Homework 1

Vanja Stojanović

March 2025

1 Time Complexity

We analyse within the RAM model, and assign a cost c_k to the k-th instruction.

1.1 Exercise - function `t1`

We assume a `for` loop will check the condition and execute its operation in one cycle. Let $T_{t1}(n)$ be the running-time complexity of the function `t1` with the input of size n. Then we have the following

$$T_{t1}(n) = c_1 + c_2 \sum_{i=1}^{n+1} t_i + c_3 \sum_{i=1}^{n} t_i + c_4$$

where $t_i = 1$ is the number of times the inner `for`. We can expand to

$$\begin{aligned} T_{t1}(n) &= c_1 + c_2(n+1) + c_3 n + c_4 \\ &= n(c_2 + c_3) + c_1 + c_2 + c_4 \end{aligned}$$

we conclude that $T_{t1}(n) = O(n)$.

1.2 Exercise - function `t2`

We assume a more strict RAM, where each part of the `for` loop counts as one instruction. Let $T_{t2}(n)$ be the running-time complexity of the function `t2` with the input of size n. Then we have the following

$$\begin{aligned} T_{t2}(n) = c_1 + &\underbrace{c_2 + c_3(n+1) + c_4 n}_{\texttt{for}_1} \\ &+ \underbrace{\sum_{i=1}^{n} (c_4 + c_5(\log_2(i) + 1) + c_6 \log_2(i) + \sum_{i}^{log_2(i)} T_{t1}(i)) + c_7}_{\texttt{for}_2} \end{aligned}$$

we conclude that $T_{t2}(n) = O(n^2 \log n)$, since the first loop will runn n times and the second loop will loop $log_2(i) \cdot i$ times (i gets to n in the last iteration).

1.3 Exercise - function `cantor`

Let T_{cantor} be the running-time complexity of the function `cantor`. Since the function is recursive we can use Master's theorem to analyse its time complexity. It states that

$$T(n) = \begin{cases} \Theta(n^k) & ;\ a < b^k \\ \Theta(n^k \log n) & ; a = a^k \\ \Theta(n^{\log_b a}) & ; a > b^k \end{cases}$$

where the time complexity is of the following form

$$T(n) = aT(\frac{n}{b}) + cn^k \tag{1}$$

We define a as the number of recursive calls, in this case $a = 2$, b as the factor by which the problem size is divided, in this case $b = 2$, and k as the exponent of the non-recursive work, which in this case is linear (or non-dependant of n) so $k = 0$. We substitute the values into equation 1.3 and derive the following

$$T_{cantor}(n) = 2T(\frac{n}{2}) + cn^0 \tag{2}$$

Since this is the condition where $a > b^k$, we conclude that $T_{cantor}(n) = \Theta(n^{\log_2 2}) = \Theta(n)$.

2 Generating Functions

2.1 Exercise 4

We seek to find the number of distinct nonnegative integer solutions to the following equation

$$x_1 + 2x_2 + 5x_3 + 10x_4 = 13 \tag{3}$$

We define a family of generating functions $G_n(x)$ where n represents the leading factor of the terms above as

$$G_n(x) = 1 + x^n + x^{2n} + x^{3n} + ...$$

Since for $|x| < 1$ the geometric series converges as follows

$$\sum_{n=0}^{\infty} ax^n = \frac{a}{1-x}$$

with our common factor we can reduce the generating functions to

$$G_n(x) = \frac{1}{1-x^n}$$

We then calculate the number of solutions by multiplying the functions $G_1(x)$, $G_2(x)$, $G_5(x)$, $G_{10}(x)$ as shown below

$$\begin{aligned}G_1(x)G_2(x)G_5(x)G_{10}(x) &= (1 + x + x^2 + x^3 + ...)\\ &\cdot\ (1 + x^2 + x^4 + x^6 + ...)\\ &\cdot\ (1 + x^5 + x^{10} + x^{15} + ...)\\ &\cdot\ (1 + x^10 + x^{20} + x^{30} + ...)\\ &= 1 + x + 2x^2 + 2x^3 + 3x^4 + 4x^5 + 5x^6 + 6x^7 + 7x^8\\ &+\ 8x^9 + 11x^{10} + 12x^{11} + 15x^{12} + 16x^{13} + ...\end{aligned}$$

