욕심쟁이 방법_2

2020년도 2학기 최 희석

목차

- 최단 경로
- 작업 스케줄링 문제
- 작업 선택 문제
- 허프만 코딩

최단 경로

최단 경로의 개념과 원리

- 가중 방향 그래프에서 두 정점을 연결하는 경로 중에서 간선의 가중치의 합이 가장 작은 경로

플로이드 Floyd 알고리즘	데이크스트라 Dijkstra 알고리즘
모든 정점 간의 최단 경로	특정한 하나의 정점에서 다른 모든 정점으로의 최단 경로(“단일 출발점 최단 경로”)
동적 프로그래밍 방법 적용	욕심쟁이 방법 적용
$O(\|V\|^3)$	$O(\|V\|^2)$
가중치의 합이 음수인 사이클이 없다고 가정	음의 가중치를 갖는 간선이 없다고 가정

최단 경로의 개념과 원리

- **데이크스트라(다익스트라) 알고리즘**
  - 거리 **d[v]**
    - 출발점에서 현재까지 선택된 정점 집합 S를 경유하여 정점 v에 이르는 최소 경로의 길이
  - 출발점에서 시작하여 거리 d[]가 최소인 정점을 차례로 선택하여 최단 경로를 구하는 방법
    - 초기화 → 출발점 **d [s]=0**, 나머지 모든 정점 v의 **d [v] = ∞**, 선택 정점 집합 **S={ }**
    - **미 선택 정점 집합 V-S에서 d[ ]가 가장 작은 정점 u를 선택**

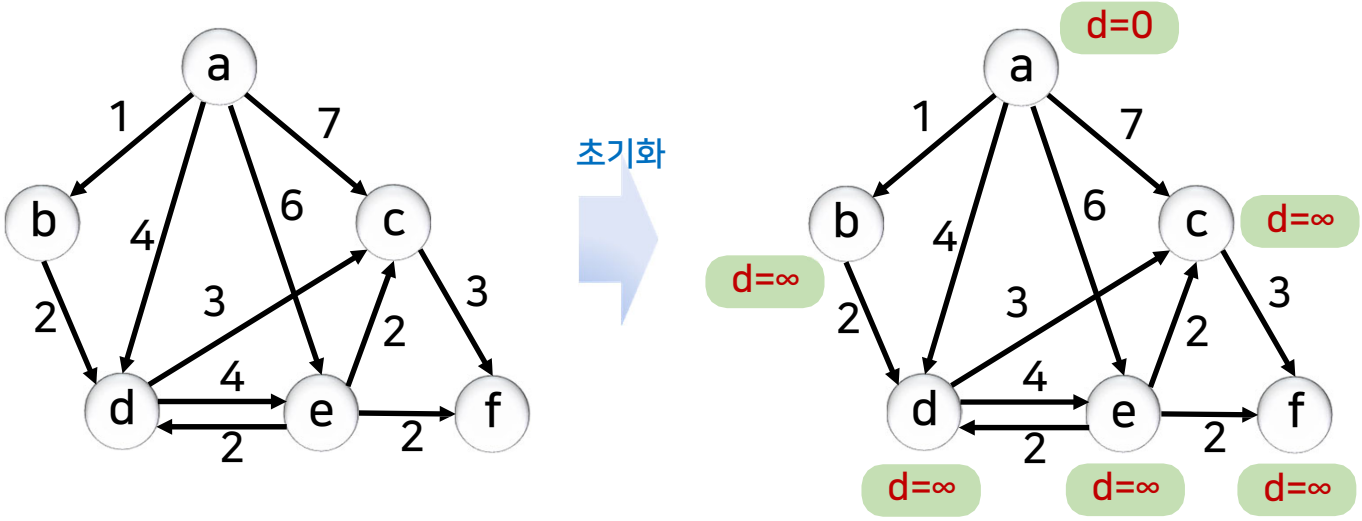
      → u 의 인접 정점에 대해서 u를 경유하는 거리와 기존 거리 중에서 작은 것을 새로운 거리 값으로 조정

최단 경로의 알고리즘

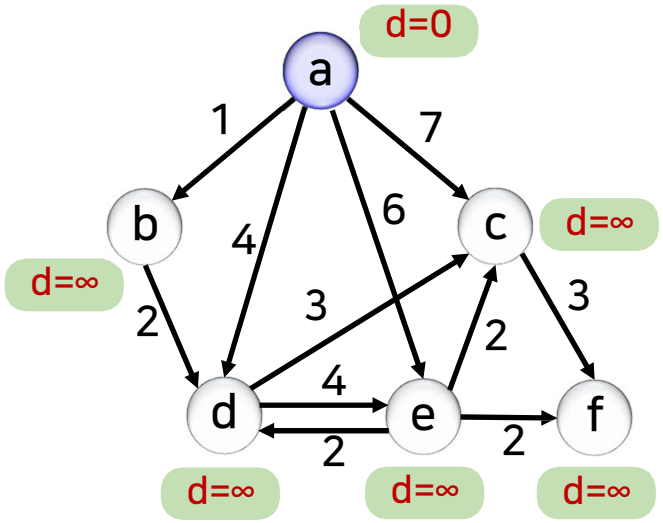
```
Dijkstra (G, s)
{
   S = Ø;   d[s]=0;
   for (모든 정점 v≠s) d[v] = ∞;
   for (모든 정점 v에 대해서) phi[v] = Null;
   while ( S != V ) {
      d[u]가 최소인 정점 u∈V-S를 선택
      S = S ∪ { u };
      for ( u에 인접한 모든 정점 v에 대해서 )
         if ( d[v] > d[u] + (u,v)의 가중치 ) {
            d[v] = d[u] + (u,v)의 가중치;
            phi[v] = u;
         }
   }
   return d[], phi[];
}
```

