Algorithmics I

Tutorials – Sheet 2

Dr. Gethin Norman

School of Computing Science
University of Glasgow

gethin.norman@glasgow.ac.uk

Question 1

Explain how the Huffman-tree-building algorithm can be extended so that it also determines the weighted path length of the tree that it constructs

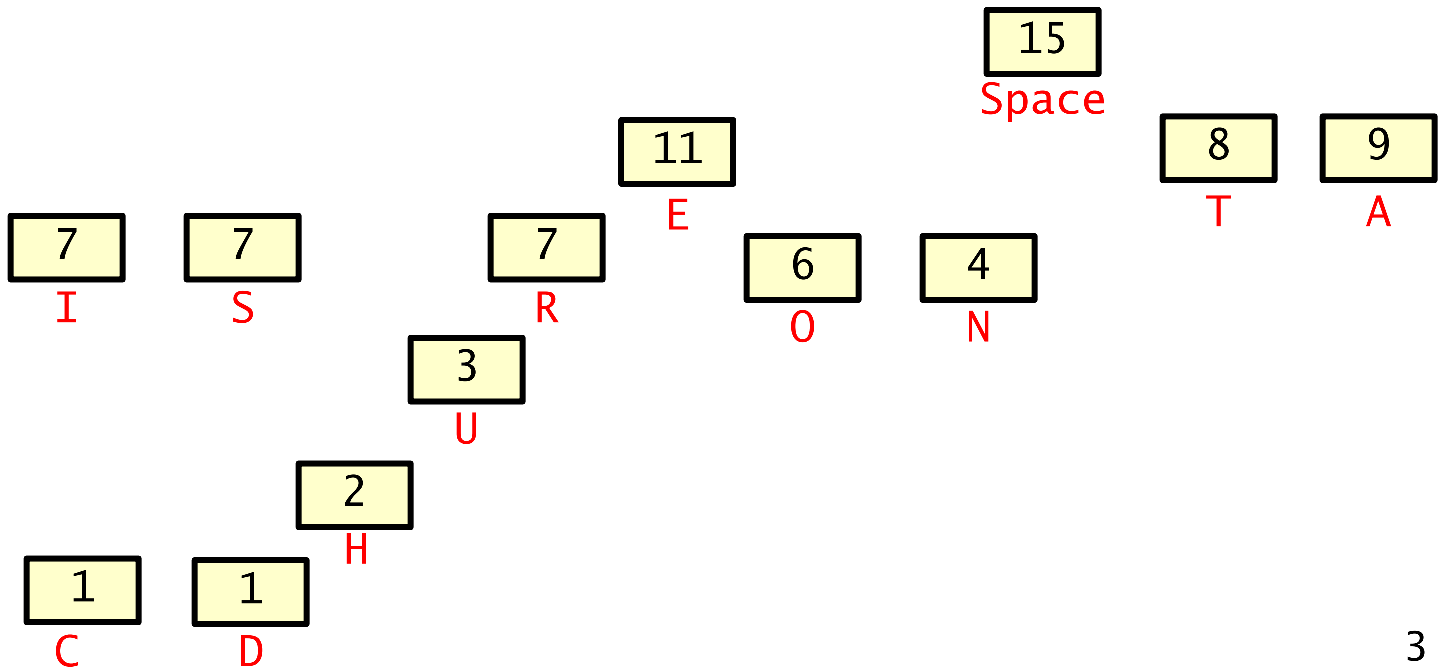
Question 1

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

First add leaves of Huffman tree

- characters with their frequencies label the leaf nodes



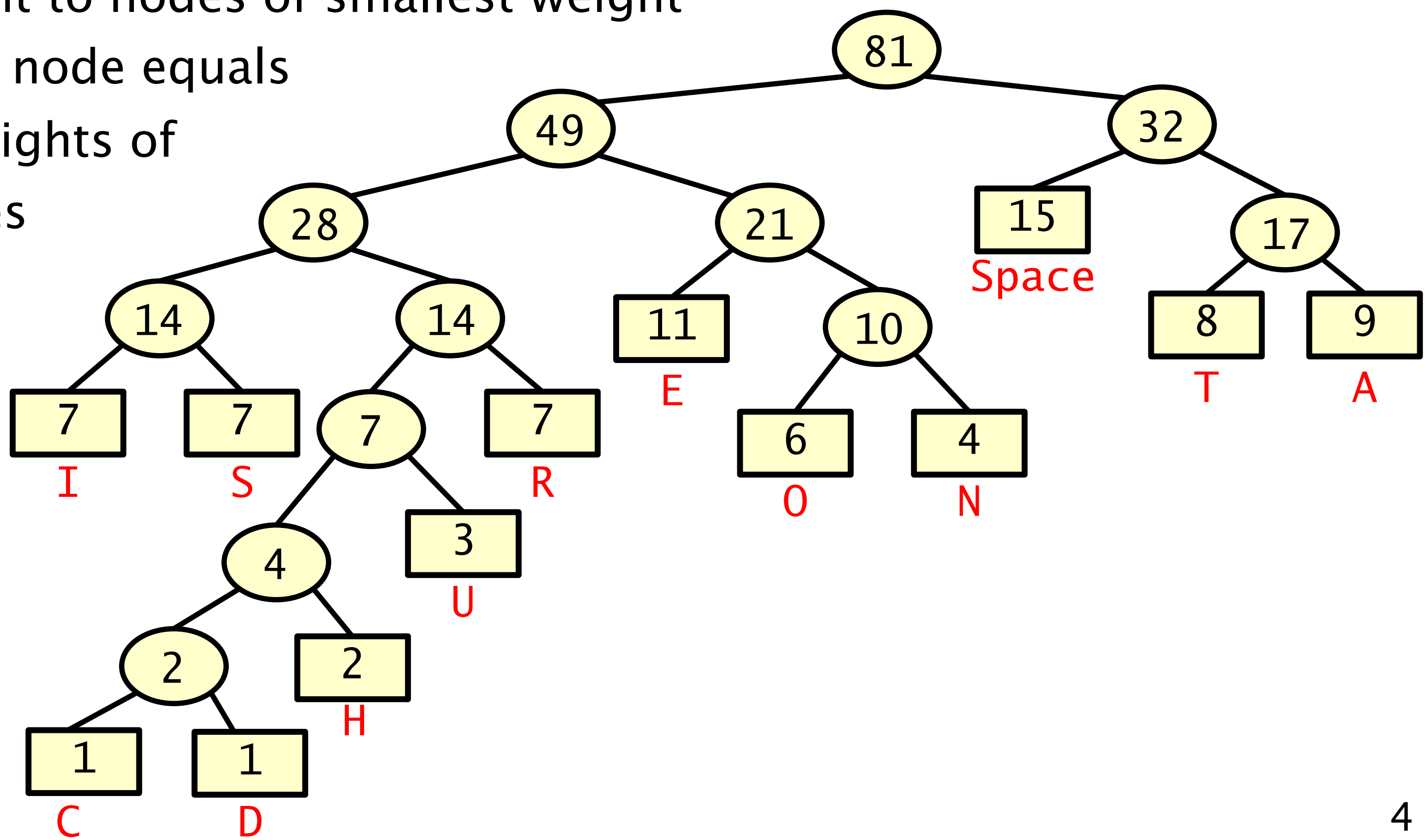
Question 1

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

Next, while there is more than one parentless node

- add new parent to nodes of smallest weight
- weight of new node equals sum of the weights of the child nodes



Question 1

Weighted path length (WPL) of a tree T

- `Σ (weight)×(distance from root)` where sum is over all leaf nodes

Algorithm for calculating the weight:

- initialise a variable `w` to zero
- each time a new `internal` node is created add weight of this node to `w`
- upon termination return `w`

So why is w equal to the weighted path length?

- for each leaf node, its weight is added to `w` every time an ancestor of the leaf is created
- the number of ancestors of a leaf equals its distance from the root

Question 2

Apply LZW algorithm to:

`peter piper picked a peck of pickled pepper`

Assume the dictionary initially contain the **128** characters of the basic `ascii` character set, represented by the code words `0,1,...,127`

Next available code word is **128**

Will leave this for you to do – the result will be included in the notes

LZW compression – Pseudo code

```
set current text position i to 0;
initialise codeword length k (say to 8);
initialise the dictionary d;

while (the text t is not exhausted) {

 identify the longest string s, starting at position i of text t
 that is represented in the dictionary d;
 // there is such string, as all strings of length 1 are in d

 output codeword for the string s; // using k bits

 // move to the next position in t
 i += s.length(); // move forward by the length of string just encoded
 c = character at position i in t; // character in next position

 add string s+c to dictionary d, paired with next available codeword;
