1. (7 pts)
Using adjacency matrix M

```c
for( i = 1; i <= |V|; i++)
    for( j = 1; j <= |V|; j++)
        for(k = 1; k <= |V|; k++)
            {
                if( i != j != k && M[i,j] == 1 && M[j,k] == 1 && M[k,i] == 1)
                    return TRUE;
            }
```

Complexity: nested loop will be executed in $O(|V|^3)$

2. (13 pts)
Using adjacency list

For every edge $e = (i, j)$, check the adjacency lists of vertex $i$ and vertex $j$ to see if the two lists have a common vertex $k$, i.e. vertex $k$ appears both in the adjacency lists of $i$ and $j$.

Implementation detail: We maintain a bit vector $BV$ of size $|V|$. When checking adjacency lists of vertex $i$, we set $BV[x] = 1$ if vertex $x$ appears in the list, $BV[x] = 0$ otherwise. Then we check adjacency lists of vertex $j$ to see if there is any vertex $y$ that $BV[y] = 1$, if so, $y$ is the common vertex of the two lists.

Complexity: for every edge, the processing time is at most $2*|V-1|$ as the max degree of any vertex is $|V-1|$, so the total time complexity is $O(|V|^3|E|)$

Common mistake:
DFS on $G$, when encountering a back edge, check if the cycle is of length 3.

counter example:

Assume the DFS order is 1->2->3->4->5, (5,2) and (5,1) are two back edges. The length of the two
cycles are 4 and 5, and you will miss the triangle formed by \{1, 2, 5\}

2.
Use Kruskal’s or Prim’s algorithm to find the MST M of G first. Then for a non-MST edge \((i, j)\) (edge of G which is not in M), find the path P from i to j in MST M. P and edge \((i, j)\) form a cycle. Easy to see that remove any edge in path P away from MST and add edge \((i, j)\) to MST will still result a spanning tree (not minimum). If we can change the cost of edge \((i, j)\) to some value smaller than any of the edges in path P, then the cost of MST will change. Traverse the path P to find the largest weighted edge \((x, y)\) and record the difference between edge \((i, j)\) and edge\((x, y)\):

\[
\text{min}_\text{cost} = W(i, j) - W(x, y)
\]

For every non-MST edge repeat the above procedure, and update min_cost if smaller difference between non-MST and MST edges found, also record the non-MST edge\((i, j)\) which cause the smallest difference.

When every non-MST edges are checked, the smallest change will be \(\text{min}_\text{cost}+1\) and the non-MST edge which can be changed is the recorded non-MST edge.

Common mistake:
Construct MST first, then for every vertex, calculate the smallest difference between its MST edge and non-MST edge.

Counter Example:

```
\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{counter_example}
\caption{Counter Example for minimum spanning tree}
\end{figure}
```

The smallest change should be \((4 - 3) + 1 = 2\) instead of \((4-2) + 1 = 3\)

3.
\textbf{a[]}: partial solution
\textbf{k}: current size of partial solution
\textbf{ITEM_SET}: set of n items
\textbf{K}: the size of subset which should be listed

\textbf{is_a_solution(int a[], int k, int K)}
{
    return (k == K);
}
```
construct_candidate(int a[], int k, int n, int c[], int * ncandidate)
{
    for(i = k; i <= n ; i++)
    {
        c[ i - k] = ITEM_SET[i];
    }
    ncandidate = n – k + 1;
}

Note:
The other functions are basically the same as those on the text book.
The important thing is when constructing candidate, the candidates should only be those items
that are larger than the largest in the current partial solution, otherwise redundancy will be
introduced in the final result.

4. Let P denotes the pattern string, T denotes the text string; D[i, j] denotes the edit distance
from T[1:j] to P[1:i].

\[
D[i, j] = \min \begin{cases} 
  D[i - 1, j] + 1 & \text{(insertion)} \\
  D[i, j - 1] + 1 & \text{(delete)} \\
  D[i - 1, j - 1] + [P[i] == T[j]] & \text{(substitution)} \\
  D[i - 2, j - 2] + 1, \text{if } P[i - 1] == T[j] \text{ and } P[i] == T[j - 1] & \text{(swap)}
\end{cases}
\]

where \([p[i] == T[j]] = 1\) if \(p[i] == T[j]\), otherwise 0.

Boundary conditions are:

\[
D[i, 0] = i, \text{ for } i=1, 2, \ldots \text{ length}(T) \\
D[0, j] = j, \text{ for } j=1, 2, \ldots \text{ length}(P) \\
D[0, 0] = 0
\]

5. (a) For 12 cents, when using greedy algorithm, 12 = 10 + 1 + 1, so we need 3 coins; But the
minimum number of coins is actually 2  \((12 = 6 + 6)\).

(b) Let \(C(n)\) denotes the minimum number of coins needed to make change of \(n\) units using
denominations \(\{d_1, \ldots, d_k\}\).

\[
C(n) = \min \{ C(n-d_1) + 1, C(n-d_2) + 1, \ldots, C(n-d_k) + 1 \}
\]

Where \(C(n-d_i) + 1\) means to use at least one coin of \(d_i\) denomination to make change of \(n\) units.

Boundary conditions are:

\(C(0) = 0, \text{ } C(m) = \text{INFINITY if } m<0.\)

Time complexity:

We calculate from \(C(0)\) to \(C(n)\), for each \(C(i)\), it takes \(O(k)\) comparisons, so the total time
complexity is \(O(nk)\).