Chapter 8
근사 알고리즘

차례

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 증명하지 못했다.

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

NP-완전 문제의 해결

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

- 근사 알고리즘은 NP-완전 문제를 해결하기 위해 3번째 것을 포기한다.
  - 최적해에 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 가까운지를 나타내는 근사 비율 (Approximation Ratio)을 알고리즘과 함께 제시하여야
- 근사 비율은 근사해의 값과 최적해의 값의 비율로서, 1.0에 가까울수록 정확도가 높은 알고리즘
- 근사 비율을 계산하려면 최적해를 알아야 하는 모순 발생
- 최적해를 대신할 수 있는 '간접적인' 최적해를 찾고, 이를 최적해로 삼아서 근사 비율을 계산

8.1 여행자 문제

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

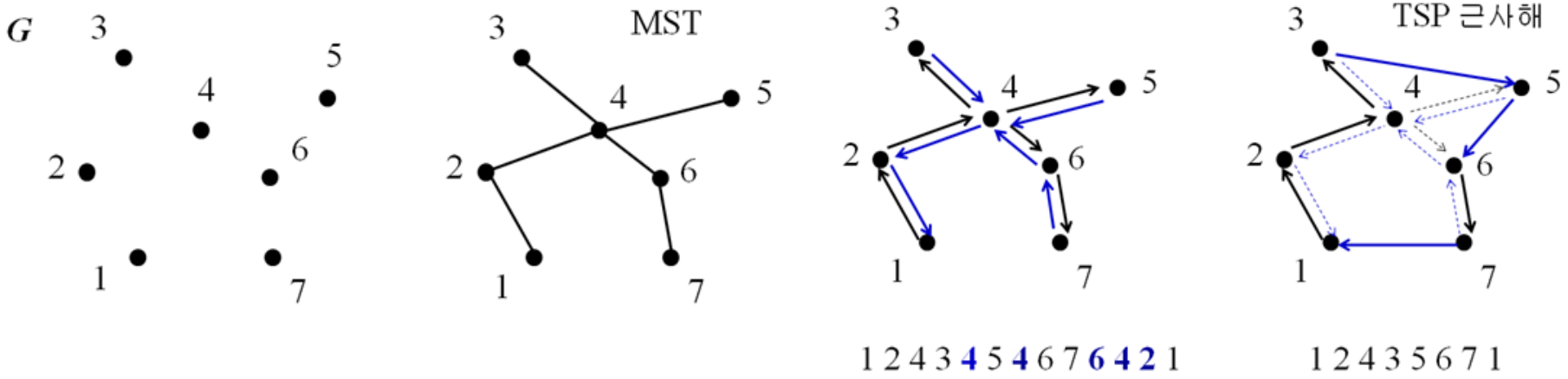
- 여행자 문제의 조건
  - 도시 A에서 도시 B로 가는 거리는 도시 B에서 도시 A로 가는 거리와 같다. (대칭성)
  - 도시 A에서 도시 B로 가는 거리는 도시 A에서 다른 도시 C를 경유하여 도시 B로 가는 거리보다 짧다. (삼각 부등식 특성)

여행자 문제

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

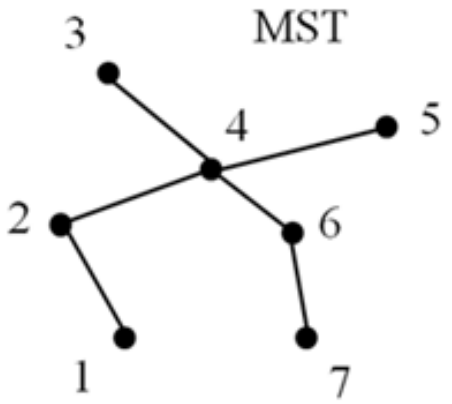
MST를 활용한 근사해 찾는 과정

- MST를 활용하여 여행자 문제의 근사해를 찾기 위해 삼각 부등식 원리를 적용

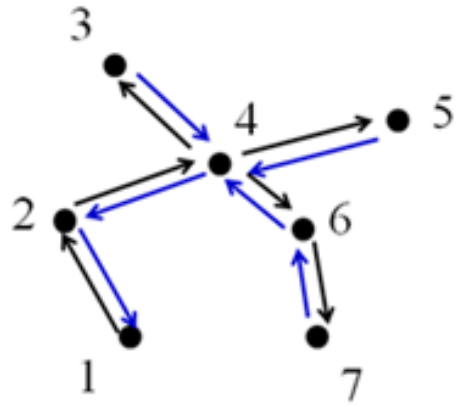


MST에서 방문 순서 찾기

- 그래프에서 크러스컬 또는 프림 알고리즘을 이용하여 최소 신장 트리 MST를 찾는다.

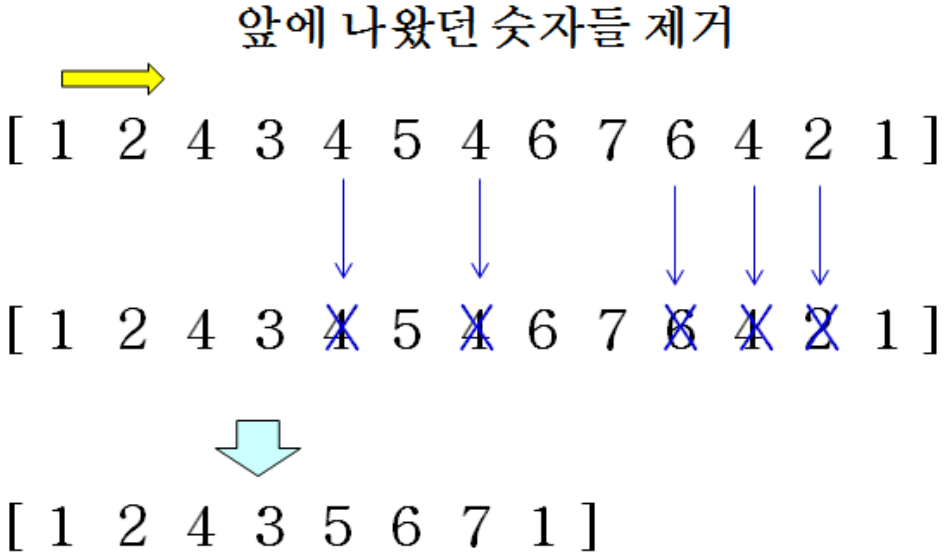


- 임의의 도시 (그림에서는 도시 1)에서 출발하여 트리의 간선을 따라서 모든 도시를 방문하고 돌아오는 도시의 방문 순서를 찾는다.
  - [1 2 4 3 4 5 4 6 7 6 4 2 1].



방문 순서에서 중복 방문 점 제거

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



- 중복하여 방문하는 도시를 제거는 과정에 삼각형 부등식 원리 적용

알고리즘

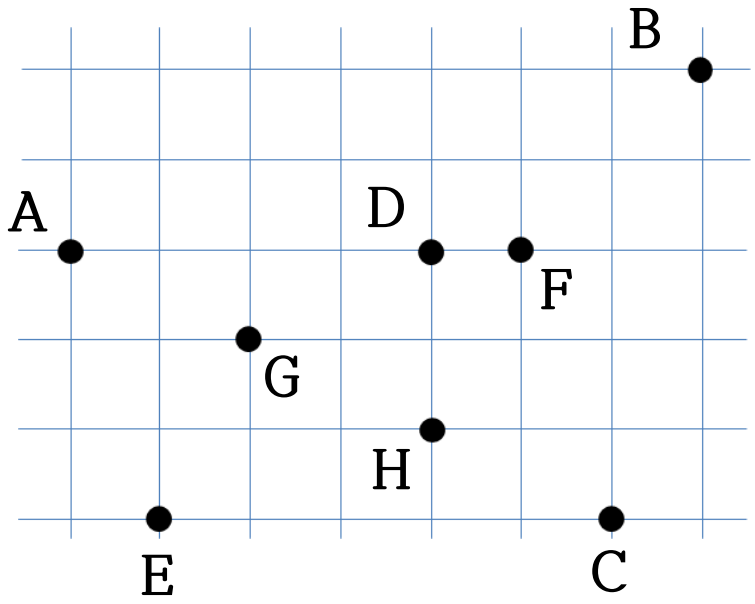
Approx_MST_TSP

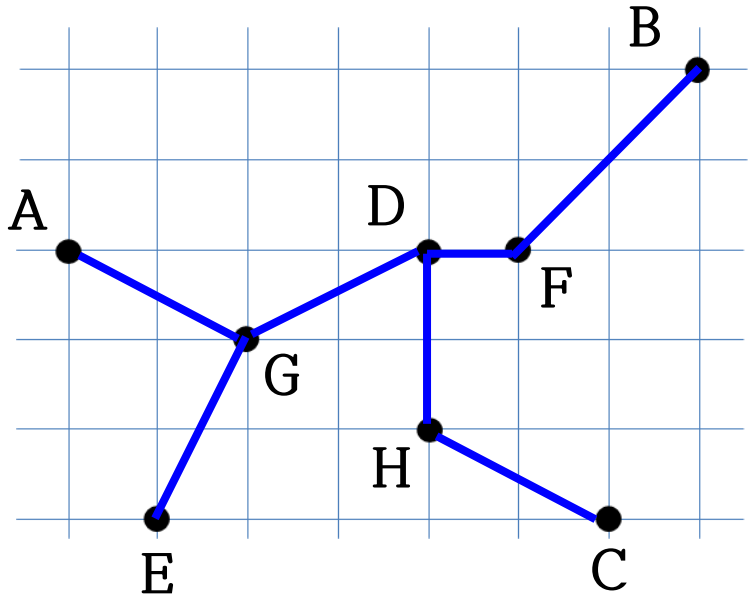
입력: n개의 도시, 각 도시간의 거리

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

1. 입력에 대하여 MST를 찾는다.
2. MST에서 임의의 도시로부터 출발하여 트리의 간선을 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.)

