Kódovanie znakov

(zdroj: Internet, upravený preklad Mark Pilgrim: Ponořme se do Python(u) 3)
Treba vedieť odseky Unicode a UTF-8!

Úvod do problematiky

Premýšľa o tom len málo ľudí, ale text je neuveriteľne komplikovaný. Začneme s abecedou. Obyvatelia Bougainvillu používajú najmenšiu abecedu na svete. Ich abeceda Rotokas sa skladá len z 12 písmen: A, E, G, I, K, O, P, R, S, T, U a V. Na opačnom konci zoznamu nájdeme jazyky, ako sú čínština, japončina a kórejčina, ktoré používajú tisíce znakov. Angličtina používa 26 písmen — alebo 52, pokiaľ počítate zvlášť malé a veľké písmená — a k tomu niekoľko interpunkčných znakov, ako sú *!@#$%&*.
Ak, v súvislosti s počítačmi hovoríte o „texte“, potom pravdepodobne myslíte „znaky a symboly na počítačovej obrazovke“. Ale počítače nepracujú so znakmi a symbolmi. Pracujú s bitmi a bajtmi. Každý kúsok textu, ktorý ste kedy videli na počítačovej obrazovke, bol v skutočnosti uložený v určitom znakovom kódovaní. Približne povedané, kódovanie znakov zachytáva vzťah medzi tým, čo vidíte na obrazovke, a tým, čo je v skutočnosti uložené v pamäti počítača a na disku. Znakových kódovaní sa používa veľmi veľa. Niektoré sú optimalizované pre konkrétny jazyk, akým je ruština, čínština alebo angličtina. Iné kódovania sa môžu používať pre viac jazykov. V skutočnosti je to ešte oveľa komplikovanejšie. Rad znakov je spoločný pre viac rôznych kódovaní, ale každé kódovanie môže pre ich uloženie v pamäti alebo na disku používať inú postupnosť bajtov. Takže o znakovom kódovaní môžete uvažovať ako o dešifrovacom kľúči. Kedykoľvek vám niekto poskytne postupnosť bajtov - súbor, webovú stránku, čokoľvek - a bude tvrdiť, že to je „text“, budete k úspešnému dekódovaniu bajtov na znaky musieť vedieť aj to, aké kódovanie znakov bolo použité. Ak vám niekto poskytne zlý kľúč alebo vám dokonca nedá žiadny, postaví vás pred nevyhnutnú úlohu rozlúsknuť kód. Môže sa stať, že pri tom urobíte chybu a výsledok nebude dávať zmysel.
Každý význačný jazyk na svete má definované svoje znakové kódovanie. Každé kódovanie znakov bolo kvôli rozdielom v jazykoch optimalizované pre konkrétny jazyk, pretože pamäťový a diskový priestor boli v minulosti veľmi drahé. Mám tým na mysli to, že pre reprezentáciu znakov jazyka používalo každé kódovanie rovnaký interval čísel (0-255). Pravdepodobne poznáte napríklad kódovanie ASCII, ktoré ukladá anglické znaky ako čísla z intervalu 0 až 127. (65 je veľké A, 97 je malé a atď.) Angličtina má veľmi jednoduchú abecedu, ktorá môže byť úplne vyjadrená menej ako 128 číslami. Pre tých z vás, ktorí vedia počítať v dvojkovej sústave, na to stačí 7 z 8 bitov v bajte.
Západoeurópske jazyky, akou sú francúzština, španielčina a nemčina, používajú viac znakov ako angličtina. Presnejšie povedané, nájdete v nich písmená kombinovaná s rôznymi diakritickými značkami, ako napríklad pri znaku ñ používaného v španielčine. Najbežnejším kódovaním je u týchto jazykov CP-1252. Označuje sa tiež „windows-1252“, pretože sa používa v Microsoft Windows. Kódovanie CP-1252 zdieľa znaky v intervale 0-127 s ASCII, ale využíva aj interval 128-255. Nájdeme v ňom také znaky ako ñ (241), ü (252) atď.[1]. Stále ale ide o jednobajtové kódovania. Najväčšie možné číslo (255) sa stále zmestí do jedného bajtu.
Potom tu ale máme jazyky, ako je čínština, japončina a kórejčina, ktoré používajú také množstvo znakov, že vyžadujú viacbajtové znakové sady. Každý ich „znak“ je vyjadrený dvojbajtovým číslom v intervale 0-65535. Ale pri rôznych viacbajtových kódovaniach sa stále stretávame s rovnakým problémom, ako pri rôznych jednobajtových kódovaniach. Každé z nich používa rovnaké čísla pre vyjadrenie rôznych vecí. Používajú len širší interval čísel, pretože musia vyjadriť oveľa viac znakov.
Vo svete, ktorý ešte nebol prepojený sieťami a kde „text“ bolo niečo, čo ste si sami napísali a príležitostne vytlačili, to väčšinou bolo prijateľné. Zdrojové texty boli v ASCII a všetci ostatní používali textové procesory, ktoré definovali svoje vlastné (netextové) formáty. Tie si spolu s informáciami o štýle ukladali tiež informáciu o znakovom kódovaní. Ľudia tieto dokumenty čítali prostredníctvom rovnakých textových procesorov, aké použil pôvodný autor, takže všetko viac-menej fungovalo.

