3일차 알고리즘 보고서

로빛 20기 인턴 최세영

목차

1. 알고리즘 개요

어떤 작업을 수행하기 위해 입력을 받아 원하는 출력을 만들어 내는 계산 절차

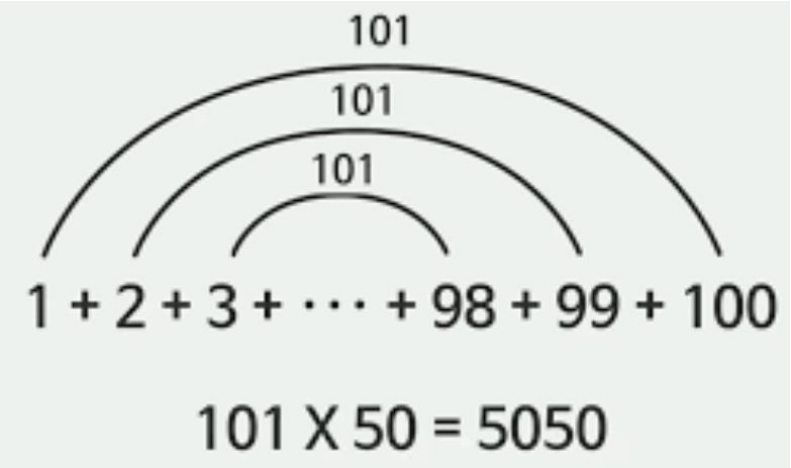
설계 시 중요한 점

- 입력과 출력이 무엇인지 명확하게 정의해야 함

- 절차는 구체적이고 모호하지 않게 설계해야 함

- 효율성을 고려한 최적화 필요

최적화 예시: 가우스 계산법



- 단순 덧셈을 반복하는 방식 vs. 가우스 공식 (1 + n) × n ÷ 2

- 작은 수에서는 차이 없으나 수가 커질수록 차이가 커짐

알고리즘 활용 분야

- 정렬 알고리즘: 선택, 버블, 삽입, 병합 등

- 탐색 알고리즘: 행렬, 배열, 트리, 그래프 등

2. 선택 정렬 (Selection Sort)

동작 방식

- 1 차원 데이터에서 가장 작은 값을 찾아 맨 앞(또는 맨 뒤)로 이동

- 첫 요소부터 마지막 요소까지 확인하면서 최솟값의 위치를 기억

- 최솟값을 앞으로 교환 → 남은 구간에 대해 반복

특징

- 장점: 구조 단순, 메모리 사용 적음, 버블 정렬보다 효율적

- 단점: 매번 모든 데이터를 확인해야 하므로 데이터가 많아질수록 비효율적

개선 알고리즘

- 이중 선택 정렬(Double Selection Sort)

- 한 번의 탐색에서 최소값과 최대값을 동시에 찾아 각각 맨 앞·맨 뒤에 배치

패스	테이블	최솟값
0	[9,1,6,8,4,3,2,0]	0
1	[0,**1,6,8,4,3,2,9**]	1
2	[0,1,**6,8,4,3,2,9**]	2
3	[0,1,2,**8,4,3,6,9**]	3
4	[0,1,2,3,**4,8,6,9**]	4
5	[0,1,2,3,4,**8,6,9**]	6
6	[0,1,2,3,4,6,**8,9**]	8

3. 버블 정렬 (Bubble Sort)

동작 방식

- 인접한 두 요소를 비교 → 앞 값이 크면 교환

- 큰 값이 뒤로 이동하며, 반복할수록 마지막 요소부터 정렬 확정

특징

- 장점: 구조 단순, 이해·구현 용이, 데이터가 이미 정렬된 경우 조기 종료 가능

- 단점: 모든 요소를 반복적으로 비교 → **시간 복잡도 $O(n^2)$**

- 교환 연산이 많아 실제 대규모 데이터 처리에는 부적합

응용 알고리즘

- 칵테일 정렬(Cocktail Shaker Sort)

- 정렬을 앞→뒤, 뒤→앞으로 번갈아 수행(왕복 형태)

- 전체 효율은 버블 정렬과 동일

$[7, 2, 0, 1, 5, 6, 4]$

$[2, 0, 1, 5, 6, 4, 7]$ 1회

$[0, 1, 2, 5, 4, 6, 7]$ 2회

$[0, 1, 2, 4, 5, 6, 7]$ 3회

$[0, 1, 2, 4, 5, 6, 7]$ 4회

4. 삽입 정렬 (Insertion Sort)

