HW 3. Greedy Algorithm 구현

1. CoinChange algorithm 구현

○ 2780원의 거스름돈을 최소 동전 수로 받는 CoinChange 알고리즘 구현
- 동전은 500원, 100원, 50원, 10원짜리가 있음

○ 결과를 다음과 같이 출력
- 2780 Won - 500 Won: 5, 100 Won: 2, 50 Won: 1, 10 Won: 3

○ python을 이용하여 구현

2. Kruskal algorithm 구현

○ Input : 수업에서 사용한 그래프 (Kruskal algorithm - 강의자료 15페이지에 있는 그래프)
- Input 형태: vertex a 와 vertex b의 간선의 가중치가 c라고 하면 (a, b, c)로 표현

○ Output : Minimum spanning tree
- Output 형태: Input 형태와 마찬가지로 vertex a 와 vertex b의 간선의 가중치가 c라고 하면 (a, b, c)로 표현

※ 결과는 다음 값이 나와야 함 (a:0, b:1, c:2, d:3, e:4, f:5로 정의)
- (1, 2, 1)
  (2, 5, 1)
  (0, 3, 2)
  (3, 4, 3)
  (1, 3, 4) (각 벡터는 Kruskal algorithm에 의해 계산된 순서대로 표현)
- 알파벳이 앞선 글자가 먼저 표현됨 (예: (a, b, 1)은 가능, (b, a, 1)은 불가능)

○ C++을 이용하여 구현

○ running time 출력

3. Prim algorithm 구현

○ Input : 수업에서 사용한 그래프 (Prim algorithm - 강의자료 31페이지에 있는 그래프)
- Input 형태: vertex a -> vertex b의 간선의 가중치가 c라고 하면 (a, b, c)로 표현

○ Output : Minimum spanning tree

- Output 형태: Input 형태와 마찬가지로 vertex a -> vertex b의 간선의 가중치가 c라고 하면 (a, b, c)로 표현

※ 결과는 다음 값이 나와야 함 (a:0, b:1, c:2, d:3, e:4, f:5로 정의)

- (2, 1, 1)
  (2, 5, 1)
  (1, 0, 3)
  (0, 3, 2)
  (0, 4, 4)

○ Java를 이용하여 구현 (AI학과는 Java로 구현하면 100점 만점, C++로 구현하면 70점 만점)

○ running time 출력

4. Dijkstra Shortest Path algorithm 구현

○ input: Chapter 4-1, 52p의 그래프

○ output: 아래의 형태로 각 도시 간 거리 출력 (안에 거리는 실제와 다를 수 있음)

	서울 Seoul	인천 Incheon	수원 Suwon	대전 Daejeon	전주 Jeonju	광주 Gwangju	대구 Daegu	울산 Ulsan	부산 Busan
서울 Seoul		40.2	41.3	154	232.1	320.4	297	407.5	432
인천 Incheon			54.5	174	253.3	351.6	317.6	447	453
수원 Suwon				132.6	189.4	299.6	268.1	356	390.7
대전 Daejeon					96.9	185,2	148.7	259.1	283.4
전주 Jeonju						105.9	219.7	331.1	322.9
광주 Gwangju							219.3	329.9	268
대구 Daegu								111.1	135.5
울산 Ulsan									52.9
부산 Busan									

○ C를 이용하여 구현 (선은 그리지 않아도 됨)

○ O(nlogn)의 복잡도로 구현

○ running time 출력

5. Fractional Knapsack algorithm 구현

○ input: Chapter 4-2, 6p의 그림

○ output: 아래와 같은 형태로 출력 (A: 백금, B: 금, C: 은, D: 주석)

Goods	Weight of goods in knapsack	Value of goods in knapsack
A		
B		
C		
D		
Sum		

○ C++을 이용하여 구현 (선은 그리지 않아도 됨)

6. Set cover algorithm 구현 (Optimal solution도 함께 구현)

○ input: Chapter 4-2, 11p의 그림 (아래에서 오른쪽 그림)

- 각 연결된 마을끼리는 거리가 15분으로 동일하다고 가정

○ output: Chapter 4-2, 22p에 나와 있는 집합 C를 출력

※ Optimal solution도 구현하여 running time 비교

○ Java를 이용하여 구현 (AI학과는 Java로 구현하면 100점 만점, C로 구현하면 70점 만점)

7. Job Scheduling algorithm 구현

○ input: Chapter 4-2, 27p에 있는 t1-t7

○ output: 아래와 같은 형태로 출력

time	0	1	2	3	4	5	6	7	8	9
Machine 3		t3	t3	t3	t3	t3	t4	t4	t4	t4
Machine 2		t7	t7	t7	t7	t7	t7	t6	t6	
Machine 1	t5	t5		t2	t2	t2	t2	t2	t1	

○ python을 이용하여 구현

8. Huffman compression algorithm

○ input: Huffman_input.txt 파일

○ output1: Huffman code (‘A’ = ?, ‘T’ = ?, ‘G’ = ?, ‘C’ = ?)

○ output2: 압축된 101100100011101010100에 대해 압축을 해제한 결과

- Chapter 4-2, 43p 참조

○ C를 이용하여 구현

■ 제출 목록

- source code
- 보고서 (결과 출력본 포함)
- 모든 제출 자료는 zip 파일로 제출 (과목, 분반, 학번, 이름을 zip 파일명에 명시)
- 4장 종료 1주 일 후의 오후 11:59

※ 유의사항
- 모든 시뮬레이션은 Linux 환경에서 구현
- 각 알고리즘 별로 100점이고 총 800점 만점