10주차 3차시 근사알고리즘 II

【학습목표】

1. 궤 채우기 알고라즘를 이해할 수 있다.
2. 작업 스케줄링 알고리즘를 이해할 수 있다.

지난 강의 정리

- 쉬운 문제, 어려운 문제
효율적인 알고리즘이 존재하는 문제를 쉬운 문제 (tractable problem) 라고 하고 효율적인 알고리즘이 존재하지 않는 문제를 어려운 문제 (intractable problem) 라고 한다.

- NP 완전 문제 (NP-complete problem)
쉬운 문제와 어려운 문제를 나누는 기준은 모든 경우를 생각해야 하는 것에 있다. 즉 모든 경우를 생각해야 하는 문제를 어려운 문제라고 하고 그렇지 않은 문제를 쉬운 문제라고 한다. 문제가 쉽다는 것을 증명하기 위해서는 효율적인 알고리즘을 개발하면 그것으로 증명이 된다. 반대로 어렵다는 것을 증명하기 위해서는 어떤 알고리즘을 이용하더라도 그 문제를 효율적으로 해결할 수 없다는 것을 증명할 필요가 있기 때문에 그다지 간단하지는 않다. 어렵다는 것이 증명되어 있는 문제는 몇 가지 있지만 그 수는 많지 않다. 그러나, 이보다 다루기 어려운 것은 쉬운지 어려운지가 (지금까지) 알려져 있지 않는 문제이다. 이들 문제에는 효율적인 알고리즘이 알려져 있지 않을 뿐만 아니라 다항식 시간의 알고리즘이 존재하지 않는다는 것도 증명되어 있지 않다. 그리고 실제로도 어려운 것을 하려고 하면 금방 이들 문제에 부딪힌다고 해도 과언이 아닐 정도로 이런 문제는 매우 많이 존재한다. 이와 같은 문제들을 NP 완전 문제 (NP-complete problem) 라고 한다.

- 부류 P
일반적인 알고리즘에서는 각 스텝에서 어떤 조작을 한다는 것이 명확하게 정해져서 기술되어 있으므로 데이터를 입력하면 어떤 조작을 어떤 순서로 한다는 것을 알 수 있다. 이러한 알고리즘을 특히 결정성 알고리즘 (deterministic algorithm) 이라고 부른다. 결정성 알고리즘에 의해 다항식 시간에 해결할 수 있는 문제를 모두 클래스 P(class P) 에 속하는 문제라고 한다.

- 부류 NP
NP 라고 하는 것은 비결정성 알고리즘 (nondeterministic algorithm) 을 이용하면 다항식 시간에 해결할 수 있는 것을 의미한다. 비결정성 알고리즘이란 개개의 스텝에서 취할 수 있는 경로가 여러 개로 나눠져 있어서 그 중에서 적당한 경로를 택해서 실행해 나가는 것이다. 경로를 택한다고 하더라도 if 문에 의한 조건 분기와 같이 어느 경로를 선택할 것인가에 대한 정보가 주어져 있는 것이 아니다. 아무튼 어느 경로든지 하나의 경로를 택해서 앞으로 실행해 나가는 수밖에 없다. 그리고 이들 경로 중 가장 적합한 경로를 잘 택했을 때에 다항식 시간으로 답을 찾아내는 문제의 집합이 클래스 NP (class NP) 이다.

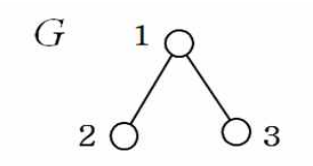
- 비결정론적 알고리즘
① 결정론적 알고리즘 : 모든 연산의 결과가 유일하게 정의된 것
② 비결정론적 알고리즘 : 알고리즘의 결과가 유일하지 않은 연산을 가질 수 있도록 즉 지정된 연산 결과의 집합 중 하나를 선택할 수 있도록 허용 한 것(상황에 따라 달라진다.)
- choice(X) : 집합 X의 원소 중 하나를 임의로 선택한다.
- failure : 알고리즘이 실패로 끝났음을 알린다.

