BUS Exercise 7
Group 23

Thilo Metzlaff
406247

Mats Frenk
393702

Emma van Emelen
406008

July 1, 2020

Contents

Introduction

Do i welcome you again or do we know each other well enough by this point? Welp, whatever it is, today's exercise is about fitting big things into small spaces and vice versa. A topic we all understand. After all, we have access to the internet. As long as you don't try to square a circle, try to make a sphere fall over or solve $P = NP$, all is well. This sentence is a Lie. But please appreciate my 5AM tikz trees... they were more work than i expected[1]...

THE FORMAT

- Every file will be named similar to the sections in here, so
  `2.1-stack_exercise.c` is Exercise 2, section 1.

- Every Solution **WILL** be in this pdf, but not necessarily anything predefined by the exercise.

- AnysegmentxtbfWARNING: Humor may or may not be used. If you are allergic to humor, that sounds like a personal problem.

- **WARNING:** Backing up your data is important. Although linux doesn't have the necessary shame to remove itself, unlike windows[2], please do back up your data. And try to keep track of your periods... they seem to be notoriously hard to find

---

[1] i hope no period is missing this time...

[2] Happened to me... too often

1 Fitting things into spaces

1.1 Give it to me

The order of incoming requests, as i understood it, is:

$req_1 \rightarrow$ 1024 Byte

$req_2 \rightarrow$ 256 Byte

$req_3 \rightarrow$ 192 Byte

$req_4 \rightarrow$ 320 Byte

The following will be using this sequence. Now let S_n be the n-th useable memory segment, we get the following results, showing how much memory is free in each segment.

1.1.1 First Fit

	S_1	S_2	S_3
start	1024 Byte	512 Byte	256 Byte
req_1	0 Byte	512 Byte	256 Byte
req_2	0 Byte	256 Byte	256 Byte
req_3	0 Byte	256 Byte	256 Byte
req_4	0 Byte	64 Byte	256 Byte
	Cannot fit into S_2		

1.1.2 Rotating First Fit

I am unsure wether a Segment that is full would be skipped, but it makes sense that it would, so i assumed that it does so.

	S_1	S_2	S_3
start	1024 Byte	512 Byte	256 Byte
req_1	0 Byte	512 Byte	256 Byte
req_2	0 Byte	256 Byte	256 Byte
req_3	0 Byte	256 Byte	64 Byte
req_4	0 Byte	256 Byte	64 Byte
	Cannot fit into S_2		

1.1.3 Expectation - it all fits

This is so far the only algorithm that can allocate these requests properly.

	S_1	S_2	S_3
start	1024 Byte	512 Byte	256 Byte
req_1	0 Byte	512 Byte	256 Byte
req_2	0 Byte	512 Byte	0 Byte
req_3	0 Byte	320 Byte	0 Byte
req_4	0 Byte	0 Byte	0 Byte

1.1.4 Reality - this is bigger than expected

	S_1	S_2	S_3
start	1024 Byte	512 Byte	256 Byte
req_1	0 Byte	512 Byte	256 Byte
req_2	0 Byte	256 Byte	256 Byte
req_3	0 Byte	64 Byte	256 Byte
req_4	0 Byte	256 Byte	64 Byte
	Cannot fit anywhere		

1.2 It's not you, it's me

1.2.1 First Fit

	S_1	S_2	S_3
start	512 Byte	1024 Byte	256 Byte
req_3	320 Byte	1024 Byte	256 Byte
req_4	192 Byte	1024 Byte	256 Byte
req_1	0 Byte	0 Byte	256 Byte
req_2	0 Byte	0 Byte	0 Byte

1.2.2 She loves me, she loves me not

	S_1	S_2	S_3
start	512 Byte	1024 Byte	256 Byte
req_3	320 Byte	1024 Byte	256 Byte
req_1	320 Byte	0 Byte	256 Byte
req_2	320 Byte	0 Byte	0 Byte
req_4	0 Byte	0 Byte	0 Byte

1.2.3 makin it good

No change necessary, this algorithm is already the only one that can allocate the original configuration. I would argue that this can actually allocate most configurations correctly but may sometimes be less efficient, since we have to find the best segment. This would make first-fit most efficient in execution time, since it just allocates wherever it first finds the space, but only useful when we basically have large memory segments, since it is pretty dumb, like my personal memory management, which is basically a garbage dump, so at least i can say i'm sorta similar to a computer.

