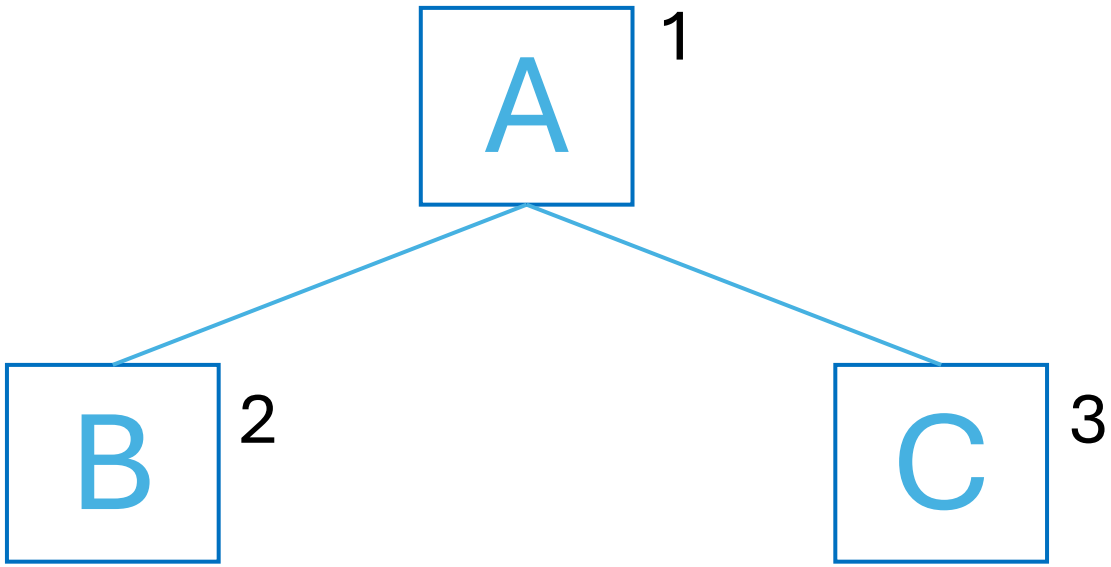
자료구조 (Data Structure
10주차: 그래프 최소 트리

트리 정리

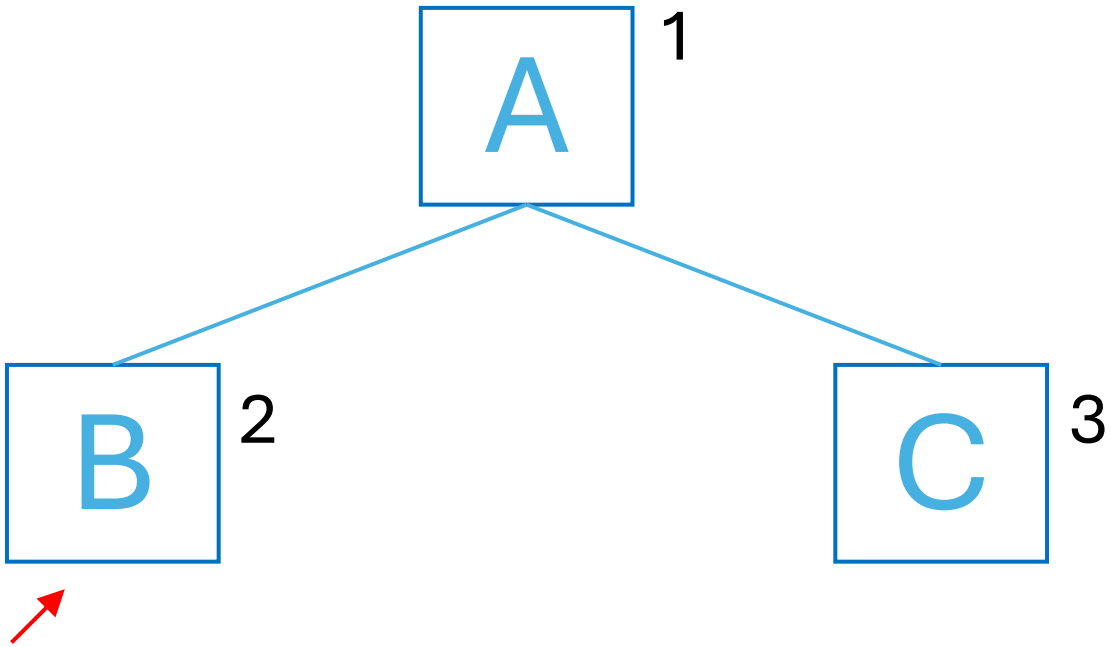
- 최소힙: 모든 노드 데이터가 자식 노드보다 작음.