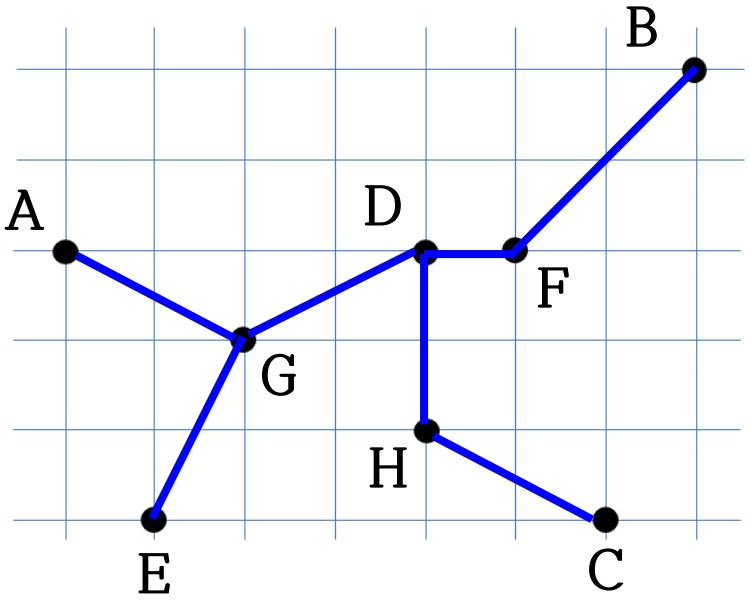
MST 찾기

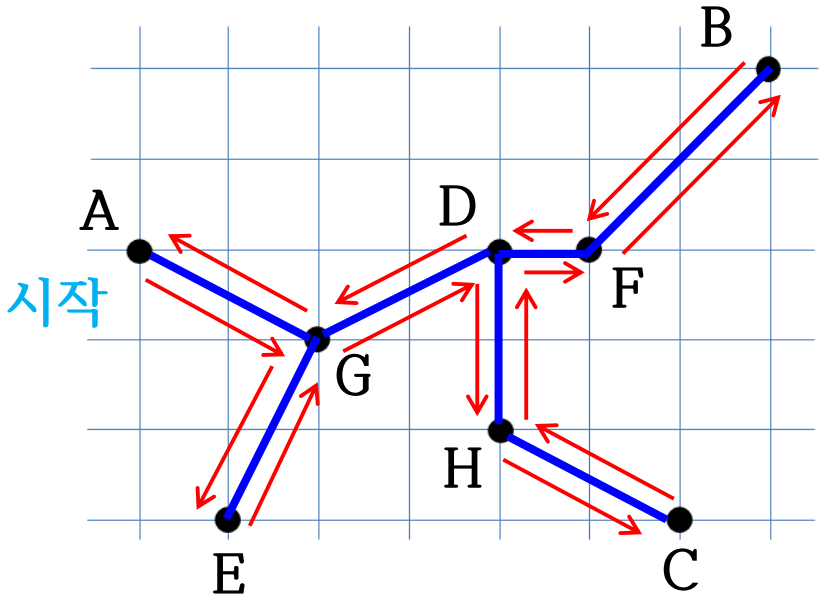




MST

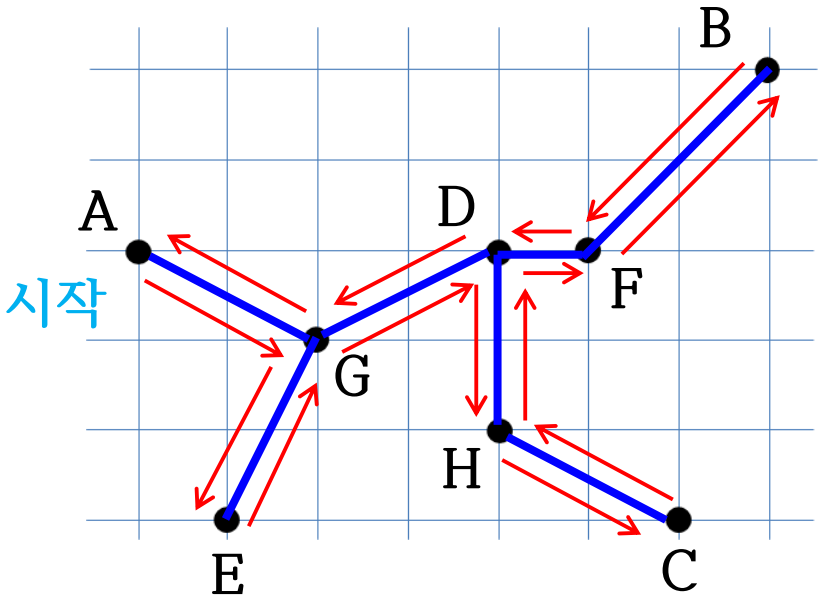
MST 방문 순서



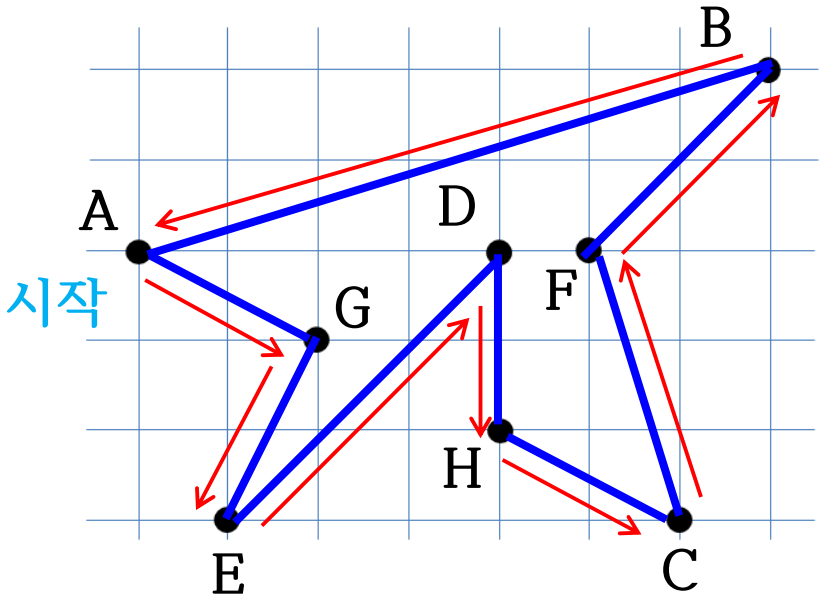


A G E G D H C H D F B F D G A

중복 방문 제거

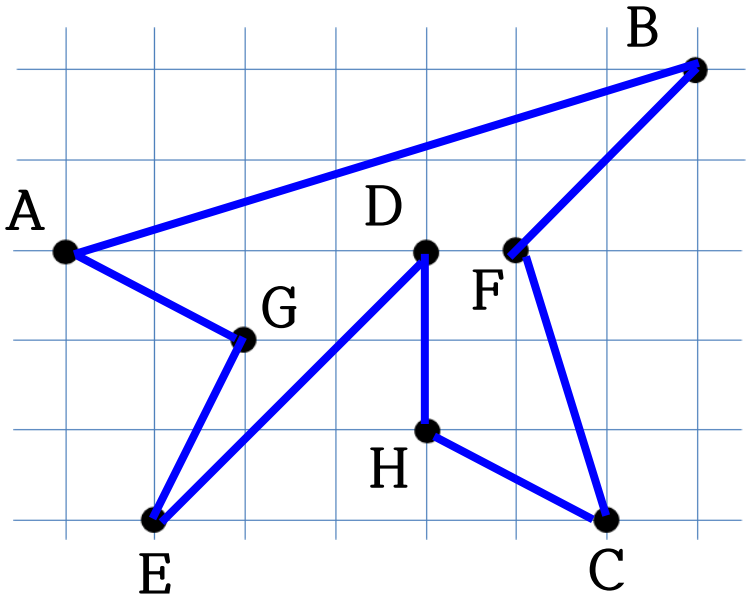


A G E G D H C H D F B F D G A

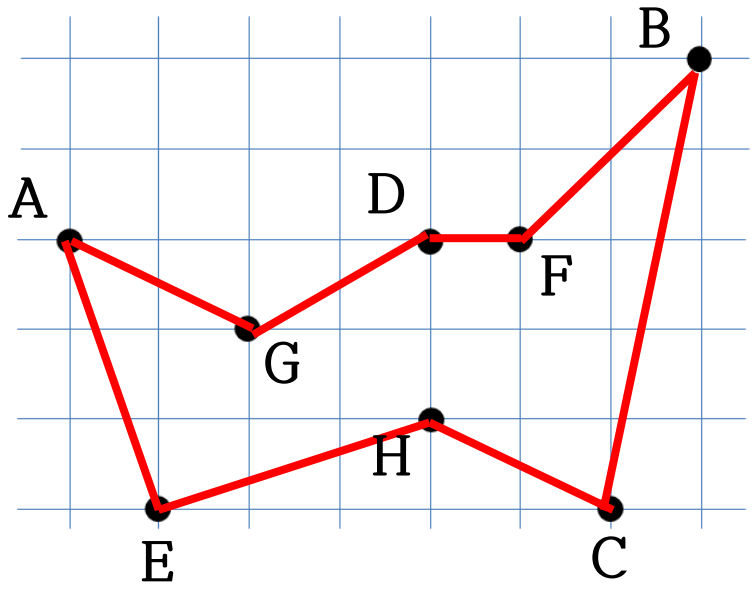


A G E G D H C H D F B F D G A

근사해 vs. 최적해



근사해



최적해

시간 복잡도

- Line 1
  - 크러스컬 알고리즘: $O(m \log m)$, m은 간선의 수
  - 프림 알고리즘: $O(n^2)$, n은 점의 수
- Line 2
  - 트리 간선을 따라서 도시 방문 순서를 찾는 데는 $O(n)$ 시간
  - 왜냐하면 트리의 간선 수가 (n-1)이기 때문
- Line 3
  - 방문 순서를 따라가며 단순히 중복된 도시를 제거하므로 $O(n)$ 시간
- 시간 복잡도: 크러스컬 또는 프림 알고리즘의 시간 복잡도

근사 비율

- 여행자 문제의 최적해를 실질적으로 알 수 없으므로, '간접적인' 최적해인 MST 간선의 가중치의 합(M)을 최적해의 값으로 활용
  - 왜냐하면 실제의 최적해의 값이 M보다 항상 크기 때문

- Approx_MST_TSP 알고리즘이 계산한 근사해의 값은 2M보다는 크지 않다.
  - Line 2에서 MST의 간선을 따라서 도시 방문 순서를 찾을 때 각 간선이 정확히 2번 사용되었으므로 경로의 총 길이는 2M

근사 비율

- Line 3에서는 삼각 부등식의 원리를 이용하여 지름길로 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 = 2.0보다 크지 않다.

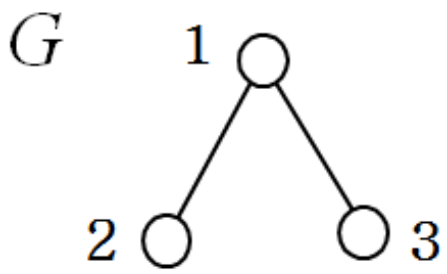
- 근사해의 값이 최적해의 값의 2배를 넘지 않는다.

8.2 정점 커버 문제

- 정점 커버 (Vertex Cover)
  - 주어진 그래프 G=(V, E)에서 각 간선의 양 끝점들 중에서 적어도 하나의 끝점을 포함하는 점들의 집합들 중에서 최소 크기의 집합을 찾는 문제

- 그래프의 모든 간선이 정점 커버에 속한 점에 인접해 있다.
  - 정점 커버에 속한 점들로 그래프의 모든 간선을 '커버'하는 것

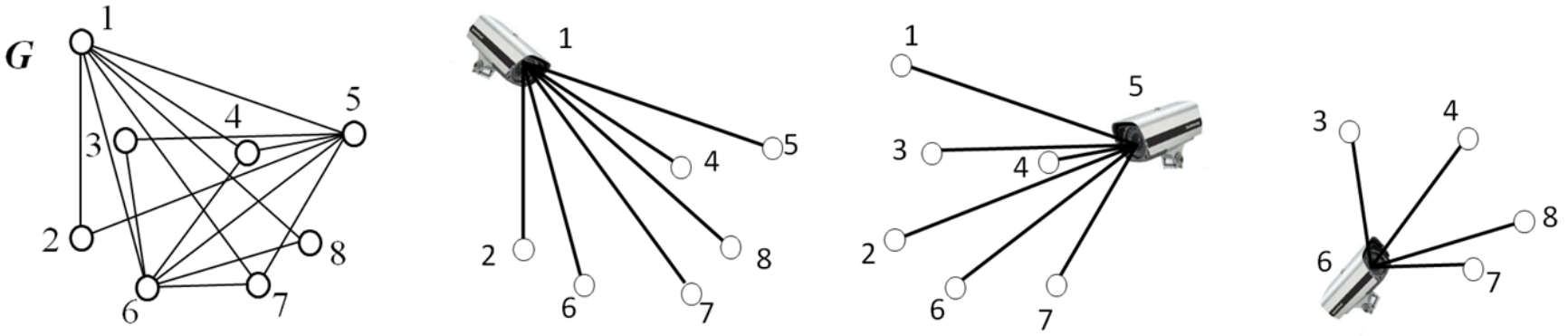
정점 커버 예제



- 위의 그래프에서 {1, 2, 3}, {1, 2}, {1, 3}, {2, 3}, {1}이 각각 정점 커버
- {2} 또는 {3}은 정점 커버가 아니다.
  - {2}는 간선 (1, 3)을 커버하지 못하고,
  - {3}은 간선 (1, 2)를 커버하지 못한다.
- 정점 커버 문제의 해는 {1}

커버의 의미

- G는 어느 건물의 내부도면
- 건물의 모든 복도를 감시하기 위해 가장 적은 수의 CCTV 카메라를 설치하려고 한다.
- 3대의 카메라를 각각 점 1, 5, 6에 설치하면 모든 복도(간선)을 '커버'한다.

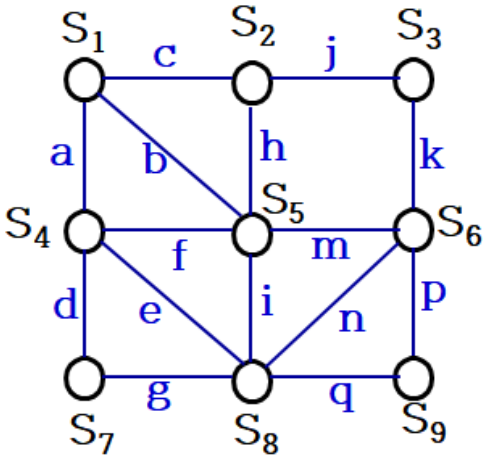


집합 커버로 해결하기

- 집합 커버 (Set Cover) 문제
  - n개의 원소를 가진 집합 U가 있고,
  - U의 부분집합들을 원소로 하는 집합 F가 주어질 때,
  - F의 원소들인 집합들 중에서 어떤 집합들을 선택하여 합집합하면 U와 같게 되는가?
  - 집합 F에서 선택하는 집합들의 수를 최소화하는 문제

- 정점 커버 문제의 입력 그래프를 집합 커버 문제의 입력으로 변환하여 SetCover 알고리즘으로 해를 찾아서, 그 해를 정점 커버의 해로 삼는다.

