근사 알고리즘
(Approximation Algorithms)

NP-완전 문제의 해결

- NP-완전 문제들은 실 세계의 광범위한 영역에 활용되지만, 이 문제들을 다항식 시간에 해결할 수 있는 알고리즘이 아직 발견되지 않았다.

- 또한 아직까지 그 누구도 이 문제들을 다항식 시간에 해결할 수 없다고 증명하지도 못했다.

- 대부분의 학자들은 이 문제들을 해결할 다항식 시간 알고리즘이 존재하지 않을 것이라고 추측하고 있다.

NP-완전 문제의 해결

- 이러한 NP-완전 문제들을 어떤 방식으로든지 해결하려면 다음의 3가지 중에서 1가지는 포기해야 한다.
  - 다항식 시간에 해를 찾는 것
  - 모든 입력에 대해 해를 찾는 것
  - 최적해를 찾는 것

- 근사 알고리즘은 NP-완전 문제를 해결하기 위해 3 번째 것을 포기한다.
  - 즉, 최적해에 아주 근사한 (가까운) 해를 찾아주는 것이 근사 알고리즘 (Approximation algorithm)이다.

근사 알고리즘

- 근사 알고리즘은 근사해를 찾는 대신에 다항식 시간의 복잡도를 가진다.
- 근사 알고리즘은 근사해가 얼마나 최적해에 근사한지 (즉, 최적해에 얼마나 가까운지)를 나타내는 근사 비율 (Approximation Ratio)을 알고리즘과 함께 제시하여야 한다.
- 근사 비율은 근사해의 값과 최적해의 값의 비율로서, 1.0에 가까울수록 정확도가 높은 알고리즘이다.
- 그런데 근사 비율을 계산하려면 최적해를 알아야 하는 모순이 생긴다.
- 따라서 최적해를 대신할 수 있는 ‘간접적인’ 최적해를 찾고, 이를 최적해로 삼아서 근사 비율을 계산한다.

1. 여행자 문제

- 여행자 문제 (Traveling Salesman Problem, TSP)
  - 여행자가 임의의 한 도시에서 출발하여 다른 모든 도시를 1번씩만 방문하고 다시 출발했던 도시로 돌아오는 여행 경로의 길이를 최소화하는 문제

- 여행자 문제의 조건
  - 도시 A에서 도시 B로 가는 거리는 도시 B에서 도시 A로 가는 거리와 같다. (대칭성)
  - 도시 A에서 도시 B로 가는 거리는 도시 A에서 다른 도시 C를 경유하여 도시 B로 가는 거리보다 짧다. (삼각 부등식 특성)

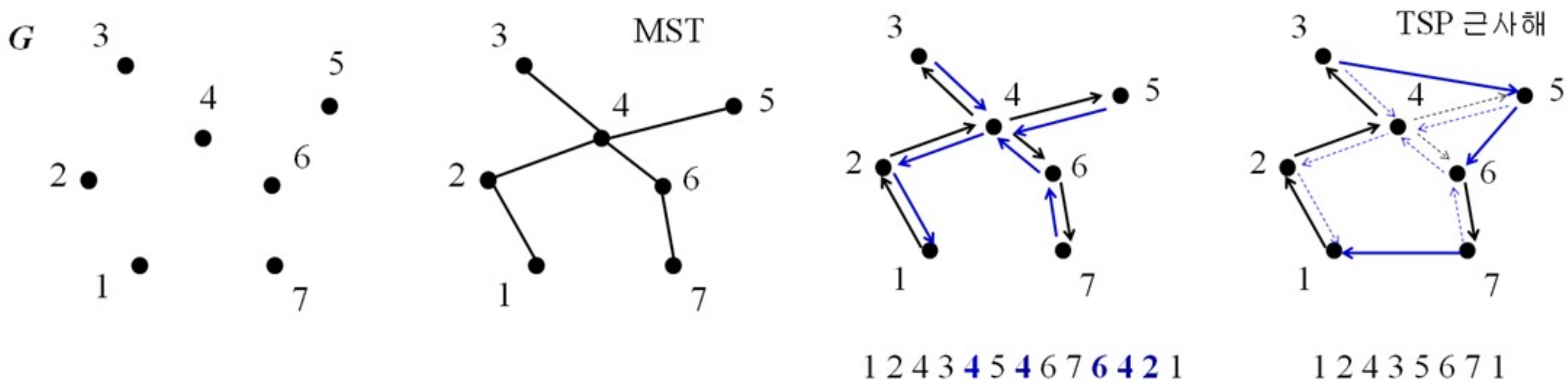
여행자 문제

- TSP를 위한 근사 알고리즘을 고안하려면, 먼저 다항식 시간 알고리즘을 가지면서 유사한 특성을 가진 문제를 찾아서 활용한다.
  - TSP와 비슷한 특성을 가진 문제는 최소 신장 트리 (Minimum Spanning Tree, MST) 문제이다.
  - MST 는 모든 점을 사이클 없이 연결하는 트리 중에서 트리 선분의 가중치 합이 최소인 트리이다.
  - MST 의 모든 점을 연결하는 특성과 최소 가중치의 특성을 TSP에 응용하여, 시작 도시를 제외한 다른 모든 도시를 트리 선분을 따라 1번씩 방문하도록 경로를 찾는 것이다.

여행자 문제

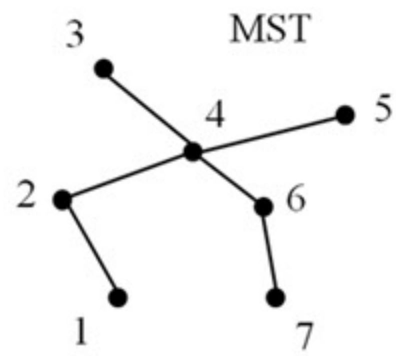
❑ MST를 활용한 근사해 찾는 과정

– MST를 활용하여 여행자 문제의 근사해를 찾는 데에는 삼각 부등식 원리를 적용한다.

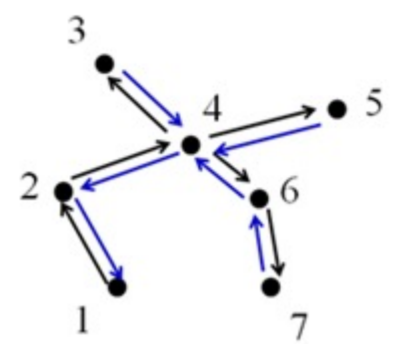


여행자 문제

- 먼저 그래프 G에서 크러스컬 또는 프림 알고리즘을 이용하여 최소 신장 트리 MST를 찾는다.

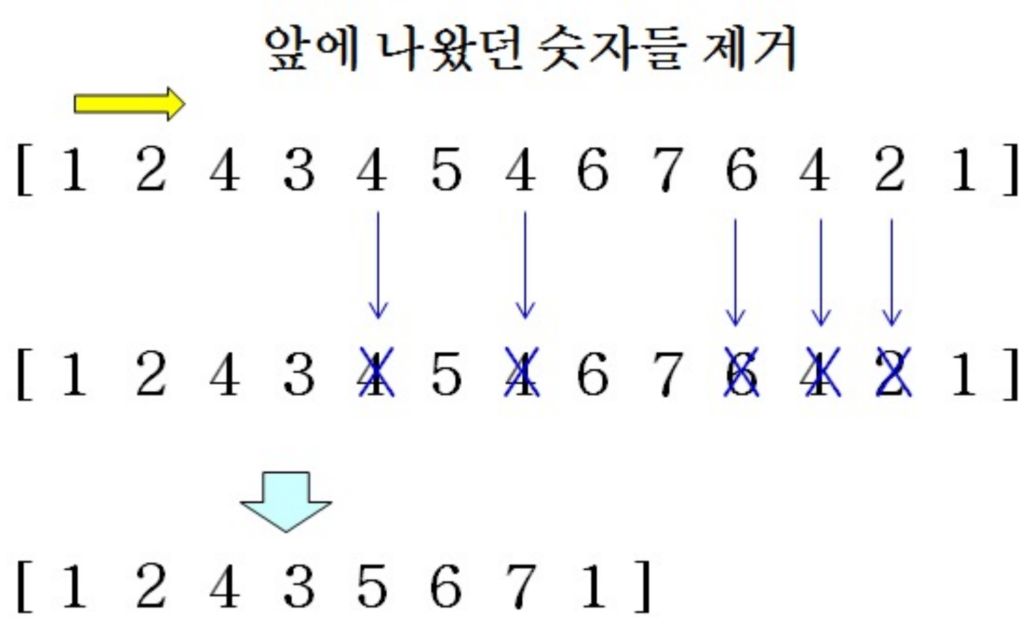


- 다음으로 임의의 도시 (그림에서는 도시 1)에서 출발하여 트리의 선분을 따라서 모든 도시를 방문하고 돌아오는 도시의 방문 순서를 구한다.
  – [1 2 4 3 4 5 4 6 7 6 4 2 1].



여행자 문제

- 마지막으로 이 순서를 따라서 도시를 방문하되 중복 방문하는 도시를 순서에서 제거한다.
  - 단, 도시 순서의 가장 마지막에 있는 출발 도시 1은 중복되어 나타나지만 제거하지 않는다.



  - 중복하여 방문하는 도시를 제거하는 과정에 삼각형 부등식 원리가 사용된 것이다.

Approx_MST_TSP

입력: n개의 도시, 각 도시간의 거리

출력: 출발 도시에서 각 도시를 1번씩만 방문하고 출발 도시로 돌아오는 도시 순서

1. 입력에 대하여 MST를 찾는다.
2. MST에서 임의의 도시로부터 출발하여 트리의 선분을 따라서 모든 도시를 방문하고 다시 출발했던 도시로 돌아오는 도시 방문 순서를 찾는다.
3. return 이전 단계에서 찾은 도시 순서에서 중복되어 나타나는 도시를 제거한 도시 순서 (단, 도시 순서의 가장 마지막의 출발 도시는 제거하지 않는다.)