결정론적 알고리즘에 의해 다항식 시간에 해결할 수 있는 문제는 비결정론적 알고리즘에 의해서도 다항식 시간에 해결할 수 있으므로 P ⊆ NP 가 성립한다. 그러나 NP ⊆ P 즉 NP 에 속하는 어떤 문제라도 다항식 시간에 해결할 수 있다면 P = NP 가 성립한다. 이것이 사실인지 아니면 P = NP 즉, 다항식시간에 해결할 수 있는 알고리즘이 존재하지 않는 문제가 NP 속에 있는지 하는 문제이다. 이 문제는 P = NP 문제라고 불리는 문제로서 컴퓨터 과학 분야에서는 최대의 과제 중 하나이다.

- NP-hard 문제
NP-hard 문제란 NP 에 속하는 어떤 문제보다도 어렵거나 같은 정도로 어려운 문제를 말한다. NP 완전 또는 NP-hard 문제에 대해서 현재 알려져 있는 알고리즘의 복잡도는 최악의 경우 입력 크기에 대해 지수 함수가 된다. 그래서 NP-hard 라는 것이 알려져 있는 최적화 문제에 대해서 구하는 답이 반드시 최적해가 아니라고 하더라도 최적 해에 가까운 것이면 된다.

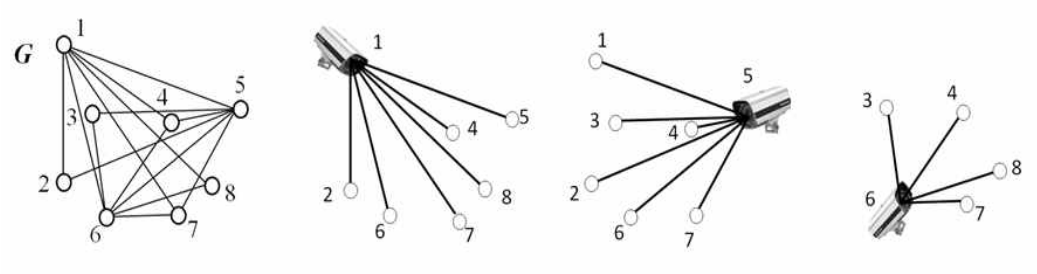
- 근사 알고리즘(approximation algorithm)
근사 알고리즘(approximation algorithm)은 어떤 최적화 문제에 대한 근사해를 구하는 알고리즘을 의미한다. 이 알고리즘은 가장 최적화되는 답을 구할 수는 없지만, 비교적 빠른 시간에 계산이 가능하며 어느 정도 보장된 근사해를 계산할 수 있다. 근사 알고리즘은 NP-완전 문제 등 현재 알려진 빠른 최적화 알고리즘이 없을 문제에 대해 주로 사용된다. 근사알고리즘은 근사해가 얼마나 최적해에 근사한지를 나타내는 근사비율(approximation ratio)을 알고리즘과 함께 제시해야한다. 근사비율은 근사행의 값과 최적해의 비율로 1에 가까울수록 정확도가 높은 알고리즘이라 할 수 있다. 근사 알고리즘은 정점 커버 (Vertex Cover) 문제, 통(궤) 채우기(Bin Packing) 문제, 여행자 문제, 0-1배낭문제, 작업 스케줄링 (Job Scheduling) 문제 등이 있다.

* 정점 커버 (Vertex Cover) 문제
- 정점 커버 (Vertex Cover) 문제는 주어진 그래프 G=(V,E)에서 각 선분의 양 끝점들 중에서 적어도 하나의 끝점을 포함하는 점들의 집합들 중에서 최소 크기의 집합을 찾는 문제이다.
- 정점 커버를 살펴보면, 그래프의 모든 선분이 정점 커버에 속한 점에 인접해있다.
- 즉, 정점 커버에 속한 점으로서 그래프의 모든 선분을 '커버'하는 것이다.