[1] Pozri napr. vo Worde: Vložiť - Symbol - Kód znaku:... z ASCII - desiatkovo

Teraz si predstavte vzostup globálnych sietí s elektronickou poštou a s webom. Hŕby „prostých textov“ lietajú okolo zemegule - boli napísané na jednom počítači, prenesené cez druhý a zobrazované na treťom počítači. Počítače vidia len čísla. Ale čísla môžu znamenať rôzne veci. Ach nie! Čo budeme robiť? Takže systém musel byť navrhnutý tak, aby si každý „obyčajný text“ so sebou niesol informáciu o kódovaní. Pripomeňme si, že ide o dešifrovací kľúč, ktorý premieňa čísla zrozumiteľné počítaču na znaky čitateľné človekom. Chýbajúci dešifrovací kľúč vedie ku skreslenému textu, zmätkom alebo k niečomu horšiemu.

Teraz si predstavte, že by sme viac druhov textu chceli uložiť na rovnakom mieste, ako napríklad v rovnakej databázovej tabuľke uchovávajúcej doručenú elektronickú poštu. Pre každý druh musíme rovnako uložiť aj znakové kódovanie, aby sme text dokázali správne zobraziť. Myslíte si, že to nie je príliš ťažká požiadavka? Skúste vo svojej e-mailovej databáze vyhľadávať. To znamená, že počítač bude musieť za behu realizovať prevody medzi rôznymi kódovaniami. Tu prestáva „sranda“, že?

Teraz si predstavte, že by ste mali viacjazyčné dokumenty, v ktorých sa znaky z rôznych jazykov vyskytujú vedľa seba, v tom istom dokumente. (Pomocník: Programy, ktoré sa o to pokúšali, typicky používali pomocné kódy (escape) pre prepínanie "režimov". Prásk, teraz ste v ruskom režime KOI8-r, takže 241 znamená Я; bum, teraz ste gréckom režimu pre Mac, takže 241 znamená ώ.) Aj v takýchto dokumentoch by ste samozrejme chceli vedieť vyhľadávať. Tak a teraz plačte, pretože všetko, čo ste si mysleli, že o reťazcoch viete, je vám k ničomu. **Nič také ako „obyčajný text“ neexistuje.**

