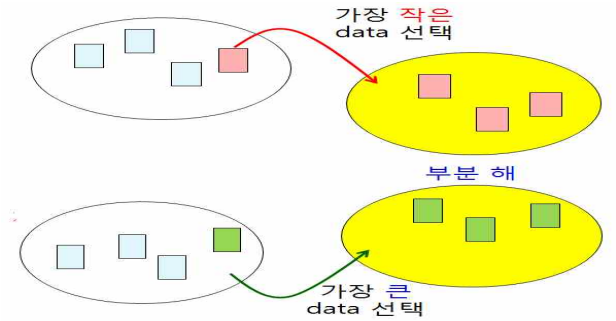
12주차 3차시 그리디 알고리즘의 이해 Ⅲ

【학습목표】

1. 동전 거스름돈 알고리즘을 이해할 수 있다.
2. 작업 스케줄링 알고리즘을 이해할 수 있다.

지난 강의 정리

그리디 알고리즘은 욕심쟁이 알고리즘, 탐욕적 방법, 탐욕 알고리즘 등으로 불리기도 한다.
그리디 알고리즘은 (입력) 데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심내어' 최소값 또는 최대값을 가진 데이터를 선택하여 최적해를 구하는 알고리즘으로 최소 신장 트리, 최단경로, 부분 배낭문제, 집합커버, 동전 거스름돈, 작업 스케줄링 등이 있다.

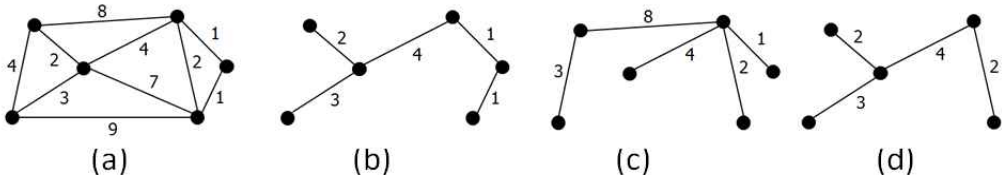


그리디 알고리즘 수행 과정

- 그리디 알고리즘은 일단 한번 선택하면, 이를 절대로 번복하지 않는다.
- 즉, 선택한 데이터를 버리고 다른 것을 취하지 않는다.
- 이러한 특성 때문에 대부분의 그리디 알고리즘들은 매우 단순하며, 또한 제한적인 문제들만이 그리디 알고리즘으로 해결된다.

* 최소 신장 트리

- 최소 신장 트리 (Minimum Spanning Tree): 주어진 가중치 그래프에서 사이클이 없이 모든 점들을 연결시킨 트리들 중 선분들의 가중치 합이 최소인 트리
- (a)주어진 가중치 그래프, (b) 최소 신장 트리 (c),(d)는 최소 신장 트리 아님
  ☞ (c)는 가중치의 합이 (b)보다 크고, (d)는 트리가 주어진 그래프의 모든 노드를 포함하지 않고 있다.



- 주어진 그래프의 신장 트리를 찾으려면 사이클이 없도록 모든 점을 연결시키면 된다. 그래프의 점의 수가 n이면, 신장 트리에는 정확히 (n-1)개의 선분이 있다.
- 트리에 선분을 하나 추가시키면, 반드시 사이클이 만들어진다.
- 최소 신장 트리를 찾는 대표적인 그리디 알고리즘으로는 크러스컬 (Kruskal)과 프림 (Prim) 알고리즘이 있다.

* 최단 경로 찾기

- 최단 경로 (Shortest Path) 문제는 주어진 가중치 그래프에서 어느 한 출발점에서 또 다른 도착점까지의 최단 경로를 찾는 문제이다.
- 최단 경로를 찾는 가장 대표적인 알고리즘은 다이스트라 (Dijkstra) 최단 경로 알고리즘이며, 이 또한 그리디 알고리즘이다
  ☞ 단일 출발점 최단 경로.(다이스트라 알고리즘)
  ☞ 모든 쌍 최단 경로.(플로이드 알고리즘)

* 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.

