Institut für angewandte Mathematik *Adrianus Kleemans [07-111-693]*

Theoretische Aufgaben

Aufgabe 1

Zu zeigen:

$$r_i(j) = 2^{n-j} \tag{1}$$

Induktionsanfang $j = n$,

$$r_j(n) = 2^{n-n} = 1 \tag{2}$$

was äquivalent zur Gleichung 15.8 ist:

$$r_1(n) = r_2(n) = 1 \tag{3}$$

Induktionsschritt: $j = j - 1$
Zu zeigen:

$$r_i(j-1) = 2^{n-(j-1)} \tag{4}$$
$$r_i(j-1) = r_1(j) + r_2(j) \tag{5}$$
$$= 2^{n-j} + 2^{n-j} \tag{6}$$
$$= 2^{n-j+1} = 2^{n-(j-1)} \tag{7}$$

Zu zeigen:

$$\sum_{i=1}^{2} \cdot \sum_{j=1}^{2} \cdot r_i(j) = 2_{n+1} - 2 \tag{8}$$

Indexverschiebung:

$$\sum_{j=1}^{n} \cdot 2^{n-j} = 2^{n-i} + 2^{n-2} + ... + 2^{n-n} = \sum_{j=0}^{n-1} 2^j = \frac{1-2^n}{1-2} = 2^n - 1 \tag{9}$$

Wird zu:

$$\sum_{i=1}^{2} \sum_{j=1}^{n} \cdot 2^{n-j} = \sum_{i=1}^{2} (2^n - 1) = 2^{n+1} - 2 \tag{10}$$

Aufgabe 2

Da die Teile vor oder nach jedem Arbeitsschritt vom einen aufs andere Band wechseln können, gibt es sehr viele Möglichkeiten. Viele dieser Möglichkeiten enthalten aber gleiche Teile, und will man den schnellsten Weg finden, kann in einem bestimmten Abschnitt eine optimale Teillösung gefunden werden, der in der optimalen Lösung enthalten ist. Die Teillösung ist aber abhängig von anderen Stationen (da Sie in direktem Austausch zu diesen steht) und teilt sich dieses Teilproblem mit einem andern Teilproblem.

Aufgabe 3

Auf folgender Abbildung ist die funktionsweise eines LCS-Algorithmus zu sehen. Eine mögliche längste gemeinsame Teilsequenz wäre demnach `101010`.

	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	
	0	0	0	0	0	0	0	0	
1	0	0	1	1	1	1	1	1	1
0	0	1	1	2	2	2	2	2	0
0	0	1	1	2	2	2	2	2	
1	0	1	2	2	3	3	3	3	1
0	0	1	2	3	3	4	4	4	0
0	0	1	2	3	3	4	4	4	
1	0	1	2	3	4	4	5	5	1
0	0	1	2	3	4	5	5	5	0
1	0	1	2	3	4	5	6	6	

Aufgabe 4

Pseudocode für sichersten weg (bottom-up):

Listing 1: Aufgabe 4

```
1  //returns value of 'safest' way
2  int safestWay(int[][] area)
3  costs = new int[][] with same size as area
4  costs[0][0] = 0
5
6  for i = 0 to area.height - 1
7     for j = 1 to area.width - 1
8        if i == 0
9           costs[i][j] = costs[i][j-1] + area[i][j]
10       else if j == 0
11          costs[i][j] = costs[i-1][j] + area[i][j]
12       else
13          costs[i][j] = min(costs[i][j-1], costs[i-1][j]) + area[i][j]
14 return costs[costs.height-1][costs.width-1]
```

Laufzeit des Algorithmus ist die Anzahl Felder, welche vorkommen, also $\Theta(h \cdot b)$, wenn h die Höhe und b die Breite des Feldes repräsentiert.

T	8	9	11	14	19	26
8	11	12	22	16	25	33
20	20	16	25	18	32	33
27	30	24	40	21	26	35
28	30	26	38	36	34	37

Der Algorithmus ergibt die folgende Lösungstabelle. Startet man beim Endpunkt und vergleicht immer das obere und linke Feld und wählt dann das kleinere (dasjenige mit der tieferen Zahl) bis zum Anfang, so

hat man den optimalen Weg gefunden. Laufzeit ist die Anzahl Felder, welche durchlaufen werden müssen: $\Theta(h + b)$

Praktische Aufgaben

Aufgabe 1 + 2

Siehe Anhang `SeamCarving.java`.

Aufgabe 3

Siehe Bildergalerie.








PORTAL 2
APERTURE SCIENCE RETURNS
FOR THE PEOPLE WHO ARE STILL ALIVE

PORTAL 2