bzip2 and libbzip2, version 1.0.3

A program and library for data compression

Julian Seward, http://www.bzip.org

bzip2 and libbzip2, version 1.0.3: A program and library for data compression

Table of Contents

2. How to use bzip2

Table of Contents

2.3. DESCRIPTION

`bzip2`

50 bytes. Random data (including the output of most file compressors) is coded at about 8.05 bits per byte, giving an expansion of around 0.5%.

than a block. For example, compressing a file 20,000 bytes long with the flag `-9` will cause the compressor to allocate around 7600k of memory, but only touch 400k + 20000 * 8 = 560 kbytes of it. Similarly, the decompressor will allocate 3700k but only touch 100k + 20000 * 4 = 180 kbytes.

Here is a table which summarises the maximum memory usage for different block sizes. Also recorded is the total compressed size for 14 files of the Calgary Text Compression Corpus totalling 3,141,622 bytes. This column gives some feel for how compression varies with block size. These figures tend to understate the advantage of larger block sizes for larger files, since the Corpus is dominated by smaller files.

worst-case and average-case compression time is in the region of 10:1. For previous versions, this figure was more

3. Programming with `libbzip2`

Table of Contents

Yoshioka also contributed modification4-250hoka lown4-250hheth lraryn4-250hokaindowon4-250Dmo

`BZ_MEM_ERROR`

Returned when a request to allocate memory failed. Note that the quantity of memory needed to decompress a stream cannot be determined until the stream's header has been read. So `BZ2_bzDecompress` and `BZ2_bzRead` may return `BZ_MEM_ERROR` even though some of the compressed data has been re0 1]TJ -189.867 -11.

the standard sorting algorithm to a fallback algorithm. The fallback is slower than the standard algorithm by perhaps a factor of three, but always behaves reasonably, no matter how bad the input.

Lower valu1s of `workFactor` reduce the amount of effort the standard algorithm will expend before r1sorting to the fallback. You should set this parameter carefully; too low, and many inputs will be handled by the fallback algorithm and so compress rather slowly, too high, and your average-to-worst case compression times can become very large. The default valu1 of 30 gives reasonabl1 behaviour over a wid1 rang1 of circumstances.

Allowabl1 valu1s rang1 from 0 to 250 inclusive. 0 is a special case, equivalent to using the default valu1 of 30.

Note that the compressed output generated is the same regardless of whether or not the fallback algorithm is used.

```
BZ_CONFIG_ERROR
  if the library has been mis-compiled
BZ_PARAM_ERROR
  if ( small != 0 && small != 1 )
  or (verbosity <; 0 || verbosity > 4)
BZ_MEM_ERROR
  if insufficient memory is available
```

```
BZ_PARAM_ERROR
  if strm is NULL or strm->s is NULL
  or strm->avail_out < 1
BZ_DATA_ERROR
  if a data integrity error is detected in the compressed stream
BZ_DATA_ERROR_MAGIC
  if the compressed stream doesn’t begin with the right magic bytes
BZ_MEM_ERROR
  if there wasn’t enough memory available
BZ_STREAM_END
  if the logical end of the data stream was detected and all
  output in has been consumed, eg s-->avail_out > 0
BZ_OK
  otherwise
```

Allowable next actions:

```
BZ2_bzDecompress
  if BZ_OK was returned
BZ2_bzDecompressEnd
  otherwise
```

3.3.6. BZ2_bzDecompressEnd

```
int BZ2_bzDecompressEnd ( bz_stream *strm 33 Qytes
```

- If bzerror

```
BZ_PARAM_ERROR
  if b is NULL or buf is NULL or len < 0
BZ_SEQUENCE_ERROR
  if b was opened with BZ2_bzWriteOpen
BZ_IO_ERROR
  if there is an error reading from the compressed file
BZ_UNEXPECTED_EOF
  if the compressed file ended before
  the logical end-of-stream was detected
BZ_DATA_ERROR
  if a data integrity error was detected in the compressed stream
BZ_DATA_ERROR_MAGIC
  if the stream does not begin with the requisite header bytes
  (ie, is not a 0 259.-426(data)-425(file).)-852(This)-426(is)-426(really)]TJ 0 -11.955 T
```

```
  BZ_CONFIG_ERROR
    if the library has been mis-compiled
  BZ_PARAM_ERROR
    if f is NULL
    or blockSize100k < 1 or blockSize100k > 9
  BZ_IO_ERROR
    if ferror(f) is nonzero
  BZ_MEM_ERROR
    if insufficient memory is available
  BZ_OK
    otherwise
```

Possible return values:

```
  Pointer to an abstract BZFILE
    if bzerror is BZ_OK
  NULL
    otherwise
```

Allowable next action 0 j 0 w 0 0 468 59.7t o09action 0 j 0 w 0 0 468 59.7t o 0 6860.772 cm 0.949 0.949 0.97646 rg 0.949 0.949 0.976
if bzerror is BZ_OKless(fZ_OK
otherwise

```
  if the library has been mis-compiled
  BZ_CONFIG_ERROR
```

3.7. Using the library in a `stdio`-free environment

3.7.1. Getting rid of `stdio`

In a deeply embedded application, you might want to use just the memory-to-memory functions. You can do this conveniently by compiling the library with preprocessor symbol `BZ_NO_STDIO` defined. Doing this gives you a library containing only raryfollowing ea

Everything related to Windows has been contributed by Yoshioka Tsuneo (`QWF00133@niftyserve.or.jp` / `tsuneo-y@is.aist-nara.ac.jp`

- Recompile the program with no optimisation, and see if it works. And/or try a different compiler. I heard all sorts of stories about various flavours of GNU C (and other compilers) generating bad code for `bzip2`, and I've run across two such examples myself.

  2.7.X versions of GNU C are known to generate bad code from time to time, at high optimisation levels. If you get problems, try using the flags `-O2 -fomit-frame-pointer -fno-strength-reduce`. You should specifically *not110(ou)-295(shou470.564 640.299 cm 0 0 0 rg 0 0 0 RG 1 0 0 1 -470.564 -640.299 cm BT /F33 9.963 Tf1510.037 64*
  ˝bv not

If you want a compressor and/or library which is faster, uses less memory but gets pretty good compression, and has minimal latency, consider Jean-loup Gailly's and Mark Adler's work, `zlib-1.2.1`