We take the coefficient of the term x^{13} and conclude that there are 16 nonnegative integer solutions to equation 2.1. The next equation for which we seek the number of distinct nonnegative integer solutions is

$$x_1 + 2x_2 = 13 \tag{4}$$

We again multiply our generating functions, now only $G_1(x)$, $G_2(x)$

$$\begin{aligned}G_1(x)G_2(x) &= (1 + x + x^2 + x^3 + ...)\\ &\cdot\ (1 + x^2 + x^4 + x^6 + ...)\\ &= 1 + x + 2x^2 + 2x^3 + 3x^4 + 3x^5 + 4x^6 + 4x^7 + 5x^8\\ &+\ 5x^9 + 6x^{10} + 6x^{11} + 7x^{12} + 7x^{13} + ...\end{aligned}$$

We again take the coefficient of the term x^{13} and conclude that there are 7 nonnegative integer solutions to equation 2.1.

■

2.2 Exercise 5

A triangle number n is a number of the form

$$T_n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

We derive the generating function $G(x)$ by starting with the geometric series

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$$

we differentiate

$$\frac{1}{(1-x)^2} = \sum_{n=0}^{\infty} nx^{n-1} = \sum_{n=1}^{\infty} nx^{n-1} = \sum_{n=0}^{\infty}(n+1)x^n$$

and we differentiate again

$$\frac{2}{(1-x)^3} = \sum_{n=0}^{\infty} n(n+1)x^{n-1} = 2\sum_{n=0}^{\infty} T_n x^{n-1} = 2\sum_{n=1}^{\infty} T_n x^{n-1} = 2\sum_{n=0}^{\infty} T_{n+1} x^n$$

thus

$$G(x) = \sum_{n=0}^{\infty} T_{n+1} x^n = \frac{1}{(1-x)^3}$$

Then calculating the 100th triangular number equates to finding the coefficient of the term where the exponent is 100. We input $n = 99$

$$G(x) = x + 3x^2 + 6x^3 + ... + 5050x^{100} + ...$$

concluding that the 100th triangular number is 5050.

■

2.3 Exercise 6

We wish to show, that every positive integer can be written in exactly one way as the sum of distinct powers of 2. We can derive a generating function to show that we can either choose an exponent or no as follows

$$G(x) = (1 + x^{2^0})(1 + x^{2^1})(1 + x^{2^2})(1 + x^{2^3}) \cdots = \prod_{k=0}^{\infty} (1 + x^{2^i})$$

when expanding this product each factor $(1 + x^{2^i})$ contributes either 1 or x^{2^i}. Therefore, any monomial in the expansion is of the form

$$x^{\epsilon_0 \cdot 2^0 + \epsilon_1 \cdot 2^1 + \epsilon_2 \cdot 2^2 + \cdots}$$

with each $\epsilon_i \in \{0, 1\}$. The exponent is exactly the sum of a subset of distinct powers if 2. Notice that every nonnegative integer has a unique binary expansion. In other words, for every nonnegative integer n there exists a unique sequence $(\epsilon_0, \epsilon_1, \epsilon_2, ...)$ with $\epsilon_i \in \{0, 1\}$ such that

$$n = \epsilon_0 \cdot 2^0 + \epsilon_1 \cdot 2^1 + \epsilon_2 \cdot 2^2 + \cdots$$

Thus, in the generating function $G(x)$ the monomial x^n appears exactly once since there is only one way to form n as a sum of distinct powers of 2.

■

3 Turing Machines

3.1 Exercise 7

The Turing machine checks that there is a corresponding letter from the input alphabet on the other side of the given sequence, yielding a guarrantee for a palindrome. Once the tape is completely blank we are finished, or when there are no defined transitions then the sequence was not a palindrome.