1.2.4 The Last of us

	S_1	S_2	S_3
start	1024 Byte	256 Byte	512 Byte
req_1	0 Byte	256 Byte	512 Byte
req_3	0 Byte	256 Byte	320 Byte
req_2	0 Byte	256 Byte	0 Byte
req_4	0 Byte	0 Byte	0 Byte

2 Addresses

2.1 Memory block size

To calculate the total size of available memory, we can just add the size of each segment like this:

$$\text{let: } S = \{999, 45, 410, 175, 191, 53, 65\} \tag{1}$$

$$\text{then: } S_{total} = \sum_{n=1}^{|S|} x_i \tag{2}$$

$$\Leftrightarrow S_{total} = 1938 \tag{3}$$
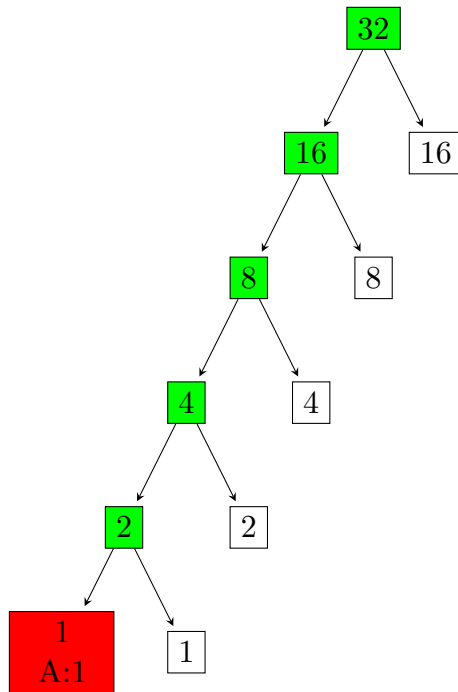
But size doesn't matter, right?

2.2 l o g i c

1. $1895 \rightarrow (2,0)$

2. $1575 \rightarrow (3,175)$

3. $1935 \rightarrow (2,40)$

4. $1050 \rightarrow (0,999)$

5. $4444 \rightarrow (5,44)$
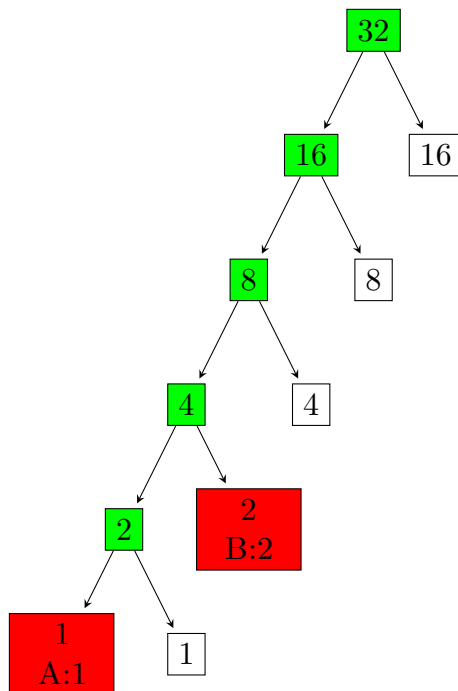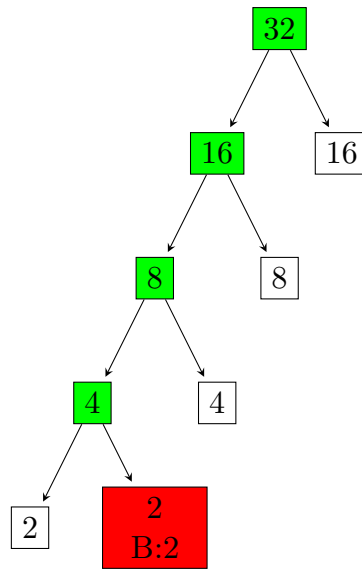
6. $1894 \rightarrow (1,44)$

