Assignment 3
by Cheuk Him Calvin, Lo #100514352

A. [10] Design an algorithm that can identify cycles in a *directed* graph.

```
def search(G):
	visited = empty array of |V|
	cycle = empty array of |V|

	for u in V(G):
		if (isCycle(u) = TRUE):
			return TRUE

	return FALSE

def isCycle(u):
	visted[u] = true
	cycle[u] = true
	for v in Adj[u]:
		if visited[v] = FALSE && isCycle(v) = TRUE:
			return TRUE
		else if (cycle[v] = TRUE)
			return TRUE
	cycle[u] = FALSE
	return FALSE
```

B. *Coin denomination problem*

You have large amount of coins of each denominations: 1 cent, 5-cent, 10-cent, 25-cent, and 1 dollar. The optimal denomination problem is defined as:

PROBLEM: a total amount x cents to be made up in coins.

SOLUTION: Generate N_1, N_5, N_{10}, N_{25}, N_{100} such that

1. We minimize $N_1+N_5+N_{10} + N_{25}+N_{100}$
2. while subject to the constraint $N_1 + 5*N_2 + 10*N_{10} + 25*N_{25} + 100*N_{100} = x$.

Devise an algorithm that solves the optimal denomination problem using dynamic programming by following the these steps.

1. [10] How do you generate sub-problems?

   we can find the minimum number of coins to make (x - each denomination) cents.

2. [10] How do you synthesize the solution of a larger problem based on the solutions of the subproblems.

We can use each denomination as the first coin and find the next minimum by solving the sub-problems recursively. Finally, we can find the final answer from finding the minimum coins from each denomination that started.

3. [10] Write down the recursive solution.

```
static int find_min(int x){
	int[] deno = {1,5,10,25,100}
	int[] deno_temp_min = new int[5];
	int[] deno_final_min = new int[5];
	int min;

	if (x == 0)
		return(0);

	for (int i = 0; i < deno.length; i++)
		deno_final_min[i] = -1;

	for (int i = 0; i < deno.length; i++)  {
		if (deno[i] <= x)  {
			deno_temp_min[i] = find_min(x-deno[i]);
			deno_final_min[i] = deno_temp_min[i] + 1;
		}
	}

	//find min
	min = -1;
	for (int i = 0; i < deno.length; i++)  {
		if (deno_final_min[i] >= 0)  {
			if (min == -1 || deno_final_min[i] < min)  {
				min = deno_final_min[i];
			}
		}
	}

	return(min);
}
```

4. [10] What is the runtime of the recursive solution?

$O(2^x)$

5. [20] Formulate the bottom-up computation so that it runs in polynomial time complexity.

```
static int find_min(int x)  {
       int[] deno = {1,5,10,25,100};
       int[] result;
       int[] deno_temp_min, deno_final_min;
       int min;

       result = new int [x + 1];
       deno_temp_min = new int[deno.length];
       deno_final_min = new int[deno.length];

       result[0] = 0;

       for (int i = 1; i <= x; i++ )  {
              for (int j = 0; j < deno.length; j++ )
                            deno_final_min[j] = -1;


              for (int j = 0; j < deno.length; j++)  {
                     if (deno[j] <= i)
                            deno_final_min[j] = result[i-deno[j]] + 1;
              }

              result[i] = -1;

              for (int j = 0; j < deno.length; j++ )
              {
                     if (deno_final_min[j] >= 0 )  {
                            if (result[i] == -1 || deno_final_min[j] < result[i]) {
                                   result[i] = deno_final_min[j];
                            }
                     }
              }
       }

       return(result[x]);
}
```

6. [10] Formulate the memoization version of the recursive solution so that it runs in polynomial time complexity.

```
static int[] result; // initialize in main()

static int find_min(int x){
    int[] den = {1,5,10,25,100};
    int[] deno_temp_min = new int[5];
    int[] deno_final_min = new int[5];
    int min;

    if (x == 0)
        return(0);

    for (int i = 0; i < den.length; i++)
        deno_final_min[i] = -1;


    for (int i = 0; i < den.length; i++)  {
        if (den[i] <= x)  {
            if (result[x-den[i]] == 0) {
                deno_temp_min[i] = find_min(x-den[i]);
                result[x-den[i]] = deno_temp_min[i];
            }
            else {
                deno_temp_min[i] = result[x-den[i]];
            }

            deno_final_min[i] = deno_temp_min[i] + 1;
        }
    }

    //find min
    min = -1;
    for (int i = 0; i < den.length; i++)  {
        if (deno_final_min[i] >= 0)  {
            if (min == -1 || deno_final_min[i] < min)  {
                min = deno_final_min[i];
            }
        }
    }

    return(min);
}
```

C. Solve the optimal denomination problem using a greedy algorithm.

1. [10] What is the complexity of the greedy algorithm.

$O(x)$

2. [10] For the following values of x, compute the optimal number of coins used by both dynamic programming and the greedy algorithm.

(Assuming we have infinite coins of each denomination).

x	by dynamic programming	by greedy
104	5	5
122	5	5
141	5	5
156	5	5
157	6	6
167	7	7
188	8	8
189	9	9
200	2	2