Selected solutions for past entrance exams

dept. of Computer Science, IST,
The University of Tokyo

Przemek

January 2020

Contents

Chapter 1

Winter 2019

1.1 Problem 1 – Logic Circuit Design *(send help)*

Question 1 *Joint effort.* Thanks to Igor who first brute-forced the solution for $n = 3$. Takato did the same but using *k-maps* for each M_i.

Taku first draw a table for $n = 2$:

K_1	K_0	M_3	M_2	M_1	M_0
0	0	0	0	0	1
0	1	0	0	1	1
1	0	0	1	1	1
1	1	1	1	1	1

And then for $n = 3$:

K_2	K_1	K_0	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
0	0	0								1
0	0	1							1	1
0	1	0						1	1	1
0	1	1					1	1	1	1
1	0	0				1	1	1	1	1
1	0	1			1	1	1	1	1	1
1	1	0		1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

Notice the repetitive pattern above. Let's build solution for $n = 3$ using $n = 2$. A circuit for $n = 2$ is easy to build: let it have inputs $K_1^{(2)}, K_0^{(2)}$ and outputs $M_i^{(2)}$ for $i = 0, 1, 2, 3$. Let the solution for $n = 3$ have inputs $K_j^{(3)}$ for $j = 0, 1, 2$ and outputs $M_l^{(3)}$ for $l = 0, 1, \cdots 7$. Now set:

$$\begin{aligned}
K_0^{(2)} &:= K_0^{(3)} \\
K_1^{(2)} &:= K_1^{(3)} \\
M_i^{(3)} &:= M_i^{(2)} \vee K_2^{(3)} && \text{for } i = 0, 1, 2, 3 \\
M_i^{(3)} &:= M_i^{(2)} \wedge K_2^{(3)} && \text{for } i = 4, 5, 6, 7
\end{aligned}$$

This way we can generalize: to build solution for $n+1$, we can reuse solution for n with addition of 2^n more gates. More specifically, solution for n has 2^n outputs. We *or* first 'half' of output wires with K_{n+1}, and *and* the rest.

With this recursive definition, we can easily compute $S(n)$ *size* of the circuit with n inputs:

$$S(n) = S(n-1) + 2^n = \sum_{i=0}^{i=n} 2^i = 2^{n+1} - 1 = O(N)$$

Let $D(n)$ denote *depth* of the circuit with n inputs. Recall that $N = 2^n$ stands for number of outputs.

$$D(n) = D(n-1) + 1 = O(n) = O(\log N)$$

Don't forget to add a switch Z at the end of the final answer: simply *and* every M_i with negated Z. It adds 1 to depth and 2^n to size. Although, it doesn't change asymptotic size and depth, respectively.

Question 2 (send help) Limitation on depth makes it difficult.

1.2 Problem 2 – Bucket Sort

Setting of the problem: we got N element and K buckets.

Question 1

```
B[m].get(j)
```

Question 2 The input sequence might be in an **increasing** order and all elements might fall into one bucket. Thus, line 6 will be executed:

$$C = 0 + 1 + \cdots + (n-1) = \frac{n(n-1)}{2}$$

times. Note that it has to be an increasing order, because items are being inserted at the beginning of a bucket. If we wanna do evil, we must make every item traverse the whole bucket, until the end. It is possible when every inserted item is bigger than any element in the bucket, that is, input sequence is of increasing order.

Question 3 Let n_i denote size of i-th bucket. Line 6 is de facto an *insertion sort*, which means that i-th bucket will be sorted in $O(n^2)$. Total running time will be to:

$$C = \sum_{i=1}^{K} O(n_i^2)$$

Taking expectation:

$$\mathbb{E}(C) = \mathbb{E}[\sum_{i=1}^{K} O(n_i^2)] = \sum_{i=1}^{K} \mathbb{E}[O(n_i^2)] = \sum_{i=1}^{K} O(\mathbb{E}[n_i^2])$$

What is $[\mathbb{E}n_i^2]$? Note that, by definition $Var(X) = \mathbb{E}[X^2] - (\mathbb{E}X)^2$ for any random variable X. Let $X_{i,j}$ be an indicator random variable:

$$X_{i,j} = \begin{cases} 1 & \text{element } j \text{ went to bucket } i \\ 0 & \text{otherwise} \end{cases}$$

$$n_i = \sum_{j=1}^{N} X_{i,j}$$

Since we have K buckets and every of them is equally likely, probability of "going to bucket i" is $\frac{1}{K}$. Thus, expectation of n_i is:

$$\mathbb{E}(n_i) = \sum_{j=1}^{N} \mathbb{E}(X_{i,j}) = \frac{N}{K}$$

We can also notice, that, n_i is just a binomial random variable with expectation np and variance $np(1-p)$. Here a trial is mapping an item into bucket, and the success is placing it into i-th bucket. There are $n = N$ trials and probability of success if $\frac{1}{K}$.

$$\begin{aligned}\mathbb{E}[n_i^2] = Var[n_i] + (\mathbb{E}[n_i])^2 &= N\frac{1}{K}(1 - \frac{1}{K}) + (\frac{N}{K})^2 \\ &= \frac{N^2 + NK - N}{K^2}\end{aligned}$$

Finally:

$$\mathbb{E}[C] = O(\sum_{i=1}^{K} \mathbb{E}[n_i^2]) = O(\sum_{i=1}^{K} \frac{N^2 + NK - N}{K^2}) = O(\frac{N^2}{K} + N)$$

Question 3 The following contribute to total expected running time:

- finding a bucket for each element $O(N)$
- sorting each bucket, $O(\frac{N^2}{K} + N)$
- for each bucket, getting its content: $O(K\frac{N}{K}) = O(N)$

Total running time:

$$O(\frac{N^2}{K} + 3N)$$

When K is $O(N)$, then we get $O(N)$ expected running time. When K is $O(1)$, then the running time is $O(N^2)$.

Question 4

- BS is stable, QS isn't
- BS isn't in-place, QS is
- BS runs $O(N)$ expected, QS is $O(NlogN)$
- BS has upper limit on keys, QS hasn't

1.3 Problem 3 – Automata Theory

Question 1

```
--> o --a--> o --a--> o --b--> (o)
   / \      / \      / \       / \
   \ /      \ /      \ /       \ /
    E        E        E         E
```

Question 2 Write

$$L_2 = \{w \in \Sigma^* | v \preceq w \text{ for some } v \in L\}$$

where $v \preceq w$ indicates that v is subsequence of w. In other words, to in order to get a word from L_2, we first fix a word $v \in L$ and then intertwine its letters with some "garbage" from Σ^*.

Let's fix $v \in L$. Let $p_1, p_2, \cdots p_{|v|}$ be sequence of states that $\mathcal{A}$ visits when reading v. Label transitions between p_i's with next letters of v. Moreover, for each p_i, add a loop to itself labeled with Σ. Make p_1 the start state.

Question 3 First,

$$L' = \{w \in \Sigma^* | v \in L \text{ for every } v \in \Sigma^* \text{ such that } v \preceq w\}$$

is kinda quirky. We already supposed that $L \subseteq \Sigma^*$, so why do we need $v \in \Sigma^*$ above?

Let's start with constructing a NFA $\mathcal{N}$ which accepts L'. Let $\mathcal{N}$ be a copy of $\mathcal{A}$, where we additionally put Σ-labeled loops on every state. Intuitively, while traversing $\mathcal{A}$, in every state, we can have a "detour", i.e. accept some garbage.

Having constructed NFA $\mathcal{N}$, move to constructing equivalent DFA $\mathcal{D}$. It is feasible, and subset construction tells us how to it. Formally, define automaton $\mathcal{A}$ accepting L as $\mathcal{A} = (Q, \Sigma, \delta, q_o, F)$. NFA $\mathcal{N}$ such that $\mathcal{L}(\mathcal{N}) = L'$ is defined as:

$$\mathcal{N} = (Q, \Sigma, \delta_N, q_0, F)$$

where

$$\delta_N(q, a) = \{q\} \cup \{\delta(q, a)\}$$

is a "loop" on every state. Now, equivalent DFA $\mathcal{D}$ is obtained by subset construction:

$$\begin{aligned}
\mathcal{D} &= (Q_D, \Sigma, \delta_D, q_0, F_D) \\
Q_D &= \mathcal{P}(Q) \\
F_D &= \{X \in Q_D | X \cap F \neq \emptyset\} \\
\delta_D(X, a) &= \bigcup_{q \in X} \delta_N(q, a)
\end{aligned}$$

Question 4 Since every DFA has equivalent NFA and vice versa, I'll prove (Q3) for NFA $\mathcal{N}$. Two steps:

- $w \in L' \Rightarrow \mathcal{N}$ accepts w. Since $w \in L'$, then there must exits $v \in L$ such that $v \preceq w$. v is accepted both in $\mathcal{A}$ and $\mathcal{N}$, since $\mathcal{N}$ has exactly the same states as $\mathcal{A}$. Then, w must be accepted by $\mathcal{N}$ by following the loops on letters of w which do not contribute to v.
- $\mathcal{N}$ accepts $w \Rightarrow w \in L'$. To get v such that $v \preceq w$, simulate $\mathcal{N}$ on w, skip the loops. Obviously $v \in L$, because states of $\mathcal{N}$ and $\mathcal{A}$ are the same and v and w will end up in the same final state of $\mathcal{N}$.

