Implicit Encoding: Exponential Factorization and Information Transfer

September 20, 2015

Abstract

We show a method where one can achieve exponential data compression, compressing and uncompressing in polynomial time with minimal space requirements. An program using the method is developed and a 2 gigabyte file is compressed into 4 kilobytes, and then uncompressed with lossless data integrity. The example program is available on github here. A youtube summary is available here.

Algorithm, in brief

All programs are fundamentally, just numbers. Very large numbers, yes, but numbers. An unusually large program might have a terabyte of data. this would correspond to a number in the range of $2^{1000000000000}$, much bigger than the hypothetical googol, at 10^{100}.

If Bob and Alice wanted to communicate exceptionally large numbers to one another using only their fingers and counting, they might agree beforehand on a scheme like this: the left thumb, if raised, increases the base of the number system by one. Then for any given number, Bob could communicate a portion of it using base two, raise his left thumb several times, and suddenly all of his fingers would represent powers of five. His fingers would encode exponents: $5^0, 5^1, 5^2, ...$

We can generate relatively even distributions of large numbers by taking large exponents of primes and multiplying them together. So one could take the permutations of a list like [4,5,6,7,8,9]

1 Performance Tuning

The algorithm presents a large number of opportunities for performance tuning at pretty much every step.

A Fun to think about

Some of the crazier ideas that've happened across the author's brain: one could, instead of, or in addition to, using the first 32 digits in the number to compare the keys to the value you are trying to represent with them, one could fit a trigonometric function to the distribution of the digits in the binary and then describe that function with coefficients. This would also require fitting a trig function to the encoded number though, so would probably really really increase the time taken by the encoding step. But yes in general it might also be possible to do some kind of fourrier decomposition to describe the file. Fun to think about anyway.

2.4.4

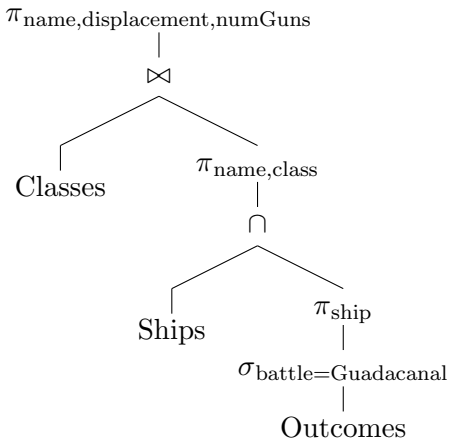
b

Find the ships launched prior to 1921.

$$\pi_{\text{name}}$$
$$\sigma_{\text{launched}<=1921}$$
$$\text{Ships}$$

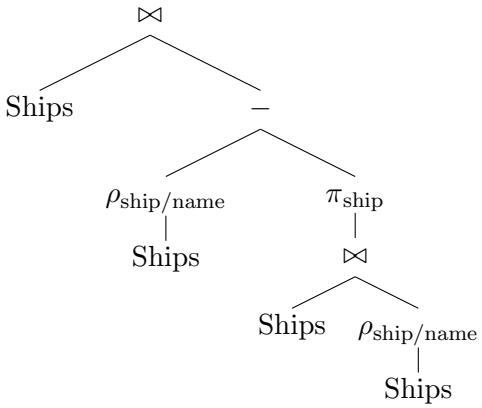
e

List the name, displacement, and number of guns of the ships engaged in the battle of Guadalcanal.



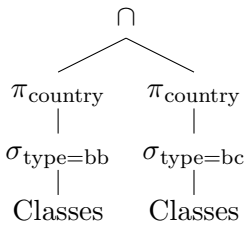
f

List all the capital ships mentioned in the database. (Remember that all these ships may not appear in the ships relation.)



h

Find those countries that had both battleships and battlecruisers.



i

Find those ships that lived to fight another day; they were damaged in
one battle, but later fought in another.

2.4.7

a

max:n+m
min:n or m, whichever is greater

b

max:n*m
min: n or m, whichever is greater

c

max:n*m
min:number of tuples in n that satisfy C or m, whichever is greater

d

max:n
min:0

2.4.8

$\pi_{a_1,..,a_n}(R \bowtie S)$ where $a_1, ..., a_n$ are the attributes held in common between R and S.
it follows then that another representation is in terms of more basic operations than the natural join:
project the unique attributes in the union of R and S, after selecting

the common attributes in the cross product.
$\sigma_{\text{r.a1=s.a1,...,r.an=s.an}}(\pi_{R \cup S}(R \times S))$
where a1,...,an are the common attributes between s and r, and $R \cup S$ are the unique attributes between the two
$\sigma_{\text{r.a1=s.a1,...,r.an=s.an}}(\pi_{S \cup R}(S \times R))$
where a1,...,an are the common attributes between s and r, and $S \cup R$ are the unique attributes between the two

2.5.2

a

$\pi_{\text{bore>16in}}(Classes) = \theta$

b

$\pi_{\text{bore>14in}}(\sigma_{\text{numGuns>9}}(Classes)) = \theta$

c

$\sigma_{\text{count(class)>2}}(Classes) = \theta$

2.3.2

a

CREATE TABLE Classes(class CHAR(1OO), type CHAR(2), numGuns INT, bore INT, displacement INT)

c

CREATE TABLE Battles(name CHAR(1OO), date DATE)