Algorithm Class Assignment 3 Dynamic Programming

Question 1

Explain Why The Problem Is (Or Not) Good For Dynamic Programming

The problem is good for dynamic programming because for calculating the biggest total preference value, we will need to condiser a lot of probabilities and there are conditions that recalculate the result gained before. By implementing dynamic programming, we can reduce the redundant steps to improve efficiency.

Design & Implement The Algorithm For The Problem

Design & Implement:

```cpp
#include <iostream>
using namespace std;

// find the bigger integer between the two
int Maximum(int a, int b) {
    if (a > b) {
        return a;
    }

    else {
        return b;
    }
}

// bottom up method dynamic programming
// find the maximum value of preference that can be accomplished under the budg
int Preference(int budget, int item_price[], int item_preference[], int items)
    int i;
    int j;
    int K[items + 1][budget + 1];

    for(i = 0; i <= items; i++) {
        for(j = 0; j <= budget; j++) {
            if (i == 0 || j == 0) {
                K[i][j] = 0;
            }

            else if (item_price[i - 1] <= j) {
                K[i][j] = Maximum(item_preference[i - 1] + K[i - 1][j - item_p
            }

            else {
                K[i][j] = K[i - 1][j];
            }
        }
    }

    return K[items][budget];
}

int main() {
    // sample item preference list
    int item_preference[] = {80, 28, 70, 55, 43};
    // sample item price list
    int item_price[] = {40, 20, 60, 75, 59};
    // sample budget
    int budget = 150;
    // amount of items
    int items = sizeof (item_preference) / sizeof (item_preference[0]);

    cout << Preference(budget, item_price, item_preference, items);

    return 0;
}
```

Analyze The Complexity Of The Algorithm

Time Complexity:

- time complexity for the implementation is O(n * m)

‘n’ to be the amount of the items
‘m’ to be the total budget

Space Complexity:

- space complexity for the implementation is O(n * m)

‘n * m’ is the size of a two-dimensional array

Question 2

Explain Why The Problem Is (Or Not) Good For Dynamic Programming

Dynamic programming is good for this question is because the result of the problems is related. By using brute force, we will spend a lot of time wasting on same calculation, which dynamic programming resolve this problem.

Design & Implement The Algorithm For The Problem

Design & Implement:

```cpp
#include <iostream>
using namespace std;

// find the larger value
int Maximum(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}

// find the maximum earnings for the government
// accomplished by the limited shore range
// bottom up method dynamic programming
int Earnings(int length, int section[], int price[], int amount) {
    int i;
    int j;
    int S[amount + 1][length + 1];

    for(i = 0; i <= amount; i++) {
        for(j = 0; j <= length; j++) {
            if (i == 0 || j == 0) {
                S[i][j] = 0;
            }

            else if (section[i - 1] <= j) {
                S[i][j] = Maximum(price[i - 1] +
                        S[i - 1][j - section[i - 1]],
                        S[i - 1][j]);
            }

            else {
                S[i][j] = S[i - 1][j];
            }
        }
    }

    return S[amount][length];
}

int main() {
    // price list for different shore section
    int price[] = {15, 25, 10, 20, 5};
    // partition for different shore section
    int section[] = {27, 22, 19, 15, 31};
    // total length of the shore
    int length = 50;
    // amount of the partitions
    int amount = sizeof (price) / sizeof (price[0]);

    cout << Earnings(length, section, price, amount);

    return 0;
}
```

Analyze The Complexity Of The Algorithm

<u>Time Complexity:</u>

- time complexity for the implementation is O(n * m)

'n' to be the total partitions we have split
'm' to be the length of the shore

<u>Space Complexity:</u>

- space complexity for the implementation is O(n * m)

'n * m' is the size of a two-dimensional array

Question 3

Textbook 3.4

<u>Binomial Theorem Using One-Dimensional Array:</u>

```
// Reference For nCk:
//
// nCk = n! / [k! * (n - k)!]
//     = [n * (n - 1) * ... * (n - k + 1)] / k!

int binomial_theorem_with_1D_array (int n, int k) {
    index i;
    int result[k + 1];

    for (i = 1; i <= k; i++) {
        result[i] = ((n + 1 - i) * result[i - 1]) / i;
    }

    int answer = result[k];

    return answer;
}
```

Textbook 3.5

D1: (START)

	1	2	3	4	5	6	7
1	0	4	INF	INF	INF	10	INF
2	3	0	INF	18	INF	INF	INF
3	INF	6	0	INF	INF	INF	INF
4	INF	5	15	0	2	19	5

5	INF	INF	12	1	0	INF	INF
6	INF	INF	INF	INF	INF	0	10
7	INF	INF	INF	8	INF	INF	0

P1: (START)

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

D2:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	4	INF	INF	INF	10	INF
2	3	0	INF	18	INF	13	INF
3	INF	6	0	INF	INF	INF	INF
4	INF	5	15	0	2	19	5
5	INF	INF	12	1	0	0	INF
6	INF	INF	INF	INF	INF	0	10
7	INF	INF	INF	8	INF	INF	0

P2:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0

7	0	0	0	0	0	0	0

D3:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	4	INF	22	INF	10	INF
2	3	0	INF	18	INF	13	INF
3	9	6	0	24	INF	19	INF
4	8	5	15	0	2	18	5
5	INF	INF	12	1	0	INF	INF
6	INF	INF	INF	INF	INF	0	10
7	INF	INF	INF	8	INF	INF	0

