**Kazeep!**

**the perfect compressor that wont run**

**1- Preface**

**2- Explanation**

**4- Problems**

**Preface**

I been more than a year trying to figure out the best way to make a file smaller, and I been long time working with randomizers and procedural generation. And then I had an idea, why don’t I use randomization as a way to make files smaller?

I spent about a year thinking of a way that I could at least code and put on practice and I finally found what I wanted and now I will talk about it.

**Explanation**

The compressor is really basic, the idea is that you split the file into many smaller parts and then reverse-engineer the seed and position in the seed that have this strip of data.

For this I created 3 main variables: the seed, the offset and the amount of bytes.

1. Seed: the seed for the randomizer, where it found the block of data. (64bit integer for higher number of possibilities)
2. Offset: the offset from the start of the generated data, it points where the generated data start matching the real data. (16bit integer since there’s no reason to keep in same seed for too long)
3. Bytes: The amount of bytes each block of data have. (16bit integer, because even if it will be more compressed.. the higher the block value, the longer will take to find a matching data)

Example by steps:

1. Original data: I am a text
2. The algorithm start with seed 0 and right now the first characters of this string generated with seed 0 looks like this: jm84u8 m37yt53mc9m9m3 39tu
3. Since “I am a text” isn’t there, it will offset for 2000 characters (as max offset is set to 2000 in this example). if it find “I am text” the algorithm stops and save the seed, bytes and offset as instructions for the “decompressor”.
4. In seed 2948103 in offset 536 we got: f4 2I am a textfek3ijj2jf92w
5. We want to get the “I am a text”, so the algorithm will keep doing the offset until it reaches 540 and find the desired text.
6. It will save the instructions to find “I am text” and the size of this text (each character in c++ is a byte).
7. The decompressor takes the seed and start generating data, the offset tells the decompressor when the desired data is inside the generated value and when it finally reach the first value of the data it then start placing on the bytes array.

Now lets imagine we keep the 64bit seed, 32 bit offset and byte size of data strips for the results.

Each instruction will be the size of 16 bytes, so anything above 16 bytes given enough time will eventually compress to only 16 bytes.

The idea of this compressor isn’t really compressing, but giving instructions to a randomizer to find the data by using a seed and offset (the idea of offset was created so we are not dependent on the amount of seeds).

So if we want to compress a file that measures 256 bytes into 16bytes we just create a single instruction with the “bytes” set to 256, but in case we have a huge file like 1gb, we need to split it into multiple smaller parts (like 256 bytes) and the file generated from this algorithm will be a 1/16th of the original size, and since this type of compression isn’t dependent on any other method than seeds we are able to compress the same file over and over again.

We don’t keep the file, we create instructions to re-build it later by using a random number generator.

**Problems**

Now we got to the bad news about this overpowered compressor, it requires too much power. Until we be able to compress a 256 bytes block into a 16byte instruction in less than a second instead of endless more than billions of years, this method of compression can’t be used in practice.

**Conclusion**

Despite how exciting it was to create this project and noticing there are new undiscovered methods for compression and data management, in the current state of technology there is no way to use this algorithm. Hopefully I will be alive to watch technology someday being able to use this, but in the meantime I will be creating more stuff related to optimization over time.

Unfinished source code for the project (stopped developing once realized how long it would take to finish a mere 123byte file): https://github.com/J-K-Tech/kazeep