University of Limerick
Department of Mathematics and Statistics
Dr. S. Franz

MA4016 - Engineering Mathematics 6

Solutions to Final Examination

1. Suppose a factory consists of two machines A and B and a storage unit C. Let A_k and B_k be the input for those machines and C_k be the content of the storage C at time $k \in \mathbb{N}$. In each time step

   - Machine A produces from its input 20% final products and 80% scrap that is reused half as input for machine A and half as input for machine B in the next timestep.
   - Machine B produces from its input 60% final products and 40% scrap that is reused as input for machine B in the next timestep.
   - Storage unit C collects the final products from machines A and B and adds them to its previous content.

   (a) Find a system of three recurrence relations for the inputs A_k and B_k, and the content C_k.

$$\begin{aligned} A_{k+1} &= 0.4A_k \\ B_{k+1} &= 0.4A_k + 0.4B_k \\ C_{k+1} &= 0.2A_k + 0.6B_k + C_k \end{aligned}$$

   (b) Solve this recurrence if in its initial states machine A has an input of 1 and machine B has an input of 0, and the storage unit is empty

   ---

Start with A_k using $A_0 = 1$.

$$A_{k+1} = 0.4A_k \quad \Rightarrow \quad A_k = c_A(0.4)^k,\ A_0 = c_A = 1 \quad \Rightarrow \quad \boxed{A_k = (0.4)^k}$$

Substitute A_k in second recurrence to get nonhomogeneous recurrence relation

$$B_{k+1} = 0.4B_k + (0.4)^{k+1}.$$

Solve corresponding homogeneous and use ansatz for nonhomogeneous recurrence relation

$$\begin{aligned} B_k^{hom} &= c_B(0.4)^k \\ B_k^{part} &= p_0k(0.4)^k \quad \Rightarrow \quad p_0(k+1)(0.4)^{k+1} = 0.4p_0k(0.4)^k + (0.4)^{k+1} \quad \Leftrightarrow \quad p_0 = 1 \\ B_k &= B_k^{hom} + B_k^{part} = (c_B + k)(0.4)^k,\ B_0 = c_B = 0 \quad \Rightarrow \quad \boxed{B_k = k(0.4)^k} \end{aligned}$$

Finally we get for C_k the nonhomogeneous recurrence relation

$$C_{k+1} = C_k + (0.2 + 0.6k)(0.4)^k$$

University of Limerick
Department of Mathematics and Statistics
Dr. S. Franz

The corresponding homogeneous solution is $C_k = c_C$ and we use the ansatz $C_k^{part} = (p_0 + p_1k)(0.4)^k$ to get

$$\begin{aligned} (p_0 + p_1(k+1))(0.4)^{k+1} &= (0.2 + 0.6k)(0.4)^k + (p_0 + p_1k)(0.4)^k \quad \Leftrightarrow \\ 0.4(p_0 + p_1) &= 0.2 + p_0 \\ 0.4p_1 &= 0.6 + p_1 \end{aligned}$$

with the solution $p_0 = p_1 = -1$. Thus

$$C_k = c_C - (1+k)(0.4)^k,\ C_0 = c_C - 1 = 0 \quad \Rightarrow \quad \boxed{C_k = 1 - (1+k)(0.4)^k}.$$

Alternatively the discrete *Putzer* algorithm can be applied...

   (c) For which minimal value of k has the storage unit more than 0.9?

We have

$$\begin{aligned} C_0 &= 0 \\ C_1 &= 0.2 \\ C_2 &= 0.52 \\ C_3 &= 0.744 \\ C_4 &= 0.8720 \\ C_5 &= 0.93857 \end{aligned}$$

So starting with $k = 5$ the storage has more than 0.9.

2. (a) Describe *Dijkstra*'s shortest path algorithm for weighted graphs.