- 위의 그래프 G에서 {1, 2, 3}, {1, 2}, {1, 3}, {2, 3}, {1}이 각각 정점 커버이다.
- {2} 또는 {3}은 정점 커버가 아니다. 왜냐하면 {2}는 선분(1,3)을 커버하지 못하고, {3}은 선분 (1,2)를 커버하지 못한다.
- 따라서 위의 그래프에 대한 정점 커버 문제의 해는 {1}이다.

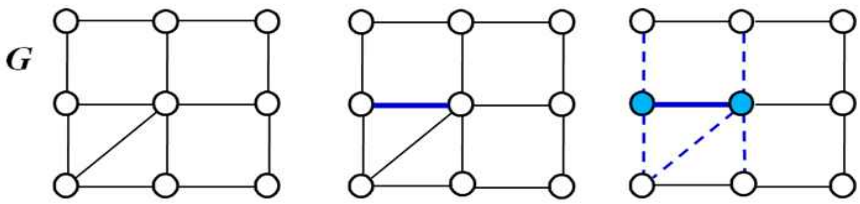
- 커버한다'는 용어의 의미를 다음의 예제를 통해서 이해하여 보자.
- 아래의 그래프 G는 어느 건물의 내부도면을 나타낸다.
- 건물의 모든 복도를 감시하기 위해 가장 적은 수의 CCTV 카메라를 설치하고자 한다.
- 이를 위해서 3대의 카메라를 각각 점 1, 5, 6에 설치하면 모든 복도 (선분)을 '커버'할 수 있다



- 따라서 그래프 G의 정점 커버를 찾는 것은 건물 내부 경비를 위한 최소의 카메라 수와 각 카메라의 위치를 구하는 것과 동일하다.
- 주어진 그래프의 모든 선분을 커버하려면 먼저 어떤 점을 선택해야 하는지 생각해 보아야 한다.
- 먼저 차수 (degree)가 가장 높은 점을 우선 선택하면 많은 수의 선분이 커버될 수 있다. 이 전략이 집합 커버 문제의 근사 알고리즘이다. 그러나 이때의 근사 비율은 logn이다.

- 또 다른 방법은 점을 선택하는 대신에 선분을 선택하는 것이다.
- 선분을 선택하면 선택된 선분의 양 끝점에 인접한 선분이 모두 커버된다.
- 따라서 정점 커버는 선택된 각 선분의 양 끝점들의 집합이다.
- 정점 커버를 만들어가는 과정에서, 새 선분은 자신의 양 끝점들이 이미 선택된 선분의 양 끝점들의 집합에 포함되지 않을 때에만 선택된다.

- 그림에서 1개의 선분이 임의로 선택되었을 때, 선택된 선분 주변의 5개의 선분 (점선으로 표기된 선분)은 정점 커버를 위해 선택되지 않는다.
- 그 이유는 선택된 선분의 양 끝점 (파란색 점)들이 점선으로 표시된 선분을 커버하기 때문이다



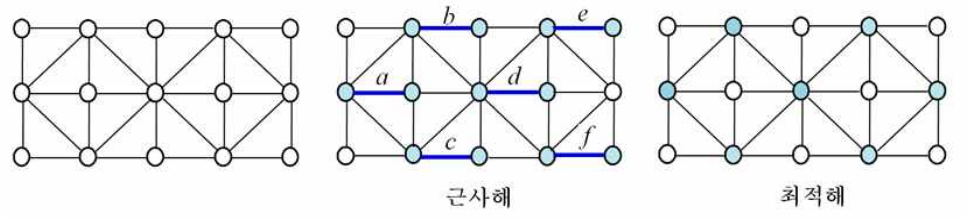
- 이러한 방식으로 선분을 선택하다가 더 이상 선분을 추가 수 없을 때 중단한다.
- 이렇게 선택된 선분의 집합을 극대 매칭(maximal matching)이라고 한다.
- 매칭(matching)이란 각 선분의 양쪽 끝점들이 중복되지 않는 선분의 집합이다.
- 극대 매칭은 이미 선택된 선분에 기반을 두고 새로운 선분을 추가하려 해도 더 이상 추가할 수 없는 매칭을 말한다.