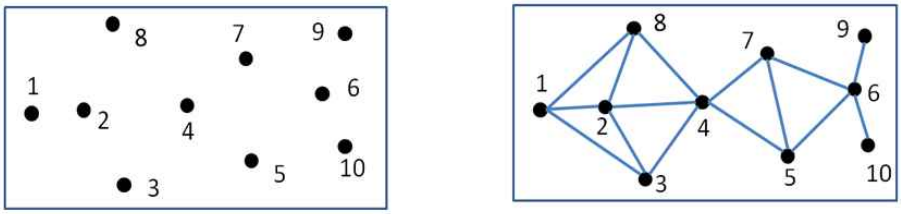
* 집합 커버 문제

n개의 원소를 가진 집합인 U가 있고, U의 부분 집합들을 원소로 하는 집합 F가 주어질 때, F의 원소들인 집합들 중에서 선택하는 집합들의 수를 최소화하는 알고리즘.

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

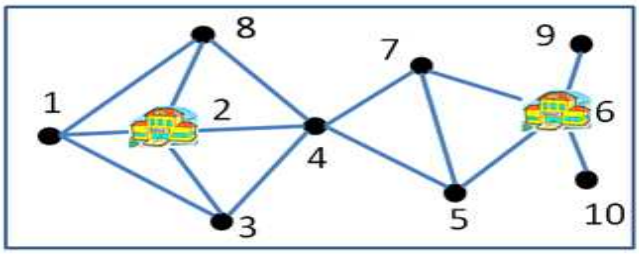
☞ 학교는 마을에 위치해야 한다.

☞ 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.



학습내용1 : 동전 거스름돈

1. 동전 거스름돈 (Coin Change) 문제

- 동전 거스름돈 (Coin Change) 문제를 해결하는 가장 간단하고 효율적인 방법은 남은 액수를 초과하지 않는 조건하에 '욕심내어' 가장 큰 액면의 동전을 취하는 것이다.
- 다음은 동전 거스름돈 문제의 최소 동전 수를 찾는 그리디 알고리즘이다. 단, 동전의 액면은 500원, 100원, 50원, 10원, 1원이다.

* 알고리즘

CoinChange
입력: 거스름돈 액수 W
출력: 거스름돈 액수에 대한 최소 동전 수
1. change=W, n500=n100=n50=n10=n1=0
// n500, n100, n50, n10, n1은 각각의 동전 카운트
2. while (change ≥ 500)
change = change-500, n500++ // 500원짜리 동전 수를 1 증가
3. while (change ≥ 100)
change = change-100, n100++ // 100원짜리 동전 수를 1 증가
4. while (change ≥ 50)
change = change-50, n50++ // 50원짜리 동전 수를 1 증가
5. while (change ≥ 10)
change = change-10, n10++ // 10원짜리 동전 수를 1 증가
6. while (change ≥ 1)
change = change-1, n1++ // 1원짜리 동전 수를 1 증가
7. return (n500+n100+n50+n10+n1) // 총 동전 수를 리턴 한다.

- Line 1: change를 입력인 거스름돈 액수 W로 놓고, 각 동전 카운트를 n500 = n100 = n50 = n10 = n1 = 0으로 초기화
- Line 2~6: 차례로 500원, 100원, 50원, 10원, 1원짜리 동전을 각각의 while-루프를 통해 현재 남은 거스름돈 액수인 change를 넘지 않는 한 계속해서 같은 동전으로 거슬러 주고, 그 때마다 각각의 동전 카운트를 1 증가시킨다.
- Line 7에서는 동전 카운트들의 합을 리턴 한다.

- CoinChange 알고리즘은 남아있는 거스름돈인 change에 대해 가장 높은 액면의 동전을 거스르며, 500원짜리 동전을 처리하는 line 2에서는 100원짜리, 50원짜리, 10원짜리, 1원짜리 동전을 몇 개씩 거슬러 주어야 할 것인지에 대해서는 전혀 고려하지 않는다.
- 이것이 바로 그리디 알고리즘의 근시안적인 특성이다.

- 거스름돈 760원에 대해 CoinChange 알고리즘이 수행되는 과정을 살펴보자.
- Line 1: change = 760, n500 = n100 = n50= n10 = n1 = 0으로 초기화된다.
- Line 2: change가 500보다 크므로 while-조건이 '참'이어서 change = change-500 = 760-500 = 260이 되고, n500=1이 된다.
- 다음은 change가 500보다 작으므로 line 2의 while-루프는 더 이상 수행되지 않는다.