동작 방식

- 선택 정렬처럼 앞에서부터 하나씩 정렬해 나감

- 하지만 최솟값을 찾는 대신, 정렬된 구간에 새로운 데이터를 알맞은 위치에 삽입

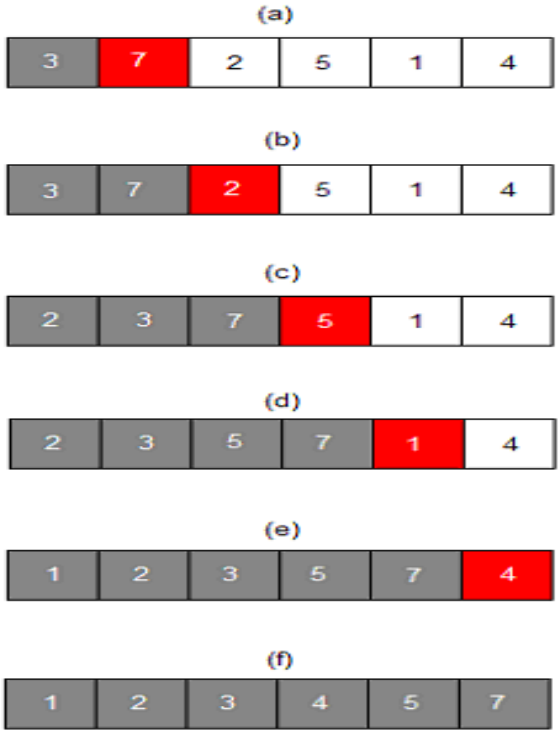
- 첫 번째 데이터는 이미 정렬된 상태로 간주 → 이후 데이터들을 비교하여 적절한 위치에 끼워 넣음

특징

- 장점: 구현 직관적, 데이터가 거의 정렬된 상태라면 매우 빠름

- 단점: 무작위 데이터에서는 비교·이동이 많아져 비효율적

- 데이터 크기와 상태에 따라 성능 차이가 큼



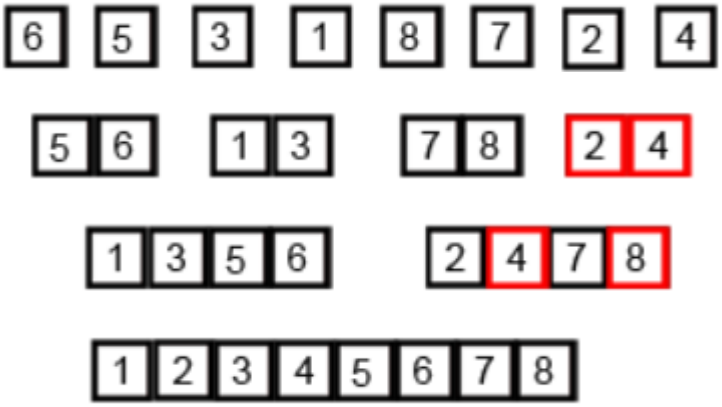
5. 병합 정렬 (Merge Sort)

동작 방식

- 분할 정복(divide and conquer) 기법 사용
- 데이터를 더 이상 쪼갤 수 없을 때까지 반으로 나눔
- 이후 두 개씩 병합하면서 정렬

특징

- 시간 복잡도: 항상 O(n log n) (데이터 크기·상태와 무관)
- 안정 정렬(stable sort) → 동일한 값의 기존 순서 유지
- 대량 데이터 처리에 효율적이고 예측 가능한 성능 보장
- 단점: 병합 과정에서 추가 메모리 공간 필요



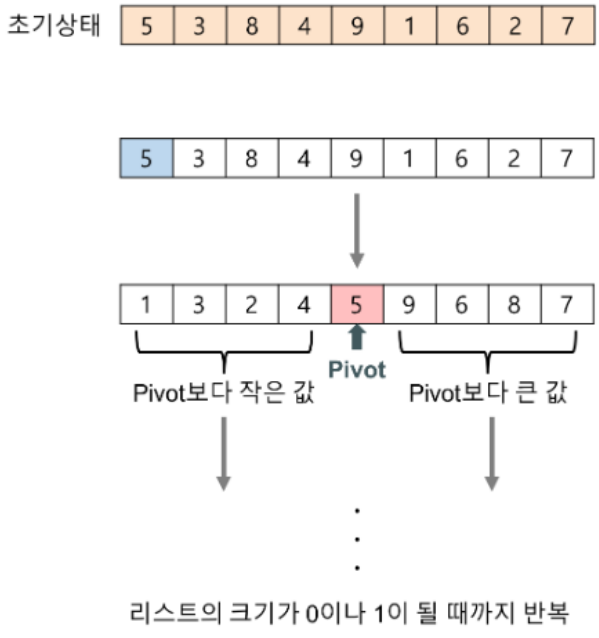
6. 퀵 정렬 (Quick Sort)

동작 방식

- 배열에서 임의의 원소를 중심으로 정함
- 해당 중심을 기준으로 작은 값과 큰 값을 좌우로 배치
- 좌우에서 다시 중심을 각각 정함
- 위 알고리즘을 부분 배열 크기가 1 이 될 때까지 반복

특징

- 평균적으로 빨리 정렬 가능
- 분할 정복(부분화 정리)으로 구현 쉬움
- 임의로 선택된 중심점에 따라 성능이 달라짐
- 메모리 사용률 적음



7. 트리 (Tree)

