ALGORITMA: Jurnal Ilmu Komputer dan Informatika
Volume: 02, Number : 01, April 2018 ISSN 2598-6341 (online)

ANALISIS PERBANDINGAN PEMAMPATAN DATA TEKS DENGAN MENGGUNAKAN METODE *HUFFMAN* DAN *HALF – BYTE*

Supiyandi[1], Okta Frida[2]
[1]Fakultas Ilmu Komputer Program Studi Sistem Komputer
[2]Program Studi Teknik Komputer
[1]Universitas Pembangunan Panca Budi Medan
[2]STMIK BUDIDARMA Medan
[1]Jl Gatot Subroto KM 4.5 Medan
[2]Jl. S.M. Raja Medan
Email : *supiyandi@dosen.pancabudi.ac.id[1], karyaprima.lkp@gmail.com[2]*

Abstract

Generally the use of conventional methods in data compression can not overcome the performance of data manipulation in saving memory usage and speed in data communications. Using a new manipulation method that applies the Huffman compression algorithm and the Half-Byte alphabet can compress data from the user interface before it is stored in the repository, so performance in manipulating the data will be better. With the new method applied to an application software using programming language. Which of the two types of algorithms has a larger ratio (ratio) as a benchmark in the performance of data manipulation. With good performance, it will be able to save the use of memory space on the repository and fast in transmitting data in communicate between the user interface with the repository in the data manipulation.

1. Pendahuluan

Pada saat ini kebutuhan akan kapasitas penyimpanan yang besar semakin penting. Kebutuhan ini, disebabkan oleh data yang harus disimpan makin lama semakin bertambah banyak, khususnya bagi dunia usaha dan perbankan ataupun instansi yang sudah menggunakan komputerisasi. Perusahaan tersebut umumnya sangat membutuhkan kapasitas yang sangat besar, untuk menyimpan semua data penting. Penyimpanan tersebut bukan hanya dialokasikan pada satu tempat saja. Tetapi mereka juga akan menyimpan data atau *file* pada tempat yang lain. Meskipun yang disimpan tersebut sama, hal ini berguna untuk *backup data*. *Backup data* sangat perlu dilakukan karena tidak ada yang menjamin suatu data pada tempat penyimpanan didalam komputer tidak akan mengalami kerusakan. Adapun kapasitas penyimpanan yang harus disediakan untuk menampung semua hal tersebut.

Pada dasarnya data-data tersebut di-mampat-kan (*compress)* terlebih dahulu sehingga tempat yang dibutuhkan memori semakin sedikit dan waktu yang dibutuhkan untuk berkomunikasi lebih pendek, sehingga kegagalan dalam manipulasi data lebih sedikit. Metode pemampatan yang digunakan saat ini adalah dengan cara pemampatan *file* yang sudah jadi (data asal) lalu dimampatkan dan baru dikomunikasikan. Setelah sampai data hasil pemanpatan (*compress*) tersebut pada si penerima, lalu dilakukan penirmampatan (*de-compress)* untuk mengembalikan ke bentuk asal, baru *file* tersebut dapat digunakan.

Dalam aplikasi komunikasi data pada saat ini banyak melakukan transmisi data per-*record* dalam bentuk isi data (*contens)* yang berbentuk teks, seperti pengisian (*entry data*), pembacaan data (informasi). Sehingga lambatnya kinerja komputer karena *performance* rendah, setelah menghabiskan beberapa waktu, komunikasi data menjadi lambat dan bahkan terputus sehingga harus melakukan lagi komunikasi dari awal, apalagi dengan item (*field*) yang banyak.

2. Metodologi

Metode merupakan suatu cara atau teknik yang sistematik untuk memecahkan suatu kasus sehingga memberikan hasil sesuai dengan yang diharapkan. Penulis menggunakan studi kepustakaan (*library research*), yaitu menggunakan sumber-sumber melalui buku, jurnal, tugas akhir, tesis maupun disertasi, *browsing* melalui internet, serta sumber-sumber lain yang relevan untuk digunakan dalam penulisan ilmiah ini. Studi kepustakaan dalam penelitian ini adalah hal-hal yang berkaitan dengan permasalahan pemampatan *file* dengan penyelesaian menggunakan metode *Huffman* dan *Half-Byte*.