입력: G=(V,E), s : 시작 정점
출력: d[] : s로부터 각 정점으로의 최단 경로의 길이
phi[] : 최단 경로를 만드는 선행 정점

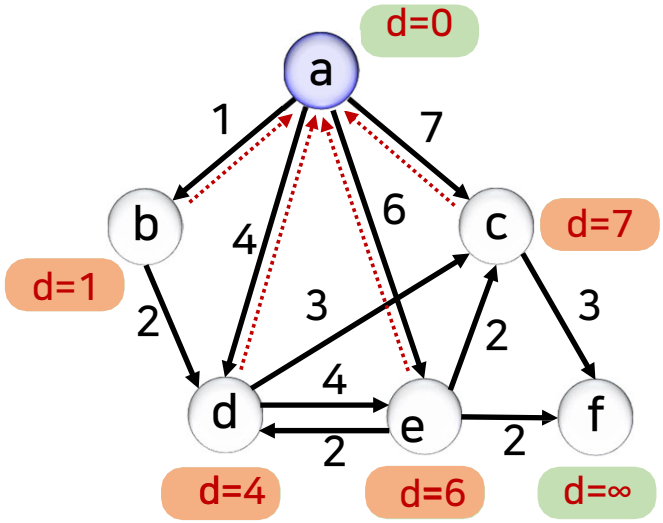
데이크스트라 알고리즘의 적용 예_1



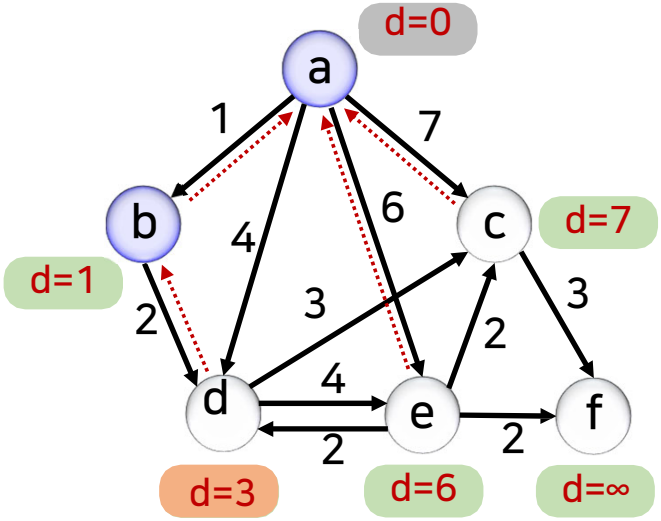
데이크스트라 알고리즘의 적용 예_1



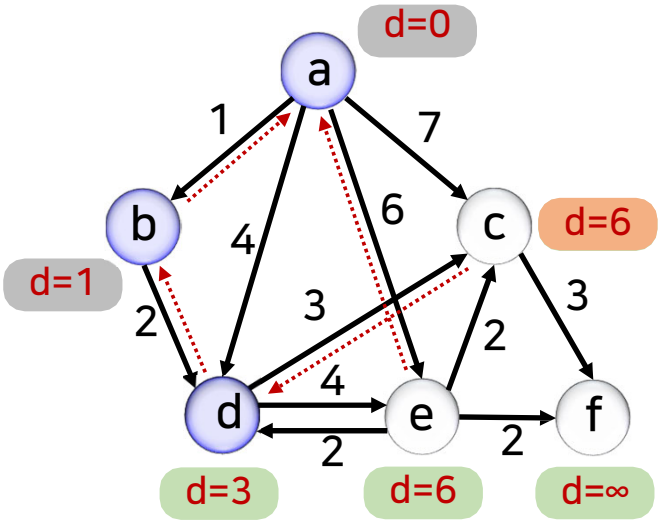
데이크스트라 알고리즘의 적용 예_1



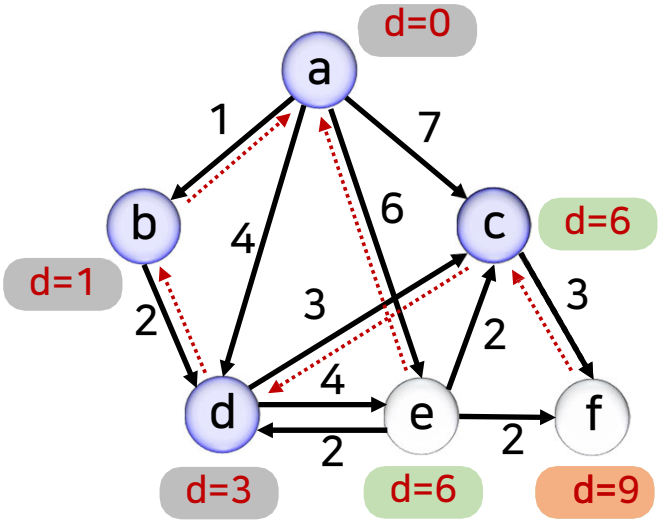
데이크스트라 알고리즘의 적용 예_1



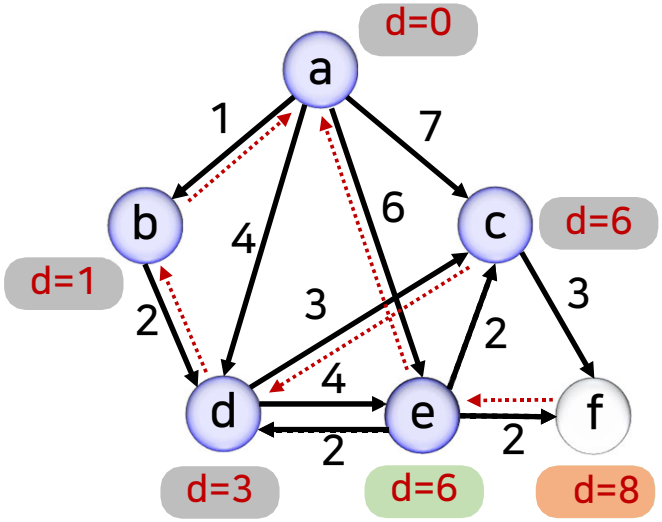
데이크스트라 알고리즘의 적용 예_1



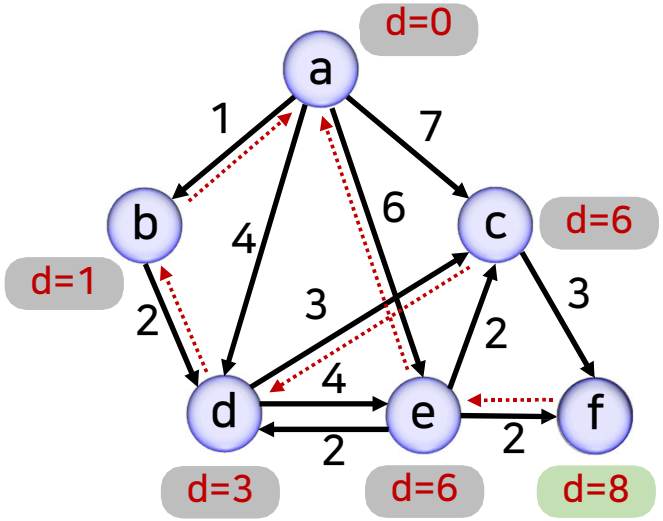
데이크스트라 알고리즘의 적용 예_1



데이크스트라 알고리즘의 적용 예_1

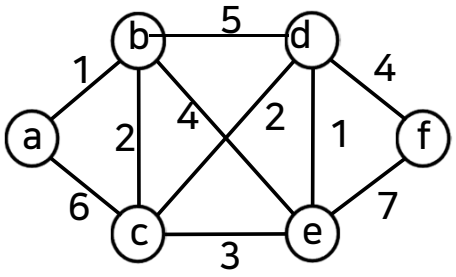


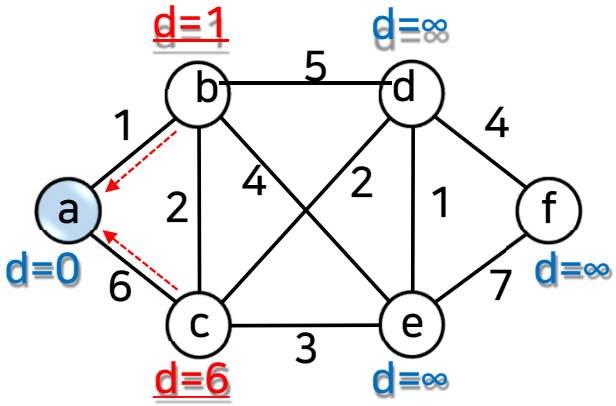