The algorithm is a labelling algorithm. We start with labelling vertex A with $[0, A]$ (there is a path of length 0 from A to A) and all other vertices with label $[\infty, \emptyset]$ (no path defined yet). Moreover the set of finished vertices S is defined empty.
Now we apply iteratively

- include in S the vertex V with smallest label
- update labels for all vertices not in S if shorter path is possible via V
- stop if all vertices are in S, otherwise continue iteration

   (b) Apply the algorithm to the undirected weighted graph represented by its adjacency matrix below, to find the shortest distances from vertex A to each of the other vertices.

	A	B	C	D	E	F	G
A	-	15	-	-	9	21	-
B	15	-	7	10	3	-	-
C	-	7	-	2	-	-	-
D	-	10	2	-	-	-	7
E	9	3	-	-	-	6	4
F	21	-	-	-	6	-	1
G	-	-	-	7	4	1	-

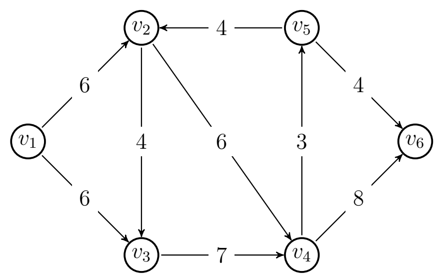
We build a table with all vertices and the set S. In each column the labels of the corresponding vertices are written and in the last column the elements of S. Underlined labels are terminal ones.

A	B	C	D	E	F	G	S
$[0,A]$	$[\infty,\emptyset]$	$[\infty,\emptyset]$	$[\infty,\emptyset]$	$[\infty,\emptyset]$	$[\infty,\emptyset]$	$[\infty,\emptyset]$	$\emptyset$
$\underline{[0,A]}$	$[15,A]$			$[9,A]$	$[21,A]$		$\{A\}$
	$[12,E]$			$\underline{[9,A]}$	$[15,E]$	$[13,E]$	$\{A,E\}$
	$\underline{[12,E]}$	$[19,B]$	$[22,B]$				$\{A,B,E\}$
			$[20,G]$		$[14,G]$	$\underline{[13,E]}$	$\{A,B,E,G\}$
					$\underline{[14,G]}$		$\{A,B,E,F,G\}$
		$\underline{[19,B]}$					$\{A,B,C,E,F,G\}$
			$\underline{[20,G]}$				$\{A,B,C,D,E,F,G\}$

3. (a) Describe the *Ford-Fulkerson* algorithm to find a maximal flow in a transport network.

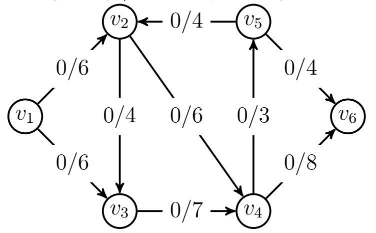
- We start with a flow satisfying the capacities (i.e. zero-flow)
- Search for a path P satisfying
  for each properly oriented edge (i,j) in P
  $$F_{i,j} < C_{i,j} \quad \text{flow less than capacity}$$
  for each improperly oriented edge (i,j) in P
  $$0 < F_{i,j} \quad \text{flow larger than minimal capacity}$$
  If no such path exists stop, flow is maximal.
- Define $\Delta = \min\{X_{i,j}\}$ with
  $$X_{i,j} = \begin{cases} C_{i,j} - F_{i,j} & \text{if } (i,j) \text{ is properly oriented edge in } P, \\ F_{i,j} & \text{if } (i,j) \text{ is improperly oriented edge in } P. \end{cases}$$
  Increase flow through P by setting $F = F^*$ with
  $$F^*_{i,j} = \begin{cases} F_{i,j} + \Delta & \text{if } (i,j) \text{ is properly oriented edge in } P, \\ F_{i,j} - \Delta & \text{if } (i,j) \text{ is improperly oriented edge in } P, \\ F_{i,j} & \text{if } (i,j) \text{ is not in } P. \end{cases}$$

(b) Apply the algorithm to the transport network given below.



