# 4 Batched Rendering

You are a graphics programmer trying to optimize your code. You want to use a technique called *batched rendering*, which means sending multiple objects grouped together in a batch to the GPU at a time to be drawn. Your task for this problem is, given a number of texture units and a list of objects each with its own `meshID` and `textureID`, to find the minimum number of batches you can group the objects into.

Ordinarily, this would be easy: because of the way this technique works, objects with the same mesh can be batched together. This means that the minimum number of batches you can group objects into is the number of unique meshes among all the objects. For example, if you had a data set that looked like this:

```
{{meshId = 4}, {meshId = 6}, {meshId = 2}, {meshId = 4}, {meshId
= 6}, {meshId = 4}, {meshId = 2}, {meshId = 4}, {meshId = 2},
{meshId = 4}}
```

The minimum number of batches you could group the objects into would be 3, because there are only three unique meshes across the whole set of objects (meshIDs 2, 4, and 6).

The GPU has a limitation that makes this more complicated than that, though: the GPU has a limited number of *texture units*, or slots in which it can hold the texture of something that's currently being drawn. This means whereas we would ideally be able to batch all objects that have the same mesh, instead, if one group of objects with the same mesh has more unique textures than the GPU has texture units, we cannot render them all in the same batch. For example, if our GPU has 2 texture units and we have a data set that looks like this:

```
{{meshId = 1, textureId = 1}, {meshId = 1, textureId = 2},
{meshId = 1, textureId = 3}, {meshId = 1, textureId = 5},
{meshId = 1, textureId = 2}, {meshId = 1, textureId = 4}}
```

Even though they all have the same mesh, we have to use 3 batches to render them all, because there are 5 unique textureIDs (1, 2, 3, 4, and 5) but only two texture units

## 4.1 Input

Your input will come through `stdin`, meaning that you will have to use `cin` to obtain your input. The input data will come in the following sequence:

1. `nTexUnits`: an unsigned integer representing the number of texture units on the GPU.
2. `nObjects`: an unsigned integer representing the number of objects in the scene.
3. `meshIDs`: list of length `nObjects` that stores the mesh IDs for each object such that the mesh ID of object $i$ is `meshIDs[i]`.
4. `textureIDs`: list of length `nObjects` that stores the texture IDs for each object such that the texture ID of object $i$ is `textureIDs[i]`.

## 4.2 Input Constraints

$$1 \leq \texttt{nTexUnits} \leq 16$$
$$0 \leq \texttt{nObjects} \leq 255$$
$$1 \leq [\text{Any given meshID}] \leq 255$$
$$1 \leq [\text{Any given textureID}] \leq 255$$

## 4.3 Output

Your output will come through `stdout`, meaning that you will have to `cout` your output. Your program should output one integer equal to the minimum number of batches required to render the input set. Nothing else should be in your output; do not print `std::endl`.

## 4.4 Sample Input/Output

Given the following inputs, we'd expect the following output:

| **Sample Input** | **Sample Output** |
|---|---|
| 3<br>20<br>4 2 1 4 1 2 2 3 2 2 3 2 3 4 4 1 1 4 4 4<br>1 2 3 4 5 2 2 6 3 3 7 3 8 9 10 11 12 13 14 15 | 7 |

Here's how to parse the inputs:

```
nTexUnits = 3
nObjects = 20
meshIDs = {4, 2, 1, 4, 1, 2, 2, 3, 2, 2, 3, 2, 3, 4, 4, 1, 1, 4, 4, 4}
textureIDs = {1, 2, 3, 4, 5, 2, 2, 6, 3, 3, 7, 3, 8, 9, 10, 11, 12,
13, 14, 15}
```

The correct output would be **7** batches.

## 4.5 Explanation

We know that since only objects of the same mesh can be batched, we need at least one batch per unique meshID. There are 4 unique meshIDs in the set: 1, 2, 3, and 4. Therefore, we can group our set based on meshIDs:

| **meshID** | **textureIDs** |
|---|---|
| 1 | 3 5 11 12 |
| 2 | 2 2 2 2 3 3 3 |
| 3 | 6 7 8 |
| 4 | 1 4 9 10 13 14 15 |

We also know that if one of these groups shares more than 3 unique textureIDs between them, we can't group them in the same batch. We can thus split them into groups that contain a maximum of 3 unique textureIDs:

| **meshID** | **textureIDs** | **# of groups** |
|---|---|---|
| 1 | (3 5 11) (12) | 2 |
| 2 | (2 2 2 3 3 3) | 1 |
| 3 | (6 7 8) | 1 |
| 4 | (1 4 9) (10 13 14) (15) | 3 |

The sum of the number of groups, 7, is the minimum possible number of batches such that all members of one batch share the same meshID and there are no more than three unique textureIDs within each batch.