FUNCTION COMPRESS－それは何か？

Last Update 05/5/1

FUNCTION COMPRESSとは？

【Function Compress とは？】

「Function Compress」とは、現在主流となっている、「ZIP」や「LZH」、「CAB」といった、データの圧縮の仕組みのひとつです。上の3つの圧縮規格の詳しい説明や、圧縮についての基礎知識などの解説は、今回割愛させていただきました。

いつかそういった知識コーナーも公開しますので、当面は圧縮のことに精通している方以外は、難しい内容のページになることをご容赦ください。

＜意義＞

「Function Compress」は、現状の圧縮方式の弱点をカバーし、また、シンプルな圧縮理論を実現することを念頭において作られた圧縮形式です。つまり、現状の圧縮方式の抱える問題

1：画像や音楽の可逆圧縮はあまり効果がない。

2：圧縮性能と処理時間が反比例。

3：制御用ライブラリや理論が複雑

を、解決する圧縮方式です。

＜アルゴリズム＞

では、アルゴリズムの解説に移ります。

データは、所詮、数字の羅列です。そして、それは計算、つまりは、式で表せる有限項数列とみなせます。始まりがあって終わりのある数列です（一般解が果てしなく複雑と考えてください）。

これを考えたときに、この「数列」を「式」で表すにはどうするのがよいでしょうか？

いちばん簡単なのは、値をそのまま列挙する、つまりは圧縮しないことです。しかし、これでは何にも意味がありません。

では、当然のことですが、値の操作をしましょう。式で表す際に、まず考えられるのは、数列を数個ずつ区切って、同じ物が出たら、その値と、個数を記録する、と言うもの。式で表すと

$$a_n = (x * n個) \text{ or } y$$

のようになるでしょうか？まあ、これはどうでもよいです。例ですので。

それで、このような「式」でデータを表すというのが、主なアルゴリズムになります。では、その「式の表し方」を説明しましょう。

$$F(x) = Ax^a + Bx^b + Cx^c + Dx^d + Ex^e + Fx^f + Gx^g$$

という、変数係数があり、べき乗を変数で表した「項」の和が7つ並んだ関数があるとします。

例：データが与えられたときに、ある値が「1」だったとします（1バイトです）。そして、その先に連続する7つの値が、前の値に1ずつ足された、

f(x)=x

という連続関数で、8つの値が与えられるものだとしましょう。

解：このとき、この8つの値（データ）を式で表すとどうなるでしょう？上にあげたとおりになりますが、それを「式」の形で表現しましょう。

F(x)=1x^1+0x^0+0x^0+0x^0+0x^0+0x^0+0x^0+0x^0

ですね。さて、これをx=0,1,2,3・・・について代入すると、値はそれぞれ「0,1,2,3」になります。xは、連続しています。数列の第x項ということです。そして、これで式で値(データ)を表せたわけです。

このとき、xの増分を「1ずつ」と決めておくと、与えられた8つの値は順々に求まります。つまり、1ずつと決めたときに、運良く、数列（データ）が1・2・3・4・5・6・7・8とくれば、式を例のように決めておいたときに、初期値を与えるだけで全てのデータをあらわせるわけで、これはすなわち、圧縮できたことになります。

さて、重要なのはここからです。データが与えられたとき、「代入しても値が一致しなかった」というようなデータが与えられたらどうしましょう。圧縮できないような場面・・・つまり、定めておいた式で表せなかったときです。そのときは、「圧縮できない」とします。

圧縮時には、「圧縮できる」と「圧縮できない」の2つの状況しか存在しえませんので、圧縮できるか否かは「1ビット」で表現できます。ですので、一致しない、つまり「圧縮できない」時まで「圧縮できる」わけですから、そのときはxの増加に身を任せていればよいのです。そして、「圧縮できな」くなったら、「圧縮できない＝0」という状態を1ビットの情報の付加で実現します。しばらくしてまた、「圧縮できる」時がきたら「圧縮できる＝1」という状態を付加します。

つまり、「圧縮できない」と「圧縮できる」の状態が交互に続くという、圧縮データの構造になります。

しかし、ある問題が発生します。それは「いつ圧縮できなくなるかわからない」と言うことです。x=5の時に圧縮できなくなった、のか、x=8のときなのか、まったく予測はつきません。1ビットの情報で圧縮の可否の区別は実現できます。しかし、圧縮できている状況で、xがいくつのとき圧縮できなくなるのかまでは表せません。

ですから、xの範囲を「0<=x<=n」と定めるのです（ただし、nは定数）。

例：たとえば、nが4のとき。

F(x)が1つ一致しようと、2つ一致しようと、3つ一致しようと、関係ありません。「4つ」一致したとき、「圧縮できる」状態なのです。つまり、圧縮できるときに表せる数列の個数（データサイズ＝圧縮サイズ）は当然「n個」です。この時、データサイズを「8÷n」バイトにまで下げることができます、一方、1ビットの付加が存在するため、最終的には「64÷n + 1」ビットになります。

そして、次に圧縮できるか否かは、直前の値をF（n）として、F(n)+G(x)で決められます。そして、もしこれで圧縮できなかったら、圧縮できない1ビットを付加するわけですが、これには大きな問題があります。

圧縮できないとき、データサイズが「１」ビット増えてしまうのです。これを防ぐには、圧縮できないときが続いたことを表す「圧縮できない回数＝vビット」を打ち消す方法が必要になります。ですので、