Approx_MST_TSP

- Line 1
  - 크러스컬 (Kruskal)이나 프림 (Prim) 알고리즘을 사용하여 MST를 찾는다.
- Line 2
  - 하나의 도시에서 어떤 트리 선분을 선택하여 다른 도시를 방문할 때 지켜야 할 순서가 정해져 있지 않으므로, 임의의 순서로 방문해도 괜찮다.
- Line 3
  - 삼각 부등식의 원리를 적용함과 동시에 각 도시를 1번씩만 방문하기 위해서 중복 방문된 도시를 제거한다.
  - 단, 가장 마지막 도시인 출발 도시는 중복되지만 TSP 문제의 출발 도시로 돌아와야 한다는 조건에 따라 제거하지 않는다.

시간복잡도

- Line 1
  - MST를 찾는 데에는 크러스컬이나 프림 알고리즘의 시간복잡도만큼 시간이 걸린다.
  - 크러스컬 알고리즘: O(mlogm), m은 선분의 수
  - 프림 알고리즘: $O(n^2)$, n은 점의 수
- Line 2
  - 트리 선분을 따라서 도시 방문 순서를 찾는 데는 O(n) 시간이 걸린다.
  - 왜냐하면 트리의 선분 수가 (n-1)이기 때문이다.
- Line 3
  - line 2에서 찾은 도시 방문 순서를 따라가며, 단순히 중복된 도시를 제거하므로 O(n) 시간이 걸린다.

시간복잡도

- 시간복잡도
  - MST를 찾는 시간복잡도 + O(n) + O(n) 이므로 크러스컬이나 프림 알고리즘의 시간복잡도와 같다.

근사 비율

- 여행자 문제의 최적해를 실질적으로 알 수 없으므로, '간접적인' 최적해인 MST 선분의 가중치의 합(M)을 최적해의 값으로 활용한다.
  - 왜냐하면 실제의 최적해의 값이 M보다 항상 크기 때문이다.

- 그런데 Approx_MST_TSP 알고리즘이 계산한 근사해의 값은 2M보다는 크지 않다.
  - 왜냐하면 Line 2에서 MST의 선분을 따라서 도시 방문 순서를 찾을 때 사용된 트리 선분을 살펴보면, 각 선분이 2번 사용되었다. 따라서 이 도시 방문 순서에 따른 경로의 총 길이는 2M이다.

근사 비율

- Line 3에서는 삼각 부등식의 원리를 이용하여 새로운 도시 방문 순서를 만들기 때문에, 이전 도시 방문 순서에 따른 경로의 길이보다 새로운 도시 방문 순서에 따른 경로의 길이가 더 짧다.

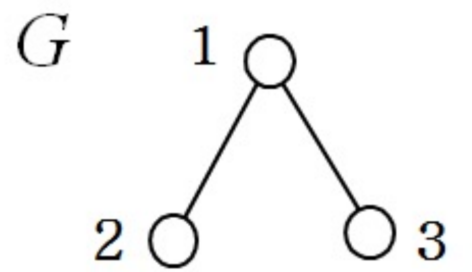
- 따라서 이 알고리즘의 근사비율은 2M/M=2 보다 크지 않다.
  - 즉, 근사해의 값이 최적해의 값의 2배를 넘지 않는다.

2. 정점 커버 문제

- 정점 커버 (Vertex Cover) 문제
  - 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.

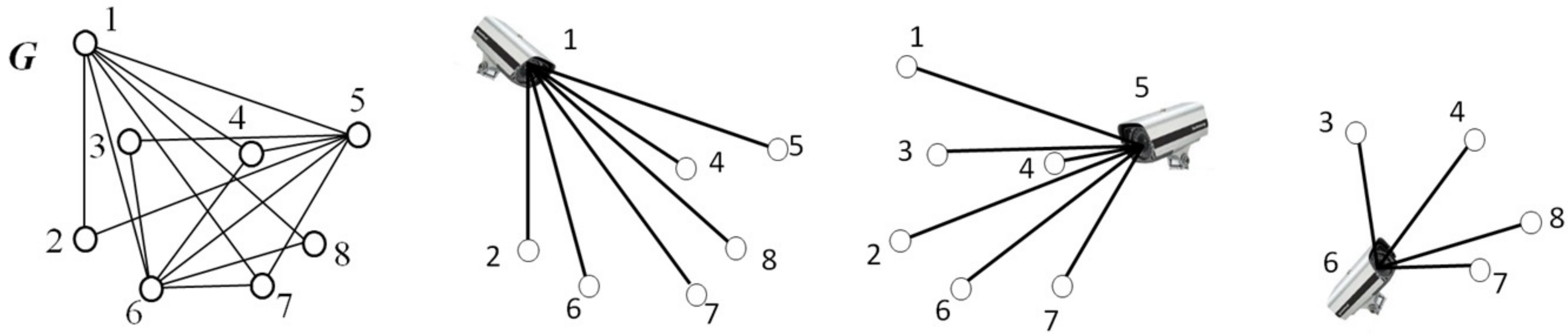
정점 커버 문제



- 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.

정점 커버 문제

- ‘커버한다’는 용어의 의미를 이해하기 위한 예제
  - 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.



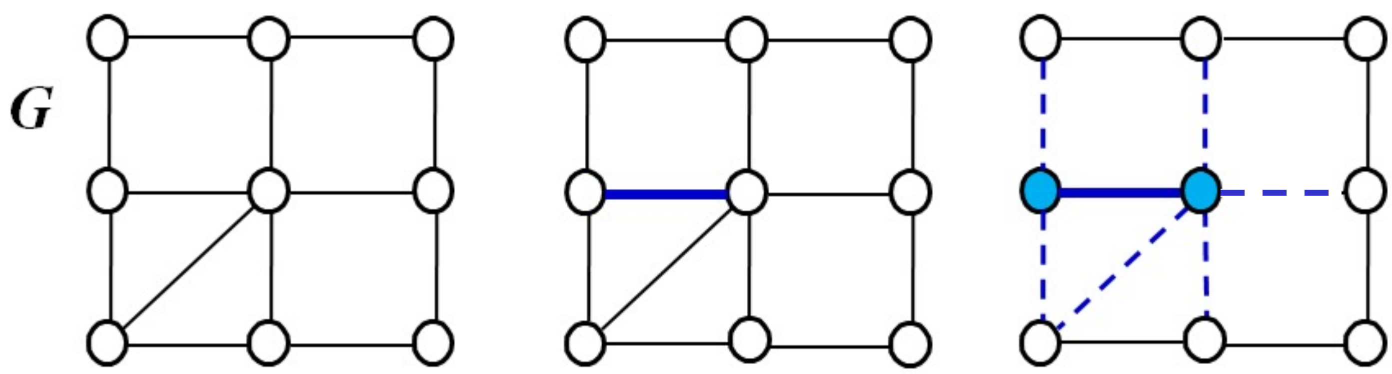
정점 커버 문제

- 주어진 그래프의 모든 선분을 커버하려면 먼저 어떤 점을 선택해야 하는지 생각해 보아야 한다.

- 선분 선택 방법
  - 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.

정점 커버 문제

- 그림에서 1개의 선분이 임의로 선택된 경우
  - 선택된 선분 주변의 6개의 선분 (점선으로 표기된 선분)은 정점 커버를 위해 선택되지 않는다.
  - 그 이유는 선택된 선분의 양 끝점 (파란색 점)들이 점선으로 표시된 선분을 모두 커버하기 때문이다.



  - 이러한 방식으로 선분을 선택하다가 더 이상 선분을 추가 수 없을 때 중단한다.

정점 커버 문제

- 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.

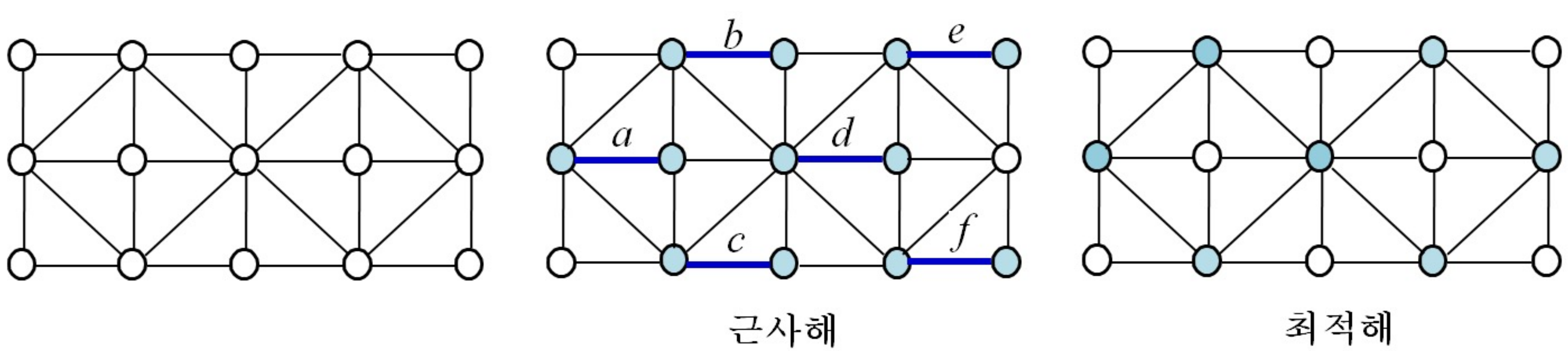
Approx_Matching_VC

입력: 그래프 G=(V,E)
출력: 정점 커버

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

Approx_Matching_VC

- 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 시간복잡도
  - 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 걸린다.
  - 따라서 Approx_Matching_VC 알고리즘의 시간복잡도는 O(n) x m = O(nm)이다.

근사 비율

- 근사 비율을 계산하기 위해서 극대 매칭을 '간접적인' 최적해로 사용한다.
  - 즉, 매칭에 있는 선분의 수를 최적해의 값으로 사용한다.
  - 그 이유는 어떠한 정점 커버라도 극대 매칭에 있는 선분을 커버해야 하기 때문이다.