// this involves adding a new leaf node if d is represented by a trie
// may have to increment the codeword length k to make this possible
}
```

Question 2

`text = peter piper picked a peck of pickled pepper`

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116

ASCII encoding of the characters appearing in the text to be compressed (given in the question)

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1				

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116

I will use the integer representation for the bit sequence to simplify the presentation

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112		

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101		

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129
3	3	t	116		

Question 2

`text = peter piper picked a peck of pickled pepper`

dictionary: "space":32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129
3	3	t	116	te	130

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129
3	3	t	116	te	130
4	4	e	101	er	131

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129
3	3	t	116	te	130
4	4	e	101	er	131
5	5	r	114	“r ”	132

Question 2

`text = `**`peter piper picked a peck of pickled pepper`**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129
3	3	t	116	te	130
4	4	e	101	er	131
5	5	r	114	“r ”	132
6	6	“ ”	32	“ p”	133

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129
3	3	t	116	te	130
4	4	e	101	er	131
5	5	r	114	“r ”	132
6	6	“ ”	32	“ p”	133
7	7	p	112	pi	134

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: "space":32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 "r ":132 " p":133 pi:134 ip:135

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	p	112	pe	128
2	2	e	101	et	129
3	3	t	116	te	130
4	4	e	101	er	131
5	5	r	114	"r "	132
6	6	" "	32	" p"	133
7	7	p	112	pi	134
8	8	i	105	ip	135

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: "space":32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 "r ":132 " p":133 pi:134 ip:135 per:136

step	position in string	longest string in dictionary	b	add to dictionary	code
9	1	pe	128	per	136

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137

step	position in string	longest string in dictionary	b	add to dictionary	code
9	9	pe	128	per	136
10	11	“r ”	132	"r p"	137

