MATH 409
WEEK TWO EXERCISES

Solution to Exercise 3, Lecture 4.

(i) $1025 = 2^{10} + 1 = 1 \cdot 2^{10} + 0 \cdot 2^9 + 0 \cdot 2^8 + 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$ so the binary representation of 1025 is 10000000001, plus the sign-bit (to record that the number is positive). Its bit complexity is $= 1 + \lceil \log_2(1025 + 1) \rceil = 1 + \lceil 10.003 \rceil = 1 + 11 = 12$.
The binary representation of 1024 is 10000000000 plus the sign-bit in 0 because it is positive. Its bit complexity is $= 1 + \lceil \log_2(1024 + 1) \rceil = 1 + \lceil 10.001 \rceil = 1 + 11 = 12$.
For -1023 the binary representation is 1111111111 plus the sign-bit in 1 because it is negative, and its bit complexity is $= 1 + \lceil \log_2(1023 + 1) \rceil = 1 + \lceil 10 \rceil = 1 + 10 = 11$.
Notice that $\lceil \log_2(\text{ number } +1) \rceil$ is the number of bits necessary to write down the binary representation of the corresponding number.

(ii)

$$\begin{array}{rcl} 11100101 & \longrightarrow & 229 \\ +\underline{\quad 11001} & \longrightarrow & +\underline{\ 25} \\ 11111110 & \longrightarrow & 254 \end{array}$$

(iii)

$$\begin{array}{rcll} 425 & \cdot & 213 & = 90525 \\ \downarrow & & \downarrow & \downarrow \\ \underline{110101001} & \cdot & 11010101 & = 10110000110011101 \end{array}$$

$$\begin{array}{r} 110101001 \\ 000000000 \\ 110101001 \\ 000000000 \\ 110101001 \\ 000000000 \\ 110101001 \\ 110101001 \end{array}$$

□

Solution to Exercise 3, Lecture 5.

(i) An $m \times n$ matrix has m rows and n columns, and for simplicity we will assume that $n \leq m$.

The arithmetic model. To eliminate the first element of the second row we need: one division, $n-1$ multiplications and $n-1$ subtractions, i.e., to make one row elimination we need $O(n)$ operations. To eliminate all the first column below the diagonal we need $(m-1)*O(n) = O(mn)$ operations. To clear any other column we need roughly the same number of operations, so the whole Gaussian elimination requires $O(mn^2)$ many operations. Is necessary to observe that sometimes we may need to switch rows when a zero appears in the diagonal. To make a switch of rows takes constant time (of course it depends on your data structure, but in the worst case it takes $O(n)$ operations), and at most we need n switches, i.e. a complexity of $O(n^2)$, so this does not increases the complexity. (If $n > m$ the complexity will be $O(m^2 n)$).

The bit complexity model. In this case we need to keep track of the size of the numbers. Let M be the maximum bit complexity of the integers of the matrix. In order to avoid divisions (which will make the complexity even higher) we will do the elimination of the second row in the following way, replace the second row by: (first row times $a_{2,1}$) - (second row times $a_{1,1}$). This requires $2n$ products and n subtractions. Each product has complexity $O(M^2)$ and the result are numbers of bit complexity $2M$; the subtractions will take $O(2M) = O(M)$. Therefore one row elimination has complexity $O(nM^2)$, and the elimination of the whole first column will have complexity $O(mnM^2)$. Now we need to do the elimination of the second column, but we need to observe that the bit complexity of the integer now is $2M$. It might be tempting to say that this is just $O(M)$ and keep ahead ignoring the increment on the bit complexity of the numbers. But after a second column elimination the bit complexity of the numbers will be $4M$, after three column eliminations it will be $8M$, and so on. After j column eliminations the bit complexity of the numbers will be $2^j M$, so the multiplication of two numbers at that step has complexity $O(2^{2j} M^2)$. The worst case is when $j = n$, at that point the bit complexity of the numbers can be $2^n M$.

Since we consider the worst case for our complexity computations we will consider all multiplications to have complexity $O(2^{2n}M^2)$ and for subtractions we assume a complexity of $O(2^nM)$. This gives a complexity of $O(2^{2n}M^2)$ for each operations. The Gaussian elimination requires $O(mn^2)$ operations (from the arithmetic model), so the complexity for the binary case is $O(mn^2 2^{2n}M^2) = O(m2^{2n}M^2)$. (If $n > m$ the complexity will be $O(n2^{2m}M^2)$).

(ii) To do the Gauss-Jordan elimination we first do the elimination downwards and then we do it upwards. This requires at most twice the number of operations, which does not change the complexity.