정점 커버 입력을 집합 커버 입력으로 변환

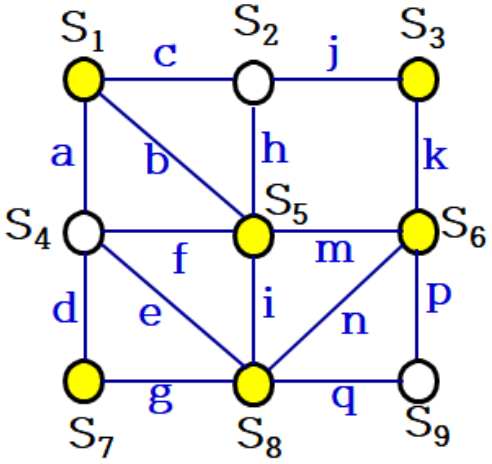


U = {a, b, c, d, e, f, g, h, i, j, k, m, n, p, q}

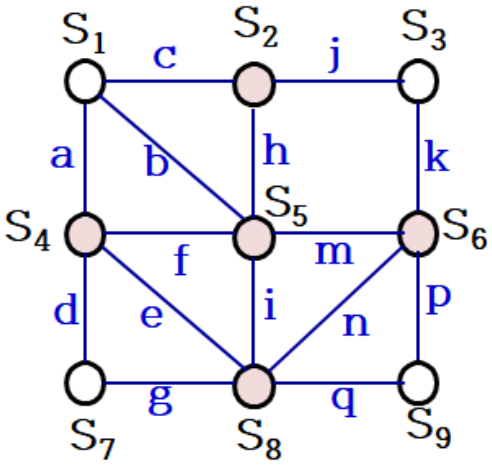
F = {S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9}

S_1 = {a, b, c}
S_2 = {c, h, j}
S_3 = {j, k}
S_4 = {a, d, e, f}
S_5 = {b, f, h, i, m}
S_6 = {k, m, n, p}
S_7 = {d, g}
S_8 = {e, g, i, n, q}
S_9 = {p, q}

집합 커버의 근사해 vs. 최적해



근사해



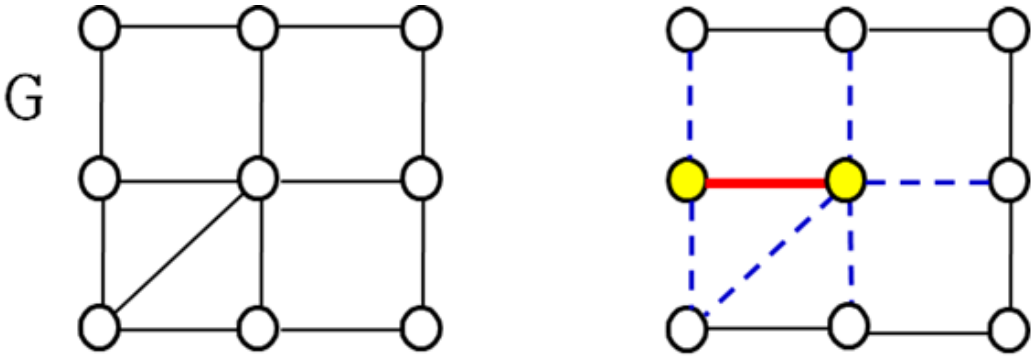
최적해

- 집합 커버의 근사 비율은 $K \ln n$이다.

집합 커버보다 작은 근사 비율

간선을 선택하여 정점 커버를 찾자.

- 선택한 간선(red)의 양 끝점에 인접한 모든 간선(점선)들은 양 끝 점에 의해 커버된다.



극대 매칭

- 극대 매칭 (Maximal Matching)
  - 매칭(matching)이란 각 간선의 양쪽 끝점들이 중복되지 않는 간선의 집합
  - 극대 매칭은 이미 선택된 간선에 기반을 두고 새로운 간선을 추가하려 해도 더 이상 추가할 수 없는 매칭을 말한다.

- 극대 매칭을 이용하여 정점 커버를 해결하자.
  - 간선의 양 끝점이 이미 커버된 간선의 끝점이 아닐 때에만 선택

알고리즘

Approx_Matching_VC

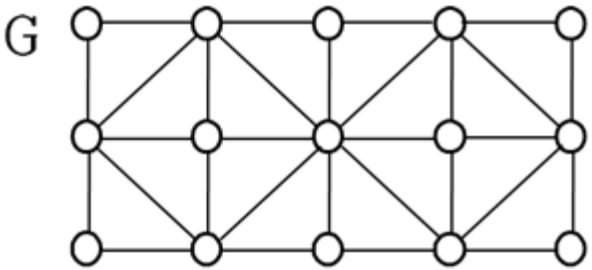
입력: 그래프 G=(V, E)

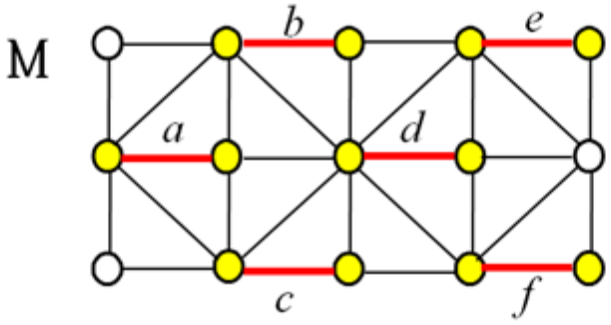
출력: 정점 커버

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

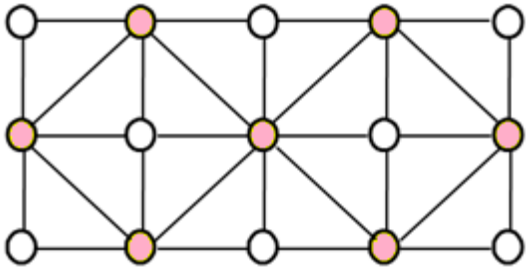
Approx_Matching_VC

- G에서 극대 매칭 M으로 간선 a, b, c, d, e, f 선택
- 근사해는 간선 a, b, c, d, e, f의 양 끝점들
  - 근사해: 12개
- 최적해: 7개





근사해


최적해

시간 복잡도

- 그래프에서 극대 매칭을 찾는 시간 복잡도와 동일
- 극대 매칭을 찾기 위해 하나의 간선을 선택할 때
  - 양 끝점이 이미 선택된 간선의 끝점인지를 검사해야 하므로 O(n)시간
- 입력 그래프의 간선 수가 m이면, 각 간선에 대해서 O(n) 시간
- 시간 복잡도는 O(n) x m = O(nm)

근사 비율

- 근사 비율을 계산하기 위해서 극대 매칭을 '간접적인' 최적해로 사용
  - 매칭에 있는 간선의 수를 최적해의 값으로 사용
  - 어떠한 정점 커버라도 극대 매칭에 있는 간선을 커버해야 하기 때문

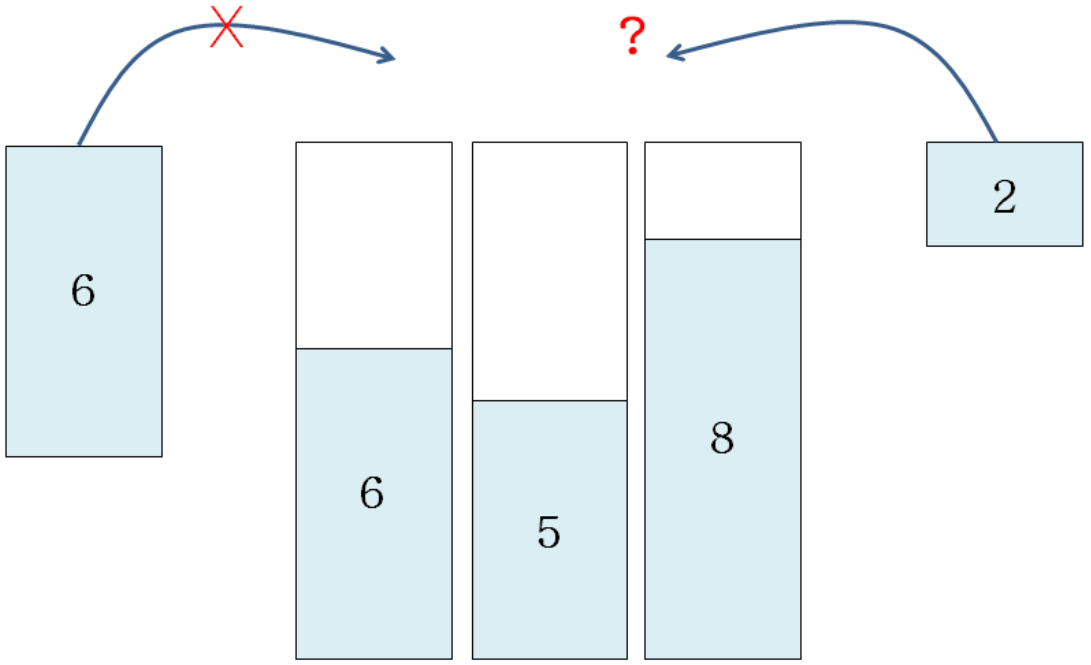
- Approx_Matching_VC 알고리즘
  - 극대 매칭의 각 간선의 양쪽 끝점들의 집합을 정점 커버의 근사해로서 반환하므로, 근사해의 값은 극대 매칭의 간선 수의 2배
  - 근사 비율 = (극대 매칭의 각 간선의 양 끝점들의 수)/(극대 매칭의 간선 수) = 2.0

8.3 통 채우기 문제

- 통 채우기(Bin Packing) 문제
  - 통 (bin)의 용량이 C일 때 n개의 물건을 가장 적은 수의 통에 채우는 문제
  - 단, 각 물건의 크기는 C보다 크지 않다.

물건의 크기가 6, 2일 때

- 각각 어느 통에 넣어야 할까?

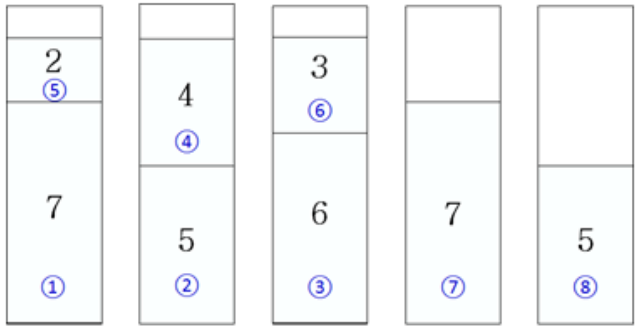


그리디 아이디어

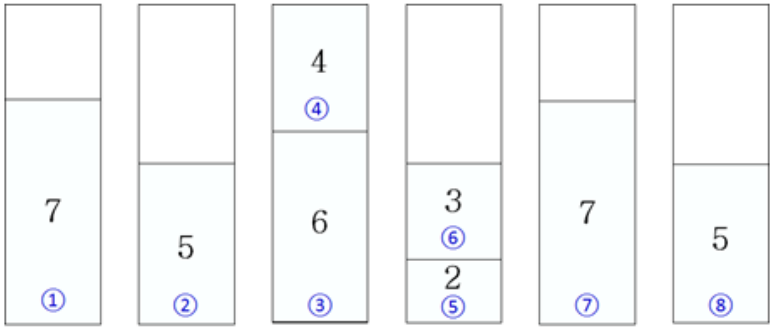
- 그리디 방법은 '무엇에 욕심을 낼 것인가'에 따라 4종류로 분류
  - 최초 적합 (First Fit): 첫 번째 통부터 차례로 살펴보며, 가장 먼저 여유가 있는 통에 새 물건을 넣는다.
  - 다음 적합 (Next Fit): 직전에 물건을 넣은 통에 여유가 있으면 새 물건을 넣는다.
  - 최선 적합 (Best Fit): 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 작은 통에 새 물건을 넣는다.
  - 최악 적합 (Worst Fit): 기존의 통 중에서 새 물건이 들어가면 남는 부분이 가장 큰 통에 새 물건을 넣는다.
- 각 방법으로 새 물건을 기존의 통에 넣을 수 없으면, 새로운 통에 새 물건을 담는다.