* Approx_Matching_VC

입력: 그래프 G=(V,E)
출력: 정점 커버

1. 입력 그래프에서 극대 매칭 M을 찾는다.
2. return 매칭 M의 선분의 양 끝점들의 집합

- 아래의 그림에서 극대 매칭으로서 선분 a, b, c, d, e, f가 선택되었다. 따라서 근사해는 선분 a, b, c, d, e, f의 양 끝점들의 집합이다.
- 즉, 근사해는 총 12개의 점으로 구성된다. 반면에 오른쪽 그림은 입력 그래프의 최적해로서 7개의 점으로 구성되어 있다.

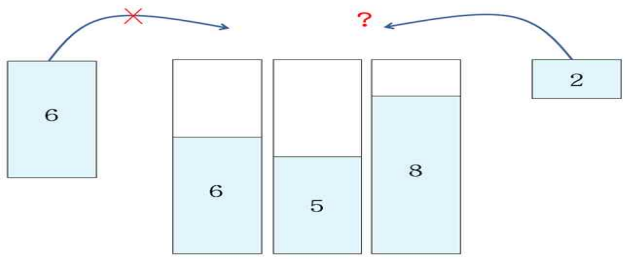


* 시간복잡도
- Approx_Matching_VC 알고리즘의 시간복잡도는 주어진 그래프에서 극대 매칭을 찾는 과정의 시간복잡도와 같다.
- 극대 매칭을 찾기 위해 하나의 선분을 선택할 때, 양 끝점들이 이미 선택된 선분의 양 끝점과 동일한지를 검사해야 하므로, 이는 O(n) 시간이 걸린다.
- 그런데 입력 그래프의 선분 수가 m이면, 각 선분에 대해서 O(n) 시간이 걸리므로, Approx_Matching_VC 알고리즘의 시간복잡도는 O(n)xm = O(nm)이다.

학습내용1 : 궤 채우기 알고리즘

1. 통 채우기(Bin Packing) 문제

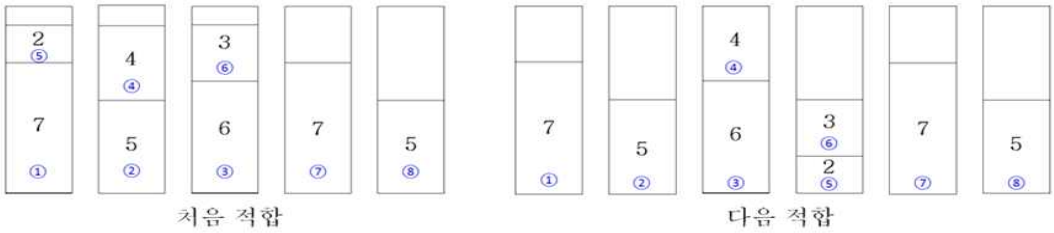
- 통 채우기(Bin Packing) 문제는 n개의 물건이 주어지고, 통 (bin)의 용량이 C일 때, 주어진 모든 물건을 가장 적은 수의 통에 채우는 문제이다. 단, 각 물건의 크기는 C보다 크지 않다.
- 물건을 통에 넣으려고 할 때에는 그 통에 물건이 들어갈 여유가 있어야만 한다.
- 예를 들어, 통의 크기가 10이고, 현재 3개의 통에 각각 6, 5, 8 만큼씩 차있다면, 크기가 6인 물건은 기존의 3개의 통에 넣을 수 없고 새 통에 넣어야 한다. 그런데 만일 새 물건의 크기가 2라면, 어느 통에 새 물건을 넣어야 할까?



- 질문에 대한 간단한 답은 그리디 방법으로 넣을 통을 정하는 것이다.
- 그리디 방법은 '무엇에 욕심을 낼 것인가'에 따라서 아래와 같이 4 종류로 분류할 수 있다.