2.1 Pengkodean Karakter

Setiap data dalam berbentuk karakter yang digunakan oleh *User* akan diterjemahkan kedalam kumpulan jajaran bit. Untuk menterjemahkan karakter-karakter tersebut digunakan ASCII (*American Standart Code for Information Interchange*), merupakan kode yang digunakan secara umum pada saat ini. ASCII merupakan kombinasi kode 8 bit, yang terdiri atas 7 bit data dan 1 bit parity, sehingga mempunyai 2^7 atau 128 kode karakter yang berbeda dan unik yang terdiri dari bit 0 dan bit 1. Kode ini digunakan dalam Personal

Computer (PC). Piranti yang menggunakan kode ini perlu menterjemahkan 1 bit didepan sebagai parity. Bit parity berfungsi sebagai tanda kesalahan dalam pengiriman data selama komunikasi, yang terdiri atas parity genap (bit 1 apabila jumlah bit 1 dalam 7 deretan bit data berjumlah genap) dan parity ganjil (bit 1 apabila jumlah bit 1 dalam 7 deretan bit data berjumlah ganjil). Pengkodean karakter ASCII dapat dilihat pada gambar 1 berikut ini :

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	sp
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

Gambar 1. Kode ASCII dibakukan oleh ANSI (*American National Standards Institute*) menjadi standar ANSI X3.4-1986

Untuk pengembangan karakter lanjut, bit parity dapat dijadikan data sehingga menjadi Extended ASCII seperti gambar 2 berikut ini :

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	Pts	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Gambar 2. Kode ASCII *Extended* Karakter

Dalam penulisan karya ini menggunakan pengkodean ASCII, karena kode ASCII merupakan kode standar yang digunakan dalam komputer mikro atau *Personal Computer* (PC). Pengujian dan aplikasi penelitian ini nantinya akan menggunakan personal komputer.

2.2 Bentuk Data

Data merupakan segala sesuatu yang mempunyai nilai (*Value*) yang dapat diolah menjadi informasi. Spesifikasi dari data terdiri atas :

1. Nama Data (*Data name*)
2. Type Data (*data type*)
3. Panjang Data (*Data Width*)

Dari Spesifikasi data dapat dikelompokkan atas beberapa bentuk data yang akan diolah, yaitu :

1. Data Teks, yaitu data yang mempunyai nilai berbentuk karakter (*character*), dimana karakter ini terdiri atas :
   a. Abjad, yaitu karakter yang mempunyai nilai A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, dan Z, baik huruf besar maupun huruf kecil.
   b. Angka, yaitu karakter yang mempunyai nilai 0, 1, 2, 3, 4, 5, 6, 7, 8, dan 9.
   c. Simbol, yaitu karakter yang mempunyai nilai selain huruf dan angka, seperti ; ., , , /, \, `, ‘, “, ;, :, ~, !, @, #, $, %, ^, &, *, (,), _, -, =, +, |, dan lain-lain.
2. Data gambar yaitu data yang mempunyai nilai yang berbentuk gambar atau visual, seperti :
   a. Dilarang berhenti (*Stop*)
   b. Dilarang merokok, dll.

3. Data Suara, yaitu data yang mempunyai nilai yang hanya bisa ditangkap oleh alat pendengaran, *microphone* atau peralatan elektronik lainnya.
4. Data *Video*, yaitu data yang mempunyai nilai gambar yang dapat bergerak.

Bentuk data yang digunakan dalam penelitian ini adalah data teks, dimana dalam aplikasi pengolahan data banyak berbentuk karakter yang dapat diinputkan dan ditampilkan pada *User Interface*.