1.4 Problem 4 – Linear Algebra

Question 1 Condition number $\kappa(A)$ of A is:

$$\kappa(A) = ||A||\,||A^{-1}||$$

where $||\cdot||$ denotes a norm of a matrix:

$$||A|| = \max_{x\in\mathbb{R}^{\mathbb{N}}} \frac{||Ax||}{||x||} = \max_{x:||x||=1} ||Ax||$$

Question 2 Not sure if OK...
We know $Ax = b$ and let $r = b - A\hat{x}$:

$$\begin{aligned}(A+E)\hat{x} &= b\\ A\hat{x} + E\hat{x} &= b\\ E\hat{x} &= b - A\hat{x}\\ E\hat{x} &= r\end{aligned}$$

But how do we know if E is rank 1?

Question 3

$$\begin{aligned}(A+\delta A)(x+\delta x) &= b\\ Ax + A(\delta x) + \delta A(x+\delta x) &= b\end{aligned}$$

Substituting $Ax = b$:

$$\begin{aligned}A(\delta x) + \delta A(x+\delta x) &= 0\\ -A(\delta x) &= \delta A(x+\delta x)\\ -\delta x &= A^{-1}(\delta A)(x+\delta x)\end{aligned}$$

because matrix A is not singular, so inverse A^{-1} exist. Now, take the norm both sides. We are assuming that $A + \sigma A$ is not singular, so it has inverse and thus a norm:

$$\begin{aligned}||\delta x|| &= ||A^{-1}\,[\delta A(x+\delta x)]||\\ &\le ||A^{-1}||\,||\delta A(x+\delta x)||\\ &\le ||A^{-1}||\,||\delta A||\,||x+\delta x||\end{aligned}$$

$$\begin{aligned}\frac{||\delta x||}{||x+\delta x||} &\le ||A^{-1}||\,||\delta A||\\ &= ||A^{-1}||\,\frac{||A^1||}{||A^1||}||\delta A||\\ &= \kappa(A)\frac{||\delta A||}{||A||}\end{aligned}$$

Question 4 When an n dimensional real matrix C is singular, there is a non-zero, real vector y such that $Cy = 0$. *In particular*, we can choose a unit vector $y' = \frac{y}{||y||}$.

Let $x \in nullspace(B)$ be a unit vector i.e. $Bx = 0$ and $||x|| = 1$.

$$\begin{aligned} ||A - B|| &= ||A - B||\,||x|| \\ &\geq ||(A - B)x|| \\ &= ||Ax - Bx|| \\ &= ||Ax|| \end{aligned}$$

On the other hand:

$$\begin{aligned} 1 = ||x|| = ||A^{-1}\,(Ax)|| &\leq ||A^{-1}||\,||Ax|| \\ ||Ax|| &\geq \frac{1}{||A^{-1}||} \end{aligned}$$

More on condition numbers:
`https://blogs.mathworks.com/cleve/2017/07/17/what-is-the-condition-number-of-a-matrix/`.

Chapter 2

Winter 2018

2.1 Problem 1 – Linear Algebra

Question 1 Let x be an eigenvector with corresponding eigenvalue λ: $Ax = \lambda x$. Let $\mu \in \mathbb{R}$.

$$(A + \mu I)x = Ax + \mu x = \lambda x + \mu x = (\lambda + \mu)x$$

So x is an eigenvector of $(A + \mu I)$ with corresponding eigenvalue of $\lambda + \mu$.

Question 2 Let $v \in \mathbb{R}^{\mathbb{N}}$ be a unit vector which maximizes $v^T Av$. Because A is real and symmetric, then it can be diagonalized $A = Q^T \Lambda Q$, where Q is orthogonal matrix, which columns form orthogonal basis. In following equation, let's substitute $y = Qv$:

$$\begin{aligned} v^T Av &= v^T Q^T \Lambda Q v = y^T \Lambda y \\ &= \sum_{i=1}^{N} \lambda_i y_i^2 \\ &\leq \sum_{i=1}^{N} \lambda_{max}(A)\, y_i^2 = \lambda_{max}(A) \sum_{i=1}^{N} y_i^2 \\ &= \lambda_{max}(A)\, y^T y = \lambda_{max}(A)\, v^T Q^T Q v \\ &= \lambda_{max}(A) \end{aligned}$$

On the other hand, let x_{max} be eigenvector corresponding to $\lambda_{max}(A)$. Because v maximizes $v^T Av$, then:

$$v^T Av \geq x_{max}^T (Ax_{max}) = x_{max}^T x_{max}\, \lambda_{max}(A) = \lambda_{max}(A)$$

Here we got:

$$\lambda_{max}(A) \leq v^T Av \leq \lambda_{max}(A)$$

For vector v which maximizes $v^T Av$. Hence,

$$\lambda_{max}(A) = \max\{v^T Av | v \in \mathbb{R}^N, v^T v = 1\}$$

and the maximum is reached for eigenvector x_{max}.

Question 2 Let $v \in \mathbb{R}^N$. From question (2) we know that $v^T Av \leq \lambda_{max}(A) v^T v$ for arbitrary v: look at second to the last line of Q2 calculations. Now if we know that fact, we get the inequality trivially:

$$v^T(\lambda_{max}(A)I - A)v = \lambda_{max}(A)v^T v - v^T Av \geq 0$$

Question 3 Since A, B are symmetric and real, then $A = Q^T \Lambda_A Q$ and $B = P^T \Lambda_B P$ for some orthogonal Q and P. From what we obtained in (Q2): let w be a unit vector which maximizes $w^T(A + B)w$. Then:

$$\begin{aligned} \Lambda_{max}(A + B) &= w^T(A + B)w \\ &= w^T Aw + w^T Bw \\ &= w^T Q^T \Lambda_A Q w + w^T P^T \Lambda_B P w = (*) \end{aligned}$$

Substitute $y = Qw$ and $z = Pw$. Of course $y^T y = z^T z = 1$, because $y^T y = w^T Q^T Q w = w^T w = 1$ (the same for z). Combining this and our knowledge form previous questions:

$$\begin{aligned}(*) &= y^T \Lambda_A y + z^T \Lambda_B z \\ &\leq \lambda_{max}(A)\, y^T y + \lambda_{max}(B)\, z^T z \\ &= \lambda_{max}(A) + \lambda_{max}(B)\end{aligned}$$

2.2 Problem 2 – Automata theory

Question 1 *This one was also solved in* ***Automata Theory, Languages and computation 3rd ed****, 2.2.4, Example 2.4*

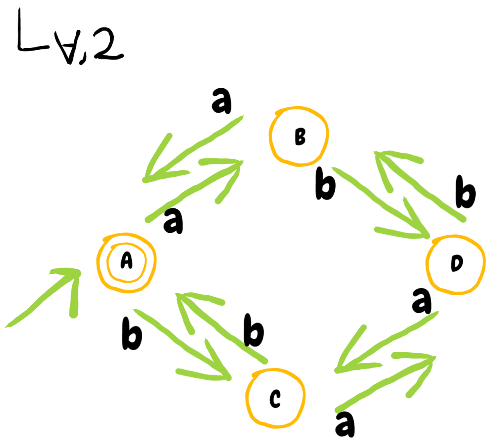

Figure 2.1: Question 1.

Explanation:

- A – "number of a's is **even**, number of b's is **even**"
- B – "number of a's is **odd**, number of b's is **even**"
- C – "number of a's is **even**, number of b's is **odd**"
- D – "number of a's is **odd**, number of b's is **odd**"

Question 2 Begin with an ϵ-NFA as depiced in Fig. 2.2. It "guesses" which letter appears even number of times. To make it ϵ-free, we either follow him: `https://youtu.be/sq-dLKAd6bo?t=1714` or consider the followig: starting from start state, how far, i.e. which states can we reach on letter a? There are 3 such states. We simply draw an edge to those states. Do the same for b, c. The final answer is on Fig. 2.3

Question 3 *Prove that, for every $n \geq 1$, every* ***deterministic*** *finite state automaton that accepts $L_{\exists,n}$ has at least 2^n states.*

First, I'll show that there is a DFA with 2^n states. We can encode states of DFA $L_{\exists,n}$ as a binary string $B = (b_n, b_{n-1}, \cdots, b_2, b_1)$ of length n, where $b_i = 1$ iff numbers of a_i's is odd in so-far read sequence, else 0. A state is accepting if its binary representation has at least one zero. There are 2^n such strings.

