Úkolem je vytvořit sadu C/C++ funkcí, které dokáží dekomprimovat a komprimovat vstupní soubor Huffmanovým kódem. Úloha nabízí různé stupně obtížnosti:

- povinná část úlohy požaduje pouze funkci pro dekompresi, navíc v rozbalených datech se nachází pouze ASCII znaky (0-127),
- nepovinné testy zkoušejí správnost dekomprese, v rozbalených datech jsou texty v kódování UTF-8,
- bonusové testy zkoušejí správnost komprese i dekomprese, komprimují/dekomprimují se soubory s texty v kódování UTF-8.

Huffmanův kód je princip komprese dat, který využívá statistických vlastností ukládaných dat. V typických souborech jsou různé hodnoty (např. znaky) zastoupeny s velmi nerovnoměrnou četností. Například mezery jsou v typickém textu velmi časté, naopak znak ř bude málo frekventovaný. Huffmanův kód zpracuje analýzu četnosti výskytu jednotlivých znaků a podle četností přidělí jednotlivým znakům kódy. Kódy mají různou délku, typicky 1 až desítky bitů. Často se vyskytující znaky dostanou kódy kratší, málo časté znaky dostanou kódy delší. To v důsledku vede k úspoře místa.

Zavedením různé délky kódů pro jednotlivé znaky se ale objeví jiný problém. Musíme být schopni detekovat, kde kódovaný znak končí, tedy kolik bitů je potřeba načíst, abychom správně dekódovali právě jeden znak. Pro fixní počet bitů na znak je to snadné. Např. ASCII má 8 bitů/znak, tedy co bajt, to znak. UTF-8 je složitější, jeden znak zabírá 1 až 4 bajty a je potřeba kontrolovat strukturu čtených bajtů, aby kód čtené UTF-8 bajty správně seskupil. U Huffmanova kódu je to ještě obtížnější. Čtečka zpracovává jednotlivé bity a podle podoby načtených bitů pozná, kdy má skončit. Huffmanův kód je kódem prefixovým, tedy žádný kód není prefixem jiného kódu. Pokud například mezeru kódujeme dvojicí bitů 00, pak posloupnosti 001, 000, 0001, ... nejsou obsazené žádným jiným kódem. Tím je garantována jednoznačnost dekódování.

Druhým problémem je doplnění na celé bajty a správná detekce posledního znaku při dekódování. Protože kódy mohou mít různé délky, nemusí být počet bitů po komprimaci násobkem 8. Tedy v posledním bajtu mohou být některé bity nevyužité. V souborech ale musíme pracovat s celými bajty, tedy zbývající bity musíme nějakými nulami nebo jedničkami doplnit. Při ukládání to není problém (prostě něco přidáme), ale při načítání bychom takto přidané bity navíc mohli dekódovat jako další znaky navíc. Proto ukládaná data rozdělíme do úseků (chunks), pro které bude známá jejich délka. Na začátku každého chunku uložíme indikátor:

- jeden bit s hodnotou 1. Tento bit indikuje, že následuje chunk ukládající 4096 znaků (kódovaných Huffmanovým kódem),
- jeden bit s hodnotou 0 následovaný 12 bity, které udávají délku chunku (tedy délka chunku bude 0 až 4095 znaků). Takto bude v souboru označen poslední chunk, aby bylo jasně určeno, kde skončit s dekompresí (poslední chunk může kódovat 0 znaků, pokud vstupní soubor obsahoval počet znaků, který je násobkem 4096).