데이크스트라 알고리즘의 적용 예_1

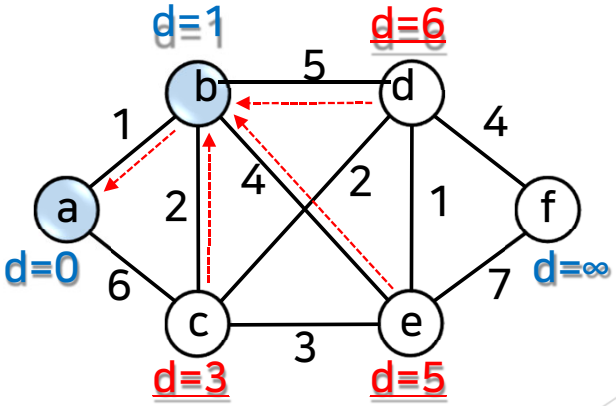


데이크스트라 알고리즘의 적용 예_2

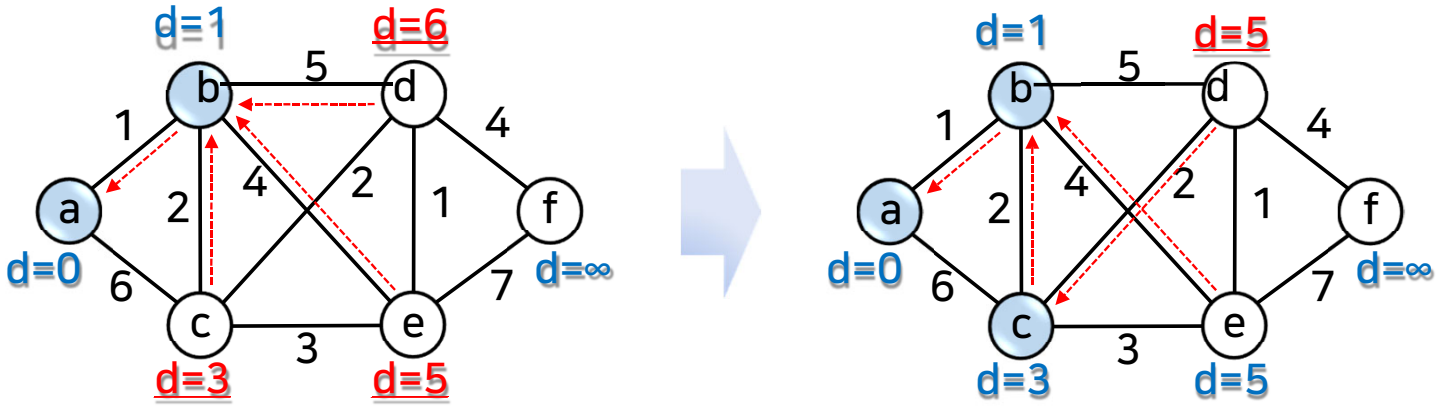
무방향 그래프



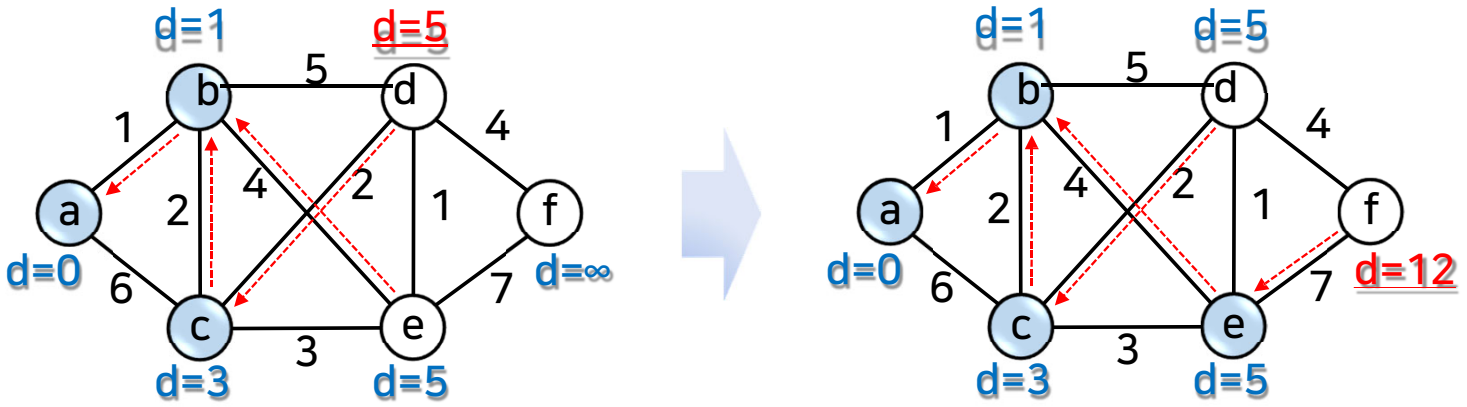




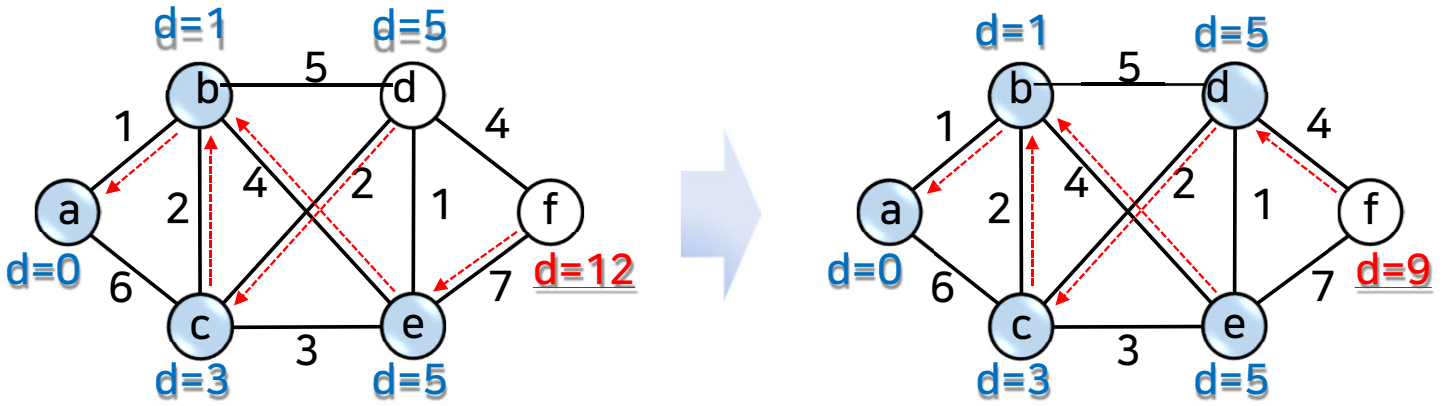
데이크스트라 알고리즘의 적용 예_2



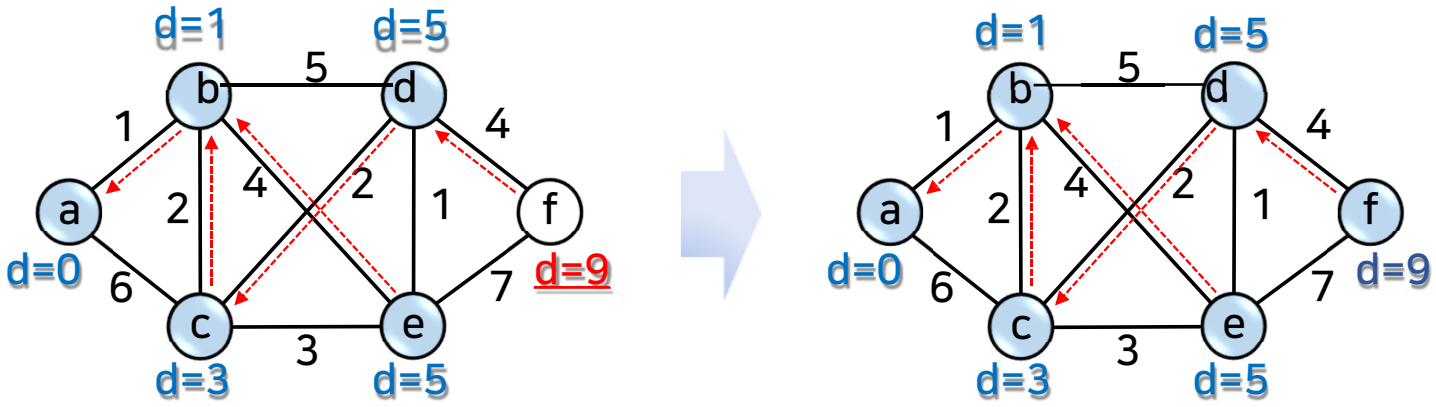
데이크스트라 알고리즘의 적용 예_2



데이크스트라 알고리즘의 적용 예_2



데이크스트라 알고리즘의 적용 예_2



데이크스트라 알고리즘의 적용 예_2

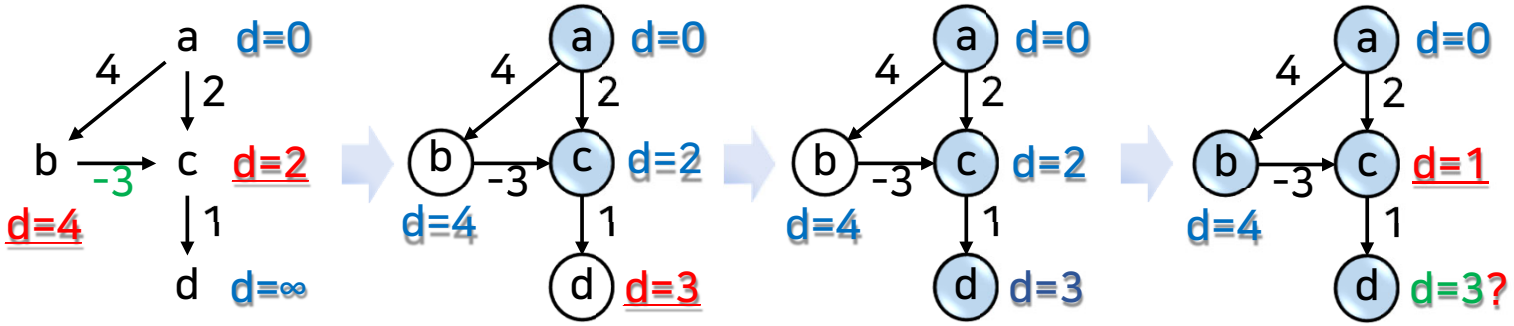
```
Dijkstra (G, s)
{
   S = Ø;   d[s]=0;
   for (모든 정점 v≠s) d[v] = ∞;
   for (모든 정점 v에 대해서) phi[v] = Null;
   while ( S != V ) {
      d[u]가 최소인 정점 u∈V-S를 선택