통 채우기 문제

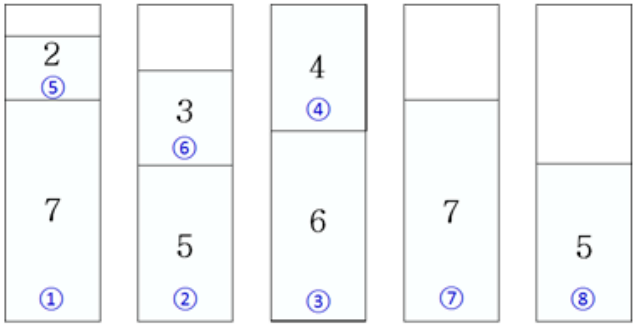
- 통의 용량 C=10, 물건의 크기 [7, 5, 6, 4, 2, 3, 7, 5]일 때, 최초 적합, 다음 적합, 최선 적합, 최악 적합을 적용한 결과



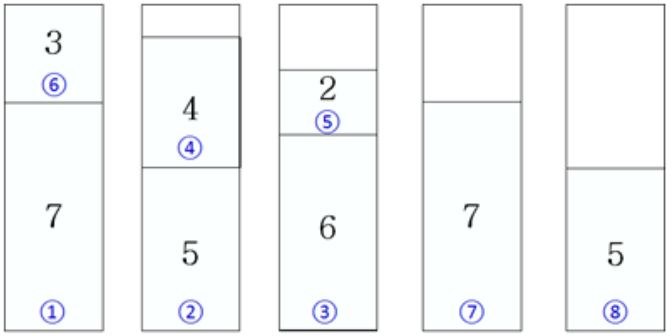
최초 적합



다음 적합



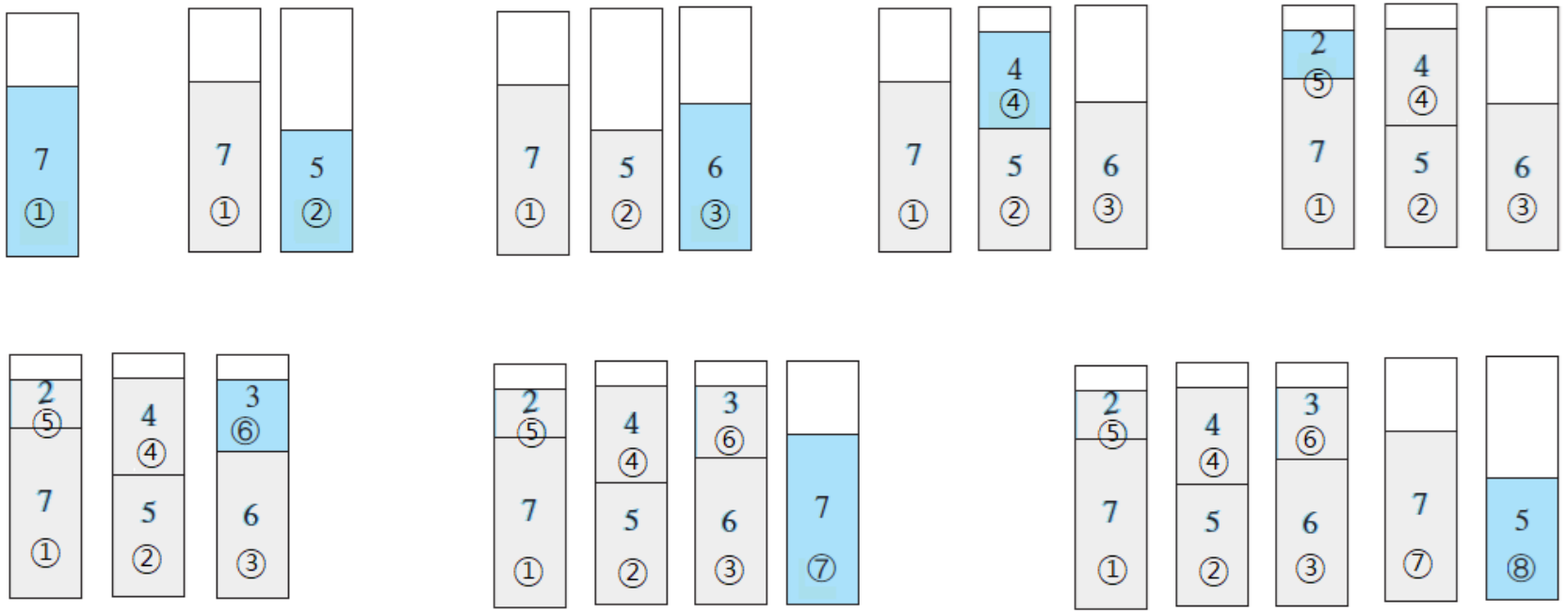
최선 적합



최악 적합

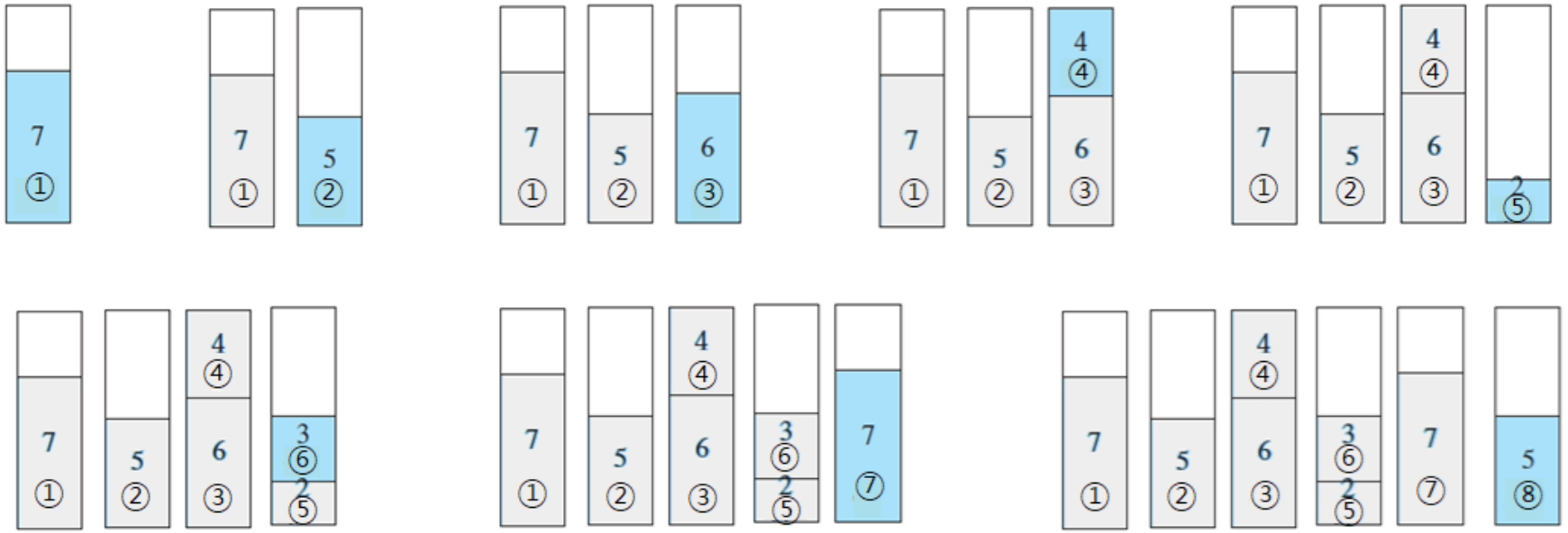
최초 적합

통의 용량 C = 10, 물건의 크기 [7, 5, 6, 4, 2, 3, 7, 5]일 때 그림에서 원 숫자들은 물건이 채워지는 순서



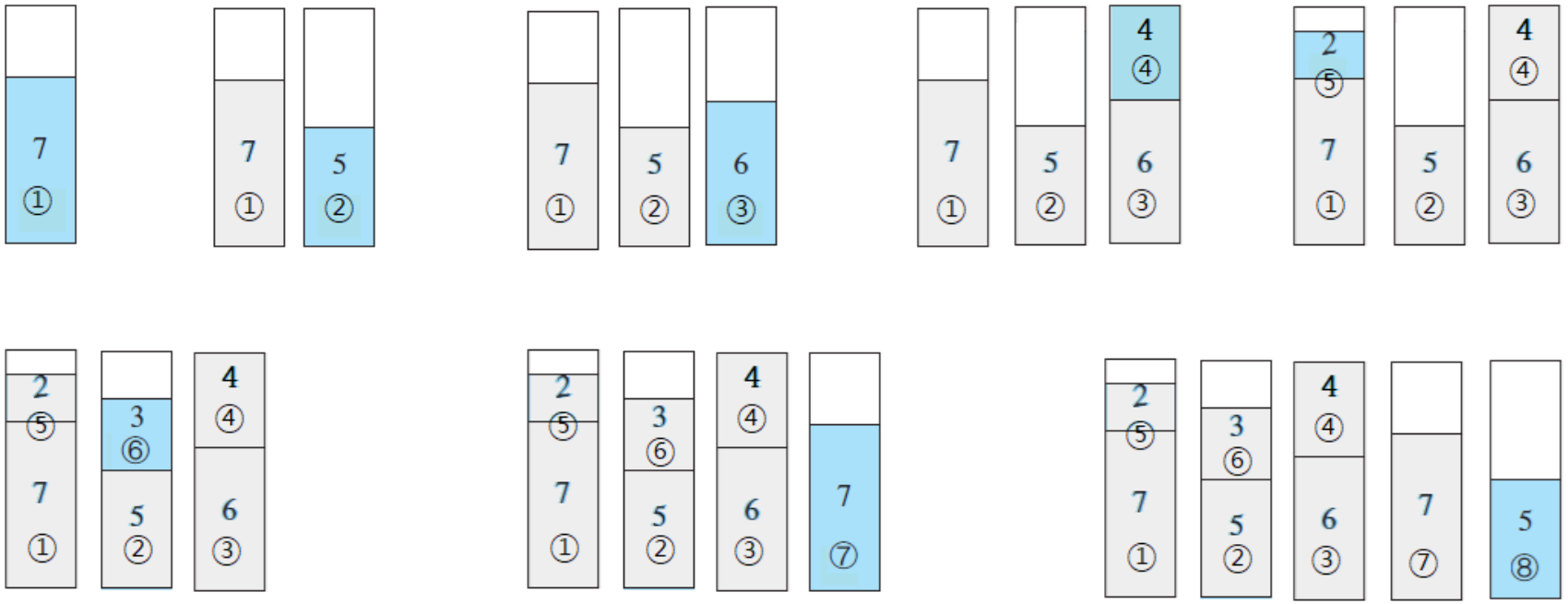
다음 적합

C = 10, 물건의 크기 [7, 5, 6, 4, 2, 3, 7, 5]일 때



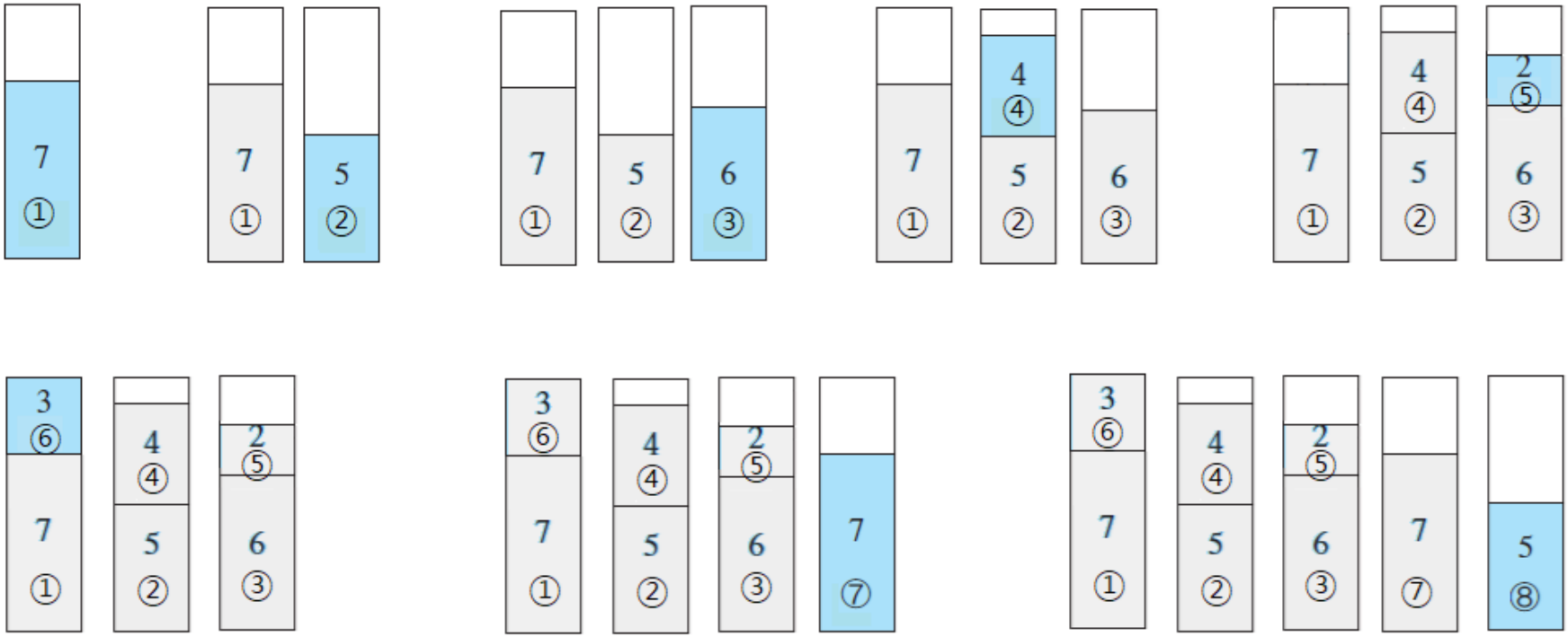
최선 적합

C = 10, 물건의 크기 [7, 5, 6, 4, 2, 3, 7, 5]일 때



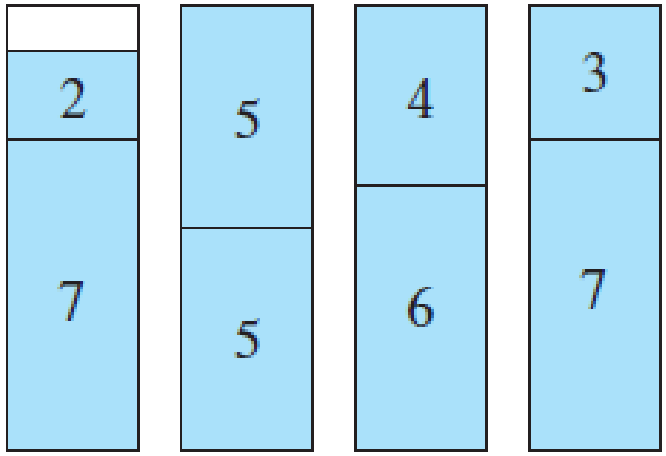
최악 적합

C = 10, 물건의 크기 [7, 5, 6, 4, 2, 3, 7, 5]일 때



최적해

C = 10, 물건의 크기 [7, 5, 6, 4, 2, 3, 7, 5]일 때



알고리즘

Approx_BinPacking

입력: n개 물건의 각각의 크기

출력: 모든 물건을 넣는데 사용된 통의 수 B

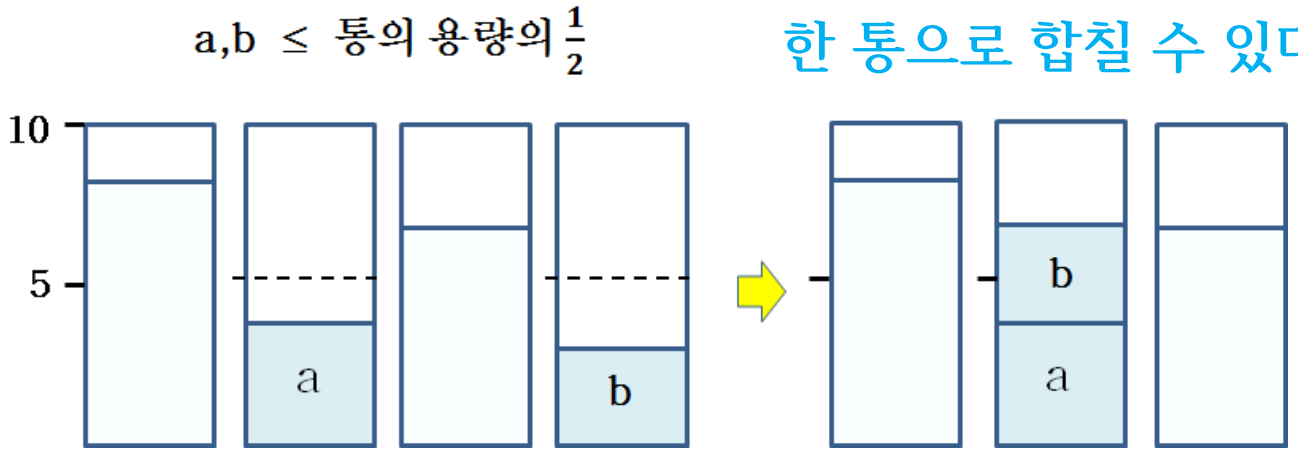
```
1. B = 0   // 사용된 통의 수
2. for i = 1 to n
3.     if 물건 i를 넣을 여유가 있는 기존의 통이 있으면
4.         그리디 방법에 따라 정해진 통에 물건 i를 넣는다.
5.     else
6.         새 통에 물건 i를 넣는다.
7.         B = B +1   // 통의 수를 1 증가시킨다.
8. return B
```

시간 복잡도

- 최초, 최선, 최악 적합
  - 새 물건을 넣을 때마다 기존의 통들을 살펴보아야
  - 통의 수가 n을 넘지 않으므로 수행 시간은 $O(n^2)$
- 다음 적합
