Problem 1

In a graph, \( G = (V, E) \), two nodes are allowed to marry if they are not connected by an edge, this marriage joins the two nodes into one new node with edges from both nodes.

Specifically, given \( u, v \in V \) and \( (u, v) \notin E \), \( \text{marry}(G, u, v) \) creates a new graph \( G' = (V', E') \), where \( V' = V \setminus \{u, v\} \cup \{w\} \) and \( E' = E \setminus \{(u, x_i) \mid \forall x_i \in V\} \cup \{(v, x_i) \mid \forall x_i \in V\} \cup \{(w, z_i) \mid \forall z_i \in \text{neighborhood}(u) \cup \text{neighborhood}(v)\} \).

The following figure shows two consecutive marriages in a graph.

![Marriage Graph]

GraphMarriageEquality =
\{\langle G, H \rangle \mid \exists \text{ a series of marriages which transform } G \text{ into } G', \text{ where } G' \text{ is isomorphic to } H\}\}

Prove that GraphMarriageEquality is NP-complete.

Hint: You can do a reduction from 3-COLOR.

Proof idea

We construct an \( G'' \) from a graph \( G \) by adding a 3-clique disconnected from the rest of the graph to ensure that \( G'' \) contains a 3-clique. We let \( H \) be a 3-clique.

If \( G \) is 3-colorable, then \( G'' \) is 3-colorable. By marrying all the nodes of the same color in \( G'' \), we arrive at \( H \). Therefore, \( \langle G'', H \rangle \in \text{GraphMarriageEquality} \).

If \( G \) is not 3-colorable, then neither is \( G'' \). For any graph \( I \) that can be married to a 3-clique, \( I \) is 3-colorable. Therefore, \( \langle G'', H \rangle \notin \text{GraphMarriageEquality} \).
Proof. GraphMarriageEquality $\in$ NP. The following algorithm accepts in polynomial time if $\langle G, H \rangle \in$ GraphMarriageEquality:

i. Non-deterministically select a series of marriages.
ii. Execute the marriages to construct $G'$.
iii. Non-deterministically select a mapping from $G'$ to $H$ to check isomorphism.
iv. If $G'$ is isomorphic to $H$, ACCEPT.

For arbitrary graph $G = \langle V, E \rangle$, define the reduction from 3-COLOR to GraphMarriageEquality as follows:

$$f(\langle G \rangle) = \langle G'', H \rangle$$

where $G'' = \langle V', E' \rangle$ such that $V' = V \cup \{a, b, c\}$ for $a, b, c \notin V$, and $E' = E \cup \{\langle a, b \rangle, \langle b, c \rangle, \langle a, c \rangle\}$. $H = \langle \{v_1, v_2, v_3\}, \{(v_1, v_2), (v_1, v_3), (v_2, v_3)\} \rangle$. So $G''$ is the same as $G$ with the exception that three vertices have been added to form a 3-clique subgraph, and $H$ is a 3-clique.

Since $f$ merely adds three vertices to $V$, three edges to $E$, and appends $H$, it is computable in linear time $O(n)$.

Suppose $\langle G \rangle \in$ 3-COLOR. Then $G'' \in$ 3-COLOR. Take a 3-coloring of $G''$ which uses red, green, and blue. By marrying every vertex of the same color, eventually a 3-clique is reached.

Given that $G''$ contains a 3-clique, and given that no vertices of the same color are connected by an edge, marrying all vertices of the same color must result in a 3-clique. Suppose, without loss of generality, that we married all but the three vertices within the 3-clique subgraph of $G''$. Then, by marrying the two remaining red vertices, there would be a red vertex which is connected to at least one green and at least one blue. Since the same holds for marriages of the green and blue vertices, we are left with three vertices, each one connected to two others, or, a 3-clique.

Therefore, there is a series of marriages which transform $G''$ into a $G'$ which is isomorphic to $H$, so $\langle G'', H \rangle \in$ GraphMarriageEquality.