- Line 3: change〉 100이므로 while-조건이 '참'이 되어서 change = change-100 = 260-100 = 160이 되고, n100 = 1이 된다. 다음도 change〉 100이므로 while-조건이 역시 '참'이라서 change = change-100 = 160-100 = 60이 되고, n100=2가 된다.
- 그러나 그 다음엔 change가 60이므로 100보다 작아서 while-루프는 수행되지 않는다.

- Line 4: change〉 50이므로 while-조건이 '참'이라서 change = change-50 = 60-50 = 10이 되고, n50=1이 된다. 다음은 change가 50보다 작으므로 while-루프는 수행되지 않는다.
- Line 5: change〉10이므로 while-조건이 '참'이라서 change = change-10 = 10-10 = 0이 되고, n10=1이 된다. 그 다음엔 change가 10보다 작으므로 while-루프는 수행되지 않는다.

- Line 6: change=0이므로 while-조건이 '거짓'이 되어 while-루프는 수행되지 않는다.
- Line 7에서는 n500+n100+n50+n10+n1 = 1+2+1+1+0 = 5를 리턴한다.

- 그런데 만일 한국은행에서 160원짜리 동전을 추가로 발행한다면, CoinChange 알고리즘이 항상 최소 동전 수를 계산할 수 있을까?
- 거스름돈이 200원이라면, CoinChange 알고리즘은 160원짜리 동전 1개와 10원짜리 동전 4개로서 총 5개를 리턴한다.

CoinChange 알고리즘의 결과

최소 동전의 거스름돈

- CoinChange 알고리즘은 항상 최적의 답을 주지 못한다.

학습내용2 : 작업 스케줄링

1. 작업 스케줄링 (Task Scheduling) 문제

- 기계에서 수행되는 n개의 작업 t1, t2, …, tn이 있고, 각 작업은 시작시간과 종료시간이 있다.
- 작업 스케줄링 (Task Scheduling) 문제는 작업의 수행 시간이 중복되지 않도록 모든 작업을 가장 적은 수의 기계에 배정하는 문제이다.
- 작업 스케줄링 문제는 학술대회에서 발표자들을 강의실에 배정하는 문제와 같다.
  발표= '작업', 강의실= '기계'

- 작업 스케줄링 문제에 주어진 문제 요소
→ 작업의 수
→ 각 작업의 시작시간과 종료시간
→ 작업의 시작시간과 종료시간은 정해져 있으므로 작업의 길이도 주어진 것이다.
- 여기서 작업의 수는 입력의 크기이므로 알고리즘을 고안하기 위해 고려되어야 하는 직접적인 요소는 아니다.
- 그렇다면, 시작시간, 종료시간, 작업 길이에 대해 다음과 같은 그리디 알고리즘들을 생각해볼 수 있다.

- 빠른 시작시간 작업 우선 (Earliest start time first) 배정
- 빠른 종료시간 작업 우선 (Earliest finish time first) 배정
- 짧은 작업 우선 (Shortest job first) 배정
- 긴 작업 우선 (Longest job first) 배정

위의 4가지 중 첫 번째 알고리즘을 제외하고 나머지 3가지는 항상 최적해를 찾지 못한다.

* JobScheduling 알고리즘

입력: n개의 작업 t_1, t_2, …, t_n
출력: 각 기계에 배정된 작업 순서
1. 시작시간의 오름차순으로 정렬한 작업 리스트: L
2. while (L ≠Ø) {
3. L에서 가장 이른 시작시간 작업 t_i를 가져온다.
4. if (t_i를 수행할 기계가 있으면)
5. t_i를 수행할 수 있는 기계에 배정한다.
6. else
7. 새로운 기계에 t_i를 배정한다.
8. t_i를 L에서 제거한다.
   }
9. return 각 기계에 배정된 작업 순서

