12주차 2차시 그리디 알고리즘의 이해 II

【학습목표】

1. 부분 배낭 문제 알고리즘을 이해할 수 있다.
2. 집합 커버 문제 알고리즘을 이해할 수 있다.

학습내용1 : 부분 배낭 문제

1. 부분 배낭 (Fractional Knapsack) 문제

배낭 (Knapsack) 문제는 n개의 물건이 있고, 각 물건은 무게와 가치를 가지고 있으며, 배낭이 한정된 무게의 물건들을 담을 수 있을 때, 최대의 가치를 갖도록 배낭에 넣을 물건들을 정하는 문제 이다. 원래 배낭 문제는 물건을 통째로 배낭에 넣어야 되지만, 부분 배낭 (Fractional Knapsack) 문제는 물건을 부분적으로 담는 것을 허용한다.
부분 배낭 문제에서는 물건을 부분적으로 배낭에 담을 수 있으므로, 최적해를 위해서 '욕심을 내어' 단위 무게 당 가장 값나가는 물건을 배낭에 넣고, 계속해서 그 다음으로 값나가는 물건을 넣는다. 그런데 만일 그 다음으로 값나가는 물건을 통째로'배낭에 넣을 수 없게 되면, 배낭에 넣을 수 있을 만큼만 물건을 부분적으로 배낭에 담는다.

* FractionalKnapsack

입력: n개의 물건, 각 물건의 무게와 가치, 배낭의 용량 C
출력: 배낭에 담은 물건 리스트 L과 배낭에 담은 물건의 가치 합 v
1. 각 물건에 대해 단위 무게 당 가치를 계산한다.
2. 물건들을 단위 무게 당 가치를 기준으로 내림차순으로 정렬하고, 정렬된 물건 리스트를 S라고 하자.
3. L=Ø, w=0, v=0
// L은 배낭에 담을 물건 리스트, w는 배낭에 담긴 물건들의 무게의 합, v는 배낭에 담긴 물건들의 가치의 합이다.
4. S에서 단위 무게 당 가치가 가장 큰 물건 x를 가져온다.
5. while ((w+x의 무게) ≤ C) {
6. x를 L에 추가시킨다.
7. w = w +x의 무게
8. v = v +x의 가치
9. x를 S에서 제거한다.
10. S에서 단위 무게 당 가치가 가장 큰 물건 x를 가져온다.
}
11. if ((C-w) > 0) { // 배낭에 물건을 부분적으로 담을 여유가 있으면
12. 물건 x를 (C-w)만큼만 L에 추가한다.
13. v = v +(C-w)만큼의 x의 가치
}
14. return L, v

- Line 1~2: 각 물건의 단위 무게 당 가치를 계산하여, 이를 기준으로 물건들을 내림차순으로 정렬한다.
- Line 5~10의 while-루프를 통해서 다음으로 단위 무게 당 값나가는 물건을 가져다 배낭에 담고, 만일 가져온 물건을 배낭에 담을 경우 배낭의 용량이 초과되면, (즉, while-루프의 조건이 '거짓'이 되면) 가져온 물건을 '통째로' 담을 수 없게 되어 루프를 종료한다.
- Line 11: 현재까지 배낭에 담은 물건들의 무게 w가 배낭의 용량 C 보다 작으면, (즉, if- 조건이 '참'이면) line 12~13에서 해당 물건을 (C-w)만큼만 배낭에 담고, (C-w)만큼의 x의 가치를 증가시킨다.
- Line 14: 최종적으로 배낭에 담긴 물건들의 리스트 L과 배낭에 담긴 물건들의 가치 합 v 를 리턴 한다.

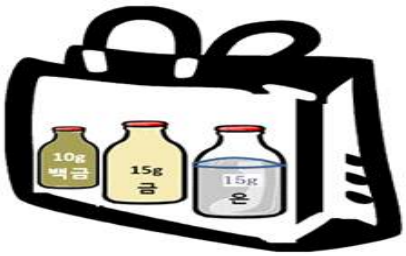
- 4개의 금속 분말이 다음의 그림과 같이 있다.
  배낭의 최대 용량이 40그램일 때,
  FractionalKnapsack 알고리즘의 수행 과정
- Line 1~2의 결과: S=[백금, 금, 은, 주석]