부모 노드 밑에 여러 자식 노드가 연결, 자식 노드 밑에 또 자식 노드가 연결돼있는 재귀적 형태의 그래프 자료구조

구조

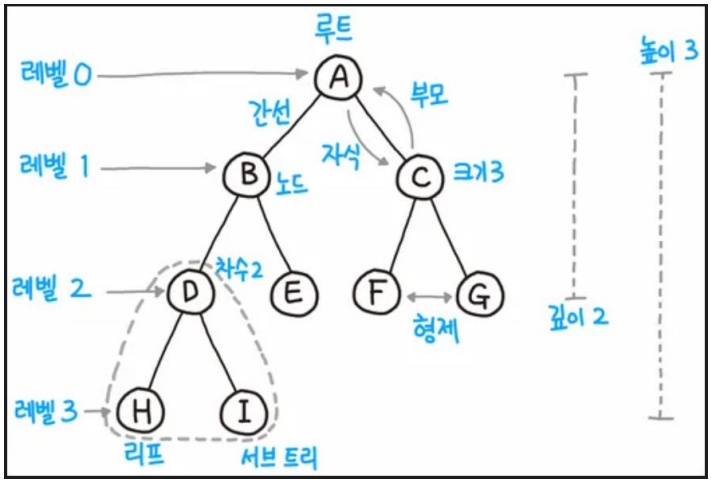
- 루트: 부모가 없는 최상위 노드

- 부모: 해당 노드 위에 있는 노드

- 자식: 해당 노드 아래에 있는 노드

- 깊이: 루트에서 해당 노드로 가는 최단 경로 길이

- 높이: 해당 노드의 최대 깊이 값



종류

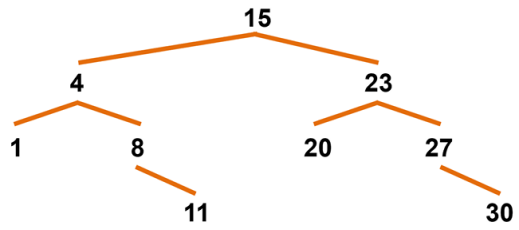
- 이진 탐색 트리

- AVL 트리 등

8. 이진 탐색 트리 (BST)

- 데이터를 입력할 때부터 정렬

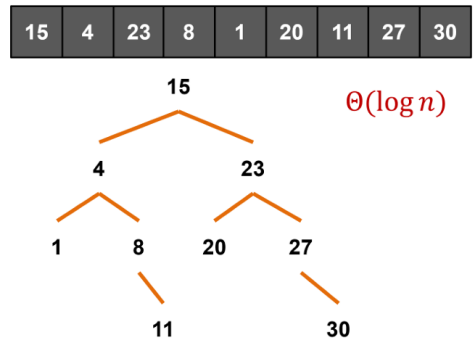
- 모든 노드를 왼쪽 < 루트 < 오른쪽으로 나눔



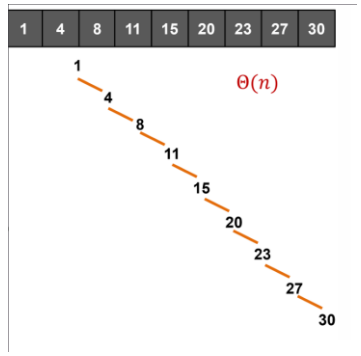
특징

- 데이터 입력 순서에 따라 성능이 달라짐

- 최선의 트리:



- 최악의 트리(편향 트리):