2.3 *Repository*

Repository adalah media penyimpanan data yang digunakan pada system computer yang bersifat permanent, baik itu merupakan program atau data yang akan dioperasikan oleh *processor* komputer. Data dan program yang disimpan dalam *Repository* bersifat *non volatile*, dimana data dan program yang ada tidak akan hilang walaupun tegangan listrik terputus ke *Repository*. Setiap data dan program yang dibutuhkan oleh *User Interface* dapat diambil atau dikomunikasikan dari *Repository*.

2.4 Pemampatan (*Compress*) *File* dan Data

Pemampatan *file* akan menjadikan kapasitas memori yang dibutuhkan file lebih kecil dengan cara melakukan pengkodean baru (*coding*) terhadap isi *file*. Jika pemampatan berhasil dikecilkan separuh, maka kecepatan penghantaran secara tidak langsung bertambah dua kali lipat (dengan menggunakan peranti pemampatan *file* kita dapat menggunakan jalur atau kabel yang berkecepatan lebih rendah). Peranti ini mungkin akan menggantikan frasa atau data yang panjang dengan satu data khas yang pendek.

Dalam dunia *computer* dan internet, pemampatan *file* digunakan dalam berbagai keperluan, mem-*backup* data tidak perlu menyalin semua *file* aslinya, dengan memampatkan (mengecilkan ukurannya) *file* tersebut terlebih dahulu, maka kapasitas tempat penyimpanan yang diperlukan akan menjadi lebih kecil. Jika sewaktu-waktu data tersebut diperlukan, baru dikembalikan ke *file* aslinya.

2.4.1 Kategori Pemampatan

Pemampatan terhadap isi (*value*) dari data yang dilakukan dapat dikelompokan atas 2 (dua) kategori, yaitu :

1. *Lossless*, yaitu pemampatan yang dilakukan tidak menghilangkan kandungan asal data, seperti membuang atau merubah kandungan asal selama terjadinya pemampatan. Kategori ini banyak digunakan dalam pemampatan data teks.
2. *Lossy*, yaitu pemampatan yang dilakukan dengan membuang sedikit kandungan asal dari data, dimana data tersebut banyaknya terjadi penumpukan nilai atau adanya nilai yang tidak dibutuhkan (mengandung nilai yang tidak mempunyai makna), seperti bingkai gambar, ruang kosong dan lain-lain. Contoh pemampatan ini digunakan pada data gambar (*image*) atau suara.

Karena data yang diolah adalah data teks, maka kategori pemampatan yang digunakan *Lossless*, dimana tidak dilakukan pemotongan atau pembuanagn terhadap data teks yang tidak diperlukan dalam suatu isi atau *contents* setiap item.

2.4.2 Nisbah (Ratio) Pemampatan

Nisbah (*Ratio*) hasil pemampatan (*compress*) merupakan indicator untuk mengetahui *performance* dari hasil sebuah metoda pemampatan. Untuk mendapatkan nisbah dengan menggunakan rumus sebagai berikut :

$$Nisbah = 100\% - \left(\frac{kapasitas\_hasil\_kompres}{kapasitas\_file\_asli} x100\% \right)$$

Besarnya nilai nisbah (*ratio*) yang didapatkan dari media pemampatan, maka *performance* manipulasi data dari *User Interface* ke *Repository*akan meningkat.

2.5 Algoritma *Half-Byte*

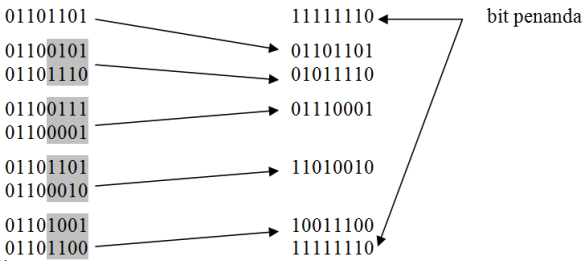
Alogaritma *Half-Byte* memanfaatkan empat *bit* sebelah kiri yang sering sama secara berurutan terutama pada file-file teks. Misalnya pada suatu file teks berisi tulisan “mengambil”, dalam heksadesimal dan biner karakter-karakter tersebut diterjemahkan pada tabel 1 sebagai berikut :