물건	단위 그램당 가치
백금	6만원
금	5만원
은	4천원
주석	1천원

- Line 3: L=Ø, w=0, v=0로 각각 초기화한다.
- Line 4: S=[백금, 금, 은, 주석]로부터 백금을 가져온다.
- Line 5: while-루프의 조건 ((w+백금의 무게) ≤ C) = ((0+10)<40)이 '참'이다.
- Line 6: 백금을 배낭 L에 추가시킨다. 즉, L=[백금]이 된다.
- Line 7: w = w(백금의 무게) = 0+10g = 10g
- Line 8: v = v(백금의 가치) = 0+60만원 = 60만원
- Line 9: S에서 백금을 제거한다. S=[금, 은, 주석]
- Line 10: S에서 금을 가져온다.

- Line 5: while-루프의 조건 ((w+금의 무게) ≤ C) = ((10+15)<40)이 '참'이다.
- Line 6: 금을 배낭 L에 추가시킨다. L=[백금, 금]
- Line 7: w = w+(금의 무게) = 10g+15g = 25g
- Line 8: v = v+(금의 가치) = 60만원+75만원 = 135만원
- Line 9: S에서 금을 제거한다. S=[은, 주석]
- Line 10: S에서 은을 가져온다.
- Line 5: while-루프의 조건 ((w+은의 무게) ≤ C) = ((25+25)<40)이 '거짓'이므로 루프 를 종료한다.

- Line 11: if-조건 ((C-w) >0)이 '참'이다. 즉, 40-25 = 15 > 0이기 때문이다.
- Line 12: 따라서 은을 C-w=(40-25)=15g만큼만 배낭L에 추가시킨다.
- Line 13: v = v+(15g x 4천원/g) = 135만원+6만원 = 141만원
- Line 14: 배낭 L=[백금 10g, 금 15g, 은 15g]과 가치의 합 v = 141만원을 리턴 한다.



2. 시간 복잡도

- Line 1: n개의 물건 각각의 단위 무게 당 가치를 계산하는 데는 O(n) 시간 걸리고, line 2에서 물건의 단위 무게 당 가치에 대해서 내림차순으로 정렬하기 위해 O(nlogn) 시간이 걸린다.
- Line 5~10의 while-루프의 수행은 n번을 넘지 않으며, 루프 내부의 수행은 O(1) 시간이 걸린다. 또한 line 11~14도 각각 O(1) 시간 걸린다.
- 따라서 알고리즘의 시간 복잡도는 O(n)+O(nlogn)+nxO(1)+O(1) = O(nlogn)이다

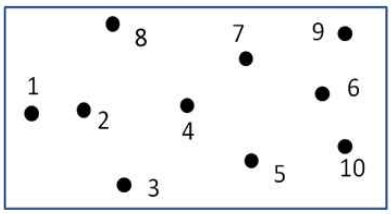
학습내용2 : 집합 커버 문제

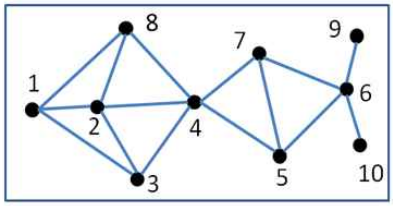
1. 집합 커버 문제의 개요

n개의 원소를 가진 집합인 U가 있고, U의 부분 집합들을 원소로 하는 집합 F가 주어질 때, F의 원소들인 집합들 중에서 어떤 집합들을 선택하여 합집합하면 U와 같게 되는가?
집합 커버 (cover) 문제는 F에서 선택하는 집합들의 수를 최소화하는 문제이다.

- 예제: 신도시계획학교배치
- 10개의 마을이 신도시에 있다.
- 이때 아래의 2가지 조건이 만족되도록 학교의 위치를 선정하여야 한다고 가정하자.

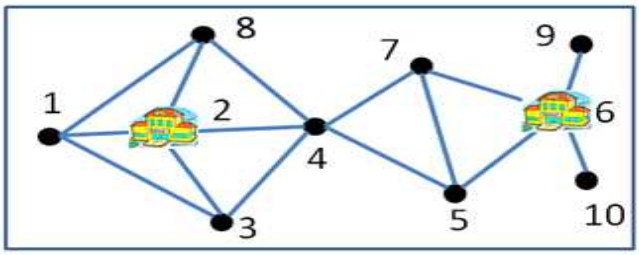
☞ 학교는 마을에 위치해야 한다.
☞ 등교 거리는 걸어서 15분 이내이어야 한다.





