Форма 7: разбор

1:

> Мы на лекции обсудили формулу mask & (mask - 1), которая дает для маски ее копию без последней единицы. Поиссследуем другие операции на масках:

> Что сделает mask1 | mask2? В ответе расскажите, что произойдет с множествами, которые были заданы масками. Дополнительно, если интересно, можно подумать, как реализованы O_CREAT и O_WRONLY в функции open на C, если их нужно объединять через | (пример тут https://linuxhint.com/posix-open-function-c-programming/), это не будет оцениваться

> Что сделает mask | (mask + 1)?

`mask1 | mask2` сделает маску, в которой будут стоять единицы везде, где они были хотя бы в одной из исходных масок. Значит, мы сделали объединение множеств. Так, например, работают всякие маски с флагами в C, когда нужно скомбинировать разные режимы.

`mask | (mask + 1)` добавляет еще одну единицу в маску. Какую единицу? Можно посмотреть на то, что делает `mask + 1`. Последний блок вида `011...11` заменится на `100...00`. При объединении получится, что мы добавили новую единицу на месте самого правого нуля.

2:

> Нам для подсчета суммы в подмножестве нужно уметь по степени двойки понять ее логарифм. 1 -> 0, 2->1, 4->2, 8->3, 16->4, и так далее. Предложите как предподсчитать такую информацию заранее, и потом вычислять нужное значение во время пересчета за O(1)

Самый простой вариант - создать массив (или хеш-таблицу) `memory_log[x]`, и заполнить ее предварительно нужными числами.

```
for i in range(log):
    memory_log[2 ** i] = i
```

Тогда когда нам понадобится понять, какой логарифм у $x=2^k$, мы можем просто взять `memory_log[x]`