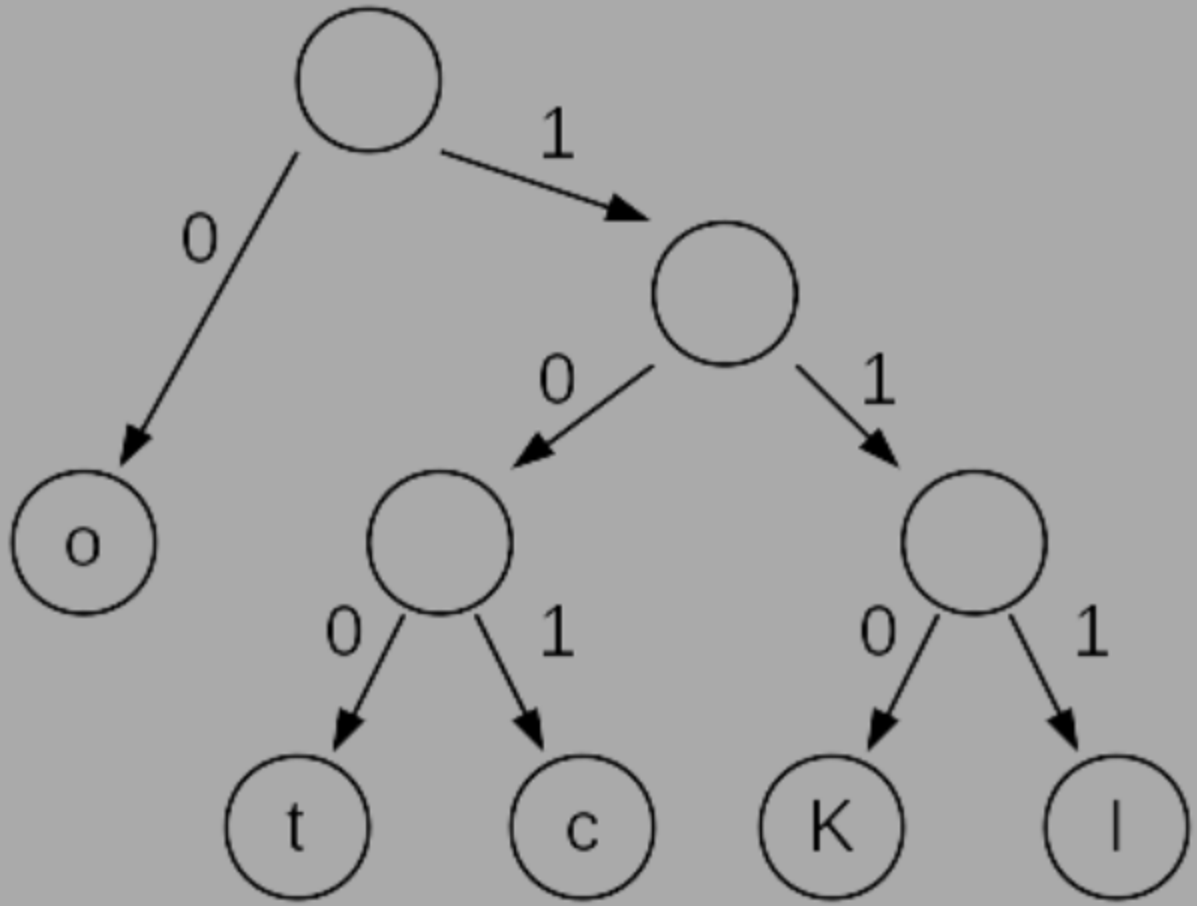
Například pokud bylo v souboru 15800 znaků, budou chunky uložené následovně:

```
1 <bity chunku kódující znaky 0 až 4095>
1 <bity chunku kódující znaky 4096 až 8191>
1 <bity chunku kódující znaky 8192 až 12287>
0 110110111000 <bity chunku kódující znaky 12288 až 15799>
<doplnění na celé bajty>
```

Kódování a dekódování si ukážeme na příkladech:

```
vstupní soubor obsahuje 7 znaků:
Kolotoc

Pro tento soubor by Huffmanův kód mohl být například:
K       110
o       0
l       111
t       100
c       101

Tedy po zakódování dostaneme posloupnost bitů:
K   o l   o t   o c
110 0 111 0 100 0 101

Protože pro takový vstup by byl pouze jeden chunk (byl by zároveň poslední),
předcházelo by vlastním kódům indikátor posledního chunku (bit 0) a počet znaků
(7 znaků jako 12 bitové číslo, tj. 000000000111):

0 000000000111 110 0 111 0 100 0 101
```

Komprimace zkrátila původních 7 bajtů na 4, tedy na cca 50%. Aby bylo možné soubor dekomprimovat, je ale potřeba znát, jaký kód byl použit. Dekódovací tabulku lze např. uložit na začátek souboru před vlastní datový obsah (před první chunk). Pro ukládání a dekódování se hodí udržovat tabulku ve formě stromu. Pro ukázkový příklad je strom uveden na obrázku níže.



Takový strom lze do souboru uložit například průchodem pre-order. Za každý navštívený uzel zapíšeme do souboru jeden bit. Vnitřní uzel zapíšeme bitem 0, list zapíšeme jako bit 1 následovaný hodnotou kódovaného znaku. Kódovaný znak zabere 8 bitů (ASCII) případně 8 až 32 bitů (UTF-8). Písmena v ukázce jsou z ASCII, tedy zaberou 8 bitů každé. Ukázkový strom bude uložen jako:

```
Pre-order průchod stromem:
0 1 'o' 0 0 1 't' 1 'c' 0 1 'K' 1 'l' <chunks>

Náhrada znaků jejich ASCII/UTF-8 kódy:
0 1 01101111 0 0 1 01110100 1 01100011 0 1 01001011 1 01101100 <chunks>

Pro ukázkový text (bitová posloupnost z minulého odstavce):
0 1 01101111 0 0 1 01110100 1 01100011 0 1 01001011 1 01101100
0 000000000111  110 0 111 0 100 0 101

Seskupení bitů do bajtů:
01011011 11001011 10100101 10001101 01001011 10110110
00000000 00011111 00111010 00101xxx

Doplněno na celé bajty nulovými bity:
01011011 11001011 10100101 10001101 01001011 10110110
00000000 00011111 00111010 00101000

Přepis na bajty (10 bajtů, ukázkový soubor test0.huf):
5b       cb       a5       8d       4b       b6
00       1f       3a       28
```