  - 새 물건에 대해 직전에 사용된 통만을 살펴보므로 수행 시간은 $O(n)$

근사 비율

- 최초, 최선, 최악 적합
  - 모든 물건을 담는데 사용된 통의 수는 최적해에서 사용된 통의 수의 2배를 넘지 않는다.
  - 각 방법이 사용한 통을 살펴보면 2개 이상의 통이 1/2 이하로 차 있을 수 없다.



최초, 최선, 최악 적합 근사 비율

- 최적해에서 사용된 통의 수를 OPT라고 하면,

  OPT ≥ (모든 물건의 크기의 합)/C

  - 단, C는 통의 크기
- 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 물건들의 크기의 합은 ((OPT′-1) x C/2) 보다는 크다.

    (모든 물건의 크기의 합) > (OPT′-1) x C/2

- (모든 물건의 크기의 합) > (OPT′-1) x C/2

  ⇒ (모든 물건의 크기의 합)/C > (OPT′-1)/2 양변에 C 나누면

  ⇒ OPT > (OPT′-1)/2, OPT ≥ (모든 물건의 크기의 합)/C 이므로

  ⇒ 2OPT > OPT′-1 양변에 2 곱하면

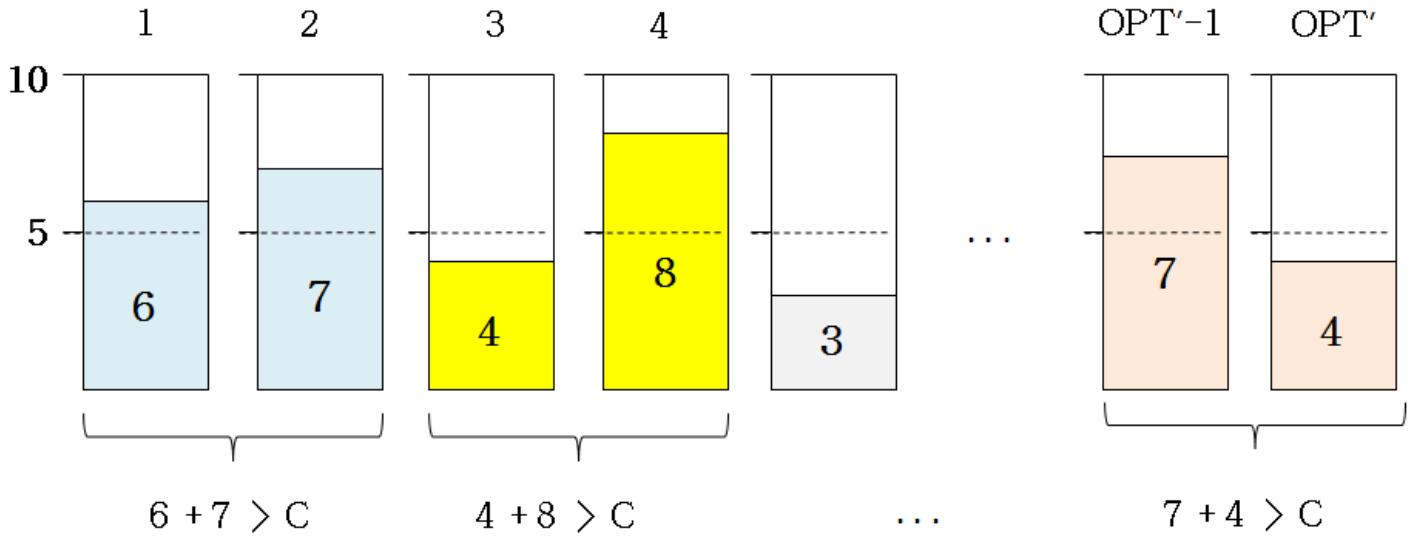
  ⇒ 2OPT + 1 > OPT′ 양변에 1 더하면

  ⇒ 2OPT ≥ OPT′

- 따라서 3가지 방법의 근사 비율은 2.0

다음 적합 근사 비율

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



(모든 물건의 크기의 합) > OPT′/2 x C

⇒ (모든 물건의 크기의 합)/C > OPT′/2 양변에 C 나누면

⇒ OPT > OPT′/2 OPT ≥ (모든 물건의 크기의 합)/C 이므로

⇒ 2OPT > OPT′ 양변에 2 곱하면

- 따라서 다음 적합의 근사 비율은 2.0

8.4 작업 스케줄링 문제

- n개의 작업의 수행 시간 t_i, i = 1, 2, 3, …, n, m개의 동일한 기계가 주어질 때 모든 작업이 가장 빨리 종료되도록 작업을 기계에 배정하는 문제

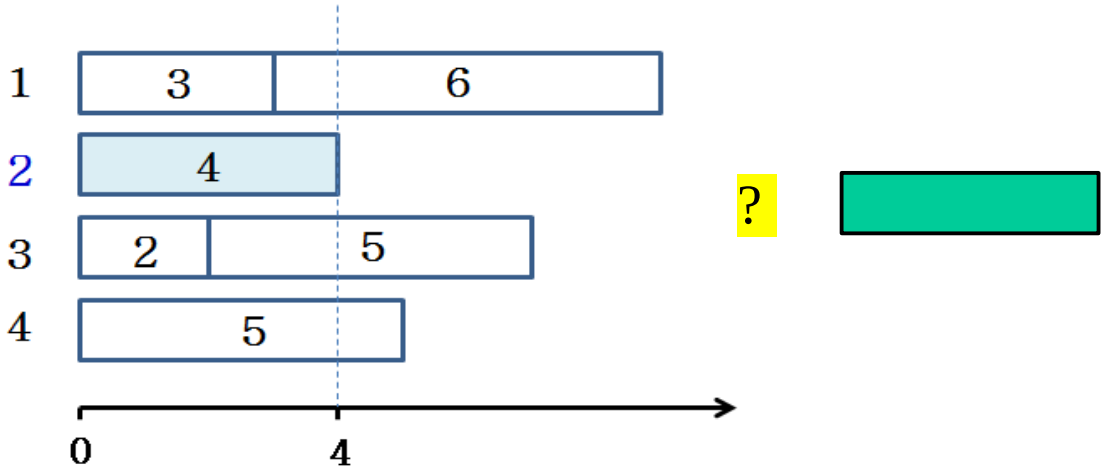
  - 단, 한 작업은 배정된 기계에서 연속적으로 수행되어야
  - 또한 기계는 한 번에 하나의 작업만을 수행한

어떻게 배정하여야 모든 작업이 가장 빨리 종료될까?