Suppose $f(\langle G \rangle) = \langle G'', H \rangle \in$ GraphMarriageEquality. Then there is some series of marriages which transforms $G''$ into $H$, where $G''$ is a graph which contains a 3-clique subgraph and $H$ is a 3-clique. Let all those vertices that converge into the red vertex of $G'$ be colored red, those that converge to the blue vertex colored blue, and those that vertex to the green colored green. Since these vertices are all married, none of them are connected by an edge, so this is a valid 3-coloring.

Since $G''$ is 3-colorable, so is $G$. Therefore, $\langle G \rangle \in$ 3-COLOR.

\[\square\]
Problem 2

The bin packing problem is given a set of \( n \) items, each with a weight, \( w_i \), and set of bins, \( B \), each bin, has a fixed capacity, \( c \), find the minimum number of bins needed to pack all items.

Formally rephrase this as a decision problem, and specify it as a set. Prove that this problem is NP-complete. Do a reduction from SUBSET – SUM.

Proof idea

We define sets BinPack and Partition. Partition is the set of sets which can be divided into two pieces such that the elements of each piece sum to the same number.

By adding values to SUBSET-SUM, we guarantee that it sums to \( 2t \), and is in Partition if and only if there is a subset of the original \( S \) which sums to \( t \). We then reduce Partition to BinPack, which is done by setting the values as weights, pairing each with a natural number. We then simulate the Partition problem on a set \( A \) as a bin packing problem by setting the number of bins to two and setting their capacity to one half the sum of all elements of \( A \).

Proof. Let BinPack be defined

\[
\text{BinPack} = \{ \langle W, c, b \rangle \mid \text{there is a minimum of } b \text{ disjoint } I_x \text{ such that } I_1 \cup I_2 \cup \ldots \cup I_n = W \text{ and for each } I_x, \text{ for all } \langle i, w_i \rangle \in I_x, \sum w_i \leq c \}
\]

where each \( i \) is an item and each \( w_i \) is its weight.

BinPack \( \in \) NP. The following algorithm accepts in polynomial time if \( \langle W, c, b \rangle \in \text{BinPack} \).

i. Non-deterministically select \( b \) disjoint \( I \).
ii. Check that, for each \( I \), for all \( \langle i, w_i \rangle \in I, \sum w_i \leq c \).
iii. Use the same procedure to determine if \( W \) can be packed into \( b - 1 \) bins of size \( c \).

If so, REJECT. If not, ACCEPT.

Let Partition be defined

\[
\text{Partition} = \{ \langle A \rangle \mid \text{there are disjoint } B, C \text{ such that } B \cup C = A \text{ and } \sum_{b \in B} b = \sum_{c \in C} c \}
\]

SUBSET-SUM \( \leq^p \) Partition. Construct \( \langle A, u \rangle \) from \( \langle S, t \rangle \) as follows. Let \( n \) be the sum of all members of \( S \). Let \( A = S \cup \{|n - 2t|\} \) if \( 2t \neq n \), else let \( A = S \cup \{t\} \) (allowing for duplicates).

Define the reduction from SUBSET-SUM to Partition as follows:

\[
f(\langle S, t \rangle) = \langle A \rangle \quad [f \text{ is clearly } O(n^k)]
\]

Let \( a \) be the sum of all members of \( A \). This formulation gives us the following results:

\[
t = (1/2)a \quad \text{if } 2t \geq n \text{ or}
\]

\[
t = (1/2)a - n \quad \text{if } 2t < n
\]

Suppose \( \langle S, t \rangle \in \text{SUBSET-SUM} \). Then there is a subset \( x \) whose elements add up to \( t \) and a disjoint subset \( y \) whose elements add up to \(|n - t|\). If \( 2t < n \), then \( x + |n - 2t| \) and \( y \) each sum to
If $2t \geq n$, then $x$ and $y + |n - 2t|$ each sum to $(1/2)a$. In either case, $A$ can be partitioned into two equal parts, and $f(\langle S, t \rangle) = \langle A, u \rangle \in \text{Partition}$. Suppose $\langle S, t \rangle \not\in \text{SUBSET-SUM}$. Then there is no subset of $S$ whose elements add up to $t$. If $2t < n$, then there is no $x$ such that $x + |n - t| = (1/2)a$. If $2t \geq n$, then there is no $x$ such that $x = (1/2)a$ (where $x$ does not include $|n - t|$). In either case, there is no partition on $S \cup \{|n - 2t|\}$ such that the elements of both sides sum to $t$. Therefore, $f(\langle S, t \rangle) = \langle A \rangle \not\in \text{Partition}$.