Tabel 1. Pengkodean Algoritma *Half-Byte*

Karakter	Heksadesimal	Biner
m	6D	01101101
e	65	01100101
n	6E	01101110
g	67	01100111
a	61	01100001
m	6D	01101101
b	62	01100010
i	69	01101001
l	6C	01101100

Karakter-karakter tersebut memiliki empat bit sebelah kiri yang sama yaitu 0110. Kondisi seperti inilah yang dimanfaatkan oleh Algoritma *Half-Byte*. Saat karakter yang empat bit pertamanya sama diterima secara berderet tujuh kali atau lebih, algoritma ini memampatkan data tersebut dengan *bit* penanda kemudian karakter pertama dari deretan empat *bit* yang sama diikuti dengan pasangan empat *bit* terakhir deretan berikutnya dan ditutup dengan *bit* penutup.

Algoritma ini paling efektif pada *file* teks dimana biasanya berisi teks-teks yang memiliki empat *bit* pertama yang sama. Agar lebih jelas Algoritma *Half-Byte* dapat digambarkan pada gambar 3 berikut ini :



Gambar 3. *Bit parity* Algoritma *Half-Byte*

Deretan data sebelah kiri merupakan deretan data pada file asli, sedangkan deretan data sebelah kanan merupakan deretan data hasil pemampatan dengan Algoritma *Half-Byte*. Langkah-langkah yang dilakukan adalah :

1. Lihat apakah terdapat deretan karakter yang 4 bit pertamanya sama secara berurutan tujuh karakter atau lebih, jika memenuhi lakukan pemampatan. Pada contoh diatas deretan karakter yang sama secara berurutan sebanyak 9 karakter, jadi dapat dilakukan pemampatan.
2. Berikan bit penanda pada file pemampatan, bit penanda disini berupa 8 deretan bit (1 byte) yang boleh dipilih sembarang asalakan digunakan secara konsisten pada seluruh bit penanda pemampatan. Bit penanda ini berfungsi untuk menandai bahwa karajter selanjutnya adalah karakter pemampatan sehingga tidak membingungkan pada saat mengembalikan file yang sudah dimampatkan ke file aslinya. Pada contoh diatas bit penanda ini dipilih 11111110.
3. Tambahkan karakter pertama 4 bit kiri berurutan dari file asli, pada contoh diatas karakter pertama 4 bit kiri berurutan adalah 01101101.
4. Gabungkan 4 bit kanan karakter kedua dan ketiga kemudian tambahkan kefile pemampatan. Pada contoh diatas karakter kedua dan ketiga adalah 01100101 dan 01101110, gabungan 4 bit kanan kedua karakter tersebut adalah 01011110. Lakukan hal ini sampai akhir deretan karakter dengan 4 bit pertama yang sama.
5. Tutup dengan bit penanda pada file pemampatan.

Maka akan di dapat peroleh nisbah :

$$Nisbah = 100\% - \left(\frac{56}{72} x100\%\right) = 22{,}33\%$$

Untuk melakukan proses pengembalian ke data asli atau penirmampatan (*de-compress*), dilakukan langkah-langkah berikut ini :

1. Lihat karaketr pada hasil pemampatan satu-persatu dari awal sampai akhir, jika ditemukan *bit* penanda, lakukan proses pengembalian.
2. Lihat karakter setelah *bit* penanda, tambahkan karakter tersebut pada *file* pengembalian.
3. Lihat karakter berikutnya, jika bukan *bit* penanda, ambil 4 *bit* kanannya lalu gabungkan dengan 4 *bit* kanan karakter dibawahnya. Hasil gabungan tersebut ditambahkan pada *file* pengembalian. Lakukan sampai ditemukan *bit* penanda.