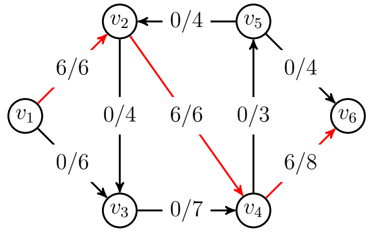
Graphical solution: f/c denotes flow through edge/ capacity of edge.

Step 0: Initial flow zero on each edge.



Step 1: Use augmenting path $v_1v_2v_4v_6$ (red in graph).

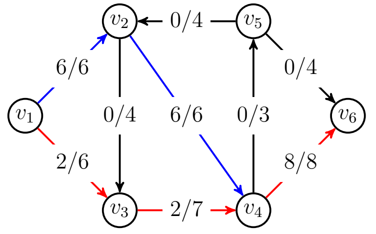
$\min\{6-0, 6-0, 8-0\} = 6 = \Delta$



Step 2: Use augmenting path $v_1v_3v_4v_6$ (red in graph).

$\min\{6-0, 7-0, 8-6\} = 2 = \Delta$

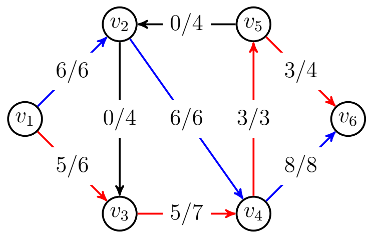
Note $\{v_1,v_2\}$ and $\{v_2,v_4\}$ have flows with maximum capacity (blue in graph).



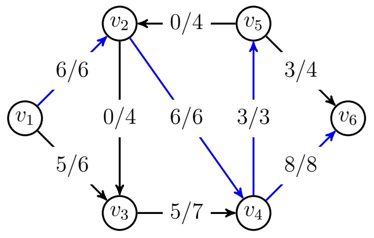
Step 3: Use augmenting path $v_1v_3v_4v_5v_6$ (red in graph).

$\min\{6-2, 7-2, 3-0, 4-0\} = 3 = \Delta$

Note $\{v_1, v_2\}$, $\{v_2, v_4\}$ and $\{v_4, v_6\}$ have flows with maximum capacity (blue in graph).



Step 4: Note $\{v_1, v_2\}$, $\{v_2, v_4\}$, $\{v_4, v_6\}$ and $\{v_4, v_5\}$ have flows with maximum capacity (blue in graph). No augmenting path from v_1 to v_6 can be found.



4. (a) Use the Master Theorem to find the asymptotic bounds of the following *divide-and-conquer* recurrences.

(i)

$$T(n) = 5T(n/2) + n^3$$

With $a = 5$, $b = 2$ $\log_b a = \log_2 5 \in (2, 3)$ case 3 of the Master Theorem applies with

$$g(n) = n^3 = \Omega(n^{\log_2 5+\varepsilon})$$

with $0 < \varepsilon < 3 - \log_2 5$. The condition $ag(n/b) < cg(n)$ with $c < 1$ holds because of

$$5(n/2)^3 = \frac{5}{8}n^3 < n^3$$

and we get

$$T(n) = \Theta(g(n)) = \Theta(n^3).$$

(ii)

$$T(n) = 9T(n/3) + n^2 \log n$$

With $a = 9$, $b = 3$, $\log_3 9 = 2$ holds case 2 of the Master Theorem

$$g(n) = n^2 \log n = \Theta(n^2 \log^k n)$$

with $k = 1$. Therefore

$$T(n) = \Theta(n^2 \log^{k+1} n) = \Theta(n^2 \log^2 n).$$

(b) Describe the Merge Sort algorithm for sorting a set of numbers into nondecreasing order.
Write down a recurrence relation for the number of comparisons required by the algorithm to sort a set of n numbers and estimate it using the Master Theorem.

