Vienna University of Technology

184.725 High Performance Computing

TU Wien Informatics

Exercise 1

Authors:
Camilo Tello Fachin
12127084

Supervisor:
Prof. Dr. Jesper Larsson Träff

December 5, 2022

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Abstract

Here documented the results of exercise 1.

Contents

1 Exercise 1 - Closed Form Expressions **1**

1.1 $\sum_{i=0}^{d} k^i \quad \text{for } k > 0$ (Ex1.1) 1

1.2 $\sum_{i=1}^{d} ik^i \quad \text{for } k > 0$ (Ex1.4) 1

1.3 $\sum_{i=1}^{d} i2^{d-i}$ (Ex1.3) 1

1.4 $\sum_{i=1}^{d} i2^{i}$ (Ex1.2) 1

2 Exercise 2 - Graph Tree's with Canonical Numbering **2**

2.1 T_k^d with $k = 3$ and $d = 3$ 2

2.2 B_k^d with $k = 3$ and $d = 4$ 2

2.3 Complete unbalanced binary tree 2

3 Exercise 3 - Planar Graph H_d **3**

4 Exercise 4 - Gray Code Embedding in a Hypercube **4**

5 Exercise 5 - Inverse Gray Code's and Benchmarks **5**

6 Exercise 6 - not done.. **7**

7 Exercise 7 - not done.. **7**

1 Exercise 1 - Closed Form Expressions

1.1 $\sum_{i=0}^{d} k^i$ for $k > 0$ (Ex1.1)

$$\begin{aligned}
\sum_{i=0}^{d} k^i &= \sum_{i=0}^{d} k^i \\
\sum_{i=0}^{d} k^i - k\sum_{i=0}^{d} k^i &= \sum_{i=0}^{d} k^i - k\sum_{i=0}^{d} k^i \\
\sum_{i=0}^{d} k^i - k\sum_{i=0}^{d} k^i &= \sum_{i=0}^{d} k^i - \sum_{i=1}^{d+1} k^i \\
\sum_{i=0}^{d} k^i (1-k) &= 1 - k^{d+1} \\
\sum_{i=0}^{d} k^i &= \frac{k^{d+1}-1}{k-1}
\end{aligned} \tag{1}$$

1.2 $\sum_{i=1}^{d} ik^i$ for $k > 0$ (Ex1.4)

$$\begin{aligned}
\sum_{i=1}^{d} ik^i = \sum_{i=0}^{d} ik^i = \sum_{i=0}^{d} k\frac{\mathrm{d}}{\mathrm{d}k}k^i = k\frac{\mathrm{d}}{\mathrm{d}k}\overbrace{\sum_{i=0}^{d} k^i}^{\text{use (1)}} \\
\sum_{i=1}^{d} ik^i = k\frac{\mathrm{d}}{\mathrm{d}k}\frac{1-k^{d+1}}{1-k} = \frac{dk^{d+2}-(d+1)k^{d+1}+k}{(1-k)^2}
\end{aligned} \tag{2}$$

1.3 $\sum_{i=1}^{d} i2^{d-i}$ (Ex1.3)

$$\begin{aligned}
&\sum_{i=1}^{d} i2^{d-i}, \quad \text{use k instead of 2} \\
&\sum_{i=1}^{d} ik^{d-i} = \sum_{i=0}^{d} dk^{d-i} - \sum_{i=0}^{d}(d-i)k^{d-i} \\
&\qquad = d\underbrace{\sum_{j=0}^{d} k^j}_{\text{use (1)}} - \underbrace{\sum_{j=0}^{d} jk^j}_{\text{use (2)}} \quad \text{with } j := d-1 \\
&\qquad = \frac{d(k^{d+1}-1)}{k-1} - \frac{dk^{d+2}-(d+1)k^{d+1}+k}{(1-k)^2} \quad \text{set } k \text{ back to 2} \\
&\qquad = d2^{d+1} - d - d2^{d+2} + d2^{d+1} + 2^{d+1} - 2 \\
&\sum_{i=1}^{d} i2^{d-i} = 2^{d+1} - 2 - d
\end{aligned} \tag{3}$$

1.4 $\sum_{i=1}^{d} i2^i$ (Ex1.2)

$$\sum_{i=1}^{d} i2^i = d2^{d+2} - (d+1)2^{d+1} + 2 \quad \text{with use of (2)} \tag{4}$$

2 Exercise 2 - Graph Tree's with Canonical Numbering

Plots of the Graph Tree's with Canonical Numbering done with the Python library `networkx`.

