Solutions to exercises of lecture 13

Excercise 13.1. The successor matrix is an implicit representation of the shortest paths. A more explicit representation of the shortest paths would be a sequence of vertices describing the shortest path for every pair. Give a graph with $\Omega(n^2)$ pairs of distance $\Omega(n)$ from each other to show that such a representation will require $\Omega(n^3)$ to compute.

Solution. A path already does the job: the first $n/4$ vertices are of distance at least $n/2$ from the last $n/4$ vertices.

Excercise 13.2. In Exercise 5.3 you gave an algorithm for determining whether a clique of size k exists with running time $O(n^k k^2)$. Unfortunately, researchers believe that clique parameterized by k is not FPT, and in this exercise you are asked to find the currently asymptotically fastest algorithm for this problem for the case when k is a multiple of 3: show how to determine whether a clique on k vertices exists in $O(n^{\omega k/3}\texttt{poly}(k))$ time in this case. Hint: start with $k = 3, 6$.

Solution. For $k = 3$ this is exactly the problem of finding a triangle which we solved by raising the adjacency matrix to the power 3. For $k = 3l$ with $l > 1$ and input graph $G = (V, E)$ construct graph $G' = (V^l, E')$ (i.e. G' has a vertex for every l-tuple of vertices from G), where $((v_1, \ldots, v_l), (v'_1, \ldots, v'_l)) \in E'$ if $\{v_1, \ldots, v_l, v'_1, \ldots, v'_l\}$ is a clique on $2l$ vertices in G. We see that G' has a triangle if and only if G has a clique on $3l = k$ vertices, since the triangle implies all needed edges are present in G and if G has a clique of size k we can partition its vertices in three parts and the associated tuples will give a triangle in G'.

Constructing the adjacency matrix of G' costs $O(n^{2l}k^2)$ time while finding a triangle in G' costs $O(n^{\omega k/3})$ time.

Excercise 13.3. In the Max-Cut problem we are given an undirected graph $G = (V, E)$ and need to find a partition of V into V_1, V_2 maximizing $E \cap (V_1 \times V_2)$. It is known that Max-Cut is NP-complete. Show that MAX-2-SAT is NP-complete and solve Max-Cut in $O^*(2^{\omega n/3})$ time.

Solution. Given an instance of Max-Cut we can create an instance of MAX-2-SAT on n variables $v_1, \ldots, v_n$ indicating whether a vertex is in v_1 or v_2, and with for every edge $(v_1, v_2) \in E$ clause $v_1 \vee v_2$ and $\neg v_1 \vee \neg v_2$. We see that every assignment satisfies at least one of these clauses and both are simultaneously satisfied if and only if $v_1 \neq v_2$. Thus we can satisfy at least $m + x$ clauses if and only if we have a cut splitting at least x edges. Since we know how to solve MAX-2-SAT in $O^*(2^{\omega n/3})$ time and the reduction did not increase n this can be used to solve Max-Cut in $O^*(2^{\omega n/3})$ time.

Excercise 13.4. It is a big open problem to solve MAX-3-SAT (which has the same definition of MAX-2-SAT except we are given a 3-CNF formula) in $O^*((2-\epsilon)^n)$ time for some $\epsilon > 0$. Why can we not use the approach from Section 13.3 for MAX-2-SAT for this?

Solution. We do not know where to account for the clauses with a variable in all three parts.

Excercise 13.5. The n'th Fibonacci number f_n is defined as follows: $f_1 = 1$,$f_2 = 1$ and for $n > 2$, $f_n = f_{n-1} + f_{n-2}$. Show that $\begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$. Show how to compute the n'th Fibonacci number using $O(\log_2(n))$ arithmetic operations. Why is this 'running time' misleading?

Solution. Proof by induction. Holds for $n = 1$ (if we let $f_0 = 0$). For the induction step just apply the definition of matrix multiplication:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_n & f_{n-1} \\ f_{n-1} & f_{n-2} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_n + f_{n-1} & f_n \\ f_{n-1} + f_{n-2} & f_{n-1} \end{pmatrix}.$$

If A is the 2×2 matrix above we can compute A^n with $O(\log(n))$ matrix multiplications by using $A^{2x} = A^x A^x$ and $A^{2x+1} = A^{2x}A$. The running time is misleading since f_n is exponential in n, so representing f_n already requires $O(n)$ bits, so the arithmetic operations used by this procedure take $O(n)$ time.

Excercise 13.6. The transitive closure of a directed acyclic graph $G = (V, E)$ is the graph $G^* = (V, E^*)$ where $(u, v) \in E^*$ whenever there is a path from u to v in G. Compute the transitive closure of a directed acyclic graph in $O(n^\omega \log(n))$ time.

Solution. In Section 13.4 we define the square of a graph/adjacency matrix G as the graph G' with an edge between two vertices iff the two vertices are of distance at most 2 in G. The adjacency matrix of the square of a directed graph can still be compute via matrix multiplication: if $B = A^2$, b_{ij} still is the number of walks on two edges from i to j in the directed setting. Therefore, we let denote $sq(A)$ for the square of A, we can simply compute $A^* = sq(sq(\ldots sq(A)))$ (e.g. squaring $\log_2 n$ times) and since all if there is a path from u to v the distance of u to v is at most n, G^* will have an edge between such u and v if A^* is the adjacency matrix of G^*.

Excercise 13.7. Why doesn't Algorithm `succ` work for directed graphs?

Solution. Among others, it is not clear how to determine the parities of the d_{ij} since (13.1) does not hold.