Question 1: From each clause according to its ability to be satisfied. ......................... 10 points

A disjunctive normal form formula is the converse of CNF; i.e., it is an or of a number of clauses where each clause is an and of some terms. E.g.:

\[(x \wedge y \wedge z) \lor (z \wedge \overline{y} \wedge w) \lor (x \wedge \overline{z})\]

DNF-Sat is the analog problem of CNF-Sat: given a DNF formula \(f\) determine if there is a satisfying assignment of the corresponding variables that renders the formula true.

(a) (4) Design and analyze an efficient algorithm for DNF-Sat.

(b) (4) Demonstrate a reduction from 3Sat to DNF-Sat and analyze its runtime. (Hint: use the distributive law.)

(c) (2) Why does this not prove that \(P = NP\)?

Question 2: Solomonic decision problem ................................................................. 10 points

(a) (3) Suppose you are given an algorithm partitionable, given a set \(X\) of positive integers, determines whether \(X\) can be partitioned into two sets \(A\) and \(B\) such that \(\sum A = \sum B\). (partitionable returns True or False). Design an analyze an “efficient” (see below) algorithm that computes such a partition if it exists. In other words, on an input \(X\) you should return two sets \(A\) and \(B\) such that \(X = A \cup B\), \(A \cap B = \emptyset\), and \(\sum A = \sum B\), or return an error if such partition does not exist.

Your algorithm should call partitionable as a subroutine; its efficiency will therefore depend on the efficiency of partitionable. You should analyze both the amount of work that is done within your algorithm, and the number of calls to partitionable that are made, expressing both in asymptotic terms (e.g., “The algorithm performs \(\Theta(N^2)\) work and makes \(\Theta(N)\) calls to partitionable”). Your algorithm should run in polynomial time under the assumption that partitionable runs in polynomial time.

(b) (3) Design and analyze an efficient algorithm for 2Partition: given a set \(X\) of \(2n\) integers, partition them into \(n\) disjoint pairs such that the sum of each pair is equal. I.e., given \(X\) with \(|X| = 2n\), create sets \(S_1, \ldots, S_n\) such that \(|S_i| = 2\), \(S_i \cap S_j = \emptyset\) for \(i \neq j\), \(X = \bigcup_{i=1}^{n} S_i\) and \(\sum S_i = \sum S_j\) for any \(1 \leq i, j \leq n\).

(c) (4) Show that the problem 7Partition is NP-Hard. 7Partition is defined as above, taking a set of \(7n\) integers and splitting them into \(n\) disjoint sets of size 7 with the sum of each set being equal. You may assume that 3Partition is NP-hard.

Question 3: Charon’s crossing ................................................................. 10 points

Solve problem 42 in Chapter 12 of the textbook. Below is a restatement:

You are given a graph \(G = (V, E)\) and the number \(k\). The problem can be described as defining a sequence of subsets of vertices \(X_0, \ldots, X_{2m+1}\) with the following properties:

- \(X_0 = V, X_{2m+1} = \emptyset\)
- \(X_i\) and \(X_{i+1}\) differ by at most \(k\) vertices.
- \(X_{2i+1}\) is an independent set (in \(G\)) for all \(i = 0, \ldots, m\)
- \(V - X_{2i}\) is an independent set (in \(G\)) for all \(i = 0, \ldots, m\)
Less formally, you can think about having two sets $X$ and $Y$. All vertices start in $X$ and at each odd “move” you shift up to $k$ vertices from $X$ to $Y$, at each odd “move” you shift up to $k$ vertices from $Y$ to $X$. After every odd move, $X$ cannot have two vertices that are connected by an edge; after every even move $Y (= V - X)$ cannot contain two vertices that are connected by an edge.

The decision problem **CHARON** is: given a graph $G$ and a number $k$, is there a sequence of valid moves that ends with $X = \emptyset$? Your job is to show that **CHARON** is NP-hard.

As an example, for the classical formulation where $V = \{\text{Goat, Wolf, Cabbage}\}$, $E = \{\text{Goat} - \text{Cabbage}, \text{Wolf} - \text{Goat}\}$, and $k = 2$ we have a solution where:

<table>
<thead>
<tr>
<th>Move</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${\text{Goat, Wolf, Cabbage}}$ initial configuration</td>
</tr>
<tr>
<td>1</td>
<td>${\text{Wolf, Cabbage}}$ move Goat</td>
</tr>
<tr>
<td>2</td>
<td>${\text{Wolf, Cabbage}}$ empty move</td>
</tr>
<tr>
<td>3</td>
<td>${\text{Cabbage}}$ move Wolf</td>
</tr>
<tr>
<td>4</td>
<td>${\text{Goat, Cabbage}}$ move Goat</td>
</tr>
<tr>
<td>5</td>
<td>$\emptyset$ move Goat, Cabbage, final configuration</td>
</tr>
</tbody>
</table>

**Solved Problem**

**Question 5**: Encore

A double-Hamiltonian tour in an undirected graph $G$ is a closed walk that visits every vertex in $G$ exactly twice. Prove that it is NP-hard to decide whether a given graph $G$ has a double-Hamiltonian tour.