트리 정리

- 최소힙: 모든 노드 데이터가 자식 노드보다 작음.

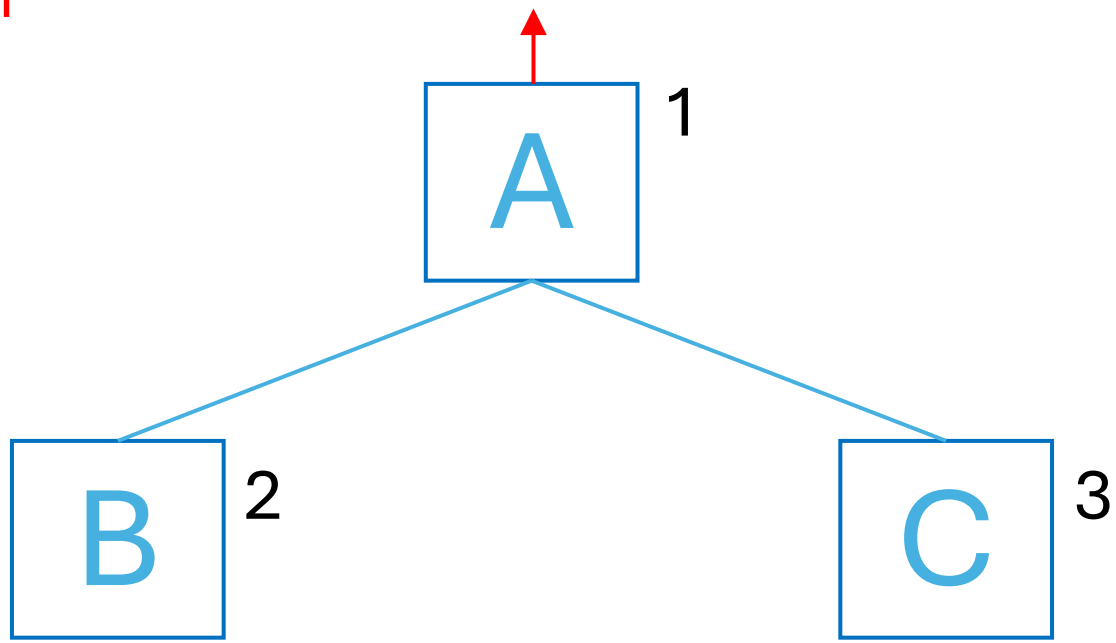
추가하기



트리 정리

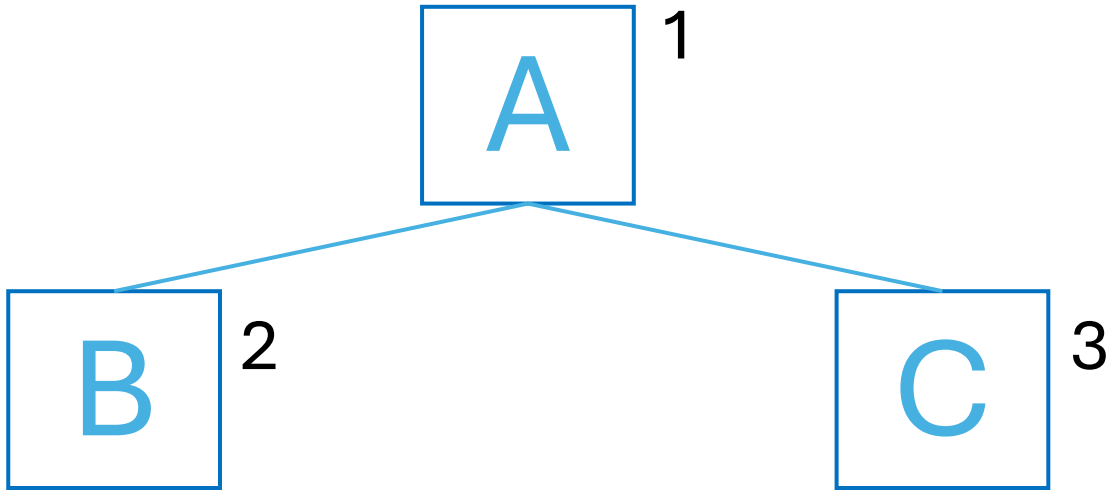
- 최소힙: 모든 노드 데이터가 자식 노드보다 작음.