2.6 Algoritma *Huffman*

Tidak seperti algoritma *Half-Byte*, algoritma *Huffman* lebih rumit penanganannya. Dasar pemikiran algoritma ini adalah bahwa setiap karakter ASCII biasanya diwakili oleh 8 *bits*. Jadi misalnya suatu *file* berisi deretan karakter ABACAD" maka ukuran *file* tersebut adalah 6 x 8 bits = 48 *bit* atau 6 *bytes*. Jika setiap karakter tersebut diberi kode lain misalnya A=1, B=00, C=010, dan D=011, berarti kita hanya perlu file dengan ukuran 11 bits (10010101011), yang perlu diperhatikan ialah bahwa kode-kode tersebut harus unik atau dengan kata lain suatu kode tidak dapat dibentuk dari kode-kode yang lain.

Pada contoh diatas jika kode D kita ganti dengan 001, maka kode tersebut dapat dibentuk dari kode B ditambah dengan kode A yaitu 00 dan 1, tapi kode 011 tidak dapat dibentuk dari kode-kode yang lain. Selain itu karakter yang paling sering muncul, kodenya diusahakan lebih kecil jumlah bitnya dibandingkan dengan karakter yang jarang muncul, pada contoh diatas karakter A lebih sering muncul (3 kali), jadi kodenya dibuat lebih kecil jumlah *bit*-nya dibanding karakter lain.

Untuk menentukan kode- kode dengan kriteria bahwa kode harus unik dan karakter yang sering muncul dibuat kecil jumlah bitnya, kita dapat mengunakan algoritma *Huffman*. Sebagai contoh, sebuah file yang akan dimampatkan berisi karakter-karakter "PERKARA" dslsm kode ASCII masing-masing karakter dikodekan sebagai :

P = 50H = 01010000B
E = 45 H = 010000101B
R = 52H = 01010010B
K = 4BH = 01001011B
A = 41H = 01000001B

Maka jika diubah dalam rangkaian bit," PERKARA" menjadi:

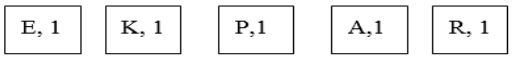
01010000010001010101001001001011010000010101001001000001
P E R K A R A

Yang berukuran 56 bit. Tugas kita yang pertama adalah menghitung frekuensi kemunculan masing-masing karakter, jika kita dihitung ternyata p muncul sebanyak satu kali, E sebanyak satu kali, R sebanyak 2 kali, K

sebanyak 1 kali dan A 2 kali. Jika disusun dari yang kecil:

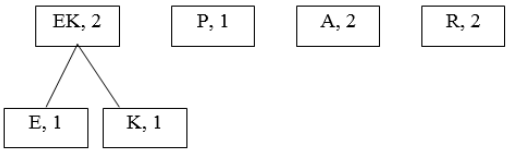
E= 1 A = 2
K = 1 R = 2
P= 1

Untuk karakter yang memiliki Frekunsi kemunculan sama E, K dan P disusun menurut kode ASCII-nya, begitu pula untuk A dan R. Selanjutnya buatlah node masing-masing karakter beserta frekuensinya seperti terlihat pada gambar 4.



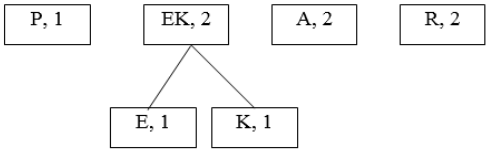
Gambar 4. Langkah 1 Algoritma *Huffman*

Ambil 2 node yang paling kiri (P dan E), lallu buat node baru yang merupakan gabungan dua node, node gabungan ini akan memoiliki cabang masing-masing node yang digabungkan tersebut. Frekuensi dari node gabungan ini adalah jumlah frekuensi cabang-cabangnya. Jika kita gambarkan akan menjadi seperti gambar 5.



