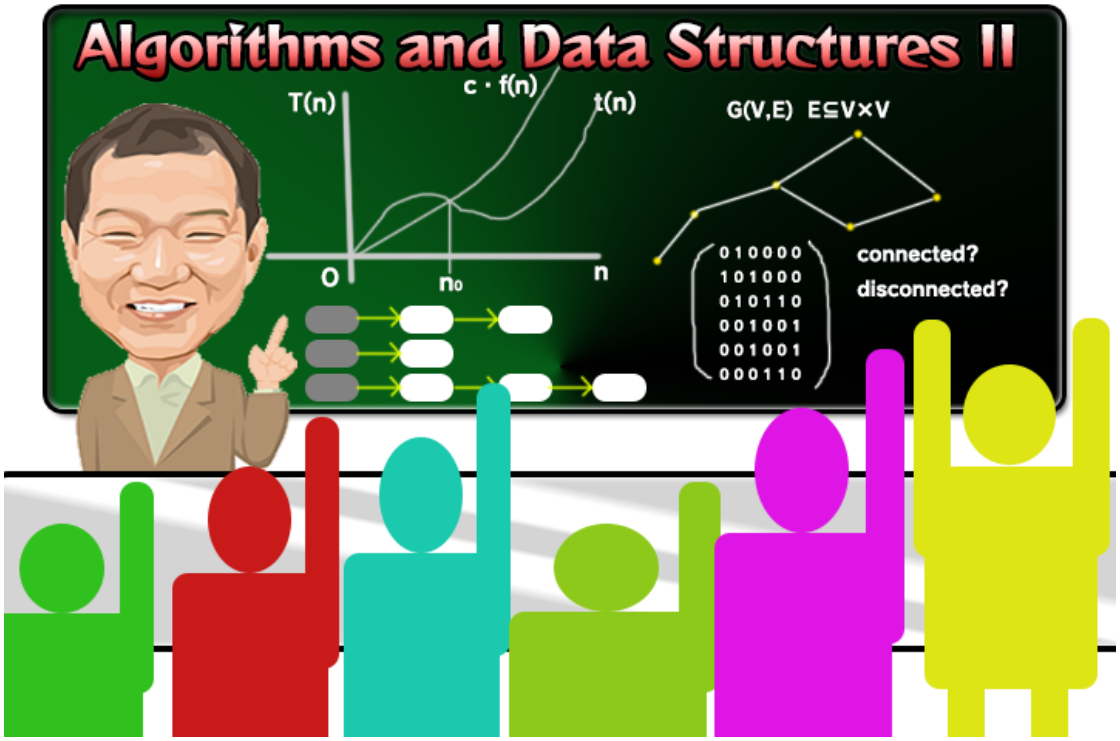


アルゴリズムとデータ構造 II
Algorithms and Data Structures II

Exercise 08. Algorithm Design Techniques I

Exercise 08-11のために

アルゴリズム設計のためのストラテジー
（問題を解くための、広い意味での解法の考え方）

- Greedy Algorithm (Ex08)
- Divide and Conquer (Ex09)
- Dynamic Programming (Ex10)
- Backtracking (Ex11)

今回からは、「どのようにプログラムを書いたか？」までが重要です。
プログラムのソースコードの提出も必須です。

[Ex08] Greedy Algorithm (貪欲法)

・最適解を貪欲に狙っていく

⇒ 自分が現在どんな状況に置かれていても、その時で一番ベストな選択肢をとる

具体例

・Prim's Algorithm

・Dijkstra's Algorithm

⇒ 部分解TからV-Tに接続する頂点のうち、最小の重みで到達可能なものを選ぶ

・Kruskal's Algorithm

⇒ 全ての辺から最小の重みを次々取っていく

・Huffman Encoding (Ex08)

Huffman Encoding

・文字列を圧縮する

各文字が 8bit のバイナリで表されるとすると…

⇒ 30文字のテキストは240文字必要

⇒ 出現頻度が高い文字になるべく小さい バイナリを付ける

⇒ Huffman Encoding によって、全体の文字列のbit数が減らせる

（貪欲法を適用するポイント）

各文字の出現頻度を調べ、頻度の少ないものを２つ選び

二分木を構成していく

Huffman Encoding (Skelton)

・今回のスケルトンはやや大型で複雑

⇒ “**Greedy Algorithms**” を念頭に置いた上で

・ヒープ

・連結リスト

・Depth First Search

の合わせ技

プログラム全体の大枠は、**本日のクイズ（裏側）**で掴むこと

⇒ どういう処理が“**Greedy Algorithms**” なのか？

Huffman Encoding (Skelton)

・スケルトンのデータ構造

Node構造体　・・・講義資料に出てくる    らのこと

```
8
9   /* Heap structure */
10  typedef struct{
11    char word;                // Character
12    int count;                // Appearance
13    int visited;              // Visited flag (for DFS)
14    char bincode[SIZE];         // Encoded
15    int bits;
16    int articulation;         // Articulation flag (for Huffman binary tree creation)
17  }Node;
```

