Задача A. Числа Фибоначчи

Эта задача разбиралась на лекции. Код на Python:

```
n = int(input())
f = [1, 1]
while len(f) < n:
    f.append(f[-1] + f[-2])
print(f[-1])
```

Задача B. Кузнечик

Введем $dp[i]$ как количество способов добраться на i-ый столбец. База: $dp[1] = 1, dp[2] = 1$. Далее делаем переход для i-ой позиции. Понятно, что мы пришли в этот столбец непосредственно с $i-1$-ой позиции. или с $i-2$-ой позиции. Поэтому, так как эти множества вариантов непересекаемы, $dp[i] = dp[i-1] + dp[i-2]$.

Код на эту задачу совпадает с кодом для предыдущей задачи :)

Задача C. Лестница

Пусть dp_i = максимальное количество денег, если в итоге будем стоять на i-й ступеньке. Начальное заполнение: $dp_0 = a_0, dp_1 = \max(0, a_0) + a_1$. Переход $dp_i = \max(dp_{i-1}, dp_{i-2}) + a_i$

Код на Python:

```
n = int(input())
a = list(map(int, input().split()))
dp = [0] * n
dp[0] = a[0]
if n > 1:
    dp[1] = max(0, a[0]) + a[1]
for i in range(2, n):
    dp[i] = max(dp[i - 1], dp[i - 2]) + a[i]
print(dp[-1])
```

Задача D. Кузнечик-K

Та же идея, что с обычным кузнечиком, только переход у нас в столбик i из отрезка $[max(1, i-k), i-1]$. База динамики: $dp[1] = 1$.

Код на Python:

```
n, k = map(int, input().split())
dp = [1]
for i in range(1, n):
    if i < k:
        dp.append(sum(dp))
    else:
        dp.append(sum(dp[-k:]))
print(dp[-1])
```

Задача E. Кузнечик и лягушки

Пусть dp_i будет количество способов добраться на позицию i. База динамики — $dp_1 = 1$. Переходы на колонку i будет из колонок от $\max(1, i-k)$ до $i-1$, там, где нет лягушек. Если в этом отрезке на какой-то позиции нет лягушки, то прибавляем: $dp_i += dp_j$. Проще всего сначала посчитать очередное dp_i, а потом поставить туда 0 (если в клетке лягушка).

Код на Python:

```
n, k = map(int, input().split())
m = int(input())
```

```
a = set(map(int, input().split()))
dp = [1]
for i in range(1, n):
    if i < k:
        dp.append(sum(dp))
    else:
        dp.append(sum(dp[-k:]))
    if i + 1 in a:
        dp[-1] = 0
print(dp[-1])
```

Задача F. Калькулятор

Пусть dp_i = минимальное количество операций, чтобы получить число i. Тогда какие есть переходы? Всегда можно сделать $dp_i = dp_{i-1} + 1$. Если i делится на два, то есть еще переход $dp_{i/2} + 1$. Аналогично для случая, когда i делится на три.

Код на Python:

```
n = int(input())
dp = [0, 0]
for i in range(2, n + 1):
    dp.append(dp[i - 1] + 1)
    if i % 2 == 0:
        dp[i] = min(dp[i], dp[i // 2] + 1)
    if i % 3 == 0:
        dp[i] = min(dp[i], dp[i // 3] + 1)
print(dp[-1])
```

Задача G. Калькулятор

В этой задаче надо сделать восстановление ответа, как разбиралось на лекции. Для каждого числа сохраним то число, из которого получили текущее. Потом будем прыгать по предкам, пока не дойдем до начала.