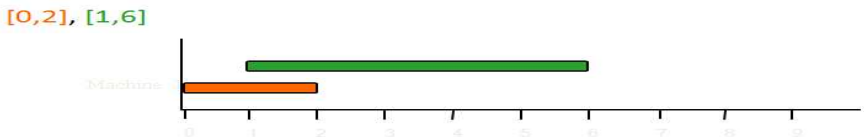
- Line 1: 시작 시간에 대해 작업을 오름차순으로 정렬
- Line 2~8의 while-루프는 L에 있는 작업이 다 배정될 때까지 수행된다.
- Line 3: L에서 가장 이른 시작시간을 가진 작업 t_i를 선택
- Line 4~5: 작업 t_i를 수행 시간이 중복되지 않게 수행할 기계를 찾아서, 그러한 기계가 있으면 t_i를 그 기계에 배정한다.
- Line 6~7: 기존의 기계들에 t_i를 배정할 수 없는 경우에는 새로운 기계에 t_i를 배정한다.
- Line 8: 작업 t_i를 L에서 제거하여, 더 이상 t_i가 작업 배정에 고려되지 않도록 한다.
- Line 9: 마지막으로 각 기계에 배정된 작업 순서를 리턴

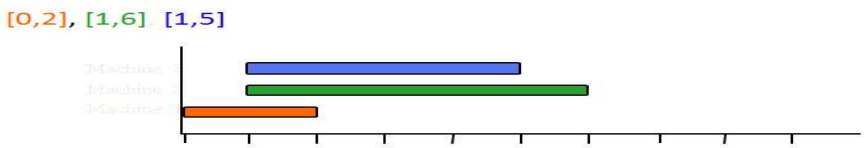
- Line 1: 시작 시간에 대해 작업을 오름차순으로 정렬
- Line 2~8의 while-루프는 L에 있는 작업이 다 배정될 때까지 수행된다.
- Line 3: L에서 가장 이른 시작시간을 가진 작업 t_i를 선택
- Line 4~5: 작업 t_i를 수행시간이 중복 되지 않게 수행할 기계를 찾아서, 그러한 기계가 있으면 t_i를 그 기계에 배정한다.
- Line 6~7: 기존의 기계들에 t_i를 배정할 수 없는 경우에는 새로운 기계에 t_i를 배정한다.
- Line 8: 작업 t_i를 L에서 제거하여, 더 이상 t_i가 작업 배정에 고려되지 않도록 한다.
- Line 9: 마지막으로 각 기계에 배정된 작업 순서를 리턴