(iii) If A is a non-singular matrix then $n = m$. To compute the inverse of A we augment the $n \times n$ identity matrix to the right of A and do Gauss-Jordan elimination to the enlarged matrix; since Gauss-Jordan elimination is equivalent to left multiplication by A^{-1}, the inverse of A will appear augmented to the identity matrix. We now have $n' = 2m'(n' > m' = m = n)$, our previous analyses apply and the complexity of computing the inverse in the arithmetic model will be $O((m')^2n') = O(n^3)$. In the bit complexity model it will be $O(2^{4n}M^2)$.

(iv) One way of computing the rank of a matrix is just by doing Gaussian elimination and counting the number of non-zero rows. This has the complexity computed in part (i). □

Solution to Exercise 1, Lecture 6. Assume that $T\setminus\{e\}$ is connected. Let $e = \{v_i, v_j\}$; since $T \setminus \{e\}$ is connected, there exists some path $e_0, e_1, \ldots, e_n$, from v_j to v_i in $T \setminus \{e\}$. Then the path $e_0, \ldots, e_n, e$ is a cycle, contradicting that T is a tree. We conclude that $T \setminus \{e\}$ is disconnected; in particular, we have shown that there is no path from v_i to v_j.

Now we want to prove that $T \setminus \{e\}$ is a forest of two trees. Since T is connected, for each vertex $v \in V(T)$, there is a path from v to v_i, say $e_1, \ldots, e_n$. This path is unique because T is acyclic. Let V_1 consist of those vertices v of T such that this path does not contain e; let V_2 be those vertices v of T such that this path does contain e. This partitions the set $V(T)$. Any edge of $T \setminus \{e\}$ joining a vertex $v_1 \in V_1$ and $v_2 \in V_2$ yields a path from v_i to v_1 to v_2 to v_j in $T \setminus \{e\}$; this is absurd. Thus we may let E_1 be all of those edges of $T \setminus \{e\}$ between vertices of V_1 and let E_2 be those edges between vertices of V_2.

It remains only to show that (V_i, E_i) is connected, $i = 1, 2$. Choose any two vertices $u, v \in V_1$. By construction, we have a path p_u from

u to v_i and p_v from v to v_i. Then $p_u p_v^{-1}$ is a path from u (to v_i) to v. The case of V_2 is similar. □

Solution to Exercise 3, Lecture 3. We will prove that $T' = (V, E(T) \setminus \{k,l\} \cup \{i,j\})$ is a tree; by Lemma 2 of lecture 6, it is sufficient to show that T' is a spanning connected subgraph of G with $n-1$ edges (n is the number of vertices of G).

It is clear that T' is spanning because its node set is V. Since T is a spanning tree, we know that it has $n-1$ edges; removing $f := \{k,l\}$ (which is in T) and adding $e := \{i,j\}$ (which is not in T) preserves this number of edges. It remains only to prove that T' is connected.

We first fix a path Q from i to j in T. This path is unique since T is a tree. We can factor this path Q as $Q = Q_i \cdot f \cdot Q_j$ from i to k to l to j. Now choose any two vertices $u, v \in V(T')$. Find a path in P from u to v in T. Observe that f cannot appear in Q_i, Q_j. If f is not in P, then P is also a path in T', and so u and v are connected in this case. Otherwise, f is in P. This means that $P = P_u \cdot f \cdot P_v$ is a path from u to (say) k to l to v; we know that f does not appear in P_u or P_v, so these are also paths in T'. It follows that

$$P_u Q_i^{-1} \cdot e \cdot Q_j^{-1} P_v$$

is a path in T' from u to v, so that T' is connected. Thus T' is a tree.

The number of TSP tours in K_n is $(n-1)!/2$, this number is much smaller that n^{n-2}, actually $(n-1)!/2n^{n-2} \to 0$ whenever $n \to \infty$. □

Solution to Exercise 6, Lecture 3. Suppose that G is a graph with possibly negative edge weights. Since there are finitely many edges, there exists a minimal edge weight, $w \in \mathbb{R}$. For each edge $e \in E(G)$, replace the edge weight c_e with $c_e + |w| + 1$. This construction yields a new weighted graph G', for which all edge weights are positive.

Observe that the two sets of spanning trees for G and G' are identical; furthermore, we know that each spanning tree contains $n-1$ edges. Let tot be the function that gives the sum of the edge weights for any spanning tree of G, and tot' be the same function for G'. It is easy to see that

$$tot(T) + (n-1)(|w|+1) = tot'(T)$$

for every subgraph T. It follows that a minimal spanning tree T of G, i.e. a minimum of tot, is also a minimum for tot' since these two functions differ by a constant, and so is a minimal spanning tree for G', and conversely. □