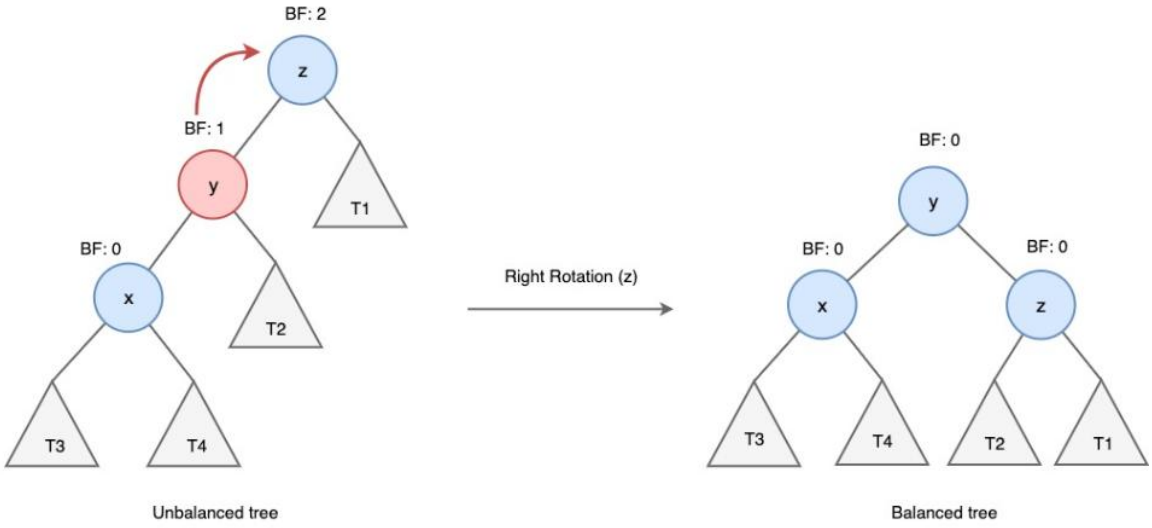


9. AVL 트리

- 자가 균형 이진 트리

- 좌우 트리의 높이 차이가 1 이하

- 좌우 트리의 높이 차이가 2 이상이면 회전(Rotation)으로 최적화



특징

- 항상 최선의 트리가 나옴 (복잡도 항상 O(logN))

- 탐색 자체는 빠르지만 균형을 엄격하게 잡으므로 삽입과 제거가 느림

회전

-불균형인 노드의 자식의 위치를 반대편 트리에 삽입하면서 균형을 맞춤

-LL, LR, RL, RR 로 4 가지의 경우의 수가 존재함

10. 그래프 (Graph)

- 노드(정점)와 간선, 가중치의 집합

- 한 노드에서 시작하여 차례대로 모든 노드를 탐색하는 것

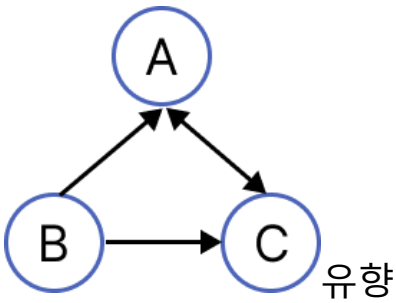
구조

- 노드: 그래프에서 하나의 점, 데이터 저장

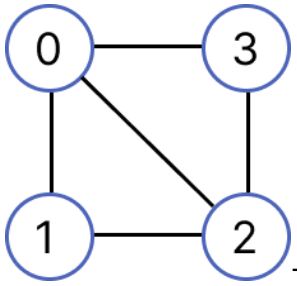
- 간선: 노드 사이를 연결하는 선

- 가중치: 간선에 할당된 값 또는 비용

- 유향 그래프: 노드간 이동 방향 및 그에 따른 가중치가 있는 그래프



유향



무향

- 무향 그래프: 노드간 간선과 그에 따른 가중치만 있는 그래프

종류

- 너비 우선 탐색 (BFS)

- 깊이 우선 탐색 (DFS)

- 다익스트라 (Dijkstra)

- 최우선 탐색 (Greed Best-First)

- A*

11. 너비 우선 탐색 (BFS)

한 노드에서 시작해서 해당 노드와 인접한 노드를 우선적으로 탐색

특징

- 큐 자료구조를 이용하여 구현됨

- 거쳐가는 경로가 해당 노드의 최단거리임

- 어떤 노드를 방문했었는지 여부를 반드시 검사 해야함

알고리즘

1. 시작 노드를 방문
2. 큐에 방문한 노드를 삽입(초기 상태에는 시작 노드만 저장)
3. 큐에서 맨 앞에 있는 노드와 인접한 노드를 차례로 방문
4. 큐에 방문한 노드들을 순서대로 삽입한다.
5. 큐가 전부 비워질 때까지 반복한다.

장점

- 순환 경로에 빠지는 일이 없다.

- 모든 경로를 동시에 진행

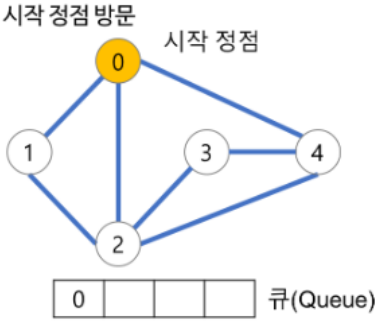
- 최단 경로 탐색에 유리함

단점

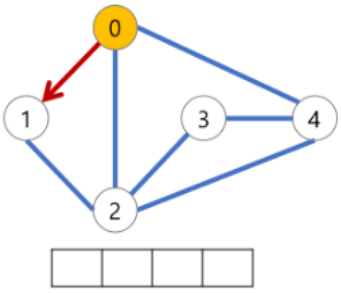
- 광범위하게 탐색하여 탐색시간이 길

- 메모리 사용량이 큼

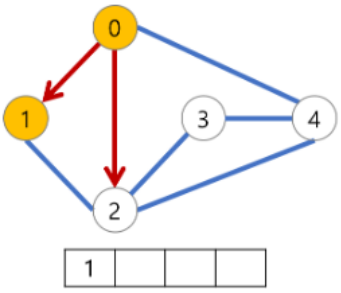
(1) 시작 정점 방문



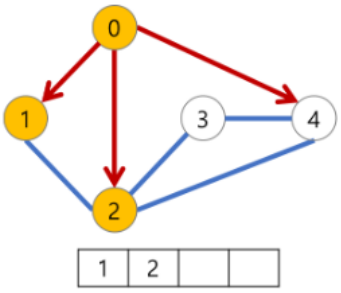
(2)