Question 2

`text = ` **`peter piper picked a peck of pickled pepper`**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138

step	position in string	longest string in dictionary	b	add to dictionary	code
9	9	pe	128	per	136
10	11	“r ”	132	"r p"	137
11	13	pi	133	pic	138

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139

step	position in string	longest string in dictionary	b	add to dictionary	code
9	9	pe	128	per	136
10	11	“r ”	132	"r p"	137
11	13	pi	133	pic	138
12	15	c	99	ck	139

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140

step	position in string	longest string in dictionary	b	add to dictionary	code
9	9	pe	128	per	136
10	11	“r ”	132	"r p"	137
11	13	pi	133	pic	138
12	15	c	99	ck	139
13	16	k	107	ke	140

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141

step	position in string	longest string in dictionary	b	add to dictionary	code
9	9	pe	128	per	136
10	11	“r ”	132	"r p"	137
11	13	pi	133	pic	138
12	15	c	99	ck	139
13	16	k	107	ke	140
14	17	e	101	ed	141

Question 2

`text = ` **`peter piper picked a peck of pickled pepper`**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142

step	position in string	longest string in dictionary	b	add to dictionary	code
9	9	pe	128	per	136
10	11	“r ”	132	"r p"	137
11	13	pi	133	pic	138
12	15	c	99	ck	139
13	16	k	107	ke	140
14	17	e	101	ed	141
15	18	d	100	“d ”	142

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143

step	position in string	longest string in dictionary	b	add to dictionary	code
9	9	pe	128	per	136
10	11	“r ”	132	"r p"	137
11	13	pi	133	pic	138
12	15	c	99	ck	139
13	16	k	107	ke	140
14	17	e	101	ed	141
15	18	d	100	“d ”	142
16	19	“ ”	97	“ a”	143

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144
18	21	“ p”	133	“ pe”	145

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144
18	21	“ p”	133	“ pe”	145
19	23	e	101	ec	146

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144
18	21	“ p”	133	“ pe”	145
19	23	e	101	ec	146
20	24	ck	139	“ck ”	147

Question 2

`text = ` **`peter piper picked a peck of pickled pepper`**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144
18	21	“ p”	133	“ pe”	145
19	23	e	101	ec	146
20	24	ck	139	“ck ”	147
21	25	“ ”	32	“ o”	148

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144
18	21	“ p”	133	“ pe”	145
19	23	e	101	ec	146
20	24	ck	139	“ck ”	147
21	25	“ ”	32	“ o”	148
22	26	o	111	of	149

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144
18	21	“ p”	133	“ pe”	145
19	23	e	101	ec	146
20	24	ck	139	“ck ”	147
21	25	“ ”	32	“ o”	148
22	26	o	111	of	149
23	27	f	102	“f ”	150

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150 “ pi”:151

step	position in string	longest string in dictionary	b	add to dictionary	code
17	20	a	97	“a ”	144
18	21	“ p”	133	“ pe”	145
19	23	e	101	ec	146
20	24	ck	139	“ck ”	147
21	25	“ ”	32	“ o”	148
22	26	o	111	of	149
23	27	f	102	“f ”	150
24	28	“ p”	133	“ pi”	151

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150 “ pi”:151 ic:152

step	position in string	longest string in dictionary	b	add to dictionary	code
25	30	i	105	ic	152

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150 “ pi”:151 ic:152 ckl:153

step	position in string	longest string in dictionary	b	add to dictionary	code
25	30	i	105	ic	152
26	31	ck	139	ckl	153

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150 “ pi”:151 ic:152 ckl:153 le:154

step	position in string	longest string in dictionary	b	add to dictionary	code
25	30	i	105	ic	152
26	31	ck	139	ckl	153
27	33	l	108	le	154

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: "space":32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 "r ":132 " p":133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 "d ":142 " a":143 "a ":144 " pe":145 ec:146 "ck ":147 " o":148 of:149 "f ":150 " pi":151 ic:152 ckl:153 le:154 "ed ":155

step	position in string	longest string in dictionary	b	add to dictionary	code
25	30	i	105	ic	152
26	31	ck	139	ckl	153
27	33	l	108	le	154
28	34	ed	141	"ed "	155

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150 “ pi”:151 ic:152 ckl:153 le:154 “ed ”:155 “ pep”:156

step	position in string	longest string in dictionary	b	add to dictionary	code
25	30	i	105	ic	152
26	31	ck	139	ckl	153
27	33	l	108	le	154
28	34	ed	141	“ed ”	155
29	36	“ pe”	145	“ pep”	156

Question 2

text = **peter piper picked a peck of pickled pepper**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150 “ pi”:151 ic:152 ckl:153 le:154 “ed ”:155 “ pep”:156 pp:157

step	position in string	longest string in dictionary	b	add to dictionary	code
25	30	i	105	ic	152
26	31	ck	139	ckl	153
27	33	l	108	le	154
28	34	ed	141	“ed ”	155
29	36	“ pe”	145	“ pep”	156
30	39	p	112	pp	157