例：圧縮できないときを「v」とし、圧縮できるときを「s」とすると、

vvvvvvvv（8回失敗）で1バイトの無駄が出ますが、vvvvvvvs（8回目に成功）なら、ある方法でこれを防げます。それは、「項の定義」です。

F(x)は、7つの単項式の「和」です。つまり、F（x）の各項の係数（ t(s)とします ）を、0かt(s)にするかで、関数を最大128通り定義することができます。これは、7つの項の「存在・・・0かあるか」を決定する7ビットの情報で式の状態を表すことができます。もうお気づきですね。これに先ほどの「圧縮の是非」の1ビットをたした8ビットで、圧縮できるとき、その時点での関数F（x）が何かを1バイトの情報で表せるのです。これには、圧縮できるときにはnバイト分の情報があります。

そして、先ほどの例のような「圧縮できないとき」が7つまで続くときに限って、以下のようにします。これは、データの膨張を防ぐ手段です。

0vvvvvvv 0vvvvvvv 0vvvvvvv 0vvvvvvv 0vvvvvvv 0vvvvvvv 0vvvvvvv 0abcdefg

最後の1バイトは、各々の0の位置に本来あったデータです。どういう意味かを例を使って紹介します。

例：圧縮できない状態が7回まで続かないときは・・・圧縮できないときを3個とすると、

0vvvvvvv0vvvvvvv0vvvvvvv1abcFFFF

とします。4回目で成功するときに、左側3つの項が必要でない場合、無理矢理、圧縮できるときの情報として、0の位置に有ったデータをずれ込ませるのです。Fは、先ほどの各項の係数の是非です。項の是非を以下のように定義すると、

1ABCDEFG

（つまり、左から順に項の是非とします。1のときだから1がある。）

7つのうち、左からn(<=7)個無くなっても、つまり、本来F(x)の項数が右からn個までだったと仮定しても、それでデータが表せれば問題ないのです。だから、ずれ込ませているのです。そして、左から順にずれ込ませています。これは右からでも同じことですが。ただ、右にx^2のような重要な項をおくことが多いでしょうから、数学にあわせるなら避けたほうが良いかもしれません。

こうすると、データがまったく圧縮できないとき（7バイト単位）、8÷7＝14%ほど、元データよりサイズが増えますが、圧縮できるときは、xの範囲を指定すれば（0<=x<=n）、データサイズは（1÷n）バイトですみます。

さて、範囲指定nを4ビットの範囲で指定します。そして、xの増分、つまり、G(x)=F(t･x)の「t」を4ビットで指定します。これは、正負あります。

次に、関数各項の「べき数」と「係数」を4ビット×２で7つ指定します。これは7バイトです。

この2つを足して8バイトになったのを「関数ブロック」とします。これを、圧縮すべきファイルの中で、格納したデータがmバイトずつになった時点で再定義します。つまり、全体をm個のブロックで区切るのです。ブロックで区切ると、ファイルのデータの偏差に対応できるようになります。というより、対応できる可能性が増えます。

そして、ブロックF(x)が実現した圧縮後のデータのサイズ、これを4バイトで表現します。(最大データサイズは、4GBです。)

すべて足すと、12バイトの「ブロックInfo」になります。これをブロックのはじめでそれぞれ定義します。そうすると、この規格のほぼ全ての操作は終わりです。

・・・ふう、これくらいでよろしいでしょうか。疲れました。うまくできてるでしょ！この規格。

さて、まだ実現していないですが、将来的な夢のアイデアをほんの少しご紹介！

【夢のアイデア】

皆さんがご利用なさっている「デジカメのメモリー」、つまりは「フラッシュメモリー」ですが、簡単に言うと、中にチップが入っています。これは、ハードディスクの中にある円盤のような、実際にデータを記録する場所です。しかし、面白いのは、このチップ、なんと中で「ブロック」に分かれているのです。勘のいい人はおわかりでしょう。このブロックと、先ほどのブロックは、似ていますね～。

チップの中は、皆さんがデータの記録に使う部分と、「じょちょうぶ」という部分があり、製品の容量は、「じょちょうぶ」を無視した値です。なぜかというと、じょちょうぶは万が一のときの補助用に使うので、普段の記録にはつかえないのです。じつは、これはフラッシュメモリーの寿命と関連します。

フラッシュメモリーは、書き換え回数が数十万回程度ですが、これは「ブロック」単位でしか書き換えできない性質のチップを使っているからという、あるひとつの理由があります。この寿命のせいで、万が一にも書き込みや読み込みでエラーを起こし、ブロックが使えなくなることがあるのです。説明すると長いので、詳しいことは割愛させていただきますが、とりあえずブロックがあるということが重要です。このブロックの大体アドレスを書き込んだりすることがあるのです。

しかし、じょちょうぶは使えないと言いましたが、実際は、使ってもよいのです。使えることは使えるのだ、という言い方をしておきましょう。たいていのメーカーは、このじょちょうぶの使い方に対して、ある程度の範囲を決めていますが、後は自由にしています。これを、先ほどの「関数ブロック」の定義に使ったらどうでしょうか？中のチップのブロックは、関数ブロックのように振舞いますよね？そして、関数ブロックには「データサイズ」があります。これが味噌です。もうお気づきでしょう。フラッシュメモリーをそのまま圧縮してしまうんですね。こんなことができるかもしれないのです。ね、隙間の無い規格でしょう。

また、CDにもブロックがあります。最近はやりの「パケットライト」に、この圧縮を適用すれば、CDの容量が何倍にもなるのです。もう、完全に夢物語ですが、近い将来、実現しましょう！

一番最高なのは、「Windows」がこの圧縮を標準サポートして、ハードディスクの「クラスタ」をブロックとみなし、「extend FAT」等として、普通にフォーマットできるようになれば、冥利に尽きますですけど・・・もう夢のまた夢ですね。不貞のデータサイズになるし。昔はそういう圧縮ソフトもありましたけどねぇ。