      S = S ∪ { u };
      for ( u에 인접한 모든 정점 v에 대해서 )
         if ( d[v] > d[u] + (u,v)의 가중치 ) {
            d[v] = d[u] + (u,v)의 가중치;
            phi[v] = u;
         }
   }
   return d[], phi[];
}
```

인접 행렬 → $O(|V|^2)$

인접 리스트+힙 → $O((|V|+|E|)\log|V|)$

데이크스트라 알고리즘의 특징

- 음의 가중치를 갖는 간선이 없어야 함



작업 스케줄링 문제

작업 스케줄링 문제의 개념과 원리

- 가장 적은 개수의 기계를 사용해서 작업 간의 충돌이 발생하지 않도록 모든 작업을 기계에 할당하는 문제
  - 작업의 집합 T = {t1, t2, …, tn}, ti = (si, fi) (1≤i≤n)
    - si : 작업 시작 시간, fi : 작업 완료 시간
  - 작업이 시작되면 중단됨 없이 해당 기계에서 완료되어야 함
    - 작업 간 충돌 → 한 기계에서 두 개 이상의 작업이 동시에 수행되는 것
    - 충돌을 피하기 위해 만족해야 할 조건 → **fi≤sj** 또는 **fj≤si**
- 기본 아이디어
- 각 단계에서 '**시작 시간이 빠른 작업**' 을 우선적으로 선택

  → 충돌이 발생하지 않으면 해당 기계에 배정

  → 충돌이 발생하면 새로운 기계에 할당

작업 스케줄링 문제의 알고리즘

```
TaskScheduling (T[][], n)
{
   작업을 시작 시간의 오름차순으로 정렬;
   m = 1;
   for (i=0; i<n; i++) {
      if ( 작업 t_i를 수행할 기계 p(1≤p≤m)가 있으면 )
         작업 t_i를 기계 p에 할당;
      else {
         m = m + 1;
         작업 t_i를 새 기계 m에 할당;
       }
   }
   return m;
}
```

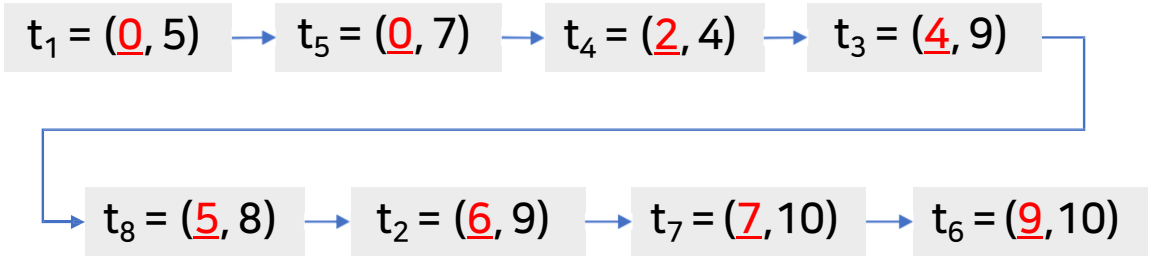
입력: T[0..n-1][0..1] :
T[i][0] : 작업 i의 시작 시간
T[i][1] : 작업 i의 완료 시간
n : 작업의 개수
출력: m : 모든 작업을 완료하는데 필요한 기계의 개수