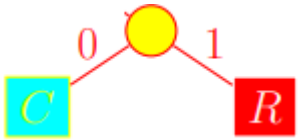
基本的には、文字と出現回数を保持するもの

変数 articulation は 出現回数の和をもつ  のことを表す

Huffman Encoding (Skelton)

・スケルトンのデータ構造

list構造体　・・・講義資料に出てくるツリーのこと




```
19  /* Linked list structure declaration */
20  struct list {
21    Node vertex;              // Contains of "Node" structure (Nested structure)
22    struct list *c_left;      // A pointer for left child in the binary tree
23    struct list *c_right;     // A pointer for right child in the binary tree
24  };
25  typedef struct list *NodePointer;
```

文字と**出現回数**を保持するNode構造体と、

その左右に何が繋がっているかを表す。

⇒　これをリスト形式にして繋ぐことでHuffman Treeを作る

Huffman Encoding (Skelton)

Step 1: 出現回数を数える

1次元配列 S[ ]に、入力文字列が入っている。

1文字ずつ順に調べて、それぞれが何回現れるかをカウント

```
97
98  /* (Step. 1) Compute the appearance of each character */
99   for(i=0;i<SIZE;i++) index[i]=0;
100
101  for(i=0;S[i]!='\0';i++){
102    /* [ Complete Here!! ] */;  // Count the appearance of each character
103  }
```

indexは255要素の1次元配列。

それぞれが、各文字の出現数を表すカウンタと思って数える。例えば・・・

index[65]は A の出現回数

Index[122]は z の出現回数

文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進
NUL	0	00	DLE	16	10	SP	32	20	0	48	30	@	64	40	P	80	50	`	96	60	p	112	70
SOH	1	01	DC1	17	11	!	33	21	1	49	31	A	65	41	Q	81	51	a	97	61	q	113	71
STX	2	02	DC2	18	12	"	34	22	2	50	32	B	66	42	R	82	52	b	98	62	r	114	72
ETX	3	03	DC3	19	13	#	35	23	3	51	33	C	67	43	S	83	53	c	99	63	s	115	73
EOT	4	04	DC4	20	14	$	36	24	4	52	34	D	68	44	T	84	54	d	100	64	t	116	74
ENQ	5	05	NAK	21	15	%	37	25	5	53	35	E	69	45	U	85	55	e	101	65	u	117	75
ACK	6	06	SYN	22	16	&	38	26	6	54	36	F	70	46	V	86	56	f	102	66	v	118	76
BEL	7	07	ETB	23	17	'	39	27	7	55	37	G	71	47	W	87	57	g	103	67	w	119	77
BS	8	08	CAN	24	18	(	40	28	8	56	38	H	72	48	X	88	58	h	104	68	x	120	78
HT	9	09	EM	25	19	)	41	29	9	57	39	I	73	49	Y	89	59	i	105	69	y	121	79
LF*	10	0a	SUB	26	1a	*	42	2a	:	58	3a	J	74	4a	Z	90	5a	j	106	6a	z	122	7a
VT	11	0b	ESC	27	1b	+	43	2b	;	59	3b	K	75	4b	[	91	5b	k	107	6b	{	123	7b
FF*	12	0c	FS	28	1c	,	44	2c	<	60	3c	L	76	4c	\¥	92	5c	l	108	6c	|	124	7c
CR	13	0d	GS	29	1d	-	45	2d	=	61	3d	M	77	4d	]	93	5d	m	109	6d	}	125	7d
SO	14	0e	RS	30	1e	.	46	2e	>	62	3e	N	78	4e	^	94	5e	n	110	6e	~	126	7e
SI	15	0f	US	31	1f	/	47	2f	?	63	3f	O	79	4f	_	95	5f	o	111	6f	DEL	127	7f

Huffman Encoding (Skelton)

Node構造体変数に、文字と出現回数を記録して…

```
106  for(i=0;i<SIZE;i++){
107    if(index[i]>0){//存在する文字の分だけ、ノードを作る（ヒープのための準備）
108      ++num_of_chars;
109      data[num_of_chars].word = i;
110      data[num_of_chars].count = index[i];
111      data[num_of_chars].visited = 0;
112      data[num_of_chars].articulation = -1;
113    }
114  }
115
116   for(i=1;i<=num_of_chars;i++){
117     printf("ID:%d   character:%c   appearance:%d\n",i,data[i].word,data[i].count);
118   }
119   printf("\n");
120
121
122
123  // ここからがHuffman Encodingのメインルーチン（貪欲法が絡んでくる）
124  /* (Step. 2) Create a Huffman binary tree based on the table made at Step. 1 */
125   i = num_of_chars;
126   construct_2(data,i);//出現数を持った文字たちをヒープに入れる
```

