A bináris számok nem megfelelőek a szakaszhossz kódoláshoz

Tegyük fel, hogy az i darab 0-ból, és utána egy 1-esből álló szakaszhoz az i egész szám bináris értékét rendeljük. Akkor a 000101 bitvektor két szakaszból áll, amelyeknek a hossza 3, illetve 1. Ezek az egészek binárisan ábrázolva 11 és 1, tehát a 000101 szakaszhossz kódolásának eredménye 111. Azonban, hasonló számítás mutatja, hogy a 010001 bitvektor kódja szintén 111, és a 010101 a harmadik olyan, amelynek a kódja szintén 111. Így a 111 nem dekódolható egyértelműen bitvektorrá.

5.4.2. Tömörített bittérképek

Tegyük fel, hogy egy n rekordot tartalmazó állomány F mezőjén van egy bittérkép-indexünk, és m különböző érték fordul elő az állományban az F mezőben. Ekkor az index összes bitvektorában a bitek száma mn. Ha a blokkok mondjuk 4096 bájt hosszúak, akkor 32 768 bit fér egy blokkba, tehát a szükséges blokkok száma: $mn/32768$. Ez a szám lehet kicsi az egész állomány tárolásához szükséges blokkok számához viszonyítva, de minél nagyobb az m értéke, annál több helyet foglal le a bittérkép-index.

Viszont, ha az m nagy, az 1-es a bitvektorokban nagyon ritka lesz, pontosabban annak valószínűsége, hogy egy tetszőleges bit 1-es: $1/m$. Ha az 1-es ritka, akkor lehetőségünk van úgy kódolni a bitvektorokat, hogy átlagosan sokkal kevesebb, mint n bitet tartalmazzanak. Egy szokásos módszer a *szakaszhossz kódolásnak* vagy sorkifejtő kódolásnak nevezett, ahol egy szakaszt – ami i darab egymás utáni 0 bitből, majd egy ezeket követő 1-esből áll – az i egész szám valamilyen megfelelő bináris kódjával ábrázolunk. Majd egymás után rakjuk a kódokat az összes szakaszra, és az így kapott bitsorozat a bitvektor kódolt változata.

Elképzelhető, hogy az i egészet egyszerűen úgy ábrázoljuk, hogy bináris számként írjuk fel. Azonban ez az egyszerű szerkezet nem mindig megfelelő, mert nem lehet a kódok sorozatát a benne foglalt szakaszok hosszának egyértelmű meghatározásával szétszedni (lásd a „A bináris számok nem megfelelőek a szakaszhossz kódoláshoz” című bekeretezett részt). Így az i egész szám kódja, ami a szakasz hosszát mutatja, bonyolultabb kell legyen, mint az egyszerű bináris ábrázolás.

A sok lehetséges kódolási szerkezet közül egyet fogunk használni. Létezik jobb, bonyolultabb szerkezet, ami az itt elért tömörítés mértékét majdnem kétszeresére tudja javítani, de csak akkor, ha a jellemző szakaszhossz nagyon nagy. A szerkezetünknél először meg kell határoznunk, hogy az i binárisan ábrázolva hány bitből áll. Ez a j szám, ami közelítőleg $\log_2 i$, unárisan ábrázolva $j - 1$ darab 1-esből és egy 0-sból áll. Aztán folytathatjuk az i bináris értékével.[1]

5.23. példa: Ha $i = 13$, akkor $j = 4$, azaz 4 bit kell az i bináris ábrázolásához. Így az i kódoltan 1110-val kezdődik. Ezt követi az i binárisan, vagyis 1101. Tehát a 13 kódolva 11101101.

Az $i = 1$ kódolva 01, és az $i = 0$ kódolva 00. Mindkét esetben $j = 1$, tehát egyetlen 0 a kezdet, és ezt a 0-t követi az i-t ábrázoló egy bit. •

Ha egymás mögé rakjuk a kódolt egészek sorozatait, mindig vissza tudjuk állítani a szakaszhosszak sorozatát, és ezért az eredeti bitvektor visszaállítható. Tegyük fel, hogy átnéztünk már valahány kódolt bitet, és most egy bitsorozat elején vagyunk, amely egy bizonyos i egész szám kódja. Továbbmegyünk az első 0-ig, így meghatározzuk a j értékét. Azaz, j egyenlő azon bitek számával, amennyit le kell olvasnunk, amíg elérünk az első 0-hoz (beleértve ezt a 0-t is a bitek számába). Ha már ismerjük a j értékét, akkor vegyük a következő j bitet, ez adja kettes számrendszerben ábrázolva az i egész számot. Továbbá, ha végignéztük az i-t ábrázoló biteket, akkor tudjuk, hol van a következő egész kódjának a kezdete, így meg tudjuk ismételni az eljárást.