삭제하기



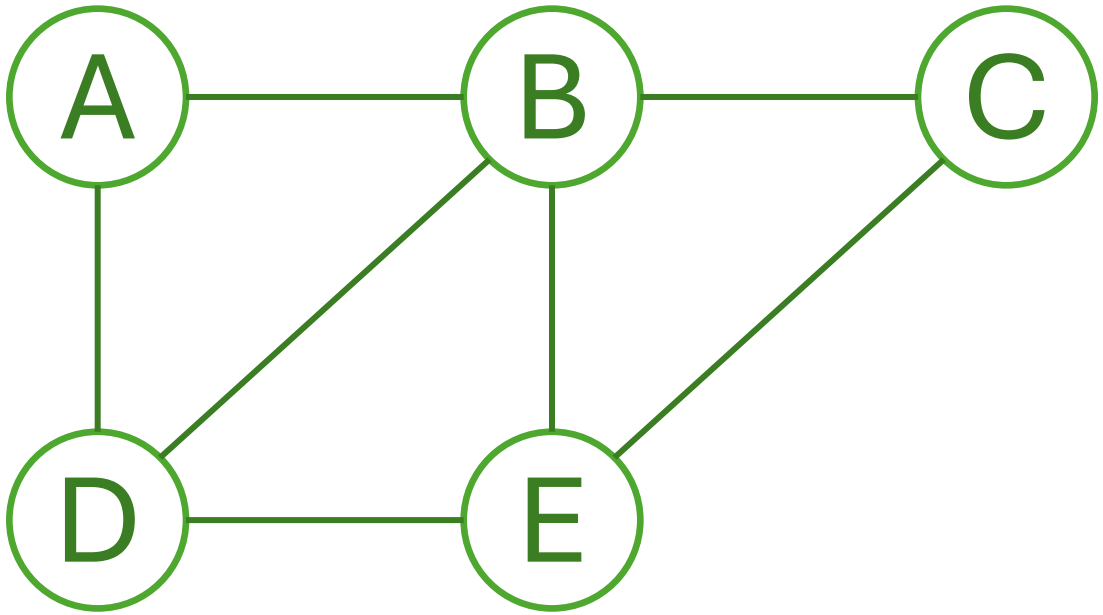
트리 정리

- 최소힙을 쓰면 가장 작은 값부터 꺼낼 수 있다



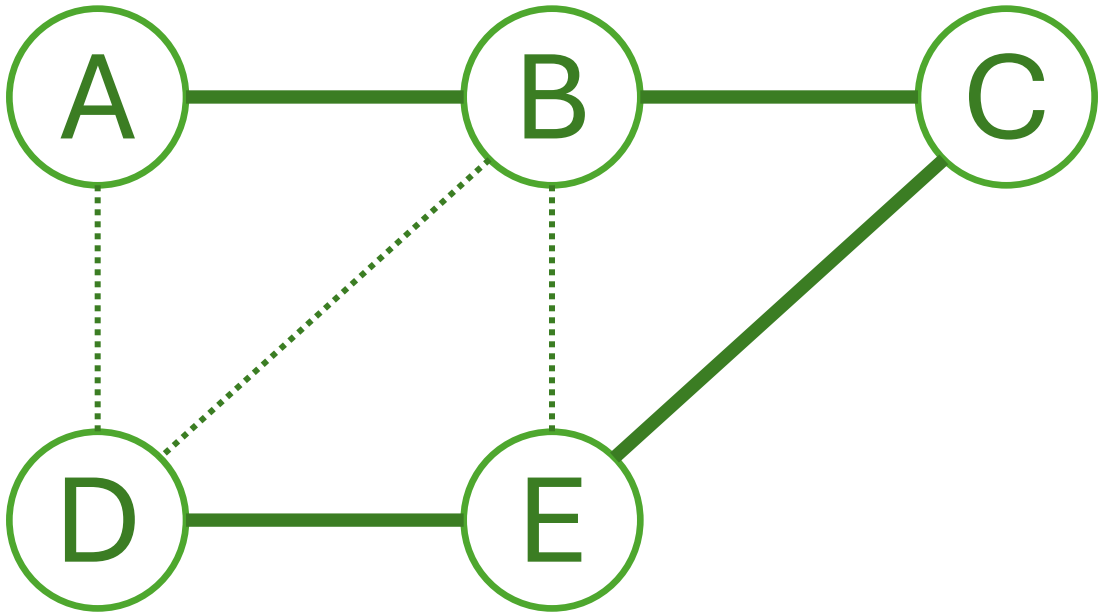
그래프 정리

- 그래프: 노드들이 연결선으로 연결된 구조



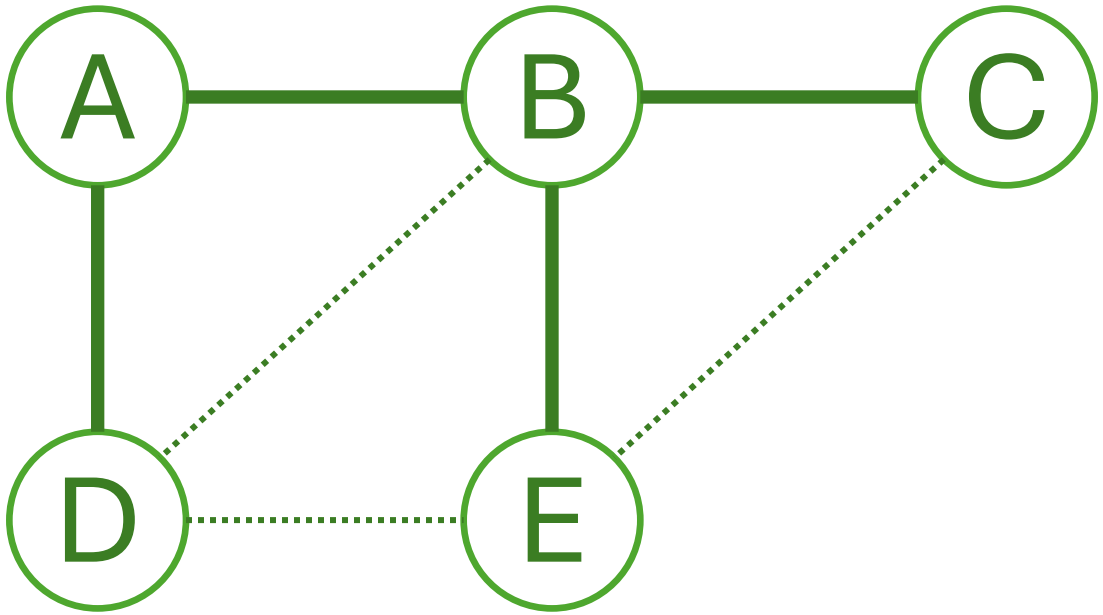
그래프 정리

- 깊이우선탐색(DFS): 이웃마다 연결된 모든 노드를 방문해서 트리를 만들고, 다음 이웃 방문



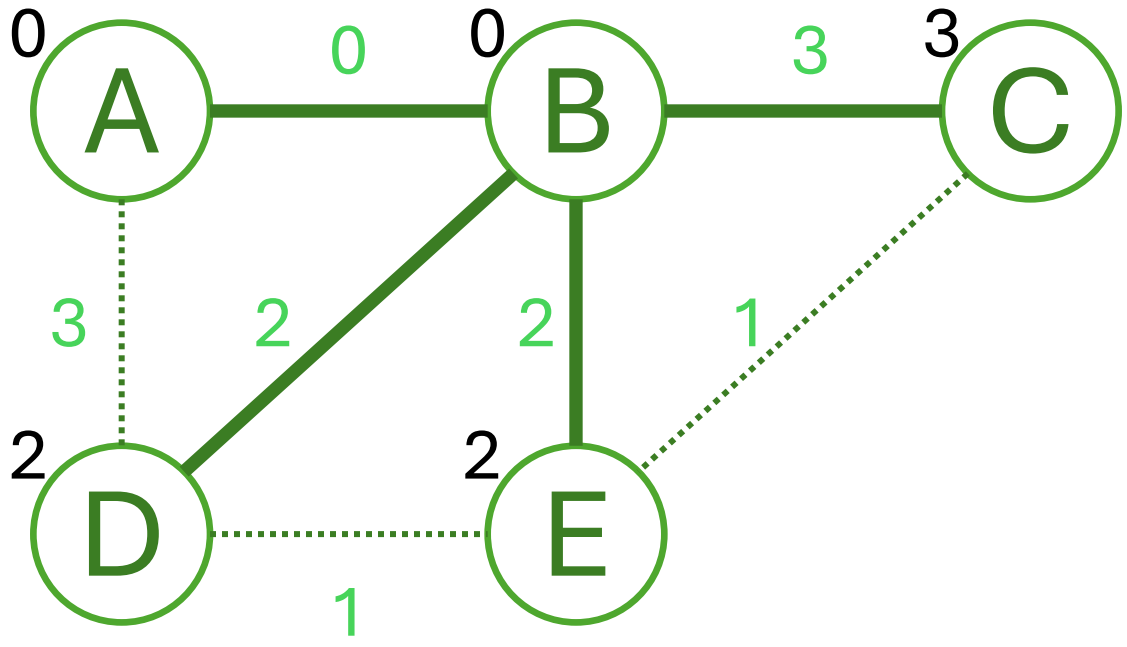
그래프 정리

- 너비우선탐색(BFS): 이웃들을 먼저 방문해서 트리를 만들고, 이웃의 이웃들을 방문



