Задача A. Самое частое слово

Самое сложное в этой задаче — ввести данные. Пример кода на C++:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    string input;
    map<std::string, int> cnt;
    while (std::cin >> input) {
        cnt[input]++;
    }
    string best;
    for (auto [s, c] : cnt) {
        if (best.empty() || c > cnt[best]) {
            best = s;
        }
    }
    cout << best << '\n';
}
```

И на Python:

```
import sys
from collections import defaultdict

d = defaultdict(lambda: 0)
for line in sys.stdin.read().strip().split():
    d[line] += 1
best_cnt = max(d.values())
best_strings = [k for k, v in d.items() if v == best_cnt]
print(min(best_strings))
```

Задача B. Предметный указатель

В этой задаче надо просто сделать то что просят в условии. Пример кода на C++:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    map<int, set<string>> keywords;
    for (int i = 0; i < n; ++i) {
        string word;
        int page;
        cin >> word >> page;
        keywords[page].insert(word);
    }
    for (auto i : keywords) {
        cout << i.first;
        for (const auto &word : i.second)
            cout << " " << word;
        cout << "\n";
    }
}
```

И на Python:

```
from collections import defaultdict

keywords = defaultdict(lambda: set())
n = int(input())
for i in range(n):
    word, page = input().split()
    keywords[int(page)].add(word)
for k, v in sorted(keywords.items()):
    print(k, *sorted(list(v)))
```

Задача C. Симметрическая разность множеств

Опять же, сделаем ровно то, что просят в условии (C++):

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    set<int> a;
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        a.insert(x);
    }
    int m;
    cin >> m;
    set<int> b;
    for (int i = 0; i < m; i++) {
        int x;
        cin >> x;
        b.insert(x);
    }
    set<int> res;
    for (auto i : a) {
        if (b.find(i) == b.end()) {
            res.insert(i);
        }
    }
    for (auto i : b) {
        if (a.find(i) == a.end()) {
            res.insert(i);
        }
    }
    cout << res.size() << '\n';
    for (auto i : res) {
        cout << i << ' ';
    }
}
```

В случае Python можно сделать намного проще:

```
n = int(input())
a = set(map(int, input().split()))
m = int(input())
b = set(map(int, input().split()))
res = a ^ b
print(len(res))
print(*sorted(res))
```

Задача D. Минимум на отрезке

В этой задаче можно просто хранить в `set` все элементы в текущем окне длины k. Соответственно, при сдвиге окна вправо надо добавить элемент в сет. Если размер сета больше k, то надо удалить элемент, который был с индексом $i - k$.

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    set<pair<int, int>> s;
    for (int i = 0; i < n; i++) {
        s.insert({a[i], i});
        if (i >= k) {
            s.erase({a[i - k], i - k});
        }
        if (s.size() >= k) {
            cout << s.begin()->first << ' ';
        }
    }
}
```

У этой задачи нет простого решения на Python. Как вариант, можете ознакомиться с модулем `heapq`. Либо послушать одну из следующих лекций — там будет разобран другой алгоритм.

Задача E. Англо-латинский словарь

Для этой задачи приведем решение на языке Python. Тут надо было просто сделать "обратное"отображение.

```
from collections import defaultdict

n = int(input())
d = defaultdict(lambda: set())
for _ in range(n):
    s = input()
    data = s.split()
    data.pop(1)  # delete '-'
    for i in data[1:]:
        if i[-1] == ',':
            i = i[:-1]  # delete ','
        d[i].add(data[0])
```

```
print(len(d))
for k, v in sorted(d.items()):
    print(k, '-', ' '.join(sorted(v)))
```

Задача F. Приготовление милкшейков

В этой задаче надо было просто промоделировать то что происходит. Однако можно было напороться на неожиданную проблему: если прерывать считывание ингредиентов после первого, которого не хватает, то решение не будет работать. Это звучит логично, но на самом деле плохо — так как цикл закончится, но данные во вводе все еще будут. В итоге вы считаете не тип запроса, а ингредиент и его количество. Пример кода на C++:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int q;
    cin >> q;
    map<string, int> have;
    while (q--) {
        int type;
        cin >> type;
        string name;
        int cnt;
        if (type == 1) {
            vector<pair<string, int>> need;
            string shakeName;
            int n;
            cin >> shakeName >> n;
            bool ok = true;
            for (int i = 0; i < n; i++) {
                cin >> cnt >> name;
                need.push_back({name, cnt});
                if (have[name] < cnt) {
                    ok = false;
                }
            }
            if (!ok) {
                cout << ":(\n";
                continue;
            }
            for (int i = 0; i < n; i++) {
                have[need[i].first] -= need[i].second;
            }
            cout << "Milkshake " << shakeName << " is ready\n";
        } else {
            cin >> cnt >> name;
            have[name] += cnt;
            cout << name << ": " << have[name] << '\n';
        }
    }
}
```

Пример кода на Python:

```
from collections import defaultdict

q = int(input())
mapik = defaultdict(lambda: 0)

for _ in range(q):
    line = input().split()
    tt = int(line[0])
    if tt == 1:
        name = line[1]
        ok = True
        n = int(input())
        need = []
        for i in range(n):
            need.append(input().split())
            need[-1][0] = int(need[-1][0])
            if mapik[need[-1][1]] < need[-1][0]:
                ok = False
        if not ok:
            print(':(')
            continue
        for cnt, item in need:
            mapik[item] -= cnt
        print(f'Milkshake {name} is ready')
    else:
        k, s = int(line[1]), line[2]
        mapik[s] += k
        print(f'{s}: {mapik[s]}')
```

Задача G. Вычислительная ихтиология

Будем в сете хранить пары вида (в какой момент в аквариуме появится рыбка, номер аквариума). Также будем хранить текущую позицию Игоря. Теперь надо делать следующее:

1. Смотрим на минимальный элемент в сете
2. Если Игорь не успеет дойти до него, то мы нашли ответ
3. Иначе удалим эту пару из сета, посчитаем время появления новой рыбки, и добавим обратно в сет

Будем повторять это, пока не найдем ответ на задачу. Понятно, что этот процесс закончится, так как при $n >= 2$ с какого-то момента новая рыбка будет появляться в обоих аквариумах, но Игорь не сможет сделать обе записи. Приводим код на C++ (для решения задачи на Python вам предлагается изучить модуль heapq):

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    set<pair<int, int>> s;
```

```
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        s.insert({max(1, 1000 - a[i]), i});
    }
    int curt = 0;
    int curpos = 0;
    while (1) {
        int nxt = (*s.begin()).first;
        int idx = (*s.begin()).second;
        if (nxt < curt + abs(idx - curpos)) {
            cout << nxt;
            return 0;
        }
        curt = nxt;
        curpos = idx;
        s.erase(s.begin());
        a[idx]++;
        s.insert({curt + max(1, 1000 - a[idx]), idx});
    }
}
```

Задача H. Ладьи-защитники

Автор задачи: Михаил Кондрашин, разработка: Михаил Кондрашин

Рассмотрим некоторый подпрямоугольник. Заметим, что каждая его клетка атакована какой-то ладьей тогда и только тогда, когда в каждой строке x ($x_1 \leqslant x \leqslant x_2$) или в каждом столбце y ($y_1 \leqslant y \leqslant y_2$) есть хотя бы одна ладья.

Теперь будем решать задачу, пользуясь данным критерием. Создадим множество $freeRows$, в котором будем хранить номера строк, в которых **нет** ни одной ладьи. Аналогично в множестве $freeCols$ будем хранить номера столбцов, в которых **нет** ни одной ладьи. Теперь для того, чтобы ответить на запрос третьего типа, нужно проверить, правда ли в множестве $freeRows$ есть хотя бы один элемент x, такой что $x_1 \leqslant x \leqslant x_2$, или же в множестве $freeCols$ есть хотя бы один элемент y, такой что $y_1 \leqslant y \leqslant y_2$. Если хранить множества упорядоченными, то данные проверки можно выполнять за $\mathcal{O}(\log n)$ при помощи бинарного поиска.

Для того, чтобы отвечать на запросы первого и второго типа, будем для каждой строки и для каждого столбца хранить, сколько ладей находятся в соответствующей строке и в соответствующем столбце. При добавлении новой ладьи нужно увеличить счетчики для ее строки и столбца, а также удалить строку из множества $freeRows$ и столбец из множества $freeCols$. При удалении ладьи нужно уменьшить счетчики для ее строки и столбца, а также, если в соответствующей строке или столбце не осталось ни одной ладьи, нужно добавить номер этой строки или столбца в $freeRows$ или $freeCols$.

Асимптотика: $\mathcal{O}(q \log n)$.

Приведем код на C++. На Python эта задача не имеет простого решения, так как нам нужно уметь находить наименьший элемент, который $>= x$ для произвольного x:

```
#include <bits/stdc++.h>

using namespace std;

const int MAXN = 1e5 + 5;
int horCnt[MAXN], verCnt[MAXN];

int main() {
        ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
```

```
        int n, q;
        cin >> n >> q;
        set<int> hor, ver;
        for (int i = 0; i <= n + 1; i++) {
                hor.insert(i);
                ver.insert(i);
        }

        while (q--) {
                int type;
                cin >> type;
                if (type == 1) {
                        int x, y;
                        cin >> x >> y;
                        horCnt[x]++, verCnt[y]++;
                        hor.erase(x), ver.erase(y);
                } else if (type == 2) {
                        int x, y;
                        cin >> x >> y;
                        if (--horCnt[x] == 0) {
                                hor.insert(x);
                        }
                        if (--verCnt[y] == 0) {
                                ver.insert(y);
                        }
                } else {
                        int l1, r1, l2, r2;
                        cin >> l1 >> r1 >> l2 >> r2;
                        if (*hor.lower_bound(l1) > l2 || *ver.lower_bound(r1) > r2)
                                cout << "Yes\n";
                        } else {
                                cout << "No\n";
                        }
                }
        }
}
```

Задача I. Машинки

Для каждой машинки научимся узнавать момент, когда она в следующий раз понадобится (назовем это число `next`). Затем для добавления новой машинки есть три случая:

1. Если на полу есть место, то добавим нужную машинку
2. Если машинка уже на полу, то просто обновим для нее `next`
3. Иначе просто уберем машинку, для которой `next` максимален, а новую добавим.