Tedy původní text o délce 7 bajtů se zkomprimuje na novou délku 10 bajtů. Nárůst je ale dán pouze uložením kódovacího stromu, pro delší vstup by již bylo dosaženo úspory.

Při dekompresi je potřeba nejprve načíst serializovaný kódovací strom. Načítání je rekurzivní. Rekurzivní funkce čte jeden bit ze souboru. Pokud načte bit 0, vytvoří vnitřní uzel a 2x se rekurzivně zavolá pro levý a pravý podstrom. Pokud načte bit 1, načte ještě následující ASCII/UTF-8 znak a vytvoří pro něj odpovídající listový uzel.

Huffmanův strom lze s výhodou použít i pro dekompresi. Po načtení stromu stačí z komprimovaného souboru číst jednotlivé bity a podle hodnoty 0/1 procházet strom vlevo/vpravo. Po dosažení listu máme dekódovaný jeden znak a začínáme znovu od kořene.

Při dekompresi je samozřejmě potřeba pracovat s chunky. Délku chunku lze vždy načíst do proměnné, touto proměnnou pak řídit počet iterací cyklu načítající daný chunk. Zpracováním posledního znaku posledního chunku dekomprese končí.

Huffmanův strom lze s výhodou použít i pro dekompresi. Po načtení stromu stačí z komprimovaného souboru číst jednotlivé bity a podle hodnoty 0/1 procházet strom vlevo/vpravo. Po dosažení listu máme dekódovaný jeden znak a začínáme znovu od kořene.

Při dekompresi je samozřejmě potřeba pracovat s chunky. Délku chunku lze vždy načíst do proměnné, touto proměnnou pak řídit počet iterací cyklu načítající daný chunk. Zpracováním posledního znaku posledního chunku dekomprese končí.

Úkolem je realizovat dvě funkce s rozhraním níže. Obě funkce mají parametrem dvě jména souborů: zdrojový a cílový. Funkce čtou zdrojový soubor a zapisují výsledek komprimace/dekomprese do cílového souboru. Návratovou hodnotou obou funkcí je příznak úspěchu (`true`) nebo chyby (`false`). Pokud se během požadované operace komprimace/dekomprese cokoliv nepodaří (otevřít soubor / vytvořit soubor / číst zdroj / zapisovat cíl / nesprávný formát UTF-8 dat / ...), funkce bude vracet hodnotu `false`.

Implementace dekomprese je snazší, referenční řešení má cca 200 zdrojových řádek. Implementace funkce pro dekompresi je požadovaná, pokud nebude fungovat, neprojde program závaznými testy. Funkce pro komprimaci je pracnější. Referenční řešení obou funkcí má přibližně 500 zdrojových řádek. To je na domácí úlohu větší rozsah, proto je tato část úlohy pouze volitelná za bodový bonus. Algoritmus konstrukce Huffmanova kódu je popsán např. na **Wikipedii**. Pokud se rozhodnete řešit pouze závaznou část, ponechte funkci pro komprimaci podle ukázky a v jejím těle vždy vracejte hodnotu `false`.

Vstupní text je kódovaný v UTF-8 kódování. V povinných testech se ale zpracovávají pouze soubory obsahující pouze znaky z rozsahu ASCII (0 až 127). Pro tento rozsah není rozdíl mezi kódováním ASCII a UTF-8, zpracování se ale zjednoduší (jeden znak zabírá vždy 8 bitů). Pokud se nechcete zabývat kódováním UTF-8, můžete načítat znaky v Huffmanově stromu jako 8 bitové a zapisovat do dekomprimovaného výstupu pouze 8 bitové znaky. Takové řešení projde povinnými testy, ale neprojde nepovinnými testy, tedy bude hodnoceno méně než 100% bodů.

Při implementaci máte k dispozici datové struktury z STL, viz ukázka. Struktury z STL použít můžete, ale nemusíte. Pokud budete řešit pouze funkci pro dekompresi, patrně je nevyužijete. STL datové struktury se hodí zejména pro implementaci komprimace.