Midterm 2

Name:

Penn State ID:

Student ID number:

Instructions: Answer all questions (no % 20 option on the exam). Read them carefully first. Be precise and concise. Handwriting needs to be neat (if we can't understand what you write, we can't grade it). Box numerical final answers. Write only on the provided space; if you need extra space for a question, put a note in the question and use the extra pages provided at the end.

Please clearly write your name and your PSU Access User ID (i.e., xyz1234) in the box on top of every page.

Good luck!

Name:	PSU Access ID (xyz1234):

True/False *(40 points)*

True or false? Mark the right answer with an "x". No justification is needed. No points will be subtracted for wrong answers. In some cases, it wil be helpful to come up with counterexamples or short proofs to convince yourself.

T **F**

☐ ☐ The shortest paths between all pairs of vertices in a graph with both positive and negative weights (but no negative cycles) can be found in $O(|V|^3)$ time.

True. The Floyd-Warshall algorithm we saw in class has this running time.

☐ ☐ All dynamic programming algorithms run in polynomial time with respect to the size of the input.

False. The running time of a DP algorithm can essentially be anything. As an specific counterexample, we saw DP algorithms for the Knapsack problem with exponential running times.

☐ ☐ In a dynamic programming algorithm, the time required by the algorithm is determined by the number of nodes in the DAG of the subproblems.

False. It's linear in the number of nodes and edges of the DAG. This one turned out to be a tricky one, so anyone who got it wrong got 1pt.

☐ ☐ The dynamic programming algorithm for KNAPSACK WITHOUT REPETITION runs in polynomial time.

False. The DP Knapsack algorithm runs in exponential time with respect to the input.

☐ ☐ Every NP-Hard problem is in NP.

False. By definition, an NP-Hard problem need not be in NP.

☐ ☐ Every NP-complete problem is in NP.

True. By definition.

☐ ☐ If a problem A reduces to a problem B (in polynomial time), and B is NP-complete, then A is NP-complete (Think of A and B as problems in NP.)

False. The reduction is in the wrong direction to make this claim. This would imply that all problems in NP are NP complete (since B being NP-complete means by definition that all NP problems reduce to it, and so A reduces to it in particular.)

☐ ☐ 2SAT is in P.

True. As you showed in Homework 1, 2SAT can be solve in polynomial time.

☐ ☐ In practice, using recursion and memoization is slower than the standard dynamics programming approach.

True. There is the additional overhead of the recursive calls when doing memoization, etc.

☐ ☐ The class P contains the set of all problems that can be solved in polynomial time.

False. P only contains the decision problems that can be solved in polynomial time.

☐ ☐ If P=NP, then there exists an algorithm with $O(n^5)$ running time for the Traveling Salesman Problem.

False. If P=NP, then there is a polynomial time algorithm for TSP, but there is not guarantee on the actual polynomial; i.e., could be $O(n^100)$.

☐ ☐ All problems in NP can be solved with polynomial space.

True. This was proved in class.

☐ ☐ A problem X is NP-complete if there is a polynomial time reduction from every problem in NP to X.

False. X needs to be in NP.

☐ ☐ For a given problem X, if there is a polynomial time reduction from X to every problem in NP, then X is NP-hard.

False. Note for example, that if X is in P, then it has a trivial reduction to anything in NP.

☐ ☐ If there is a polynomial time reduction from 3SAT to X $\in$ NP, then X is NP-complete.

True. Since we know that 3SAT is NP complete.

☐ ☐ There is a polynomial time reduction from the problem of finding the maximum flow in a graph to 3SAT.

True. Simply run the polynomial time algorithm for max flow we saw in class.

☐ ☐ Let A and B be two problems in a complexity class $\mathcal{C}$. If there exists a polynomial time reduction from A to B, then there exists a polynomial time reduction from B to A.

False. This is not necessarily the case. For example, anything in P reduces to anything in NP in polynomial time, but the converse is not known to be true. So, as an specific counterexample, you can take $\mathcal{C} = P \cup NP$. This one is a tricky one, so anyone who got it wrong will get 1pt.

☐ ☐ The problem of counting all self-avoiding walks in the two-dimensional grid is in PSPACE.

True. As seen on the homework. (HW4, P10)

☐ ☐ The variant of the 3SAT problem in which each literal appears at most twice in the formula can be solved in polynomial time.

False. As seen on the homework. (HW4, P8)

☐ ☐ If an n-vertex graph has an independent set of size k, then it has a vertex cover of size $n - k$.

True. If I is an independent set, then $V \setminus I$ is a vertex cover.

Name:	PSU Access ID:

Short questions *(10 points)*

(a) Consider a binary tree T with root r. Let v and u denote the children of the root r, v_1 and v_2 the children of v, and u_1 and u_2 the children of u. Let $S(w)$ denote the size of the largest independent of in the subtree of T rooted at vertex w. Provide a recurrence for $S(r)$ in terms of $S(v)$, $S(u)$, $S(v_1)$, $S(v_2)$, $S(u_1)$ and $S(u_2)$. Justify briefly.

$$S(r) = \max\{S(u_1) + S(u_2) + S(v_1) + S(v_2) + 1, S(u) + S(v)\}.$$

This was a DP example from lecture. For the explanation check the notes/videos or the textbook.

