                vertices[next] = vertices[currentTime] + 1;
                queue.add(next);
            }
        }
    }
    System.out.println(storePresses + " " + (storeTime - goalTime));
}
}
```

Sample Solution in Java

```
// Code Starts Here
int testCases = sc.nextInt();
for (int a = 0; a < testCases; a++) {
    int buttons = sc.nextInt();
    int goalTime = sc.nextInt();
    int[] buttonsArray = sc.nextIntArray(buttons);

    int[] vertices = new int[3601];
    vertices[0] = 0;
    for(int b = 1; b < 3601; b++) {
        vertices[b] = Integer.MAX_VALUE;
    }
    int storeTime = Integer.MAX_VALUE;
    int storePresses = Integer.MAX_VALUE;;
    LinkedList<Integer> queue = new LinkedList<>();
    if(goalTime == 0) {
        storeTime = 0;
        storePresses = 0;
    }else{
        queue.add(0);
    }
```

```
while(queue.size() > 0) {
    int currentTime = queue.poll();
    int presses = vertices[currentTime];
    for(int edge : buttonsArray) {
        int next = currentTime + edge;
        next = Math.max(0, Math.min(3600, next));
        if(vertices[next] > vertices[currentTime] + 1) {
            vertices[next] = vertices[currentTime] + 1;
            queue.add(next);
        }
    }
}
for(int i = goalTime; i <= 3600; i++) {
    if(vertices[i] != Integer.MAX_VALUE) {
        System.out.println(vertices[i] + " " + (i - goalTime));
        break;
    }
}
```

Problem C - Phone List Summary

- Determine whether a list of phone numbers is ‘consistent’
- Consistent in this case is that no phone number is a prefix of another
- 40 test cases, 10,000 phone numbers, length 10, in 3 seconds

Sample Input 1

```
2
3
911
97625999
91125426
5
113
12340
123440
12345
98346
```

Sample Output 1

```
NO
YES
```

Commentary

- Sorting by length and searching is too slow
- Use hashing
- Hash the phone numbers and see if the set has its prefix
- Setup
  - Set of phone number strings
  - Array of phone numbers
  - Optional variable to store minimum length
- Steps
  - Save the phone numbers in the set and array
  - Iterate through the array and iterate over the length of the phone number to see if the set contains the prefix

TLE Solution

```
int t = sc.nextInt();
for(int a = 0; a < t; a++){
    int n = sc.nextInt();
    boolean result = true;
    boolean cont = true;
n   String[] phoneNumbers = sc.nextStringArray(n);
n log n  Arrays.sort(phoneNumbers, comp);
n   for(int i = 0; i < n; i++){
        if(cont){
          n for(int j = i + 1; j < n; j++){
                if(phoneNumbers[j].startsWith(phoneNumbers[i])){
                    result = false;
                    cont = false;
                    break;
                }
            }
        }else{
            break;
        }
    }
    if(result){
        System.out.println("YES");
    }else{
        System.out.println("NO");
    }
}
```

Sample Solution in Java

```
// Code Starts Here
int t = sc.nextInt();
for(int i = 0; i < t; i++) {
    int n = sc.nextInt();
    HashSet<String> set = new HashSet<>();
    String[] queue = new String[n];
    //PriorityQueue<String> queue = new PriorityQueue<>();
    boolean consistent = true;
    int min = Integer.MAX_VALUE;
 n  for(int j = 0; j < n; j++) {
        String input = sc.nextLine();
        if(input.length() < min) {
            min = input.length();
        }
        set.add(input);
        queue[j] = input;
    }
 n  for(int j = 0; j < n; j++) {
        String str = queue[j];
   10   for(int a = min; a < str.length(); a++) {
          1  if(set.contains(str.substring(0, a))) {
                consistent = false;
                j = n;
            }
        }
    }
    System.out.println(consistent? "YES" : "NO");
}
```

Problem D - Dragon Ball I

Objective: find the length of the shortest path that spans across all of the given cities

Floyd would be too slow, Dijkstra only gives us single source shortest distance

Solution: run dijkstra between all pairs of the targeted city. Calculate the total distance for each eligible path. Find the minimum length.

Problem E - Mountain Biking

Objective: calculate the end velocity of a mountain biker starting at a certain point on a mountain

Solution: Physics!
https://www.dummies.com/education/science/physics/how-to-calculate-a-change-in-velocity-based-on-acceleration/

https://github.com/JonSteinn/Kattis-Solutions/blob/master/src/Mountain%20Biking/C/main.c

Function to calculate the end velocity from a given start velocity, starting coordinates, and ending coordinates

For loop to determine end velocity across multiple hill segments. Another for loop to calculate end velocity from each possible starting point

Problem F - Playing the Slots