Код на Python:

```
n = int(input())
dp = [0, 0]
pred = [-1, -1]
for i in range(2, n + 1):
    best_pred = i - 1
    dp.append(dp[i - 1] + 1)
    if i % 2 == 0 and dp[i // 2] < dp[i]:
        best_pred = i // 2
        dp[i] = min(dp[i], dp[i // 2] + 1)
    if i % 3 == 0 and dp[i // 3] + 1 < dp[i]:
        best_pred = i // 3
        dp[i] = min(dp[i], dp[i // 3] + 1)
    pred.append(best_pred)
print(dp[-1])
ans, cur = [], n
while cur > 0:
    ans.append(cur)
    cur = pred[cur]
ans.reverse()
print(*ans)
```

Задача H. Покупка билетов

Пусть dp_i = сколько минимум времени надо, чтобы дать билеты первым i людям. Тогда есть следующие варианты для подсчета dp_i:

- $dp_{i-1} + a_i$
- $dp_{i-2} + b_{i-1}$
- $dp_{i-3} + c_{i-2}$

Понятно, что из этих вариантов берем минимальный.

Код на Python:

```
n = int(input())
dp = [0]
q = [()]
for i in range(1, n + 1):
    a, b, c = map(int, input().split())
    q.append((a, b, c))
    cur = dp[i - 1] + a
    if i >= 2:
        cur = min(cur, dp[i - 2] + q[-2][1])
    if i >= 3:
        cur = min(cur, dp[i - 3] + q[-3][2])
    dp.append(cur)
print(dp[-1])
```

Задача I. Драгоценный камень

Получить 20 баллов можно, просто перебрав всевозможные варианты покупок и продаж турмалина. Существует 2^N последовательностей дней, в которые осуществляются покупки и продажи. Каждый из них можно обработать за $O(N)$. Получаем решение за $O(2^N \cdot N)$.

Рассмотрим решение на 50 баллов. Воспользуемся динамическим программированием. Пусть $dp[i][0]$ — это максимальная прибыли при условии, что мы рассмотрели первые i дней, и при этом в день i произошла покупка. Аналогично, $dp[i][1]$ — максимальная прибыли при условии, что были рассмотрены первые i дней, но в i-й день произошла продажа.

Тогда $dp[i][0] = \max\limits_{j=1}^{i-1} dp[j][1] - a_i$, а $dp[i][1] = \max\limits_{j=1}^{i-1} dp[j][0] + a_i$. Данное решение работает за $O(N^2)$.

Теперь заметим, что можно не считать каждый раз значение максимума, а поддерживать его в некоторой переменной. Будем поддерживать две переменные: max_0 и max_1, в которых будут храниться соответствующие максимальные значения динамики. Тогда описанное выше решение оптимизируется до $O(N)$.

Код на Python:

```
n = int(input())
a = list(map(int, input().split()))
max_sell, max_buy = 0, 0
dp_sell, dp_buy = [0], [0]
for i in range(0, n):
    dp_sell.append(a[i] + max_buy)
    dp_buy.append(-a[i] + max_sell)
    max_sell = max(max_sell, dp_sell[-1])
    max_buy = max(max_buy, dp_buy[-1])
print(max(dp_sell))
```

Задача J. Играем в игры

Заметим, что нас не интересует порядок чисел — нам нужно только то, сколько раз какое число встречается (обозначим это за cnt_i). Пусть dp_i = максимальное кол-во очков, которое можно получить, если уничтожить первые i чисел и при этом для операции было выбрано число i. Тогда для подсчета dp_i нас интересует максимум из следующих величин:

- dp_{i-1} — в этом случае число i уже уничтожено (ведь при уничтожении всех $i-1$ также уничтожаются i и $i-2$), а потому ничего добавлять не надо.
- $dp_{i-2}+i*cnt_i$ — потому что после dp_{i-2} остались только числа $i-1, i$ (и может кто-то больше), а потому необходимо уничтожить все числа i

Код на Python:

```
MAXN = 200_000
dp = [0] * MAXN
n = int(input())
a = list(map(int, input().split()))
cnts = [0] * MAXN
for i in a:
    cnts[i] += 1
dp[1] = cnts[1]
for i in range(2, MAXN):
    dp[i] = max(dp[i - 1], dp[i - 2] + i * cnts[i])
print(dp[-1])
```