![Graph](image)

This graph contains the double-Hamiltonian tour $a \rightarrow b \rightarrow d \rightarrow e \rightarrow b \rightarrow d \rightarrow c \rightarrow f \rightarrow a \rightarrow c \rightarrow f \rightarrow g \rightarrow e \rightarrow a$.

**Solution**: We prove the problem is NP-hard with a reduction from the standard Hamiltonian cycle problem. Let $G$ be an arbitrary undirected graph. We construct a new graph $H$ by attaching a small gadget to every vertex of $G$. Specifically, for each vertex $v$, we add two vertices $v^\uparrow$ and $v^\downarrow$, along with three edges $vv^\uparrow$, $vv^\downarrow$, and $v^\uparrow v^\downarrow$.

I claim that $G$ has a Hamiltonian cycle if and only if $H$ has a double-Hamiltonian tour.

$\implies$ Suppose $G$ has a Hamiltonian cycle $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n \rightarrow v_1$. We can construct a double-Hamiltonian tour of $H$ by replacing each vertex $v_i$ with the following walk:

$$\cdots \rightarrow v_i \rightarrow v_i^\uparrow \rightarrow v_i^\downarrow \rightarrow v_i^\uparrow \rightarrow v_i^\downarrow \rightarrow v_i \rightarrow \cdots$$
A vertex in $G$, and the corresponding vertex gadget in $H$.

$\Leftarrow \Rightarrow$ Conversely, suppose $H$ has a double-Hamiltonian tour $D$. Consider any vertex $v$ in the original graph $G$; the tour $D$ must visit $v$ exactly twice. Those two visits split $D$ into two closed walks, each of which visits $v$ exactly once. Any walk from $v^\flat$ or $v^\sharp$ to any other vertex in $H$ must pass through $v$. Thus, one of the two closed walks visits only the vertices $v$, $v^\flat$, and $v^\sharp$. Thus, if we simply remove the vertices in $H \setminus G$ from $D$, we obtain a closed walk in $G$ that visits every vertex in $G$ once.

Given any graph $G$, we can clearly construct the corresponding graph $H$ in polynomial time.

With more effort, we can construct a graph $H$ that contains a double-Hamiltonian tour that traverses each edge of $H$ at most once if and only if $G$ contains a Hamiltonian cycle. For each vertex $v$ in $G$ we attach a more complex gadget containing five vertices and eleven edges, as shown on the next page.

A vertex in $G$, and the corresponding modified vertex gadget in $H$.

Common incorrect solution (self-loops): We attempt to prove the problem is NP-hard with a reduction from the Hamiltonian cycle problem. Let $G$ be an arbitrary undirected graph. We construct a new graph $H$ by attaching a self-loop every vertex of $G$. Given any graph $G$, we can clearly construct the corresponding graph $H$ in polynomial time.

Suppose $G$ has a Hamiltonian cycle $v_1 \to v_2 \to \cdots v_n \to v_1$. We can construct a double-Hamiltonian tour of $H$ by alternating between edges of the Hamiltonian cycle and self-loops:

$$v_1 \to v_1 \to v_2 \to v_2 \to v_3 \to \cdots v_n \to v_n \to v_1.$$

On the other hand, if $H$ has a double-Hamiltonian tour, we cannot conclude that $G$ has a Hamiltonian cycle, because we cannot guarantee that a double-Hamiltonian tour in $H$ uses any self-loops. The graph $G$ shown below is a counterexample; it has a double-Hamiltonian tour (even before adding self-loops) but no Hamiltonian cycle.

$\Diamond$
This graph has a double-Hamiltonian tour.

**Rubric (for all polynomial-time reductions):** 10 points =

+ 3 points for the reduction itself
  
  – For an NP-hardness proof, the reduction must be from a known NP-hard problem. You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course).

+ 3 points for the “if” proof of correctness

+ 3 points for the “only if” proof of correctness

+ 1 point for writing “polynomial time”

• An incorrect polynomial-time reduction that still satisfies half of the correctness proof is worth at most 4/10.

• A reduction in the wrong direction is worth 0/10.