Понятно, что это оптимально, так как нет необходимости снимать машинку раньше, чем она нам понадобится. Также нет смысла держать на полу ту машинку, которая нам долго не будет нужна. Для быстрого определения ближайшего момента, когда потребуется машинка, заранее предпосчитаем это. Для этого просто пройдемся с конца и будем в мапе поддерживать ближайший справа момент, когда будет нужна машника. Код на C++:

```
#include <bits/stdc++.h>
```

```
using namespace std;

int main()
{
    int n, k, p;
    cin >> n >> k >> p;
    vector<int> a(p), next(p);
    map<int, int> lastPos;
    for (int i = 0; i < p; i++) cin >> a[i];
    for (int i = p - 1; i >= 0; i--) {
        auto it = lastPos.find(a[i]);
        if (it != lastPos.end()) {
            next[i] = it->second;
        } else {
            next[i] = p;
        }
        lastPos[a[i]] = i;
    }
    set<pair<int, int>> times;
    set<int> onFloor;
    int ans = 0;
    for (int i = 0; i < p; i++) {
        if (onFloor.find(a[i]) != onFloor.end()) {
            times.erase({-i, a[i]});
            times.insert({-next[i], a[i]});
        } else {
            ans++;
            if (times.size() >= k) {
                onFloor.erase(times.begin()->second);
                times.erase(times.begin());
            }
            times.insert({-next[i], a[i]});
            onFloor.insert(a[i]);
        }
    }
    cout << ans << '\n';
}
```

На питоне, опять же, предлагается изучить библиотеку `heapq`.

Задача J. Трамвай

Будем идти по остановкам и поддерживать множество сидящих и стоящих людей. Также будем хранить текущее суммарное удовлетворение. Что тогда надо делать при переходе к следующей остановке:

1. Убрать из множеств тех, кто сейчас выходит
2. Добавить всех заходящих людей как стоящих
3. Пока есть люди, которые больше хотят сидеть (и есть сидячие места), надо взять человека с максимальным значением $a_i - b_i$ и переложить его в множество сидящих

Понятно, что это оптимально, так как на каждом шаге мы максимизируем удовлетворенность между остановками. А значит и суммарную тоже. Код на C++:

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int n, m, p;
    cin >> n >> m >> p;
    long long ans = 0;
    vector<vector<int>> events(p + 1);
    vector<int> a(n + 1);
    vector<int> b(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> a[i] >> b[i];
        int c, d;
        cin >> c >> d;
        events[c].push_back(i);
        events[d].push_back(-i);
    }
    set<pair<int, int>> curSitting;
    set<pair<int, int>> curStanding;
    long long cur_ans = 0;
    for (int q = 1; q <= p; ++q) {
        ans += cur_ans;
        for (int i : events[q]) {
            if (i > 0 && b[i] >= a[i]) {
                cur_ans += b[i];
                continue;
            }
            if (i < 0 && b[-i] >= a[-i]) {
                cur_ans -= b[-i];
                continue;
            }
            if (i < 0) {
                i = -i;
                if (curSitting.count({a[i] - b[i], i})) {
                    curSitting.erase({a[i] - b[i], i});
                    cur_ans -= a[i];
                }
                else {
                    curStanding.erase({a[i] - b[i], i});
                    cur_ans -= b[i];
                }
                continue;
            }
            if (i > 0) {
                curStanding.insert({a[i] - b[i], i});
                cur_ans += b[i];
            }
        }
```

```
        while (curSitting.size() < m && !curStanding.empty()) {
            auto it = prev(curStanding.end());
            int i = it->second;
            cur_ans += a[i] - b[i];
            curSitting.insert(*it);
            curStanding.erase(it);
        }

        if (curStanding.empty())
            continue;

        while (curSitting.begin()->first < prev(curStanding.end())->first) {
            auto sittingPair = *curSitting.begin();
            auto standingPair = *prev(curStanding.end());
            cur_ans += b[sittingPair.second] - a[sittingPair.second];
            cur_ans += a[standingPair.second] - b[standingPair.second];
            curSitting.erase(curSitting.begin());
            curStanding.erase(prev(curStanding.end()));
            curSitting.insert(standingPair);
            curStanding.insert(sittingPair);
        }
    }
    cout << ans;
    return 0;
}
```