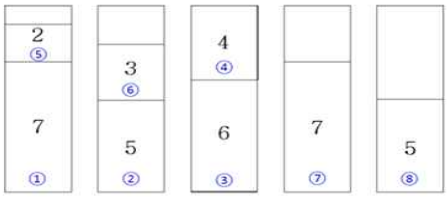
1) 최초 적합 (first fit): 첫 번째 통부터 차례로 살펴보며, 가장 먼저 여유가 있는 통에 새 물건을 넣는다.
2) 다음 적합 (next fit): 직전에 물건을 넣은 통에 여유가 있으면 새 물건을 넣는다.
3) 최선 적합 (best fit): 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 작은 통에 새 물건을 넣는다.
4) 최악 적합 (worst fit): 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 큰 통에 새 물건을 넣는다.

- 통의 용량 C=10이고, 물건의 크기가 각각 [7, 5, 6, 4, 2, 3, 7, 5]일 때, 최초 적합, 다음 적합, 최선 적합, 최악 적합을 각각 적용한 결과는 다음과 같다.

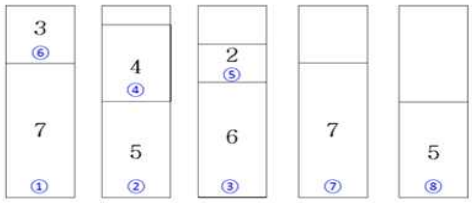


처음 적합

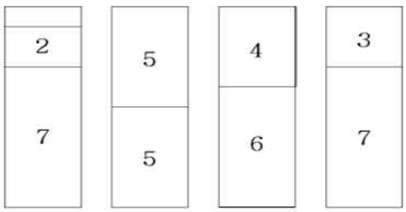
다음 적합



최선 적합



최악 적합



최적해

최후 적합 (Last Fit), 감소순 최초 적합 (First Fit Decrease), 감소순 최선 적합 (Best Fit Decrease) 등이 있다.

2. 알고리즘

입력: n개의 물건의 각각의 크기
출력: 모든 물건을 넣는데 사용된 통의 수

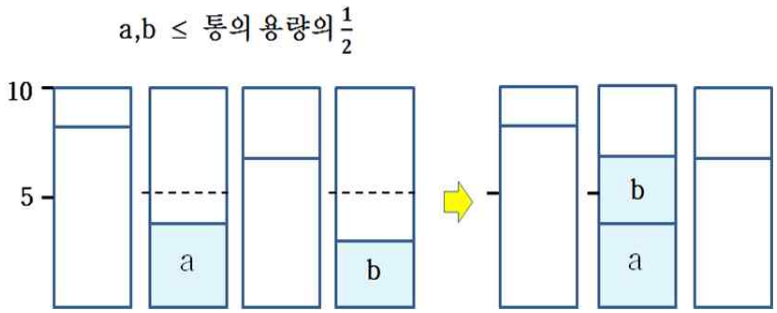
```
1. B = 0 // 사용된 통의 수
2. for i = 1 to n
3.         if (물건 i를 넣을 여유가 있는 기존의 통이 있으면)
4.                 그리디 방법에 따라 정해진 통에 물건 i를 넣는다.
5.         else
6.                 새 통에 물건 i를 넣는다.
7.                 B = B +1    // 통의 수를 1 증가시킨다.
8. return B
```

3. 시간복잡도

- 다음 적합을 제외한 다른 3가지 방법은 새 물건을 넣을 때마다 기존의 통을 살펴보아야 한다.
- 따라서 통의 수가 n을 넘지 않으므로 수행시간은 O(n2)이다.
- 다음 적합은 새 물건에 대해 직전에 사용된 통만을 살펴보면 되므로 수행시간은 O(n)이다.