등교 거리가 15분 이내인 마을 간의 관계

- 어느 마을에 학교를 신설해야 학교의 수가 최소로 되는가?
- 2번 마을에 학교를 만들면 마을 1, 2, 3, 4, 8의 학생들이 15분 이내에 등교할 수 있고 (즉, 마을 1, 2, 3, 4, 8이 '커버'되고),
- 6번 마을에 학교를 만들면 마을 5, 6, 7, 9, 10이 커버된다.
- 즉, 2번과 6번이 최소이다.



- 신도시 계획 문제를 집합 커버 문제로 변환:
- 여기서 S_i는 마을 i에 학교를 배치했을 때 커버되는 마을의 집합이다.

U= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } // 신도시의 마을 10개

F= {S1, S2, S3, S4, S5, S6, S7, S8, S9, S10}

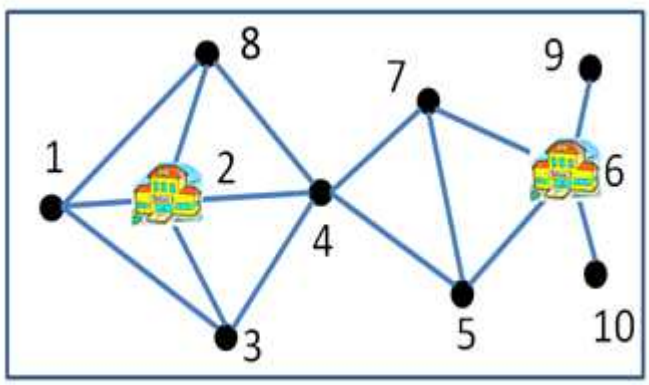
S1= {1, 2, 3, 8 }　　S5= {4, 5, 6, 7}　S9= {6, 9 }

S2= {1, 2, 3, 4, 8 }　　S6= {5, 6, 7, 9, 10 }　　S10= {6, 10 }

S3= {1, 2, 3, 4 }　　S7= {4, 5, 6, 7 }

S4= {2, 3, 4, 5, 7, 8 }　S8= {1, 2, 4, 8 }

- S_i 집합들 중에서 어떤 집합들을 선택 하여야 그들의 합집합이 U와 같은가?
  단, 선택된 집합의 수는 최소이어야 한다.
- 이 문제의 답은$S_2 \cup S_6$ = {1, 2, 3, 4, 8 }∪{5, 6, 7, 9, 10 } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} = U이다.



- 집합 커버 문제의 최적해는 어떻게 찾아야 할까
- F에 n개의 집합들이 있다고 가정해보자.
- 가장 단순한 방법은 F에 있는 집합들의 모든 조합을 1개씩 합집합 하여 U가 되는지 확인 하고, U가 되는 조합의 집합 수가 최소인 것을 찾는 것이다.

- 예를 들면, F= {S_1, S_2, S_3}일 경우, 모든 조합이란, S_1, S_2, S_3, $S_1 \cup S_2$, $S_1 \cup S_3$, $S_2 \cup S_3$, $S_1 \cup S_2 \cup S_3$이다.
→ 집합이 1개인 경우 3개 = ${}_5C_1$,
→ 집합이 2개인 경우 3개 = ${}_3C_2$,
→ 집합이 3개인 경우 1개 = ${}_3C_3$이다.
→ 총합은 3+3+1= 7 = 2^3-1 개이다.

- n개의 원소가 있으면 (2^n-1)개를 다 검사하여야 하고, n이 커지면 최적해를 찾는 것은
실질적으로 불가능하다.
- 이를 극복하기 위해서는 최적해를 찾는 대신에 최적해에 근접한 근사해 (approximation solution)를 찾는 것이다.