Question 2

`text = `**`peter piper picked a peck of pickled pepper`**

dictionary: “space”:32 a:97 c:99 d:100 e:101 f:102 i:105 k:107 l:108 o:111 p:112 r:114 t:116 pe:128 et:129 te:130 er:131 “r ”:132 “ p”:133 pi:134 ip:135 per:136 r p:137 pic:138 ck:139 ke:140 ed:141 “d ”:142 “ a”:143 “a ”:144 “ pe”:145 ec:146 “ck ”:147 “ o”:148 of:149 “f ”:150 “ pi”:151 ic:152 ckl:153 le:154 “ed ”:155 “ pep”:156 pp:157

step	position in string	longest string in dictionary	b	add to dictionary	code
25	30	i	105	ic	152
26	31	ck	139	ckl	153
27	33	l	108	le	154
28	34	ed	141	“ed ”	155
29	36	“ pe”	145	“ pep”	156
30	39	p	112	pp	157
31	40	per	136		

Question 3

Suppose that the LZW compression algorithm is applied to the text:

ababababababab...

of length 100 (i.e. alternating sequence of a's and b's and 100 chars)

Determine the compression ratio and space saved

- assume that each character in the source file occupies 8 bits
- the initial dictionary size is 128
- the initial codeword size is 8

Hint:

- apply the LZW algorithm until a pattern emerges
- use this to compute the steps required to reach the 100th (and last) char
- from this you can find the number of code words used to encode the text

LZW compression – Pseudo code

```
set current text position i to 0;
initialise codeword length k (say to 8);
initialise the dictionary d;

while (the text t is not exhausted) {

 identify the longest string s, starting at position i of text t
 that is represented in the dictionary d;
 // there is such string, as all strings of length 1 are in d

 output codeword for the string s; // using k bits

 // move to the next position in t
 i += s.length(); // move forward by the length of string just encoded
 c = character at position i in t; // character in next position

 add string s+c to dictionary d, paired with next available codeword;
 // may have to increment the codeword length k to make this possible
}
```

Question 3

```
text = abababababababababababababab...
```

```
dictionary: a:0 b:1
```

Question 3

text = ababababababababababababab…

dictionary: a:0 b:1

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a		
2				
3				
4				
5				
6				
7				
8				
9				
10				

Question 3

```
text = ababababababababababababab...
```

`dictionary: a:0 b:1 ab:2`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2				
3				
4				
5				
6				
7				
8				
9				
10				

add: string + char at next position

Question 3

`text = ababababababababababababab…`

`dictionary: a:0 b:1 ab:2`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b		
3				
4				
5				
6				
7				
8				
9				
10				

Question 3

`text = ababababababababababababab...`

`dictionary: a:0 b:1 ab:2 ba:3`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3				
4				
5				
6				
7				
8				
9				
10				

add: `string + char at next position`

Question 3

`text = ababababababababababababab...`

`dictionary: a:0 b:1 ab:2 ba:3`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab		
4				
5				
6				
7				
8				
9				
10				

Question 3

text = **ababababababababababababab…**

dictionary: a:0 b:1 ab:2 ba:3 aba:4

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4				
5				
6				
7				
8				
9				
10				

add: string + char at next position

Question 3

`text = ababababababababababababababab…`

```
dictionary: a:0 b:1 ab:2 ba:3 aba:4
```

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba		
5				
6				
7				
8				
9				
10				

Question 3

text = **abababababababababababababab...**

dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5				
6				
7				
8				
9				
10				

add: string + char at next position

Question 3

text = **ababababababababababababababab...**

dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba		
6				
7				
8				
9				
10				

Question 3

`text = ababababababababababababab...`

dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6				
7				
8				
9				
10				

add: string + char at next position

Question 3

`text = ` **abababababababababababababab...**

`dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab		
7				
8				
9				
10				

Question 3

text = **ababababababababababababababab...**

dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7				
8				
9				
10				

add: string + char at next position

Question 3

`text = ababababababababababababababab...`

`dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7	13	abab		
8				
9				
10				

Question 3

`text = ababababababababababababab...`

`dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7 ababa:8`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7	13	abab	ababa	8
8				
9				
10				

add: `string + char at next position`

Question 3

text = **ababababababababababababab…**

dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7 ababa:8

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7	13	abab	ababa	8
8	17	ababa		
9				
10				

Question 3

`text = ababababababababababababab...`

`dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7 ababa:8 ababab:9`

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7	13	abab	ababa	8
8	17	ababa	ababab	9
9				
10				

add: `string + char at next position`

Question 3

text = **ababababababababababababababab...**

dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7 ababa:8 ababab:9

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7	13	abab	ababa	8
8	17	ababa	ababab	9
9	22	baba		
10				

Question 3

text = **abababababababababababababab...**

dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7 ababa:8 ababab:9 babab:10

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7	13	abab	ababa	8
8	17	ababa	ababab	9
9	22	baba	babab	10
10				

add: string + char at next position

Question 3

`text = ababababababababababababababab...`

```
dictionary: a:0 b:1 ab:2 ba:3 aba:4 abab:5 bab:6 baba:7 ababa:8 ababab:9
            babab:10 bababa:11
```

step	position in string	longest string in dictionary	add to dictionary	code
1	1	a	ab	2
2	2	b	ba	3
3	3	ab	aba	4
4	5	aba	abab	5
5	8	ba	bab	6
6	10	bab	baba	7
7	13	abab	ababa	8
8	17	ababa	ababab	9
9	22	baba	babab	10
10	26	babab	bababa	11

