Analysis of Algorithms WS 2022
Prof. Dr. P. Rossmanith
M. Gehnen, H. Lotze, D. Mock

RWTH AACHEN UNIVERSITY

Date: 31 October 2022

Exercise Sheet with solutions 01

Due date: 6 November 14:30

Task T1.1

Consider the following algorithm that computes the maximum element in an array of positive integers. We assume that all elements are pairwise different and that each permutation occurs with equal probability.

```
int maxElem(int *a, int N)
{
  int i,max;

  max = -1;              /*  1  */
  for (i=0; i<N; i++)    /*  2  */
    if (a[i] > max)      /*  3  */
      max = a[i];        /*  4  */
  return max;            /*  5  */
}
```

- How often are the lines 3 and 4 executed in the *worst case* (in the *best case*)?
- What is the probability that this worst case occurs?
- How often are the lines 3 and 4 executed in the *average case*?

Solution

- Line 3 is executed N times. Since `max` is negative at the beginning, it must be overwritten in line 4 at least once even in the best case, when the largest element is at the beginning. In the worst case, the array is sorted and increasing, the condition in line 3 is always true and line 4 is executed N times.
- The best case occurs if the first element is the maximum element. The probability for this event is $1/N$. The worst case happens with a probability of only $1/N!$, since among the $N!$ permutations of the array only one is the increasing sequence.
- Line 3 is executed N times. The estimation for line 4 is more complicated, since the probability that line 4 is executed for $a[i]$ depends on the previous $a[j]$, $j < i$, and is thus not independent for all i. We can tackle this problem with the linearity of the expected value.
  The probability that the kth element is larger than its $k-1$ predecessors is $1/k$, since among the $k!$ permutations of the first k elements there are exactly $(k-1)!$, s.t. the largest element is at position k. The expected number of executions of line 4 is therefore
  $$\sum_{i=0}^{N-1} \frac{1}{i+1} = \sum_{k=1}^{N} \frac{1}{k} = H_N = \ln N + \gamma + o(1),$$
  where H_N is the Nth harmonic number and $\gamma \approx 0.78$.

Task T1.2
Let w be a random word in $\{a, b\}^n$ chosen independently and with uniform probability. What is the expected number of iterations of the `while`-loop in the following algorithm? The function `is_palindrome` checks if the given word is a palindrome, i.e., if the word and its reverse are identical.

```
i = 2;
while (i <= n)
   if (is_palindrome(w[1],...,w[i]))
       return true;
   i++;
return false;
```

Solution
We w.l.o.g. assume w starts with a. If $w_2 = a$, too, which happens with a probability of $\frac{1}{2}$, then we already found a palindrome after one iteration. Otherwise we obtain the prefix ab. In the following iteration, we get a palindrome if $w_3 = a$, which again happens with probability $\frac{1}{2}$. Otherwise, we get the prefix abb. This observeration carried forward, we easily see that we find a palindrome once $w_i = a$ is read.
The expected number of iterations therefore is

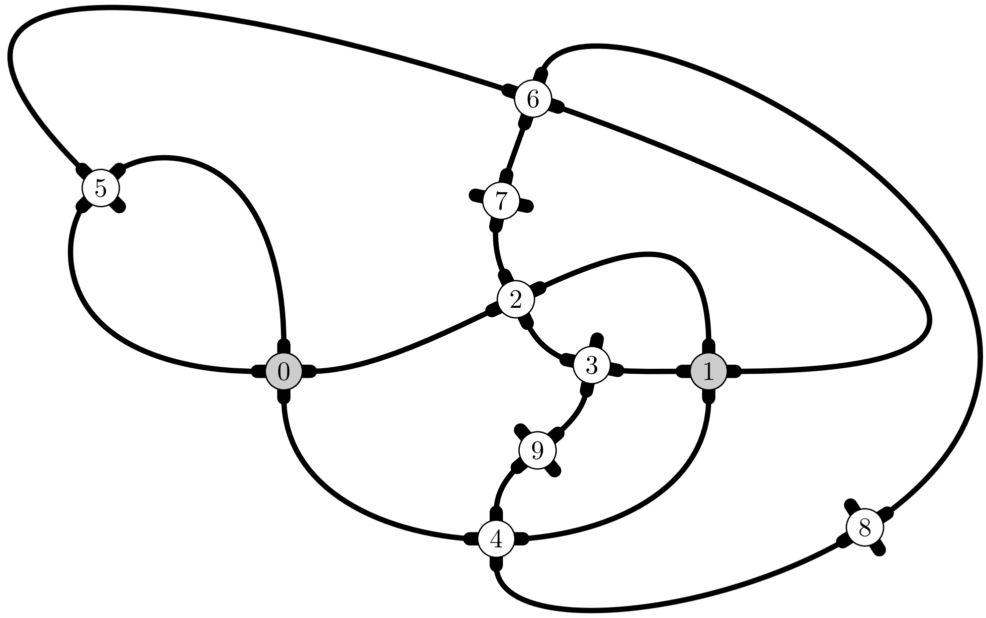
$$\sum_{k=2}^{n} \frac{1}{2^{k-1}}(k-1) + \frac{1}{2^{n-1}}(n-1),$$

