

Aufgabe 1

1 + 1 + 4 + 2 + 2 = 10T

Nicht optimaler Greedy

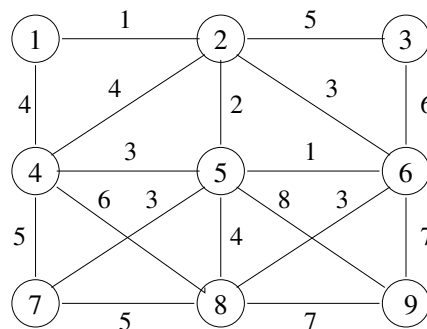
- Geben Sie (informell) einen Greedy-Algorithmus an, welcher zu einem gegebenen Rechteck eine Zerlegung in möglichst wenige Quadrate berechnet.
- Zeigen Sie anhand eines Beispiels, daß der Algorithmus nicht immer eine optimale Zerlegung findet.
- Geben Sie eine Folge von Rechtecken an, in deren Zerlegung nur Quadrate unterschiedlicher Seitenlängen vorkommen.
- Geben Sie für beliebige Rechtecke aus Teilaufgabe ?? eine geschlossene Form für die (absteigende) Folge der Seitenlängen der in der Zerlegung vorkommenden Quadrate an.
- Begründen Sie, daß der Greedy-Algorithmus für Rechtecke aus Teilaufgabe ?? optimal ist.

Aufgabe 2

4T

Algorithmus von Kruskal

Bestimmen Sie mit dem Algorithmus von Kruskal den minimal spannenden Baum des folgenden Graphen, und geben Sie die Kosten des minimal spannenden Baumes an. Geben Sie Zwischenschritte an!



Aufgabe 3

3T

Und weil's so schön war ...

Können Sie die Aufgabe 3 von Blatt 2 auch für 4 Schalter lösen?

Aufgabe 4

2+4+9=15R

Huffman-Codierung, Datentyp Baum

- Die folgende Deklaration definiert den Datentyp Binärbaum:

```
infix 5 :^:
data Baum a = Blatt a | Baum a :^: Baum a deriving (Eq,Read,Show)
```

Die Klasse Functor a definiert eine Funktion fmap, die analog zu map auf jedes Element des Datentyps a eine Funktion anwendet. Implementieren Sie eine Instanz für Baum a.

- Ausgehend vom Datentyp F2 wird mit

```
type Bin = [F2]
```

eine einfach Darstellung von Binärworten realisiert. Implementieren Sie die folgenden Funktionen:

```
char2Bin :: Char -> Bin    -- z.B. char2Bin 'a' ergibt [I,0,0,0,0,0,I,0]
bin2Char :: Bin -> Char
bin2String :: Bin -> String
string2Bin :: String -> Bin
```

Vergessen Sie dabei die führenden Nullen nicht! Ein Aufruf von `string2Bin.bin2String` soll die identische Abbildung ergeben.

c) Nun zu Huffman: Die Funktionen

```
huffmanBaum :: String → Baum Char
encodeHuffman :: String → Baum Char → String
decodeHuffman :: Eq a ⇒ String → Baum Char → String
```

sollen die übergebenen Zeichenketten Huffman-codieren bzw. den zugehörigen Huffmanbaum berechnen.

Hinweis: Die Module `Data.List` und `Data.Char` enthalten einige für die Teilaufgaben nützliche Funktionen:

`ord`, `chr`, `nub`, `sortBy`, `insertBy`, `compare`

Eine effizientere `encodeHuffman` Funktion können Sie z.B. durch Verwendung eines Arrays erreichen. Dann sollten Sie auch noch das Modul `Data.Array` einbinden.