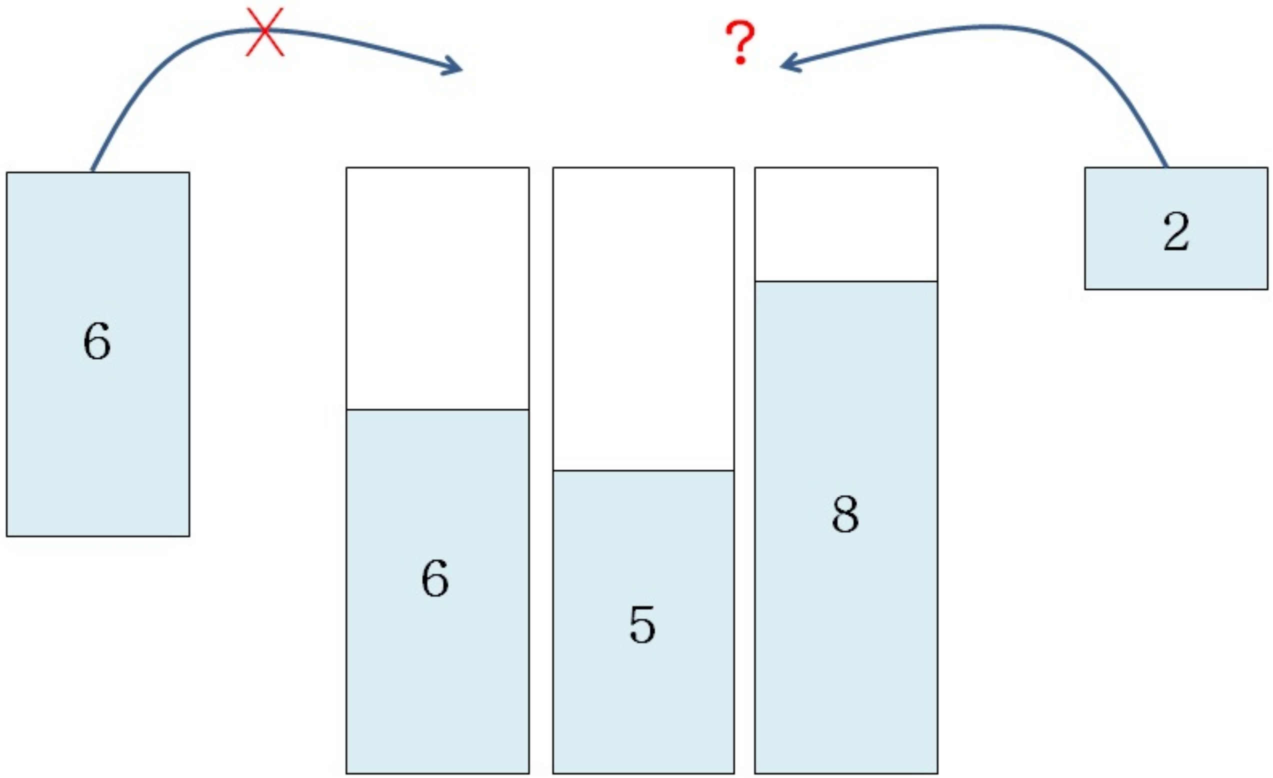
- Approx_Matching_VC 알고리즘
  - 극대 매칭의 각 선분의 양쪽 끝점들의 집합을 정점 커버의 근사해로서 리턴하므로, 근사해의 값은 극대 매칭의 선분 수의 2배이다.
  - 따라서 근사 비율은 (극대 매칭의 각 선분의 양 끝점들의 수)/(극대 매칭의 선분 수) = 2이다.

3. 통 채우기 문제

- 통 채우기(Bin Packing) 문제
  - n개의 물건이 주어지고, 통 (bin)의 용량이 C일 때, 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 차 있다면, 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?

통 채우기 문제



- 질문에 대한 간단한 답은 그리디 방법으로 넣을 통을 정하는 것이다.

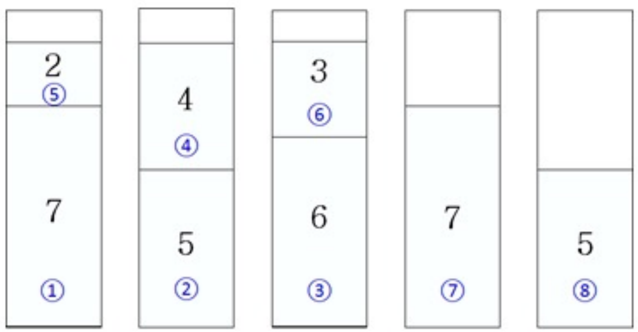
통 채우기 문제

- 그리디 방법은 '무엇에 욕심을 낼 것인가'에 따라서 아래와 같이 4 종류로 분류할 수 있다.
  - 최초 적합 (First Fit)
    - 첫 번째 통부터 차례로 살펴보며, 가장 먼저 여유가 있는 통에 새 물건을 넣는다.
  - 다음 적합 (Next Fit)
    - 직전에 물건을 넣은 통에 여유가 있으면 새 물건을 넣는다.
  - 최선 적합 (Best Fit)
    - 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 작은 통에 새 물건을 넣는다.
  - 최악 적합 (Worst Fit)
    - 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 큰 통에 새 물건을 넣는다.

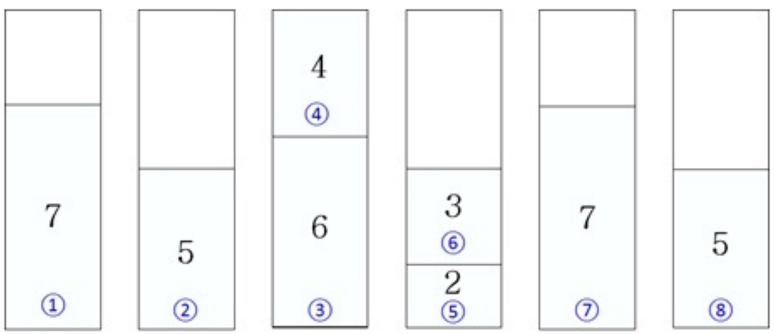
  - 각 방법으로 새 물건을 기존의 통에 넣을 수 없으면, 새로운 통에 새 물건을 넣는다.

통 채우기 문제

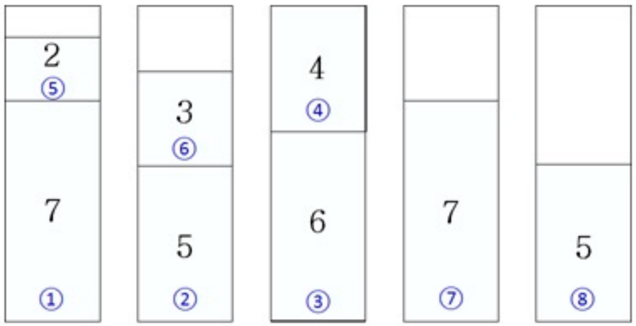
- 통의 용량 C=10이고, 물건의 크기가 각각 [7, 5, 6, 4, 2, 3, 7, 5]일 때, 최초 적합, 다음 적합, 최선 적합, 최악 적합을 각각 적용한 결과는 다음과 같다.
  - 그림에서 원 숫자는 물건이 채워지는 순서를 나타낸다.



