Data Structures and Algorithms
Assignment 6 / **Solution**

May 16, 2016

Hash Tables

Task 1. Hashing with chaining

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #define A (sqrt(5)−1)/2
6  #define TABLE_SIZE 8
7
8  struct element {
9      int value;
10     struct element ∗next;
11 };
12
13 void init(struct element ∗H[]) {
14     int i;
15     for (i = 0; i < TABLE_SIZE; i++)
16         H[i] = NULL;
17 }
18
19 int h(int k) {
20     return TABLE_SIZE∗(A∗k − (int)(A∗k));
21 }
22
23 void insert(int key, struct element ∗H[]) {
24     int i = h(key);
25     struct element∗ e = malloc(sizeof(struct element));
26
27     e−>value = key;
28     e−>next = H[i];
29     H[i] = e;
30 }
31
32 int search(int k, struct element ∗H[]) {
33     int hkey = h(k);
34     struct element ∗e = H[hkey];
35
```

```
36      while (e != NULL) {
37          if (e->value == k) { return hkey; }
38          else { e = e->next; }
39      }
40
41      return -1;
42  }
43
44  void printHash(struct element *H[]) {
45      struct element *e;
46      int i;
47
48      printf("Table size: %d\n", TABLE_SIZE);
49      for (i = 0; i < TABLE_SIZE; i++) {
50          if (H[i] != NULL) {
51              printf("i: %d\t key:", i);
52              e = H[i];
53              while (e != NULL) {
54                  printf(" -> %d", e->value);
55                  e = e->next;
56              }
57              printf("\n");
58          }
59      }
60  }
61
62  void main () {
63      struct element *H[TABLE_SIZE];
64      int searchValues[] = {1, 10112, 1113, 5568, 337};
65      int i;
66
67      init(H);
68      insert( 111, H);
69      insert(10112, H);
70      insert(1113, H);
71      insert(5568, H);
72      insert(63, H);
73      insert(1342, H);
74      insert(21231, H);
75
76      printHash(H);
77
78      for (i = 0; i < 5; i++) {
79          printf("Searching for %d, found %d\n", searchValues[i],
80                  search(searchValues[i], H));
81      }
82  }
83
84  // Linux, Mac: gcc task1.c -o task1; ./task1
85  // Windows: gcc task1.c -o task1; task1
```

Output:

```
Table size: 8
i: 1  key: -> 5568
i: 3  key: -> 21231 -> 1342
i: 4  key: -> 10112 -> 111
i: 6  key: -> 1113
i: 7  key: -> 63
Searching for 1, found -1
Searching for 10112, found 4
Searching for 1113, found 6
Searching for 5568, found 1
Searching for 337, found -1
```

Task 2: Maximum Wine Profit

```
1  #include<stdio.h>
2
3  #define SIZE_A 4
4  #define SIZE_B 5
5
6  #define MAX_N SIZE_B
7
8  int m[MAX_N][MAX_N];
9
10 int max(int x, int y) {
11     if (x > y) return x; else return y;
12 }
13
14 void prepareMemoizationArray(int n) {
15     int i, j;
16
17     for(i = 0; i < n; i++) {
18         for(j = 0; j < n; j++) {
19             m[i][j] = -1;
20         }
21     }
22 }
23
24 void printMemoizationArray(int n) {
25     int i, j;
26
27     for(i = 0; i < n; i++) {
28         for(j = 0; j < n; j++) {
29             printf("%2d ", m[i][j]);
30         }
31         printf("\n");
32     }
33 }
34
35 int wineprofitRecursive(int price[], int n, int begin, int end) {
36     int year = n - (end-begin+1) + 1;
37
38     if (begin > end) return 0;
39
40     return max(
41         wineprofitRecursive(price, n, begin+1, end) + year * price[begin],
42         wineprofitRecursive(price, n, begin, end-1) + year * price[end]
43     );
44 }
45
46 int wineprofitMemoized(int price[], int n, int begin, int end) {
47     if (begin > end) return 0;
48     if (m[begin][end] != -1)
49         return m[begin][end];
50     int year = n - (end-begin+1) + 1;
51     return m[begin][end] = max(
52         wineprofitMemoized(price, n, begin+1, end) + year * price[begin],
```

```
53          wineprofitMemoized(price, n, begin, end−1) + year ∗ price[end]
54      );
55  }
56
57  int wineprofitDynamic(int price[], int n) {
58      int i, j;
59      int begin, end, year;
60
61      for (i = 0; i < n; i++) {
62          m[i][i] = price[i] ∗ n;
63      }
64
65      for (j = 1; j < n; j++) {
66          for (i = 0; i < n − j; i++) {
67              begin = i;
68              end = i + j;
69              year = n − (end − begin);
70
71              m[begin][end] = max(
72                  m[begin + 1][end] + year ∗ price[begin],
73                  m[begin][end − 1] + year ∗ price[end]
74              );
75          }
76      }
77      return m[0][n − 1];
78  }
79
80
81  void main() {
82      int i;
83      int testPriceA[] = {1,4,2,3};
84      int testPriceB[] = {2,3,5,1,4};
85
86      printf("prices: ");
87      for(i=0; i<SIZE_A; i++)
88          printf("%2d ", testPriceA[i]);
89      printf("\n");
90      printf("wineprofitRecursive:\n");
91      printf("Max profit A: %d\n", wineprofitRecursive(testPriceA, SIZE_A, 0, SIZE_A−1));
92      printf("\n");
93
94      printf("wineprofitMemoized:\n");
95      prepareMemoizationArray(SIZE_A);
96      printf("Max profit A: %d\n", wineprofitMemoized(testPriceA, SIZE_A, 0, SIZE_A−1));
97      printf("\n");
98
99      printf("wineprofitDynamic:\n");
100     printf("Max profit A: %d\n", wineprofitDynamic(testPriceA, SIZE_A));
101     printf("Memoized Array A:\n");
102     printMemoizationArray(SIZE_A);
103
104     printf("\n~~~~~~~~~\n\n");
105
106     printf("prices: ");
```