Merge Sort is a recursive algorithm that has as input a list L of n unsorted elements and calls itself twice, first with the left half of L and then with the right half of L. The output of Merge Sort is the sorted input, so after those two calls we have two sorted sublists L_1 and L_2. As the final step Merge Sort merges those two sorted sublists to one sorted list—the output.
The number of comparisons required can be described by

$$T(n) = 2T(n/2) + g(n)$$

because two subproblems of half the size are used. The amount of comparisons needed in the merging step is measured by $g(n)$ and is linear in n, so

$$T(n) = 2T(n/2) + cn.$$

Case 2 of the Master Theorem yields $T(n) = \Theta(n \log n)$ because $a = b = 2$, $\log_2 2 = 1$ and $g(n) = cn = \Theta(n \log^k n)$ with $k = 0$.

5. Suppose we choose for the RSA-cryptosystem the primes $p = 7$ and $q = 11$, and the encryption exponent $e = 7$.

(a) Compute the decryption exponent d with the (extended) *Euclidean* algorithm.

With $p = 7$ and $q = 11$ we get $n = p \cdot q = 77$ and $\varphi(n) = (p-1)(q-1) = 60$.
For the decoding exponent holds $ed \equiv 1 \pmod{\varphi(n)}$.
We compute the gcd$(e, \varphi(n))$=gcd(7,60) by the *Euclidean* algorithm

$$\begin{aligned} 60 &= 7 \cdot 8 + 4 \\ 7 &= 4 \cdot 1 + 3 \\ 4 &= 3 \cdot 1 + 1 \end{aligned}$$

and we conclude $1 = 4-3 = 4-(7-4) = 2\cdot4-7 = 2(60-7\cdot8)-7 = 2\cdot60-17\cdot7$. Thus

$$-17 \cdot 7 \equiv 43 \cdot 7 \equiv 1 \pmod{60} \quad \Rightarrow \quad d = 43.$$

(b) Use the RSA-cryptosystem to encode the message $M = 20$.

The rule for encoding is $C = M^e \bmod n = 20^7 \bmod 77$. We can compute this directly with a calculator or use the fast modular exponentiation

$$\begin{aligned} 20^1 &\bmod 77 = 20 \\ 20^2 &\bmod 77 = 15 \\ 20^4 &\bmod 77 = 15^2 \bmod 77 = 71 \\ \Rightarrow \quad C &= 20 \cdot 15 \cdot 71 \bmod 77 = 48. \end{aligned}$$

(c) Decode the message $C = 40$ with the Chinese Remainder Theorem.

The rule for decoding is $D = C^d \bmod n = 40^{43} \bmod 77$. We use the equivalence

$$D \equiv 40^{43} \pmod{77} \quad \Leftrightarrow \quad \begin{cases} D \equiv 40^{43} \pmod 7 \\ D \equiv 40^{43} \pmod{11} \end{cases}$$

and simplify the right hand system (using *Fermat*'s Little Theorem) to

$$\begin{aligned} D &\equiv 40^{43} \equiv 5^{43} \equiv 5 \cdot (5^6)^7 \equiv 5 \pmod 7, \\ D &\equiv 40^{43} \equiv 7^{43} \equiv 7^3 \cdot (7^{10})^4 \equiv 7^3 \equiv 2 \pmod{11}. \end{aligned}$$

For the Chinese Remainder Theorem we need the inverse y_q of p modulo q and the inverse y_p of q mod p,

$$\begin{aligned} 11y_p \equiv 1 \pmod 7 \equiv 4y_p \pmod 7 \quad &\Rightarrow \quad y_p = 2 \\ 7y_q \equiv 1 \pmod 7 \quad \Rightarrow \quad y_q = 8 & \end{aligned}$$

and we obtain

$$D = (5 \cdot y_p \cdot 11 + 2 \cdot y_q \cdot 7) \bmod 77 \ = 222 \bmod 77 \ = 68.$$

6. Let A be a $n \times n$ matrix and B be a $n \times r$ matrix of real entries with $r < n$. Consider calculating $A^{n-1}B$ using repeated application of the standard algorithm for matrix multiplication.