Question 3

Looking at how much of text is used up in each step we find the pattern: 1,1,2,3,2,3,4,5,4,5,6,7,6,7,8,9,8,9,10,11, …

step	position in string	longest string in dictionary	add to dictionary	code
1	1	**a**	**ab**	2
2	2	**b**	**ba**	3
3	3	**ab**	**aba**	4
4	5	**aba**	**abab**	5
5	8	**ba**	**bab**	6
6	10	**bab**	**baba**	7
7	13	**abab**	**ababa**	8
8	17	**ababa**	**ababab**	9
9	22	**baba**	**babab**	10
10	26	**babab**	**bababa**	11

Question 3

Looking at how much of text is used up in each step we find the pattern: 1,1,2,3,2,3,4,5,4,5,6,7,6,7,8,9,8,9,10,11,...

- therefore the number of steps required to encode the text we need to know how many of these terms are need for the summation to equal 100
- after a bit of adding up we find the first 19 terms sum to 100

End up with 21 code words so never have to increase the dictionary

- 2 code words to start with and then add 1 code word in each step
- 8 bits to encode the text so can fit 2^8 code words

19 steps means 19 (8 bit) code words in the compressed file

- input size $100 \cdot 8$ and output size $19 \cdot 8$
- compression ratio: 152/800=0.19 and space saved: $(1-0.19) \times 100 = 81\%$

Question 4

Find the distance between the strings

`s = agcgatc` and `t = ctacgaccg`

and derive an optimum alignment

Question 4

Find the distance between the strings

`s = agcgatc` and `t = ctacgaccg`

and derive an optimum alignment

Recall the recurrence relation is given by:

$$
\texttt{d(i,j)} = \begin{cases} \texttt{d(i-1,j-1)} & \text{if } \texttt{s[i-1]=t[j-1]} \\ \texttt{1 + min\{d(i,j-1), d(i-1,j), d(i-1,j-1)\}} & \text{otherwise} \end{cases}
$$

subject to `d(i,0)=i` and `d(0,j)=j` for all $\texttt{i} \le \texttt{n-1}$ and $\texttt{j} \le \texttt{m-1}$

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0											
1	a										
2	g										
3	c										
4	g										
5	a										
6	t										
7	c										

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1									
2	g	2									
3	c	3									
4	g	4									
5	a	5									
6	t	6									
7	c	7									

table is initialised by filling in row 0 and column 0 using the initial conditions of the recurrence relation

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1								
2	g	2									
3	c	3									
4	g	4									
5	a	5									
6	t	6									
7	c	7									

The entries are calculated one by one by application of the formula

- d(1,1)=1+0=1 (since s[0]≠t[0] and min(d(1,0),d(0,1),d(1,1))=0)

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1	2							
2	g	2									
3	c	3									
4	g	4									
5	a	5									
6	t	6									
7	c	7									

The entries are calculated one by one by application of the formula

- d(1,2)=1+1=2 (since s[0]≠t[1] and min(d(1,0),d(0,2),d(0,1))=1)

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	**2**	**3**	4	5	6	7	8	9
1	a	1	1	**2**	**2**						
2	g	2									
3	c	3									
4	g	4									
5	a	5									
6	t	6									
7	c	7									

The entries are calculated one by one by application of the formula

- `d(1,3)=2` (since `s[0]=t[2]` and `d(0,2)=2`)

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1	2	2	3	4	5	6	7	8
2	g	2	2	2	3	3	3	4	5	6	7
3	c	3	2	3	3	3	4	4	4	5	6
4	g	4	3								
5	a										
6	t										
7	c										

The entries are calculated one by one by application of the formula

- `d(4,1)=1+2=3` (since `s[3]≠t[0]` and `min(d(4,0),d(3,1),d(3,0))=2`)

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1	2	2	3	4	5	6	7	8
2	g	2	2	2	3	3	3	4	5	6	7
3	c	3	2	3	3	3	4	4	4	5	6
4	g	4	3	3	4	4	3	4	5	5	5
5	a	5	4	4	3	4	4	3	4	5	6
6	t	6	5	4	4	4	5	4	4	5	6
7	c	7	6	5	5	4	5	5	4	4	5

The entries are calculated one by one by application of the formula

- the final table: d(7,9)=5 so the string distance is 5

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1	2	2	3	4	5	6	7	8
2	g	2	2	2	3	3	3	4	5	6	7
3	c	3	2	3	3	3	4	4	4	5	6
4	g	4	3	3	4	4	3	4	5	5	5
5	a	5	4	4	3	4	4	3	4	5	6
6	t	6	5	4	4	4	5	4	4	5	6
7	c	7	6	5	5	4	5	5	4	4	5

traceback: **d(7,9)=1+d(7,8)**

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1	2	2	3	4	5	6	7	8
2	g	2	2	2	3	3	3	4	5	6	7
3	c	3	2	3	3	3	4	4	4	5	6
4	g	4	3	3	4	4	3	4	5	5	5
5	a	5	4	4	3	4	4	3	4	5	6
6	t	6	5	4	4	4	5	4	4	5	6
7	c	7	6	5	5	4	5	5	4	4	5

traceback: **d(7,8)=d(6,7)**

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1	2	2	3	4	5	6	7	8
2	g	2	2	2	3	3	3	4	5	6	7
3	c	3	2	3	3	3	4	4	4	5	6
4	g	4	3	3	4	4	3	4	5	5	5
5	a	5	4	4	3	4	4	3	4	5	6
6	t	6	5	4	4	4	5	4	4	5	6
7	c	7	6	5	5	4	5	5	4	4	5

traceback: d(6,7)=1+d(5,6)