4. 근사 비율

- 다음 적합을 제외한 3가지 방법에서 모든 물건을 넣는데 사용된 통의 수는 최적해에서 사용된 통의 수의 2배를 넘지 않는다.
- 이는 각 방법이 사용한 통을 살펴보면 2개 이상의 통이 1/2 이하로 차있을 수 없기 때문이다.
- 다음 그림에서와 같이 만일 2개의 통이 각각 1/2 이하로 차있다면, 각 방법은 새 통을 사용하지 않고, 이 2개의 통에 있는 물건을 1통으로 합친다.



- 최적해에서 사용된 통의 수를 OPT라고 하면, OPT ≥ (모든 물건의 크기의 합)/C 이다.
- 단, C는 통의 크기이다.

- 따라서 각 방법이 사용한 통의 수가 OPT'라면, OPT'의 통 중에서 1개의 통이 1/2 이하로 차있으므로, 각 방법이 (OPT'-1)개의 통에 각각 1/2 넘게 물건을 채울 때 그 물건의 크기의 합은 ((OPT'-1)xC/2)보다는 크다.
- 그러므로 다음과 같은 부등식이 성립한다.

(모든 물건의 크기의 합) > (OPT'-1) x C/2
⇒ (모든 물건의 크기의 합)/C > (OPT'-1)/2
⇒ OPT > (OPT'-1)/2, OPT≥(모든 물건의 크기의 합)/C이므로
⇒ 2OPT > OPT'-1
⇒ 2OPT + 1 > OPT'
⇒ 2OPT ≥ OPT'

- 따라서 3가지 방법의 근사 비율은 2이다.

문제)
궤의 용량이 모두 1이고 각 물체의 크기 xi가 다음과 같이 주어졌을 때, 최초법에 의해 물체를 궤에 집어넣을 때 필요한 궤의 최소 개수는?

(xi)=(0.4, 0.7, 0.2, 0.6, 0.5, 0.4)

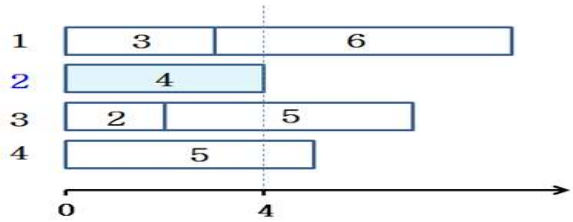
서로 다른 크기의 물체들과 동일한 크기의 궤들이 있을 때 최소 개수의 궤를 써서 모든 물체를 궤에 집어넣는 문제가 궤 채우기 문제이다. 이 문제는 NP-완전문제로서 결정론적 다항식 시간 알고리즘을 작성하는 일이 대단히 힘들기 때문에 실용적인 측면에서 근사 알고리즘을 이용하게 된다. 이 문제를 풀기 위한 방법은 최초법, 최선법, 감소순 최초법, 감소순 최선법이 있다. 주어진 문제에 대해 최초법은 (0.4,0.2,0,4)(0.7),(0.6),(0.5)의 4개의 궤가 필요하다.

학습내용2 : 작업 스케줄링 알고리즘

1. 작업 스케줄링 (Job Scheduling) 문제

- 작업 스케줄링 (Job Scheduling) 문제는 n개의 작업, 각 작업의 수행 시간 ti, i = 1, 2, 3, …, n, 그리고 m개의 동일한 기계가 주어질 때, 모든 작업이 가장 빨리 종료되도록 작업을 기계에 배정하는 문제이다.
- 단, 한 작업은 배정된 기계에서 연속적으로 수행되어야 한다.
- 또한 기계는 1번에 하나의 작업만을 수행한다.

- 작업을 어느 기계에 배정하여야 모든 작업이 가장 빨리 종료될까?
- 이에 대한 간단한 답은 그리디 방법으로 작업을 배정하는 것이다.
- 즉, 현재까지 배정된 작업에 대해서 가장 빨리 끝나는 기계에 새 작업을 배정하는 것이다.
- 아래 예제에서는 2 번째 기계가 가장 빨리 작업을 마치므로, 새 작업을 2 번째 기계에 배정한다.