(a) Describe how the task should be done efficiently, and determine how many multiplications and additions (of real numbers) are needed.

A multiplication of a $n \times n$ matrix with a $n \times n$ matrix costs n^3 scalar multiplications and the multiplication of a $n \times n$ matrix with a $n \times r$ matrix costs $n^2 r$ scalar multiplications. Thus for $r < n$ the matrix multiplication should be don in the following way

$$\begin{aligned} A^{n-1}B = A(A(A(\ldots A(AB)\ldots))) \quad &\Leftrightarrow \\ B_0 = B & \\ B_k = AB_{k-1} \quad k = 1, \ldots, n-1 & \end{aligned}$$

With $A^{n-1}B = B_{n-1}$. We have $n-1$ matrix multiplications with $n^2 r$ scalar multiplications and $n(n-1)r$ scalar additions each. Thus a total of $rn^2(n-1) = \Theta(rn^3)$ scalar multiplications and $rn(n-1)^2 = \Theta(rn^3)$ scalar additions is needed. Note that by computing the $(n-1)$-st power of A first we need a total of $\Theta(n^4 + n^2 r)$ operations.

(b) Illustrate the method with

$$A = \begin{pmatrix} 1 & -2 & 3 \\ 0 & -2 & 4 \\ 3 & -1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

We have

B				1
				0
				1
$A \cdot B$	1	-2	3	4
	0	-2	4	4
	3	-1	0	3
$A \cdot (A \cdot B)$	1	-2	3	5
	0	-2	4	4
	3	-1	0	8

Where we used $2 \cdot 3 \cdot 3 = 18$ scalar multiplications and $2 \cdot 3 \cdot 2 = 12$ scalar additions.

7. (a) Describe the main features of a *Turing* Machine.

A *Turing* Machine $T = (S, I, f, s_0)$ is a theoretical model of a computer. It has a control unit (head) that is in one of finitely many states $s_i \in S$ in each step. It has an infinite tape, divided into cells with symbols of an alphabet I on it. The head has read and write capabilities on the tape using the alphabet I. When it reads a symbol from the tape a rule $f : S \times I \to S \times I \times \{R, L\}$ depending on the symbol and the current state is triggered, changing the state of the head, deciding which symbol to write on the tape and in which direction the head moves along the tape. If there is no rule for the combination of state and symbol the machine stops.

(b) Construct a Modulo-5 Machine, i.e. a *Turing* Machine which takes as tape input a string of symbols representing an integer and produces as tape output the string of symbols representing the remainder after division by 5 of the integer.

Assume the integer n is represented by $n+1$ 1's on the tape. We want to erase 1's in sets of five, as long as there are at least six 1's left. We can do this iteratively, checking for the presence of six 1's, erasing them and reposition the head to the first 1' left to repeat this task. we will use the following rules

S\ I	1	B
s_0	s_1 1 R	
s_1	s_2 1 R	
s_2	s_3 1 R	
s_3	s_4 1 R	
s_4	s_5 1 R	
s_5	s_6 1 L	
s_6	s_7 B L	
s_7	s_8 B L	
s_8	s_9 B L	
s_9	s_{10} B L	
s_{10}	s_{11} B R	
s_{11}		s_{12} B R
s_{12}		s_{13} B R
s_{13}		s_{14} B R
s_{14}		s_0 B R

Using a different representation of the integer gives rise to a more compact machine. Here a quintary representation (basis 5) works fine. $8 = (13)_5$. Our alphabet is 0, 1, 2, 3, 4, B and we have to check for two non-blank symbols on the tape and erase the left one (idea: n mod 5=$(n-5k)$ mod 5 for integers k).

mod 5	0	1	2	3	4	B	
s_0	s_1 0 R	s_1 1 R	s_1 2 R	s_1 3 R	s_1 4 R		check for two symbols
s_1	s_2 0 L	s_2 1 L	s_2 2 L	s_2 3 L	s_2 4 L		
s_2	s_0 B R	s_0 B R	s_0 B R	s_0 B R	s_0 B R		delete left one and restart

(c) Illustrate the operation of the Modulo-5 Machine on the input string representing the number *eight*.

