Assignment 3, CS 6363  
due at the beginning of Lecture on 10/17  
no late homework would be accepted

Homework consists of Problems 1-7.

1 (Exercise 23.1-6) Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not true by giving a counterexample.

Solution For contradiction, suppose there are two distinct minimum spanning trees $T_1^*$ and $T_2^*$. Consider an edge $(u, v)$ in $T_1^* \setminus T_2^*$. Deleting $(u, v)$ from $T_1^*$ would break $T_1^*$ into two parts $T_{11}^*$ and $T_{12}^*$ containing $u$ and $v$, respectively. Now, we consider cut $(V(T_{11}^*), V(T_{12}^*))$ where $V(T)$ denotes the vertex set of $T$. Let $p(u, v)$ be the path connecting $u$ and $v$ in $T_2^*$. $p(u, v)$ must contain an edge $e$ in cut $(V(T_{11}^*), V(T_{12}^*))$. By Cut-Optimality Condition, $(u, v)$ is a light edge. Since every cut has unique light edge, we have weight$(u, v) <$ weight$(e)$. This means that tree $(T_2^* \setminus e) \cup (u, v)$ has weight smaller than $T_2^*$, contradicting the minimality of $T_2^*$.

Figure 1: A graph have unique minimum spanning tree.

The graph in Fig. 1 has unique minimum spanning tree consisting of three edges with weight one, but cut $\{A, D\}, \{B, c\}$ has two light edges $(A, B)$ and $(D, C)$. \hfill \Box
2 (Exercise 23.2-8) Professor Toole proposes a new divide-and-conquer algorithm for computing minimum spanning tree, which goes as follows. Given a graph \( G = (V, E) \), partition the set \( V \) of vertices into two sets \( V_1 \) and \( V_2 \) such that \( |V_1| \) and \( |V_2| \) differ by at most 1. Let \( E_1 \) be the set of edges that are incident only on vertices in \( V_1 \), and let \( E_2 \) be the set of edges incident only on vertices in \( V_2 \). Recursively solve a minimum-spanning-tree problem on each of the two subgraphs \( G_1(V_1, E_1) \) and \( G_2 = (V_2, E_2) \). Finally, select the minimum-weight edges in \( E \) that crosses the cut \( (V_1, V_2) \), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of \( G \), or provide an example for which the algorithm fails.

**Solution** The algorithm does not work. Consider the example in Fig. 1 and partition \((\{A, D\}, \{B, C\})\).

We first obtain two edges \((A, D)\) and \((B, C)\) and then connected them by either edge \((A, B)\) or \((D, C)\), which is not a minimum spanning tree. \(\square\)

3 (Exercise 16.4-3) Show that if \((S, I)\) is a matroid, then \((S, I')\) is a matroid, where

\[
I' = \{A' \mid S - A' \text{ contains some maximal } A \in I\}.
\]

That is, the maximal independent sets of \((S, I)\) are just complements of the maximal independent sets of \((S, I)\).

**Solution** To show that \((S, I')\) is a matroid, it suffices to prove the following:

1. If \(B' \subseteq A' \in I'\), then \(B' \in I'\).
2. If \(A', B' \in I'\) with \(|A'| > |B'|\), then there exists \(x \in A' \setminus B'\) such that \(B' \cup \{x\} \in I'\).

First, we show (1). For \(A' \in I'\), there exists maximal independent A \(\in I\) such that \(A \subseteq S - A'\). If \(B' \subseteq A'\), then \(A \subseteq S - A' \subseteq S - B'\) and hence \(B' \in I'\).

Next, we show (2). Suppose \(A\) and \(B\) are two maximal independent subsets in \(I\) such that \(A \subseteq S - A'\) and \(B \subseteq S - B'\). If \((A' \setminus B') \setminus B \neq \emptyset\), then choose \(x \in (A' \setminus B') \setminus B\) and we would have \(B' \cup \{x\} \subseteq S - B\), hence \(B' \cup \{x\} \in I'\). Now, suppose \((A' \setminus B') \setminus B = \emptyset\), i.e.,
\( (A' \setminus B') \subseteq B \). Choose \( x \in A' \setminus B' \). Since \( |A| = |B| > |B - \{x\}| \), there exists \( y \in A \setminus (B - \{x\}) \) such that \( (B - \{x\}) \cup \{y\} \in \mathcal{I} \). This means that \( (B - \{x\}) \cup \{y\} \) is a maximal independent set in \( \mathcal{I} \). If \( y \not\in B' \), then \( B' \cup \{x\} \subseteq S - ((B - \{x\}) \cup \{y\}) \). Thus, \( B' \cup \{x\} \in \mathcal{I}' \). We next show the existence of \( x \) such that \( y \) can be chosen to satisfy \( y \not\in B' \).

