FPC 2020 problem presentation; spoiler alert!

FPC 2020

Problem A - Alien Journey (1/3)

FPC 2020

Problem description

Searching for the smallest height, such that a squared shape UFO could travel from top left to bottom right of a map. Along the path, the height of the UFO should be greater than all the cells beneath.

Solution Part 1

First intution:

- Have a method `Check()`
- Check whether ($height = h$)
- Is h high enough for the ship to travel?

Problem A - Alien Journey (2/3)

FPC 2020

Solution Part 2

- Checking for all possible h takes took much time!, $h \leq 10^9$!
- Binary search (or PQ)!

Solution Part 3

To implement `Check(h)` there are multiple ways:

- 2D sliding window
- RMQ
- Segment tree

Problem A - Alien Journey (3/3)

FPC 2020

Pitfalls

- Allow UFO to go outside the map
- Height is very large so trying every height will not work

Problem B - Banitsa (1/3)

FPC 2020

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Observation

Using graph coloring theory, we know we need at most three toppings ("colors")

Solution

DFS, while using two alternating "colors" to color the nodes

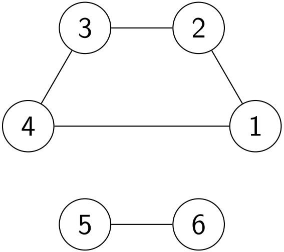
Problem B - Banitsa (2/3)

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating “colors” to color the nodes



Problem B - Banitsa (2/3)

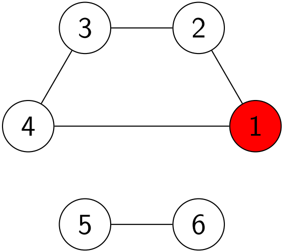
FPC 2020

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating “colors” to color the nodes



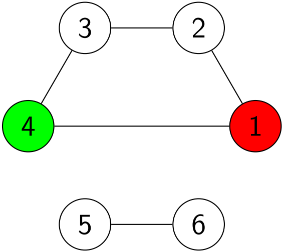
Problem B - Banitsa (2/3)

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating “colors” to color the nodes



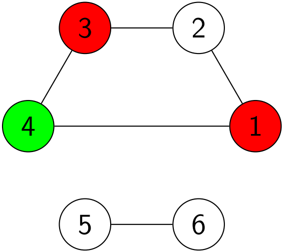
Problem B - Banitsa (2/3)

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating "colors" to color the nodes



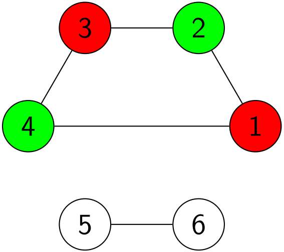
Problem B - Banitsa (2/3)

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating “colors” to color the nodes



FPC 2020

Problem B - Banitsa (2/3)

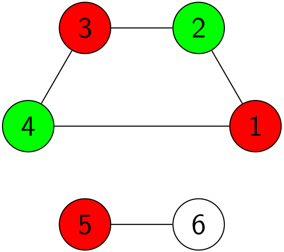
FPC 2020

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating "colors" to color the nodes



Problem B - Banitsa (2/3)

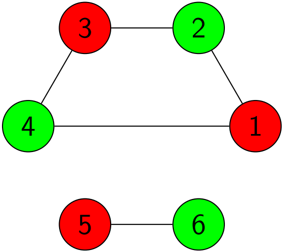
FPC 2020

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating "colors" to color the nodes



Problem B - Banitsa (3/3)

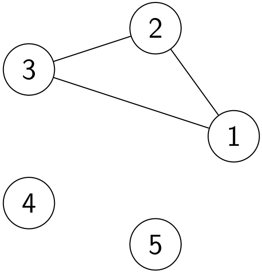
FPC 2020

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating “colors” to color the nodes



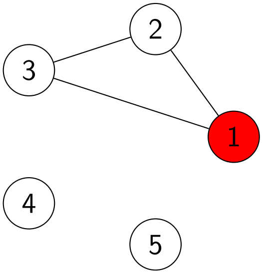
Problem B - Banitsa (3/3)

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating "colors" to color the nodes



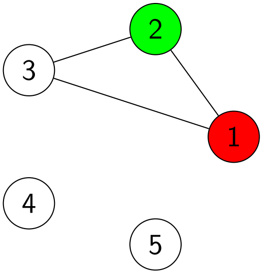
Problem B - Banitsa (3/3)

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating “colors” to color the nodes



Problem B - Banitsa (3/3)

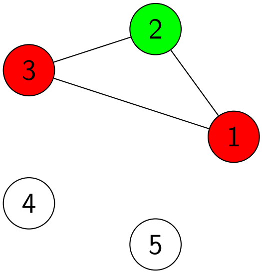
FPC 2020

Problem description

How many toppings do you need, so that all given pairs do not have the same topping?

Solution

DFS, while using two alternating “colors” to color the nodes



Problem C - Chill and Netflix (1/4)

FPC 2020

Problem description

Given a set of numbers, how many integers $<=$ n (given) can be written as a sum of numbers from the set. Using each number any times and using at least one number.