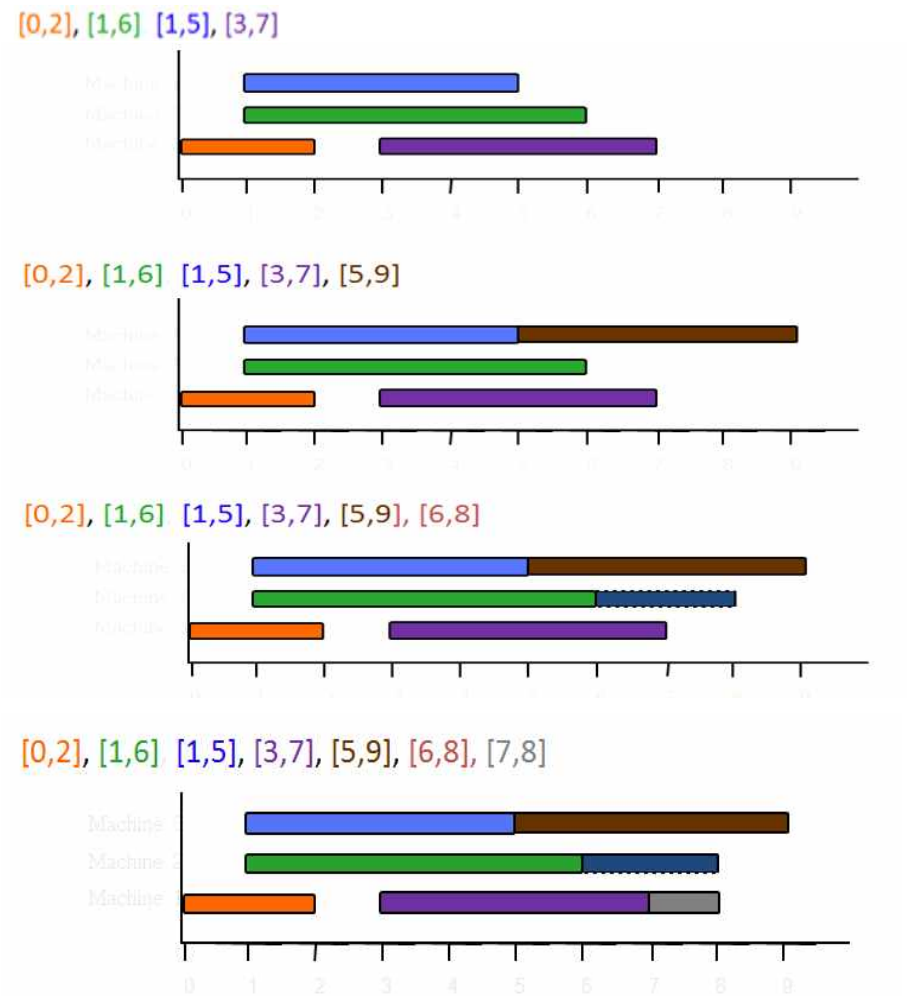
* JobScheduling 알고리즘의 수행 과정

- t_1=[7,8], t_2=[3,7], t_3=[1,5], t_4=[5,9], t_5=[0,2], t_6=[6,8], t_7=[1,6], 단, [s,f]에서, s는 작업의 시작시간이고, f는 작업의 종료시간이다.

- Line 1: 시작시간의 오름차순으로 정렬한다. 따라서 L = {[0,2], [1,6], [1,5], [3,7], [5,9], [6,8], [7,8] }이다.
- 다음은 line 2~8까지의 while-루프가 수행되면서, 각 작업이 적절한 기계에 배정되는 것을 차례로 보이고 있다.







2. 시간복잡도

- Line 1에서 n개의 작업을 정렬하는데 O(nlogn) 시간이 걸리고,
- while-루프에서는 작업을 L에서 가져다가 수행 가능한 기계를 찾아서 배정하므로 O(m) 시간이 걸린다. 단, m은 사용된 기계의 수이다.
- while-루프가 수행된 총 횟수는 n번이므로, line 2~9까지는 O(m)xn = O(mn) 시간이 걸린다.
- 따라서 JobScheduling 알고리즘의 시간복잡도는 O(nlogn)+O(mn)이다.

3. 응용

- 비즈니스 프로세싱
- 공장 생산 공정
- 강의실/세미나 룸 배정
- 컴퓨터 태스크 스케줄링 등

【학습정리】

1. 동전 거스름돈
- 동전 거스름돈 (Coin Change) 문제를 해결하는 가장 간단하고 효율적인 방법은 남은 액수를 초과하지 않는 조건하에 '욕심내어' 가장 큰 액면의 동전을 취하는 것이다.
- 다음은 동전 거스름돈 문제의 최소 동전 수를 찾는 그리디 알고리즘이다. 단, 동전의 액면은 500원, 100원, 50원, 10원, 1원이다.
- CoinChange 알고리즘은 남아있는 거스름돈인 change에 대해 가장 높은 액면의 동전을 거스르며, 500원짜리 동전을 처리하는 line 2에서는 100원짜리, 50원짜리, 10원짜리, 1원짜리 동전을 몇 개씩 거슬러 주어야 할 것인지에 대해서는 전혀 고려하지 않는다.
- 이것이 바로 그리디 알고리즘의 근시안적인 특성이다.

2. 작업 스케줄링
- 작업 스케줄링 (Task Scheduling) 문제는 작업의 수행 시간이 중복되지 않도록 모든 작업을 가장 적은 수의 기계에 배정하는 문제이다.
- 작업 스케줄링 문제는 학술대회에서 발표자들을 강의실에 배정하는 문제와 같다.
  발표= '작업', 강의실= '기계'

- 작업 스케줄링 문제에 주어진 문제 요소
→ 작업의 수
→ 각 작업의 시작시간과 종료시간
→ 작업의 시작시간과 종료시간은 정해져 있으므로 작업의 길이도 주어진 것이다.
- 여기서 작업의 수는 입력의 크기이므로 알고리즘을 고안하기 위해 고려되어야 하는 직접적인 요소는 아니다.
- 그렇다면, 시작시간, 종료시간, 작업 길이에 대해 다음과 같은 그리디 알고리즘들을 생각해볼 수 있다.

- 빠른 시작시간 작업 우선 (Earliest start time first) 배정
- 빠른 종료시간 작업 우선 (Earliest finish time first) 배정
- 짧은 작업 우선 (Shortest job first) 배정
- 긴 작업 우선 (Longest job first) 배정

위의 4가지 중 첫 번째 알고리즘을 제외하고 나머지 3가지는 항상 최적해를 찾지 못한다.