Now, I'll show that there is no DFA with less than 2^n states. If the DFA had fewer than 2^n states, then there would be some state q such that the DFA can be in state q after reading two different sequences of length n, say $a = a_1 a_2 \cdots a_n$ and $b = b_1 b_2 \cdots b_n$. If the sequences were of *different parity*, i.e. one of the sequences had a letter which appears even number of times, and the other sequence hadn't, then q would be both accepting and nonaccepting state. If the sequences were of the same parity, then we could append some sequence $c_1 \cdots c_m \in \Sigma^*$ to a and b such that $a_1 a_2 \cdots a_n c_1 \cdots c_m$ and $b_1 b_2 \cdots b_n c_1 \cdots c_m$ have different parity. Consider state p that the DFA enters after reading $c_1 \cdots c_m$. Then p must be both accepting and nonaccepting, since either of $a_1 a_2 \cdots a_n c_1 \cdots c_m$ and $b_1 b_2 \cdots b_n c_1 \cdots c_m$ is accepted and the other isn't.

This problem could be also one-lined using *Myhill-Nerode theorem.* The only one difficulty here is to define 2^n equivalent classes. Try fiddling with binary strings.

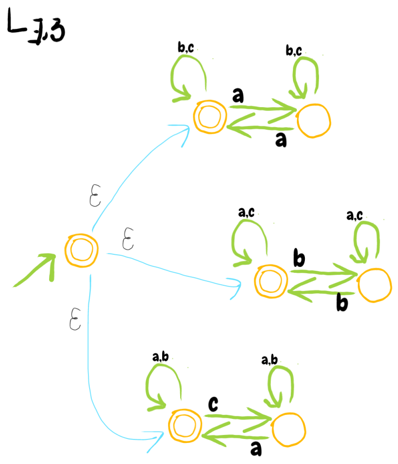

Figure 2.2: Question 2. ϵ-NFA.

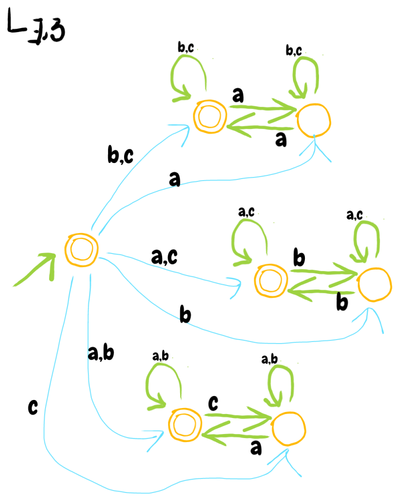

Figure 2.3: Question 2. ϵ-free NFA. We followed former ϵ-transitions and added new edges (in blue). Thanks Taku!

Question 4 *Prove that, for every $n \geq 1$, every* ***non-deterministic*** *finite state automaton (without ϵ-transitions) that accepts $L_{\forall,n}$ has at least 2^n states.* Begin with constructing DFA A accepting $L_{\forall,n}$ the same way as in Q3 – except that A has only one accepting state, a string of all 0's. Obviously this DFA is also a NFA. The equivalent NFA simply cannot *guess* parity of some letters: it has to remember information of parity of each letter. As shown in Q3, every letter requires 2 bits, so minimum of 2^n states are required.

s e n d h e l p

2.3 Problem 3 – Union-Find

See chapter 21 *of Cormen (3rd edition) for more.*

We have set of 2^N elements. $MERGE(A, B)$ changes A's root's pointer so that it points root of B.

Question 1 $2^N - 1$ merge operations are required to merge all the subsets.

Question 2 Minimum tree height: 1 – one root with $2^N - 1$ leafs. How to: Fix root r and perform $MERGE(v, r)$ for every other element $v \neq r$.

Question 3 Maximum tree height: $2^N - 1$. How to: start with a tree consisting of one element. While merging the tree and a node, make the tree point the node. In pseudo-code:

```
for i in range(1, 2**N):
    merge(v[i-1], v[i])
```

Question 4 New MERGE(A, B): change pointer of the root node of subset with smaller height so that it points to root node of the other subset. This technique is called *merge (union) by rank* (rank - upper bound on the height of the node). How to get a tree with maximum height? The observation is, merging trees having the same height will result in a taller tree. If A and B have height of h, then merged tree is of height $h + 1$.

Suppose we had a maximum-height tree with 2^N nodes. It must have been obtained by merging of two maximum-height trees with 2^{N-1} nodes.

$$H(2^N) = 1 + H(2^{N-1}) = N$$

Question 4 During execution of FIND, we make every node on the find-path point directly to the root. This technique is called *path compression.*

In details, first we find a path from a node its root. Then, we go through the path again and change pointers of nodes on the path. Original FIND was linear in length of the path. Now we scan the path twice, which still yields linear time.

2.4 Problem 4 – Synchronization

Question 1 Its 1 and 2. On a sequential execution we get 2. We can get 1 when one process loads a value before another process stores it.

Question 2 Test-and-set writes 1 to memory and returns previously stored value.

```
int TestAndSet(int *a) {
    int b;
    b = *a;
    *a = 1;
    return b;
}
```

Question 3 Swap:

```
void Swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Answer:

```
int key = 1;
while (key == 1)
    Swap(&key, &lock);
x = x + 1;
lock = 0;
```

Chapter 3

Winter 2017

3.1 Problem 1 – Linear Algebra

Question 1

$$||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \leq \sum_{i=1}^{n} \sqrt{x_i^2} = \sum_{i=1}^{n} |x_i| = ||x||_1$$

On the other hand, let $y \in \mathbb{R}^n$ be a unit vector:

$$\begin{aligned} ||x||_1 = |x \cdot y| \leq ||x||_2 ||y||_2 \\ &= ||x||_2 \sqrt{1 + 1 + \cdots + 1} \\ &= \sqrt{n} ||x||_2 \end{aligned}$$

The first inequality comes from Cauchy-Schwarz inequality: $|xy| \leq ||x||_2 ||y||_2$.

Question 2.1 Matrix A defines some transformation and $||A||_p$ measures how original vector will be stretched by the transformation. If $||A||_p < 1$ then the transformation *shrinks* the vector. By applying the transformation multiple times, we will shrink the vector more and more. In particular, applying it infinitely many times will shrink the vector to zero. Formally:

$$\begin{aligned} 0 &\leq \lim_{k\to\infty} ||A^k x_0||_p \\ &\leq \lim_{k\to\infty} ||A^k||_p ||x_0||_p \\ &= ||x_0||_p \lim_{k\to\infty} ||A^k||_p \\ &= ||x_0||_p \, 0 \\ &= 0 \end{aligned}$$

First equality holds because norm is always non-negative, second is a property of a norm. Moreover, limit $\lim_{k\to\infty} ||A^k||_p = 0$, because:

$$\lim_{k\to\infty} ||A^k||_p \leq \lim_{k\to\infty} (||A||_p)^k = 0$$

NB: $||AB||_p \leq ||A||_p ||B||_p$ for any matrices $A, B \in \mathbb{R}^{n\times n}$.

Question 2.2 Since A is symmetric, then it can be diagonalized $A = Q^T \Lambda Q$, where Q is orthogonal matrix (i.e. $Q^T Q = I$). Note that

$$\max_x \frac{||Ax||_p}{||x||_p} = \max_{x:||x||_p=1} ||Ax||_p$$

Let x be a unit vector which maximizes the above. We have:

$$\begin{aligned} ||Ax||_2 &= (Ax)^T (Ax) \\ &= x^T Q^T \Lambda Q Q^T \Lambda Q x \\ &= x^T Q^T \Lambda^2 Q x \end{aligned}$$

By substituting $y = Qx$:

$$\begin{aligned} ||Ax||_2 &= y^T \Lambda^2 y \\ &= \sum \lambda_i^2 y_i^2 \\ &\leq \sum \lambda_{max} y_i^2 \\ &= \lambda_{max} \sum y_i^2 \\ &= \lambda_{max}\, y^T y \\ &= \lambda_{max} \end{aligned}$$

Because $y^T y = (Qx)^T(Qx) = x^T Q^T Q x = x^T x = 1$. On the other hand, let q be eigenvector corresponding to λ_{max}. Of course $||q||_2 = 1$, because Q is orthogonal.

$$\begin{aligned} ||Aq||_2 &\geq ||Aq||_2 \\ &= ||\lambda_{max} q||_2 \\ &= \lambda_{max} ||q||_2 \\ &= \lambda_{max} \end{aligned}$$

Question 3 Let $e^{(j)} = x^{(j)} - x$ be the error vector. We want $\lim_{j\to\infty} e^{(j)} = 0$. In other words, we want $e^{(j+1)} < e^{(j)}$ for all sufficiently *big* j.

$$\begin{aligned} e^{(j+1)} = x^{(j+1)} - x &= b - x + (I - A)x^{(j)} \\ &= Ax - x + (I - A)x^{(j)} \\ &= -(I - A)x + (I - A)x^{(j)} \\ &= (I - A)(x^{(j)} - x) \end{aligned}$$

Here, we obtain: $e^{(j+1)} = (I - A)e^{(j)}$. Taking a norm both sides:

$$\begin{aligned} ||e^{(j+1)}|| &= ||(I - A)e^{(j)}|| \\ &\leq ||I - A||\, ||e^{(j)}|| \end{aligned}$$

Thus, the sequence converge to real solution if

$$||I - A|| < 1$$

More reading: `http://runge.math.smu.edu/Courses/Math3315_Spring10/iterative_linear.pdf`

3.2 Problem 2 – Automata Theory

Question 1 $x = a$, $y = bc$ and $z = c$

Question 2 Classic proof of Pumping Lemma (PL) for regular languages. Let $\mathcal{M}$ be an automaton with k states. Let $w = a_1 a_2 \cdots a_k \in L(M)$ such that $|w| > k$. Now let's simulate run of $\mathcal{M}$ of word w. Define states $p_i = \hat{\delta}(q_0, w_1 w_2 \cdots w_i)$. That is, p_i is a state in which $\mathcal{M}$ is after reading first i inputs. From pigeonhole principle, at lest two of those state must be exactly the same state. Let $p_i = p_j$ be the state that is visited the second time for the first time (i.e. i is the smallest among all such states).

I claim that: $w = xyz$, where

- $x = a_1 a_2 \cdots a_{i-1}$
- $y = a_i a_{i+1} \cdots a_{j-1}$
- $z = a_j a_{j+1} \cdots a_k$

Obviously, $|y| > 0$ because $i \neq j$ and $|xy| = j - 1 \leq n$. States $p_i, \ldots, p_j$ create a loop in the automaton - it can be traversed any number of times, thus $xy^n z \in L(\mathcal{M})$. For $n = 0$ we simply "skip" the loop, for $n \geq 1$ we traverse the loop n times.

Question 3 $L(M) = \{a^m b^n | n, m \in \mathbb{N}, 0 < m < n\}$ Assume that L is regular language. Then pumping lemma must hold. Consider $w = a^k b^{k+1} \in L$, where k is the pumping lemma constant. Because $|w| = 2k + 1 > k$, then there must exist a partitioning $xyz = w$ such that $|xy| < k$, $|y| > 0$ and $xy^n z \in L$ **for all** $n \in \mathbb{N}$. Notice that xy consists of a's only. Let's "pump up" y. For example, $xy^{42}z$ contains of significantly more a's than b's. This word does not belong to the language. Contradiction with statement that $xy^n z \in L$ for all $n \in N$. $L(M)$ is not regular, thus there exist no automaton recognizing it.

3.3 Problem 3 – Heapsort

Question 1

1. Building the heap – level-by-level, starting from the deepest level.
2. Extracting minimum n times. Extracting means printing the root value, replacing it with the last heap element, and then fixing heap order. If we represent the heap in an array of length n, we can swap first element (minimum) with the last element, fix heap order and call recursively on first $n - 1$ elements of array. Resulting array is sorted in descending order. Reverse it for ascending order.

Question 2 Let's put elements of $[3, 8, 1, 5, 4, 9, 7]$ into a binary tree:

```
        3
      /   \
     8      1
   /  \   /  \
  5    4  9   7          ] proper heaps
```

Note that the deepest level consists of 4 proper, one-element heaps. Let's fix the heap order on second to last deepest level:

```
        3
      /   \
     4      1            ]
   /  \   /  \           ] proper heaps
  5    8  9   7          ]
```

Now we have two proper heaps of height 1. Now, swap 3 and 1 to get the final answer:

```
        1
      /   \
     4      3
   /  \   /  \
  5    8  9   7
```

Question 3 For simplicity, assume that $n = 2^k - 1$, i.e. the heap the deepest level is fully occupied.

Phase 1, building the heap is $O(n)$ when building the heap from the bottom. On the deepest level we do nothing: it consists of $\lceil \frac{n}{2} \rceil$ nodes, each of them is a proper one-element heap. On second-to-the-last level, we perform two comparisons, and possibly one swap. In general we'll perform, ceil omitted for simplicity:

$$0 \cdot \frac{n}{2} + 1 \cdot \frac{n}{4} + 2 \cdot \frac{n}{8} + \cdots + O(h-1) \cdot 2 + O(h) \cdot 1 = \sum_{h=1}^{\lceil \log_2 n \rceil} \lceil \frac{n}{2^{h+1}} \rceil \cdot O(h) = O(n \sum_{h=1}^{\lceil \log_2 n \rceil} \frac{h}{2^{h+1}})$$

Now, let's evaluate the right-hand side:

$$\begin{aligned} n\sum_{h=1}^{\lceil \log_2 n\rceil} \frac{h}{2^{h+1}} &\leq \frac{n}{4}\sum_{h=1}^{\infty}\frac{h}{2^{h-1}} && \text{substitute } x = \frac{1}{2} \\ &= \frac{n}{4}\sum_{h=1}^{\infty} hx^{h-1} \\ &= \frac{n}{4}\frac{d}{dx}\sum_{h=1}^{\infty} x^h \\ &= \frac{n}{4}\frac{d}{dx}\frac{x}{1-x} \\ &= \frac{n}{4}\frac{1}{(1-x)^2} \\ &= \frac{n}{4}\cdot 4 = n \end{aligned}$$

Thus, we can bound the running time as:

$$O(n\sum_{h=1}^{\lceil \log_2 n\rceil}\frac{h}{2^{h+1}}) = O(n)$$

See Cormen's *Introduction to Algorithms, 3rd ed, Chapter 6.3* for more.

Phase 2 is $O(n\log_2 n)$: for each element out of n we perform extract-min, which takes $O(log_2 n)$.

Bonus: heapsort pseudocode:

```
heapsort(T[1..n])
    build-heap(T[1..n])
    for i = n..2
        swap(T[1], T[i])
        shift-down(T[1..(i-1)], 1)
```

Question 4 $O(n + k\log_2 n)$. First, we build heap in $O(n)$, then we perform ext tract-min k times.

Question 5 Let's look at an example. If input array is already sorted, then mergesort will just scan through the array with no swaps. Same with partition in quicksort. On the other hand heapsort, after building a heap ($O(n)$), will extract minimum and replace it with the last element of array – which is maximum. The maximum will be later "bubbled" down back to the deepest level. This means two comparisons and a swap at every of $O(\log_2 n$ iterations. This is pricey.

The other reason might be caching. In heapsort, accessing a parent might result in memory accesses which are distant more than $\lceil \frac{n}{2} \rceil$ (in particular leaves' parents). If n is big, we might have a lot of cache misses.

3.4 Problem 4 – Logic Circuit Design

Question 1 $a \veebar b = (a \wedge \neg b) \vee (\neg a \wedge b)$

Question 2 I'm not quite sure. What happens when both clock and input go 1? See Fig. 3.1

Question 3 See Fig. 3.2. We can construct xor_{256} using 8 2-input xors ("binary tree").

Give C_1 input C_0. For $i > 1$, C_i takes C_{i-1}'s output as an input. Additionally, we AND clock signal with negated W and provide it to every C_i.

We only modify X_1's input: it takes output of a 2:1 mutex with inputs X_0 (0) and the original xor_{256} value (1). See Fig. 3.3

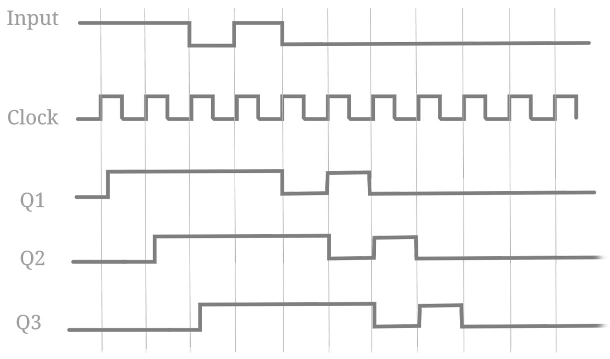

Figure 3.1: Question 2

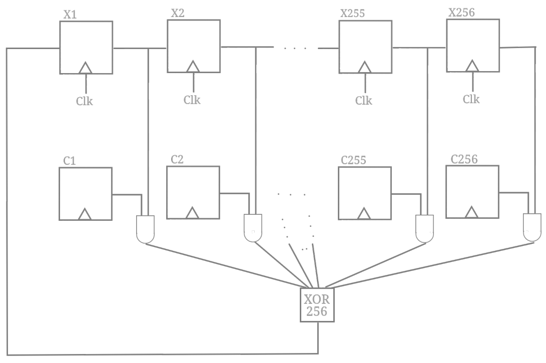

Figure 3.2: Question 3

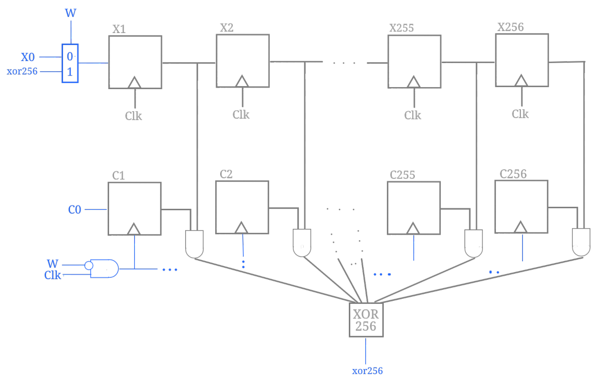

Figure 3.3: Question 4. Changes marked blue

Chapter 4

Winter 2016

4.1 Problem 1 – Linear Algebra – LU decomposition

Question 1 A lower triangular matrix M has $M_{i,j} = 0$ for $i < j$. Let $L, L' \in \mathbb{R}^{n\times n}$ be a lower triangular matrices. Let's look closer at entries above diagonal of LL', i.e. $(LL')_{i,j}$ for $i < j$:

$$(LL')_{i,j} = \sum_{k=1}^{n} L_{i,k} \cdot L'_{k,j}$$

Every item of the summation yields either $i < k$ or $k < j$. Thus, $(LL')_{i,j} = 0$ for $i < j$.

The same another way:

$$\begin{aligned}(LL')_{i,j} &= \sum_{k=1}^{n} L_{i,k} \cdot L'_{k,j} \\ &= \sum_{k=1}^{j-1} L_{i,k} \cdot L'_{k,j} + \sum_{k=j}^{n} L_{i,k} \cdot L'_{k,j} \\ &= \sum_{k=1}^{j-1} L_{i,k} \cdot 0 + \sum_{k=j}^{n} 0 \cdot L'_{k,j} \\ &= 0\end{aligned}$$

Question 2 Let $L, L' \in \mathbb{R}^{n\times n}$ be a *unit* lower triangular matrices, i.e $L_{i,j} = L'_{i,j} = 1$ for $i = j$. We know from Q1, that LL' is lower triangular. Let's examine diagonal items, $(LL')_{i,i}$:

$$(LL')_{i,i} = \sum_{k=1}^{n} L_{i,k} \cdot L'_{k,i} = L_{i,1}L'_{1,i} + L_{i,2}L'_{1,i} + \cdots + L_{i,i}L'_{i,i} + \cdots + L_{i,n}L'_{n,i} = 1$$

Question 3 Inverse of a unit lower triangular matrix is also unit lower triangular. So, we need to find only one entry of the inverse of $L1$, such that:

$$L_1L_1^{-1} = \begin{pmatrix}1 & 0\\ 2 & 1\end{pmatrix} \cdot \begin{pmatrix}1 & 0\\ x & 1\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

Obviously, $x = -2$. Similary, L_2:

$$L_2L_2^{-1} = \begin{pmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 3 & 2 & 1\end{pmatrix} \cdot \begin{pmatrix}1 & 0 & 0\\ a & 1 & 0\\ b & c & 1\end{pmatrix} = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$$

Here, $a = -2$, $b = 1$, $c = -2$.

Question 4 Prove that if A has two different LU decomposition, that is $A = LU = L_1U_1$ and L, L_1 are lower *unit* matrices, then $L = L_1$ and $U = U_1$.

Assume that inverses of U, U_1 exist. Start with $A = LU = L_1U_1 = A$ and multiply right-hand by U^{-1} and left-hand by L_1^{-1}:

$$\begin{aligned} LU &= L_1U_1 \\ L_1^{-1}LUU^{-1} &= L_1^{-1}L_1U_1U^{-1} \\ L_1^{-1}L &= U_1U^{-1} \end{aligned}$$

We know from (Q2) that left-hand side of the last equation is lower unit triangular matrix. In similar manner, we can show that right-hand is upper triangular. Lower and upper triangular matrices can be equal iff they are both diagonal. Moreover, since $L_1^{-1}L$ has ones on diagonal, so U_1U^{-1} must have. We conclude:

$$L_1^{-1}L = Id = U_1U^{-1}$$

That is, $L = L_1$ and $U = U_1$.

4.2 Problem 2 – Automata Theory

Question 1 $a^*b^*a^*$

Questions 2 - 5 See Fig. 4.1 - 4.4

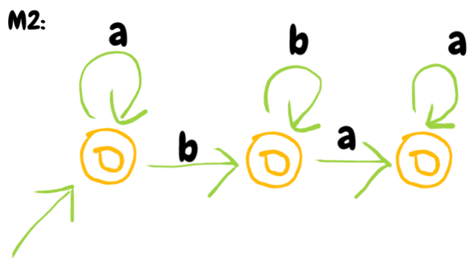

Figure 4.1: Question 2

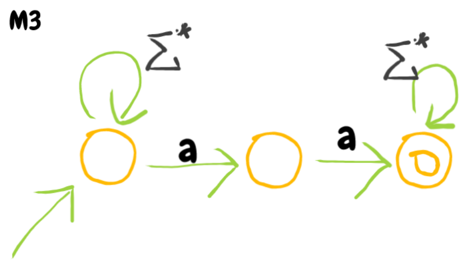

Figure 4.2: Question 3

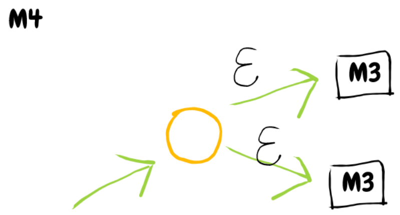

Figure 4.3: Question 4. One of the boxes should be labeled M_1.

4.3 Problem 3 – AVL Tree

Question 1 Easy

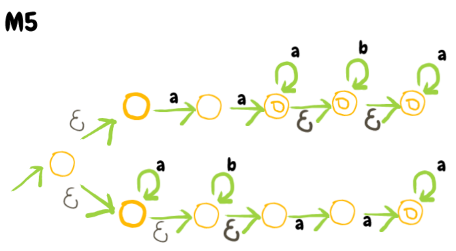

Figure 4.4: Question 5. $L(M_5)$ contains aa and is of form of $aaa^*b^*a^* + a^*b^*aaa^*$

Question 2 Minimum height of BST with n nodes is $\lfloor log_2 n \rfloor$. Maximum height: $n-1$.

Question 3 AVL tree with 7 nodes has min. height of 2 (full binary tree), and max. height of 3. See Fig. 4.5

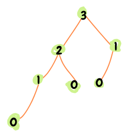

Figure 4.5: Max. height AVL Tree with 7 nodes. Numbers in nodes indicate height of a tree rooted in the node.

Question 4 Just $\lfloor log_2 n \rfloor$ – it's a full binary tree.

Question 5 Show that the height of a balanced binary tree (AVL) with n nodes is no more than $2log_2 n$.

Let's consider the smallest possible AVL trees wrt number of nodes. Let $N(h)$ indicate minimum number of nodes in a balanced tree of height h. The minimum tree of n nodes and height h consists of a root, a *minimum* subtree of of height $h-1$ and a *minimum* subtree of height $h-2$:

$$\begin{aligned} n = N(h) &= 1 + N(h-1) + N(h-2) \\ &\geq 1 + 2 \cdot N(h-2) && \text{because } N(h) > N(h-1) \\ &> 2 \cdot N(h-2) \\ &\geq 2^{\frac{h}{2}} \end{aligned}$$

Taking log both sides: $h \leq 2 \cdot log_2(n)$.

4.4 Problem 4 – Logic Circuit Design

Question 1

1. OR(x, y) = $majority(0, x, y)$
2. AND(x, y) = $majority(1, x, y)$
3. XOR(x, y) = AND(OR(x, y), NOT(AND(x, y)) = $majority[1, majority(0, x, y), not\ majority(1, x, y)]$

XOR is "2-M_3-level".

Question 2 Classic 1-bit full adder. Just draw a table and then depict the circuit. Let a, b, c_{in} be inputs.

$$S = a \veebar b \veebar c_{in} \qquad C = (a \wedge b) \vee (c_{in} \wedge (a \vee b))$$

S yields 4 - M_3 level (2x XOR). C yields 3 - M_3 level (OR with 2-level AND). Thus, FA_1 has 4 - M_3 level.

Bonus: $C = majority(c_{in}, a, b)$. Thanks Igor!

Question 3 Connect 4 FA_1 in series. It's M_3 level is $4 \cdot 4 = 16$.

Question 4 Classic 4-bit multiplier (google for images). Connect three 4-bit adders in series and try not to get messed with connections. M_3 level is: $3 \cdot 16 + 8 \cdot 1$: each of three FA_4 has level 16 and there's one level of 8 AND gates preceding FA_4's.

Chapter 5

Winter 2015

5.1 Problem 1 – Linear Algebra

Question 1 Let $A \in R^{n\times n}$ such that $A = A^T$ and $Ax = \lambda x$ for some eigenvector x and corresponding eigenvalue λ. Suppose that $\lambda \in \mathbb{C}$, i.e. $\lambda = a + ib$ for some $a, b \in \mathbb{R}$. Let's start with $Ax = \lambda x$:

$$\begin{aligned}
Ax &= \lambda x && \text{take conjugate transpose}\\
x^H A^H &= \lambda^H x^H && A \text{ is real, so } A = A^H\\
x^H A &= \lambda^H x^H && \text{multiply by } x\\
x^H Ax &= \lambda^H x^H x && \text{now, } Ax = \lambda x\\
\lambda x^H x &= \lambda^H x^H x\\
\lambda &= \lambda^H\\
a + bi &= a - bi
\end{aligned}$$

Hence, $\text{Im}(\lambda) = 0$, which means $\lambda \in R$.

Question 2 Start with the first eigenpair:

$$\begin{aligned}
Ax_1 &= \lambda_1 x_1 && \text{take transpose. } A = A^T\\
x_1^T A &= \lambda_1 x_1^T && \text{multiply by } x_2\\
x_1^T Ax_2 &= \lambda_1 x_1^T x_2
\end{aligned}$$

Now fiddle with the second eigenpair:

$$\begin{aligned}
Ax_2 &= \lambda_2 x_2 && \text{left-multiply by } x_1^T\\
x_1^T Ax_2 &= \lambda_2 x_1^T x_2
\end{aligned}$$

If we subtract both result, we get:

$$0 = (\lambda_1 - \lambda_2) x_1^T x_2$$

Since $\lambda_1 \neq \lambda_2$, then $x_1^T x_2 = 0$. In other words, $x_1 \cdot x_2 = 0$.

Question 3 - s e n d h e l p

$$A = \begin{pmatrix} x & a_1 & b_1 & c_1\\ x & a_2 & b_2 & c_2\\ x & a_3 & b_3 & c_3\\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$f(x, y, z) = \det(A)$$

This seems to be difficult. I succeeded in computing determinant using Laplace expansion wrt first column, but it gets hardcore later. Any ideas?

Question 4 Give 3 points that lie on plane $f(x, y, z) = 0$. When is determinant $= 0$? For example, when some columns are dependent. Let's make some columns dependent. Zum Beispiel, let's take:

$$x = a_1 \qquad y = a_2 \qquad z = a_3$$

Now first and second column are dependent, so $\det(A) = f(x, y, z) = 0$. To get two others solutions, take $x = b_1, y = b_2, z = b_3$ and $x = c_1, y = c_2, z = c_3$ or simply multiply the first solution by some constants.

5.2 Problem 2 – Automata Theory

Question 1