작업 스케줄링 문제의 적용 예

$t_1 = (0, 5)$ $t_2 = (6, 9)$ $t_3 = (4, 9)$ $t_4 = (2, 4)$

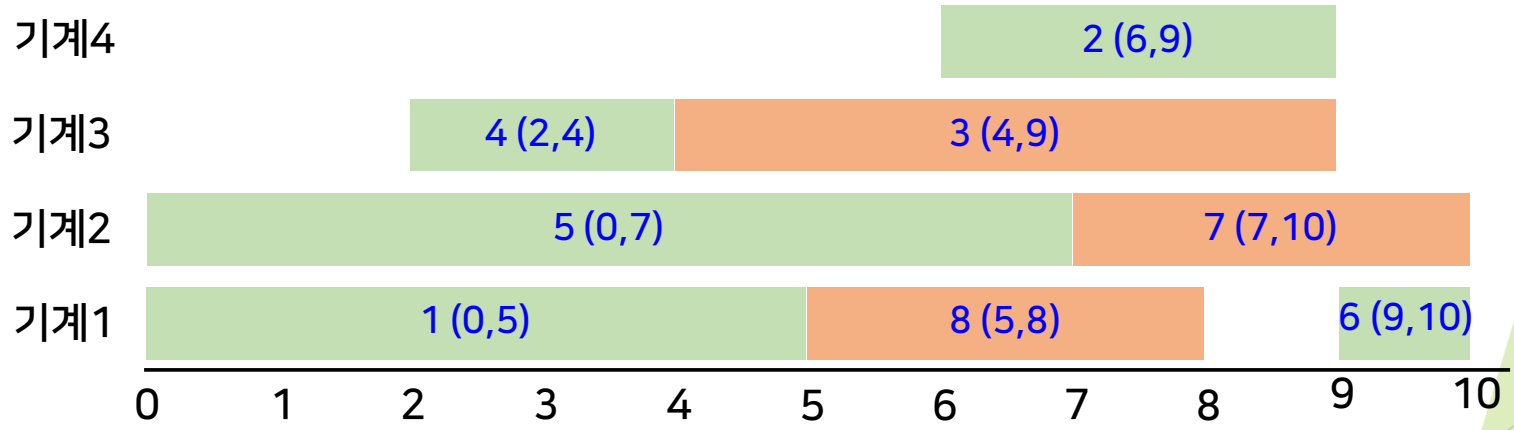
$t_5 = (0, 7)$ $t_6 = (9, 10)$ $t_7 = (7, 10)$ $t_8 = (5, 8)$

시작 시간의 오름차순으로 정렬 (시작 시간이 빠른 작업을 우선적으로 처리)



작업 스케줄링 문제의 적용 예

$t_1 = (0, 5)$ $t_5 = (0, 7)$ $t_4 = (2, 4)$ $t_3 = (4, 9)$ $t_8 = (5, 8)$ $t_2 = (6, 9)$ $t_7 = (7, 10)$ $t_6 = (9, 10)$



작업 스케줄링 문제의 성능 분석

```
TaskScheduling (T[][], n)
{
   작업을 시작 시간의 오름차순으로 정렬;
   m = 1;
   for (i=0; i<n; i++) {
      if ( 작업 t_i를 수행할 기계 p(1≤p≤m)가 있으면 )
         작업 t_i를 기계 p에 할당;
      else {
         m = m + 1;
         작업 t_i를 새 기계 m에 할당;
       }
   }
   return m;
}
```

O(nlogn)

O(n)

O(logn)

O(nlogn)

작업 선택 문제

작업 선택 문제의 개념과 원리

- 하나의 기계만을 사용해서 충돌 없이 최대 개수의 작업을 기계에 할당하는 문제
  - 작업의 집합 T = {t1, t2, …, tn}, ti = (si, fi) (1≤i≤n)
- 기본 아이디어
  - 각 단계에서 '**완료 시간이 빠른 작업**'을 우선적으로 선택

    → 충돌이 발생하지 않으면 기계에 배정

    → 충돌이 발생하면 해당 작업을 버림

작업 선택 문제의 알고리즘

```
ActivitySelection (T[][], n)
{
   작업을 완료 시간의 오름차순으로 정렬;
   m = 0;
   for (i=0; i<n; i++)
      if ( 작업 t_i가 충돌을 일으키지 않으면 ) {
         m = m + 1;
         작업 t_i를 기계에 할당;
       }
   return m;
}
```

입력: T[0..n-1][0..1] :
T[i][0] : 작업 i의 시작 시간
T[i][1] : 작업 i의 완료 시간
n : 작업의 개수
출력: m : 하나의 기계에서 처리할 수 있는 최대 작업 개수

