1. **소스코드 및 주석**
   1. **1번문제**

#include <stdio.h>

#define max(x, y) (x) > (y) ? (x) : (y)

#define n 4

#define bag\_capacity 10

struct Thing {

int weight, val;

};

int value[5][11];

int main() {

Thing thing[5];

bool selected[5] = { false, };

thing[1] = { 5 ,10 };

thing[2] = { 4 ,40 };

thing[3] = { 6 ,30 };

thing[4] = { 3 ,50 };

int vi = 0, wi = 0;

for (int j = 0; j <= bag\_capacity; j++) {

value[0][j] = 0;

}

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= bag\_capacity; j++) {

wi = thing[i].weight;

vi = thing[i].val;

if (wi > j) {

value[i][j] = value[i - 1][j];

}

else {

value[i][j] = max(value[i - 1][j], vi + value[i - 1][j - wi]);

}

}

}

int pre\_wi = 0;

for (int j = 1; j <= bag\_capacity; j++) {

if (value[n][j] != value[n][j - 1]) {

vi = value[n][j] - value[n][j - 1];

wi = j - pre\_wi;

for (int index = 1; index <= n; index++) {

if (selected[index] != true && thing[index].val == vi && thing[index].weight == wi) {

selected[index] = true;

}

}

pre\_wi = j;

}

}

printf("value 표\n");

for (int j = 0; j <= bag\_capacity; j++) {

printf("%d\t", j);

}

printf("\n");

for (int i = 0; i <= n; i++) {

for (int j = 0; j <= bag\_capacity; j++) {

printf("%d\t", value[i][j]);

}

printf("\n");

}

printf("\n최대 가치 : %d만원 \n", value[n][bag\_capacity]);

printf("선택된 아이템 : ");

for (int i = 1; i <= n; i++) {

if (selected[i] == true)

printf("Item %d ", i);

}

return 0;

}

**1-2. 2번문제**

#include <stdio.h>

#include <limits.h>

#define max(x, y) ((x) > (y) ? (x) : (y))

#define n 4

#define bag\_capacity 10

struct Thing {

int weight, val;

};

struct State {

int val; // 총 가치

int weight; // 달성한 총 무게

};

State state[5][5][11]; // [i][k][j]

void print\_state\_table() {

for (int i = 0; i <= n; i++) {

printf("==== i = %d 번째 물건까지 고려 ====\n", i);

for (int k = 0; k <= n; k++) {

printf("선택한 물건 개수 k = %d:\n", k);

for (int j = 0; j <= bag\_capacity; j++) {

State s = state[i][k][j];

if (s.val == 0 && s.weight == 0)

printf(" - ");

else

printf("(%2d,%2d) ", s.val, s.weight);

}

printf("\n");

}

printf("\n");

}

}

int main() {

Thing thing[5];

thing[1] = { 5 ,10 };

thing[2] = { 4 ,40 };

thing[3] = { 6 ,30 };

thing[4] = { 3 ,50 };

// 초기화

for (int i = 0; i <= n; i++)

for (int k = 0; k <= n; k++)

for (int j = 0; j <= bag\_capacity; j++)

state[i][k][j] = { 0, 0 };

for (int i = 1; i <= n; i++) {

for (int k = 0; k <= i; k++) {

for (int j = 0; j <= bag\_capacity; j++) {

// 이전 상태 복사

state[i][k][j] = state[i - 1][k][j];

if (k > 0 && j >= thing[i].weight) {

State prev = state[i - 1][k - 1][j - thing[i].weight];

int new\_val = prev.val + thing[i].val;

int new\_weight = prev.weight + thing[i].weight;

if (new\_val > state[i][k][j].val ||

(new\_val == state[i][k][j].val && new\_weight < state[i][k][j].weight)) {

state[i][k][j] = { new\_val, new\_weight };

}

}

}

}

}

// 최적 결과 찾기

State best = { 0, 0 };

for (int k = 0; k <= n; k++) {

for (int j = 0; j <= bag\_capacity; j++) {

State s = state[n][k][j];

if (s.val <= 80 && (s.val > best.val || (s.val == best.val && s.weight < best.weight))) {

best = s;