```
-> O --a--> O --b--> O --b--> (o)
      / \
      \ /
      a,b
```

Question 2 See Fig.5.1.

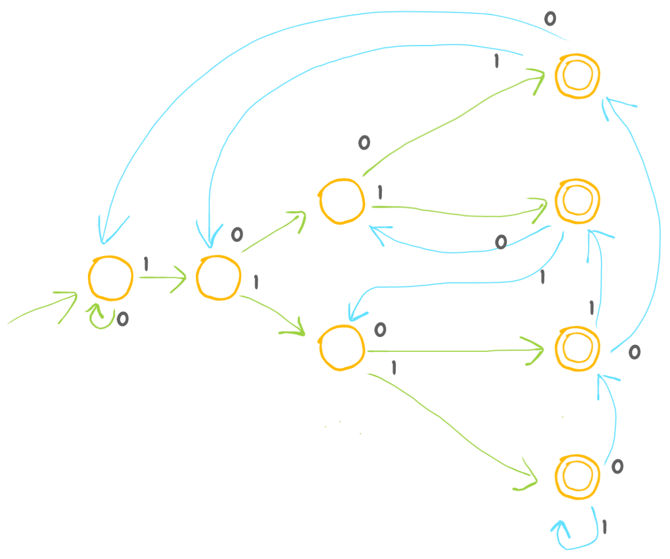

Figure 5.1: Question 2

Question 3+4 I'll prove that DFA that recognizes L_n:

$$L_n = \{w \in \Sigma^* |\ |w| \geq 3 \text{ and n-th to the last letter of } w \text{ is } a\}$$

has no less than 2^n states. *This is almost a copy-paste from* Introduction to Automata Theory, Languages and Computation 3rd ed., *Chapter 2.3.6.*

Intuitively we need 2^n states to remember every possible ending of length n: we can encode it as a binary string of length n. Now, I'll show that there's no such DFA with less than 2^n states.

Suppose that there's DFA D recognizing L_n with less tan 2^n states. If so, then tere must exist a state q in which D is after reading two different sequences, say $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_n$. Since they are different, let $i \in N$ be the last position on which they differ. By symmetry, assume that $x_i = a$ and $y_i = b$.

- if $i = 1$, then $x = a\, x_2 \cdots x_n$ and $y = b\, y_2 \cdots y_n$. Which means that state q is both accepting and non-accepting.
- if $i > 1$, then we can append $(n - i)$ b's to both x and y. Then a state p in which D is after reading $ax_{i+1} \cdots x_n b \cdots b$ and $by_{i+1} \cdots y_n b \cdots b$ is both accepting and non-accepting.

In both cases we get lead to contradiction. Thus, DFA recognizing L_n has at least 2^n states.

- **Question 3:** Let $n = 3$. Then DFA recognizing L_3 has no less than 8 states.
- **Question 4:** NFA recognizing L_n has exactly $n + 1$ states. Equivalent DFA has 2^n states.

5.3 Problem 3 – Heap

Question 1 After inserting 21, 26, the heap should look like:

```
         21
       /    \
     26       23
   /  \     /  \
  48   54  31   29
  |
  63
```

Question 2 After delete-min the heap should look like:

```
         25
       /    \
     48       42
   /  \
  35   73
```

Question 3 I understand *visiting a node* by accessing an array in which heap is stored.

To insert a node, we first put it in fist available "slot" on the deepest level and then we "bubble" it up. If the inserted element is smaller than other n elements, then we'll traverse whole tree. That means we will visit around $\lfloor log_2(n+1) \rfloor$ elements.

Question 4 First we, replace value with root with the last element in the heap array (and we erase that element). Then, we need to "bubble down" this element. At each iteration we'll visit its both children. In worst case, we'll bubble it down to the deepest level. So, we'll visit no more than $2\lfloor log_2(n-1) \rfloor$ elements.

Question 5 Maximum element will reside somewhere on the deepest level, and we need to visit them all. There are no more than $\lceil \frac{n}{2} \rceil$ elements on the deepest level.

5.4 Problem 4 – Synchronization

Question 1 Suppose that there are two processes. One of them may load value of *queue_head* before the other stores back incremented value. Then, both processes will access the same buffer element, although they must not.

Question 2 Semaphores (binary, counting), locks (test&set, swap), monitors.

Question 3 A semaphore is a shared variable with two defined operations executing atomically: *wait* and *signal*. *wait()* decrements semaphore value, and if the value becomes less than zero, then process calling *wait()* is made to sleep and is pushed to a queue. *signal()* increments semaphore value and, if the value is still negative, wakes up process at the beginning of the queue.

Note, that we only need to assure that every process is assigned different value of *queue_head*. Initial setting:

```
Semaphore mutex = Semaphore(1);
```

Code:

```
while(1) {
    wait(mutex);
    k = queue_head++;
    signal(mutex);
    ...
```

Chapter 6

Summer 2019 – S1

6.1 Problem 2 – Virtual Memory, Paging

Question 1

- *Page* – instead of loading the whole program into the memory, we divide it into fixed-sized chunks called *pages* and we load some of them to fixed-size chunks of physical memory called *frames* We load to the memory only those pages that we currently need. Bonus: motivation: processes spend 90% of their execution time accessing only 10% of their space in the memory.
- *Page Table* – stores mapping between virtual and physical addresses. It's a region in a memory where we can look-up actual page physical address.
- *Page Replacement* - when we cannot allocate a page in a memory, we need to evict some page residing in the memory.
- *Page fault* – access to the page which is not in the memory
- *TLB* - fast, hardware supported cache memory speeding up address translation (accessing an address via page table requires two actual physical memory accesses).

Question 2 A page has $4KB = 4 \cdot 1024B$. If we were to address every word (i.e. every $32b = 4B$) within a page, then there are $\frac{4 \cdot 1024}{4}$ possible addresses. To address them all we need $log_2 1024 = 10$ bits.

Thus, lower 10 bits of the virtual address make an offset within a page, and the rest of bits make index in the page table:

$$2A0F_{16} = 10.1010.0000.1111_2$$

Offset within a page: $10.0000.1111_2$, Page number, PageTable[1010_2] = 100_2. Sanity check: this page is valid, yay. The physical address corresponding to virtual $2A0F$ is:

$$1.00\,10.0000.1111_2 = 120F_{16}$$

Question 3 We can fit 1024 integers into one page. It is easier to look at A as a 1024×1024 2-dimensional array, which elements are stored continuously in the memory, row-by-row. We can fit one whole row into a page. Since memory size is $32KB$ and page has $4KB$ then 8 pages fit into the memory. However, one page is reserved, so we can store total of 7 rows of A in the memory.

Program 2 accesses A row by row. Thus, first 7 rows will be accessed with no page fault (PF). 8th and every following row row will cause PF. Hence, there will be:

$$1024 - 8 + 1 = 1017$$

page faults in total.

Program 1 accesses A column by column. Every element of a column will land in a different page, thus only first 7 accesses to A won't cause PF. We got:

$$1024 \cdot 1024 - 7 = 2^{20} - 7$$

page faults in total.

This one was also solved in Silberschatz's *Operating Systems Concepts, 9th ed.,* Chapter 9.9.5.

6.2 Problem 3 - Automata Theory

Question 1 Given DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, give an automaton accepting complement of $\mathcal{L}(\mathcal{A})$, i.e. $\Sigma^* \setminus \mathcal{L}(\mathcal{A})$.
Let $\mathcal{A}' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ Now, $w \in \mathcal{L}(\mathcal{A}')$ iff $\delta(w, q_0) \in (Q \setminus F)$ which is occurs only when $w \notin \mathcal{L}(\mathcal{A})$

Question 2 Given CFG $\mathcal{G} = (V, \Sigma, P, S)$, decide wheaterh $\mathcal{L}(\mathcal{G}) = \varnothing$.
We call symbol $A \in V$ *generating* if $A \Rightarrow^* w$ for some string w of terminals. If there's no such string, then A is *nongenerating.* Language of grammar $\mathcal{G}$ is empty iff start symbol S is nongenerating. We can find set of generating symbols using the algorithm below. Symbols that are not marked generating, are nongenerating. The algorithm:

- *Base.* Every terminal symbol form T is generating.
- *Induction.* If for some production $A \to \alpha$, α is known to be generating, then is A.

Question 3 Looks pretty obvious from the construction. A nice induction'd make it.

Question 4 If $(\mathcal{L}(\mathcal{G}) \cap \overline{\mathcal{L}(\mathcal{A})}) = \varnothing$, then $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{A})$ We can compute complement based on (Q1), intersection based on (Q3) and check for emptiness based on (Q2).

Chapter 7

Summer 2018 – S1

7.1 Problem 1 – Dijkstra

Question 1

```
s -- 3 -- a
 \       /
  4    -3
   \   /
     b
```

Dijkstra will find that the shortest path to a is 3, although it's 1.

Question 2

```
if (c[v] + d(v,u) <= c[u] {
    c[u] = c[v] + d(v,u);
}
```

Question 3

S	A	B	E
0	∞	∞	∞
0	6	3	9
0	5	3	9
0	5	3	7
0	5	3	7

Question 4 Each vertex is added to and removed from Q exactly once. Thus, (i) will run $|V|$ times, $O(|V|^2)$ in total. It takes $O(1)$ to execute (ii) once. In total, the for loop will iterate over all edges and every edge will be processed only once. Hence, (ii) will execute $O(|E|)$ in total.

Question 5 We will keep vertices in a heap, ordered by $c[v]$.

- Building the heap takes $O(|V|)$
- Looking up a minimum takes $O(1)$ (i, line 1)
- Removing minimum takes $O(\log |V|)$ (i, line 2)
- in (ii), we need to decrease key of an element in a heap. It takes $O(\log |V|)$

To conclude (i) takes $O(|V| \log |V|)$. (ii) takes $O(|E| \log |V|)$

7.2 Problem 2 – Automata Theory

Question 1 Given CFG productions: $S \to AA, A \to c, A \to aAb$, give parse tree of *aacbbc*.