Partition $\leq^p_n$ BinPack. Construct $\langle W, c, b \rangle$ from $\langle A \rangle$ as follows. Enumerate each element $a_i$ of $A$ and pair each element with its corresponding integer:

$\langle 1, a_1 \rangle, \langle 2, a_2 \rangle, \ldots, \langle m, a_m \rangle$

Let $d = \sum_{x \in A} x, c = \lfloor d/2 \rfloor$, and $b = 2$. Define the reduction from Partition to Binpack:

$g(\langle A \rangle) = \langle W, c, b \rangle$

Since $g$ simply expands $\langle A \rangle$ and appends some extra information, it runs in $O(n^2)$ time.

Suppose $\langle A \rangle \in \text{Partition}$. Then there is a way to partition $A$ into two disjoint sets $B$ and $C$ such that $B \cup C = A$ and the sum of all elements in $B$ is equal to the sum in $C$, and each sum is equal to $d/2$. So, given the values as weights, there is a way to pack them into two bins of size $\lfloor d/2 \rfloor$. Therefore, $g(\langle A \rangle) \in \text{BinPack}$.

Suppose $\langle A \rangle \not\in \text{Partition}$. Then there is no way to partition $A$ into two disjoint sets $B$ and $C$ such that $B \cup C = A$ and the sum of all elements in $B$ is equal to the sum in $C$, and each sum is equal to $d/2$. So, given a bin size of $\lfloor d/2 \rfloor$, the weights of any $X \subseteq W$ will sum to $p$ such that $p > \lfloor d/2 \rfloor$ or $p < \lfloor d/2 \rfloor$. If $p > \lfloor d/2 \rfloor$, then $X$ cannot be packed into a single bin. If $p < \lfloor d/2 \rfloor$, then $W - x$ cannot be packed into a single bin. Therefore, $g(\langle A \rangle) \not\in \text{BinPack}$. □
Problem 3

One strategy for solving the bin packing problem is to do a first fit heuristic. Here you pack the items one at a time into the first bin that the item will fit into. If it doesn’t fit into any of the current bins, you start a new bin.

Formally state this as an algorithm and show that it never uses more than 2 times the number of bins in the optimal solution.

Formal algorithm

Given $W$ and $c$:

i. $i := 1$

ii. $\phi = I_1 = I_2 = \ldots$

(a) For all $\langle y, w_y \rangle \in W$

i. For $j := 0$ to $j = i$

A. If, for all $\langle x, w_x \rangle \in I_j$, $\sum w_x + w_y \leq c$, then $I_j := I_j \cup \langle y, w_y \rangle$ and break.

B. Else, if $j = i$, increment $i$.

C. Increment $j$.

Proof idea

If the algorithm used more than twice the number of bins used in the optimal solution, it would have to allow for multiple bins to be less than half full. Since the algorithm does not allow this to happen, it can never exceed twice the amount.

Proof. Let $b$ be the number of bins in the optimal solution. Suppose the algorithm uses $2b + 1$ bins. Let $a$ be the average of all of the values held in all $2b + 1$ bins. Since all $\langle i, w_i \rangle \in W$ fit into $b$ bins of size $c$ in the optimal solution, $a < c/2$.

Suppose that there are two bins $I_j$ and $I_k$ whose elements sum to less than or equal to than $c/2$. Suppose that $j < k$ (it doesn’t matter either way). Then, by definition of the algorithm, the elements of $I_k$ would have been stored in $I_j$ rather than in $I_k$. Therefore, there is only one bin, say $I_j$, whose elements sum to less than or equal to $c/2$.