Question 4

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	1	2	3	4	5	6	7	8	9
1	a	1	1	2	2	3	4	5	6	7	8
2	g	2	2	2	3	3	3	4	5	6	7
3	c	3	2	3	3	3	4	4	4	5	6
4	g	4	3	3	4	4	3	4	5	5	5
5	a	5	4	4	3	4	4	3	4	5	6
6	t	6	5	4	4	4	5	4	4	5	6
7	c	7	6	5	5	4	5	5	4	4	5

traceback:

Question 4

Corresponding alignment:

insertion insertion match deletion match match substitution match insertion

```
s:  c  t  a  -  c  g  a  c  c  g
t:  c  t  a  -  c  g  a  c  c  g
```

```
step: h ← h ← d ← v ← d ← d ← d ← d ← d ← h
```

Interpretation

- v (vertical) steps deletions
- h (horizontal) steps insertions
- d (diagonal) steps as substitutions or matches

Question 5 (2018–19 exam question)

A string `u` is a subsequence of a string `s` if `u` can be obtained from `s` by deleting zero or more characters. A string `u` is a common subsequence of `s` and `t` if it is a subsequence of both `s` and `t`

Design a dynamic programming algorithm to determine the length of the longest common subsequence (LCS) of two strings `s` and `t`

What are the time and space complexities of your algorithm?

Hint: base your algorithm on evaluating `l(i,j)`, the length of the LCS of the `i`th prefix of `s` and the `j`th prefix of `t`

Question 5 (2018–19 exam question)

Design a dynamic programming algorithm to determine the length of the longest common subsequence (LCS) of two strings **s** and **t**

$$\texttt{l(i,j)} = \begin{cases} & \text{if } \texttt{s[i-1]=t[j-1]} \\ & \text{otherwise} \end{cases}$$

We have found one element in common

Question 5 (2018–19 exam question)

Design a dynamic programming algorithm to determine the length of the longest common subsequence (LCS) of two strings `s` and `t`

$$
\texttt{l(i,j)} = \begin{cases} \texttt{l(i-1,j-1) + 1} & \text{if } \texttt{s[i-1]=t[j-1]} \\ & \text{otherwise} \end{cases}
$$

We have found one element in common

- LCS is 1+ LCS of what remains

Question 5 (2018–19 exam question)

Design a dynamic programming algorithm to determine the length of the longest common subsequence (LCS) of two strings `s` and `t`

$$
\texttt{l(i,j)} = \begin{cases} \texttt{l(i-1,j-1) + 1} & \text{if } \texttt{s[i-1]=t[j-1]} \\ & \\ & \text{otherwise} \end{cases}
$$

Elements do not match, so need to delete one of them

- no point deleting both as the other may match and want longest common subsequence

Question 5 (2018–19 exam question)

Design a dynamic programming algorithm to determine the length of the longest common subsequence (LCS) of two strings **s** and **t**

$$\texttt{l(i,j)} = \begin{cases} \texttt{l(i-1,j-1) + 1} & \text{if } \texttt{s[i-1]=t[j-1]} \\ \texttt{max\{l(i,j-1), l(i-1,j)\}} & \text{otherwise} \end{cases}$$

Elements do not match, so need to delete one of them

- no point deleting both as the other may match
- want longest common subsequence

Question 5 (2018–19 exam question)

Design a dynamic programming algorithm to determine the length of the longest common subsequence (LCS) of two strings **s** and **t**

$$
\texttt{l(i,j)} = \begin{cases} \texttt{l(i-1,j-1) + 1} & \text{if } \texttt{s[i-1]=t[j-1]} \\ \texttt{max\{l(i,j-1), l(i-1,j)\}} & \text{otherwise} \end{cases}
$$

Base case what is the longest common subsequence with any string and the empty string?

subject to **l(i,0)=0** and **l(0,j)=0** for all $0 \le i \le n-1$ and $0 \le j \le m-1$

Question 5 (2018–19 exam question)

Design a dynamic programming algorithm to determine the length of the longest common subsequence (LCS) of two strings `s` and `t`

$$
\texttt{l(i,j)} = \begin{cases} \texttt{l(i-1,j-1) + 1} & \text{if } \texttt{s[i-1]=t[j-1]} \\ \texttt{max\{l(i,j-1), l(i-1,j)\}} & \text{otherwise} \end{cases}
$$

subject to `l(i,0)=0` and `l(0,j)=0` for all `0≤i≤n-1` and `0≤j≤m-1`

The time and space complexities are both `O(mn)` for strings of lengths `m` and `n`, same reasoning as for the string distance algorithm

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0											
1	a										
2	g										
3	c										
4	g										
5	a										
6	t										
7	c										

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	0	0	0	0	0	0	0	0	0
1	a	0									
2	g	0									
3	c	0									
4	g	0									
5	a	0									
6	t	0									
7	c	0									

