Задача A. Финансовая задача

Посчитаем, сколько анталеров в сумме стоят w шоколадок. Заметим, что цены шоколадок образуют арифметическую прогрессию с разностью k. Таким образом суммарная стоимость равна сумме первых w членов этой прогрессии. Посчитать эту сумму можно по формуле: $s = \frac{2k+k(w-1)}{2} \cdot w$.

Если число n не меньше полученной суммы, то имеющихся денег хватит, поэтому ответом является 0. Иначе, придется занять $s - n$ анталеров.

Приведем фрагмент кода программы.

```
int main(){
        int k, n, w;
        cin >> k >> n >> w;
        cout << max(0, (2 * k + k * (w - 1)) * w / 2 - n);
        return 0;
 }
```

Сложность решения: $O(1)$.

Задача B. Музыкальная шкатулка

Переберем первую трубку, по которой нужно будет ударить – ее номер лежит в промежутке $[1; k+1]$. Обозначим номер выбранной трубки как i. Теперь звучат все трубки в промежутке $[i-k; i+k]$ (некоторых трубок из промежутка может не быть). Таким образом, первая трубка, которая не звучит, имеет номер $i+k+1$. Чтобы она зазвучала, необходимо ударить по какой-то трубке справа. Максимальный номер такой трубки равен $i+2k+1$. Ударим по этой трубке. Будем продолжать описанный процесс до тех пор, пока либо не зазвучат все трубки, либо нельзя будет ударить по какой-то трубке.

Если в конце зазвучали все трубки, сравним число трубок, по которым мы ударили с найденным ответом и выберем наилучший ответ.

Приведем фрагмент кода программы.

```
int ans = 2 * n, ansI;
for (int i = 0; i <= k; ++i) {
     int c = 1 + i;
     int curAns = 0;
     bool fl = false;
     while (c <= n) {
          ++curAns;
          if (c + k >= n) {
               fl = true;
          }
          c += 2 * k + 1;
     }
     if (fl && curAns < ans) {
          ans = curAns;
          ansI = i;
     }
}
```

```
cout << ans << endl;
int c = 1 + ansI;
for (int i = 0; i < ans; ++i) {
     cout << c << ' ';
     c += 2 * k + 1;
}
```

Сложность решения: $O(nk)$.

Задача C. Учимся рисовать Диаграмму

Отсортируем столбики в порядке неубывания высоты.

Воспользуемся двоичным поиском по ответу. Левой границей поиска является высота столбика с номером $\left\lfloor \frac{n}{2} \right\rfloor + 1$, а правой границей – высота этого столбика, увеличенная на k.

Пусть двоичный поиск зафиксировал некоторый ответ m. Необходимо понять, можно ли, пририсовав не более k клеточек, получить ответ m.

Понятно, что столбики, находящиеся слева от центрального, увеличивать не имеет смысла. Кроме того, если мы увеличили средний столбик до высоты m, мы должны были увеличить все столбики справа, меньшие m, до высоты m. Посчитаем, сколько клеточек для этого нужно. Для этого, для всех столбиков справа, высота которых меньше m, посчитаем, сколько клеточек к нему нужно приписать: $a[i] - m$.

Просуммируем все эти числа и сравним с числом k. Если нам понадобилось не более k кубиков, сдвинем левую границу двоичного поиска, иначе сдвинем правую границу.

Приведем фрагмент кода программы.

```
sort(a.begin(), a.end());
long long l = a[n / 2] - 1, r = a[n / 2] + k + 1;
while (r - l > 1) {
    long long m = (l + r) / 2;
    long long cnt = 0;
    for (int i = n / 2; i < n; ++i) {
        if (a[i] > m) {
            break;
        }
        cnt += m - a[i];
    }
    if (cnt <= (long long)k) {
        l = m;
    } else {
        r = m;
    }
}
cout << l;
```

Сложность решения: $O(nlogn + nlogk)$.

Задача D. Проблема Лайнландии

Формализуем условие задачи. Дан ориентированный граф на n вершинах. Необходимо добавить в граф минимальное количество ребер таким образом, чтобы из вершины s были достижимы все остальные вершины.

Сконденсируем граф и найдем все истоки в получившемся графе конденсации. Заметим, что если провести ребро из s в любую вершину каждого истока (кроме случая, когда вершина s содержится в компоненте сильной связности, являющейся истоком), то все вершины станут достижимыми.

Приведем фрагмент кода программы.

```
void topSort(int v) {
	used[v] = true;
	for (auto to : gr[v]) {
		if (!used[to]) {
			topSort(to);
		}
	}
	order.push_back(v);
}

void dfs(int v) {
	used[v] = true;
	cmp[v] = curCmp;
	cmps[curCmp].push_back(v);
	for (auto to : g[v]) {
		if (!used[to]) {
			dfs(to);
		}
	}
}

...

for (int i = 0; i < n; ++i) {
	for (auto to : g[i]) {
		if (cmp[i] != cmp[to]) {
			++deg[cmp[to]];
		}
	}
}
int ans = 0;
vector<pair<int, int>> res;
for (int i = 0; i < curCmp; ++i) {
	if (deg[i] == 0 && i != cmp[s - 1]) {
		++ans;
		res.push_back(make_pair(s, cmps[i].back() + 1));
	}
}
cout << ans << endl;
for (auto i : res) {
	cout << i.first << ' ' << i.second << endl;
```

```
}
```

Сложность решения: $O(n+m)$.

Задача E. Марио и трубы

Посчитаем следующие величины: $b[i] = a[i] - a[i+1]$.

Тогда поймем, при каком условии Марио сможет пройти от трубы x до трубы y, если $x < y$. Необходимо, чтобы выполнилось условие: $\min\limits_{x \le i < y} b[i] \ge -1$.

Теперь поймем, при каком условии Марио сможет пройти от трубы x до трубы y, если $x > y$. Необходимо, чтобы выполнилось условие: $\max\limits_{y \le i < x} b[i] \le 1$.

Теперь необходимо понять, что происходит с величинами $b[i]$, когда высота труб на отрезке $[l; r]$ увеличивается на некоторую константу d. Нетрудно заметить, что поменяются только две величины: $b[l-1]$ и $b[r]$ (при условии, что они существуют).

Таким образом, для ответов на запросы, необходима структура данных, позволяющая быстро изменять элемент массива, а также искать минимум и максимум на заданном отрезке. В качестве такой структуры данных можно использовать дерево отрезков.

Приведем фрагмент кода программы. В коде функция get(l, r) возвращает структуру, имеющую поля min и max, обозначающие минимум и максимум на интервале $[l; r)$, а функция add(pos, v) прибавляет к элементу с индексом pos число v.

```
for (int i = 0; i < n - 1; i++) {
    b[i] = a[i] - a[i + 1];
}
for (int i = 0; i < m; i++) {
    int type;
    cin >> type;
    if (type == 1) {
        int x, y;
        cin >> x >> y;
        x--; y--;
        if (x < y) {
            if (get(x, y).min >= -1) {
                cout << "YES" << endl;
            } else {
                cout << "NO" << endl;
            }
        } else {
            if (get(y, x).max <= 1) {
                cout << "YES\n";
            } else {
                cout << "NO\n";
            }
        }
    } else {
        int l, r, d;
        cin >> l >> r >> d;
        l--; r--;
        if (l > 0) {
            add(l - 1, -d);
```

```
            }
            if (r < n - 1) {
                add(r, d);
            }
        }
    }
```

Сложность решения: $O(mlogn)$.