The remaining $2b$ bins all sum to some value greater than $c/2$. Since each sum is greater than $c/2$, the total of all $2b$ sums is greater than $2b(c/2)$: for all $I_z$ and all $\langle i, w_i \rangle \in I_z$,

$$\sum w_i > 2b(c/2)$$

$$> (2bc/2)$$

$$> bc$$

Given that all $\langle i, w_i \rangle \in W$ fit into $b$ bins of size $c$ in the optimal solution, $\sum_{(i, w_i) \in W} w_i \leq bc$.

Therefore, the algorithm never uses more than $2b$ bins. □
Problem 4

In certain programming languages, comments appear between delimiters such as /# and #/. Let $L$ be the language of all valid delimited comment strings. A member of $L$ must begin with /# and end with #/ but have no intervening #/. For simplicity, we’ll say that the alphabet is $\{a, b, /, #\}$. Give a DFA that recognizes $L$.

Note: You may hand draw this problem and scan it to submit. Please write neatly!

*Any missing arrows are supposed to transition to state 6.*
Problem 5

Prove that for every \( k > 1 \), a language \( A_k \subseteq \{0, 1\}^* \) exists that can be recognized by a DFA with \( k \) states, but not by one with \( (k - 1) \) states.

Proof idea

We make \( A_i \) languages of strings with fixed length. We then construct \( A_{k+1} \) from \( A_k \) by adding a node and transitions into \( A_k \)'s start state as shown:

\[
\begin{array}{ccc}
  a & 0,1 & b \\
  \end{array}
\]

We prove by induction that, for any \( A_n \) no machine with fewer than \( n \) nodes can recognize \( A_n \).

Proof. Let the series of languages be defined:

\[
A_k = \underbrace{\{0 \cup 1\} \circ (0 \cup 1) \circ \ldots \circ (0 \cup 1)}_{k-2} \text{ such that }
\]

\[
A_2 = \{\epsilon\}
\]
\[
A_3 = \{(0 \cup 1)\}
\]
\[
A_4 = \{(0 \cup 1)(0 \cup 1)\}
\]
\[
A_5 = \{(0 \cup 1)(0 \cup 1)(0 \cup 1)\}
\]
\[
\vdots
\]

\[
A_2 \text{ can be recognized by a two state DFA: }
\]

\[
\begin{array}{ccc}
  1 & 0,1 & 2 \\
  \end{array}
\]

Suppose \( A_2 \) could be recognized by a DFA \( M \) with one state, \( a \). Since there is at least one string in \( A_2 \), that state would have to be an accept state. Since \( M \) is deterministic, it must have both 0- and 1-transitions which must loop back to \( a \). Therefore, \( M \) accepts everything and does not recognize \( A_2 \).

Suppose \( A_k \) can be recognized by a DFA with \( k \) states but not one with \( k - 1 \) states. By construction,

\[
A_{k+1} = \underbrace{\{0 \cup 1\} \circ (0 \cup 1)(0 \cup 1) \ldots (0 \cup 1)}_{k-2} \text{ or } \{(0 \cup 1) \circ s\} \text{ for } s \in A_k.
\]

Construct machine \( M_{k+1} \) to recognize \( A_{k+1} \).

The first transition of \( M_{k+1} \) must transition from an initial state \( i \) to some other state \( j_1 \) on input ‘0’ or ‘1’. If the initial state had self loops, then \( M_k \) would accept any string of ‘0’s and ‘1’s followed by \( s \). \( M_k \) must then recognize \( s \) using states \( j_1, j_2, j_3, \ldots \) and \( j_n \).

By the inductive hypothesis, the smallest machine \( M_k \) to recognize \( s \) has \( k \) states. If \( n < k \), then we could construct a machine to recognize \( A_k \) with fewer than \( k \) states- \( M_{k+1} \) without its initial state. \( i \) cannot be included in \( j_1 \ldots j_n \). If it were, there would be a loop such that the machine would accept indefinitely long strings, some of which are not in \( A_{k+1} \).) So, \( n = k \).

Therefore, \( M_{k+1} \) has \( k + 1 \) states.