Задача A. Правильная скобочная последовательность

Эта задача разбиралась на лекции; код на языке Python:

```
s = input()
st = []
for i in s:
    if i == '(' or i == '[' or i == '{':
        st.append(i)
        continue
    if not st:
        print('no')
        exit()
    if i == '}' and st[-1] != '{':
        print('no')
        exit()
    elif i == ')' and st[-1] != '(':
        print('no')
        exit()
    elif i == ']' and st[-1] != '[':
        print('no')
        exit()
    st.pop()
if not st:
    print('yes')
else:
    print('no')
```

Задача B. Игра в ЯКарты

Эта задача разбиралась на лекции. Код на python:

```
from collections import deque

q1 = deque(map(int, input().split()))
q2 = deque(map(int, input().split()))
cnt = 0
while q1 and q2 and cnt < 1_000_000:
    cnt += 1
    card1, card2 = q1.popleft(), q2.popleft()
    if (card1 > card2 and not (card1 == 9 and card2 == 0)) or \
            (card1 == 0 and card2 == 9):
        q1.append(card1)
        q1.append(card2)
    else:
        q2.append(card1)
        q2.append(card2)
if cnt == 1_000_000:
    print('botva')
elif not q1:
    print('second', cnt)
else:
    print('first', cnt)
```

Задача C. Удалите скобки

Посчитаем глубину скобочной последовательности. При открытой скобке увеличиваем глубину,

при закрытой - уменьшаем. Если же глубина стала отрицательной - обнуляем глубину, удаляем эту скобку(добавляем к ответу +1). Теперь удаляем все открывающие скобки, не имеющие после себя закрывающие. Для этого просто к ответу прибавляем глубину, которую мы уже посчитали. Выводим полученный ответ.

Код на Python:

```
s = input()
ans = 0
have = 0
for i in s:
    if i == '(':
        have += 1
    elif have:
        have -= 1
    else:
        ans += 1
print(ans + have)
```

Задача D. База отдыха

Автор задачи: Инесса Шуйкова, разработка: Михаил Кондрашин

Заметим, что имена групп не важны. Поэтому создадим очередь, в которой будем хранить последовательности взятых домиков. Также заведем множество, в котором будем хранить все свободные на текущий момент домики.

- Если приходит запрос добавления, то возьмем K минимальных чисел из множества. Эти числа выведем, а также добавим вектор этих чисел в конец очереди.
- Иначе просто возьмем первый элемент в очереди, после чего вернем все эти числа обратно в множество.

Асимптотика: $\mathcal{O}((\sum M_i) \log (\sum M_i))$.

Код на C++:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    std::ios::sync_with_stdio(false), std::cin.tie(0), std::cout.tie(0);

    int n, k;
    cin >> n >> k;
    set<int> free;
    queue<vector<int>> q;
    for (int i = n; i > 0; i--) free.insert(i);

    while (k--) {
        string ord;
        cin >> ord;
        if (ord[0] == 'O') {
            string name;
            int cnt;
            cin >> name >> cnt;
            q.emplace();
            while (cnt--) {
```

```
                cout << *free.begin() << ' ';
                q.back().push_back(*free.begin());
                free.erase(free.begin());
            }
            cout << '\n';
        } else {
            string name;
            cin >> name;
            auto tmp = q.front();
            q.pop();
            for (auto &i : tmp) free.insert(i);
        }
    }

    for (auto &i : free) cout << i << ' ';
    cout << '\n';
}
```

Задача E. Опять скобки

Вспомним, как работает алгоритм проверки строки на ПСП:

1. Взять следующий символ строки.
2. Изменить текущий «баланс» — количество открытых и еще не закрытых скобок. Если текущий символ равен '(', то увеличить баланс на 1, иначе уменьшить на 1.
3. Если баланс стал отрицательным, то строка не является ПСП (по причине того, что на текущий момент закрылось скобок больше, чем открылось).
4. Если в конце цикла баланс стал равен нулю, то для каждой скобки нашлась парная, поэтому строка является ПСП.

От нас же требуется понять, удаление каких символов гарантирует нам получение ПСП. Можно представить себе, что наш алгоритм проверки как будто бы дойдет до удаляемого символа и остановится перед ним. Если баланс когда-то на текущий момент был отрицательным, то удаление символа никак не поможет — итоговая строка в любом случае не ПСП. Иначе баланс всегда был неотрицательным и равен k на текущий момент. Это значит, что после удаляемого символа должно стоять окончание ПСП с балансом, равным $-k$.

Давайте запустим алгоритм проверки на ПСП с конца строки. Условие изменится на противоположное — балансы должны быть неположительными. Тогда мы для каждой позиции i запомним, какой был баланс нашего $forward$-алгоритма, и был ли на текущий момент этот индекс валидным (был ли раньше баланс отрицательным). Такие же величины посчитаем для *backward*-алгоритма. Тогда при удалении символа i мы хотим проверить, что значения $forward$-алгоритма для $i-1$ и *backward*-алгоритма для $i+1$ стыкуются — балансы равны по модулю, а также нигде на префиксе и суффиксе не нарушалось требование ПСП. Также требуется аккуратно рассмотреть случаи, когда удаляется первый или последний символ.

Решение имеет сложность $O(|s|)$.

Код на Python:

