Laboratório 6: Heap, Heapsort e Árvore de Huffman

Entrega até domingo, 21/4, às 23:59h

Em sala, discutimos a implementação de listas de prioridades usando heaps. Nesse trabalho iremos explorar essa estrutura para implementar o algoritmo de ordenação *heapsort*, e para criar a árvore de *huffman* para compactação de arquivos.

1. (4.0) Complete a implementação do módulo *heapmax.c*, terminando a implementação da função interna *heap_monta*, chamada pelo método *heap_cria*. Essa função recebe um array inicial de prioridades, e realiza a construção do heap em tempo linear (de acordo com o algoritmo que vimos em aula). Você pode comparar o tempo de execução entre criar um heap em tempo linear, e criar utilizando o método de inserção *heap_insere*. Este teste já está implementado no arquivo de *testeheap.c*, gerando uma lista de números aleatórios. Você pode mudar o tamanho da entrada pela variável *numero_de_valores*.

2. (4.0) Agora iremos implementar o *heapsort*. A função já está esboçada no arquivo *heapmax.c*, onde você deve terminar a implementação da função *cria_listaordenada*, que recebe uma lista de prioridades (um heap) e a utiliza para criar um vetor contendo os inteiros da lista inicial em ordem **decrescente**. Note que, ao invés de inserir os elementos removidos do heap no final da lista de prioridades, você deve colocá-los em um novo vetor ordenado!

3. (1.0) Por fim, temos uma implementação do algoritmo de huffman, utilizado para compressão de arquivos de texto. Caso esteja em uma máquina Linux, baixe o arquivo **huffman.o**, e gere o executável a partir da seguinte linha de código:

   gcc -o t testehuffman.c heaparvmin.c huffman.o

   Caso esteja em uma máquina Windows, e utilizando o Visual Studio, baixe o arquivo **huffmanx64.obj**, e adicione ao projeto junto do arquivo **huffman.h** (o visual studio irá sinalizar a falta da implementação dos métodos no .h que estão no .obj, mas deve rodar a aplicação sem problemas).

   Ao olhar o arquivo **huffman.h**, você verá que foram implementados os métodos relacionados com 4 estruturas:
   1. A estrutura de uma árvore onde cada nó guarda um caractere e a sua respectiva frequência.
   2. A tabela de frequências, construída a partir de um arquivo de entrada.
   3. A árvore de huffman, que apenas recebe o nó raiz de uma árvore já montada.
   4. O dicionário contendo os respectivos códigos binários associados com cada caractere.

A implementação disponibilizada possui suporte para caracteres da tabela ascii, ou seja, são suportados 128 símbolos (veja a tabela na questão 4).

Para criar a árvore de huffman, foi utilizada a implementação de um heap mínimo, definido pelo arquivo *heaparvmin.c*. Com base no heap que vimos na questão 1, termine a implementação dos métodos *heap_monta* e *corrige_acima* (lembre-se que agora estamos tratando um heap **mínimo**). Para verificar a frequência de um nó, você pode utilizar o método *arvno_frequencia(ArvNo* r)*.

4. (1.0) Agora, faça a implementação do método **heaparvmin_criaarvorehuffman**, que cria a árvore de huffman seguindo o algoritmo visto em aula. Ao finalizar a implementação, você pode testar a compactação e descompactação com o arquivo de texto enviado. Caso queira testar outros arquivos, cuide para utilizar apenas os 128 símbolos disponíveis na tabela ascii.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Faça upload dos arquivos **heapmax.c** e **heaparvmin.c** no EAD até dia 21 de abril, domingo, às 23:59h. Lembre-se de fazer a entrega mesmo que não tenha chegado ao final do exercício.