1. **소스코드 및 주석**

**1-1. 문제1**

#include<stdio.h>

#include <iostream>

#define SET\_SIZE 10

#define SUBSET\_SIZE 10

#define TRUE 1

#define FALSE 0

using namespace std;

int flag[SET\_SIZE];

void flagInit() {

for (int i = 0; i < SET\_SIZE; i++) {

flag[i] = TRUE;

}

}

typedef struct Subsets {

int\* sub;

int size;

} Subsets;

void subsetIncoding(Subsets subsets[], int\* sub[], int subSize[]) {

for (int i = 0; i < SUBSET\_SIZE; i++) {

subsets[i].sub = sub[i];

subsets[i].size = subSize[i];

}

}

bool isEmpty() {

for (int i = 0; i < SET\_SIZE; i++) {

if (flag[i] == TRUE)

return FALSE;

}

return TRUE;

}

int selectSet(int u[], Subsets subsets[]) {

int index = 0, selectedIndex = -1, count = 0, preCount = 0;

for (int i = 0; i < SUBSET\_SIZE; i++) {

count = 0;

for (int j = 0; j < SET\_SIZE; j++) {

if (flag[j] == FALSE) {

continue;

}

for (int k = 0; k < subsets[i].size; k++) {

if (u[j] == subsets[i].sub[k]) {

count++;

break;

}

}

}

if (count == 0) {

continue;

}

if (preCount < count) {

selectedIndex = i;

preCount = count;

}

}

return selectedIndex;

}

void subtractSet(int u[], Subsets subsets[], int selectedIndex) {

for (int i = 0; i < SET\_SIZE; i++) {

if (flag[i] == FALSE) {

continue;

}

for (int j = 0; j < subsets[selectedIndex].size; j++) {

if (u[i] == subsets[selectedIndex].sub[j]) {

flag[i] = FALSE;

break;

}

}

}

}

void printSubset(Subsets subsets[], int selectedIndex) {

printf("선택된 집합 s%d = { %d", selectedIndex + 1, subsets[selectedIndex].sub[0]);

for (int x = 1; x < subsets[selectedIndex].size; x++) {

printf(", %d", subsets[selectedIndex].sub[x]);

}

printf(" }\n");

}

void printRemaining(int u[], Subsets subsets[], int selectedIndex) {

cout << "선택된 집합을 제외한 나머지 원소: ";

for (int i = 0; i < SET\_SIZE; i++) {

if (flag[i] == TRUE) {

cout << u[i] << " ";

}

}

cout << endl;

}

void setCover(int u[], Subsets subsets[]) {

flagInit();

while (!isEmpty()) {

int selectedIndex = selectSet(u, subsets);

if (selectedIndex == -1) {

cout << "더이상 커버링할 수 있는 부분집합이 없습니다." << endl;

printRemaining(u, subsets, selectedIndex);

break;

}

subtractSet(u, subsets, selectedIndex);

printSubset(subsets, selectedIndex);

}

}

void printAllsets(int u[], Subsets subsets[]) {

printf("전체집합: \n");

printf("{ %d", u[0]);

for(int i = 1; i < SET\_SIZE; i++) {

printf(", %d ", u[i]);

}

printf(" }\n");

printf("모든 부분집합:\n");

for (int i = 0; i < SUBSET\_SIZE; i++) {

printf("s%d = { %d", i + 1, subsets[i].sub[0]);

for (int j = 1; j < subsets[i].size; j++) {

printf(", %d", subsets[i].sub[j]);

}

printf(" }\n");

}

printf("-------------------------\n");

}

int main(void) {

int u[SET\_SIZE] = { 1,2,3,4,5,6,7,8,9,10 };

int s1[] = { 1,2,3,8 };

int s2[] = { 1,2,3,4,8 };

int s3[] = { 1,2,3,4 };

int s4[] = { 2,3,4,5,7,8 };

int s5[] = { 4,5,6,7 };

int s6[] = { 5,6,7,9,10 };

int s7[] = { 4,5,6,7 };

int s8[] = { 1,2,4,8 };

int s9[] = { 6,9 };

int s10[] = { 6,10 };

Subsets subsets[SUBSET\_SIZE];

int\* sub[SUBSET\_SIZE] = { s1, s2, s3, s4, s5, s6, s7, s8, s9, s10 };

int subSize[SUBSET\_SIZE] = { 4,5,4,6,4,5,4,4,2,2 };

subsetIncoding(subsets, sub, subSize);

printAllsets(u, subsets);

setCover(u, subsets);

return 0;

