# Dokumentácia projektu predmetu KKO

## 1. Úvod

Zadaním projektu bolo zoznámiť sa a naimplementovať algoritmus vykonávajúci adaptívne Huffmanovo kódovanie. Jedná sa o variantu Huffmanovho kódu, kde berieme do úvahy fakt že na začiatku prenosu dát nepoznáme pravdepodobnosti výskytov jednotlivých symbolov a preto zostavujeme strom v priebehu kódovania.

Ako implementačný jazyk bol zvolený C++.

## 2. Adaptívne Huffmanovo Kódovanie

Metóda klasického Huffmanoveho kódovanie je pre prax zo príliš pomalá, čo vyplíva z nutnosti vypočítať pravdepodobnost jednotlivých symbolov a na ich základe zostaviť strom, podľa ktorého sa kóduje. Adaptívne Huffmanovo kódovanie je založené na budovaní stromu za behu a spracovania dát. Tento strom je udržovaný v rovnakom stave na obidvoch stranách prúdu. Neodoslané symboly nemaju na počiatku žiadne kódy. Keď sa symbol poprvýkrát objaví na vstupe, je odoslaný v nekódovanej forme a pridaný do stromu. V tomto momente už k nemu existuje kód, ktorý sa však za behu môže zmeniť. Ak symbol už v strome je, zvýši sa počítadlo jeho početnosti a je overené, či je strom Huffmanovský, ak nie, dôjde k preusporiadaniu elementov stromu tak, aby táto vlastnosť bola zachovaná. Otázkou zostáva, ako oznámiť dekompresoru, že odosielaný znak je v binárnej forme. Na toto sa využíva špeciálny symbol variabilnej dĺžky. Toto je riešené pridaním listu s nulovou početnosťou do stromu, ktorého kód sa odošle pred binárnimi dátami ( NYT list ).

Pre Huffmanov stom, platí že jeho elementy spĺňajú tzv. sibling property, čo znamená, že elementy nachádzajúce sa v strome vľavo a dole, musia mať nižšiu početnosť výskytu než tie čo ich nasledujú. Táto vlastnosť zaručuje, že symboly s vyššou početnosťou majú kratšie kódy.

Toto je zaručené tak, že po pridaní listu, alebo zmene váhy, sa skontroluje početnosť nasledujúceho symbolu v súrodeneckom usporiadaní. Ak je táto rovná, či vysššia je uzol(podstom) presunutý na jeho miesto. Táto vlastnosť je dalej overená pre rodičov uzlu, u ktorého zmena nastala.

## 3. Implementácia

### *3.1 Vstup a výstup*

Nakoľko metódy vstupu a výstupu štandardnej knižnice jazyka c a POSIX používajú bajty ako elementárne jednotky vstupu a výstupu bolo nutné naimplementoať bitové vstupy a výstupy. Na toto v kóde existujú triedy bitstreamreader a bitstreamwritter, ktoré obalujú typ FILE * a poskytujú metódy *getBit()*, či *putBit()* a ich varianty pre zápis väčšieho počtu bitov.

### *3.2 Strom*

Elementy stromu sú štruktúrovaného typu AHTreeNode, ktorý zapuzdruje všetky potrebné dáta t.j.: váhu, index v usporiadaní, symbol, etc..

Samotný strom je objekt typu AHTree, ktorý poskytuje metódy pre vloženie a zmenu váhy symbolu v strome ( *updateTree(uint32_t chr)* ), zápis kódu symbolu na výstup ( *outputPathForChar( uint32_t chr, bitstreamwriter * bsw )* ), či zápis kódu špeciálneho symbolu indikujúceho binárne dáta ( *outputZeronode( bitstreamwriter * bsw )* ) a nakoniec aj metódu pre nájdenie symbolu podľa binárnych dát ( *getNodeForInput( bitstreamreader * bsr )* ).

Okrem týchto metód trieda stromu obsahuje väčší počet privátnych funkcii, ktoré vykonávajú konkrétne časti algoritmu.

## 4. Fungovanie programu

Na začiatku je nutné stom zkonštruovať volaním bezparametrického konštruktora. Daľej je nutné zabaliť vstupný/výstupný FILE object do bit wrappera. Následne je možné fungovanie popísať

takto:

Kóder:

1. Dotáž sa stomu či existuje symbol ( *hasChar()* ), ak symbol neexistuje zapíš na výstup kód NYT listu a následne zaňho symbol v binárnej forme a pokračuj krokom 1. ak neexistuje choď na 3.
2. Zapíš na výstup kód symbolu
3. Uprav strom ( *updateTree(uint32_t chr)* )

Dekóder:

1. Nájdi list v strome podľa vstupných bitov
2. Ak sybol indikuje NYT načítaj zo vstupu binárny kód symbolu, ináč jeobsahuje nájdený list symbol s daným kódom.
3. Uprav strom

Metóda updateTree(), ktorá sa stará o pridanie listu do stromu, či prípadnú zmenu jeho váhy funguje takto:

1. Ak nie je sybol v strome, vytvor dva nové listy na pozícii NYT listu a to nový NYT a list obshujúci nový symbol a pokračuj s rodičom týchto uzlov. Ak symbol v strome je over, či je splnená sibling property ( privátna metóda *_enforceSiblingProperty()* ). Táto metóda pri narušení volá metódu _swapNodes(). Pokračuj s rodičom daného prvku.
2. Prechádzaj stromom smerom ku koreňu a zvyšuj váhy elementov o 1. Následne over či je splnená požadovaná vlastnosť ( *_enforceSiblingProperty()* ).
3. Ak bolo zistené, že je nutné zmeniť rozsahy váh, volaj *_rescaleTree()*

**5.Záver**

Po implementácii programu a jeho otestovaní bolo zistené, že naozaj dochádza ku kompresii vstupných dát. Pomer vstupných dát ku komprimovaným ból 768771 bytov k 438628 bytov.