작업 선택 문제의 적용 예

$t_1 = (0, 4)$ $t_2 = (4, 8)$ $t_3 = (9, 10)$ $t_4 = (1, 6)$

$t_5 = (6, 9)$ $t_6 = (1, 3)$ $t_7 = (3, 8)$ $t_8 = (4, 6)$

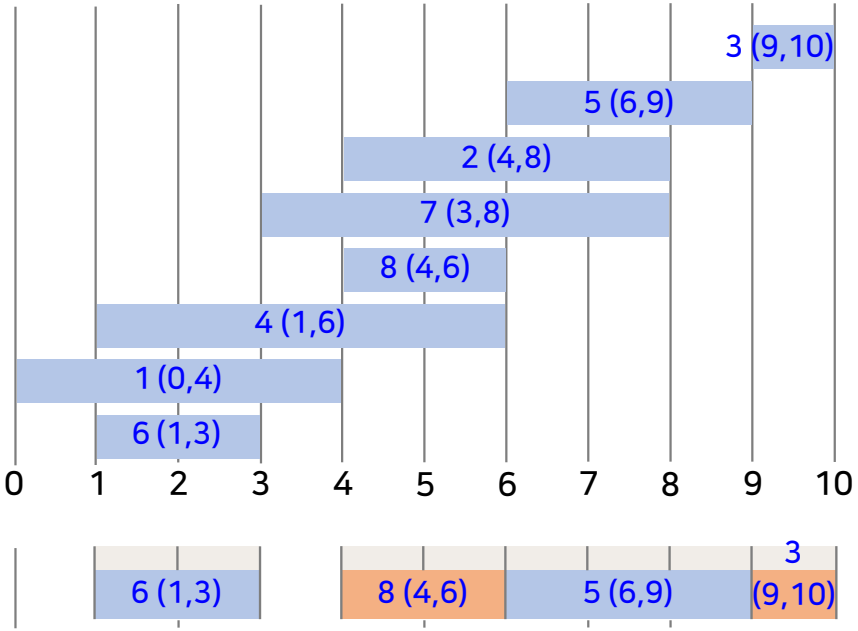
완료 시간의 오름차순으로 정렬

$t_6 = (1, 3)$ → $t_1 = (0, 4)$ → $t_4 = (1, 6)$ → $t_8 = (4, 6)$

$t_7 = (3, 8)$ → $t_2 = (4, 8)$ → $t_5 = (6, 9)$ → $t_3 = (9, 10)$

작업 선택 문제의 적용 예

$t_6 = (1, 3)$ $t_1 = (0, 4)$ $t_4 = (1, 6)$ $t_8 = (4, 6)$ $t_7 = (3, 8)$ $t_2 = (4, 8)$ $t_5 = (6, 9)$ $t_3 = (9, 10)$



작업 선택 문제의 성능 분석

```
ActivitySelection (T[][], n)
{
   작업을 완료 시간의 오름차순으로 정렬;
   m = 0;
   for (i=0; i<n; i++)
      if ( 작업 ti가 충돌을 일으키지 않으면 ) {
         m = m + 1;
         작업 ti를 기계에 할당;
      }
   return m;
}
```

O(nlogn)

O(n)

O(nlogn)

허프만 코딩

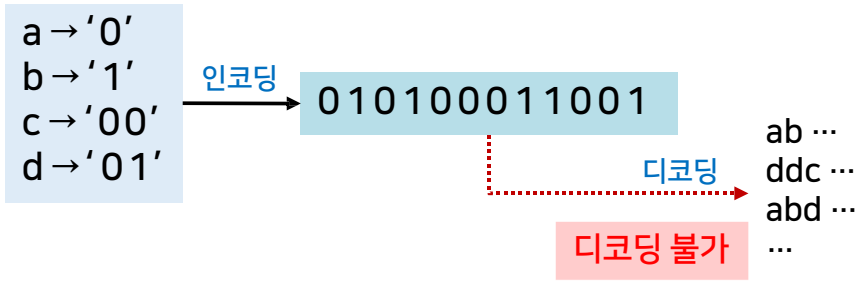
허프만 코딩의 개념과 원리

- 문자의 빈도 또는 확률 정보를 이용하는 통계적 압축 방법
  → 텍스트에서 각 문자가 출현하는 빈도수에 따라 다른 길이의 부호를 부여

- 출현 빈도수가 높은 문자 → 짧은 코드
- 출현 빈도수가 낮은 문자 → 긴 코드

허프만 코딩의 개념과 원리

- "ababcdbad"
  - 8비트 아스키 코드로 표현하는 경우 → **9문자 X 8비트 = 72비트**
  - 고정 길이 변환 코드로 표현하는 경우 → **9문자 X 2비트 = 18비트**
    - 문자 집합 = {a, b, c, d} → a: 00, b: 01, c: 10, d: 11
  - 빈도수에 따른 가변 길이 변환 코드로 표현하는 경우
    - 빈도수 → a: 3, b: 3, c: 1, d: 2



허프만 코딩의 개념과 원리

- 접두부 코드 prefix code
  - 각 문자에 부여된 이진 코드가 다른 문자에 부여된 이진 코드의 접두부가 되지 않는 코드
- 허프만 코드
  - 접두부 코드, 최적 코드
    - 최적 코드 → 인코딩된 메시지의 길이가 가장 짧은 코드
  - 인코딩 과정
    1. 주어진 텍스트에서 각 문자의 출현 빈도수를 계산
    2. 각 문자의 빈도수를 이용하여 허프만 트리를 생성하여 각 문자에 이진 코드를 부여
    3. 주어진 텍스트의 각 문자를 코드로 변환하여 압축된 텍스트를 생성

허프만 코딩의 개념과 원리

- 허프만 트리
  - 허프만 코딩에서 각 문자에 이진 코드를 부여하기 위해서 상향식으로 만드는 이진 트리
    - **욕심쟁이 방법** 적용
    - 리프 노드가 각 문자를 표시하는 전 이진트리
  - 각 문자가 개별적인 트리인 상태에서 시작해서 빈도수가 작은 두 트리를 합쳐서 보다 큰 트리를 생성하는 과정을 반복
    - 각 노드는 빈도수로 표시
    - 좌우의 두 간선은 각각 0과 1로 레이블됨
    - 합쳐지는 두 트리를 자식 노드로 갖는 부모 노드를 생성

    → 부모 노드의 빈도수는 두 자식 노드의 빈도수의 합

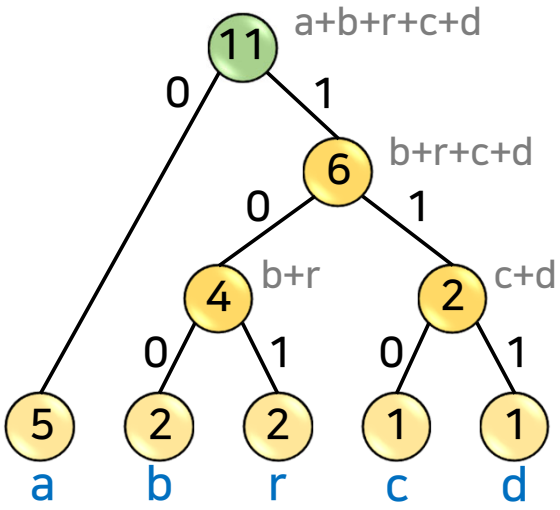
허프만 코딩의 알고리즘

```
HuffmanTree (F[], n)
{
   빈도수 F[]에 따라 최소 힙 Q 생성;
   for (i=0; i<n; i++) {
      u ← 힙 Q에서 최소값 삭제;
      v ← 힙 Q에서 최소값 삭제;
      새 노드 x를 생성하여 u와 v를 x의 두 자식 노드로 지정;
      노드 x의 빈도수 ← u의 빈도수 + v의 빈도수;
      노드 x를 힙 Q에 삽입;
   }
   return Q;
}
```