}

**1-2. 문제2**

#include<stdio.h>

#include <iostream>

#define SET\_SIZE 10

#define SUBSET\_SIZE 10

#define TRUE 1

#define FALSE 0

using namespace std;

int flag[SET\_SIZE];

void flagInit() {

for (int i = 0; i < SET\_SIZE; i++) {

flag[i] = TRUE;

}

}

typedef struct Subsets{

int\* sub;

int size;

int cost;

} Subsets;

void subsetIncoding(Subsets subsets[], int\* sub[], int subSize[], int subCost[]) {

for (int i = 0; i < SUBSET\_SIZE; i++) {

subsets[i].sub = sub[i];

subsets[i].size = subSize[i];

subsets[i].cost = subCost[i];

}

}

bool isEmpty() {

for (int i = 0; i < SET\_SIZE; i++) {

if (flag[i] == TRUE)

return FALSE;

}

return TRUE;

}

int selectSet(int u[], Subsets subsets[]) {

int index = 0, selectedIndex = -1, count = 0;

double costEffectiveness = 0, prevCostEffectiveness = 0;

for (int i = 0; i < SUBSET\_SIZE; i++) {

count = 0;

for (int j = 0; j < SET\_SIZE; j++) {

if (flag[j] == FALSE) {

continue;

}

for (int k = 0; k < subsets[i].size; k++) {

if (u[j] == subsets[i].sub[k]) {

count++;

break;

}

}

}

if(subsets[i].cost == 0)

return selectedIndex;

if (count == 0) {

continue;

}

costEffectiveness = (double)count / subsets[i].cost;

if (prevCostEffectiveness < costEffectiveness) {

selectedIndex = i;

prevCostEffectiveness = costEffectiveness;

}

}

return selectedIndex;

}

void subtractSet(int u[], Subsets subsets[], int selectedIndex) {

for (int i = 0; i < SET\_SIZE; i++) {

if (flag[i] == FALSE) {

continue;

}

for (int j = 0; j < subsets[selectedIndex].size; j++) {

if (u[i] == subsets[selectedIndex].sub[j]) {

flag[i] = FALSE;

break;

}

}

}

}

void printSubset(Subsets subsets[], int selectedIndex) {

printf("선택된 집합 s%d = { %d", selectedIndex + 1, subsets[selectedIndex].sub[0]);

for (int x = 1; x < subsets[selectedIndex].size; x++) {

printf(", %d", subsets[selectedIndex].sub[x]);

}

printf(" }\n");

}

void printRemaining(int u[], Subsets subsets[], int selectedIndex) {

cout << "선택된 집합을 제외한 나머지 원소: ";

for (int i = 0; i < SET\_SIZE; i++) {

if (flag[i] == TRUE) {

cout << u[i] << " ";

}

}

cout << endl;

}

void setCover(int u[], Subsets subsets[]) {

flagInit();

while (!isEmpty()) {

int selectedIndex = selectSet(u, subsets);

if (selectedIndex == -1) {

cout << "더이상 커버링할 수 있는 부분집합이 없습니다." << endl;

printRemaining(u, subsets, selectedIndex);

break;

}

subtractSet(u,subsets, selectedIndex);

printSubset(subsets, selectedIndex);

}

}

void printAllsets(int u[], Subsets subsets[]) {

printf("전체집합: \n");

printf("{ %d", u[0]);

for (int i = 1; i < SET\_SIZE; i++) {

printf(", %d ", u[i]);

}

printf(" }\n");

printf("모든 부분집합:\n");

for (int i = 0; i < SUBSET\_SIZE; i++) {

printf("s%d = { %d", i + 1, subsets[i].sub[0]);

for (int j = 1; j < subsets[i].size; j++) {

printf(", %d", subsets[i].sub[j]);

}

printf(" }");

printf(" (비용: %d)\n", subsets[i].cost);

}

printf("-------------------------\n");

}

int main(void) {

int u[SET\_SIZE] = { 1,2,3,4,5,6,7,8,9,10 };

int s1[] = { 1,2,3,8 };

int s2[] = { 1,2,3,4,8 };

int s3[] = { 1,2,3,4 };

int s4[] = { 2,3,4,5,7,8 };

int s5[] = { 4,5,6,7 };

int s6[] = { 5,6,7,9,10 };

int s7[] = { 4,5,6,7 };

int s8[] = { 1,2,4,8 };

int s9[] = { 6,9 };

int s10[] = { 6,10 };