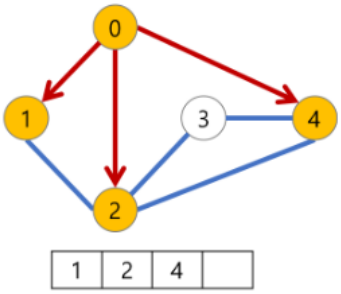
(3)



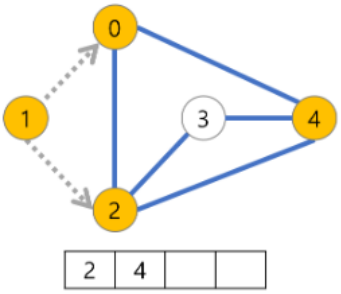
(4)



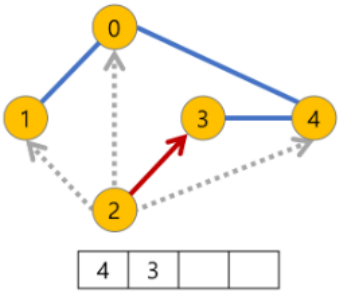
(5)



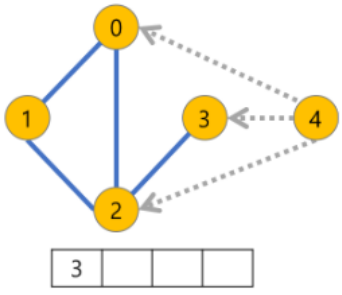
(6)



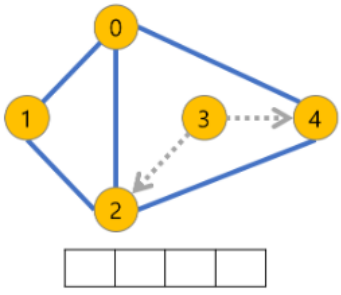
(7)



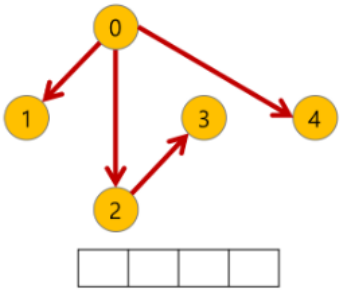
(8)



(9)



(10) 탐색 결과(방문 순서: 0,1,2,4,3)



활용 예시

-미로 탐색

-네트워크 브로드 캐스팅

12. 깊이 우선 탐색 (DFS)

BFS 와 달리 가능한 깊이 탐색 후 막히면 이전 갈림길에서 다른 경로를 깊이 탐색

특징

- 재귀호출, 스택으로 구현

- 현 경로 노드만 기억함

- 백트래킹(모든 경우의 수를 고려하는 알고리즘)에 사용됨

- 어떤 노드를 방문했었는지 여부를 반드시 검사 해야함

알고리즘

1. 시작노드 방문
2. 방문한 노드 표시
3. 시작 노드와 인접한 노드 모두 방문
4. 해당 노드를 시작하여 더 이상 못 들어갈 때까지 탐색
5. 못 들어가면 방문하지 않은 정점 찾기
6. 방문하지 않은 정점 없어질 때까지 4, 5 반복