(b) Consider the dynamic programming algorithms familiar from the textbook and the lectures for solving these problems:

- E: Edit distance of two strings
- K: Knapsack (without repetitions)
- C: Chain matrix multiplication
- I: Independent set on a tree
- F: Floyd-Warshall

Consider the DAG of subproblems where the orientation of the edges is from "smaller" subproblems to "bigger" ones. For which of these algorithms the DAG of the subproblems:

(a) has all in-degrees at most two. **K**

(b) has all in-degrees bounded from above by some constant independent of the instance. **E,F,K**

(c) "looks like a triangle" **C**

(d) is exponential in the length of the input **K**

(e) the algorithm runs in linear time **I**

(f) the algorithm runs in $\Theta(n^3)$ time **C,F**

This one turned out to be a tricky one, so everyone received +2.5 pts on this one.

Name:	PSU Access ID:

Dynamic Programming: minimize spaces *(20 points)*

Given a sequence of n words of width $w_1, \ldots, w_n$, our goal is to split the sequence of words into lines to minimize the amount of extra spaces. Every line has width W, and if it contains words i through j, then amount of extra spaces on this line is $S(i,j) = W - j + i - \sum_{k=i}^{j} w_k$, since we leave one unit of space between words. Of course, we require that $S(i,j) \geq 0$ to avoid line overflowing.

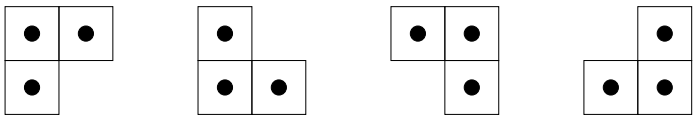
Our goal is to minimize the sum of the extra spaces on all lines. Give a dynamic programming algorithm for this problem that runs in $O(n^2)$.

Solution. This is essentially Problem 4 ("Line Fill") from HW3 with a slightly modified objective: minimize the sum of extra spaces (instead of the sum of squares) on all lines (including the last line in this variant). This modification does not change the solution. The DP solution (i.e., the subproblems and the recurrence) is the same as in the HW problem (only need to remove the squares). See the HW solution for the details.

Name:	PSU Access ID:

Dynamic Programming: patterns *(20 points)*

You are given a $10 \times n$ grid that has an integer value written in each square. You are also given a set of $10n$ rocks, and you want to place some of these on the grid squares (each rock can be placed on exactly one square) so as to maximize the sum of the integers that are covered by rocks. There is one constraint: for a placement of rocks to be legal, no three of them can be placed in adjacent squares forming any of the following patterns:



Give an $O(n)$-time dynamic programming algorithm for computing the maximum sum.

Solution. A very similar problem was on of the DP examples in lecture, the only difference being that the notion of compatibility between rock placements in adjecent columns changes; i.e., it needs to be modified according to the new given forbidden patterns. In particular, you define $f(i, p)$ to be tthe total sum of the best rock placement in columns 1 to i that ends with the p-th pattern. Note that there are at most $2^10 = O(1)$ patterns. To compute $f(i, p)$ you can do it by notting that:

$$f(i, p) = \max_{p' \sim p} f(i-1, p') + \text{ value}(p)$$

where $\sim$ denotes compatibility. Check the lecture videos for the details.

Name:	PSU Access ID:

Half-Hamiltonian (*20 points)*

Recall that the HAMILTONIAN PATH problem: given a graph $G = (V, E)$, find a path with no repeated vertices of length $|V| - 1$. This problem is NP-complete.

The HALF-HAMILTONIAN PATH problem is the following: Given a graph $G = (V, E)$, find a path with no repeated vertices of length $\left\lfloor \frac{|V|}{2} \right\rfloor$ (number of edges). Show that HALF-HAMILTONIAN PATH is NP-complete.

Solution. Checking that the problem is in NP is trivial; most of you got this part. For the reduction, you need to show how to solve Hamiltonian Path using a half-Hamiltonian Path black box. I saw two correct solutions:

- Simply create a clone G' of G (with no edges in between) and feed the graph $G' \cup G$ to the half-Hamiltonian Path black box.
- Add $|V|$ isolated vertices to G and feed the resulting graph into the half-Hamiltonian Path black box.

A common mistake was assuming that something about the HP of G was known. Another reduction attempted was to join the clones or the isolated vertices to G with edges; this can create issues since the new edges could be used to create half HP in some border cases. Another attempt was to split edges. This does not work either. Take a graph that has a long path with and additional vertex connected to the path. It does not have an HP but it would have a half HP after the splitting.

Name:	PSU Access ID:

Extra space.

Name:	PSU Access ID:

Extra space.