입력: F[1..n] : 문자 c_i의 빈도수
n : 문자 집합의 크기 ($c_1, c_2, \cdots, c_n$)
출력: Q : 허프만 트리

허프만 코딩의 적용 예

- "abracadabra" ➡ 01001010110011101001010

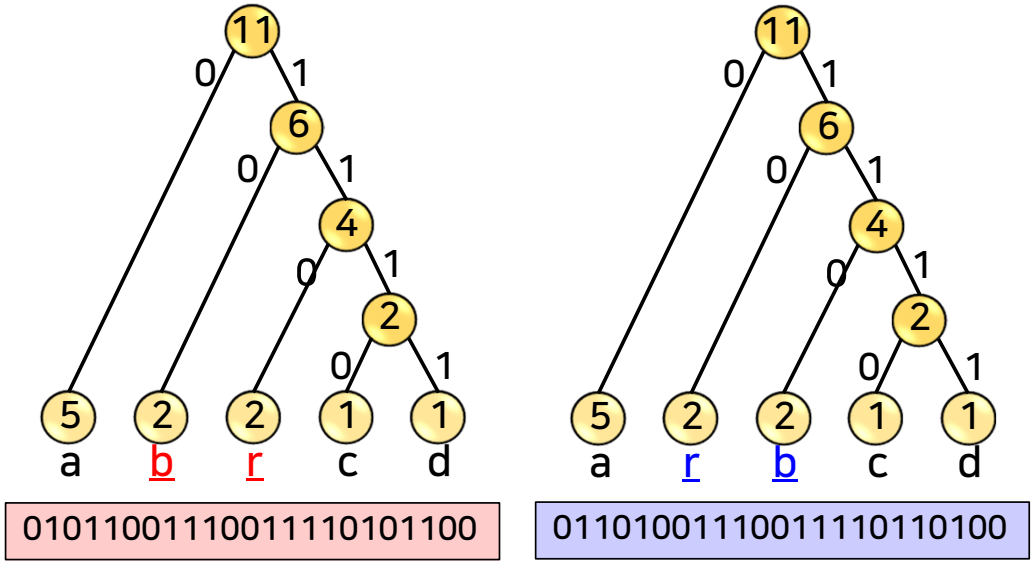


문자	코드
a	0
b	100
c	110
d	111
r	101

허프만 코딩의 적용 예

"abracadabra" → 01001010110011101001010

허프만 트리는 유일하지 않음

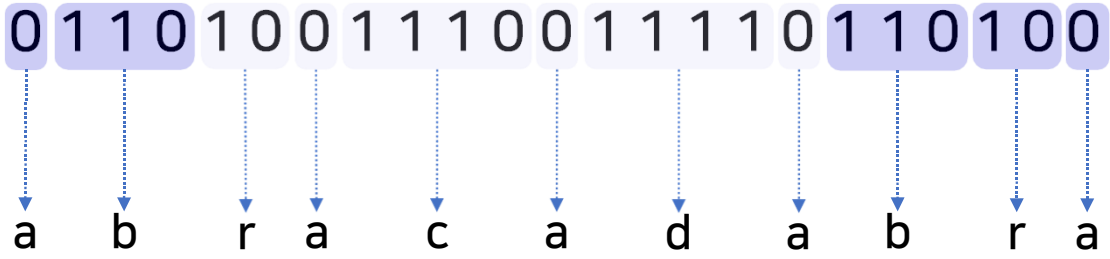


문자	코드		
a	0	0	0
b	100	10	110
c	110	1110	1110
d	111	1111	1111
r	101	110	10

비트 스트링의 디코딩 과정

압축된 스트링을 처음부터 차례대로 읽으면서 주어진 접두부 코드와 일치하는 코드가 나오면 해당 문자로 변환

문자	코드
a	0
b	1 1 0
c	1 1 1 0
d	1 1 1 1
r	1 0



허프만 코딩의 성능 분석

```
HuffmanTree (F[], n)
{
   빈도수 F[]에 따라 최소 힙 Q 생성;
   for (i=0; i<n; i++) {
      u ← 힙 Q에서 최솟값 삭제;
      v ← 힙 Q에서 최솟값 삭제;
      새 노드 x를 생성하여 u와 v를 x의 두 자식 노드로 지정;
      노드 x의 빈도수 ← u의 빈도수 + v의 빈도수;
      노드 x를 힙 Q에 삽입;
   }
   return Q;
}
```

O(n)

O(n)

O(logn)

길이 m인 텍스트의 실제 인코딩 시간

O(nlogn) + O(m) → O(nlogn+m)

문자 집합의 크기

허프만 코딩

- **각 문자의 빈도수를 모르는 경우 주어진 텍스트를 두 번 읽음**
  - ① 각 문자의 빈도수를 계산할 때, ② 텍스트를 읽으면서 실제 인코딩할 때
    - 이 경우 속도가 상당히 느려져 실용성이 없음

- **압축된 데이터를 디코딩하려면**
  - 각 문자의 빈도수, 허프만 트리에 대한 정보, 문자 집합 정보가 필요
    - 압축된 데이터의 헤더로서 필요한 정보 제공 → 실제 압축률 저하

과제 안내

8주차 과제

- 데이크스트라 알고리즘, 작업 스케줄링 문제 구현하기
  - e-Class 업로드
- 양식 (한글, 워드,PDF -> 자유)
- 파일명 (이름_학번_전공)
  - 예) 최희석_2014182009_게임공학

- 질의 응답은 e-Class 질의응답 게시판에 남겨 주시길 바랍니다.