## Yleiskuva

Tuottamassani ohjelmassa vertaillaan kahta klassikkopakkausalgoritmia, Huffman-koodausta ja LZ-78-algoritmia. Ohjelma toimii niin, että käyttäjä voi harkintansa mukaan valita joko käyttävänsä jo valmiiksi tuotettuja testimateriaaleja, tai vaihtoehtoisesti kirjoittaa itse merkkijonon tai tekstinpätkän. Testimateriaali tämän jälkeen syötetään algoritmeihin, jonka jälkeen saadaan tulos pakkauksen tehokkuudesta. Python-kielen luonteen mukaan pakkausnopeutta ei mitata, vaan absoluuttista pakkauksen pienentämistä sekä tarkistetaan, että alkuperäinen tiedosto vastaa pakattua ja avattua tiedostoa.

## Huffman-koodaus

Huffman-koodauksen ydinidea on tuottaa binääripuu, johon jokaisen tekstissä esiintyvän merkin esiintymismäärä tallennetaan. Useimmin esiintyvät merkit saavat lyhyen binäärikoodin harvemmin esiintyvät pidemmän koodin. Syöte käydään läpi ensin kerran läpi ajassa O(n), ja jokaisen merkin esiintymiskertojen määrä lasketaan. Tämän jälkeen Huffman-puu kasataan ensin muodostamalla minimikeko, jonka jälkeen puu kasataan. Tämä vie aikaa O(k log(k)), jossa k on merkkien määrä. Syöte käydään lopuksi vielä uudestaan läpi taas ajassa O(n), jolloin merkit korvataan niille määritetyillä koodeilla. Kokonaisuudessaan pakkaus vie aikaa O(n+k log(k))

Datan purku on suhteellisen nopeaa, Huffman-puun avaaminen saadaan ajassa O(n). Binääripuun avulla algoritmin toimintaa on myös helppo havainnollistaa. Huffman-koodaus ei hyödynnä toistuvia merkkijonoja, joka on sen heikkous.

## LZ78-algoritmi

LZ78-algoritmi toimii muodostamalla sanakirja, johon algoritmin edetessä lisätään jokaisen merkin ensimmäinen ilmentymä muodossa (index, symbol). Syöte käydään läpi merkki merkiltä. Jos merkki on jo sanakirjassa, algoritmi lisää sen indeksin ja etenee. Kun koko merkkijono tai tiedosto on käyty läpi, ylimääräiseksi jääneet merkit lisätään sanakirjan loppuun. Tämän myötä on saatu muodostettua sanakirja, joka on muotoa (index, symbol).

Purkaminen alkaa siten, että alustetaan ensin tyhjä sanakirja. Tämän jälkeen käydään läpi pakattu tiedosto. Alkuperäinen teksti saadaan muodostettua (index, symbol) pareista, lukemalla ensin symboli ja tarkistamalla sen indeksi.

Koko algoritmin aikavaatimus on O(n). Jokainen merkki käydään läpi vain kerran purkamisessa ja pakkauksessa. LZ78-algoritmin vahvuus on, kun data on toistuvaa. Suurella toistuuvuudella data saadaan pakattua hyvin pieneen määrään alkuperäisestä tilasta.

## Puutteet ja parannusehdotukset

Työn puuttelliset osat muodostuvat pääasiassa omasta tietämättömyydestä pakkausalgoritmeja kohtaan sekä algoritmien ydinideoiden ymmärtämisen haasteellisuudesta. En myöskään ollut koodannut mitään algoritmeja pitkään aikaan, joten lähestymistapa ongelmia kohtaan oli ruosteessa.

## Kielimallien käyttö

Käytin kielimalleja koodin syntaksin parantamiseen, tiedonhakuun sekä ohjaamaan tuotantoprosessia. Käytin OpenAI:n GPT-5 ja GPT-4.5 kielimalleja. Kysyin myös niiltä, kuinka toteuttaa tietynlaisia funktioita ja miten esimerkiksi bitit toimivat. Hyödynsin niitä myös testauksen ideoinnissa, sillä tämäntyylinen testaus ei ollut minulle alun perin tuttua.

# Lähteet

https://en.wikipedia.org/wiki/Data_compression
https://www.geeksforgeeks.org/electronics-engineering/what-are-data-compression-techniques/
https://algocademy.com/blog/understanding-data-compression-algorithms-a-comprehensive-guide/
https://en.wikipedia.org/wiki/Huffman_coding
https://en.wikipedia.org/wiki/LZ77_and_LZ78