그래프 정리

- 최단 경로(DIJKSTRA): 가장 가까운 이웃부터 방문



이번 시간 목차

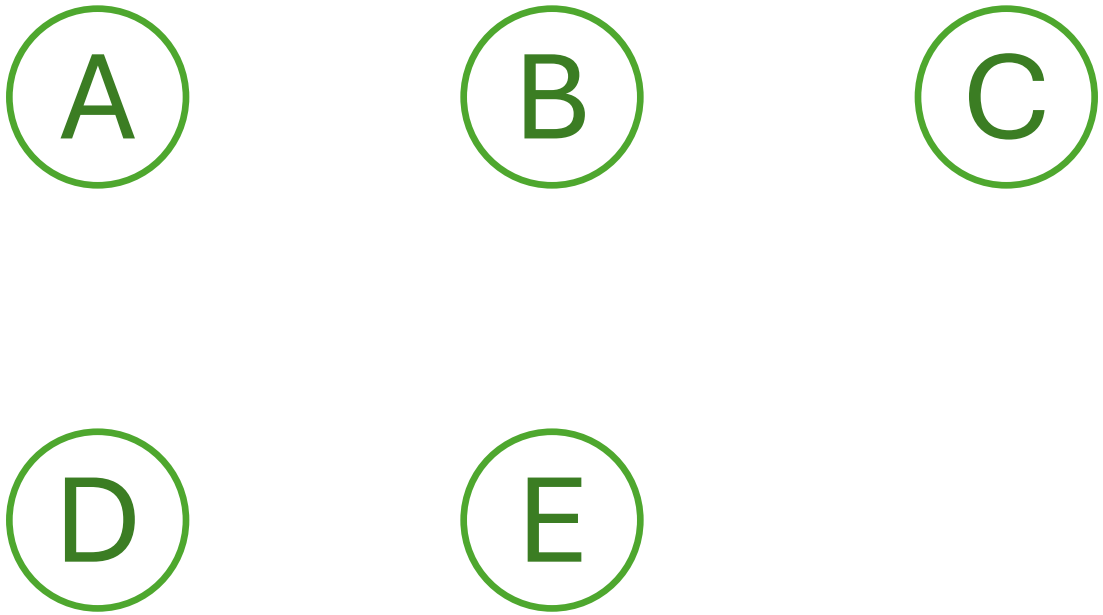
- 그래프를 최단 길이 트리로 만드는 방법 (MST)
- 최단 길이 트리 알고리즘 예제

최단 길이 트리

- 마을에 전기를 공급하는 비용을 최적화하려면?

최단 길이 트리

- 마을간 송전선 건설 비용을 모델링하면?



최단 길이 트리

- 송전선 건설 비용을 최소화 하려면?

최단 길이 트리

- spanning tree: 그래프 연결선의 일부와 노드 전체로 이뤄진 트리

- minimum spanning tree: 전체 연결선 비용의 총 합이 최소인 트리

최단 길이 트리 찾기

- DJKSTRA: 최단 거리가 정해진 집합 확장
  - 거리가 가까운 노드가 추가됨

- PRIM: 최단 길이 트리 확장
  - 트리와 연결선이 짧은 노드가 추가됨

최단 길이 트리 찾기 - PRIM 1/3

함수 PRIM(그래프 G, 시작 노드 s):

- to_visit = 우선순위 큐 (최소힙)
- 반복: G의 모든 노드 u에 대해,

    u->key = 아주 큰 수 // 트리에 더해지는 길이

    u->parent = 없음

    u->status = 트리에추가되기전

- s->key = 0
- to_visit에 s 추가

최단 길이 트리 찾기 - PRIM 2/3

- 반복: to_visit에 데이터가 있는 동안

  u = DEQUEUE(to_visit)

