P1) A domino is shape like $\square$ or $\blacksquare$. Given an $n \times n$ table where some of the squares are removed (in the picture below removed squares are marked with an $X$), design a polynomial time algorithm that outputs the maximum number of dominos that can be placed on the table which are not overlapping and don’t cover any X cells.

For example, given the table on the left the maximum number of dominos that can be placed is 2.

\begin{center}
\begin{tabular}{|c|c|}
\hline
X & \cellcolor{white} \\
\hline
\cellcolor{white} & X \\
\hline
\cellcolor{white} & X \\
\hline
\end{tabular}
\end{center}

**Solution:** We construct an instance of the bipartite matching problem: First we construct a graph $G$: We put a vertex for every square of the table which is not marked with an X; we connect two vertices $u, v$ with an edge if we can place a domino on them, i.e., if the corresponding two squares share a side.

We claim that $G$ is a bipartite graph: To see that it is enough to color the graph with two colors such that any adjacent pair of vertices have distinct colors. We color the cells of the table like a chessboard, black/white. It follows that any two neighboring squares have opposite colors, therefore $G$ is bipartite.

**Algorithm:** We return the size of the maximum matching in $G$.

**Runtime:** Runtime is the time to compute the maximum matching in $G$, $G$ has $O(n^2)$ vertices and $O(n^2)$ edges. So, max matching runs in time $O(n^4)$.

**Correctness:** We show max-matching $= \text{max number of dominos that can be placed that}$

$\text{don’t cover any X cells.}$

**Max Matching $\geq \text{Max number of dominos:}** Suppose the maximum number of dominos we can place on the table is $k$; choose the corresponding edge for every domino that is placed call this set of edges $M$. We claim that $M$ is a matching in $G$ (with $k$ edges). This is because every cell is covered by at most one domino, so every vertex of $G$ is adjacent to at most one edge of $M$. Further, since no dominos are placed on X-cells, each domino corresponds to exactly one edge of $G$. This implies Max matching in $G$ is at least the max number of dominos.

**Max number of dominos $\geq \text{Max Matching}** Let $M$ be a maximum matching with $k$ edges in $G$. Since every edge in $G$ corresponds to two adjacent cells in the table, we can put one domino on the table for every edge in $M$. Furthermore, since $G$ does not have vertices for the X-cells, no domino will be placed on the X-cells and lastly since $M$ is a matching the dominos are not overlapping. Therefore, we can put $|M|$ many dominos on the table which are not overlapping and do not cover any X’s as desired.
P2) Given an (unweighted) directed graph $G = (V, E)$, a pair of vertices $s, t$ and an integer $1 \leq k \leq n$. Design an algorithm that runs in time polynomial in $n, k$ and outputs yes if there are $k$ vertex disjoint paths from $s$ to $t$ and no otherwise. For example, in the following graph there are two edge disjoint paths from $s$ to $t$ but no two vertex disjoint paths from $s$ to $t$.

For this problem you can assume you have access to a polynomial time algorithm for the edge disjoint path problem defined as follows: Given a directed graph $G$ and a pair of vertices $s, t$ we want to find the maximum number of edge disjoint paths from $s$ to $t$. Two paths $P_1, P_2$ from $s$ to $t$ are edge disjoint if they don’t share an edge. We will discuss the solution to this problem in class on Friday.

Solution: Construct $H$ from $G$ by splitting each vertex $v \neq s, t$ to an “in” and an “out” vertex.

For any edge $u \rightarrow v$ in $G$ we connect $u_{\text{out}}$ to $v_{\text{in}}$. In the special cases of $s \rightarrow v$ or $v \rightarrow t$, we simply connect $s$ to $v_{\text{in}}$ and $v_{\text{out}}$ to $t$, respectively, in $H$ and

![Diagram](attachment:network.png)

becomes

![Diagram](attachment:network_in_out.png)

Then, we run the algorithm from class to find the maximum number of edge disjoint paths from $s$ to $t$ in $H$ and we output that number. The algorithm obviously runs in time polynomial in $n$ as it takes $O(m+n)$ to construct graph $H$ and the algorithm to find the maximum number of edge disjoint paths runs in time $O(mn)$.

Correctness: First, observe that there is a natural bijection between paths from $s$ to $t$ in $G$ and $H$. For any path from $s$ to $t$, say $v^0 = s, v^1, \ldots, v^k = t$ the path $s, v_{\text{in}}^1, v_{\text{out}}^1, v_{\text{in}}^2, v_{\text{out}}^2, \ldots, v_{\text{in}}^k, t$ is a path from $s$ to $t$ in $H$. And, conversely a path from $s$ to $t$ in $H$ is of the form $s, v_{\text{in}}^1, v_{\text{out}}^1, \ldots, v_{\text{out}}^{t-1}, v_t$; this is because the only out-going edges of $s$ go to in-vertices and every in-vertex has a unique outgoing vertex to an out-vertex, so in/out vertices should alternate and we should end at an out-vertex before we go to $t$.

We claim that the maximum number of vertex disjoint paths in $G$ is equal to the maximum number of edge disjoint paths in $H$.

$\leq$: suppose we have $k$-vertex disjoint paths in $G$ $P_1, \ldots, P_k$ from $s$ to $t$; then by the above bijection we get $k$-paths $P_1', \ldots, P_k'$ from $s$ to $t$ in $H$. These paths are edge disjoint in $H$ simply because every $v_{\text{in}} \rightarrow v_{\text{out}}$ edge can be used in at most one path, the only possible path among $P_1, \ldots, P_k$ that has vertex $v$.

$\geq$: Suppose we have $k$-edge disjoint paths $P_1, \ldots, P_k$ from $s$ to $t$ in $H$. Then, by the above bijection, they map to $k$ paths $P_1', \ldots, P_k'$ in $G$ from $s$ to $t$ in $H$. Observe that each of the edges $v_{\text{in}} \rightarrow v_{\text{out}}$ can be used in at most one of $P_1, \ldots, P_k$. This implies that the paths $P_1', \ldots, P_k'$ are vertex disjoint.

Problem Solving Session 7-2