IPMT

Identificação

A equipe é formada pelos alunos Guilherme Fernandes Xavier da Silva e Matheus Felipe da Silva. Guilherme implementou a compressão e descompressão, onde se foi utilizado o algoritmo lz78 e Matheus implementou a busca exata e indexação, onde se foi utilizado o algoritmo Suffix Array. Os módulos restantes desse projeto foram realizados de maneira conjunta.

Implementação

Suffix Array

Indexação

Nessa etapa do projeto foi implementado o algoritmo suffix array para encontrar os sufixos de uma determinada string. Dividimos o arquivo de texto pela sua quantidade de linhas e para cada uma de suas linhas calculamos o suffix array e o contador de letras naquela determinada linha do texto.

Busca

Após a etapa de indexação estar concluída, passamos para a de busca. Inicialmente reconstruímos o texto inicial a partir de seu array de sufixos e de seu contador de letras. Após isso, o que fazemos então é rodar duas BinarySearch, que vai iterar pelo array de sufixos, a primeira BinarySearch é usada para encontrar a primeira ocorrência do padrão e outra para encontrar a última, a partir disso sabemos quantas vezes o padrão ocorreu no texto, visto que o array de sufixo está ordenado.

LZ78

Compressão

É feito o algoritmo LZ78 usando um dicionário de tamanho no máximo 254, já que no arquivo comprimido é gravado um byte para o index do dicionário e um byte

para o caractere. Assim que é calculado o index do dicionário, é gravado no arquivo para evitar consumir muita memória do computador.

Descompressão

É lido o arquivo comprimido, lendo dois caracteres por vez, para não sobrecarregar a memória do computador, e aí é calculado qual era o caractere ou caracteres antes. Aqui também tomamos cuidado para o tamanho do dicionário não passar de 254, para evitar incongruências com a compressão.

Testes

Os testes foram feitos em um computador com sistema operacional Ubuntu 22.04, processador Intel Core i5 7200U com 2.50 GHz base podendo chegar a 3.10 GHz, memória 8GB 2133MHz DDR4 e SSD. O compilador utilizado foi o GNU GCC v11.2.0. Os códigos foram otimizados com a flag -O3.

LZ78

Arquivos de texto usados:

- English: texto em inglês de 2,2GB, contendo somente palavras em inglês.
- DNA: texto de tamanho 403,9MB, em que cada linha é uma cadeia de DNA, em que se repetem os caracteres A,G,C e T. Cada linha é enorme, com milhares de caracteres por linha.
- Protein: texto de tamanho 1,2GB, em que cada linha é uma cadeia de aminoácidos que compõem uma proteína, em que se repetem 20 caracteres maiúsculos. Cada linha é muito grande com mais de 1000 caracteres.

Tempos de execução e taxa de compressão:

Nome do arquivo	Tempo de compressão	Tempo de descompressão	Tamanho original	Tamanho comprimido	Taxa de compressão
English	10min 54s	3min 18s	2,2GB	1,9 GB	13,6%
DNA	1min 29s	22s	403,9 MB	204,8 MB	49,2%
Protein	5min 50s	1min 41s	1,2GB	1 GB	16,6%

Como pode ser visto na tabela acima, o texto de DNA foi o mais rápido para comprimir e descomprimir, e isso se deve ao fato de ter somente 4 caracteres sendo repetidos, e isso leva a padrões serem repetidos. Também é o arquivo que teve a maior taxa de compressão, mas isso se deve ao fato de ser o menor arquivo. Também pode ser observado que por conta da limitação do tamanho do dicionário ser 254, tivemos um claro limite de quanto podemos comprimir, algo em torno de 200~300MB.

Suffix Array

Arquivos de texto usados:

- English: texto em inglês de 105MB, contendo apenas palavras em inglês sem caracteres especiais.
- Protein: texto de 105MB em que cada linha é uma cadeia de aminoácidos que compõem uma proteína, em que se repetem 20 caracteres maiúsculos Cada linha é muito grande com mais de 1000 caracteres.

Tempos de execução

Nome do Arquivo	Tamanho do Arquivo	Tamanho do Index	Tempo para indexar
English	105 MB	1.3 GB	1 min e 7 seg
Proteínas	105 MB	572 MB	1 min e 45 seg

English

Palavra	Tempo de busca
love	1 min e 30 seg
pathway	1 min e 35 seg
avocado	1 min e 41 seg

Protein

Palavra	Tempo de busca
AIYKQG	57 seg
GFSNLLFSVC	59 seg
RDIPMSDSL	57 seg

Apesar do texto em inglês ter muito mais linhas do que o texto proteínas, o tempo para indexar foi relativamente menor. Isso se dá ao fato de que o texto proteínas contêm muito mais caracteres por linhas do que o texto em inglês.

Em relação ao tempo de busca, os padrões do texto proteínas rodaram bem mais rápido do que os padrões do texto inglês, um fator que fez com que isso acontecesse foi a quantidade de linhas, que é bem menor no texto proteínas.

Foram feitos testes com textos menores, pois o algoritmo consome muita memória, o que ficou claro ao tentar testar em um arquivo de DNA. Nosso computador não conseguiu criar o index, já que a memória RAM mais a memória de swap do ubuntu não eram suficientes.

Conclusão

Sendo assim, em relação ao algoritmo de compressão e descompressão, ficou clara sua limitação em relação à taxa de compressão devido ao tamanho máximo do dicionário. Em uma futura versão, seria implementada uma versão em que o índice máximo do dicionário fosse pelo menos 2 bytes, o que com certeza aumentaria a taxa de compressão. Já em relação ao algoritmo de indexação e busca exata, o algoritmo está consumindo memória demais, essa limitação ficou visível ao testar o arquivo DNA, que tem uma linha muito grande, provavelmente isso se dá ao fato de o algoritmo implementado usando binary trees, o que consome muita memória em linhas muito grandes.