Practical 6
Implementation of 0/1 Knapsack with dynamic programming

GAHAN M. SARAIYA, 18MCEC10

18mcec10@nirmauni.ac.in

I. INTRODUCTION

Aim of this practical is to perform chain matrix multiplication using Dynamic Programming

II. IMPLEMENTATION

I. Utility ***utility.h***

```
//
// Created by jarvis on 17/8/18.
//

#ifndef DSA_LAB_UTILITY_H
#define DSA_LAB_UTILITY_H

#include <string.h>
#include <stdarg.h>

int max(int a, int b) { return (a > b)? a : b; }
int min(int a, int b) { return (a < b)? a : b; }

int write_log(const char *format, ...) {
    if(DEBUG) {
        printf("\n[DEBUG_LOG]> ");
        va_list args;
        va_start (args, format);
        vprintf(format, args);
        va_end (args);
    }
}

int *get_min_max(int *array, int no_of_elements, int min_max[]){
    // get minimum and maximum of array
//    printf("elements of array: ");
    for(int i=0; i<no_of_elements; i++){
//        printf("%d ", *(array + i));
        if (*(array + i) < min_max[0])
```

```
30              min_max[0] = *(array + i);
31          if (*(array + i) > min_max[1])
32              min_max[1] = *(array + i);
33      }
34      return min_max;
35  }
36
37  int display_array(int *array, int no_of_elements){
38      // display given array of given size(no. of elements require because sizeof()
        ↪ returns max bound value)
39      write_log(": ");
40      for(int i=0; i<no_of_elements; i++){
41          write_log( "%d ", *(array + i));
42      }
43      return 0;
44  }
45
46  int show_2d_array(int **array, int no_of_elements){
47      // display given array of given size(no. of elements require because sizeof()
        ↪ returns max bound value)
48      write_log(": ");
49      for(int i=0; i<no_of_elements; i++){
50          printf("a[%d][i]: ", i);
51          for(int j=0; j<no_of_elements; j++) {
52  //            printf("array[%d][%d]: %d ", i, j, array[i][j]);
53              printf("%d\t", array[i][j]);
54          }
55          printf("\twhere 0<=i<=%d\n", no_of_elements-1);
56      }
57      return 0;
58  }
59
60  int display_2d_array(int **array, int no_of_elements){
61      // display given array of given size(no. of elements require because sizeof()
        ↪ returns max bound value)
62      write_log(": ");
63      for(int i=0; i<no_of_elements; i++){
64          printf("a[%d][]: ", i);
65          for(int j=0; j<no_of_elements; j++) {
66  //            printf("array[%d][%d]: %d ", i, j, array[i][j]);
67              printf("%d ", array[i][j]);
68          }
69          printf("\n");
70      }
71      return 0;
72  }
73
74
```

```
75 void swap(int *one, int *two){
76     // swap function to swap elements by location/address
77     int temp = *one;
78     *one = *two;
79     *two = temp;
80 }
81
82 #endif //DSA_LAB_UTILITY_H
```

II. Main Program - ***knapsack_DP.c***

```
1  //
    ↪ ----------------------------------------------------------------------------------------------
2  // Author: Gahan Saraiya
3  // GiT: http://github.com/gahan9/
4  // StackOverflow: https://stackoverflow.com/users/story/7664524
5  // Website: http://gahan9.github.io/
6  //
    ↪ ----------------------------------------------------------------------------------------------
7  // Implementing Knapsack Problem with Dynamic Programming
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <math.h>
12 #include "../utils/constant.h"
13 #include "../utils/utility.h"
14
15 int knapsack(int max_weight, int *weight, int *value, int total_elements) {
16     // Returns the maximum value that can be put in a knapsack(bag) of capacity
        ↪ max_weight
17     int **KnapsackMatrix = malloc((total_elements+1) * sizeof(int *));
18     for (int i=0; i < total_elements + 1; i++){
19         KnapsackMatrix[i] = malloc((max_weight + 1) * sizeof(int));
20     }
21
22     // Build table KnapsackMatrix[][]
23     for (int i = 0; i <= total_elements; i++) {
24         for (int w = 0; w <= max_weight; w++) {
25             if (i==0 || w==0) {
26                 // make initial knapsack matrix row and column to zero
27                 KnapsackMatrix[i][w] = 0;
28             }
29             else if (weight[i-1] <= w) {
30                 // select either previously selected one or
31                 KnapsackMatrix[i][w] = max(value[i - 1] + KnapsackMatrix[i - 1][w
                    ↪ - weight[i - 1]],
32                                            KnapsackMatrix[i - 1][w]
```

```
                                        );
            }
            else {
                // keep previously calculated value in knapsack matrix
                KnapsackMatrix[i][w] = KnapsackMatrix[i - 1][w];
            }
        }
    }
    // return last value of knapsack matrix as it is the result of optimal value
     ↪  of knapsack
    return KnapsackMatrix[total_elements][max_weight];
}

int main() {
    int number_of_elements = 15;
    int max_weight = 50 + rand() % 100;
    printf("Number of elements in knapsack : %d\n", number_of_elements);
    printf("Max knapsack weight : %d\n", max_weight);
    int *value = malloc(number_of_elements * sizeof(int));
    int *weight = malloc(number_of_elements * sizeof(int));
    printf("Knapsack \nweight \t value\n");
    for (int i=0; i<number_of_elements; i++) {
        value[i] = 5 + rand() % 600;
        weight[i] = 1 + rand() % 100;
        printf("%d\t%d\n", weight[i], value[i]);
    }
    printf("\nOptimal Profit Value = %d", knapsack(max_weight, weight, value,
     ↪  number_of_elements));
    return 0;
}
```

II.1 Output

```
Number of elements in knapsack : 15
Max knapsack weight : 91
Knapsack
weight          value
35        472
70        105
79        129
63        563
6        469
82        550
62        32
96        496
```

```
28        547
92        41
3        209
93        158
22        387
19        121
48        100

Optimal Profit Value = 1875
```