Subsets subsets[SUBSET\_SIZE];

int\* sub[SUBSET\_SIZE] = { s1, s2, s3, s4, s5, s6, s7, s8, s9, s10 };

int subSize[SUBSET\_SIZE] = { 4,5,4,6,4,5,4,4,2,2 };

int subCost[SUBSET\_SIZE] = { 6,10,4,12,4,8,4,4,3,4 };

subsetIncoding(subsets, sub, subSize, subCost);

printAllsets(u, subsets);

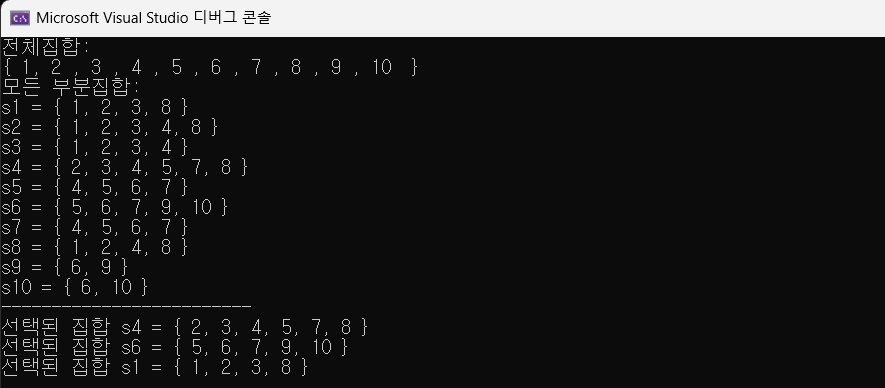
setCover(u, subsets);

return 0;

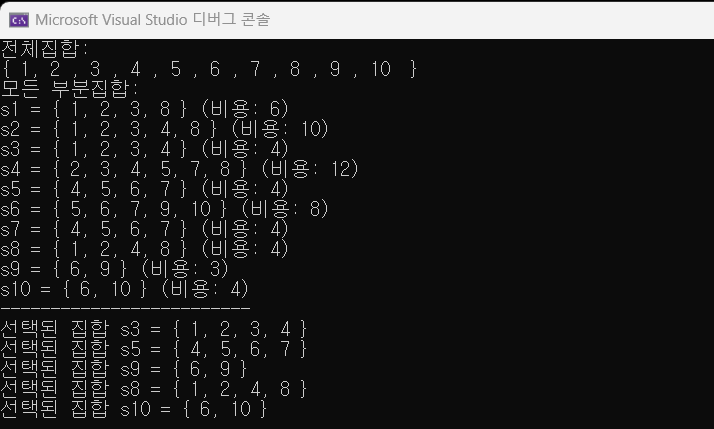
}

1. **실행화면 캡처**

**2-1. 문제1**



**2-2. 문제2**



1. **고찰**

**3-1. 문제1**

* 그리디 알고리즘을 활용하여 가장 많이 전체집합에서 가장 많이 커버하는 부분집합을 선택하도록 설계하였습니다
* 에러 핸들링 - 입력조건이 바꼈을때 에러
  + 만약, 부분집합만으로 커버할 수 없을 경우 break하고 메세지
* 전체집합, 부분집합을 메인함수의 변수로 선언
  + 전체집합을 전역변수로 선언하고 부분집합을 처음부터 배열로 만들었으면 좀 더 효율적이었을 것같은데 리펙토링할 시간이 부족합니다

**3-2. 문제2**

* 그리디 알고리즘을 활용하여 가성비(비용대비 커버링 수)가 가장 높은 부분집합을 선택하도록 설계하였습니다
* 에러 핸들링 - 입력조건이 바꼈을때 에러
  + 만약, 부분집합만으로 커버할 수 없을 경우 break하고 메세지
  + 만약, 가성비가 0이라면 break 하고 메세지

**3-3. 그리디 알고리즘 문제점**

그리디 알고리즘 문제점

* 문제1
  + 만약, 문제1에서 첫번째로 커버링 수가 높은 부분집합을 제외하고 그리디 알고리즘을 적용하면 더 효율적일 경우
    - 가장 높은것부터 선택하는 것이 효율적이지 않을 수 있습니다
  + 문제1에서 중복 문제
    - 커버링 수가 같은 부분집합이 있을경우, 우선순위에 따라 출력됩니다
* 문제2
  + 문제2도 같은 문제를 가지고있습니다