# Solution 2

## Initial solution (TOO SLOW)

| What I wrote first | Why it hurt |
|---|---|
| `unordered_set<pair<int,int>> seen` – test "visited?" | Hashing every coordinate (≈ 1 000 000 times) + pointer chasing. |
| `unordered_map<pair<int,int>, int> coord2id` – map cell → island-id | Same cost again for every insertion / lookup. |
| `unordered_map<int,int> id2paint` – island-id → picture count | Extra hash layer for something that is really an array. |
| `string` grid → `museum[i][j] = s[j]` | Millions of bounds checks on short strings. |

On the maximal 1000 × 1000 board the BFS visits ≈ $10^6$ cells.

Each visit did **three** hash operations and several heap indirections – more than 30 million hash probes total.

Run-time ➜ ~1.3 s (local) → **TLE** on the judge.

---

## Better solution (ACCEPTED)

1. **One dense 2-D array instead of two hash tables**

```
vector<vector<int>> id(n, vector<int>(m, -1)); // −1 = not visited
```

- `id[x][y] == −1` ➜ "not seen yet" (so `seen` set deleted).
- The value itself is the island number (so `coord2id` deleted).

1. **Plain vector for island data**

```
vector<long long> paint;          // push_back once per island
```

1. **Flood-fill** each island once, store its picture count in `paint[id]`.
2. **Answer queries** in O(1):

```
cout << paint[ id[x-1][y-1] ] << '\n';
```

1. Moved the `dx[]`, `dy[]` arrays outside the loop to avoid rebuilding them.

---

## Why it worked

| metric | before | after |
|---|---|---|
| hash look-ups per cell | 3–4 | 0 |
| peak heap | ≈ 30 MB | ≈ 6 MB |
| run-time 1000×1000 | > 1 s (TLE) | ≈ 0.02 s (AC) |

Array indexing is two orders of magnitude cheaper than hashing and has perfect cache locality. Once the big hash tables were gone, the exact same algorithm (BFS, $O(n\,m + k)$) flew under the time limit with plenty of margin.