3 Tell a friend

■ occupied
■ shared

3.0.1 Tree 1

```
                                    32
                              16          16
                           8        8
                        4        4
                     2        2
                  1        1
                  A:1
```

3.0.2 Tree 2

```
                                    32
                              16          16
                           8        8
                        4        4
                     2        2
                              B:2
                  1        1
                  A:1
```

3.0.3 TREE(3)

```
                              ┌────┐
                              │ 32 │
                              └────┘
                             ↙      ↘
                      ┌────┐         ┌────┐
                      │ 16 │         │ 16 │
                      └────┘         └────┘
                     ↙      ↘
              ┌───┐          ┌───┐
              │ 8 │          │ 8 │
              └───┘          └───┘
             ↙      ↘
        ┌───┐        ┌───┐
        │ 4 │        │ 4 │
        └───┘        └───┘
       ↙      ↘
   ┌───┐      ┌──────┐
   │ 2 │      │  2   │
   └───┘      │ B:2  │
              └──────┘
```

3.0.4 Tree 4

```
                              ┌────┐
                              │ 32 │
                              └────┘
                             ↙      ↘
                      ┌────┐         ┌──────┐
                      │ 16 │         │  16  │
                      └────┘         │ C:11 │
                     ↙      ↘        └──────┘
              ┌───┐          ┌───┐
              │ 8 │          │ 8 │
              └───┘          └───┘
             ↙      ↘
        ┌───┐        ┌───┐
        │ 4 │        │ 4 │
        └───┘        └───┘
       ↙      ↘
   ┌───┐      ┌──────┐
   │ 2 │      │  2   │
   └───┘      │ B:2  │
              └──────┘
```
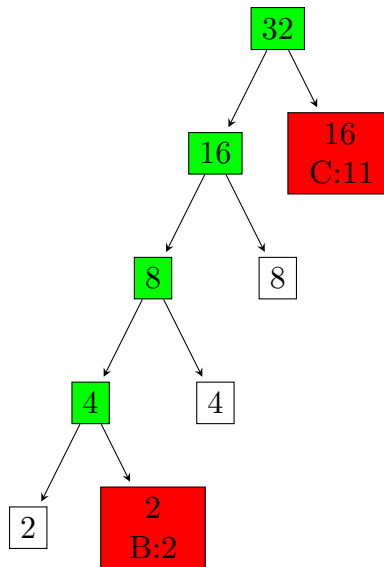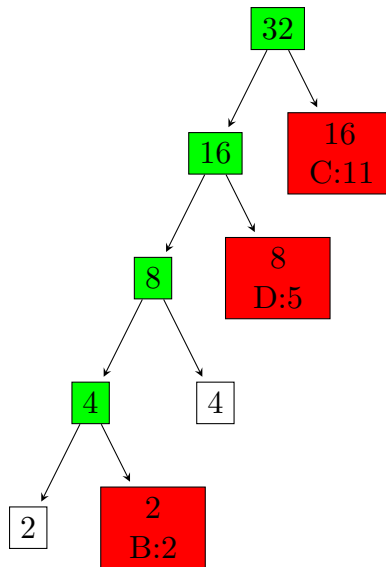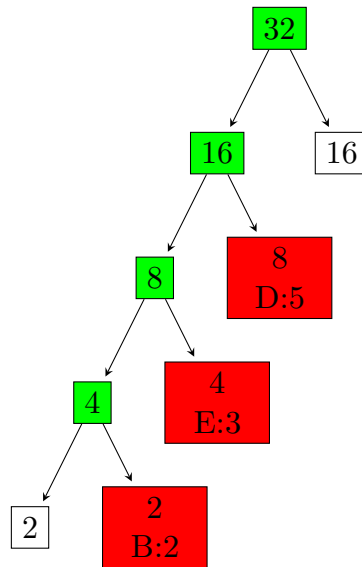
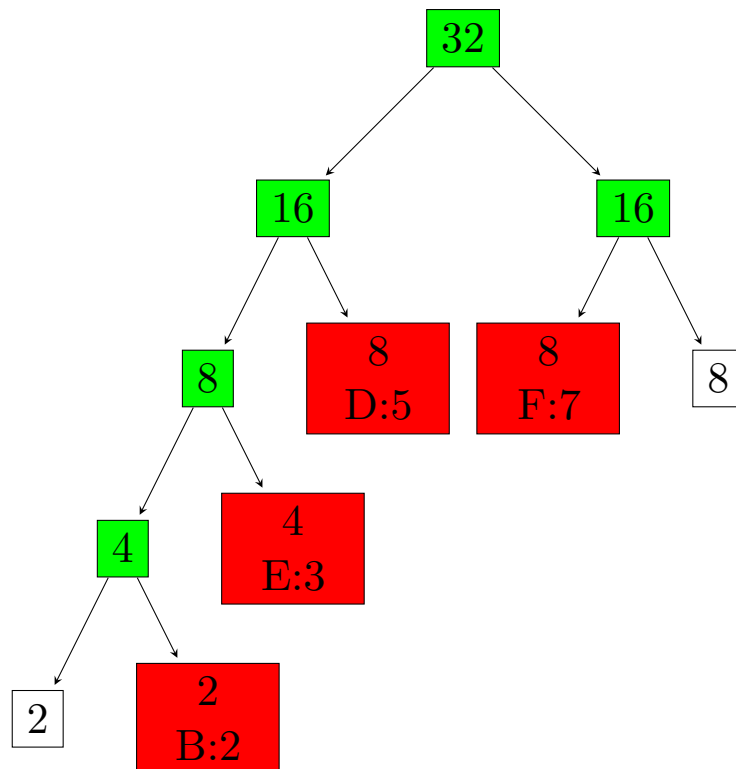### 3.0.5   Tree 5
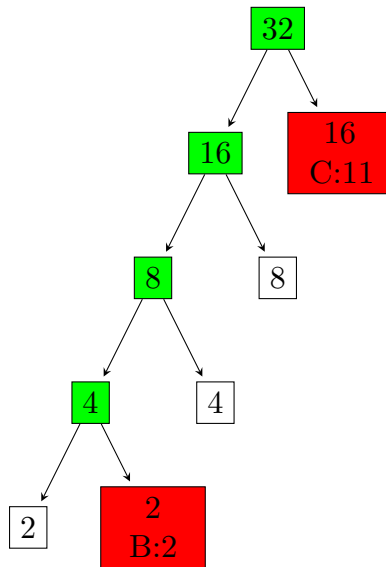
3.0.6   Tree 6