```
          S
       /     \
     A       A
  / | \      |
a   A   b    c
|   |   |
a   A   b
    |
    c
```

Question 2 Give partitioning $acbc = uvwxy$ (pumping lemma). $u = \epsilon, v = a, w = c, x = b, y = c.$

Question 3+4: See chapter 7.2 *of* Introduction to Automata Theory, Languages and Computation 3rd ed. *for detailed proofs. (Example* 7.2.1 *for Q4)*

Question 3 Prove the pumping lemma for CFG.
Let G be CFG in Chomsky Normal Form (CNF).

Lemma 7.2.1. *Consider a syntax tree of a some word $s \in \mathcal{L}(G)$. If k is the length of a longest path in this tree, then $|s| \leq 2^{k-1}$.*

Proof. Number of leaves in binary tree of height $k-1$ does not exceed 2^{k-1}. □

Let M denote number of productions in G. Then let the pumping lemma constant be equal $N = 2^{M+1}$. Let $s \in \mathcal{L}(G)$ be longer than $N = 2^{M+1}$. The longest path in a syntax tree of s cannot be shorted than M, because if otherwise, from the lemma, s would be shorter than 2^M. Thus, the longest path must have length of at least $M + 1$. From pigeonhole principle, two states at the path must be the same state, say $A^{(i)} = A^{(j)}$. Assume, that $A^{(i)}$ appear on the path first, i.e. $i < j$. Now $s = uvwxy$:

- parse tree rooted in $A^{(i)}$ yields vwx. Moreover, $|vwx| < N$, because tree the longest path in rooted in $A^{(i)}$ is shorter than $M + 1 - i$. From the lemma, $|vwx| < 2^{M+1-i} \leq 2^{M+1} = N$.
- parse tree rooted in $A^{(j)}$ yields w.
- leaves to the left of $A^{(i)}$ make u, to the right: y.

Surely, $|vx| > 0$, because production with $A^{(i)}$ on left must have produced some terminals.
$s = uv^nwx^ny \in \mathcal{L}(G)$. For $n = 0$ we can cut $A^{(j)}$ and paste in place $A^{(i)}$. For $n > 0$ we can copy $A^{(i)}$ and paste it in place of $A^{(j)}$; repeat this $n - 1$ times.

Question 4 Prove that there's no CFG generating $L_4 = \{ww | w \in \{a, b\}^*\}$.

Proof. Assume that there's a grammar generating L_4. Then, the Pumping Lemma (PL) must hold. Let N be the PL constant. Consider a word $s = 0^N1^N0^N1^N$. Because $|s| > N$, then we can partition it $s = uvwxy$, such that $|vx| > 0$, $|vwx| < N$ and $uv^nwx^ny \in L_4$. Since $|vwx| < N$, then $|vwx|$ must:

- lie within blocks of first 0's. Let $|vx| = k$ Then $0^{N-k}1^N0^N1^N \notin L_4$.
- lie within block of second 0's or within some blocks of 1's. This case is symmetric to the above.
- overlap with some 0's and then 1's. Let k denote number of 0's in vx and l - number of 1's. Then neither $0^{N-k}1^{N-l}0^N1^N \notin L_4$ nor $0^N1^N0^{N-k}1^{N-l} \notin L_4$.
- overlap with some 1's and then 0's. This is case is symmetric to the previous one.

Every possible partitioning leads to contradiction. Thus, there is no CFG generating L_4. □

Chapter 8

Summer 2018 – S2

8.1 Problem 3 – Minimum AVL Tree / Fibonacci Tree

Setting: T_n stands for AVL tree of height n. Denote $N(n)$ for number of nodes in T_n.

Question 1 20

Question 2

$$N(n) = N(n-1) + N(n-2) + 1$$

Bonus: note that $N(n) = Fib_{n+3} - 1$ where $Fib_0 = 0$. Proof goes by induction: $N(0) = 1 = 2 - 1 = Fib_3 - 1$, $N(1) = 2 = 3 - 1 = Fib_4 - 1$. Inductive step:

$$N(n) = N(n-1) + N(n-2) + 1 = (Fib_{n+2} - 1) + (Fib_{n+1} - 1) + 1 = Fib_{n+3} - 1$$

Question 3 Note that $r^n = r^{n-1} + r^{n-2}$. To see that start with:

$$r^2 = (\frac{1+\sqrt{5}}{2})^2 = \frac{1+\sqrt{5}}{2} + 1 = r + 1$$

and multiply it by r^{n-2} both sides.

The rest goes by induction. Base case: $N(0) = 1 \geq r^0$, $N(1) = 2 > r^1$. Inductive step:

$$N(n) = N(n-1) + N(n-2) + 1 \geq r^{n-1} + r^{n-2} = r^n$$

Question 4 Again, by induction. Base: tree of height 0 has $1 < r^2$ nodes. Induction:

$$N(n) = 1 + N(n-1) + N(n-2) \leq 1 + r^{n+1} + r^n = r^{n+2} + 1$$

Chapter 9

Summer 2018 – Math

9.1 Problem 1 – Linear Algebra, Least Squares

Setting of the problem: we're solving $Ax = b$, where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$.

Question 1 (i) There are 3 linearly independent column vectors among $A = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ Just run Gaussian elimination.

(ii) $b = \begin{pmatrix} 2 \\ 4 \\ 2 \end{pmatrix} = 3 \cdot a_1 + 1 \cdot a_2 + a_3$, where a_i stand for column vectors of A.

(iii) a_1, a_2, a_3 are linearly independent, but b is linearly dependent with any of them. Thus, there are 3 linearly independent vectors among a_1, a_2, a_3, a_4.

Question 2 Show that solution for $Ax = b$ exists when $\text{rank}A = \text{rank}\bar{A}$ for arbitrary m, n, A, b. $\bar{A} = (A|b)$, i.e. A with additional column of b.

Rank of A, in other words, is a number of independent vectors, either columns or row ones. $\text{rank}A = \text{rank}\bar{A}$ means that adding one more column to A did not increased nor decreased number of independent vectors. That is, the added column, b, is linearly dependent with columns of A, that is, b can be represented as a linear combination of columns of A with coefficients of x. Thus, a solution for $Ax = b$ exist.

Question 3 $m > n$, $\text{rank}A = n$ and $\text{rank}\bar{A} > \text{rank}A$. Obtain x minimizing $||b - Ax||^2$.

Error vector $b - Ax$ is perpendicular to column space of A. Therefore, $b - Ax$ is in the nullspace of A^T, that is:

$$A^T(b - Ax) = 0$$
$$A^T b = A^T Ax$$

Because $\text{rank}A = n$, that is A has n independent columns, then $A^T A \in \mathbb{R}^{m\times m}$ is invertible. We obtain:

$$x = (A^T A)^{-1} A^T b$$

Question 4 Minimize $||x||^2$ subject to $Ax - b$ using Lagrange Multiplier. Let $v \in \mathbb{R}^m$. Write:

$$\mathcal{L}(x, v) = x^T x - v^T(Ax - b) = x^x - v^T Ax + v^T b$$

Take gradient wrt x:

$$\nabla_x \mathcal{L} = 2x - A^T v$$

Set the gradient to zero: $x = \frac{1}{2} A^T v$ and plug x into $Ax = b$: $AA^T v = 2b$. Again, AA^T is invertible so we can write $v = 2(AA^T)^{-1} b$. Finally, plug it back to x to get the final answer:

$$x = A^T(AA^T)^{-1} b$$

Question 5 $APA = A, PAP = P, (AP)^T = AP, (PA)^T = PA$. This is definition of *pseudoinverse* of a matrix. See wiki (click) for detailed proofs.

Suppose that there are two matrices P_1 and P_2 satisfying the above. Write:

$$AP_1 = AP_2AP_1 = (AP_2)^T(AP_1)^T = P_2^TA^TP_1^TA^T = P_2^T(AP_1A)^T = P_2^TA^T = AP_2$$

Similarly, $P_1A = P_2A$. Now we can conclude that:

$$P_1 = P_1AP_1 = P_1AP_2 = P_2AP_2 = P_2$$

Question 6 Show that both x obtained in (3) and (4) are in form $x = Pb$.

$x^{(3)} = (A^TA)^{-1}A^Tb$ and $x^{(4)} = A^T(AA^T)^{-1}b$. Write $P^3 = (A^TA)^{-1}A^T$ and $P^{(4} = A^T(AA^T)^{-1}$. Plug them both to equations in (Q5) to verify that those equations hold. Furthermore, we know that such P is unique, thus $P^3 = P^{(4)}$.

Chapter 10

Summer 2017 – S1

10.1 Problem 1 – Automata Theory

Question 1 $\Sigma = \{a, b\}$. Give NFA $\mathcal{A}_1$ with no more than 4 states recognizing $L_1 = \{w \in \Sigma \mid \exists l \in \Sigma \, |w|_l > 1\}$.

		a	b
s	q_0	q_0,q_1	q_0,q_2
	q_1	q_3	q_1
	q_2	q_2	q_3
*	q_3	q_3	q_3

Question 2 Prove: if Σ is finite alphabet, then any finite language $L = \{w_1, \cdots, w_n\} \subseteq \Sigma^*$ is regular, $n \in \mathbb{N}$.

That is we should construct a finite automaton accepting L. We can construct a ϵ-NFA containing n "branches", each recognizing w_i. Now, for every ϵ-NFA, there must exist equivalent DFA recognizing the same language.

We can also construct the DFA explicitly. Start with automaton (NFA) recognizing w_1: there are $|w_1| + 1$ states, the last one is accepting and transitions are labeled with next letters of w_1. For w_i ($i = 2, \cdots, n$) try to traverse the automaton as far as you can, i.e. until transition for symbol w_{ij} exist. If we can go no further, that is we read the longest common prefix of w_i and some w_k, $k < i$, then we make a new branch from a current state. This branch consist of states and transitions labeled $w_{i,j+1} \cdots w_{i,|w_i|}$.

Now we need to assure that we constructed a DFA. For every state missing some transitions on some letters, add those transition leading to a "dead state", i.e. nonaccepting state with a self-loop labeled Σ.

Question 3 Give DFA recognizing complement of L_1 from (Q1), i.e $L_2 = \Sigma^* \setminus L_1$.

Obviously $L_2 = \{w \in \Sigma \mid \forall l \in \Sigma \, |w|_l \leq 1\} = \{\epsilon, a, b, ab, ba\}$.

		a	b
s,*	q_1	q_2	q_3
*	q_2	q_5	q_4
*	q_3	q_4	q_5
*	q_4	q_5	q_5
	q_5	q_5	q_5

Question 4 Given NFA $\mathcal{A}$, decide whether $\mathcal{L}(\mathcal{A})$ is empty or not. Since language of $\mathcal{A}$ is regular, then from pumping lemma we can "pump" words longer than some N – pumping lemma constant. That is, if $\mathcal{L}(\mathcal{A})$ has some word longer than N, then $\mathcal{L}(\mathcal{A})$ is infinite. We just need to check every possible word w: $N < w \leq 2N$. If any such word is accepted by $\mathcal{A}$, then $\mathcal{L}(\mathcal{A})$ is infinite. We don't need to check words longer of $2N$: if a word is longer than $2N$, then from PL, we can iterative reduce its length, so w has length shorter than $2N$.

How to choose N? We know that for every NFA $\mathcal{A}$, there exist equivalent DFA $\mathcal{D}$, that is, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{D})$ (subset construction). We don't need to construct such DFA. All we know is that, $\mathcal{D}$ might have exponentially more states than $\mathcal{A}$. Take $N = |\Sigma|^{|Q_D|} + 42$, where Q_D is set of $\mathcal{D}$'s states.