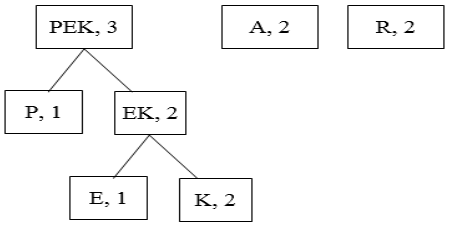
Gambar 5. Langkah 2 Algoritma *Huffman.*

Jika kita lihat frekuensi tiap node pada level paling atas, EK=2, P=1, A=2, dan R=2. Node-node tersebut harus diurutkan lagi dari yang paling kecil, jadi node EK harus digeser ke sebelah kanan node P dan ingat jika menggeser suatu node yang memiliki cabang, maka seluruh cabangnya harus diikutkan juga. Setelah diurutkan hasilnya akan menjadi sebagai seperti gambar 6.



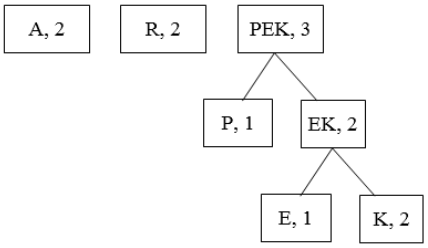
Gambar 6. Langkah 3 Algoritma *Huffman.*

Setelah node pada level paling atas diurutkan (level berikutnya tidak perlu diurutkan), berikutnya kita gabungkan kemballi 2 node paling kiri seperti yang pernah dikerjakan sebelumnya. Node P digabung dengan node EK menjadi node PEK dengan frekuensi 3 dan gambarnya akan menjaadi seperti gambar 7 :



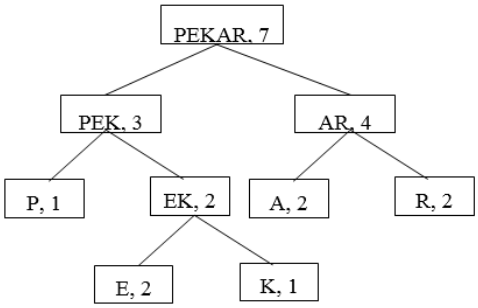
Gambar 7. Langkah 4 algoritma Huffman

Kemudian diurutkan lagi menjaadi seperti gambar 8 :



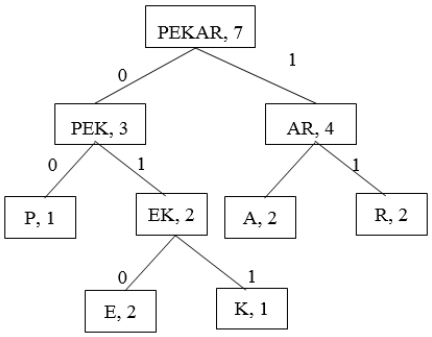
Gambar 8. Langkah 5 Algoritma Huffman

Demikian seterusnya sampai diperoleh pohon Huffman seperti gambar 9 berikut :



Gambar 10. Langkah 6 Algoritma Huffman

Setelah pohon Huffman terbentuk, berikan tanda bit 0 untuk setiap cabang ke kiri dan bit 1 setiap cabang ke kanan gambar 10 berikut :



Gambar 2-10 Langkah 7 Algoritma Huffman

Untuk mendapatkan kode Huffman masing-masing karakter, telusuri karakter tersebut dari node paling atas (PEKAR) sampai ke node karakter tersebut dan susunlah bit-bit yang dilaluinya. Untuk mendapatkan kode Karakter E, dari kode PEKAR kita harus menuju ke node PEK melalui bit 0 dan selanjutnya ke node EK melalui bit 1, dilanjutkan ke node E melalui bit 0, jadi kode dari karakter E adalah 010. Untuk mendapatkan kode karakter K, dari node PEKAR kita harus menuju ke node PEK melalui bit 0 dan selanjutnya menuju ke node EK melalui bit 1, dilanjutkan ke node K melalui bit 1, jadi kode dari karakter K adalah 011. Untuk mendapatkan kode karakter P, dari node PEKAR kita harus menuju ke node PEK melalui bit 0 dan selanjutnya ke node P melalui bit 0, jadi kode dari karakter P adalah 00. Untuk mendapatkan kode karakter A, dari node PEKAR kita harus menuju ke node AR melalui bit 1 dan selanjutnya menuju ke node A melalui bit 0, jadi kode dari karakter A adalah 10. Terakhir untuk mendapatkan kode Karakter R, dari node PEKAR kita harus menuju ke node AR melalui bit 1 dan selanjutnya menuju ke node R melalui bit 1, jadi kode dari karakter R adalah11. Hasil akhir kode *Huffman* dari file di atas adalah :