where the last summand stems from the fact, that for $w = ab^{n-1}$ the **while**-loop is executed exactly $n - 1$ times. Using the formula for the geometric progression, we therefore obtain

$$2 - \frac{2(n+1)}{2^n} + \frac{1}{2^{n-1}}(n-1) = 2 - \frac{1}{2^{n-2}}.$$

Task H1.1 (10 credits)
RWTH Aachen University has an exclusive contract with the well-known *Uranus Corporation* on the delivery of canal and pump supplies. Unfortunately, the responsible person failed to notice that Uranus sells only the “one-fits-all” product KKuRPSE (Kanalkopplungsundregulierungspumpstationeinheit). Such a KKuRPSE is a cylinder with four connectors placed north, east, south, and west of the cylinder (see the figure).
Now, the Institute for Tunnel and Canal Construction (ITCC) is conducting some research: The researchers first place n KKuRPSEs on a big green. Then they start to connect KKuRPSEs as follows: They digg a new canal between two arbitrary connectors. The new canal cannot cross another existing canal, of course. They then place a new KKuRPSE somewhere into this new canal, such that exactly two connectors on opposite sides of the new KKuRPSE are now attached to the canal.
The question is: How often can this connection procedure be repeated depending on the number n of initial KKuRPSEs on the green?

Example
Assume the researchers start with two KKuRPSEs, here labeled 0 and 1. Then a couple of connection steps are executed, where each new KKuRPSE receives consecutive numbers until no more connections are possible.



Solution

First notice the following invariants:

- The number of free connectors at each time is $4n$, since each KKuRPSE uses two connectors, but at the same time introduces two new connectors.
- In each face of the underlying planar graph there is at least one free connector.

Let f be the number of faces, v be the number of vertices, e be the number of edges, and c be the number of connected components at an arbitrary round of the game. Since the graph is planar,

$$v + f = 1 + e + c$$

by a theorem of Euler, and since in each round two new edges are introduced,

$$v = n + \frac{e}{2}.$$

The game ends when in each face only one connector remains, i.e., when $f = 4n$. At this time, the graph is connected, i.e., $c = 1$, due to the outer face. Since in each round exactly two edges are introduced,

$$n + \frac{e}{2} + 4n = 1 + e + 1$$

yields

$$\frac{e}{2} = 5n - 2,$$

and each game lasts exactly $5n - 2$ rounds.

Task H1.2 (20 credits)

Two natural numbers $m \neq n$ are called *friendly*, if the sum of all factors of m equals n — or the other way round. A son and a father wrote the following programs that compute friendly numbers. What are their running times? Assume that the constant is replaced by N.

Son

```cpp
#include <iostream>
int e[150000];
int realdiv(int a) {
  int n=0;
  for(int i=1; i+i<=a; i++)
    if(a%i==0) n+=i;
  e[a] = n;
  return n;
}
main() {
  for(int i=0; i<150000; i++) {
    int a = realdiv(i);
    if(a >= i) continue;
    if(e[a]==i) {
      std::cout << i << " "
      << realdiv(i) << "\n";
    }
  }
}
```

Father

```c
#include <stdio.h>
#define N 1000000
int factorsum[N];
int main() {
  int i;
  for(i=1; i<N; i++) {
    int p=i;
    while(p<N) {
      factorsum[p] += i;
      p += i;
    }
  }
  for(i=1; i<N; i++) {
    int a = factorsum[i]-i;
    if(a<i && i==factorsum[a]-a)
      printf("%d %d\n", a, i);
  }
  return 0;
}
```

Solution

The son's program requires $O(N^2)$ steps. In the father's program, the first for-loop dominates the running time. In the following table, lines represent the outer for-loop, columns represent the inner while-loop, and each 1 represents an execution of the inner loop (for the ease of presentation we write N instead of $N-1$):

$$
\begin{array}{ccccccccccc}
\multicolumn{11}{c}{\overbrace{\hspace{12em}}^{N}} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\
 & 1 & & 1 & & 1 & & 1 & & \cdots & 1 \\
 & & 1 & & & 1 & & & 1 & \cdots & \\
 & & & 1 & & & & 1 & & \cdots & \\
 & & & & 1 & & & & & \cdots & 1 \\
\vdots & & & & & \ddots & & & & \cdots & \vdots \\
 & & & & & & & & & & 1
\end{array}
$$

We can rewrite this as follows

$$
\begin{array}{ccccccccccc}
\multicolumn{11}{c}{\overbrace{\hspace{20em}}^{N}} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\
1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & \cdots & 1/2 \\
1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & \cdots & 1/3 \\
1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & \cdots & 1/4 \\
1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & \cdots & 1/5 \\
\vdots & & & & & & \ddots & & & \cdots & \vdots \\
1/N & 1/N & 1/N & 1/N & 1/N & 1/N & 1/N & 1/N & 1/N & \cdots & 1/N
\end{array}
$$

It is not hard to see that the sum over all entries of both tables differ by at most N (due to missing values in the last colums). Since the sum of each column is H_N, it is easy to see that the sum over the lower table is $NH_N = O(N \log N)$.