`l(i,0)=0` and `l(0,j)=0` for all $0 \le i \le n-1$ and $0 \le j \le m-1$

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	**0**	0	0	0	0	0	0	0	0
1	a	**0**	**0**								
2	g	0									
3	c	0									
4	g	0									
5	a	0									
6	t	0									
7	c	0									

`l(1,1)=0` (since `s[0]≠t[0]` and `max(l(1,0),l(0,1))=0`)

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	**a**	c	g	a	c	c	g
0		0	0	**0**	0	0	0	0	0	0	0
1	**a**	0	0	0	**1**						
2	g	0	0								
3	c	0	1								
4	g	0	1								
5	a	0	1								
6	t	0	1								
7	c	0	1								

`l(1,3)=1` (since `s[0]=t[2]` and `l(0,2)+1=1`)

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	**t**	a	c	g	a	c	c	g
0		0	0	0	0	0	0	0	0	0	0
1	a	0	0	0	**1**	1	1	1	1	1	1
2	g	0	0	**0**	1						
3	c	0	1								
4	g	0	1								
5	a	0	1								
6	**t**	0	**1**								
7	c	0	1								

`l(2,3)=1` (since `s[2]≠t[3]` and `max(l(2,1),l(1,3))=0`)

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	**t**	a	c	g	a	c	c	g
0		0	0	0	0	0	0	0	0	0	0
1	a	0	0	0	1	1	1	1	1	1	1
2	g	0	0	0	1	1	2	2	2	2	2
3	c	0	1	1							
4	g	0	1	1							
5	a	0	**1**	1							
6	**t**	0	1	**2**							
7	c	0	1								

l(6,2)=1 (since s[5]=t[1] and l(5,1)+1=2)

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	**g**
0		0	0	0	0	0	0	0	0	0	0
1	a	0	0	0	1	1	1	1	1	1	1
2	g	0	0	0	1	1	2	2	2	2	2
3	c	0	1	1	1	2	2	2	3	3	3
4	g	0	1	1	1	2	3	3	3	3	4
5	a	0	1	1	2	2	3	4	4	4	4
6	t	0	1	2	2	2	3	4	4	4	**4**
7	**c**	0	1	2	2	3	3	4	5	**5**	**5**

l(7,9)=0 (since s[6]≠t[8] and max(l(6,9),l(7,8))={4,5}=5)

Question 6

s\t		0	1	2	3	4	5	6	7	8	9
			c	t	a	c	g	a	c	c	g
0		0	0	0	0	0	0	0	0	0	0
1	a	0	0	0	1	1	1	1	1	1	1
2	g	0	0	0	1	1	2	2	2	2	2
3	c	0	1	1	1	2	2	2	3	3	3
4	g	0	1	1	1	2	3	3	3	3	4
5	a	0	1	1	2	2	3	4	4	4	4
6	t	0	1	2	2	2	3	4	4	4	4
7	c	0	1	2	2	3	3	4	5	5	5

Question 6

Substring: **a c g a c**

s:	-	-	**a**	g	**c**	**g**	**a**	-	t	**c**	-
t:	c	t	**a**	-	**c**	**g**	**a**	c	-	**c**	g

step: h ← h ← d ← v ← d ← d ← d ← h ← v ← d ← h

Interpretation

- v (vertical) steps `delete` current element from `s`
- h (horizontal) steps `delete` current element from `t`
- d (diagonal) steps `skip` (keep) current element in both `s` and `t`

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

– `b[j]` = the length of the longest border of `s[0..j-1]`

  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	a	g	c	a	g	a	c	a	g	c	a	c	g
j	0	1	2	3	4	5	6	7	8	9	10	11	12
b[j]													

Construct the above border table

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- b[j] = the length of the longest border of s[0..j-1]

  = max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }

string/pattern s	a	g	c	a	g	a	c	a	g	c	a	c	g
j	**0**	1	2	3	4	5	6	7	8	9	10	11	12
b[j]	0												

s[0..-1] = empty_string

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- `b[j]` = the length of the longest border of `s[0..j-1]`
  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	a	g	c	a	g	a	c	a	g	c	a	c	g
j	0	**1**	2	3	4	5	6	7	8	9	10	11	12
b[j]	0	**0**											

s[0..0] = a

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- `b[j]` = the length of the longest border of `s[0..j-1]`

  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	a	g	c	a	g	a	c	a	g	c	a	c	g
j	0	1	**2**	3	4	5	6	7	8	9	10	11	12
b[j]	0	0	**0**										

`s[0..1] = ag`

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- `b[j]` = the length of the longest border of `s[0..j-1]`
  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	a	g	c	a	g	a	c	a	g	c	a	c	g
j	0	1	2	**3**	4	5	6	7	8	9	10	11	12
b[j]	**0**	0	0	**0**									

s[0..2] = agc

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- `b[j]` = the length of the longest border of `s[0..j-1]`
  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	a	g	c	a	g	a	c	a	g	c	a	c	g
j	0	1	2	3	**4**	5	6	7	8	9	10	11	12
b[j]	0	0	0	0	**1**								

s[0..3] = agca and **s[0..1-1] = s[4-1..3] = a**

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- `b[j]` = the length of the longest border of `s[0..j-1]`