2.1 T_k^d with $k = 3$ and $d = 3$

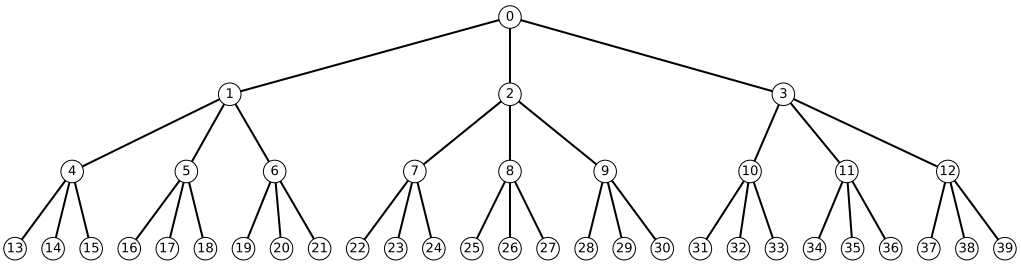

Figure 1: Fixed-degree k-ary heigh d tree T_k^d with $k = 3$ and $d = 3$

2.2 B_k^d with $k = 3$ and $d = 4$

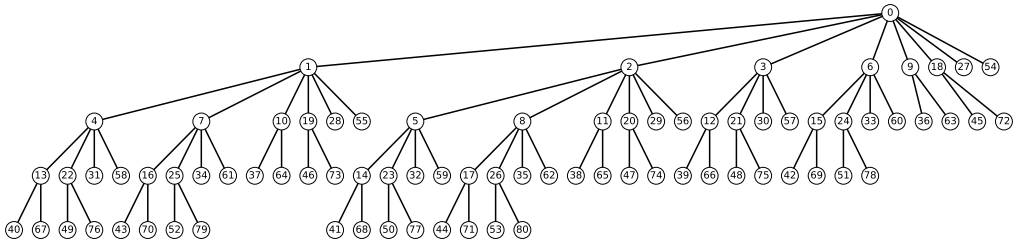

Figure 2: Complete heigh d k-nomial tree B_k^d with $k = 3$ and $d = 4$

2.3 Complete unbalanced binary tree

The tree drawn below is following the Pre-Order numbering scheme.

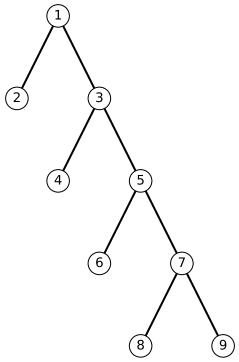

Figure 3: Complete (but unbalanced) binary tree of height $d = 4$ with $p = 9$ nodes.

3 Exercise 3 - Planar Graph H_d

For which d is the hypercube H_d a planar graph? In graph theory, a planar graph is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other. in fact this works for $d \leq 3$. It can also be shown with Wagner's theorem.

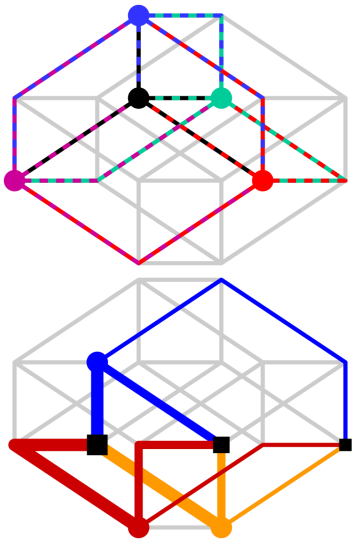

Figure 4: Proof without words that a hypercube graph is non-planar Wagner's theorems and finding either K_5 (top) or $K_{3,3}$ (bottom) subgraphs [1]

In fact for all hypercubes up to $d \leq 3$, neither K_5 nor $K_{3,3}$ can be found embedded in the hypercube. □

4 Exercise 4 - Gray Code Embedding in a Hypercube

The Gray code algorithms 2 and 3 from the HPC script are implemented in C and shown in the listing below. The executed binary indeed yields the console output `yarg and gray did not throw errors - Ex1.4 done :-)`.

C Language Listing for EX1.4

```c
uint gray(uint num){
    return num ^ (num >> 1);
}
uint yarg(uint num)s{
    uint mask = num;
    while (mask)
    {
        mask >>= 1;
        num ^= mask;
    }
    return num;
}
uint yarg32(uint num){
    num ^= num >> 16;
    num ^= num >> 8;
    num ^= num >> 4;
    num ^= num >> 2;
    num ^= num >> 1;
    return num;
}
int main(){
    int d = 20;
    uint old = gray(0);
    for (int j = 1; j < pow(2, d); j++)
    {
        uint ans = gray(j);
        uint diff = old ^ ans;
        int check = 0;
        for (int k = 0; k < d; k++)
        {
            if ( diff == (1 << k)){
                check++;
            }
        }
        if (check != 1){
            printf ("gray() error");
            break;
        }
        if (yarg(ans) != j){
            printf ("yarg() error");
            break;
        }
        old = ans;
    }
    printf ("yarg and gray did not throw errors - Ex1.4 done :-)!\n");
    return 0;
}
```

5 Exercise 5 - Inverse Gray Code's and Benchmarks

Presented below the results of the benchmark on the Student's machine. The specs of the machine are shown in the listing below the figure with the results.