Solution

2 known solutions, one with heuristics and one with graph modelling

Problem C - Chill and Netflix (2/4)

FPC 2020

Solution Graph Modelling Part 1

First observation: if integer m can be reached then for any k from the set with buttons, any number m' can be reached if (m' mod k) == (m mod k) (by adding k an arbitrary number of times)

Solution Graph Modelling Part 2

Find smallest number x from the buttons set and find for all the numbers [0,1...x-1], smallest number m that could be reached st m mod x equals that number. If m can be reached then m+x, m+2*x.. can be reached. So we only need, for each possible modulo, to find smallest reachable integer m

Problem C - Chill and Netflix (3/4)

FPC 2020

Solution Graph Modelling Part 3

Think of modulos as nodes, and buttons as edges to get from a modulo to another Apply Dijkstra for getting smallest m for each possible value modulo x. Go through all modulos and calculate biggest k st. m+k*x < total number of second Dense Graph with x nodes where x is min(buttons)

Pitfalls

Recursive solutions are too slow, they try all possible combinations which are a lot Some teams modelled the problem as a graph but insead of modulos, nodes where actual reachable moments.

Problem C - Chill and Netflix (4/4)

FPC 2020

Solution: Brute force

- Keep a boolean array of all timestamps
- For every button, iterate over array and set timestamps you can reach to `true`
- But, this is too slow

Observations

- Divide all buttons and movie length by their GCD
- Start with the two smallest buttons that are relatively prime to each other
  - Example, take 3 and 5: from this point on, you know that you reach *all* seconds after second 15
- Thus, we can do the brute force on a really small size!

Problem D - Ducks and Sharks

FPC 2020

Problem description

Calculate a ranking based on a list of matches.

Solution

Process the matches one by one, keeping track of the scores per team in a `HashMap` or dictionary, pretty straight-forward.

Pitfalls

- Only print the top 5
- Sort alphabetically

Problem E - Excursion (1/2)

FPC 2020

Problem description

Given a tree with values in each node calculate the maximum sum you can get by following a path in the tree

Solution

- Recursively calculate the maximum sum $S1$ achievable by starting at that node and moving to the children.
- Also calculate the maximum path sum $S2$ which only contains the current node (doesn't have to start here).
- Take the two highest $S1$ values among the children
- The answer is the maximum value among the $S2$ sums, which we can also keep track along the way.

Problem E - Excursion (2/2)

FPC 2020

Problem description

Given a tree with values in each node calculate the maximum sum you can get by following a path in the tree

Solution

Recursion for the win!

Pitfalls

- Always need to select one city even though all values may be negative
- Take into account that the result may not fall in `int` range

Problem F - Family Tree (1/2)

Problem description

Calculate the “width” of the given tree.

Fun Fact

Based on events in real life!

FPC 2020

Problem F - Family Tree (1/2)

FPC 2020

Problem description

Calculate the “width” of the given tree.

Fun Fact

Based on events in real life!

Problem F - Family Tree (1/2)

FPC 2020

Problem description

Calculate the “width” of the given tree.

Fun Fact

Based on events in real life!

Problem F - Family Tree (1/2)

FPC 2020

Problem description

Calculate the “width” of the given tree.

Fun Fact

Based on events in real life!

Problem F - Family Tree (1/2)

FPC 2020

Problem description

Calculate the “width” of the given tree.

Fun Fact

Based on events in real life!

Problem F - Family Tree (2/2)

FPC 2020

Problem description

Calculate the “width” of the given tree.

Solution

1. First, read in the full tree (lines are not in order)
2. Create a list of nodes L, initially only containing the root
3. While L is not empty:
   1. Retrieve all children of all nodes in L
   2. Set L to this list of all children
4. Return the maximum size of L

Pitfalls

- The lines are not necessarily in order

Problem G - Group Activities (1/2)

FPC 2020

Problem description

Find the smallest number of people that you can divide into all of the given group sizes.

Solution

Find the Least Common Multiple (LCM) of all numbers.

```
def gcd(a, b): # recursive
    if b == 0:
        return a
    return gcd(b, a % b)
```

```
def gcd(a, b): # iterative
    while b != 0:
        a, b = b, a % b
    return a
```

```
def lcm(a, b):
    return a * b / gcd(a, b)
```

Problem G - Group Activities (2/2)

FPC 2020

Problem description

Find the smallest number of people that you can divide into all of the given group sizes.

Solution

Find the Least Common Multiple (LCM) of all numbers.

Pitfalls

- For Java and C^{++}: do not multiply over the `long` limit
- Also: `Scanner.nextInt()` does not accept `long`s
- Do *not* use floating-point numbers
  (e.g. `Math.pow` in Java or `a / b` in Python)

H - Halt and Catch Fire (1/2)

FPC 2020

Problem description

Very straightforward: Create an interpreter that runs the provided program. Buffer each line of code, then run through them and run the instructions.

Solution

- Store program into buffer, create map for registers
- As long as $pc is within bounds:
  - Parse the instruction, taking into account immediate values and registers.
  - Run the instruction
  - Increment the $pc register
- Output `$out` to `stdout`

H - Halt and Catch Fire (2/2)

FPC 2020

Pitfalls

- Not buffering lines: Can't jump backward!
- Not using `$pc` as a register: Something like `mov 1 $pc` won't work
- `$pc` *can* be less than zero! Stop the program if this is the case.

Problem I - Integrity Overflow

FPC 2020

Problem description

Check whether a list of passwords is correct, allowing at most one character to be wrong.

Solution

Check each password character-by-character and count the number of characters that are different.

- Count equal to 0 or 1? ✓
- Count 2 or more? X

Pitfalls

- With a correct password being `DENIED`, system is insecure
- Passwords are not always of same length