간단한 답은 그리디 방법으로 작업을 배정

- 현재까지 배정된 작업에 대해서 가장 빨리 끝나는 기계에 새 작업을 배정

➢ 예제에서는 2번째 기계가 가장 빨리 작업을 마치므로, 새 작업을 2번째 기계에 배정



알고리즘

Approx_JobScheduling

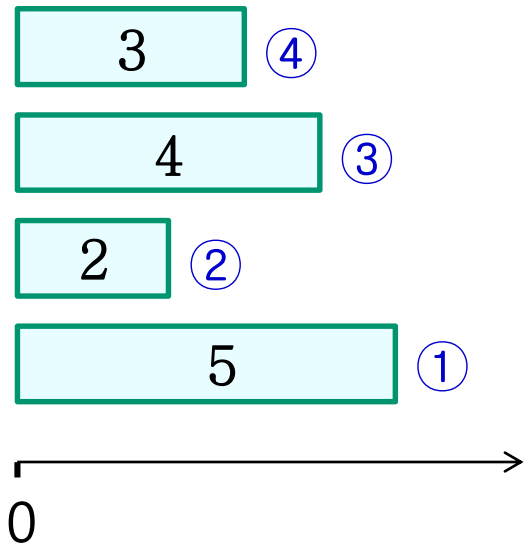
입력: n개의 작업, 각 작업 수행 시간 t_i, i = 1, 2, ⋯, n, M_j, j = 1,2, ⋯, m

출력: 모든 작업이 종료된 시간

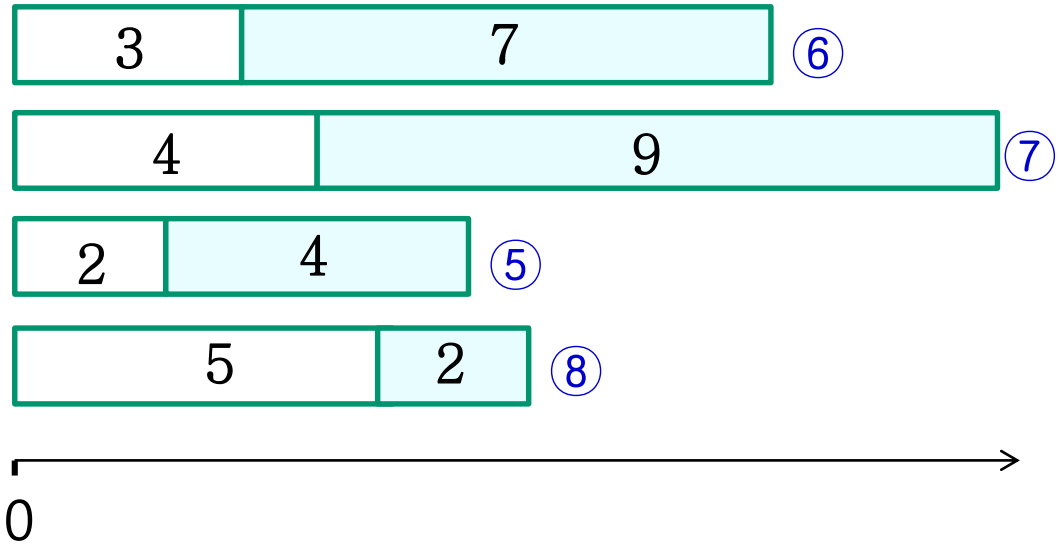
```
1.  for j = 1 to m
2.     L[j] = 0          // L[j]=기계 Mj에 배정된 마지막 작업의 종료 시간
3.  for i = 1 to n
4.      min = 1
5.      for j = 2 to m    // 가장 일찍 끝나는 기계 찾기
6.          if L[j] < L[min]
7.              min = j
8.      작업 i를 기계 Mmin에 배정한다.
9.      L[min] = L[min] + ti
10. return 가장 늦은 작업 종료 시간
```

Approx_JobScheduling 수행 과정

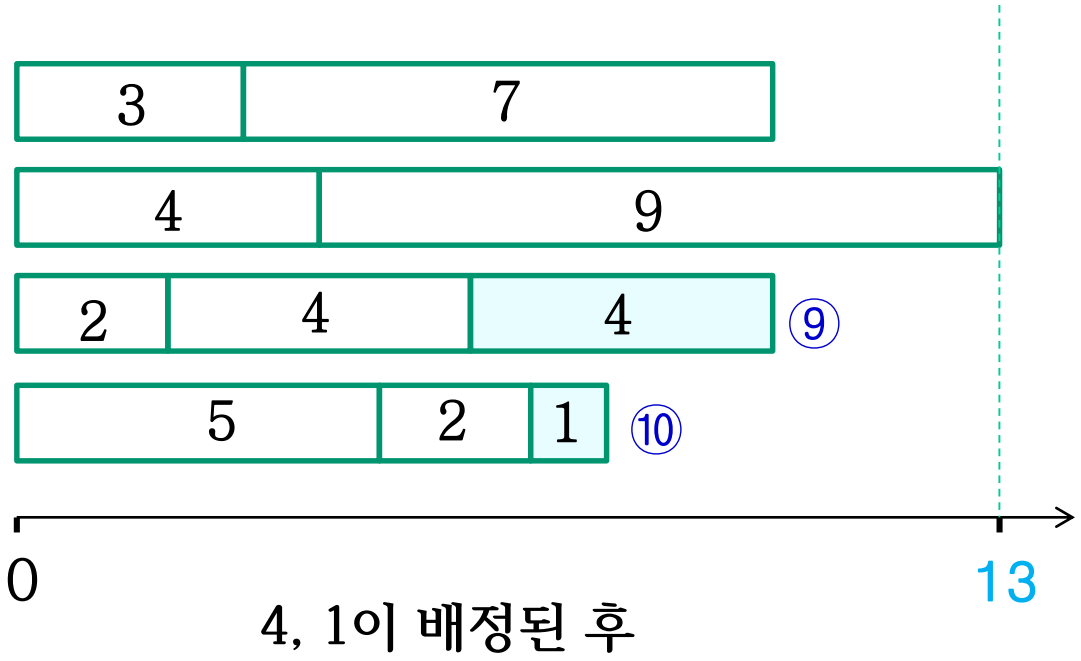
- 작업의 수행 시간이 [5, 2, 4, 3, 4, 7, 9, 2, 4, 1]
- 4개의 기계



5, 2, 4, 3이 배정된 후



4, 7, 9, 2이 배정된 후



- Approx_JobScheduling 알고리즘은 가장 늦게 끝나는 작업의 종료 시간인 13을 리턴

시간 복잡도

- n개의 작업을 하나씩 가장 빨리 끝나는 기계에 배정
- 이러한 기계를 찾기 위해 알고리즘의 line 5~7의 for-루프가 (m-1)번 수행
- 모든 기계의 마지막 작업 종료 시간인 L[j]를 살펴보아야 하므로 O(m) 시간

➢ 시간 복잡도

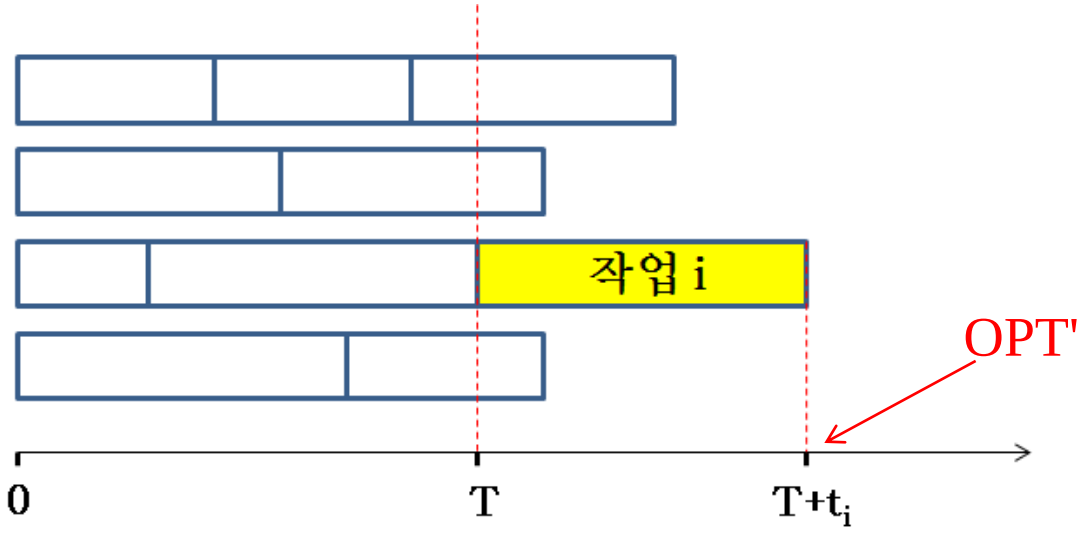
- n개의 작업을 배정해야 하고,
- line 10에서 배열 L을 탐색해야 하므로
- n x O(m) + O(m) = O(nm)

근사 비율

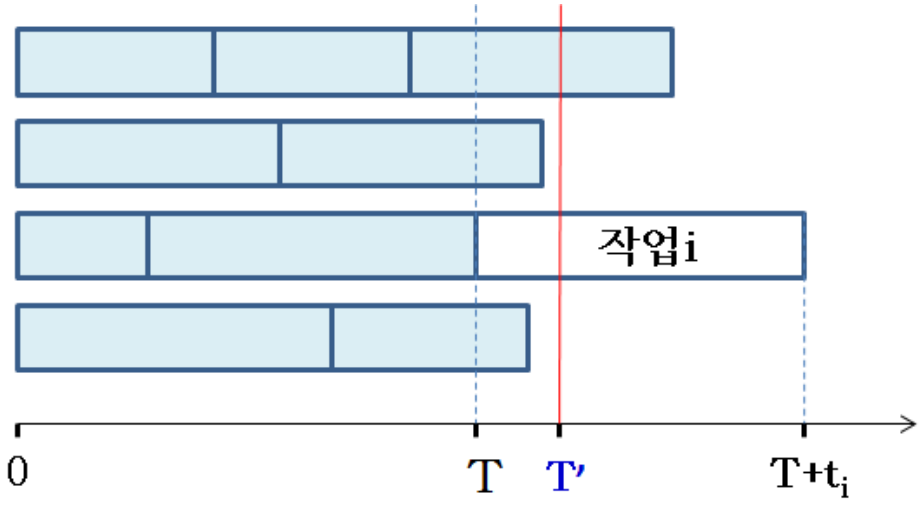
- Approx_JobScheduling 알고리즘의 근사해가 OPT′ 이고, 최적해가 OPT일 때,

$$OPT' \leq 2OPT$$

- 가장 마지막으로 배정된 작업 i가 T부터 수행되며,모든 작업이 $T+t_i$에 종료되므로 $OPT' = T + t_i$



근사 비율 계산



- T'는 작업 i를 제외한 모든 작업의 수행 시간의 합을 기계의 수 m으로 나눈 값
  - T'는 작업 i를 제외한 작업들의 평균 종료 시간
- 그러면 $T \leq T'$이다.
  - 작업 i가 배정된 (가장 늦게 끝나는) 기계를 제외한 모든 기계에 배정된 작업은 적어도 T 이후에 종료되기 때문

근사 비율

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

$$OPT' = t_i + T \le t_i + T' \quad \text{—①}$$

$$= t_i + \frac{\left(\sum_{j=1}^{n} t_j\right) - t_i}{m}, \quad \text{왜냐하면 } T' = \frac{\left(\sum_{j=1}^{n} t_j\right) - t_i}{m}$$

$$= t_i + \frac{\left(\sum_{j=1}^{n} t_j\right)}{m} - \frac{t_i}{m}$$

$$= \frac{1}{m}\sum_{j=1}^{n} t_j + \left(1 - \frac{1}{m}\right) t_i$$

$$\le OPT + \left(1 - \frac{1}{m}\right) OPT \quad \text{—②}$$

$$= \left(2 - \frac{1}{m}\right) OPT$$

$$\le 2\,OPT$$

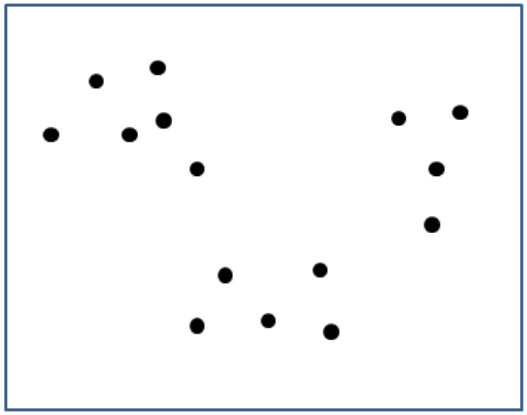
근사 비율

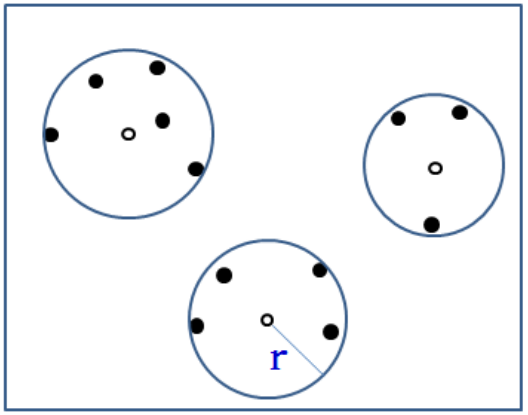
- 첫 번째 부등식 ①
  - 위의 그림에서 살펴본 $T \leq T'$ 을 이용한 것이다.

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

8.5 클러스터링 문제

- 2차원 평면의 n개의 점들 간의 거리를 고려하여 k개의 그룹으로 나누자.

