**Manus Huffman coding**

**Lysbilde 2:** Hva er Huffman encoding?

Huffman-koding er en mye brukt algoritme for tapsfri datakomprimering. Den ble utviklet av David A. Huffman i 1952. Huffman-koding er spesielt effektiv for å komprimere tekst eller andre data med en uensartet sannsynlighetsfordeling, der noen symboler (som bokstaver i et tekstdokument) forekommer hyppigere enn andre.

Den grunnleggende ideen bak Huffman-koding er å tilordne kortere koder til oftere forekommende symboler og lengre koder til sjeldnere forekommende symboler. Dette resulterer i et prefikskodeskjema med variabel lengde, der ingen kode er et prefiks til en annen kode. En slik egenskap tillater entydig dekoding av de komprimerte dataene.

**Lysbilde 3:** Oppsett av kode

Vi har valgt å implementere Huffman komprimering av bilder, koden kan ikke komprimere tekstfiler osv. Både Huffman encoding og decoding er implementert i koden.

Koden er avhengig av en tilpasset nodeklasse for å representere noder i Huffman-treet og en bildeklasse for å håndtere bilderelaterte operasjoner. Bildeklassen inneholder metoder for lesing, koding, dekoding og skriving av bilder ved hjelp av Huffman-koding.

**Lysbilde 4:** Steg i Huffman coding

Gå gjennom liste

**Lysbilde 5:** Telle frekvensantallet

Først teller algoritmen frekvensen til hvert symbol i inndataene, enten det er tegn i en tekstfil eller andre symboler. Dette utføres i funksjonen initialise som initialiserer bildedataene ved å flate ut bildet, legge til dets dimensjoner og lage et histogram.

**Lysbilde 6:** Lage en prioritetskø

Sorter tegnene i økende rekkefølge etter frekvensen. Disse lagres i en prioritetskø. Hvert symbol og dets tilknyttede frekvens blir behandlet som en bladnode i et binært tre.

I vår kode er ikke prioritetskøen eksplisitt opprettet i koden. I stedet brukes listen self.allNodes til å lagre alle nodene som representerer intensiteter, og denne listen blir senere sortert etter sannsynlighet for å simulere prioritetskøen.

**Lysbilde 7:** Bygge Huffman-tre

Mens det er mer enn én node i prioritetskøen, fjerner algoritmen gjentatte ganger de to nodene med de laveste frekvensene, kombinerer dem til en ny intern node med en frekvens lik summen av de to nodenes frekvenser, og setter inn denne nye noden tilbake i prioritetskøen. Denne prosessen fortsetter til det bare er én node igjen i køen, som blir roten til Huffman-treet.

Konstruksjonen av Huffman-treet skjer i upTree-metoden. Den slår sammen noder med de laveste sannsynlighetene til en enkelt rotnode er opprettet. Rotnoden til Huffman-treet er lagret i self.root-attributtet.

**Lysbilde 8:** Tilordne koder

Gå gjennom Huffman-treet fra roten til hver bladnode, og tilordne en binær kode til hvert symbol. Venstre grener i treet tilsvarer å legge til en "0" til koden, og høyre grener tilsvarer å legge til en "1". Siden ingen kode er et prefiks til en annen kode, er de resulterende kodene entydige.

Tildeling av Huffman-koder til bladnoder skjer i downTree-metoden. Den tildeler binære koder til hver bladnode basert på deres plassering i Huffman-treet.

**Lysbilde 9:** Komprimere data

Erstatt hvert symbol i inndataene med dens tilsvarende Huffman-kode for å lage de komprimerte dataene.

Koding av bildedata til en binær streng ved hjelp av Huffman-koder utføres i huffmanAlgo-metoden. Den går gjennom Huffman-treet for å kode bildedataene, inkludert dimensjoner og pikselverdier, og lagrer den kodede strengen i self.encodedString-attributtet.

Til slutt skrives de komprimerte dataene til en fil. Å skrive de kodede binære dataene til en fil gjøres i sendBinaryData-metoden. Den konverterer den binære strengen i self.encodedString til et binært format og skriver den til en fil.

**Lysbilde 10-12:** Dekomprimere data

For å dekode de komprimerte dataene, traverserer du Huffman-treet fra roten, og bruker hver bit i de komprimerte dataene til å navigere i treet til en løvnode er nådd, hvor det opprinnelige symbolet rekonstrueres.

Først leses de komprimerte dataene fra filen som ble laget ved komprimering av dataen. Lesing av de kodede binære dataene fra en fil og dekoding utføres i dekodemetoden. Den leser de binære dataene, dekoder dem ved hjelp av Huffman-treet og lagrer de dekodede verdiene i self.decodeList.

Rekonstruering av bildet fra de dekodede verdiene gjøres i decodeIm-metoden. Den trekker ut dimensjonene og intensitetsverdiene fra self.decodeList og rekonstruerer utdatabildet.