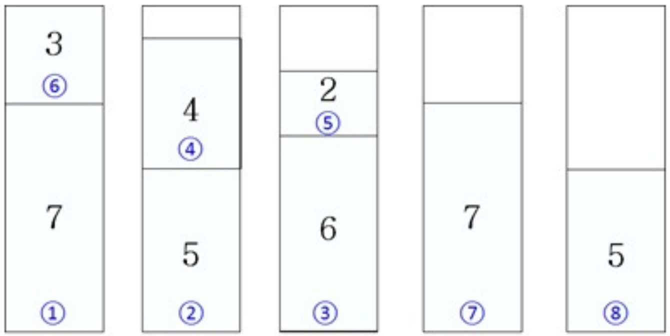
최초 적합



다음 적합

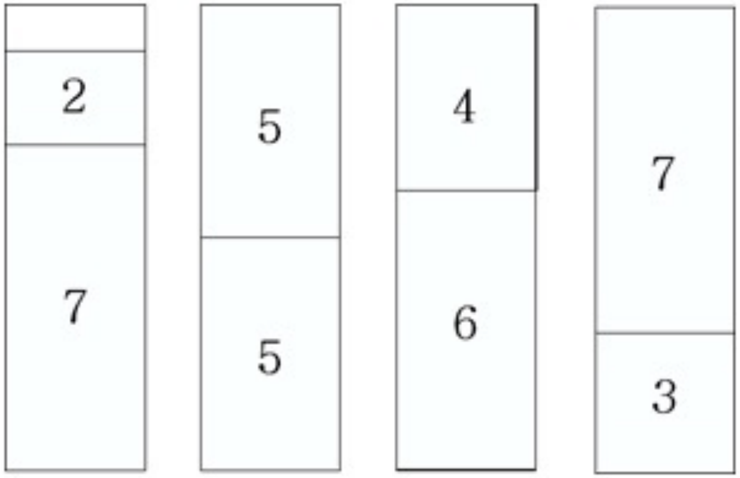


최선 적합



최악 적합

통 채우기 문제



최적해

Approx_BinPacking

입력: n개 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
```

시간복잡도

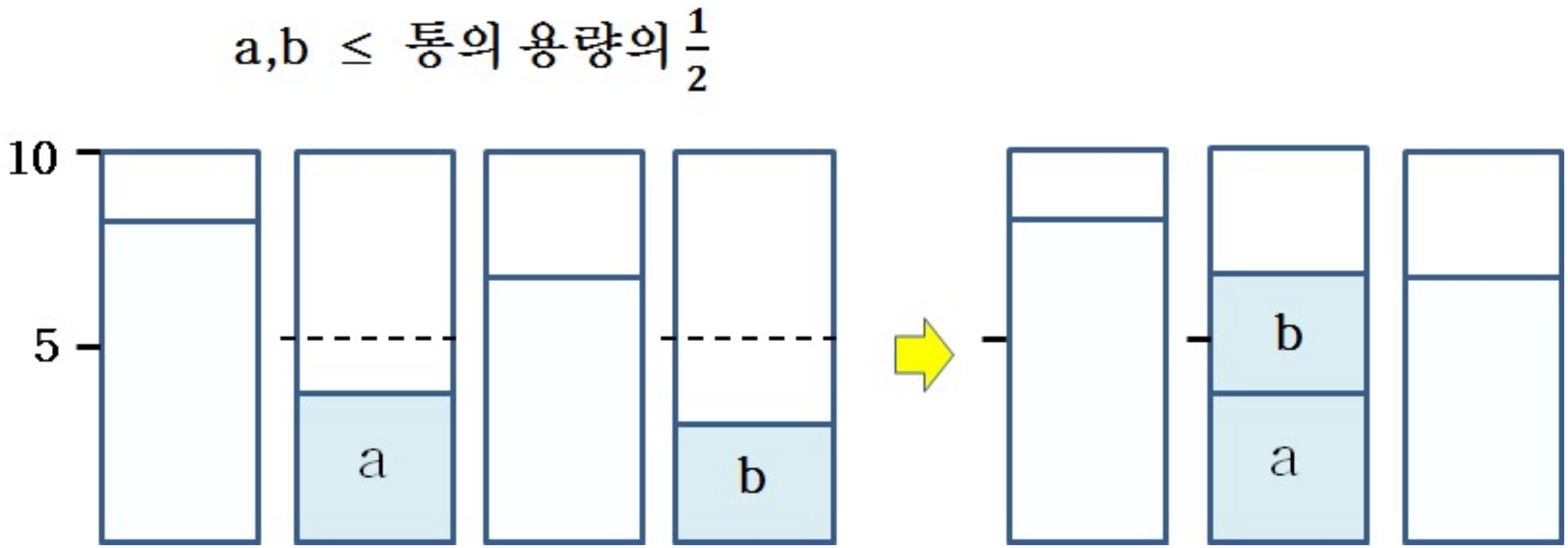
- 다음 적합을 제외한 다른 3가지 방법
  - 새 물건을 넣을 때마다 기존의 통을 살펴보아야 한다.
  - 따라서 통의 수가 n을 넘지 않으므로 수행시간은 $O(n^2)$이다.

- 다음 적합
  - 새 물건에 대해 직전에 사용된 통만을 살펴보면 되므로 수행시간은 $O(n)$이다.

근사 비율

- 다음 적합을 제외한 3가지 방법
  - 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 차 있을 수 없기 때문이다.
  - 만일 2개의 통이 각각 1/2 이하로 차 있다면, 각 방법은 새 통을 사용하지 않고, 이 2개의 통에 있는 물건을 1통으로 합친다.

근사 비율



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

근사 비율

- 따라서 각 방법이 사용한 통의 수가 OPT′ 라면,
  - OPT′의 통 중에서 기껏해야 1개의 통이 ½ 이하로 차 있으므로,
  - 각 방법이 (OPT′-1)개의 통에 각각 ½ 넘게 물건을 채울 때 그 물건의 크기의 합은 ((OPT′-1) x C/2) 보다는 크다.

  - 그러므로 다음과 같은 부등식이 성립한다.

    (모든 물건의 크기의 합) > (OPT′-1) x C/2

근사 비율

- (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)/C 이므로

  ⇒ 2OPT > OPT′-1

  ⇒ 2OPT + 1 > OPT′

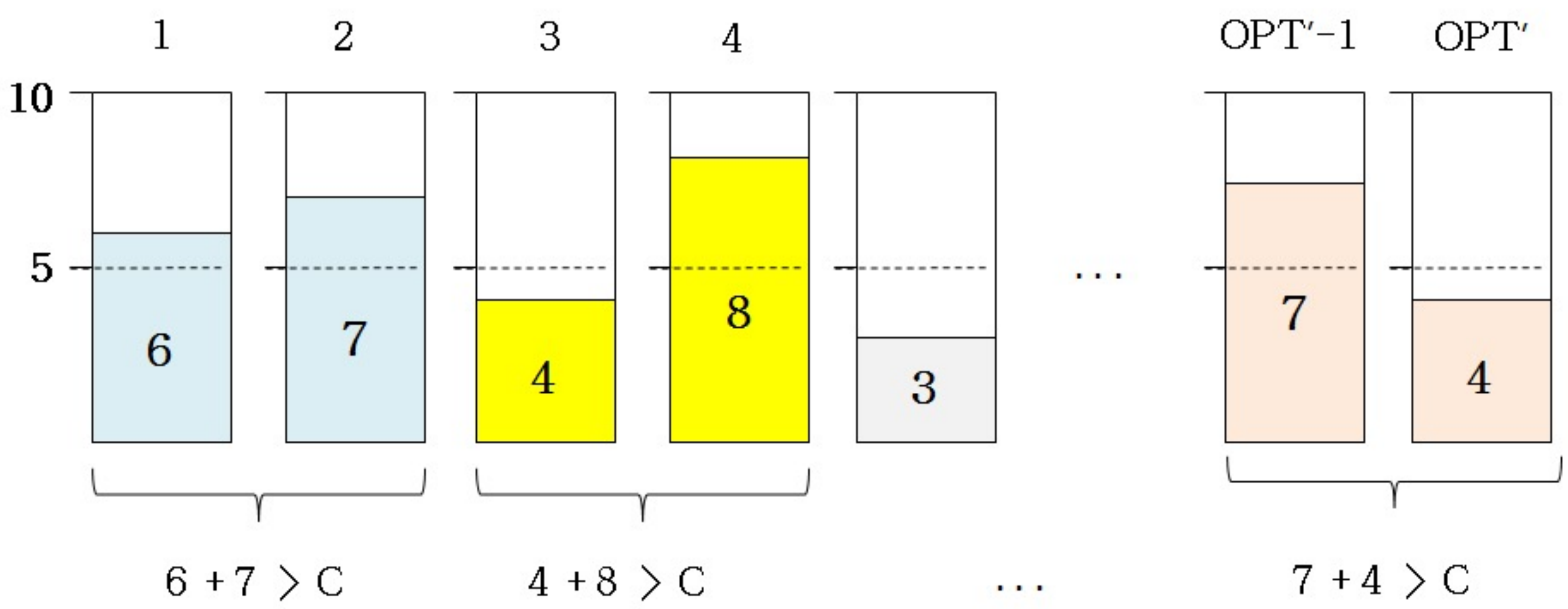
  ⇒ 2OPT ≥ OPT′

- 따라서 3가지 방법의 근사 비율은 2 이다.

근사 비율

❑ 다음 적합

- 직전에 사용된 통에 들어있는 물건의 크기의 합과 새 물건의 크기의 합이 통의 용량보다 클 때에만, 새 통에 새 물건을 넣는다.
- 다음 적합이 사용한 통의 수가 OPT′라면, 이웃한 2개의 통을 다음 그림과 같이 나타낼 수 있다.



근사 비율

(모든 물건의 크기의 합) > OPT'/2 x C

⇒ (모든 물건의 크기의 합)/C > OPT'/2

⇒ OPT > OPT'/2,

OPT≥(모든 물건의 크기의 합)/C 이므로

⇒ 2OPT > OPT'

- 따라서 다음 적합의 근사 비율은 2 이다.

4. 작업 스케줄링 문제

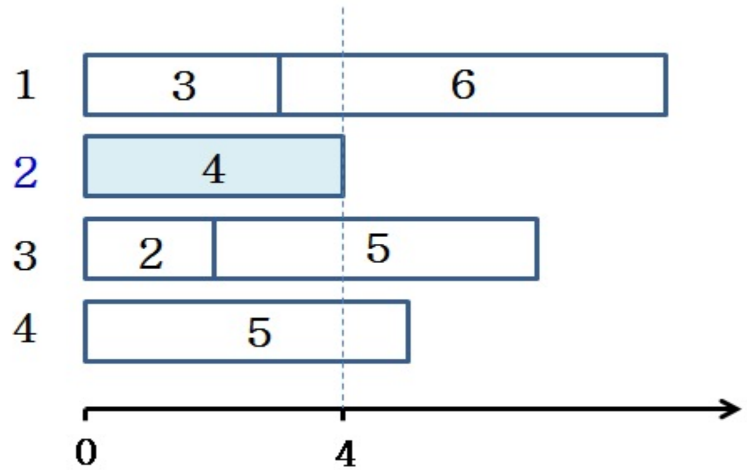
작업 스케줄링 (Job Scheduling) 문제

- n개의 작업,
- 각 작업의 수행 시간 t_i, $i = 1, 2, 3, \cdots, n$,
- 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.

작업 스케줄링 문제

- 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.



Approx_JobScheduling

입력: n개의 작업, 각 작업 수행 시간 t_i, $i = 1, 2, \cdots, n$,

기계 M_j, $j = 1,2, \cdots, m$

출력: 모든 작업이 종료된 시간

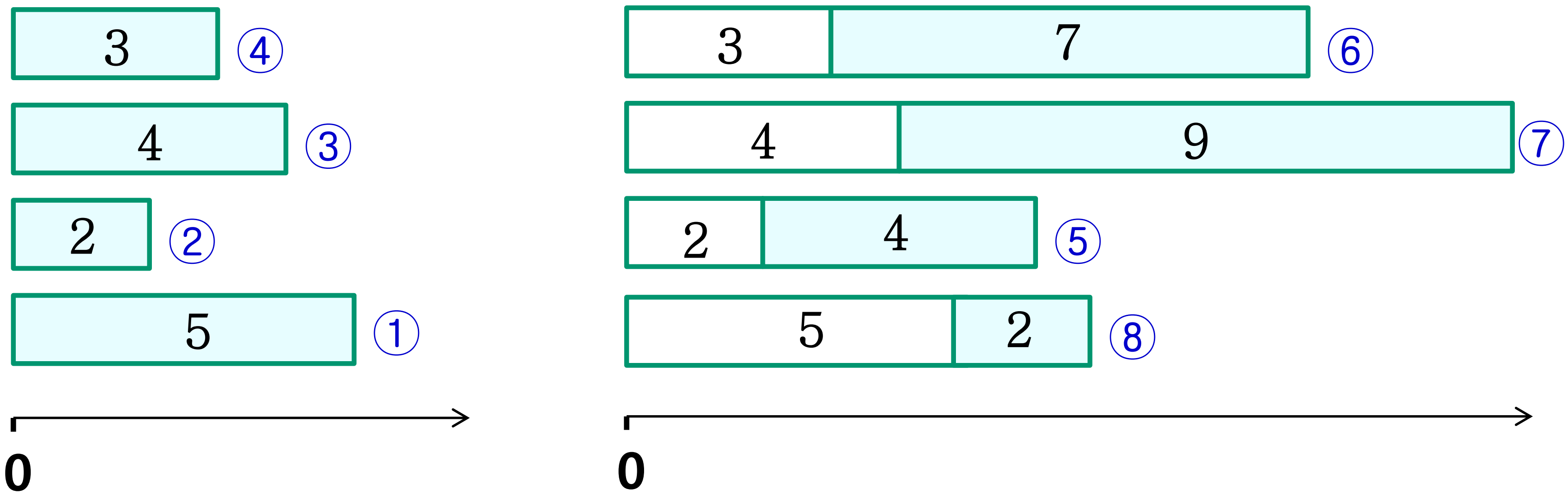
```
for j = 1 to m
    L[j] = 0            // L[j]=기계 M_j에 배정된 마지막 작업의 종료 시간
for i = 1 to n {
    min = 1