```
s = input()
n = len(s)
okpref, bal = [], []
balance = 0
for i in range(n):
    if s[i] == '(':
```

```
        balance += 1
    else:
        balance -= 1
    if balance < 0:
        okpref.append(0)
    elif i == 0:
        okpref.append(1)
    else:
        okpref.append(okpref[-1])
    bal.append(balance)
ans, ok, balance = 0, 1, 0
for i in range(n - 1, -1, -1):
    if i == 0 and balance == 0 and ok:
        ans += 1
    if i == 0:
        continue
    if ok and okpref[i - 1] and balance == bal[i - 1]:
        ans += 1
    if s[i] == ')':
        balance += 1
    else:
        balance -= 1
    if balance < 0:
        ok = 0
print(ans)
```

Задача F. Сортировка вагонов

Заметим следующий факт. Пусть самый верхний вагон в тупике имеет номер x, а первый вагон на 1-м пути имеет номер y. Тогда есть два случая:

- Если $x < y$, то нам обязательно нужно сначала вытащить вагон x на второй путь. Ведь иначе на втором пути вагоны окажутся в порядке $y\ x$, а это неправильный порядок (ведь $y > x$);
- Если $x > y$, то нам обязательно нужно поставить вагон y в тупик. Ведь иначе на втором пути вагоны окажутся в порядке $x\ y$, но это неправильный порядок (ведь $x > y$).

Осталось понять, что делать при пустом тупике. На самом деле в этом случае мы должны просто поставить первый вагон с 1-го пути в тупик. Если будет надо, то потом мы его оттуда вытащим. В конце надо проверить, что вагоны на втором пути отсортированы по возрастанию. Код на Python:

```
n = int(input())
a = list(map(int, input().split()))
dead_end, ans = [], []
cur_path2 = []
for i in range(n):
    while dead_end and dead_end[-1] < a[i]:
        ans.append((2, 1))
        cur_path2.append(dead_end.pop())
    ans.append((1, 1))
    dead_end.append(a[i])
while dead_end:
    ans.append((2, 1))
    cur_path2.append(dead_end.pop())
ok = True
```

```
for i in range(1, n):
    if cur_path2[i] < cur_path2[i - 1]:
        ok = False
if not ok:
    print(0)
else:
    print(len(ans))
    for i in ans:
        print(*i)
```

Задача G. Гоблины и очереди

Разбор

Для начала научимся обрабатывать запросы '+' и '-', для этого нам подойдёт обычная очередь, используем для этого класс deque из модуля collections. Теперь нам надо научиться быстро добавлять гоблина в середину.

Для этого воспользуемся идеей организации очереди на двух стеках – просто разделим очередь на две половины и будем при добавлении гоблина в середину или удалении гоблина из очереди восстанавливать равенство числа гоблинов в двух частях.

Асимптотика решения: $O(1)$ на запрос

Решение

```
from collections import deque

class Queue:
    def __init__(self):
        self.l = deque()
        self.r = deque()

    def push(self, v: int):
        self.l.appendleft(v)

    def push_mid(self, v: int):
        while len(self.l) > len(self.r):
            self.r.appendleft(self.l.pop())
        self.l.append(v)

    def pop(self) -> int:
        while len(self.l) > len(self.r):
            self.r.appendleft(self.l.pop())
        return self.r.pop()

q = Queue()
n = int(input())
for _ in range(n):
    cmd = input().split()
    if cmd[0] == '+':
        q.push(int(cmd[1]))
    if cmd[0] == '*':
        q.push_mid(int(cmd[1]))
    if cmd[0] == '-':
        print(q.pop())
```

Задача H. Мадагаскар [C, B']

Эта задача является обычной проверкой на ПСП с восстановлением ответа (а именно — с восстановлением пар скобок). Самое сложное в этой задаче — правильно понимать, закрывающая или открывающая сейчас скобки, а также ее номер. Код на Python:

```
s = input()
n = len(s)
st = []
ans, food2ind = [-1] * n, [-1] * n
foods = 0
for i in range(n):
    if s[i].islower():
        food2ind[i] = foods
        foods += 1
for i in range(n):
    if st and s[i].lower() == st[-1][0].lower() and s[i] != st[-1][0]:
        if s[i].isupper():
            ans[i] = food2ind[st[-1][1]]
        else:
            ans[st[-1][1]] = food2ind[i]
        st.pop()
    else:
        st.append((s[i], i))
if not st:
    print('Possible')
    for i in ans:
        if i != -1:
            print(i + 1, end=' ')
else:
    print('Impossible')
```

Задача I. Утерянная очередь

Представим, что операций удаления нет. Сопоставим каждому элементу $_i$ в утерянной очереди число c_i — количество *расширений*, затронувших этот элемент. Также будем хранить глобально число k — количество *расширений*. Другими словами, вместо элемента q_i на позиции i находятся 2^{c_i+k} элементов q_i. В такой модели, при вставке, положим $c_i = -k$, при *расширении*, увеличим k на 1.

Заметим, что $m \leqslant 2 \cdot 10^5 < 2^{20}$, поэтому если $c_1 + k \geqslant 20$, ответом на все запросы удаления будет q_1, а запросы *расширения* и добавления можно игронировать. Если $c_1 + k < 20$, вычислим реальное количество q_1 и будем честно удалять. Получили решение за $O(m)$.

Код на Python:

```
from collections import deque

t = int(input())
q = deque()
cnt, fl = 0, False
L = 20
for _ in range(t):
    line = input().split()
    if line[0] == '1':
        q.append([-cnt, int(line[1])])
    elif line[0] == '2':
        cnt += 1
```

```
        if cnt < L:
            q[0][0] *= 2
    else:
        if cnt >= L:
            print(q[0][1])
            continue
        print(q[0][1])
        q[0][0] -= 1
        if q[0][0] == 0:
            fl = False
            q.popleft()
    if not fl and q:
        fl = True
        q[0][0] = 2 ** (cnt + q[0][0])
```

Задача J. Очередь в магазине

Подзадача 1

Для получение 40 баллов по данной задаче необходимо было проэмулировать процесс, описанный в условии. Можно, например, поддерживать очередь в контейнере `deque`. Операции первого и третьего типов выполняются за $\mathcal{O}(1)$. Операцию второго типа можно выполнять явно за $\mathcal{O}(n)$, где n — количество элементов в деке. Таким образом, время работы составит $\mathcal{O}(n^2)$.

Подзадача 2

Рассмотрим решение, позволяющее набрать полный балл по данной задаче. Понятно, что для того, чтобы получить достаточно быстрое решение, необходимо научиться обрабатывать событие второго типа достаточно быстро. Заведем некоторую переменную d, изначально равную нулю, в которой мы будем накапливать некоторое значение агрессивности, которое нужно прибавить ко всем людям в очереди.

Теперь рассмотрим, как, имея посчитанное значение d, обработать каждое из трех возможных событий. Пусть произошло событие первого типа. Тогда в очередь нужно добавить число $a - d$, так как мы подразумеваем, что число d должно быть прибавлено ко всем элементам очереди. Если произошло второго типа, то нужно увеличить d на y, а первое число очереди увеличить на $x - y$. В случае, если произошло событие третьего типа, нужно извлечь первое число из очереди, и записать в ответ это число, увеличенное на d.

В таком случае все операции обрабатываются за $\mathcal{O}(1)$, а итоговое время работы равно $\mathcal{O}(n)$.

```
from collections import deque

n = int(input())
q = deque()
push = 0
for _ in range(n):
    line = input().split()
    if line[0] == '1':
        q.append(int(line[1]) - push)
    elif line[0] == '2':
        x, y = int(line[1]), int(line[2])
        push += y
        q[0] += x - y
    else:
        print(q.popleft() + push)
```