* SetCover 알고리즘

입력: U, F= {Si}, i=1,…,n
출력: 집합 커버 C
1. C=Ø
2. while (U≠Ø) do {
3. U의 원소들을 가장 많이 포함하고 있는 집합S_i를 F에서 선택한다.
4. U=U-S_i
5. S_i를 F에서 제거하고, S_i를 C에 추가한다.
}
6. return C

- Line 1: C를 공집합으로 초기화시킨다.
- Line 2~5의 while-루프에서는 집합 U가 공집합이 될 때까지 수행된다.
- Line 3: '그리디'하게U와 가장 많은 수의 원소들을 공유하는 집합 S_i를 선택한다.
- Line 4: S_i의 원소들을 U에서 제거한다. 왜냐하면 S_i의 원소들은 커버된 것이기 때문이다. - 따라서 U는 아직 커버되지 않은 원소들의 집합이다.
- Line 5: S_i를 F로부터 제거하여, S_i가 line 3에서 더 이상 고려되지 않도록 하며, S_i를 집합 커버 C에 추가한다.
- Line 6: C를 리턴한다

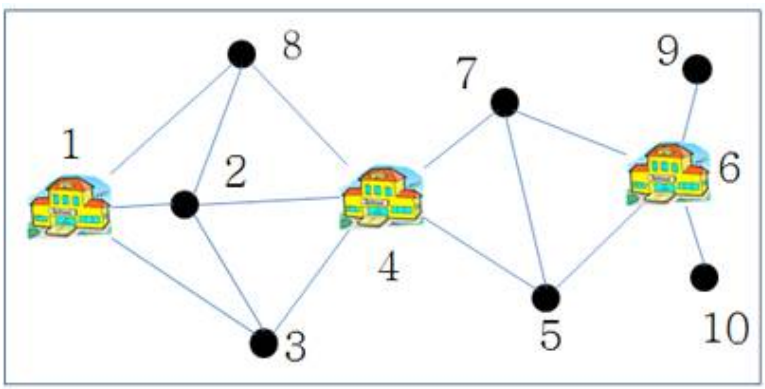
* SetCover 알고리즘의 수행 과정
U= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
F= {S1, S2, S3, S4, S5, S6, S7, S8, S9, S10}
S1= {1, 2, 3, 8 } S6= {5, 6, 7, 9, 10 }
S2= {1, 2, 3, 4, 8 } S7= {4, 5, 6, 7 }
S3= {1, 2, 3, 4 } S8= {1, 2, 4, 8 }
S4= {2, 3, 4, 5, 7, 8 } S9= {6, 9 }
S5= {4, 5, 6, 7 } S10= {6, 10 }

- Line 1: C=Ø로 초기화
- Line 2: while-조건 (U≠Ø)=({1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } ≠Ø)이 '참'이다.
- Line 3: U의 원소를 가장 많이 커버하는 집합인 S4= {2, 3, 4, 5, 7, 8 }을 F에서 선택한다.
- Line 4: U = U - S4 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } - {2, 3, 4, 5, 7, 8 } = {1, 6, 9, 10 }
- Line 5: S4를 F에서 제거하고, 즉, F = { S1, S2, S3, S4, S5, S6, S7, S8, S9, S10} - {S4} = {S1, S2, S3, S5, S6, S7, S8, S9, S10}가 되고, S4를 C에 추가한다. 즉, C = {S4}이다.

- Line 2: while-조건 (U≠Ø) = ({1, 6, 9, 10 } ≠Ø)이 '참'이다.
- Line 3: U의 원소를 가장 많이 커버하는 집합인 S6= {5, 6, 7, 9, 10 }을 F에서 선택한다.
- Line 4: U = U - S4 = {1, 6, 9, 10 } - {5, 6, 7, 9, 10 } = {1 }
- Line 5: S6을 F에서 제거하고, 즉, F= {S1, S2, S3, S5, S6, S7, S8, S9, S10} - {S6} = {S1, S2, S3, S5, S7, S8, S9, S10}이 되고, S6을 C에 추가한다. 즉, C = {S4, S6}이다.

- Line 2: while-조건 (U≠Ø) = ({1 }≠Ø)이 '참'이다.
- Line 3.: U의 원소를 가장 많이 커버하는 집합인 S1= {1, 2, 3, 8 }을 F에서 선택한다. S1 대신에 S2, S3, S8 중에서 어느 하나를 선택해도 무방하다.
- Line 4: U = U - S1 = {1 } - {1, 2, 3, 8 }=Ø
- Line 5: S1을 F에서 제거하고, 즉, F= {S1, S2, S3, S5, S6, S7, S8, S9, S10} - {S1} = {S2, S3, S5, S7, S8, S9, S10}이 되고, S1을 C에 추가한다. 즉, C = {S1, S4, S6}이다.
- Line 2: while-조건 (U≠Ø) = (Ø≠Ø)이 '거짓'이므로, 루프를 끝낸다.
- Line 6: C= {S1, S4, S6}을 리턴 한다.