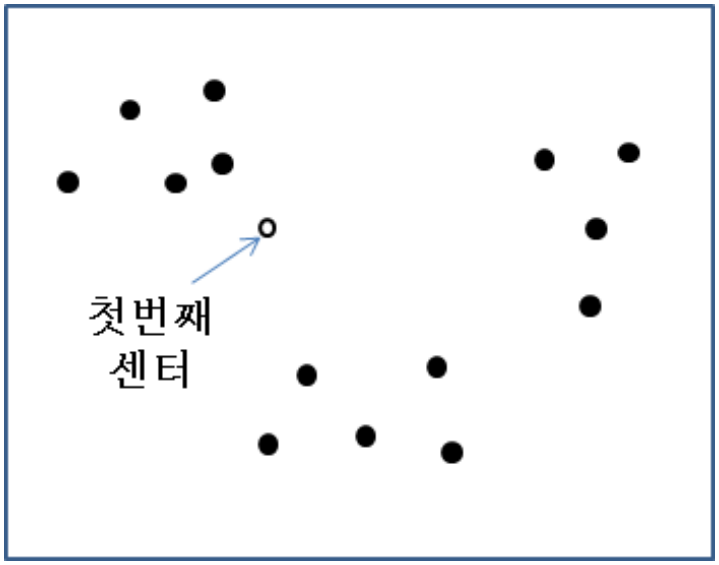




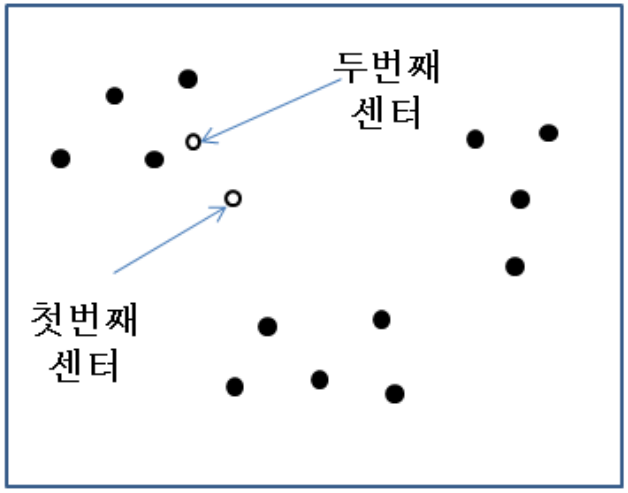
- 클러스터링 (Clustering) 문제
  - n개의 점을 k개의 그룹으로 나누고 각 그룹의 중심이 되는 k개의 점을 선택하는 문제
  - 단, 가장 큰 반경을 가진 그룹의 직경이 최소가 되도록 k개의 점을 선택해야

그리디 방법

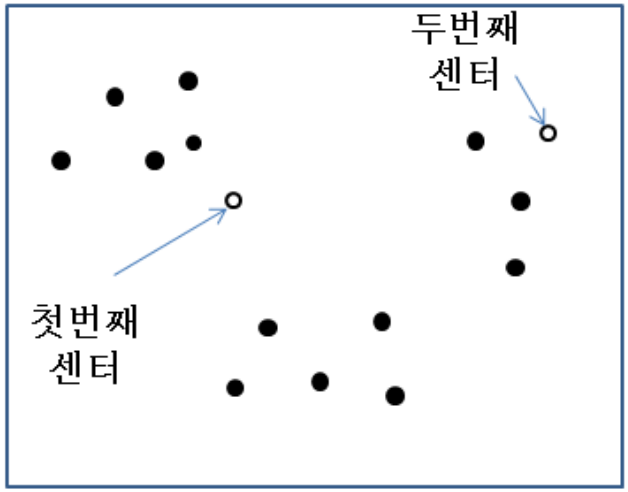
- k개의 센터 선택 방법
  - 1개씩 선택
  - 임의 점을 첫 번째 센터로 선택



두 번째 센터는 어느 점이 좋을까?



첫번째 센터에서 가장 가까운 점

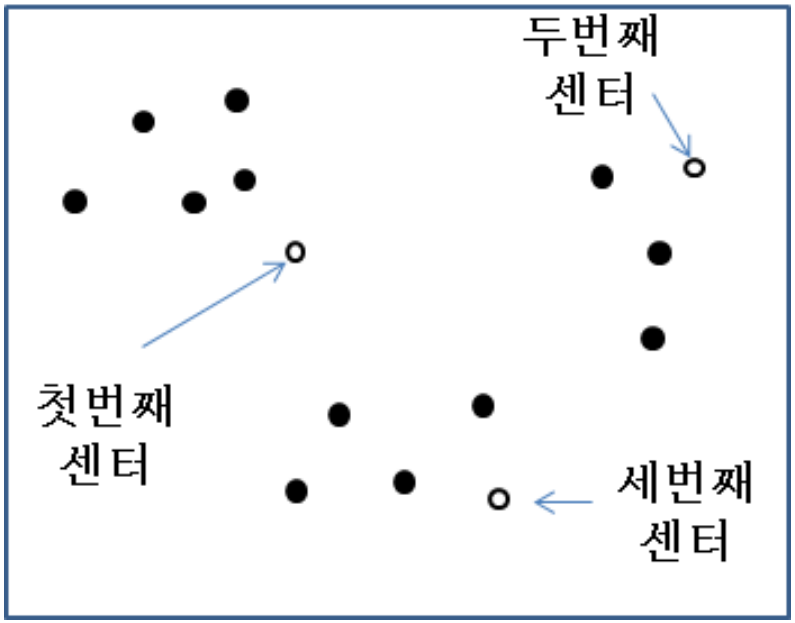


첫번째 센터에서 가장 먼 점

- 두 개의 센터가 서로 가까이 있는 것보다 멀리 떨어져 있는 것이 좋다.

세 번째 센터는?

- 첫 번째와 두 번째 센터 둘 다에서 가장 멀리 떨어진 점을 선택



알고리즘

Approx_k_Clusters

입력: n개의 점 x_i, i=0, 1, ⋯,n-1, 그룹의 수 k > 1

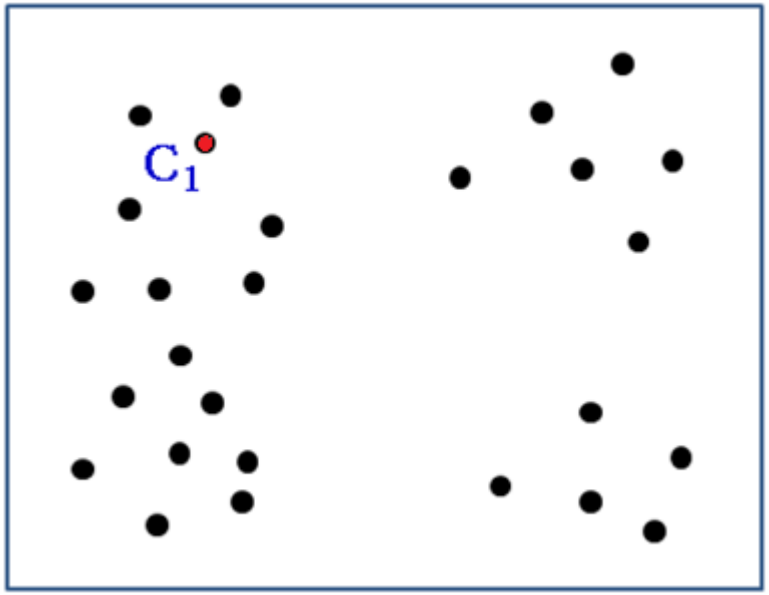
출력: k개의 점의 그룹 및 각 그룹의 센터

1. C[1] = r, 단, x_r은 랜덤하게 선택
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 D의 가장 큰 원소의 인덱스이고, x_i는 센터가 아니다.
7. 센터가 아닌 각 점 x_i로부터 앞서 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 C와 각 클러스터에 속한 점들의 리스트

첫 번째 센터

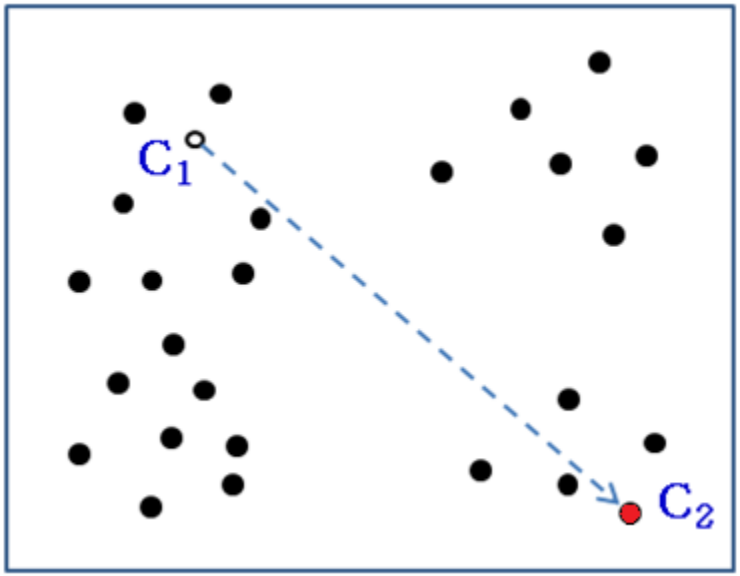
- 임의의 점 하나를 첫 번째 센터 C_1으로

k = 4



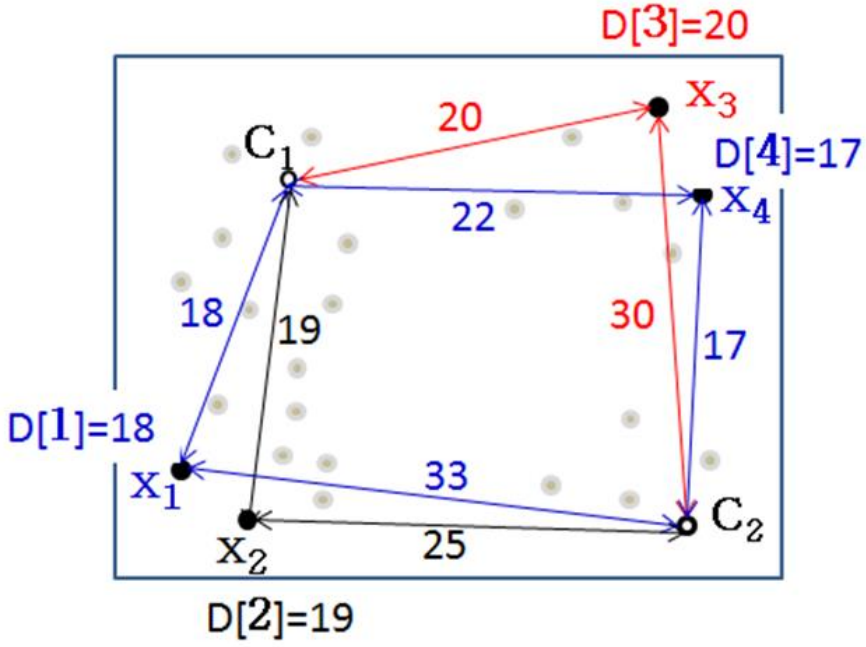
두 번째 센터

- C_1이 아닌 각 점 x_i에서 C_1까지의 거리 D[i] 계산
- C_1로부터 거리가 가장 먼 점을 다음 센터 C_2로

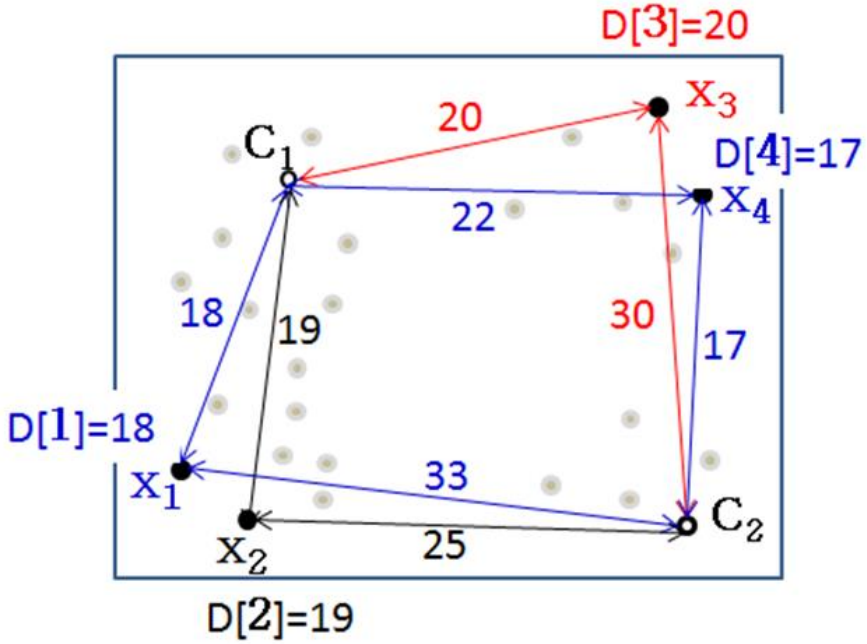


세 번째 센터

- C_1과 C_2를 제외한 각 점 x_i에서 각각 C_1과 C_2까지의 거리를 계산하여 그 중에서 작은 값을 D[i]로 정한다.
- D에서 가장 큰 값을 가진 원소의 인덱스가 i라고 하면, 점 x_i가 C_3이 된다.