To do so, suppose \( |A' \setminus B'| = k \). Then \( |A| - |B - (A' \setminus B')| = k \). Choose a subset \( C \) of \( A \) with \( |C| = k \) such that \( (B - (A' \setminus B')) \cup C \in \mathcal{I} \). Since \( |B' \setminus A'| < |A' \setminus B'| = k \), \( C \) must contain an element \( y \not\in B' \). Now, we move elements from \( B \) to \( B - (A' \setminus B') \) \( \cup \{y\} \). We can move back \( k - 1 \). Thus, one of elements in \( A' \setminus B' \) cannot move back, denoted it by \( x \). Then \( (B - \{x\}) \cup \{y\} \in \mathcal{I} \) and \( y \not\in B' \).

4 (Exercise 16.5-1) Solve the instance of the scheduling problem given in the table, but with each penalty \( w_i \) replaced by \( 80 - w_i \).

<table>
<thead>
<tr>
<th>( a_i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_i )</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>( w_i )</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

(Note: In this scheduling problem, each task \( a_i \) has unit-time and a deadline \( d_i \) with a penalty \( w_i \) if not meet the deadline. All feasible schedules form a matroid.)

**Solution** First, sort tasks in non-increasing-weight ordering.

<table>
<thead>
<tr>
<th>( a_i )</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_i )</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>80 - ( w_i )</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Since all feasible solutions form a matroid, we can use greedy algorithm to find the maximum solution according to the above ordering. The result is \{task7,task6,task5,task4,task3\}.

Tasks 2 and 1 cannot put in because at time 4, we cannot finish more than 4 tasks with deadline before or up to time 4.

5 (Exercise 16.1-3) Suppose that we have a set of activities to schedule among a large number of lecture halls. We wish to schedule all the activities using as few lecture halls as possible.
Give an efficient greedy algorithm to determine which activity should use which lecture hall.
(Note: Each activity can be represented by an interval $[s_i, f_i]$.)

**Solution** Sort all activities according to the ending time $f_i$ in non-decreasing ordering, $f_1 \leq f_2 \leq \cdots \leq f_n$. Also, give lecture hall an arbitrary ordering, $H_1, H_2, \ldots$. Now, we can make a schedule with the following greedy algorithm:

```plaintext
for $i = 1$ to $n$ do begin
  $ok \leftarrow 0$;
  $j \leftarrow 1$;
  $m \leftarrow 1$;
  while $ok = 0$ do
    if activity $i$ can be scheduled in hall $H_j$
      then assign$(i) \leftarrow j$ and $ok \leftarrow 1$
    else $j \leftarrow j + 1$;
    $m \leftarrow \max(m, j)$; end-for
output $m$ and assign$(i)$ for $i = 1, 2, \ldots, n$.
```

This algorithm gives an optimal solution. To see this, suppose $[s_i, f_i]$ is an activity scheduled to lecture hall $H_m$. Then we must have the following:

(a) $[s_{i-k}, f_{i-k})$ must be scheduled to lecture hall $H_{m-k}$ for $k = 1, 2, \ldots, m - 1$.
(b) For $0 \leq k < k' \leq m - 1$, $s_{i-k} < f_{i-k} \leq f_{i-k'}$.

From (a) and (b), we can prove by induction on $m$ that those $m$ activities require $m$ lecture halls. (The reader may fill in detail.)

6 (Exercise 16.3-4) Prove that if we order the characters in an alphabet so that their frequencies are monotonically decreasing, then there exists an optimal code whose codeword lengths are monotonically increasing.

**Solution** For contradiction, suppose there exists an optimal code whose codeword lengths are not monotonically increasing, that is, for two characters $a$ and $b$, their frequencies $f_a > f_b$, but
their codeword lengths $l_a > l_b$. We exchange their codes, that is, use code of $a$ for $b$ and code of $b$ for $a$. Then the cost will be changed in value

$$(f_a l_a + f_b l_a) - (f_a l_a + f_b l_b) = -(f_a - f_b)(l_a - l_b) < 0.$$ 

This means that the cost will be decreased, contradicting the optimality of the code. \qed

7 Let $V$ be a fixed set of $n$ vertices. Consider a sequence of $m$ undirected edges $e_1, e_2, ..., e_m$. For $1 \leq i \leq m$, let $G_i$ denote the graph with vertex set $V$ and edge set $E_i = \{e_1, ..., e_i\}$. Let $c_i$ denote the number of connected components of $G_i$. Design an algorithm to compute $c_i$ for all $i$. Your algorithm should be asymptotically as fast as possible. What is the running time of your algorithm?

**Solution** The idea is as follows: Initially, put each vertex $v_i$ into basket $i$ and set $c_0 = n$. For $j = 1, 2, ..., m$, suppose edge $e_j = (u, v)$. If $u$ and $v$ are in two different buses, then $c_j := c_{j-1} - 1$ and merge those two buses into one. Otherwise, $c_j := c_{j-1}$.

The following is the detail:

```algorithm
for i = 1 to n do A[i] := 1;
c_0 := n;
for j = 1 to m do
  if $e_j = (u, v)$ and $A[u] \neq A[v]$
    then $c_j := c_{j-1} - 1$ and
      for i = 1 to n do
          then $A[i] := A[u]$
        else $c_j := c_{j-1}$;
    output $c_j$ for $j = 1, 2, ..., m$.
```

The running time of this algorithm is $O(mn)$. \qed