```
107     for(i=0; i<SIZE_B; i++)
108         printf("%2d ", testPriceB[i]);
109     printf("\n");
110     printf("wineprofitRecursive:\n");
111     printf("Max profit B: %d\n", wineprofitRecursive(testPriceB, SIZE_B, 0, SIZE_B-1));
112     printf("\n");
113
114     printf("wineprofitMemoized:\n");
115     prepareMemoizationArray(SIZE_B);
116     printf("Max profit B: %d\n", wineprofitMemoized(testPriceB, SIZE_B, 0, SIZE_B-1));
117     printf("\n");
118
119     printf("wineprofitDynamic:\n");
120     printf("Max profit B: %d\n", wineprofitDynamic(testPriceB, SIZE_B));
121     printf("Memoized Array B:\n");
122     printMemoizationArray(SIZE_B);
123
124 }
125
126 // Linux, Mac: gcc task2.c -o task2; ./task2
127 // Windows: gcc task2.c -o task2; task2
```

Output:

```
wineprofitRecursive:
Max profit A: 29
Max profit B: 50

wineprofitMemoized:
Max profit A: 29
Max profit B: 50

wineprofitDynamic:
Max profit A: 29
Max profit B: 50

Memoized Array A:
4 19 24 29
-1 16 22 28
-1 -1 8 18
-1 -1 -1 12

Memoized Array B:
10 23 43 45 50
-1 15 37 40 48
-1 -1 25 29 41
-1 -1 -1 5 24
-1 -1 -1 -1 20
```

Task 3.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define MAX_N 10000
6
7  // ... include code from Task 2 ...
8
9  // ---------- auxiliary functions ----------
10
11 void readFile(char filename[], int output[], int *n) {
12     FILE *f;
13     int i;
14
15     f=fopen(filename, "r");
16
17     i=0;
18     while(fscanf(f, "%d", &output[i]) == 1) i++;
19     *n=i;
20
21     fclose(f);
22 }
23
24 // ---------- main ----------
25
26 void main() {
27     int a[MAX_N];
28     int n;
29     clock_t start;
30     clock_t end;
31     float seconds;
32
33     printf("                    | small    | medium   | large    |\n");
34
35     // ---------- dynamic programming ----------
36
37     // small
38     readFile("small.txt", a, &n);
39     start = clock();
40     wineprofitDynamic(a, n);
41     end = clock();
42     seconds = (float)(end-start)/CLOCKS_PER_SEC;
43     printf("Dynamic Programming | %8.4f | ", seconds);
44
45     // medium
46     readFile("medium.txt", a, &n);
47     start = clock();
48     wineprofitDynamic(a, n);
49     end = clock();
50     seconds = (float)(end-start)/CLOCKS_PER_SEC;
51     printf("%8.4f |", seconds);
52
```

```
53      // large
54      readFile("large.txt", a, &n);
55      start = clock();
56      wineprofitDynamic(a, n);
57      end = clock();
58      seconds = (float)(end−start)/CLOCKS_PER_SEC;
59      printf("␣%8.4f␣|\n", seconds);
60
61      // −−−−−−−−−−−−−−−−−−−− memoization −−−−−−−−−−−−−−−−−−−−−−
62
63      // small
64      readFile("small.txt", a, &n);
65      start = clock();
66      prepareMemoizationArray(n);
67      wineprofitMemoized(a, n, 0, n−1);
68      end = clock();
69      seconds = (float)(end−start)/CLOCKS_PER_SEC;
70      printf("Memoization␣␣␣␣␣␣␣␣␣|␣%8.4f␣|␣", seconds);
71
72      // medium
73      readFile("medium.txt", a, &n);
74      start = clock();
75      prepareMemoizationArray(n);
76      wineprofitMemoized(a, n, 0, n−1);
77      end = clock();
78      seconds = (float)(end−start)/CLOCKS_PER_SEC;
79      printf("%8.4f␣|", seconds);
80
81      // large
82      readFile("large.txt", a, &n);
83      start = clock();
84      prepareMemoizationArray(n);
85      wineprofitMemoized(a, n, 0, n−1);
86      end = clock();
87      seconds = (float)(end−start)/CLOCKS_PER_SEC;
88      printf("␣%8.4f␣|\n", seconds);
89
90      // −−−−−−−−−−−−−−−−−−−− recursive −−−−−−−−−−−−−−−−−−−−−−−−
91
92      // small
93      readFile("small.txt", a, &n);
94      start = clock();
95      wineprofitRecursive(a, n, 0, n−1);
96      end = clock();
97      seconds = (float)(end−start)/CLOCKS_PER_SEC;
98      printf("Recursion␣␣␣␣␣␣␣␣␣|␣%8.4f␣|␣", seconds);
99
100     // medium
101     readFile("medium.txt", a, &n);
102     start = clock();
103     wineprofitRecursive(a, n, 0, n−1);
104     end = clock();
105     seconds = (float)(end−start)/CLOCKS_PER_SEC;
106     printf("%8.4f␣|", seconds);
```

```
107
108     // large
109     printf("␣␣␣----␣␣␣|\n");
110 }
111
112 // Linux, Mac: gcc task3.c -o task3; ./task3
113 // Windows: gcc task3.c -o task3; task3
```

	small (30)	medium (35)	large (10000)
Dynamic Programming	0.0000	0.0000	0.5347
Memoization	0.0000	0.0000	1.2090
Recursion	10.8858	362.0109	–