D[i] 계산



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

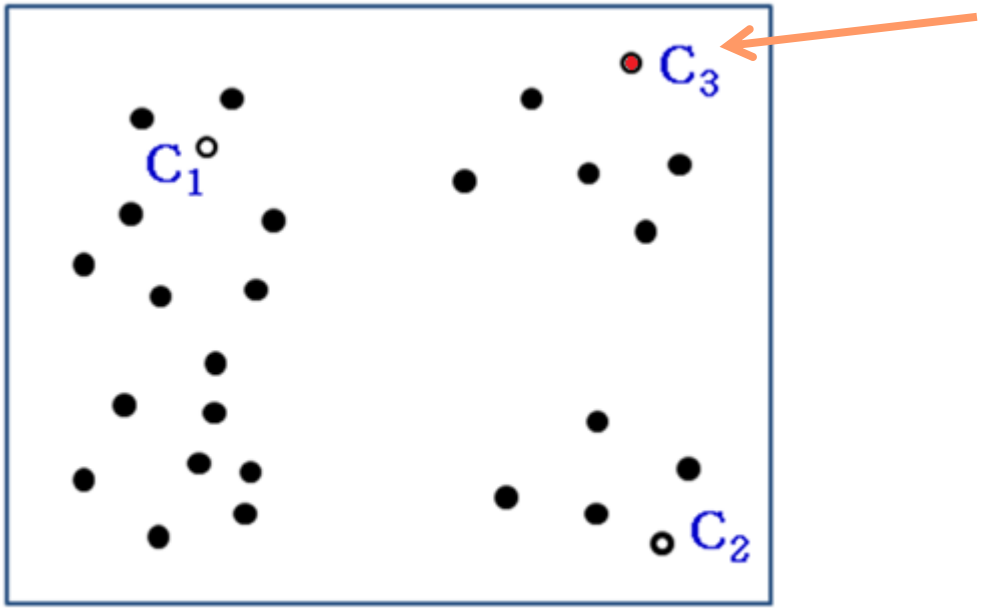
D[3]=20, min{dist(x_3,C_1), dist(x_3,C_2)}
=min{20, 30}이므로

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

다른 x_i의 D[i]는 20보다 작다고 가정

- dist(x, C)는 점 x와 센터 C 사이의 거리

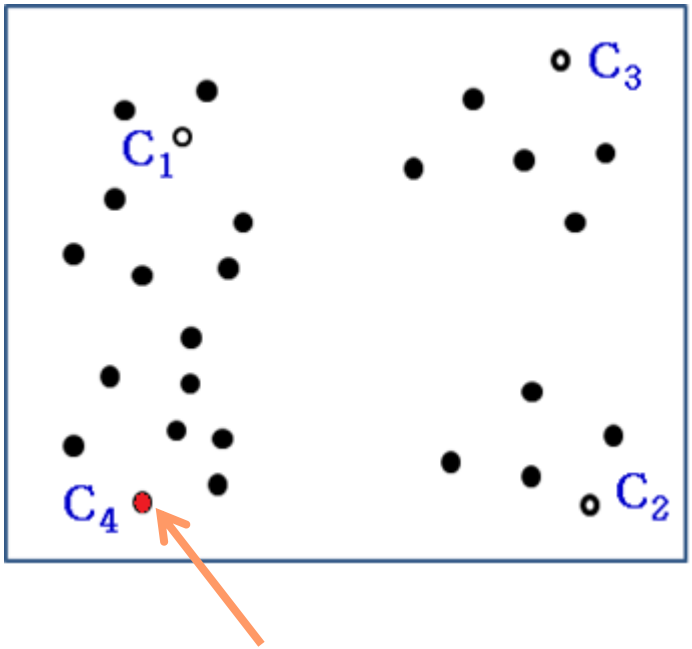
D[3]이 가장 큰 값이므로



C_1과 C_2로부터 가장 먼 점

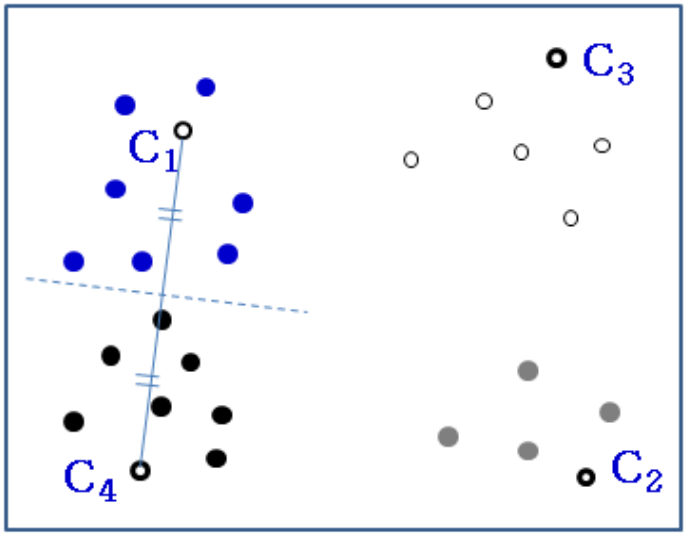
네 번째 센터

- 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.
- D에서 가장 큰 값을 가진 원소의 인덱스가 i이면, 점 x_i가 C_4



그룹으로 나누기

- 센터가 아닌 각 점 x_i로부터 위에서 찾은 4개의 센터까지 거리를 각각 계산하고 그 중에 가장 짧은 거리의 센터를 찾는다.
- x_i는 가장 가까운 센터의 그룹에 속하게 된다.

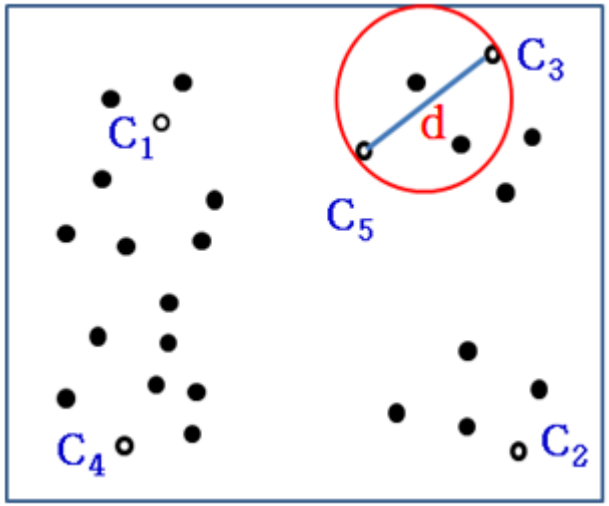


시간 복잡도

- 내부 for-루프: 각 점에서 각 센터까지의 거리를 계산하므로 O(kn) 시간
- Line 6 최대값을 찾으므로 O(n) 시간
- 외부 for-루프는 (k-1)회 반복하므로
  - O(1)+(k-1)x(O(kn)+O(n))
- Line 7 센터가 아닌 각 점으로부터 k개의 센터까지의 거리를 각각 계산하면서 최솟값을 찾으므로 O(kn) 시간
- O(1)+(k-1)x(O(kn)+O(n))+O(kn) = $O(k^2n)$

근사 비율

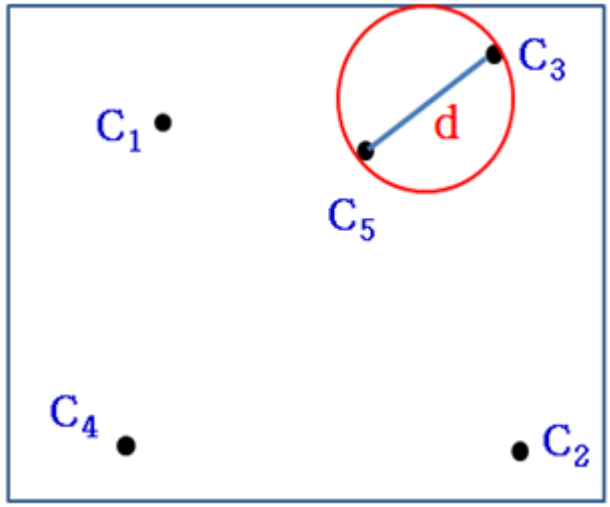
- 최적해가 만든 그룹 중에서 가장 큰 직경을 OPT라고 하자.
- OPT의 하한을 간접적으로 찾기 위해 알고리즘이 k개의 센터를 모두 찾고 나서 (k+1)번째 센터를 찾은 상황에서, 즉, k=4일 때,1개의 센터 C_5를 추가한 상황을 살펴보자.



OPT ≥ d

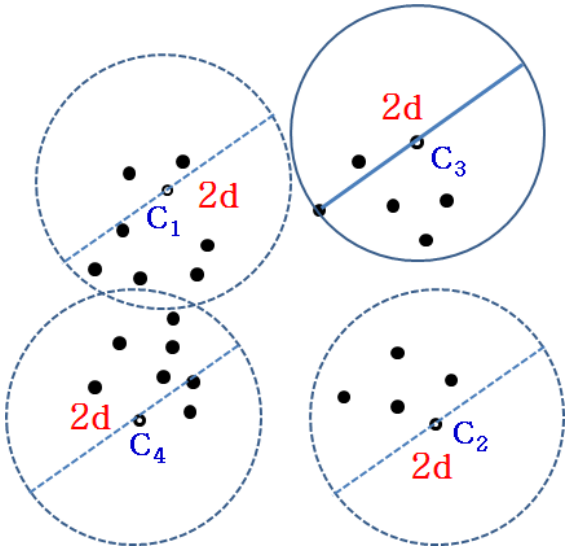
- C_5에서 가장 가까운 센터인 C_3까지의 거리를 d라고 하면
- 클러스터링 문제의 최적해를 계산하는 어떤 알고리즘이라도 위의 5개의 센터 점을 (k=4이니까) 4개의 그룹으로 분할해야 한다.
- 따라서 5개의 센터 중에서 2개는 하나의 그룹에 속해야만 한다.
- 그림에서는 C_3과 C_5가 하나의 그룹에 속한다.
- 최적해의 가장 큰 그룹의 직경인 OPT는 d보다 작을 수는 없다.

  즉, OPT ≥ d이다.



OPT′와 d의 관계

- OPT′ = 알고리즘이 계산한 근사해의 가장 큰 그룹의 직경
- (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 모든 점들이 위치한다. 따라서 OPT′ ≤ 2d



근사 비율

- 즉, OPT ≥ d이고, OPT′ ≤ 2d이므로,

  2OPT ≥ 2d ≥ OPT′

- Approx_k_Clusters 알고리즘의 근사 비율은 2.0

Approx_k_Clusters 알고리즘 실제 활용

- 첫 센터를 랜덤하게 선택하므로 보다 나은 클러스터링을 위해 알고리즘을 여러 차례 수행하여 얻은 결과 중에 best 클러스터링을 사용한다.
- 비정상적인 데이터(노이즈, outlier)에 취약한 성능을 보이므로 선처리를 통해 이들을 제거 후 사용해야 한다.

Applications

- 클러스터링 알고리즘은 대단히 많은 분야에서 활용된다. 이는 일반적으로 어떠한 데이터라도 유사한 특성 (유사도)을 가진 부분적인 데이터로 분할하여 분석할 때에 사용될 수 있기 때문
- 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)

Applications

- 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 알고리즘
- 여행자 문제를 위한 근사 알고리즘은 최소 신장 트리의 모든 점을 연결하는 특성과 최소 가중치의 특성을 이용한다. 근사비율은 2.0
- 정점 커버 문제를 위한 근사 알고리즘은 극대 매칭을 이용하여 근사해를 찾는다. 근사비율은 2.0

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 2.0
- 작업 스케줄링 문제는 가장 빨리 끝나는 기계에 새 작업을 배정하는 그리디 알고리즘으로 근사해를 찾는다. 근사비율은 2.0
- 클러스터링 문제는 현재까지 정해진 센터에서 가장 멀리 떨어진 점을 다음 센터로 정하는 그리디 알고리즘으로 근사해를 찾는다. 근사비율은 2.0