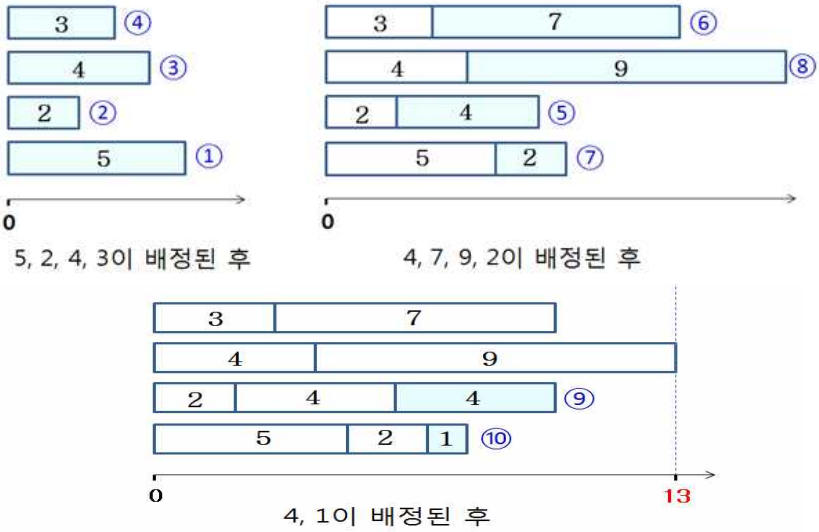
2. Approx_JobScheduling 알고리즘

입력: n개의 작업, 각 작업 수행 시간 ti, i = 1, 2, …, n, 기계 Mj, j = 1,2, …, m
출력: 모든 작업이 종료된 시간

```
1. for j = 1 to m
2. L[j] = 0 // L[j]=기계 Mj에 배정된 마지막 작업의 종료 시간
3. for i = 1 to n
4.       min = 1
5.       for j = 2 to m {     // 가장 일찍 끝나는 기계를 찾는다.
6.                 if (L[j] < L[min])
7.                           min = j  }
8.        작업 i를 기계 Mmin에 배정한다.
9.         L[min] = L[min] + ti
10. return 가장 늦은 작업 종료 시간
```

- Line 1~2: 각 기계에 배정된 마지막 작업의 종료 시간 L[j]를 0으로 초기화시킨다. 왜냐하면 초기엔 어떤 작업도 기계에 배정되지 않은 상태이기 때문이다. 단, 기계 번호는 1부터 시작된다.
- Line 3~9의 for-루프에서는 n개의 작업을 1개씩 가장 일찍 끝나는 기계에 배정한다.
- Line 4: 가장 일찍 끝나는 기계의 번호인 min을 1 (즉, 기계 M1)로 초기화시킨다.
- Line 5~7의 for-루프에서는 각 기계의 마지막 작업의 종료 시간을 검사하여, min을 찾는다.
- Line 8~9: 작업 i를 기계 Mmin에 배정하고, L[min]을 작업 i의 수행 시간을 더하여 갱신한다.
- Line 10: 배열 L에서 가장 큰 값을 찾아서 리턴한다

- 작업의 수행시간이 각각 5, 2, 4, 3, 4, 7, 9, 2, 4, 1이고, 4개의 기계가 있을 때, Approx_JobScheduling 알고리즘을 수행시킨 결과는 다음과 같다.



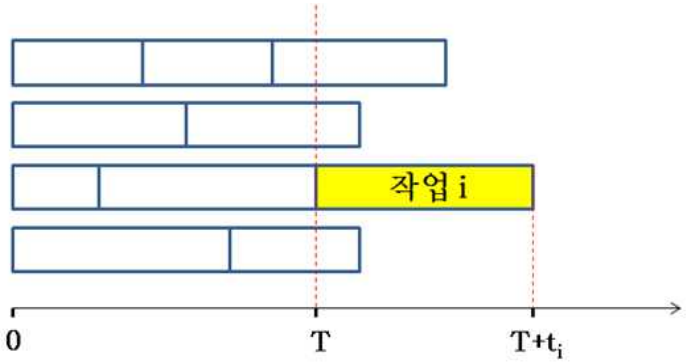
- Approx_JobScheduling 알고리즘은 가장 늦게 끝나는 작업의 종료 시간인 13을 리턴한다.