    for j = 2 to m {    // 가장 일찍 끝나는 기계를 찾는다.
        if (L[j] < L[min])
            min = j
    }
    작업 i를 기계 M_min에 배정한다.
    L[min] = L[min] + t_i
}
return 가장 늦은 작업 종료 시간
```

Approx_JobScheduling

- 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 3~9
  - for-루프에서는 n개의 작업을 1개씩 가장 일찍 끝나는 기계에 배정한다.
- Line 4
  - 가장 일찍 끝나는 기계의 번호인 min을 1 (즉, 기계 M_1)로 초기화시킨다.
- Line 5~7
  - for-루프에서는 각 기계의 마지막 작업의 종료 시간을 검사하여, min을 찾는다.
- Line 8~9
  - 작업 i를 기계 M_{min}에 배정하고, L[min]을 작업 i의 수행 시간을 더하여 갱신한다.
- Line 10
  - 배열 L에서 가장 큰 값을 찾아서 리턴한다.

Approx_JobScheduling

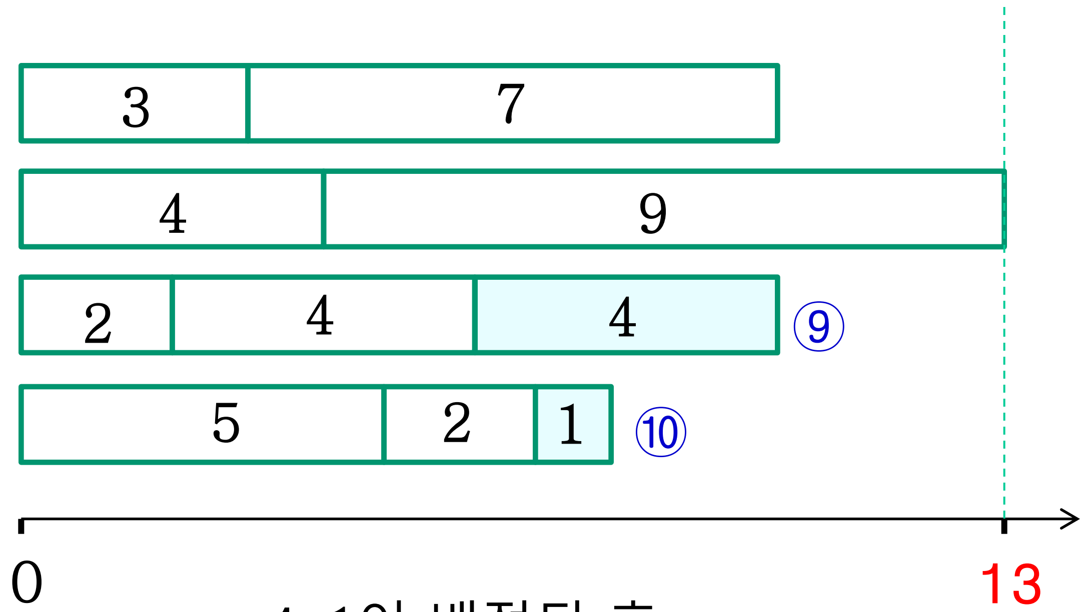
- Approx_JobScheduling 알고리즘 수행 결과
  - 작업의 수행시간이 각각 5, 2, 4, 3, 4, 7, 9, 2, 4, 1 이고
  - 4개의 기계가 있을 경우



5, 2, 4, 3이 배정된 후

4, 7, 9, 2이 배정된 후

Approx_JobScheduling



4, 1이 배정된 후

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

시간복잡도

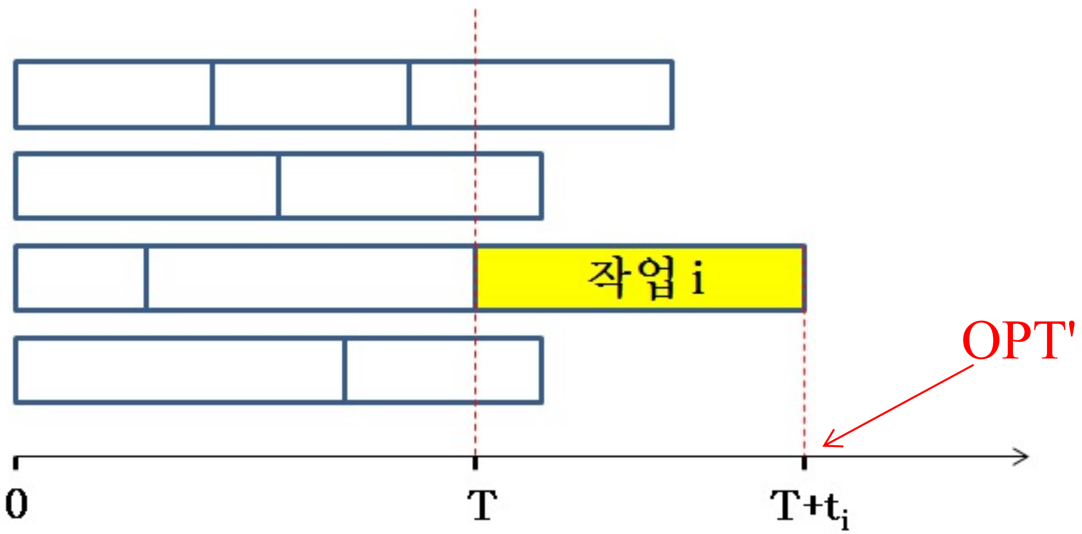
- Approx_JobScheduling 알고리즘
  - n개의 작업을 하나씩 가장 빨리 끝나는 기계에 배정한다.
  - 이러한 기계를 찾기 위해 알고리즘의 line 5~7의 for-루프가 (m-1)번 수행된다.
  - 즉, 모든 기계의 마지막 작업 종료 시간인 L[j]를 살펴보아야 하므로 O(m) 시간이 걸린다.

- 시간복잡도
  - n개의 작업을 배정해야하고,
  - line 10에서 배열 L을 탐색해야하므로
  - n x O(m) + O(m) = O(nm) 이다.

근사 비율

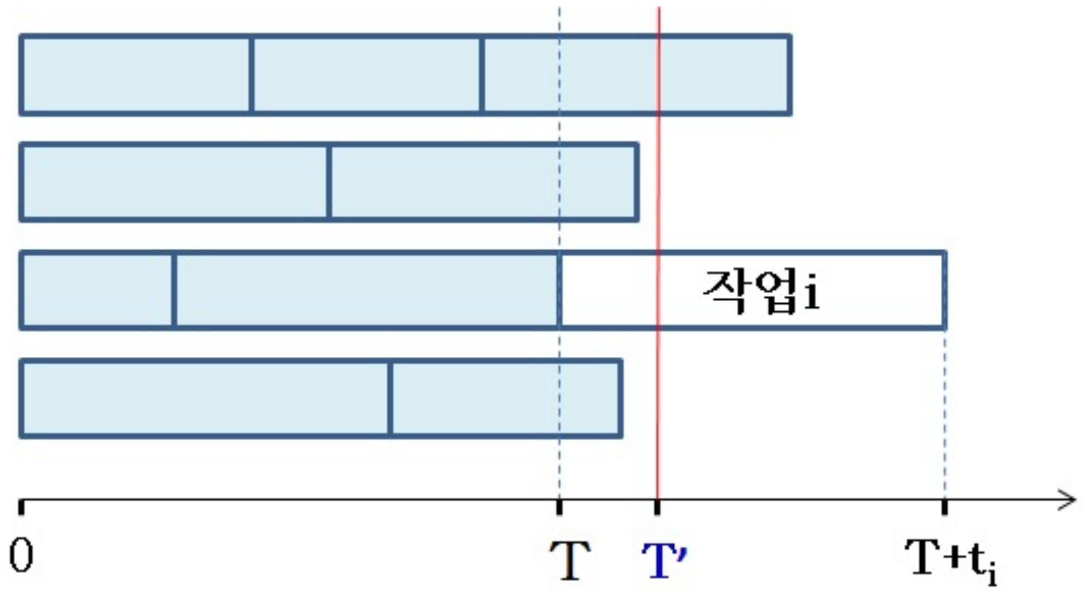
- Approx_JobScheduling 알고리즘의 근사해를 OPT'라 하고, 최적해를 OPT라고 할 때, OPT' ≤ 2xOPT 이다.
  - 즉, 근사해는 최적해의 2배를 넘지 않는다.
  - 이를 다음 그림을 통해서 이해해보자. 단, t_i는 작업 i의 수행 시간이다.

근사 비율



- 위의 그림은 Approx_JobScheduling 알고리즘으로 작업을 배정하였고,
- 가장 마지막으로 배정된 작업 i가 T부터 수행되며,
- 모든 작업이 $T+t_i$에 종료된 것을 보이고 있다.
- 그러므로 OPT' = $T+t_i$이다.

근사 비율



- 위 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.

근사 비율

❑ T와 T'의 관계인, T ≤ T'를 이용한 OPT' ≤ 2xOPT 증명

$$
\begin{aligned}
OPT' = t_i + T \;\le\; & t_i + T' \quad \text{—①} \\
= \;& t_i + \frac{\left(\sum_{j=1}^{n} t_j\right) - t_i}{m}, \quad \text{왜냐하면 } T' = \frac{\left(\sum_{j=1}^{n} t_j\right) - t_i}{m} \\
= \;& t_i + \frac{\left(\sum_{j=1}^{n} t_j\right)}{m} - \frac{t_i}{m} \\
= \;& \frac{1}{m}\sum_{j=1}^{n} t_j + \left(1 - \frac{1}{m}\right) t_i \\
