알고리즘 과제 5. 동전 거스름돈
(동적 계획 알고리즘)

제출일 : 2017/05/19

경영학과 201101328　박성환

목차 :

1. 문제 정의

지난 3번째 과제에서 탐욕 알고리즘을 통해 동전 거스름돈 문제를 해결했었다. 그리고 이번에는 동적 계획법으로 같은 문제를 해결하기로 하며 둘의 차이점도 이후 살펴보도록 한다.

1.1. 알고리즘은 어떻게 구현할 것인가?

교수님께서 동적 계획법은 알고리즘을 구상만 할 수 있으면 구현 자체는 쉽다고 하셨는데, 부분 문제를 통해 어떻게 정답까지 도출할 수 있을지에 대해 구상해본다.

1.2. 어떻게 정해진 대로 출력할 것인가?

이번에는 출력 형식을 정해주셨다. 20개씩 끊어서 출력하라고 하셨는데 어떻게 구현할 수 있는지 생각해본다.

2. 해결방안

2.1. 알고리즘은 어떻게 구현할 것인가?

```
// get_coin_change Function
//// Get minimum number of coins
int get_coin_change(int change, int* coin_units, int unit_size) {
    int i, j;
    int number;
    int cache[1000];
    int answer;
    int count;

    cache[0] = 0;
    for(i = 1; i <= change; i++) {
        cache[i] = INFINITY;
    }

    for(i = 1; i <= change; i++) {
        number = INFINITY;
        for(j = 0; j < unit_size; j++) {
            if(i >= coin_units[j]) {
```

```
                number = (number <= cache[i - coin_units[j]] + 1) ? (number) : (cache[i -
coin_units[j]] + 1);
            }
        }
        cache[i] = number;
    }

    answer = cache[change];
    print_coin_change(cache, change);

    if(answer == INFINITY) return -1;

    return answer;
}
```

핵심 알고리즘이 구현된 get_coin_change 함수이다. 이 함수는 동전 액면가와 거스름돈을 입력받아 거스름돈에 맞는 최소 동전 개수를 출력한다. 동적 계획법으로 거스름돈 문제를 풀 때는 부분문제를 1원부터 1원씩 키워 나가면서 문제를 해결하고 최종적으로 주어진 거스름돈의 동전 개수를 구하는 방식으로 답을 구할 수 있다. 다음과 같이 예를 들어볼 수 있다.

거스름돈 x 원이 있고, 동전 단위 액면가들은 (a, b, c)(모두 x보다 작다)와 같다고 할 때 x원의 최소 동전 개수는 (x-a원의 개수 + 1), (x-b원의 개수 + 1), (x-c원의 개수 + 1)의 최소값과 같다. x원의 개수를 알기 위해 그 이전까지의 모든 개수를 고려하는 이유이기도 하다.

실제 알고리즘에서는 x원의 답을 도출하기 위해 0번째 인자를 0으로 초기화하고, 1번째부터 x원까지의 인자를 무한값으로 초기화해놓은 배열을 사용해서 각 인자의 값을 구한다. 각각을 구해나가다 보면 결국 x원의 개수를 구할 수 있다.

2.2. 어떻게 정해진 대로 출력할 것인가?

```
// Print coin change
//// Print from 1 to given change
void print_coin_change(int* cache, int size) {
    int iter_length = (size % 20 == 0)? size / 20: size / 20 + 1;
    int i, j;
    if(cache[size] == INFINITY) {
        printf("\n\nWe cannot make up money you want..\n");
        return;
    }
    for(i = 0; i < iter_length; i++) {
        printf("\n\nJ =  ");
        for(j = i*20 + 1; j <= i*20 + 20; j++) {
```

```
            if(j > size) return;
            printf("%3d  ", j);
        }
        printf("\nC =  ");
        for(j = i*20 + 1; j <= i*20 + 20; j++) {
            printf("%3d  ", cache[j]);
        }
    }
}
```

출력 방법을 담은 print_coin_change 함수이다. 이 함수는 get_coin_change 안에서 작동한다. 출력 예제를 보면 j와 c가 20개씩 나눠서 출력한다. 따라서 반복문이 사용 가능하고 '전체 배열의 크기 / 20 + 1'으로 반복 횟수를 구한다. 이때 횟수가 20으로 나눠지면, 예를 들어 0을 넣어도 횟수가 1이 되기 때문에 불필요하게 한 번 더 반복하게 된다. 따라서 이 때는 반복 횟수를 '크기 / 20'으로만 한다.

그 이후 한 번의 반복에 20개씩 숫자를 출력한다.