}

}

}

// dp 표 출력하기

print\_state\_table();

// 선택한 물건 찾기

int difference\_vaㅣ = 0, difference\_weight = 0;

printf("===================================================================================\n");

printf("선택한 물건: \n");

for (int j = 1; j <= bag\_capacity; j++) {

difference\_vaㅣ = state[n][n][j].val - state[n][n][j - 1].val;

difference\_weight = state[n][n][j].weight - state[n][n][j - 1].weight;

if (difference\_weight != 0) {

for (int index = 1; index <= n; index++) {

if (thing[index].val == difference\_vaㅣ && thing[index].weight == difference\_weight) {

printf(" Item %d (%d kg, %d 만원)\n", index, thing[index].weight, thing[index].val);

}

}

}

}

printf("===================================================================================\n");

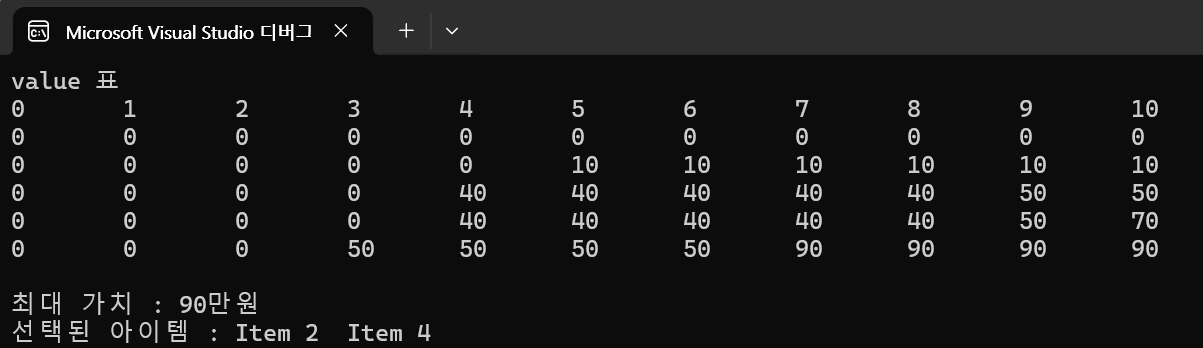
printf("최대 가치: %d, 총 무게: %d\n", best.val, best.weight);

return 0;