\le \;& OPT + \left(1 - \frac{1}{m}\right) OPT \quad \text{—②} \\
= \;& \left(2 - \frac{1}{m}\right) OPT \\
\le \;& 2\,OPT
\end{aligned}
$$

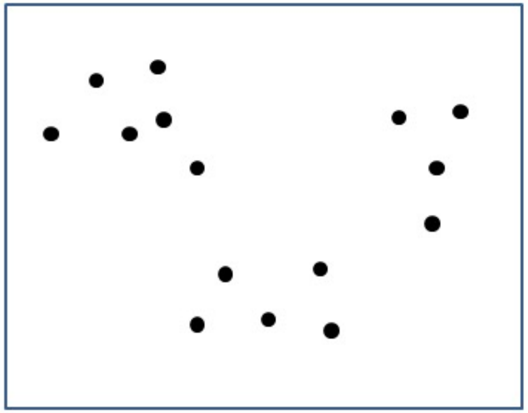
근사 비율

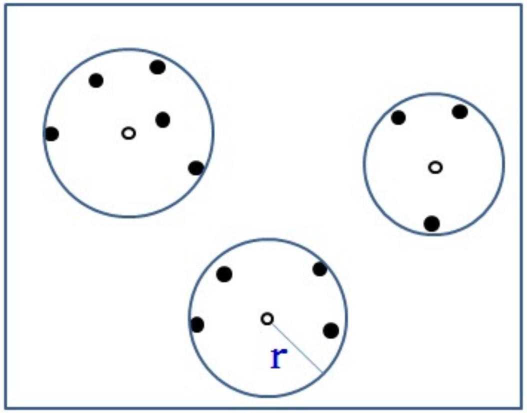
- 첫 번째 부등식 ①
  - 위의 그림에서 살펴본 T ≤ T' 을 이용한 것이다.

- 식 ②로의 변환
  - 최적해 OPT는 모든 작업의 수행 시간의 합을 기계의 수로 나눈 값 (평균 종료 시간)보다 같거나 크고 또한 하나의 작업 수행 시간과 같거나 크다는 것을 부등식에 반영한 것이다.

5. 클러스터링 문제

- n개의 점이 2차원 평면에 주어질 때, 이 점들 간의 거리를 고려하여 k개의 그룹으로 나누고자 한다.
  - 아래의 그림은 점들을 3개의 그룹으로 나눈 것을 보여준다.

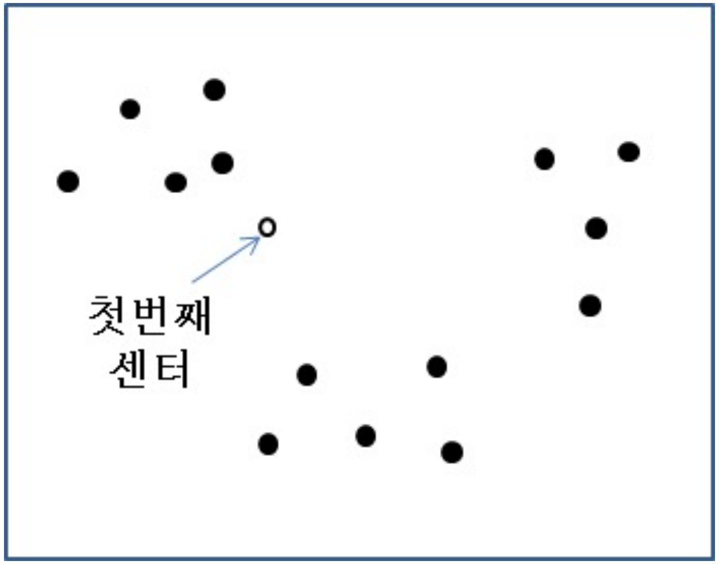




- 클러스터링 (Clustering) 문제
  - 입력으로 주어진 n개의 점을 k개의 그룹으로 나누고 각 그룹의 중심이 되는 k개의 점을 선택하는 문제이다.
  - 단, 가장 큰 반경을 가진 그룹의 직경이 최소가 되도록 k개의 점이 선택되어야 한다.

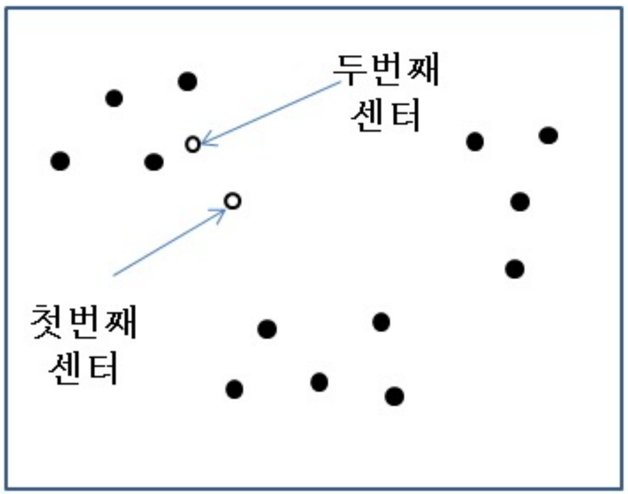
클러스터링 문제

- k개의 센터 선택 방법
  - n개의 점 중에서 k개의 센터를 선택하여야 하는데, 이를 한꺼번에 선택하는 것보다는 1개씩 선택하는 것이 쉽다.
  - 아래의 그림 같이 1 번째 센터가 랜덤하게 정해졌다고 가정해 보자.

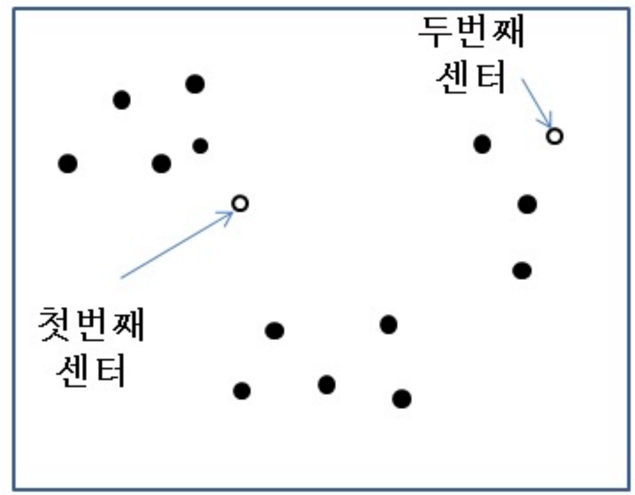


클러스터링 문제

- 어느 점이 두 번째 센터가 되면 좋을까?
- 첫 번째 센터에서 가장 가까운 점과 가장 먼 점 중에서 어느 점이 좋을까?



첫번째 센터에서 가장 가까운 점

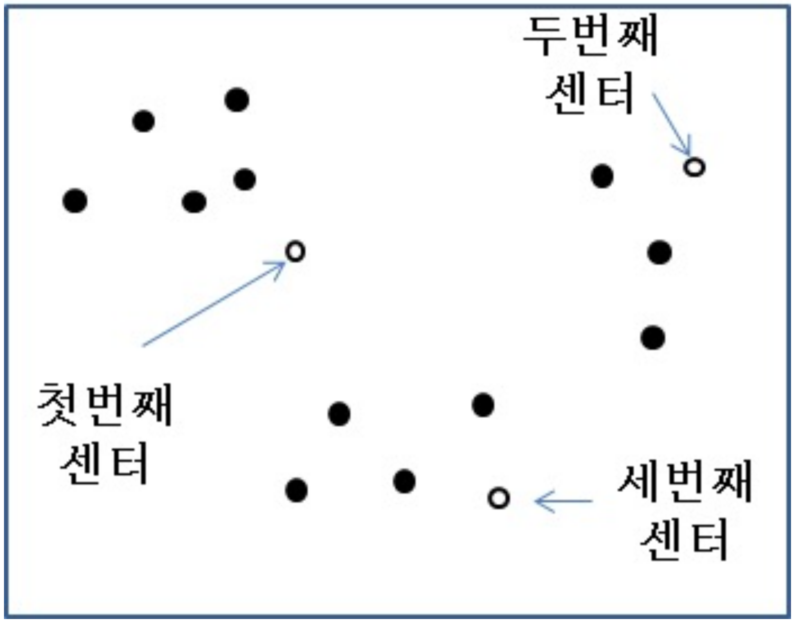


첫번째 센터에서 가장 먼 점

- 위의 그림에서 보면 2개의 센터가 서로 가까이 있는 것보다 멀리 떨어져 있는 것이 좋다.

클러스터링 문제

– 그 다음 세 번째 센터는 첫 번째와 두 번째 센터에서 가장 멀리 떨어진 점을 선택한다.



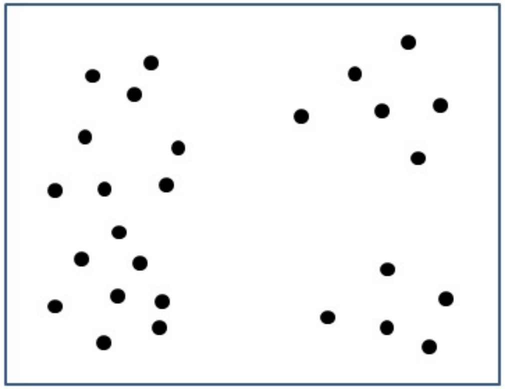
Approx_k_Clusters

입력: 2차 평면상의 n개의 점 x_i, i=0, 1, …,n-1, 그룹의 수 k > 1
출력: k개의 점의 그룹 및 각 그룹의 센터

1. C[1] = r, 단, x_r은 n개의 점 중에서 랜덤하게 선택된 점이다.
2. for j = 2 to k {
3. for i = 0 to n-1 {
4. if ($x_i \neq$ 센터)
5. x_i와 각 센터까지의 거리를 계산하여, x_i와 가장 가까운 센터까지의 거리를 D[i]에 저장한다.
}
6. C[j] = i, 단, i는 배열 D의 가장 큰 원소의 인덱스이고, x_i는 센터가 아니다
.
}
7. 센터가 아닌 각 점 x_i로부터 위에서 찾은 k개의 센터까지 거리를 각각 계산하고 그 중에 가장 짧은 거리의 센터를 찾는다. 이때 점 x_i는 가장 가까운 센터의 그룹에 속하게 된다.
8. return 배열 C와 각 클러스터에 속한 점들의 리스트

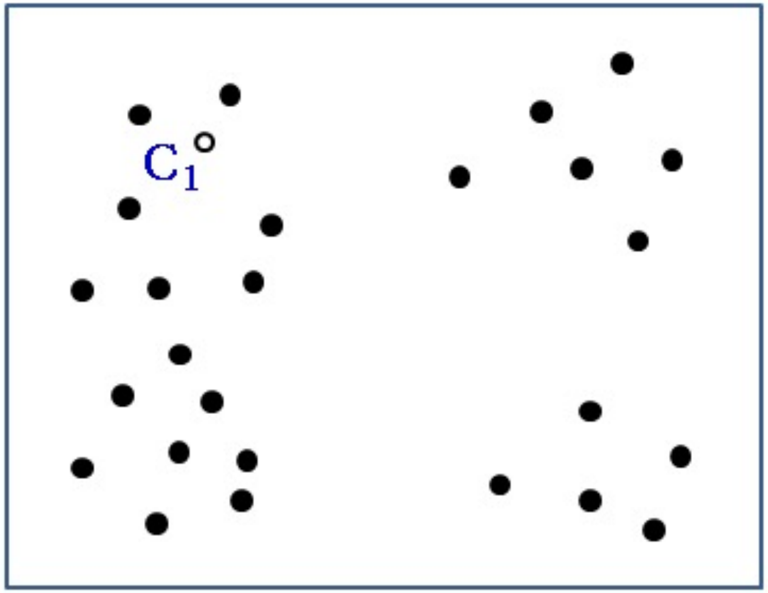
알고리즘 수행 과정

- Approx_k_Clusters 알고리즘 수행
  - 다음 그림에 주어진 점들을 Approx_k_Clusters 알고리즘을 수행하여 4개의 센터를 찾는다.
  - 나머지 점들을 4개의 센터를 기준으로 4개의 그룹으로 나눈다.



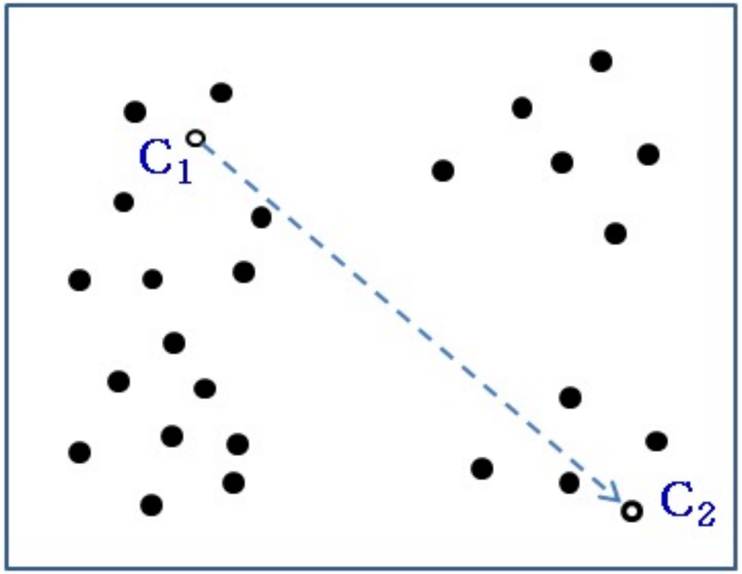