장점

- 현 경로 노드만 기억하므로 메모리가 적음

- 목표 노드가 깊을수록 빨리 구함

- 구현이 쉬움

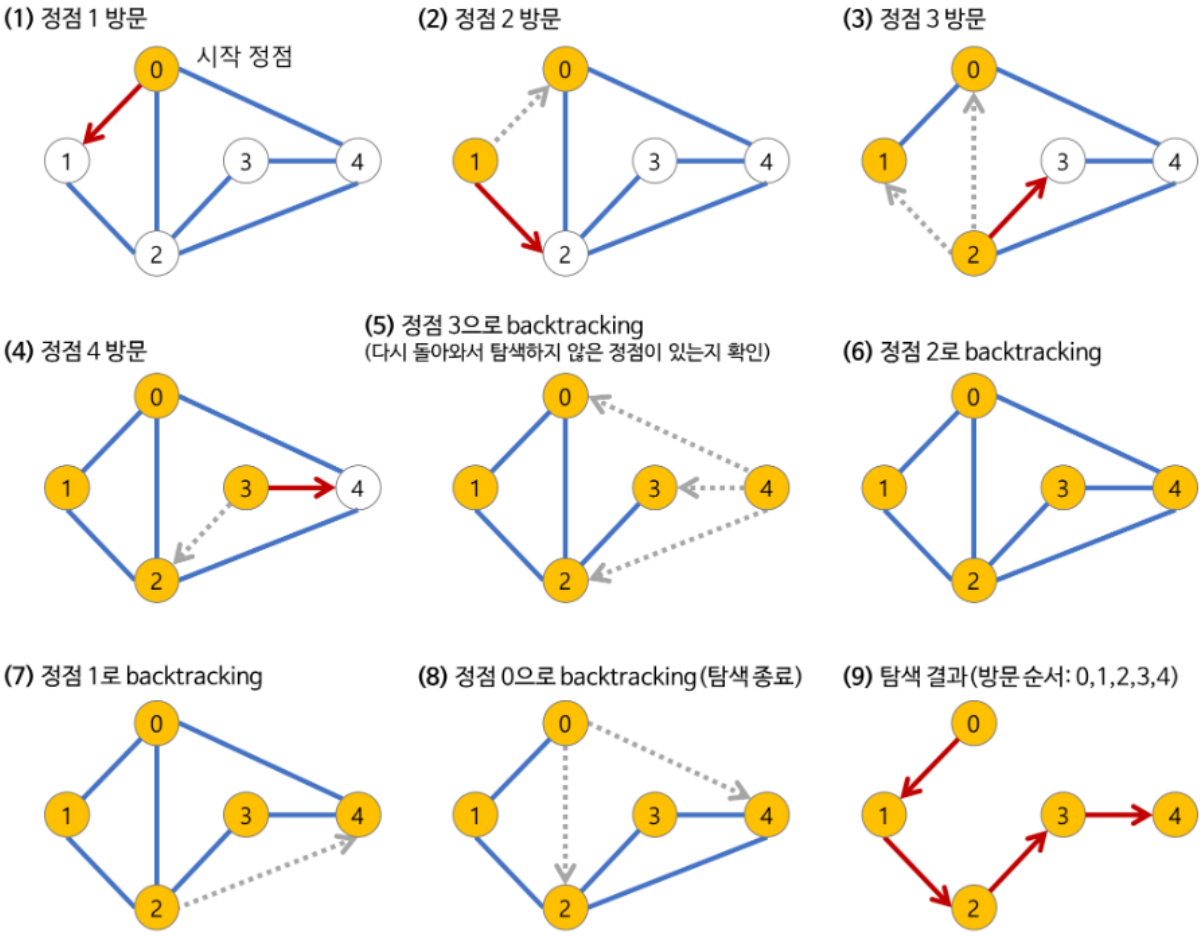
- 응용 범위 넓음

단점

- 목표 노드가 없는 경로도 깊이 탐색하여 느릴 수 있음

- 목표에 다다르면 탐색을 끝내버려 최단 경로가 아닐 수 있음

- 무한 루프 가능성



활용 예시

- 퍼즐 탐색

- 위상 정렬

13. 다익스트라 (Dijkstra)

그래프의 한 노드에서 모든 노드까지의 최단거리를 구하는 알고리즘

특징

- 그래프 방향의 유무 상관 없음
- 음수 가중치가 없는 간선에서 사용 가능
- 최단거리 보장
- 우선순위 큐(힙) 사용시 효율적

알고리즘

1. 최단거리 저장할 배열을 만듦
2. 출발 노드는 0, 이외는 INF(매우 큰 수)
3. 출발할 노드를 저장
4. 출발할 노드에서 갈 수 있는 노드들의 거리 계산을 수행
5. 출발 노드를 방문 완료 상태로 바꿈
6. 미방문한 노드 중 출발 노드에서 제일 짧은 노드를 선택 후 4, 5 반복
7. 도착 노드가 방문 완료가 되거나 더 이상 방문할 노드가 없으면 끝냄

