Block Model Compression Algorithm

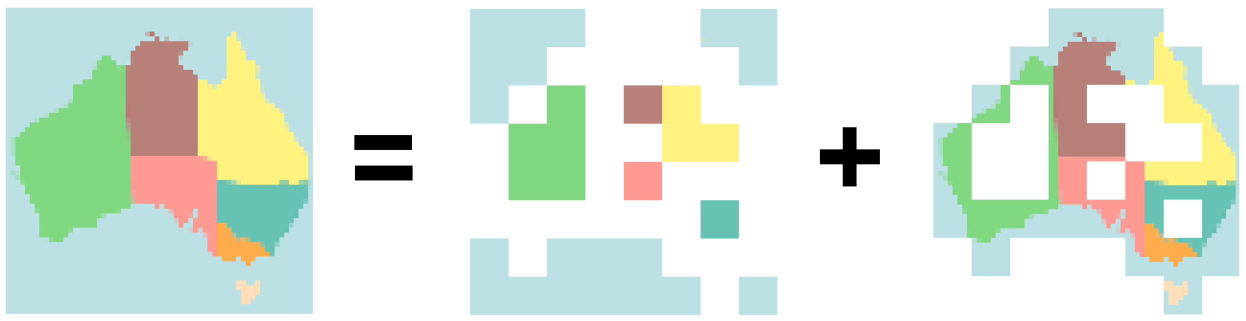
Software Engineering Project 2023, Semester 2

Introduction

This project is presented as a gamified design and implementation challenge. You are required to develop and upload either an .exe file or a Python script to a verification service on a web site. The program you deliver has to take uncompressed input data on standard input and produce compressed output data on standard output with no loss - see below for details. The verification service will execute your code and score it on processing speed and compression performance and put the results on a leaderboard showing all the results from all teams. The group that submits the best performing program in each category by the end of the semester will win the competition in that category. Each group can enter as many times as they wish. The leaderboards will always reflect your best entry to date so you can experiment with different techniques without losing your place on the leaderboard.

Algorithm Background

Images that do not contain many colours can be compressed by grouping neighbouring pixels into larger *parent* pixels of the same colour, as can be seen in the first term of the sum below. Where this is not possible, the remaining parent-sized regions, as shown in the last term, can be encoded as collections of rectangular regions of uniform colour that pack to perfectly fill each region.



This technique is especially effective for low colour resolution images in 3D. Such structures are commonly used to represent geological domains in the Earth's crust and are known as **block models**. This project involves developing algorithms for such compression on very large block models and can explore GPU as well as CPU techniques.

Functional Requirements Detail

Input

Input block models are provided via a text stream on standard input.

The first line contains 6 comma separated natural numbers:
x_count, y_count, z_count, parent_x, parent_y, parent_z

The first three (*x_count, y_count, z_count*) specify the number of columns, rows and slices in the block model respectively, much like the width and height in pixels of an image but with an additional Z dimension for the number of slices there are or *depth*. The second three (*parent_x, parent_y, parent_z*) specify the parent block size in each of these dimensions to use for compression.

For example, the first line in the stream that defines the small map of Australia above is:

```
64,64,1,8,8,1
```

This indicates that it's a width=64, height=64, depth=1 model and that it needs to be compressed using a 8x8x1 parent block scheme.

The next n lines of the stream specify *tag, label* pairs, one per line, defining all characters that will be seen in the remainder of the stream and what labels they correspond to. This is known as the **tag table**. The end of the tag table is marked with a single blank line.

For example, the tag table for the small map of Australia above appears next on the stream as:

```
o, sea
w, WA
n, NT
s, SA
q, QLD
n, NSW
v, VIC
t, TAS
<blank line>
```

Finally, the individual blocks comprising the model are provided as *x_count* tag characters per line, *y_count* lines per slice *z_count* slices with a blank line separating each slice. The columns in each row are streamed in the positive x direction (left to right order). The rows in each slice are listed in the positive y direction (bottom to top order). The slices in each model are listed in the positive z direction (bottom to top order).

For example, the first eight lines of the small map of Australia appear next on the stream as:

```
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooootttooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooootttoooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooottttooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooottttttooooooooooo
ooooooooooooooooooooooooooooooooooooooooooooooootttootooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

The origin of the block coordinate system is the left lower bottom corner of the block model, so the first block in the stream has its left lower bottom corner at coordinate [0,0,0]. All input blocks are 1x1x1 in size so the block appearing last in the stream has its left lower bottom corner at coordinate [*x_count*-1, *y_count*-1, *z_count*-1].

Compression and Output

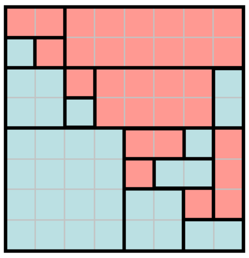
To achieve compression, block sizes greater than one up to the specified parent block size in *x*, *y* and *z* can be specified in the output. Such *sub-blocks* must completely lie within boundaries defined by the parent blocks. Parent blocks always sub-divide the model dimensions without remainder and the boundaries between parent blocks are fixed in place by this subdivision.

The output format contains one line per block and comprises seven comma separated values:
x_position, y_position, z_position, x_size, y_size, z_size, label

The first three values (x_position,y_position,z_position) specify the integer coordinate of the left lower bottom coordinate of the block. The next three values (x_size, y_size, z_size) specify the integer size of the block. The last value is the label applied to that block from the tag table in the input stream.

For example, in the small map of Australia above, the 8x8 salmon coloured parent block (the area inside what would be South Australia), located at 32, 24, 0 can be compressed into a single 8x8x1 block and output as:
`32,24,0,8,8,1,SA`

The parent region immediately below, however, is more complicated consisting of the ‘s’ and the ‘o’ tagged blocks in an irregular pattern. We can visualise this 8x8 region in the image below. One way of compressing it is shown by the thick lines here:



The encoding in this case would be output as (in any order):

```
32,16,0,4,4,1,sea
36,16,0,2,2,1,sea
38,16,0,2,1,1,sea
36,18,0,1,1,1,SA
36,19,0,2,1,1,SA
37,18,0,2,1,1,sea
38,19,0,1,1,1,sea
38,17,0,1,1,1,SA
39,17,0,1,3,1,SA
32,20,0,2,2,1,sea
32,22,0,1,1,1,sea
33,22,0,1,1,1,SA
32,23,0,2,1,1,SA
34,22,0,6,2,1,SA
34,21,0,1,1,1,SA
34,20,0,1,1,1,sea
35,20,0,4,2,1,SA
39,20,0,1,2,1,sea
```

… compressing 64 input blocks into 18 output blocks.

The order of the regions specified in each parent region of the output is not important. However, it is important that the algorithm process a block model in slices of no more than parent block thickness at a time, rather than loading the entire input stream into memory first. This is so that the program can process very large models exceeding the physical memory capacity of the processing machine.

All output models must be valid to qualify for measurement. This means:

- All input blocks are captured by one and only one output block.
- No input blocks undergo a “colour change” - ie. get assigned the wrong label.

Speed will be measured relative to a simple Python script that converts the input stream format to the output stream format with no compression. Compression will be measured as the ratio of output blocks to input blocks.

Assessment and input data constraints

All algorithms will be automatically assessed on standard test data provided. There may also be an assessment component on unseen test data. Small block models as well as large block models will be provided. The size of the tag table will not exceed 256. The block model dimensions in any axis will not exceed 65536. The parent block size in any dimension will not exceed 256 and will typically be ~10.