Advanced Algorithm: Homework 6
Solutions

Problem 1. 35.1-3(p.1027 in CLRS). This is the problem 37.1-2 in old edition.
(Hint: try a bipartite graph with vertices of uniform degree on the left and vertices of varying
degree on the right.)

Solution: How about the following example?

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
</tbody>
</table>
```

The minimum vertex cover is \{g, h, i, j, k, l\}, while Professor’s algorithm may yield a solution of
\{m, n, o, p, q, r, s, g, h, i, j, k, l\}. The ratio of 13/6 does violate the bound of 2.

Problem 2. 35.1-4(p.1027 in CLRS). This is the problem 37.1-3 in old edition.
Give an efficient algorithm that finds an optimal vertex cover for a tree in linear time.

Hints:
Suppose for the original tree \(T = <V, E>\). That is, the vertex set is \(V\), the edge set is \(E\). Also
suppose the cover set = \(C\). The algorithm can be described as follows:

1. Initialize the cover set \(C = \emptyset\)
2. while \(V \neq \emptyset\) do
3. Identify a leaf vertex \(v\)
4. Locate \(u = \text{parent}(v)\), the parent vertex of \(v\).
5. \(C = C \cup \{u\}\).
6. Remove all the edges incident to \(u\)
7. return \(C\).

A non-trivial question now is how to get leaf vertex efficiently to guarantee the above algorithm
is in linear time. Fortunately, the Breadth-First-Search (BFS) or Depth-First-Search (DFS)
The algorithm has its role in this. The BFS and DFS algorithm can be utilized to serialize tree vertex into an array. Then we could make use of this array effectively and efficiently to guarantee the linear time (I believe you can figure out how?).

**Problem 3. 35.2-3 (P.1033 in CLRS). This is the problem 37.2-2 in old edition.**

**Hints:**
The idea is as follows: First we have to observe that the sequence of vertices added by this algorithm is identical to the sequence of vertices added by Prim’s MST algorithm. That is, this algorithm is essentially similar to the previously 2-approximation algorithm for “-TSP. In other words, the process of closest-point heuristic adding vertices into the cycle is same as PRIM algorithm finding MST.

The following we will try to use induction to prove $C(H_k) \leq 2 \times C(MST_k)$.

Suppose $k$ is the number of vertices of current cycle or MST.

1. If $k=1$, then we got a trivial case and it is easy to verify;
2. Suppose for $k$, we have $C(H_k) \leq 2 \times C(MST_k)$. Now let us show it also applies to $k+1$. For $k+1$. Suppose $u$ is the closest vertex to $V_k$, $(u, v)$ is the shortest distance. So, $C(MST_{k+1}) = C(MST_k) + w(u, v)$.

Suppose $w$ follows $v$ in $H_k$. We now need to remove $(v, w)$ and insert $(v, u)$ and $(u, w)$.

$$C(H_{k+1}) = C(H_k) + w(v, u) + w(u, w) - w(v, w).$$

Based on triangle inequity, we have:

$$W(v, u) + w(v, w) > w(u, w).$$

That is:

$$w(u, w) - w(v, u) < w(v, w)$$

Therefore: $C(H_{k+1}) < C(H_k) + 2 \times w(v, u)$. *Since* $C(H_k) \leq 2 \times C(MST_k)$, we have

$$C(H_{k+1}) < C(H_k) + 2 \times w(v, u) = 2 \times C(MST_k) + 2 \times w(v, u) = 2 \times C(MST_{k+1}).$$

This completes our proof.

**Problem 4. 35.3-5 (P.1038 in CLRS).**

`GREDDY-SET-COVER` can return a number of different solutions, depending on how we break ties in line 4. Give a procedure `BAD-SET-COVER-INSTANCE(n)` that returns an $n$-element instance of the set-covering problem for which, depending on how ties are broken in line 4, `GREDDY-SET-COVER` can return number of different solutions that is exponential in $n$.