  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	**a**	**g**	**c**	**a**	**g**	a	c	a	g	c	a	c	g
j	0	1	2	3	4	**5**	6	7	8	9	10	11	12
b[j]	0	0	0	0	1	**2**							

`s[0..4]` = agcag and **`s[0..2-1]` = `s[5-2..4]` = ag**

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- `b[j]` = the length of the longest border of `s[0..j-1]`
  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	**a**	**g**	**c**	**a**	**g**	**a**	**c**	**a**	**g**	c	a	c	g
j	0	1	2	3	4	5	6	7	8	**9**	10	11	12
b[j]	0	0	0	0	1	2	1	0	1	**2**			

s[0..8] = agcagacag and **s[0..2-1] = s[9-2..8] = ag**

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

- `b[j]` = the length of the longest border of `s[0..j-1]`

  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	**a**	**g**	**c**	a	g	a	c	**a**	**g**	**c**	a	c	g
j	0	1	2	3	4	5	6	7	8	9	10	11	12
b[j]	0	0	0	0	1	2	1	0	1	2	3		

`s[0..9]` = agcagacagc and **`s[0..3-1]` = `s[9-3..9]` = agc**

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

– `b[j]` = the length of the longest border of `s[0..j-1]`
  = `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	**a**	**g**	**c**	**a**	g	a	c	a	**g**	**c**	**a**	c	g
j	0	1	2	3	4	5	6	7	8	9	10	**11**	12
b[j]	0	0	0	0	1	2	1	0	1	2	3	**4**	

s[0..10] = agcagacagca and **s[0..4-1] = s[10-4..10] = agca**

Question 7

A **border** of a string **s** is a substring that is both a prefix and a suffix and must be a strict subsequence

Border table b: array which has the same size as the string

– `b[j]` = the length of the longest border of `s[0..j-1]`

= `max { k | s[0..k-1] = s[j-k..j-1] ∧ k<j }`

string/pattern s	a	g	c	a	g	a	c	a	g	c	a	c	g
j	0	1	2	3	4	5	6	7	8	9	10	11	**12**
b[j]	0	0	0	0	1	2	1	0	1	2	3	4	**0**

`s[0..11] = agcagacagcac`

Question 8

Boyer-Moore Algorithm: the string is scanned right-to-left

- text character involved in a mismatch is used to decide next comparison
- involves pre-processing the string to record the position of the last occurrence of each character c in the alphabet
- therefore the alphabet must be fixed in advance of the search

Last occurrence array p

- `p[c]` position of character c in the string s
- i.e. equals `p[c]=max{ k | s[k]=c }` if such a k exists and -1 otherwise

Question 8

On finding a mismatch there is a jump step in the algorithm

- if the mismatch is between `s[j]` and `t[i]`
- ‘slide `s` along’ so that position `p[t[i]]` of `s` aligns with `t[i]`
  - i.e. align last position in `s` of character `t[i]` with position `i` of `t`
- if this moves `s` in the ‘wrong direction’, instead move `s` one position right
- if `t[i]` does not appear in string, ‘slide string’ passed `t[i]`
  - i.e. align position `-1` of `s` with position `i` of `t`

Question 8

a g c g **c** c t g a t a g c g a c a g t

a g c g **a**

- 1 comparison (c and a differ)
- c appears in string so move along so last c of string line up with c in text

Question 8

a g c g **c** c t g a t a g c g a c a g t

a g **c** g a

- 1 comparison (c and a differ)
- c appears in string so move along so last c of string line up with c in text

Question 8

a g c g c c **t** g a t a g c g a c a g t

a g c g **a**

- 1 comparison (c and a differ)
- 1 comparison (t and a differ)
- t does not appear in string so move past t

Question 8

a g c g c c **t** g a t a g c g a c a g t

a g c g a

- 1 comparison (c and a differ)
- 1 comparison (t and a differ)
- t does not appear in string so move past t

Question 8

a g c g c c t g a t a **g** c g a c a g t

a g c g **a**

- 1 comparison (c and a differ)
- 1 comparison (t and a differ)
- 1 comparison (g and a differ)
- g appears in string so move along so last g of string line up with g in text

Question 8

a g c g c c t g a t a **g** c g a c a g t

a g c **g** a

- 1 comparison (c and a differ)
- 1 comparison (t and a differ)
- 1 comparison (g and a differ)
- g appears in string so move along so last g of string line up with g in text

Question 8

a g c g c c t g a t a g **c** g a c a g t

a g c g **a**

- 1 comparison (c and a differ)
- 1 comparison (t and a differ)
- 1 comparison (g and a differ)
- 1 comparison (c and a differ)
- c appears in string so move along so last c of string line up with c in text

Question 8

a g c g c c t g a t a g **c** g a c a g t

a g **c** g a

- 1 comparison (c and a differ)
- 1 comparison (t and a differ)
- 1 comparison (g and a differ)
- 1 comparison (c and a differ)
- c appears in string so move along so last c of string line up with c in text

Question 8

a g c g c c t g a t **a g c g a** c a g t

a g c g a

- 1 comparison (c and a differ)
- 1 comparison (t and a differ)
- 1 comparison (g and a differ)
- 1 comparison (c and a differ)
- 5 comparisons and found string

- 9 comparisons in total