3. 테스트

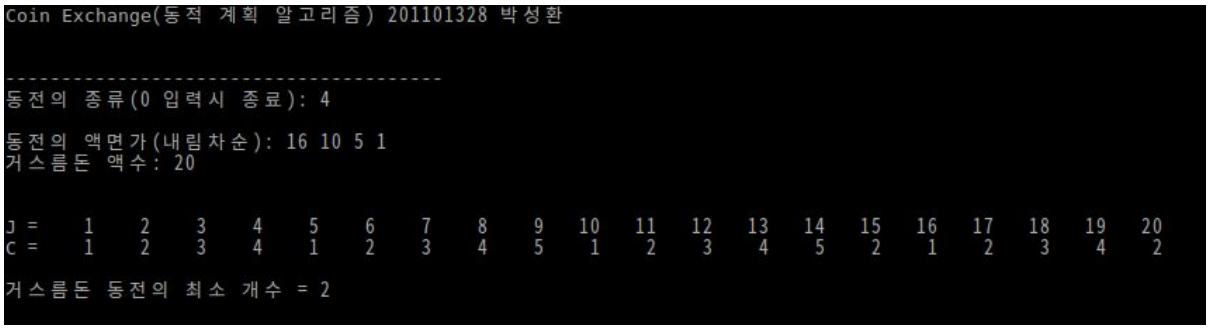
먼저 잘 작동하는지 확인한다.

```
Coin Exchange(동적 계획 알고리즘) 201101328 박성환


---------------------------------------
동전의 종류(0 입력시 종료): 5

동전의 액면가(내림차순): 100 50 10 5 1
거스름돈 액수: 200


J =    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
C =    1   2   3   4   1   2   3   4   5   1   2   3   4   5   2   3   4   5   6   2

J =   21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
C =    3   4   5   6   3   4   5   6   7   3   4   5   6   7   4   5   6   7   8   4

J =   41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
C =    5   6   7   8   5   6   7   8   9   1   2   3   4   5   2   3   4   5   6   2

J =   61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
C =    3   4   5   6   3   4   5   6   7   3   4   5   6   7   4   5   6   7   8   4

J =   81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
C =    5   6   7   8   5   6   7   8   9   5   6   7   8   9   6   7   8   9  10   1

J =  101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
C =    2   3   4   5   2   3   4   5   6   2   3   4   5   6   3   4   5   6   7   3

J =  121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
C =    4   5   6   7   4   5   6   7   8   4   5   6   7   8   5   6   7   8   9   5

J =  141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
C =    6   7   8   9   6   7   8   9  10   2   3   4   5   6   3   4   5   6   7   3

J =  161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
C =    4   5   6   7   4   5   6   7   8   4   5   6   7   8   5   6   7   8   9   5

J =  181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
C =    6   7   8   9   6   7   8   9  10   6   7   8   9  10   7   8   9  10  11   2

거스름돈 동전의 최소 개수 = 2
```

동전의 단위를 (100, 50, 10, 5, 1), 거스름돈을 200원으로 했을 때 정상적으로 작동하는 것을 알 수 있다. 또한 20개 단위로 정확히 나눠서 출력하고 있다.

이번에는 같은 문제를 동적 계획법과 탐욕 알고리즘을 해결했을 때를 비교해보도록 한다.

```
Coin Exchange (Greedy Algorithm) 201101328 박성환
Press 0 to exit

Enter Change: 20
 (16)  (1)  (1)  (1)  (1)
Total # of coins = 5
```



동전 단위를 (16, 10, 5, 1), 거스름돈을 20원으로 했을 때의 결과들이다. 윗 사진은 탐욕 알고리즘의 결과이고 밑에는 동적 계획법의 결과이다. 거스름돈이 20원일 때는 10원 동전을 2개 만드는 것이 최소값이다. 그런데 동적 계획법은 제대로 결과를 만들어냈는데 탐욕 알고리즘은 5개로 결과를 잘못 출력했다. 이것은 탐욕 알고리즘이 모든 경우의 수를 온전히 확인하지 않고 단위가 큰 값으로 계산하고 끝냈기 때문이다. 따라서 탐욕 알고리즘은 동적 계획법에 비해 제한적이다.

그대신 동적 계획법은 1원부터 20원까지 모든 부분문제들을 해결했어야 했기에 탐욕 알고리즘에 비해 더 번거롭고 시간이 오래 걸린다. 시간과 정확도의 Trade off 관계로 파악할 수 있다.

4. 결론

동적 계획법을 통해 동전 거스름돈 알고리즘을 해결할 수 있었다. 같은 문제를 탐욕 알고리즘과 비교함으로써 둘의 차이를 확인할 수 있기도 했다.

앞에서 살펴볼 수 있었듯이 핵심 알고리즘만 구상할 수 있다면 구현 코드는 너무나 간단하고 쉬워서 놀라웠다. 동적 계획법에 대해 막연한 두려움을 가지고 있었는데 문제를 더 찾아봐야 겠다는 생각을 했다.