Unicode

Unicode je systém navrhnutý tak, aby bolo možné jednoznačne vyjadriť každý znak z každého jazyka. Každé písmeno, znak alebo ideogram sa v Unicode vyjadrujú ako 4-bajtové číslo. Každé číslo vyjadruje jedinečný znak, ktorý sa používa aspoň v jednom jazyku nášho sveta. (Nie všetky čísla sú využité, ale tých použitých je viac ako 65535. To znamená, že dva bajty nestačia.) Znaky, ktoré sa používajú vo viacerých jazykoch, majú zvyčajne rovnaké číslo - ak neexistuje dobrý etymologický dôvod, aby to tak nebolo. Bez ohľadu na ďalšie okolnosti je ale pre každý znak vyhradené jedno číslo a pre každé číslo len jeden znak. Jedno číslo vždy znamená jedinú vec. Nepoužívajú sa žiadne skôr spomínané „režimy“. U+0041[1] znamená vždy 'A', a to aj v prípadoch, ak by váš jazyk 'A' nepoužíval.

Na prvý pohľad to vyzerá ako výborná myšlienka. Jedno kódovanie vládne všetko. Viac jazykov v jednom dokumente. Už nikdy viac „prepínanie režimu“ uprostred textu len kvôli kódovaniu. Ale už v tejto chvíli by vás mala napadnúť zjavná otázka. Štyri bajty? Pre každý jeden znak? To vyzerá ako hrozné plytvanie. Obzvlášť pre jazyky, ako sú angličtina alebo španielčina, ktoré na vyjadrenie každého používaného znaku potrebujú menej ako jeden bajt (256 čísel). V skutočnosti je to plytvanie aj pre jazyky založené na ideogramoch (ako je čínština), ktoré na jeden znak nepotrebujú nikdy viac ako dva bajty.

Existuje kódovanie Unicode, ktoré používa štyri bajty na znak. Nazýva sa **UTF-32**, podľa počtu 32 bitov, čo sú 4 bajty. UTF-32 je priamočiare kódovanie. Každé číslo uložené na štyroch bajtoch sa reprezentuje ako Unicode znak s rovnakým číslom. Má to svoje výhody. Najdôležitejšia z nich je tá, že N-tý znak reťazca môžeme sprístupniť v konštantnom čase. N-tý znak totiž začína na 4 × N-tom bajte. Ale má to aj nevýhody. Tá najviditeľnejšia je, že na každý znak potrebujeme štyri bajty. Znakov je v Unicode veľmi mnoho, ale ukazuje sa, že väčšina ľudí nepoužije nikdy žiadny, ktorý by ležal mimo prvých 65535. Takže tu máme ďalšie kódovanie Unicode. Nazýva sa **UTF-16** (pretože 16 bitov sú 2 bajty). V UTF-16 sa každý znak s číslom z intervalu 0-65535 kóduje do dvoch bajtov. Ak naozaj potrebujeme vyjadriť zriedka používané Unicode znaky z „astrálne roviny“ (presahujúce 65535), používa UTF-16 isté „nekalé“ triky. Najzjavnejšia výhoda: UTF-16 je priestorovo dvakrát efektívnejšie ako UTF-32, pretože pre uloženie každého znaku potrebujeme len dva bajty namiesto štyroch (s výnimkou tých, pri ktorých to neplatí). A ak budeme predpokladať, že reťazec neobsahuje žiadne znaky z astrálne roviny, môžeme ľahko nájsť N-tý znak v konštantnom čase. Tento predpoklad je celkom dobrý, ale len do doby, keď to prestane platiť. Aby to bolo ešte komplikovanejšie, ako UTF-32, tak aj UTF-16 majú ďalšie skryté nevýhody. Rôzne počítačové

[1] U znamená Unicode, 0041 je hexadecimálny zápis dekadického čísla 65; v programovaní sa používa zápis \u0041 reprezentujúci jeden znak, v tomto prípade veľké A.