장점

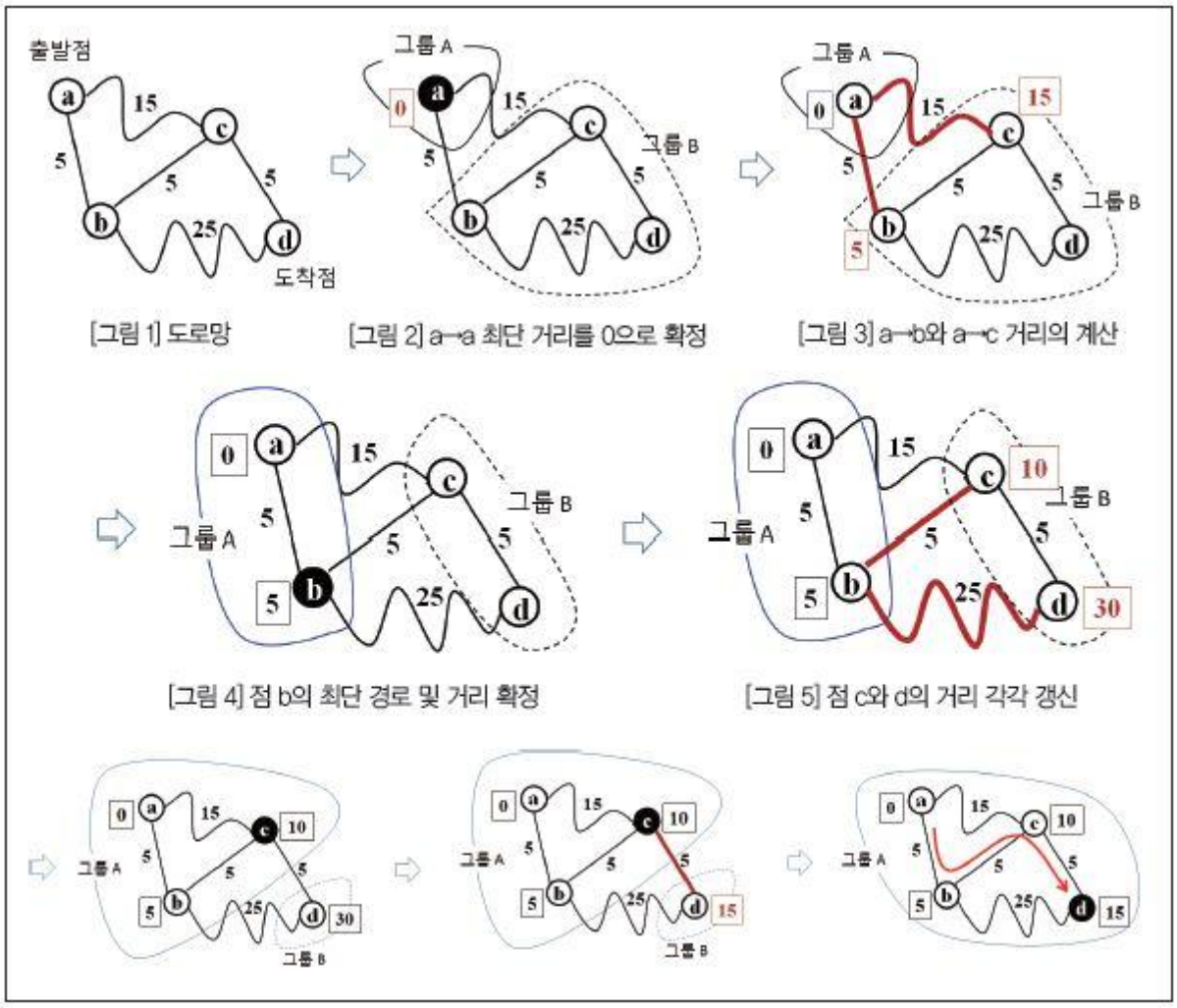
- 가능한 적은 비용
- 가장 빠른 경로 찾음
- 실용적

단점

- 음수 가중치 처리 불가

- 모든 정점 계산

- 효율적으로 만들기 복잡함



활용예시

- 네비게이션

- 루빅스 큐브 풀기(A*와 연결)

14. 탐욕 알고리즘 (Greedy Best-First)

매 경우마다 가장 최적이라 판단되는 방향으로 나아가는 알고리즘

특징

- 일부 경로만 탐색
- 매우 빠름
- 부분 최적 해(최적 보장 X)
- 현재 상태에서 최적의 경우 선택

알고리즘

1. 현 노드에서 특정 조건에 맞춰(ex 거리) 노드 경로 분석
2. 분석한 노드들 중 가장 조건에 맞는 경로 이동
3. 경로가 끝날 때 까지 1, 2 를 반복