  만약: u->status == 트리에추가됨이면,

  다음 반복으로

  u->status = 트리에추가됨

최단 길이 트리 찾기 - PRIM 3/3

반복: u의 모든 이웃 노드 v와 (u, v)를 연결선 e,
만약: v->key ≤ e->length이면,
다음 반복으로

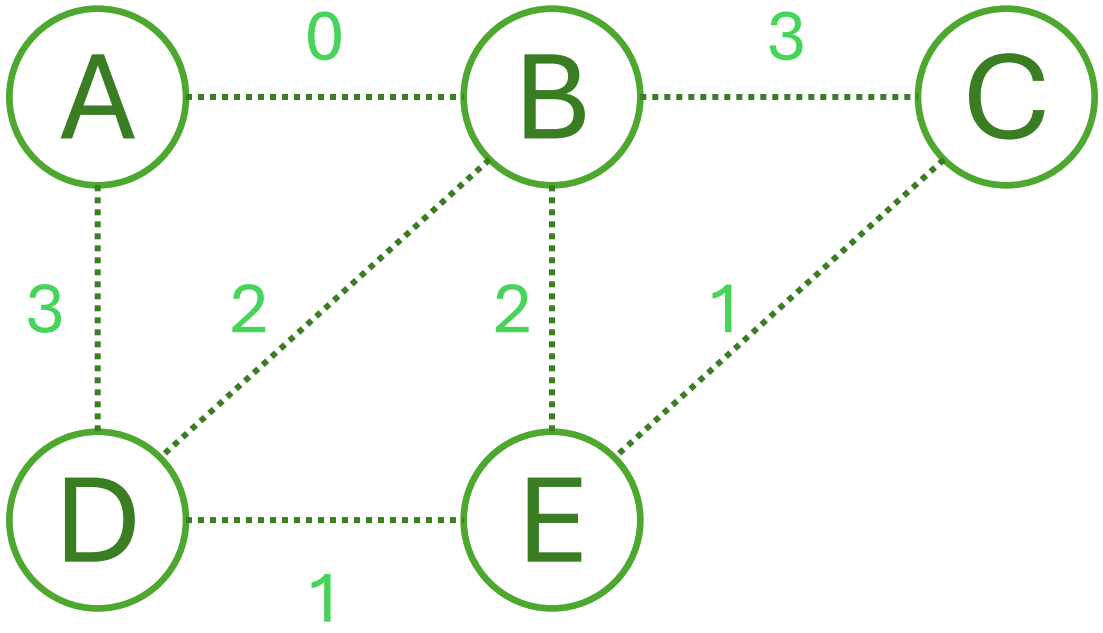
v->parent = u

v->key = e->length

ENQUEUE(to_visit, v)

최단 길이 트리 찾기 - PRIM

- PRIM 함수로 A부터 최단 길이 트리를 찾으면?



최단 길이 트리 찾기 - PRIM

- 트리가 확장되어가는 이유는?

최단 길이 트리 찾기 - PRIM

- 사이클을 형성하지 않는 이유는?

최단 길이 트리 찾기 - PRIM

- 최단 길이가 보장되는 이유는?

최단 길이 트리 찾기 - PRIM

- 비용이 너무 비싼 구간은 끊는 것이 허용된다면?

최단 길이 트리 찾기 - KRUSKAL

- 처음에 모든 노드는 각자 다른 트리의 루트
- 서로 다른 트리를 잇는 연결선 중 제일 짧은 것부터 연결된 트리를 하나로 합치기

최단 길이 트리 찾기 - KRUSKAL

- 어느 트리에 속했는지 소속을 확인할 수 있는 방법은?

최단 길이 트리 찾기 - KRUSKAL

함수 FIND_ROOT(그래프 노드 v):

- 만약: v != v->root이면,

  v->root = FIND_ROOT(v->parent)

- 반환: v->root

최단 길이 트리 찾기 - KRUSKAL

- 두 트리의 소속을 합치는 방법은?

최단 길이 트리 찾기 - KRUSKAL

- 트리 소속을 합치며 트리의 균형을 유지하는 방법은?

최단 길이 트리 찾기 - KRUSKAL

- 트리 소속을 합치며 트리의 균형을 유지하는
  더 효율적인 방법은?

최단 길이 트리 찾기 - KRUSKAL

함수 UNION(그래프 노드 u, 그래프 노드 v):

- x = find_set(u);
- y = find_set(v);
- 만약: x->rank > y->rank

  y->p = x
- 그외:

  x->p = y

  만약: x->rank == y->rank

  y->rank += 1

최단 길이 트리 찾기 - KRUSKAL

- 처음에 모든 노드는 각자 다른 트리의 루트
- 서로 다른 트리를 잇는 연결선 중 제일 짧은 것부터 연결된 트리를 하나로 합치기

최단 길이 트리 찾기 - KRUSKAL

함수 KRUSKAL(그래프 G):

- 반복: G의 모든 노드 v에 대해,

  v->parent = NULL

  v->root = v

  v->rank = 0

- G의 연결선 리스트 e를 길이 오름차순으로 정렬
- 만약: x->rank > y->rank

  y->root = y->parent = x

최단 길이 트리 찾기 - KRUSKAL

- 그외:

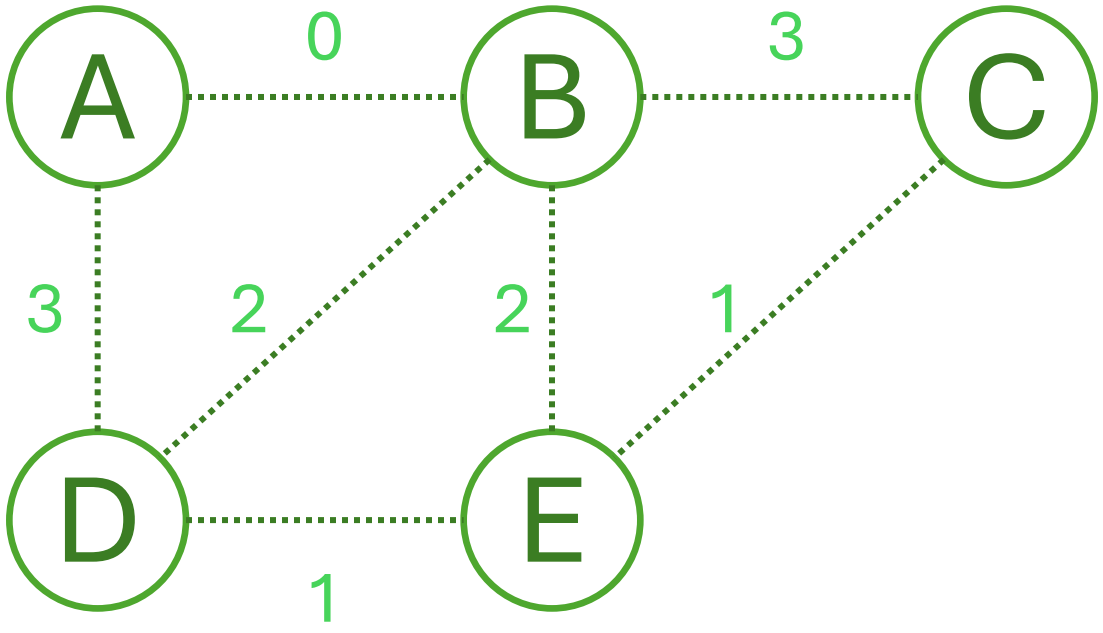
  x->root = x->parent = y

  만약: x->rank == y->rank

  y->rank += 1

최단 길이 트리 찾기 - KRUSKAL

- KRUSKAL 함수로 최단 길이 트리를 찾으면?