E = 010
K = 011
P = 00
A = 10
R = 11

Dengan kode ini, file yang berisi karakter-karakter "PERKARA" akan menjadi lebih kecil, yaitu :
00 010 00 011 10 11 10 = 16 bit
P E R K A R A

Dengan alogaritma *Huffman* berarti file ini dapat kita hemat sebanyak 56 – 16 = 40 bit.

$$\text{Nisbah} = 100\% - \left(\frac{16}{56} \text{x} 100\right) = 71{,}43\%$$

Untuk proses pengembalian ke file aslinya, kita harus mengacu kembali kepada kode Huffman yang telah dihasilkan, seperti contoh di atas pemampatan adalah : 000101101100 1110.

Ambilah satu-persatu bit hasil pemampatan mulai dari kiri, jika bit tersebut termasuk dalam daftar kode, lakukan pengembalian, jika tidak ambil kembali bit selanjutnya dan jumlahkan bit tersebut. Bit pertama dari hasil pemampatan di atas adalah 0, karena 0 tidak termasuk dalam daftar kode kita ambil lagi bit kedua yaitu 0, lalu digabungkan menjadi 00, jika kita lihat daftar kode 00 adalah kode dari karakter P.

Selanjutnya bit ketiga diambil yaitu 0, karena 0 tidak terdapat dalam daftar kode, kita ambil lagi bit keempat yaitu 1dan kita gabungkan menjadi 01. 01 juga tidak terdapat dalam daftar, jadi kita ambil kembali bit selanjutnya yaitu 0 dan digabungkan menjadi 010. 010 terdapat dalam daftar kode yaitu karakter E. Demikian selanjutnya dikerjakan sampai bit terakhir sehingga akan didapatkan hasil pengembalian yaitu PERKARA.

Pemrograman dengan mengunakan algoritma *Huffman*, menyertakan daftar kode dalam file pemampatan, selain itu jumlah byte disertakan sebagai acuan file untuk pengembalian kebentuk file aslinya.

3. Kesimpulan dan Saran

Dengan diselesaikannya penulisan ini yang mencakup perbandingan pemampatan file algorithma Huffman dengan *Half-byte* dalam melakukan manipulasi data teks dapat diperoleh beberapa kesimpulan dan saran:

- Merancang suatu metode pemampatan baru, dimana biasanya yang umum dilakukan adalah melakukan pemampatan data yang sudah tersimpan dalam *repository*, kedalam metode pemampatan data sebelum disimpan kedalam *respository*.
- Dengan melakukan perbandingan dua jenis algoritma pemampatan yaitu algorithma Huffman dengan *Half-Byte* terhadap data teks, maka yang mempunyai pemampatan yang lebih baik dengan nisbah yang lebih tinggi adalah *Huffman*.
- Data yang diolah tidak hanya dalam bentuk satu buah file data, tetapi dapat manipulasi data dalam bentuk database

4. Daftar Pustaka

Adam Osborne dan Davis Bunnel, *Pengantar* Komputer – mikro, terjemahan oleh Setiyo Utomo, Ir. Jakarta, Erlangga, 1986.

Aji Supriyanto, Pengantar Teknologi Informasi, Salemba – Jakarta, 2005

Herry Sujaini dan Yessi Mulyani, Algoritma Runlength Halfbyte & Huffman untuk Pemampatan File, Bandung, 2000

Tony Suryanto, Pemampatan File dengan Algoritma Huffman, Jakarta, Dinastindo, 1995

Willia Stalling, Data and Computer Communications, 5th Edition, Prentice Hall, 1997