알고리즘 조건

1. 이전 선택이 현재에 영향을 주지 않음
2. 부분 문제의 결과가 전체에 적용

장점

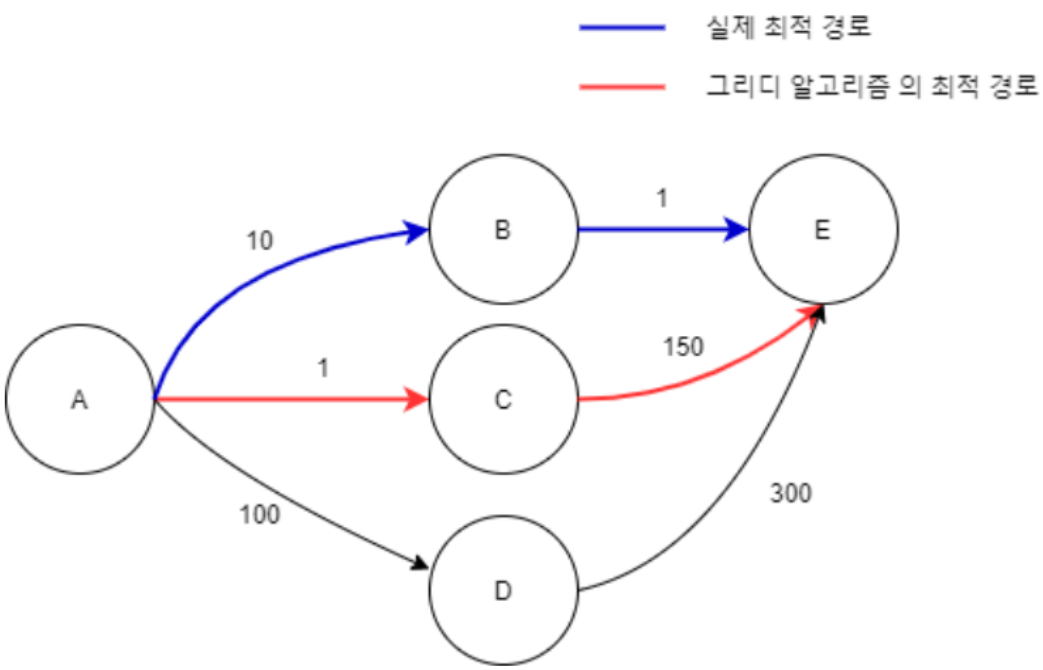
- 부분 분석으로 매우 빠름
- 단순한 알고리즘
- 무난한 수준의 해

단점

- 최적의 해 아닐 가능성 있음

- 정확성을 요구하면 사용할 수 없음

- 외판원 순회 문제와 같이 사용이 제한되는 경우가 많음



활용 예시

- 거스름돈 계산(특정 조건 한해)

- 객체 추적

- 결정 트리 학습

15. A*

다익스트라 알고리즘의 확장형, 현재 상태와 다음 상태를 고려해 최적의 경우를 먼저 탐색

특징

- 우선순위 큐 사용 (작은 값 우선 탐색)

- 현재 상태 + 다음 상태

- 다익스트라 보다 현실에서 사용하기 좋음

알고리즘

1. 시작 노드를 우선순위 큐에 담음
2. 큐에서 최우선 노드를 가져옴
3. 해당 노드에서 갈 수 있는 다음 노드를 찾음
4. 현재 비용과 그 노드들의 비용을 계산함
5. 계산 후 큐에 담음
6. 위 알고리즘을 목표 노드에 도달할 때까지 반복

장점

- 다익스트라 기반이라 최우선 경로 보장

- 직접적인 현실 문제에 적용하기 쉬움

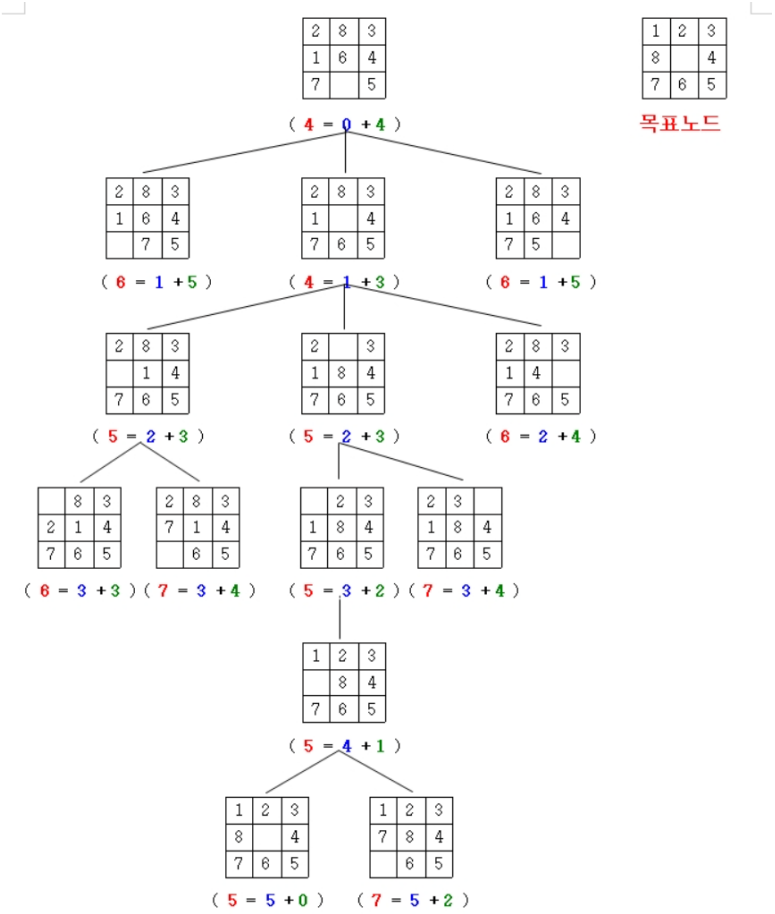
- 다익스트라 보다 효율적

- 유연한 개조

단점

- 휴리스틱(다음 상태로 이동할 때 비용 계산)에 의존성이 큼

- 다익스트라의 단점과 유사



활용예시

- 루빅스 큐브 해결

- 네비게이션

- 게임 AI