Second Turing machine applied to the encoding $8 = (13)_5$

tape	state	rule
...B $\boxed{1}$ 3 B ...	s_0	$(s_0, 1, s_1, 1,\text{R})$
...B 1 $\boxed{3}$ B ...	s_1	$(s_1, 3, s_2, 3,\text{L})$
...B $\boxed{1}$ 3 B ...	s_2	$(s_2, 1, s_0, B,\text{R})$
...B B $\boxed{3}$ B ...	s_0	$(s_0, 3, s_1, 3,\text{R})$
...B B 3 $\boxed{\text{B}}$ ...	s_1	no rule $\longrightarrow$ stop

First Turing machine applied to the encoding $8 = (111111111)_1$

tape	state	rule
...B $\boxed{1}$ 1 1 1 1 1 1 1 1 B ...	s_0	$(s_0, 1, s_1, 1,\text{R})$
...B 1 $\boxed{1}$ 1 1 1 1 1 1 1 B ...	s_1	$(s_1, 1, s_2, 1,\text{R})$
...B 1 1 $\boxed{1}$ 1 1 1 1 1 1 B ...	s_2	$(s_2, 1, s_3, 1,\text{R})$
...B 1 1 1 $\boxed{1}$ 1 1 1 1 1 B ...	s_3	$(s_3, 1, s_4, 1,\text{R})$
...B 1 1 1 1 $\boxed{1}$ 1 1 1 1 B ...	s_4	$(s_4, 1, s_5, 1,\text{R})$
...B 1 1 1 1 1 $\boxed{1}$ 1 1 1 B ...	s_5	$(s_5, 1, s_6, 1,\text{L})$
...B 1 1 1 1 $\boxed{1}$ 1 1 1 1 B ...	s_6	$(s_6, 1, s_7,\text{B,L})$
...B 1 1 1 $\boxed{1}$ B 1 1 1 1 B ...	s_7	$(s_7, 1, s_8,\text{B,L})$
...B 1 1 $\boxed{1}$ B B 1 1 1 1 B ...	s_8	$(s_8, 1, s_9,\text{B,L})$
...B 1 $\boxed{1}$ B B B 1 1 1 1 B ...	s_9	$(s_9, 1, s_{10},\text{B,L})$
...B $\boxed{1}$ B B B B 1 1 1 1 B ...	s_{10}	$(s_{10}, 1, s_{11},\text{B,R})$
...B B $\boxed{\text{B}}$ B B B 1 1 1 1 B ...	s_{11}	$(s_{11},\text{B}, s_{12},\text{B,R})$
...B B B $\boxed{\text{B}}$ B B 1 1 1 1 B ...	s_{12}	$(s_{12},\text{B}, s_{13},\text{B,R})$
...B B B B $\boxed{\text{B}}$ B 1 1 1 1 B ...	s_{13}	$(s_{13},\text{B}, s_{14},\text{B,R})$
...B B B B B $\boxed{\text{B}}$ 1 1 1 1 B ...	s_{14}	$(s_{14},\text{B}, s_0,\text{B,R})$
...B B B B B B $\boxed{1}$ 1 1 1 B ...	s_0	$(s_0, 1, s_1, 1,\text{R})$
...B B B B B B 1 $\boxed{1}$ 1 1 B ...	s_1	$(s_1, 1, s_2, 1,\text{R})$
...B B B B B B 1 1 $\boxed{1}$ 1 B ...	s_2	$(s_2, 1, s_3, 1,\text{R})$
...B B B B B B 1 1 1 $\boxed{1}$ B ...	s_3	$(s_3, 1, s_4, 1,\text{R})$
...B B B B B B 1 1 1 1 $\boxed{\text{B}}$ ...	s_4	no rule $\longrightarrow$ stop