3. 시간복잡도

- Approx_JobScheduling 알고리즘은 n개의 작업을 하나씩 가장 빨리 끝나는 기계에 배정한다.
- 이러한 기계를 찾기 위해 알고리즘의 line 5~7의 for-루프가 (m-1)번 수행된다.
- 즉, 모든 기계의 마지막 작업 종료 시간인 L[j]를 살펴보아야 하므로 O(m) 시간이 걸린다.
- 따라서 시간복잡도는 n개의 작업을 배정해야하고, line 10에서 배열 L을 탐색해야 하므로 nxO(m) +O(m) = O(nm)이다.

4. 근사 비율

- Approx_JobScheduling 알고리즘의 근사 해를 OPT'라 하고, 최적 해를 OPT라고 할 때, OPT' ≤ 2xOPT이다.
- 즉, 근사해는 최적 해의 2배를 넘지 않는다. 이를 아래의 그림을 통해서 이해해보자. 단, ti는 작업 i의 수행 시간이다.



- 앞 그림에서 T'는 작업 i를 제외한 모든 작업의 수행 시간의 합을 기계의 수 m으로 나눈 값이다.
- 즉, T'는 작업 i를 제외한 평균 종료 시간이다.
- 그러면 T ≤ T'이 된다.
- 왜냐하면 작업 i가 배정된 (가장 늦게 끝나는) 기계를 제외한 모든 기계에 배정된 작업은 적어도 T 이후에 종료되기 때문이다.

【학습정리】

1. 정점 커버 (Vertex Cover) 문제
- 정점 커버 (Vertex Cover) 문제는 주어진 그래프 G=(V,E)에서 각 선분의 양 끝점들 중에서 적어도 하나의 끝점을 포함하는 점들의 집합들 중에서 최소 크기의 집합을 찾는 문제이다.
- 정점 커버를 살펴보면, 그래프의 모든 선분이 정점 커버에 속한 점에 인접해있다.
- 즉, 정점 커버에 속한 점으로서 그래프의 모든 선분을 '커버'하는 것이다.

2. 궤(통) 채우기 알고리즘
- 통 채우기(Bin Packing) 문제는 n개의 물건이 주어지고, 통 (bin)의 용량이 C일 때, 주어진 모든 물건을 가장 적은 수의 통에 채우는 문제이다. 단, 각 물건의 크기는 C보다 크지 않다.
- 물건을 통에 넣으려고 할 때에는 그 통에 물건이 들어갈 여유가 있어야만 한다.

- 그리디 방법은 '무엇에 욕심을 낼 것인가'에 따라서 아래와 같이 4 종류로 분류할 수 있다.
1) 최초 적합 (first fit): 첫 번째 통부터 차례로 살펴보며, 가장 먼저 여유가 있는 통에 새 물건을 넣는다.
2) 다음 적합 (next fit): 직전에 물건을 넣은 통에 여유가 있으면 새 물건을 넣는다.
3) 최선 적합 (best fit): 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 작은 통에 새 물건을 넣는다.
4) 최악 적합 (worst fit): 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 큰 통에 새 물건을 넣는다.

3. 작업 스케줄링 알고리즘
- 작업 스케줄링 (Job Scheduling) 문제는 n개의 작업, 각 작업의 수행 시간 ti, i = 1, 2, 3, …, n, 그리고 m개의 동일한 기계가 주어질 때, 모든 작업이 가장 빨리 종료되도록 작업을 기계에 배정하는 문제이다.
- 단, 한 작업은 배정된 기계에서 연속적으로 수행되어야 한다.
- 또한 기계는 1번에 하나의 작업만을 수행한다.