**Hints:**
Suppose $x[1..n]$ stores the $n$-elements and our algorithm return a family $F$ of subsets of $x[1..n]$. Initially, $F = \emptyset$ and $S = \emptyset$. The procedure can be given as follows:

BAD-SET-COVER-INSTANCE($n$)
1. $S \leftarrow \{x[1], x[2], \ldots, x[n]\}$
2. $F \leftarrow F \cup S \cup \{\{x[n]\}\}$
3. For $i \leftarrow 1$ to $n-2$ do
4.   Let $P$ denote a combination of the $i$ elements in set $S = \{x[1], x[2], \ldots, x[n]\}$
5.   For any possible $P$ do
6.      $P \leftarrow P \cup \{x[n]\}$
7.      $F \leftarrow F \cup \{P\}$
8. return $F$

Analysis: We now show that given the set generated by BAD-SET-COVER-INSTANCE($n$), GREEDY-SET-COVER returns at least $2^{n-1}$ different solutions depending on how ties are broken. We can show this by looking into how many sets would be generated in line 5 of the above procedure. For each $I$, we have $C_{n-1}^I$ sets. So in total we have

$$C_{n-1}^1 + C_{n-1}^2 + \ldots + C_{n-1}^{n-2} = 2^{n-1} - 2$$

sets.

This is obviously exponential in $n$.

Now let us try to find the set cover using GREEDY-SET-COVER algorithm. Obviously, the first set we will choose is $S$ which has $(n-1)$ elements and then only $x[n]$ is unselected. So the next time we should choose a set that includes element $x[n]$. Since all the remaining sets include $x[n]$, so choose any of them would be ok and it depends on how we break tie in line 4 of GREEDY-SET-COVER algorithm. That is, we at least have $O(2^n)$ different solutions.

**Problem 5. 3(P.368 in HSR).**

Generalize Hamiltonian so that it processes a graph whose edges have costs associated with them and finds a Hamiltonian cycle with minimum cost. You can assume that all edge costs are positive.

**Hints:**
We can add 3 variables in the code: $\text{min\_cost}$, which stores the minimum cost of the Hamiltonian cycle currently found; $\text{cost}$, which is the cost of the path currently expanding; $y[1..n]$, an array to save the Hamiltonian cycle we have currently found with minimum cost. At each step when the path is expanding, we keep checking whether expanding the node would make the cost of the path larger than the minimum cost we have found, then we should discard the expanding and backtrack to the upper level. Otherwise we continue expanding the path.

In the beginning, we have to initialize the adjacency matrix $G[i, j; i=1..n, j=1..n]$. $G[i, j]$ is the cost between $i, j$ if there is no edge between them then $G[i,j] = \infty$. We also have to set $x[1] = 1$ and $x[2], x[3], \ldots, x[n]$ as 0. The updated algorithm is given as follows:
Void Hamiltonian(int k) {
int y[1..n]
int min_cost = ∞, cost = 0;

do {
    NextValue(k);
    if(!x[k]) return;
    if(k==n) {
        cost = cost + G[x[k-1], x[k]] + G[x[k], x[1]];
        if (cost < min_cost) {
            min_cost = cost;
            for(i=1;i<=n;i++) y[i] = x[i];
        }
    } else {
        cost = cost + G[x[k-1],x[k]];
        if (cost >= min_cost) return;
        else Hamiltonian(k+1);
    }
} while(1);

if(k==2 && min_cost ≠ ∞) {
    for(I=1; I<=n; I++)
        cout << y[I] << ";
    cout << "minimum cost is" << min_cost;
}
}

void NextValue(int k) {
    do {
        x[k] = (x[k]+1) % (n+1);
        if(!x[k]) return;
        if(G[x[k-1]][x[k]] < ∞)
            for(int j=1;j<=k-1;j++)
                if(x[j]==x[k]) break;
        if(j==k)
            if((k<n) || ((k==n) && G[x[n],x[1]])) return;
    } while(1)
}