알고리즘 수행 과정

- Line 1
  - 아래의 그림처럼 임의의 점 하나를 첫 번째 센터 C_1으로 정한다.



알고리즘 수행 과정

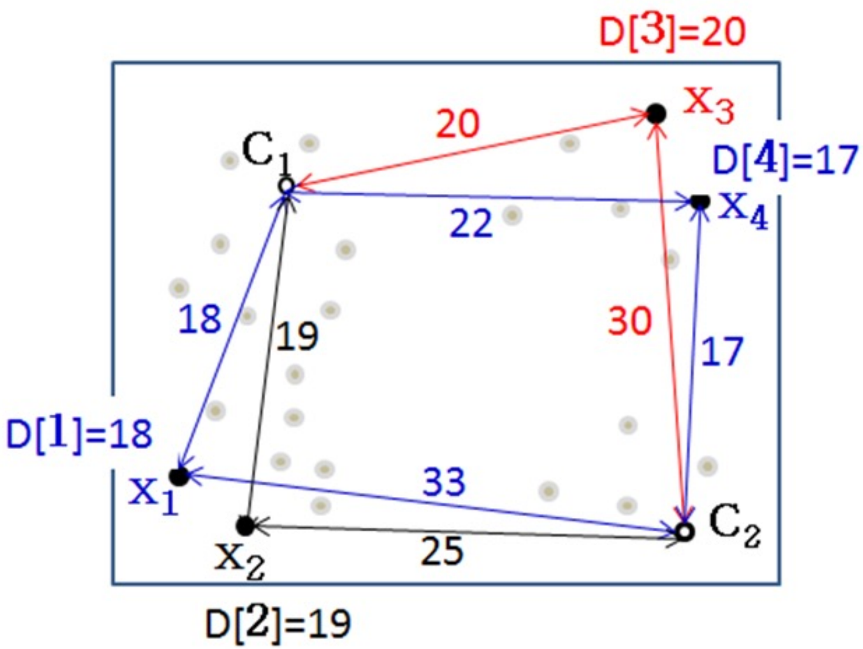
- Line 3~5
  - C_1이 아닌 각 점 x_i에서 C_1까지의 거리 D[i]를 계산한다.
- Line 6
  - C_1로부터 거리가 가장 먼 점을 다음 센터 C_2로 정한다.



알고리즘 수행 과정

- Line 2
  - j=3이 된다. 즉, 3 번째 센터를 찾는다.
- Line 3~5
  - C_1과 C_2를 제외한 각 점 x_i에서 각각 C_1과 C_2까지의 거리를 계산하여 그 중에서 작은 값을 D[i]로 정한다.
- Line 6
  - 배열 D에서 가장 큰 값을 가진 원소의 인덱스가 i라고 하면, 점 x_i가 다음 센터 C_3이 된다.
  - 단, x_i는 이전에 센터가 아닌 점이다.

알고리즘 수행 과정



D[1]=18, min{dist(x_1,C_1), dist(x_1,C_2)}
=min{18, 33}이므로

D[2]=19, min{dist(x_2,C_1), dist(x_2,C_2)}
=min{19, 25}이므로

D[3]=20, min{dist(x_3,C_1), dist(x_3,C_2)}
=min{20, 30}이므로

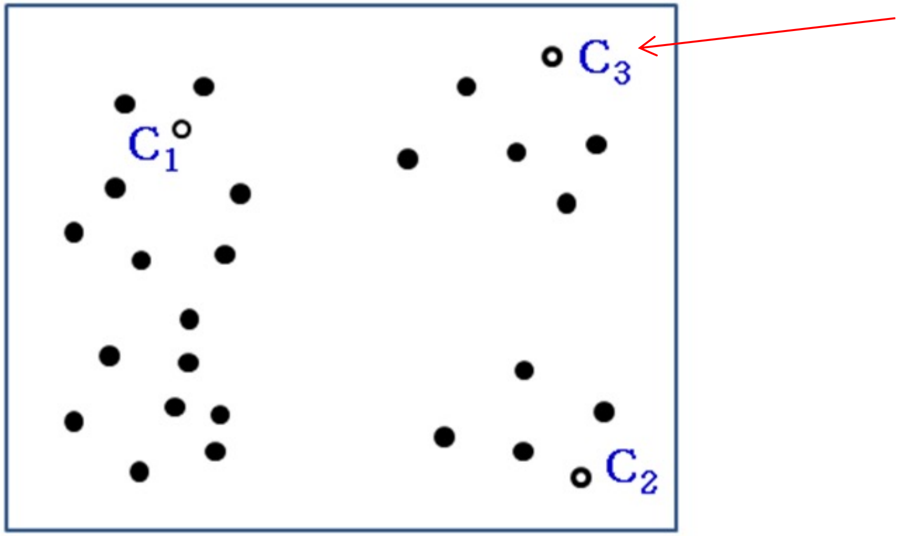
D[4]=17, min{dist(x_4,C_1), dist(x_4,C_2)}
=min{22, 17}이므로

이외의 다른 모든 점 x_i에 대해서 D[i]는 17보다 작다고 가정한다.

- 그림은 4개의 점 x_1, x_2, x_3, x_4에 대해 각각 D[x_1], D[x_2], D[x_3], D[x_4]가 계산된 것을 보여주고 있다.
  - 단, dist(x,C)는 점 x와 센터 C 사이의 거리이다.

알고리즘 수행 과정

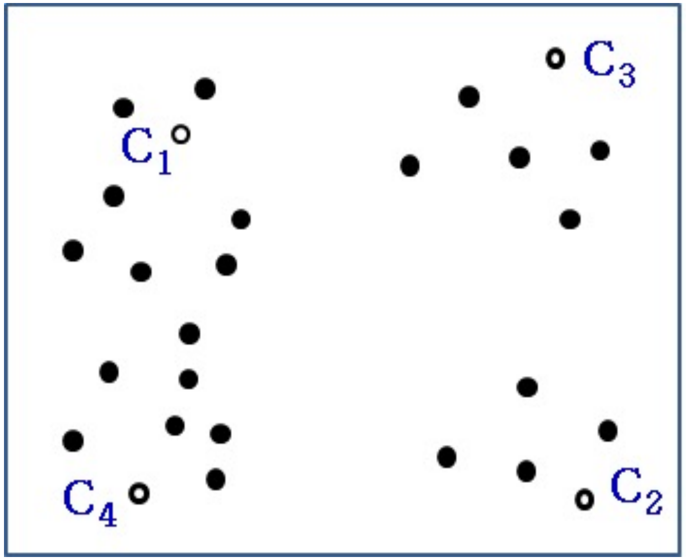
- 센터가 아닌 점 중에서 D[3]이 가장 큰 값이므로, 점 x_3이 세 번째 센터인 C_3으로 정해진다.



C_1과 C_2로부터 가장 먼 점

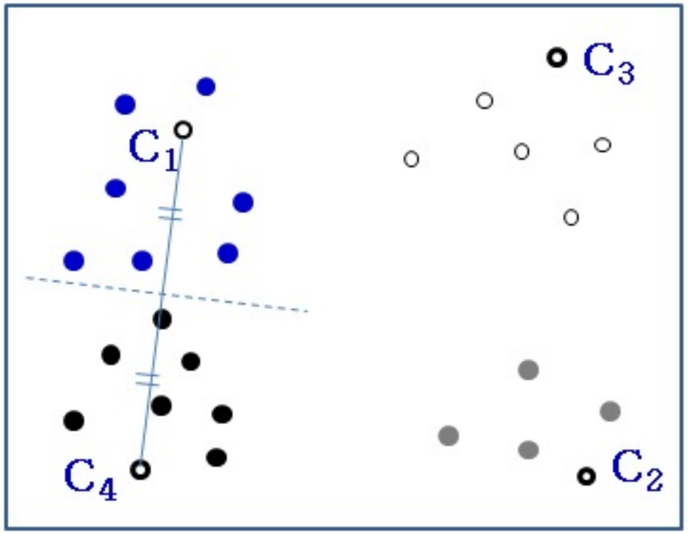
알고리즘 수행 과정

- Line 2
  - j=4가 된다. 즉, 4 번째 센터를 찾는다.
- Line 3~5
  - C_1, C_2, C_3을 제외한 각 점 x_i에서 각각 C_1, C_2, C_3까지의 거리를 계산하여 그 중에서 작은 값을 D[i]로 정한다.
- Line 6
  - 배열 D에서 가장 큰 값을 가진 원소의 인덱스가 i라고 하면, 점 x_i가 다음 센터 C_4로 된다.
  - 단, x_i는 이전에 센터가 아닌 점이다.



알고리즘 수행 과정

- Line 7
  - 센터가 아닌 각 점 x_i로부터 위에서 찾은 4개의 센터까지 거리를 각각 계산하고 그 중에 가장 짧은 거리의 센터를 찾는다.
  - 이때 점 x_i는 가장 가까운 센터의 그룹에 속하게 된다.
  - 그리고 각 점에서 가까운 센터를 찾으면 아래와 같이 4개의 그룹으로 나누어진다.



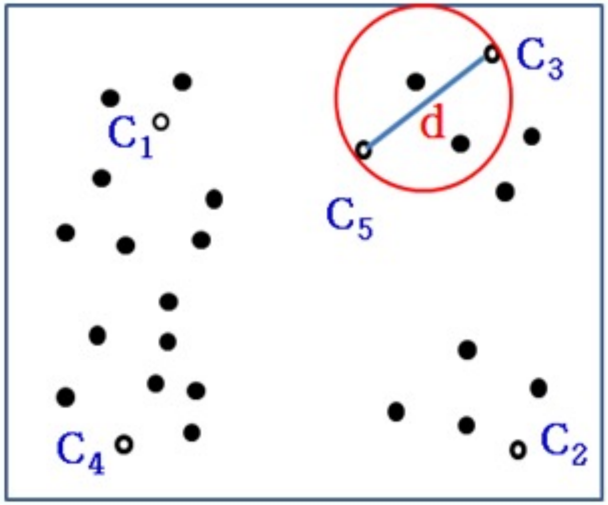
- Line 8
  - 4개의 센터와 각 클러스터에 속한 점들의 리스트를 리턴한다.

시간복잡도

- Line 1
  - 임의의 점을 선택하므로 O(1) 시간이 걸린다.
- Line 3~5
  - 각 점에서 각 센터까지의 거리를 계산하므로 O(kn) 시간이 걸리며,
- Line 6
  - 그 중에서 최대값을 찾으므로 O(n) 시간이 걸린다.
- line 2의 for-루프는 (k-1)회 반복되므로, line 6까지의 수행 시간
  - O(1)+(k-1)x(O(kn)+O(n))이다.
- Line 7
  - 센터가 아닌 각 점으로부터 k개의 센터까지의 거리를 각각 계산하면서 최솟값을 찾는 것이므로 O(kn) 시간이 소요된다.
- 시간복잡도