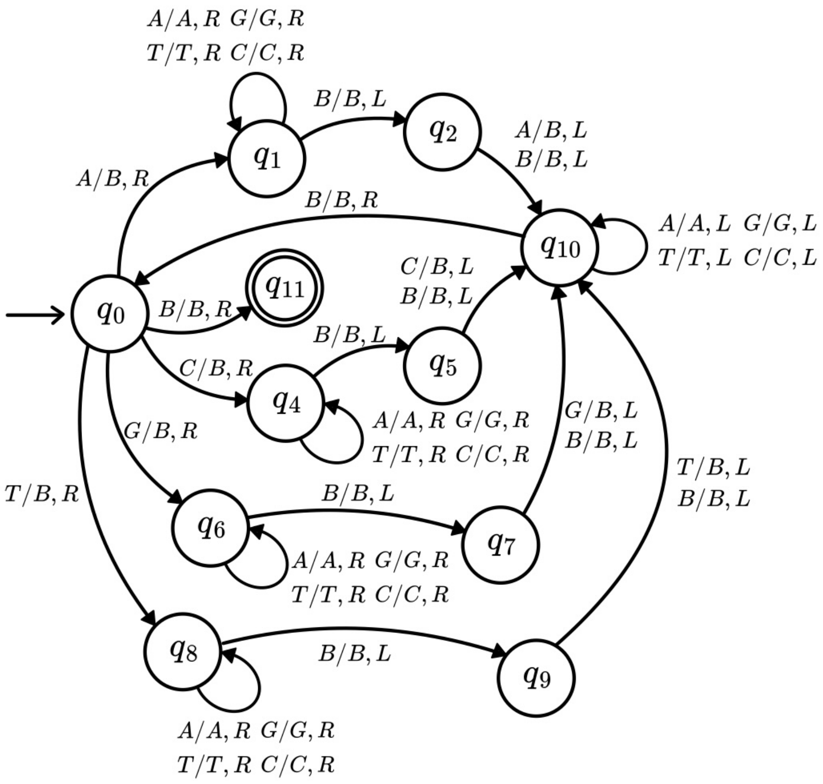

Figure 1: Turing machine for exercise 7

3.2 Exercise 8

The Turing machine first checks for unmatched parenthesis, if found it marks it with a # symbol, then it searches for a closing one, again marking it with a #. It repeats this for the entire sequence.

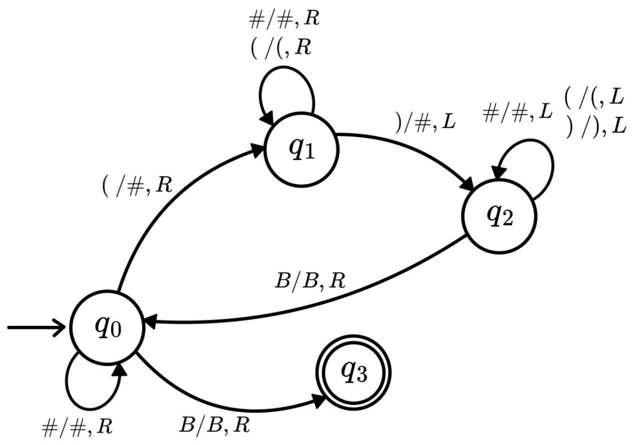

Figure 2: Turing machine for exercise 8

3.3 Exercise 9

The Turing machine marks the first unmatched symbol in the first half of w with X, then searches for an unmatched symbol in the second half of w and mark it with Y. It repeats this until the concatination gets checked and completely marked. The Turing machine can be seen below, on the next page.

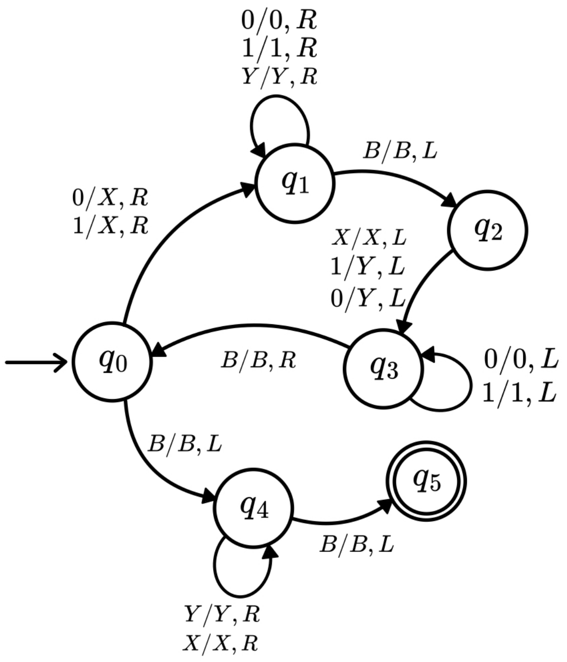

Figure 3: Turing machine for exercise 9