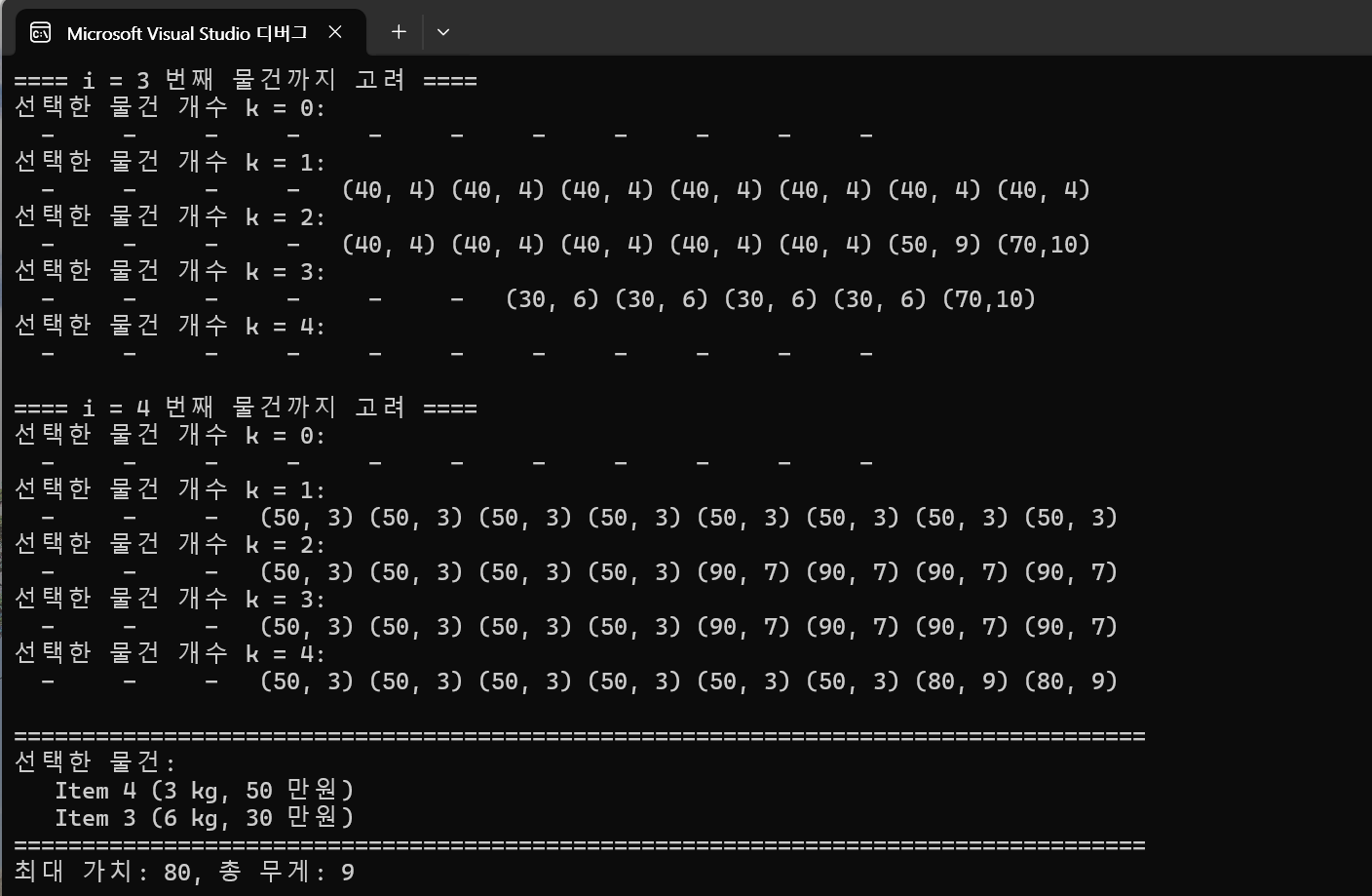
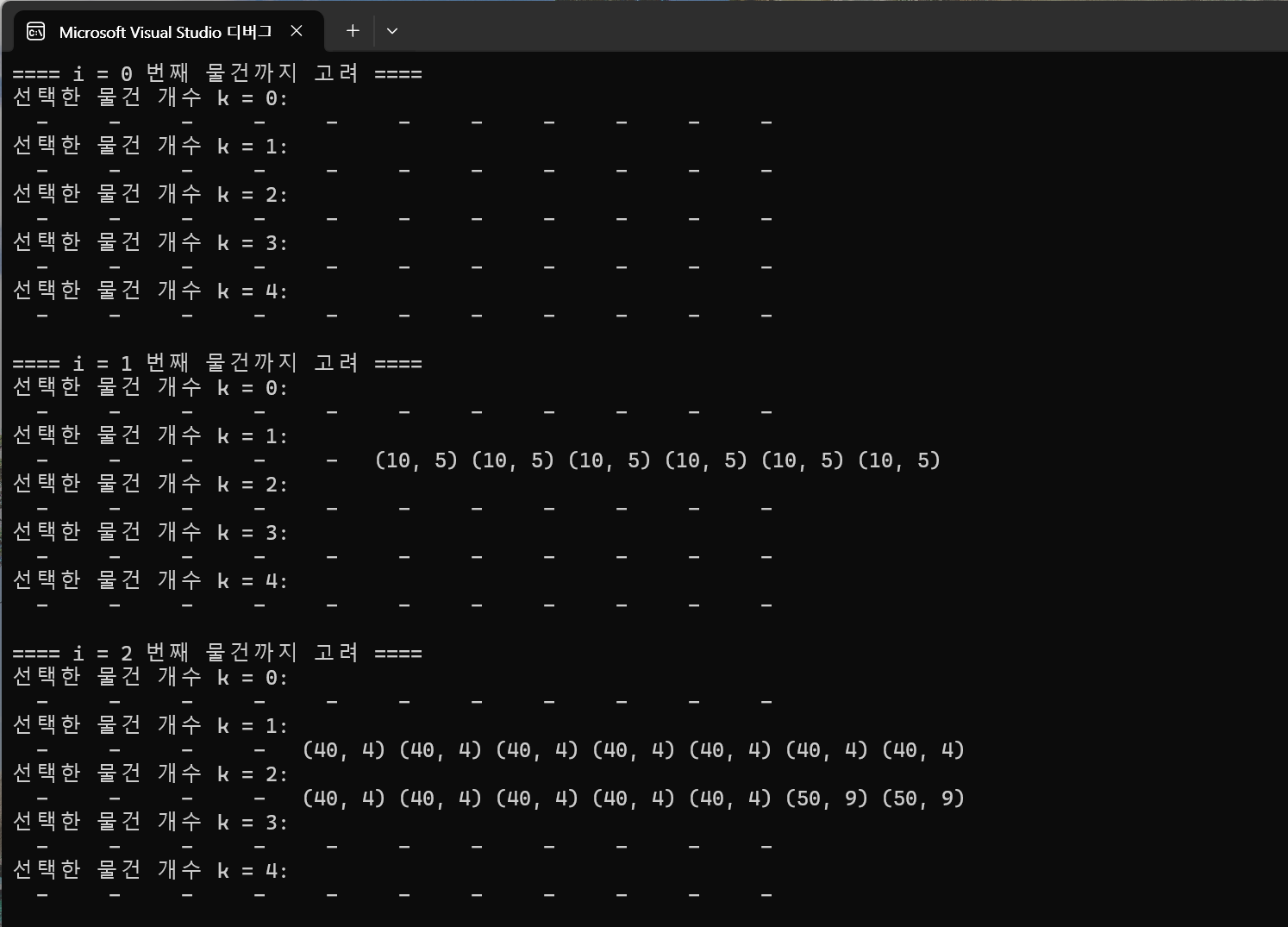
}