5.24. példa: Fejtsük vissza a 11101101001011 sorozatot. Az elején kezdve, a 4-edik biten találjuk az első 0-t, tehát $j = 4$. A következő 4 bit: 1101, tehát megállapíthatjuk, hogy az első egész a 13. A 001011 maradt, amit vissza kell fejtenünk.

Mivel az első bit 0, tudjuk, hogy a következő bit magát az egészet ábrázolja, ez a szám a 0. Így eddig a 13, 0 sorozatot fejtettük vissza, és a maradék visszafejtendő sorozat a 1011.

---

[1] Ténylegesen, a $j = 1$ esetet leszámítva (azaz, ha $i = 0$, vagy $i = 1$), biztosak lehetünk abban, hogy az i kettes számrendszerben felírva 1-gyel kezdődik. Így számonként megspórolhatunk 1 bitet, ha ezt az 1-est elhagyjuk, és csak a maradék $j - 1$ bitet használjuk.

Az első 0-t a második pozíción találjuk, amiből következik, hogy az utolsó két bit ábrázolja az utolsó egészet, ami 3. A szakaszhosszak teljes sorozata tehát 13, 0, 3. Ezekből a számokból felépíthetjük a tényleges bitvektort: 000000000000110001. •

Gyakorlatilag minden bitvektor, amit így dekódolunk, 1-essel végződik, és a záró 0-kat nem állítjuk vissza. Mivel feltételezhetően ismerjük az állományban lévő rekordok számát, a további 0-kat hozzá tudjuk adni. Azonban, mivel a 0 egy bitvektorban azt jelenti, hogy a megfelelő rekord nincs benne a kívánt halmazban, nem is kell tudnunk a rekordok teljes számát, és figyelmen kívül hagyhatjuk a záró 0-kat.

5.25. példa: Alakítsunk át néhány, az 5.22. példában szereplő bitvektort a mi szakaszhossz-kódunkra. Az első három (25, 30, 45) életkorhoz tartozó vektorok: 100000001000, 000000010000, illetve 010000000100. Ezek közül az elsőhöz a (0, 7) szakaszhossz sorozat tartozik. A 0 kódja 00, a 7 kódja 110111. Így a 25 éves életkor bitvektora a 00110111 sorozattá alakul.

Hasonlóan, a 30 éves életkornak csak egy szakasza van, hét 0-val. Tehát ennek a kódja: 110111. A 45 éves kor bitvektorának két szakasza van (1, 7). Mivel az 1 kódja 01, és mint meghatároztuk, a 7 kódja 110111, a harmadik bitvektor kódja: 01110111. •

A tömörítés az 5.25. példában nem nagy. Azonban nem láthatjuk az igazi előnyöket, ha n, a rekordok száma kicsi. Hogy méltányolni tudjuk a kódolás értékét, tegyük fel, hogy $m = n$, vagyis a mezőnek, amelyen a bittérkép-indexet létrehozzuk, minden értéke egyedi. Megjegyezzük, hogy egy i hosszú szakasz kódja körülbelül $2 \log_2 i$ bit. Mindegyik bitvektorban egyetlen 1-es van, tehát egyetlen szakaszból áll, és annak a szakasznak a hossza nem nagyobb, mint n. Így a $2 \log_2 n$ bit egy felső korlát ebben az esetben a bitvektor kódjának hosszára.

Mivel n bitvektor van az indexben (mert $m = n$), az indexet alkotó bitek teljes száma legfeljebb $2n \log_2 n$. Megjegyezzük, hogy kódolás nélkül n^2 bit kellett volna. Ha $n > 4$, akkor $2n \log_2 n < n^2$, és ahogy n nő a $2n \log_2 n$ tetszőlegesen kisebb lesz, mint n^2.