ヒープ(minimum heap)に入れる！（出現回数の大小で判定）

⇒ これが Step2 の重要な操作の１つに

ヒープ関係の関数は、「構造体のメンバーを見て」操作できるように、従来の実装を少し変える必要がある

Huffman Encoding (Skelton)

Step 2: Huffman Tree の構築

```
128  while(i>1){
129
130    /* Pick up the 2 minimum appearance characters */
131    couple[0]=  /* [ Complete Here!! (Write function call) ] */; //ヒープから？？？
132    couple[1]=  /* [ Complete Here!! (Write function call) ] */; //ヒープから？？？
133
134    /* Create the nodes for the picked up characters  木の左側により小さいものを置くようにする*/
135    if(couple[0].articulation == -1)left = make_1node(couple[0]); //左側に繋ぐ候補として、取り出したものが文字単体であればノードを作成する
136    else left = head[couple[0].articulation];  //文字たちを集めた「木」であれば、その根に繋ぐ準備をする
137    if(couple[1].articulation == -1)right = make_1node(couple[1]);
138    else right = head[couple[1].articulation];
139
140    /* Create the articulation node to combine the nodes for the picked up characters*/
141    newnode.count = /* [ Complete Here!! ] */ ; // 取り出した2つの節点の値（出現回数）の合計
142    newnode.articulation = num_of_newroot;
143    newnode.visited = 0;
144
145    /* Construct a linked-list for Huffman binary tree expressions */
146    head[num_of_newroot] = make_1node(newnode);
147    head[num_of_newroot]->c_left = /* [ Complete Here!! ] */;  //作成した左側用ノードを木の左側に接続する
148    head[num_of_newroot]->c_right = /* [ Complete Here!! ] */; //作成した右側用ノードを木の右側に接続する
149
150
151
152   num_of_newroot++;
153   /* [ Complete Here!! (Write function call) ] */; //作成したノードをヒープに戻す
154
155  }
```

とにかく、コードを書く前に
クイズの問題で
「貪欲法の流れ」を確認せよ！！

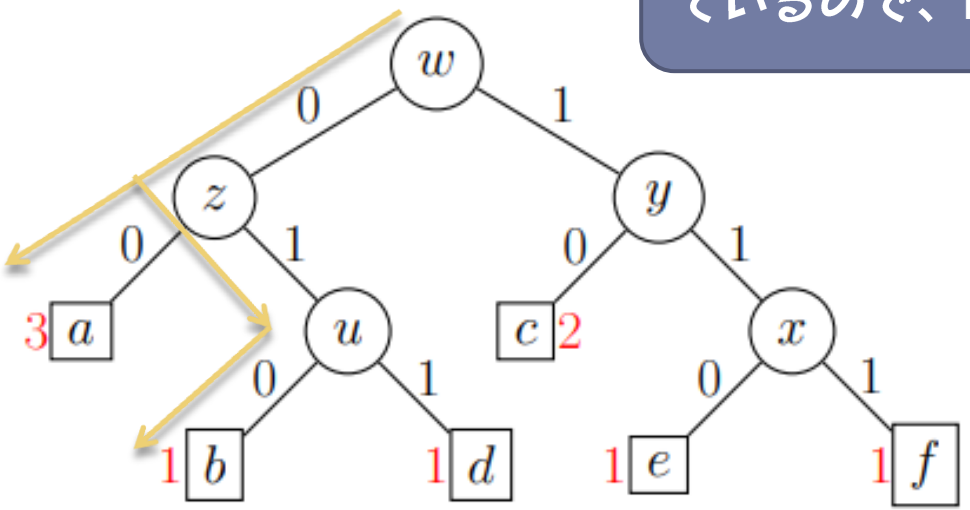
ここからは、講義スライドの通りに…

- **Node構造体変数 newnode に、新しく取り出したデータを組み込む**
- **準備した newnodeを、経過途中のTreeへ繋いでいく**

Huffman Encoding (Skelton)

出力ステップ：DFSによる探索

バイナリを付ける部分は完成しているので、DFSによる出力を！



List構造体を使って、木を繋いできたのは この出力のため。

Ex03とやることは全く同じ。再帰によって

「木を進めるだけ進む。ダメなら戻ってきて別の方向を探して進む」

（左右に進むための、アドレスを持った変数はどれだ…？）

（類題）コインの問題

・X円を硬貨で支払う場面

500円、100円、50円、10円、5円、1円玉を何枚かずつ所持

⇒ **支払いに使う硬貨の枚数を最小にする組み合わせは？**

（ただし、お釣り禁止）

（貪欲法を適用するポイント）

・X円を超えない範囲の、最大の額面の硬貨を使えるだけ使う

・使う硬貨の額面を徐々に減らす

Huffman Encoding よりも、格段に簡単。
とにかく、”Greedy Algorithms”のスタイルを掴んでください