1. **실행화면 캡처**

**2-1. 1번문제**



**2-1. 2번문제**



1. **고찰**

* 사실 2번 문제에서 2차원 다이나믹 프로그래밍으로 풀려고했지만, 그전에 선택한 아이템 조합의 무게와 가치를 고려하지 못한다는 오류가 존재하여서 3차원으로 만들었다 즉, 모든 조합의 가치와 무게를 검사하는 코드를 작성하였다
  + 더 효율적인 방법이 존재할 것도 같지만, 아직 생각나지않는다. 3차원 dp이므로, n이 조금만 커져도 매우 큰 시간복잡도를 가질것 같다는 생각이 들었다
* 모든 풀이는 1이상의 가치, 1이상의 무게가 있다고 가정하고 풀었다. 그외의 값이 들어오면 잘못된 값이 계산되지만 현실에는 그런 값이 없다고 생각하여 그렇게 풀었다. 만약, 0또는 음수의 값이 들어올때의 0-1 knapscak을 풀이하려면 pseudo code를 다시 만들어야 하며 더 복잡해질 것 같다
* 간단한 코드를 위하여, dp표를 찾은 후에 어떤 물건을 선택했는지 조사하도록 조사하였다. 원리는 가치와 무게의 차이만큼의 물건이 있는지 조사하는 것이다 하지만, 이방식은 똑같은 가치와 무게를 가진 물건이 2개 이상 존재하고 그것이 최소 아이템 조합이여도, 선택된 물건 조합이 1개라는 것이다. 아이템 조합이 여러개라는 예외 사항을 상정하진 못했다. 거의 존재하지않는 경우이지만, 다시 생각해볼 필요가 있다고 생각하였다