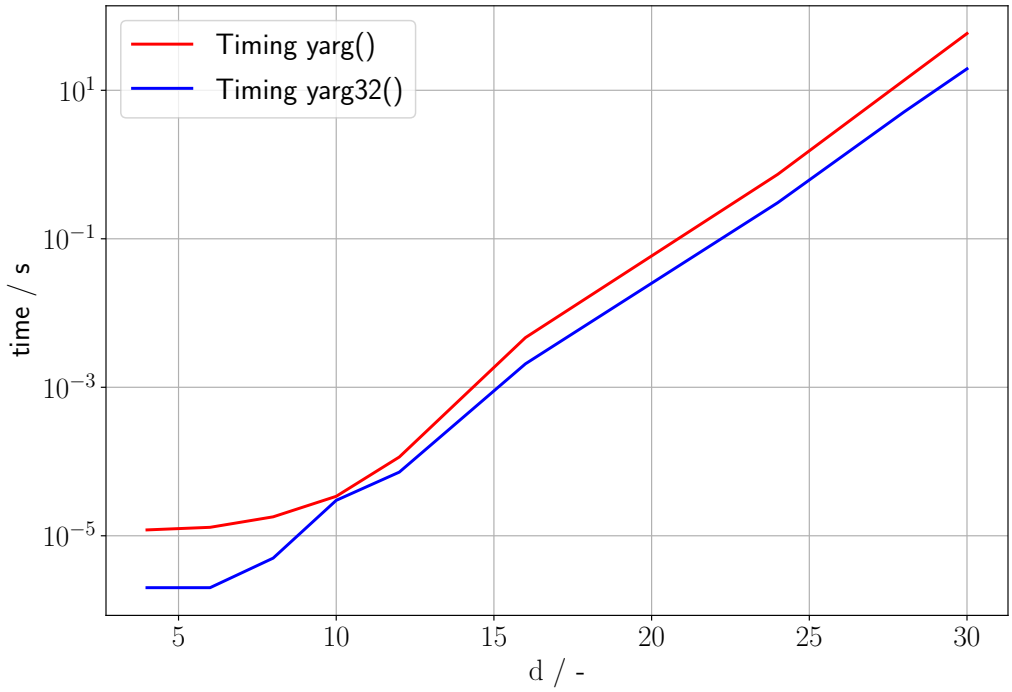

Figure 5: Benchmark both algorithms - `yarg32()` is faster than `yarg()` for all feasible values of d

Linux Terminal Output of `lscpu`

```
tellocam@DESKTOP-0AFJNHI:~/Projects/HPC/Exercise_1/Quell_Kodierung/CLANG$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          12
On-line CPU(s) list:             0-11
Thread(s) per core:              2
Core(s) per socket:              6
Socket(s):                       1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           158
Model name:                      Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
Stepping:                        10
CPU MHz:                         3696.001
BogoMIPS:                        7392.00
Hypervisor vendor:               Microsoft
Virtualization type:             full
L1d cache:                       192 KiB
L1i cache:                       192 KiB
L2 cache:                        1.5 MiB
L3 cache:                        12 MiB
```

In the listing below, both the algorithms as well as the experimental setup is shown. In line 13 - 20 for example, the `clock()` function is used which counts the number of clock ticks during the execution time. If one divides that by `CLOCKS_PER_SEC` which is constant, one obtains an acceptable execution timing of all yargcode calculations for the hypercube.

C Language Listing for EX1.5

```
1   typedef unsigned int uint;
2   uint gray(uint num){
3       // Same as in Exercise 1.4
4   }
5   uint yarg(uint num){
6       // Same as in Exercise 1.4
7   }
8
9   uint yarg32(uint num){
10      // Same as in Exercise 1.4
11  }
12
13  int main()
14  {
15      int d = 30;
16      clock_t start = clock();
17      for (int j = 1; j < pow(2, d); j++)
18      {
19          yarg(j);
20      }
21      clock_t res = clock() - start;
22      printf("yarg(): %f s\n", (double)res / CLOCKS_PER_SEC);
23
24      start = clock();
25      for (int j = 1; j < pow(2, d); j++)
26      {
27          yarg32(j);
28      }
29      res = clock() - start;
30      printf("yarg32(): %f s\n", (double)res / CLOCKS_PER_SEC);
31
32      return 0;
33  }
```

6 Exercise 6 - not done..

7 Exercise 7 - not done..

References

[1] K. Wagner. (1937). Wagner's Theorem - Über eine Eigenschaft der Ebenen Komplexe, [Online]. Available: https://en.wikipedia.org/wiki/Wagner%27s_theorem (visited on 04/12/2020).