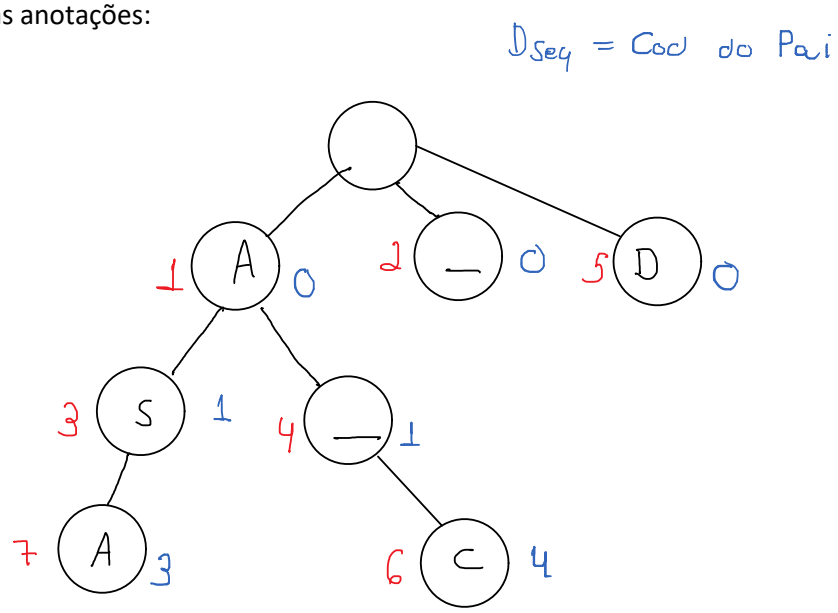
# Documentação

**Nome: Marco Túlio Machado França**
**Prática: Utilizar algoritmo LZ78 para comprimir arquivos.**

**Parte 1:**

Primeiramente tive que entender o algoritmo. Seguindo o link: https://pt.wikipedia.org/wiki/LZ78
Fiz algumas anotações:




A árvore de prefixos ficaria nesse formato. Além disso é importante saber que a cadeia que é salva contém o código do nó pai.
Então, por exemplo, quando o sétimo caractere "A" é adicionado, a saída é (3,'A') exatamente para termos o mapeamento do pai desse caractere e poder trabalhar para reconstruir a string.
Dado isso parti para a construção do código

**Parte 2:**

Eu não tinha familiaridade com python. Até o momento, praticamente tudo que fiz na faculdade foi em c++, mas por ser uma linguagem bem mais simples que c++, por mais que seja tudo novo, decidi fazer em python, e valeu a pena.

Primeiramente trabalhei com a parte de como receber as entradas da linha de comando, depois leitura e escrita em arquivos e por fim, de fato a implementação dos algoritmos.

Utilizando a ideia do algoritmo e a sugestão do professor, criei uma função para varrer o arquivo uma vez, já inserir na trie e verificar quando bytes seriam necessários para armazenar todos os inteiros. Então, por exemplo, se Dseq = 1000, que é maior 2^8 e menor que 2^16, seriam necessários 2 bytes para salvar os inteiros que representam os códigos dos nós.
Com essa ideia já deu para comprimir bem os arquivos. Então, depois de passar uma vez verificando, eu zero a trie e crio novamente, agora escrevendo no arquivo binário. Tive problemas ao converter caracteres que tinham tamanho em bytes maior que 1, apareceram só de 2 e 3 bytes, e a estratégia que utilizei para driblar essa situação foi, toda vez que encontrar um caractere que em bytes terá tamanho igual 2 vou inserir o caracter ~ (til) e se o tamanho for igual a 3, inserir ^ (circunflexo) no arquivo binário antes de inserir o caractere. Assim, na hora de descomprimir o código, sempre que encontrar um dos dois caracteres saberemos quantos bytes teremos que ler. Utilizei esses dois caracteres pois é muito improvável de tê-los em um texto sozinhos. Em todos os testes que fiz deu certo.
O trabalho ficou com três arquivos .py, a main.py que tem a parte de tratar a entrada do terminal e executar o programa em si. Trie.py que tem a estrutura para compactar o arquivo e as funções que precisei implementar para tratar o texto. E, finalmente, o arquivo descomprimir.py, onde coloquei algumas funções que precisei utilizar para descomprimir o arquivo .z78.

**Parte 3:**

Feito o algoritmo criei alguns cenários de teste além dos que o professor já havia fornecido.
Os resultados foram os seguintes:

| | tamanho .txt (KB) | tamanho .z78 (KB) | Taxa de Compressão |
|---|---|---|---|
| Ex01 | 95,5 | 61,2 | 35,92% |
| Ex02 | 102 | 66,3 | 35,00% |
| Ex03 | 43,9 | 27,9 | 36,45% |
| Ex04 | 190 | 91,3 | 51,95% |
| Ex05 | 222 | 116 | 47,75% |
| Ex06 | 147 | 82,6 | 43,81% |
| Ex07 | 169 | 98,8 | 41,54% |
| Ex08 | 1930 | 1447 | 25,03% |
| Ex09 | 1962 | 1465 | 25,33% |
| Ex10 | 1666 | 1282 | 23,05% |

**Conclusão:**

O trabalho foi uma excelente oportunidade de concretizar os conhecimentos teóricos e de aprender novas tecnologias. Foi interessante entender como funciona a compressão de arquivos, não só por conta de usar um algoritmo que ajude a comprimir como é o caso do LZ78 mas também de ter que escrever os arquivos binários e ter que tratar, dependendo do tamanho dos bytes no caso dos caracteres.