* SetCover 알고리즘의 최종해



2. 시간복잡도

- 먼저 while-루프가 수행되는 횟수는 최대 n번이다. 왜냐하면 루프가 1번 수행될 때마다 집합 U의 원소 1개씩만 커버된다면, 최악의 경우 루프가 n번 수행되어야 하기 때문이다.
- 루프가 1번 수행될 때의 시간복잡도를 살펴보자.
- Line 2의 while-루프 조건 (U≠∅)을 검사는 O(1) 시간이 걸린다. 왜냐하면 U의 현재 원소 수를 위한 변수를 두고 그 값이 0인지를 검사하면 되기 때문이다.

3. 응용

- 도시 계획 (City Planning)에서 공공 기관 배치하기
- 경비 시스템: 미술관, 박물관, 기타 철저한 경비가 요구되는 장소 (Art Gallery 문제)의 CCTV 카메라의 최적 배치
- 컴퓨터 바이러스 찾기: 알려진 바이러스들을 '커버'하는 부분 스트링의 집합 - IBM에서 5,000개의 알려진 바이러스들에서 9,000개의 부분 스트링을 추출하였고, 이 부분 스트링의 집합 커버를 찾았는데, 총 180개의 부분 스트링이었다. 이 180개로 컴퓨터 바이러스의 존재를 확인하는데 성공하였다.
- 대기업의 구매 업체 선정: 미국의 자동차 회사인 GM은 부품 업체 선정에 있어서 각 업체가 제시하는 여러 종류의 부품들과 가격에 대해, 최소의 비용으로 구입하려고 하는 부품들을 모두 '커버'하는 업체를 찾기 위해 집합 문제의 해를 사용하였다.
- 기업의 경력 직원 고용: 예를 들어, 어느 IT 회사에서 경력 직원들을 고용하는데, 회사에서 필요로 하는 기술은 알고리즘, 컴파일러, 앱 (App) 개발, 게임 엔진, 3D 그래픽스, 소셜 네트워크서비스, 모바일 컴퓨팅, 네트워크, 보안이고, 지원자들은 여러 개의 기술을 보유하고 있다. 이 회사가 모든 기술을 커버하는 최소 인원을 찾으려면, 집합 문제의 해를 사용하면 된다.
- 그 외에도 비행기 조종사 스케줄링 (Flight Crew Scheduling), 조립 라인 균형화 (Assembly Line Balancing), 정보 검색 (Information Retrieval) 등에 활용된다.

【학습정리】

1. 부분 배낭 (Fractional Knapsack) 문제
배낭 (Knapsack) 문제는 n개의 물건이 있고, 각 물건은 무게와 가치를 가지고 있으며, 배낭이 한정된 무게의 물건들을 담을 수 있을 때, 최대의 가치를 갖도록 배낭에 넣을 물건들을 정하는 문제 이다. 원래 배낭 문제는 물건을 통째로 배낭에 넣어야 되지만, 부분 배낭 (Fractional Knapsack) 문제는 물건을 부분적으로 담는 것을 허용한다.
부분 배낭 문제에서는 물건을 부분적으로 배낭에 담을 수 있으므로, 최적해를 위해서 '욕심을 내어' 단위 무게 당 가장 값나가는 물건을 배낭에 넣고, 계속해서 그 다음으로 값나가는 물건을 넣는다. 그런데 만일 그 다음으로 값나가는 물건을 통째로'배낭에 넣을 수 없게 되면, 배낭에 넣을 수 있을 만큼만 물건을 부분적으로 배낭에 담는다.

2. 집합 커버 문제
n개의 원소를 가진 집합인 U가 있고, U의 부분 집합들을 원소로 하는 집합 F가 주어질 때, F의 원소들인 집합들 중에서 어떤 집합들을 선택하여 합집합하면 U와 같게 되는가?
집합 커버 (cover) 문제는 F에서 선택하는 집합들의 수를 최소화하는 문제이다.