systémy ukladajú totiž jednotlivé bajty rôznym spôsobom. Tak napríklad znak U+4E2D by mohol byť v UTF-16 uložený buď ako 4E 2D alebo 2D 4E. Závisí to od toho, či systém používa prístup big-endian (na nižšej adrese významnejší bajt) alebo little-endian (na nižšej adrese menej významný bajt). (Pre UTF-32 existujú dokonca ešte ďalšie možnosti usporiadania bajtov.) Ak váš dokument nikdy neopustí váš počítač, je to jedno - rôzne aplikácie budú na rovnakom počítači používať rovnaké poradie bajtov. Ale v okamihu, kedy budete chcieť dokument prenášať medzi systémami, napríklad prostredníctvom webu, budeme potrebovať spôsob, ako vyjadriť nami používané poradie uložených bajtov. V opačnom prípade by cieľový systém nevedel zistiť, či dvojbajtová postupnosť 4E 2D znamená U+4E2D alebo U+2D4E. Viacbajtové kódovanie Unicode pre vyriešenie tohto problému definujú „Byte Order Mark“ (značka poradia bajtov; skrátene BOM). Ide o špeciálny netlačiteľný znak, ktorý môžete vložiť na začiatok svojho dokumentu, aby ste dali najavo, v akom poradí sú vaše bajty uvedené. Pre UTF-16 je Byte Order Mark rovný U+FEFF. Ak dostanete dokument v UTF-16 začínajúci bajtmi FF FE, potom viete, že ide o jedno z určení poradia bajtov. Ak dokument začína bajtmi FE FF, potom viete, že poradie bajtov je obrátené.

Napriek tomu UTF-16 nie je celkom ideálne. Platí to najmä v prípadoch, ak používate veľké množstvo ASCII znakov. Keď o tom popremýšľate, dokonca aj čínske webové stránky budú obsahovať veľké množstvo ASCII znakov - všetky tie značky a atribúty, ktoré obklopujú tlačiteľné čínske znaky. Ak vieme nájsť N-tý znak v konštantnom čase, je to výhodné. Ale stále tu máme nepríjemný problém so znakmi z „astrálnej roviny“. To znamená, že nemôžete zaručiť, že každý znak je uložený presne na dvoch bajtoch. Takže v skutočnosti nemôžete N-tý znak nájsť v konštantnom čase - ak si však neudržiavate oddelený index . A medzi nami, vo svete sa nachádza ohromné množstvo ASCII textov ... Týmito otázkami sa už zaoberali iní a prišli s riešením:

UTF-8

UTF-8 je kódovací systém s premenlivou dĺžkou. To znamená, že rôzne Unicode znaky zaberajú rôzny počet bajtov. Pre ASCII znaky (A...Z atď.) používa UTF-8 len jeden bajt na znak. V skutočnosti používa presne ten istý bajt (ako ASCII). Prvých 128 znakov (0-127) sa v UTF-8 teda nedá rozlíšiť od ASCII. Znaky z „rozšírenej latinky“, ako sú ñ a ü budú zaberať dva bajty. (Bajty tu nevyjadrujú kód z Unicode tak jednoduchým spôsobom, ako je tomu u UTF-16.) Čínske znaky ako 中 zaberajú tri bajty. Zriedka používané znaky z „astrálne roviny“ zaberajú štyri bajty.

Nevýhody: Pretože každý znak zaberá rôzny počet bajtov, je nájdenie N-tého znaku lineárnou zložitosťou, tzn. čím je reťazec dlhší, tým dlhšie budeme znak na určenej pozícii vyhľadávať. Pri kódovaní znakov na bajty a dekódovanie bajtov na znaky sa musíme navyše zaoberať ďalšími manipuláciami s bitmi.

Výhody: Kódovanie bežných ASCII znakov je extrémne efektívne. Pri kódovaní znakov z rozšírenej latinky nie je horšie ako UTF-16. Pre čínske znaky je lepšie ako UTF-32. A vďaka jednoznačnému spôsobu manipulácie s bitmi tu neexistujú žiadne problémy s poradím bajtov. Dokument kódovaný v UTF-8 používa na každom počítači presne rovnakú postupnosť bajtov.