P3:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	0	2	0	0	0
2	0	0	0	0	0	1	0
3	2	0	0	2	0	2	0
4	2	0	0	0	0	2	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

D4:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	4	INF	22	INF	10	INF
2	3	0	INF	18	INF	13	INF
3	9	6	0	24	INF	19	INF
4	8	5	15	0	2	18	5
5	21	18	12	1	0	31	INF
6	INF	INF	INF	INF	INF	0	10
7	INF	INF	INF	8	INF	INF	0

P4:

	1	2	3	4	5	6	7
1	0	0	0	2	0	0	0
2	0	0	0	0	0	1	0
3	2	0	0	2	0	2	0
4	2	0	0	0	0	2	0
5	3	3	0	0	0	3	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

D5:

	1	2	3	4	5	6	7
1	0	4	37	22	24	10	27
2	3	0	33	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	15	0	2	18	5
5	9	6	12	1	0	19	6
6	INF	INF	INF	INF	INF	0	10
7	16	13	23	8	10	26	0

P5:

	1	2	3	4	5	6	7
1	0	0	4	2	4	0	4
2	0	0	4	0	4	1	4
3	2	0	0	2	4	2	4
4	2	0	0	0	0	2	0
5	4	4	0	0	0	4	4
6	0	0	0	0	0	0	0
7	4	4	4	0	4	4	0

D6:

	1	2	3	4	5	6	7
1	0	4	36	22	24	10	27
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	INF	INF	INF	INF	INF	0	10
7	16	13	22	8	10	26	0

P6:

	1	2	3	4	5	6	7
1	0	0	5	2	4	0	4
2	0	0	5	0	4	1	4
3	2	0	0	2	4	2	4
4	2	0	5	0	0	2	0
5	4	4	0	0	0	4	4
6	0	0	0	0	0	0	0
7	4	4	5	0	4	4	0

D7:

	1	2	3	4	5	6	7
1	0	4	36	22	24	10	20
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	INF	INF	INF	INF	INF	0	10
7	16	13	22	8	0	26	0

P7:

	1	2	3	4	5	6	7
1	0	0	5	2	4	0	6

2	0	0	5	0	4	1	4
3	2	0	0	2	4	2	4
4	2	0	5	0	0	2	0
5	4	4	0	0	0	4	4
6	0	0	0	0	0	0	0
7	4	4	5	0	4	4	0

D8: (RESULT)

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	4	36	22	24	10	20
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	26	23	32	18	20	0	10
7	16	13	22	8	10	26	0

P8: (RESULT)

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	5	2	4	0	6
2	0	0	5	0	4	1	4
3	2	0	0	2	4	2	4
4	2	0	5	0	0	2	0
5	4	4	0	0	0	4	4
6	7	7	7	7	7	0	0
7	4	4	5	0	4	4	0

Textbook 3.6

P:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	5	2	4	0	6

2	0	0	5	0	4	1	4
3	2	0	0	2	4	2	4
4	2	0	5	0	0	2	0
5	4	4	0	0	0	4	4
6	7	7	7	7	7	0	0
7	4	4	5	0	4	4	0

Operation:

Find the shortest path from v7 to v3:

- path(7, 3)
- P[7][3] = 5
- path(7, P[7][3])
  - path(7, 5)
  - P[7][5] = 4
  - path(7, P[7][4])
    - path(7, 4)
    - P[7][4] = 0
  - mid point “v4”
  - path(P[7][5], 5)
    - path(4, 5)
    - P[4][5] = 0
- mid point “v5”
- path(5, 3)
- P[5][3] = 0

Result:

v7 -> v4 -> v5 -> v3

Textbook 3.13

Implementation:

```
Zero Interval:

M[1][1] = 0
M[2][2] = 0
M[3][3] = 0
M[4][4] = 0
M[5][5] = 0
M[6][6] = 0

One Interval:

M[1][2] = M[1][1] + M[2][2] + d0 * d1 * d2 = 200
M[2][3] = M[2][2] + M[3][3] + d1 * d2 * d3 = 400
M[3][4] = M[3][3] + M[4][4] + d2 * d3 * d4 = 200
M[4][5] = M[4][4] + M[5][5] + d3 * d4 * d5 = 2000

Two Interval:

M[1][3] = M[1][2] + M[3][3] + d0 * d2 * d3 = 1200
M[2][4] = M[2][2] + M[3][4] + d1 * d2 * d4 = 240
M[3][5] = M[3][4] + M[5][5] + d2 * d4 * d5 = 700

Three Interval:

M[1][4] = M[1][1] + M[2][4] + d0 * d1 * d4 = 320
M[2][5] = M[2][4] + M[5][5] + d1 * d4 * d5 = 400

Four Interval:

M[1][5] = M[1][4] + M[5][5] + d0 * d4 * d5 = 1320

```

Result:

	1	**2**	**3**	**4**	**5**
1	0	200	1200	320	1320
2		0	400	240	640
3			0	200	700
4				0	2000
5					0

Order & Multiply times:

Order: ((A1(A2(A3A4)))A5)

Multiply Times: 1320

tags: `Algorithm Class`