3.0.7   Tree 6

3.0.8   Tree 8

3.1   Fragmented memory

We can allocate at most 8 GB and we lose 5GB due to internal fragmentation.
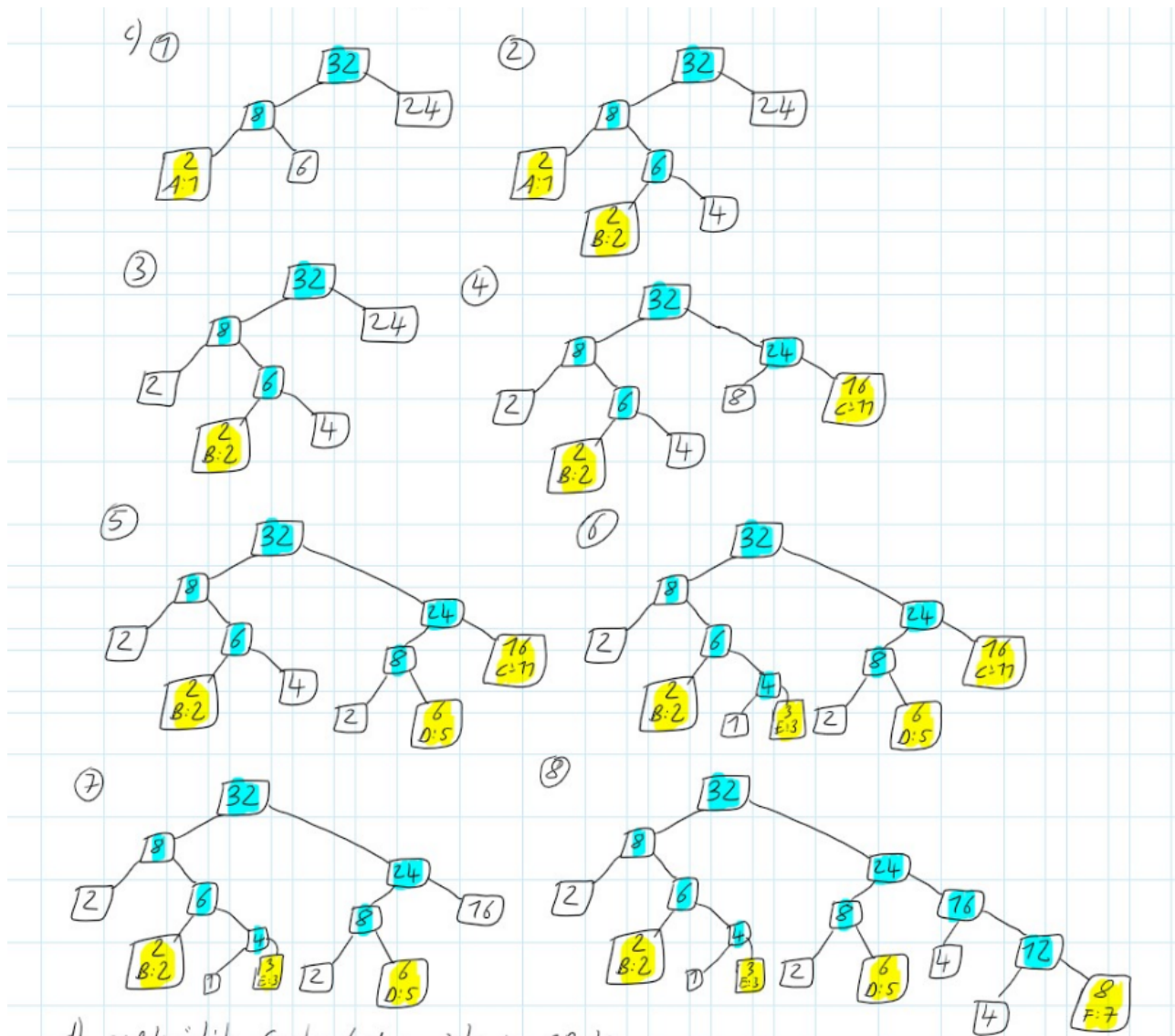
3.2 More Trees

Figure 1: since i could not be bothered to make more trees, have a screensot

3.3 More Fragmented memory

Largest possible memory allocation is 4GB. We lose 3GB of memory due to internal ffragmentation.

4 The End...

4.1 Memory Paging

4.1.1 This could be YOUR title

$$4 \text{ KiB} = 4096 \text{Byte} \tag{4}$$

$$\frac{2^{32}}{4096} = 1048576 \text{ Entries} \tag{5}$$

4.1.2 This should be MY title [3]

$$n^2 = 2^{20} \tag{6}$$

$$\Rightarrow n = 2^{10} \tag{7}$$

$$\Rightarrow n = 1024 \tag{8}$$

4.1.3 THIS IS OUR TITLE[4]

disadvantage: The longer time complexity due to address calculations. Every stage adds more overhead to calculate the address.

advantage: Not all pages are loaded to memory, which lowers memory complexity SIGNIFI-CANTLY.

---

[3]but i'm out of time and creativity, it's 5AM and i underestimated how long it would take me to tex this...

[4]*Soviet anthem starts playing*