  - O(1)+(k-1)x(O(kn)+O(n))+O(kn) = O(k^2n)이다.

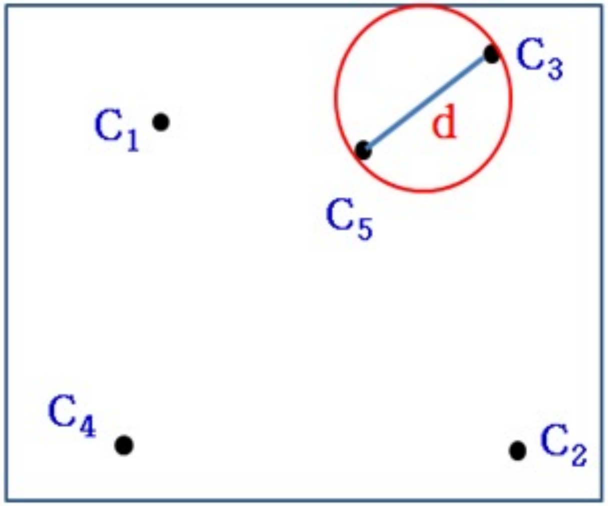
근사 비율

- 먼저 최적해가 만든 그룹 중에서 가장 큰 직경을 OPT라고 한다.
- 그리고 OPT의 하한을 간접적으로 찾기 위해서, Approx_k_Clusters 알고리즘이 k개의 센터를 모두 찾고 나서 (k+1)번째 센터를 찾은 상황을 생각해본다.
- 아래의 그림은 k=4이라서 4개의 센터를 찾은 후, 1개의 센터 C_5를 추가로 찾은 것을 나타낸다.



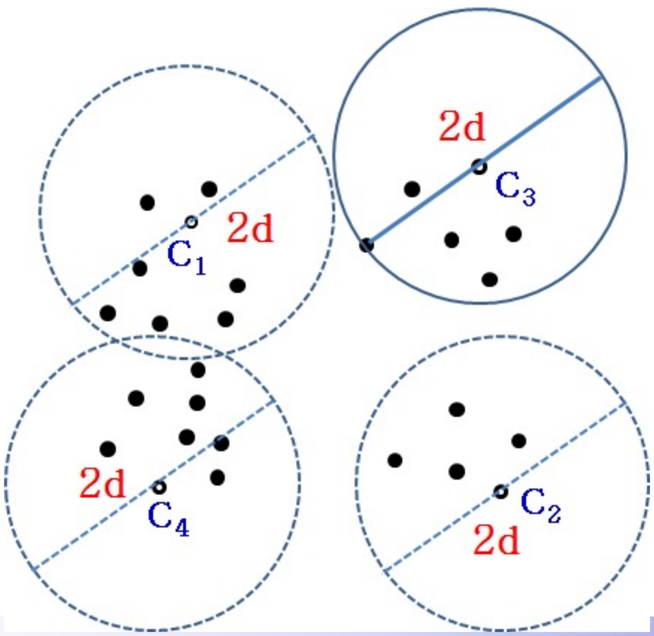
근사 비율

- C_5에서 가장 가까운 센터인 C_3까지의 거리를 d라고 하자.
- 클러스터링 문제의 최적해를 계산하는 어떤 알고리즘이라도 위의 5개의 센터 점을 (k=4이니까) 4개의 그룹으로 분할해야 한다.
- 따라서 5개의 센터 중에서 2개는 하나의 그룹에 속해야만 한다.
- 그림에서는 C_3과 C_5가 하나의 그룹에 속한 것을 보이고 있다.
- 최적해의 가장 큰 그룹의 직경인 OPT는 d보다 작을 수는 없다. 즉, OPT≥d이다.



근사 비율

- Approx_k_Clusters 알고리즘이 계산한 근사해의 가장 큰 그룹의 직경 OPT′는 d와 어떤 관계인지 살펴보자.
- 다음그림을 살펴보면, 가상의 다음 센터 C_5와 C_3사이의 거리가 d이므로, 센터가 아닌 어떤 점이라도 자신으로부터 가장 가까운 센터까지의 거리가 d보다 크지 않다.
- 따라서 각 그룹의 센터를 중심으로 반경 d 이내에 그룹에 속하는 점들이 위치한다. 따라서 OPT′≤2d이다.



근사 비율

- 즉, OPT ≥ d이고, OPT′ ≤ 2d이므로,
  2OPT ≥ 2d ≥ OPT′ 이다.

- 따라서 Approx_k_Clusters 알고리즘의 근사 비율은 2를 넘지 않는다.

응용

- 클러스터링 알고리즘은 대단히 많은 분야에서 활용된다. 이는 일반적으로 어떠한 데이터라도 유사한 특성 (유사도)을 가진 부분적인 데이터로 분할하여 분석할 때에 사용될 수 있기 때문이다.
- 추천 시스템
- 데이터 마이닝 (Data Mining)
- VLSI 설계
- 병렬 처리 (Parallel Processing)
- 웹 탐색 (Web Searching)
- 데이터베이스
- 소프트웨어 공학 (Software Engineering)
- 컴퓨터 그래픽스 (Computer Graphics)

응용

- 패턴 인식 (Pattern Recognition)
- 유전자 분석 (Gene Analysis)
- 소셜 네트워크 (Social Network) 분석
- 도시 계획, 사회학, 심리학, 의학, 금융, 통계, 유통 등 많은 분야에서 데이터의 그룹화는 매우 중요한 문제이다.

요약

- 근사 알고리즘은 최적해의 값에 가까운 해인 근사해를 찾는 대신에 다항식 시간의 복잡도를 가진다.
- 근사 비율 (approximation ratio)은 근사해가 얼마나 최적해에 가까운지를 나타내는 근사해의 값과 최적해의 값의 비율로서, 1.0에 가까울수록 실용성이 높은 알고리즘이다.
- 여행자 문제를 위한 근사 알고리즘은 최소 신장 트리의 모든 점을 연결하는 특성과 최소 가중치의 특성을 이용한다. 근사비율은 2이다.
- 정점 커버 문제를 위한 근사 알고리즘은 그래프에서 극대 매칭을 이용하여 근사해를 찾는다. 근사비율은 2이다.

요약

- 통 채우기 문제는 최초 적합 (first fit), 다음 적합 (next fit), 최선 적합 (best fit), 최악 적합 (worst fit)과 같은 그리디 알고리즘으로 근사해를 찾는다. 근사비율은 각각 2이다.
- 작업 스케줄링 문제는 가장 빨리 끝나는 기계에 새 작업을 배정하는 그리디 알고리즘으로 근사해를 찾는다. 근사비율은 2이다.
- 클러스터링 문제는 현재까지 정해진 센터에서 가장 멀리 떨어진 점을 다음 센터로 정하는 그리디 알고리즘으로 근사해를 찾는다. 근사비율은 2이다.