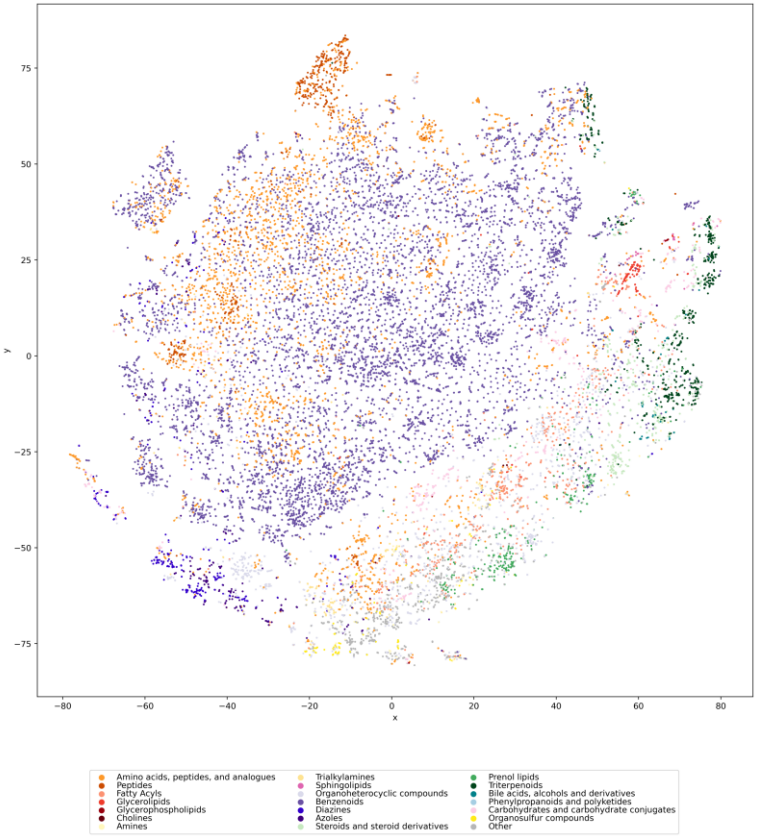


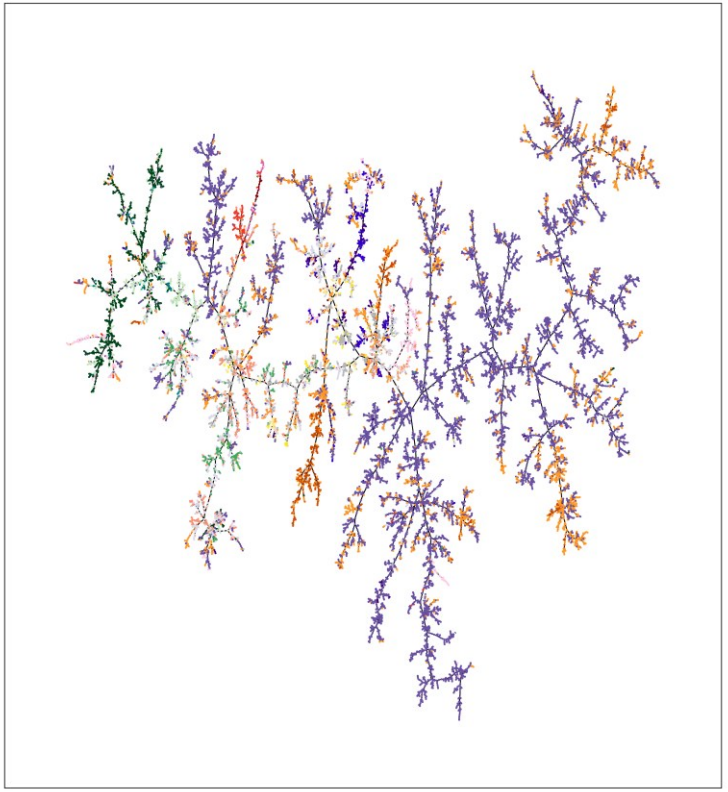
최단 길이 트리 찾기 - KRUSKAL

- KRUSKAL의 응용 - 이미지 분할

최단 길이 트리 찾기 - KRUSKAL

- KRUSKAL의 응용 - 화합물 데이터 분석





최단 길이 트리 찾기 - KRUSKAL

- KRUSKAL의 응용 - 비즈니스 연결 데이터 분석

(a) Maximum Spanning Tree (default)
- TF-IDF (default)
- Weighted Co-Occurrence
- Co-Occurrence

(b) Raw Degree

(c) None

요약

- PRIM: 한 노드에서부터 트리를 확장시킨다
- KRUSKAL: 짧은 연결선부터 숲을 확장시킨다

프로그래밍 방법론

- 욕심쟁이 방법: 주어진 상황에서 최선만 선택한다
- 변화법: 최선이 아닌 선택지도 정보를 기록한다

프로그래밍 방법론: 봉우리 찾기

- 욕심쟁이 방법: 가장 경사가 높은 방향으로 이동한다
- 변화법: 각 좌표의 고도를 기록한다

프로그래밍 방법론: 최단거리

- 욕심쟁이 방법: n번째 최단거리 노드만 확인한다
- 변화법: 모든 연결선에서 최단거리를 n번 구한다

프로그래밍 방법론: 최소 길이 트리

- 욕심쟁이 방법:

  방법 1) 가장 짧은 연결선을 처리한다

  방법 2) 현재 트리에서 나가는 가장 짧은 연결선을 처리한다

- 분할정복법: 모든 노드를 트리로 보고 합쳐나간다

프로그래밍 방법론: 정렬

- 무대포방법: 모든 순열 중에 정렬된 것을 찾는다
- 욕심쟁이방법: selection sort, insertion sort
- 분할정복방법: quick sort, merge sort

PRIM, DIJKSTRA에 특화된 피보나치 힙

- 루트 리스트와 최소 루트 주소로 관리
- 추가: 루트 리스트에 추가
- 삭제: 최소 루트 삭제, 자식들을 루트 리스트에 추가
  루트 리스트에서 자식 수가 같으면 합치기
- 값 변경: 힙 속성 불만족시 잘라서 루트리스트에 추가

일차원 자료구조 정리

- 배열리스트: 데이터를 메모리 한쪽에 모아서 순서대로 저장해놓은 구조

일차원 자료구조 정리

- 배열리스트: 데이터를 메모리 한쪽에 모아서 순서대로 저장해놓은 구조

추가하기

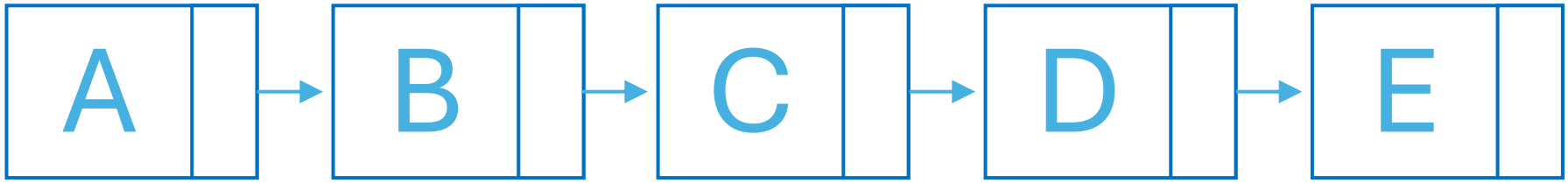
일차원 자료구조 정리

- 배열리스트: 데이터를 메모리 한쪽에 모아서 순서대로 저장해놓은 구조

삭제하기

일차원 자료구조 정리

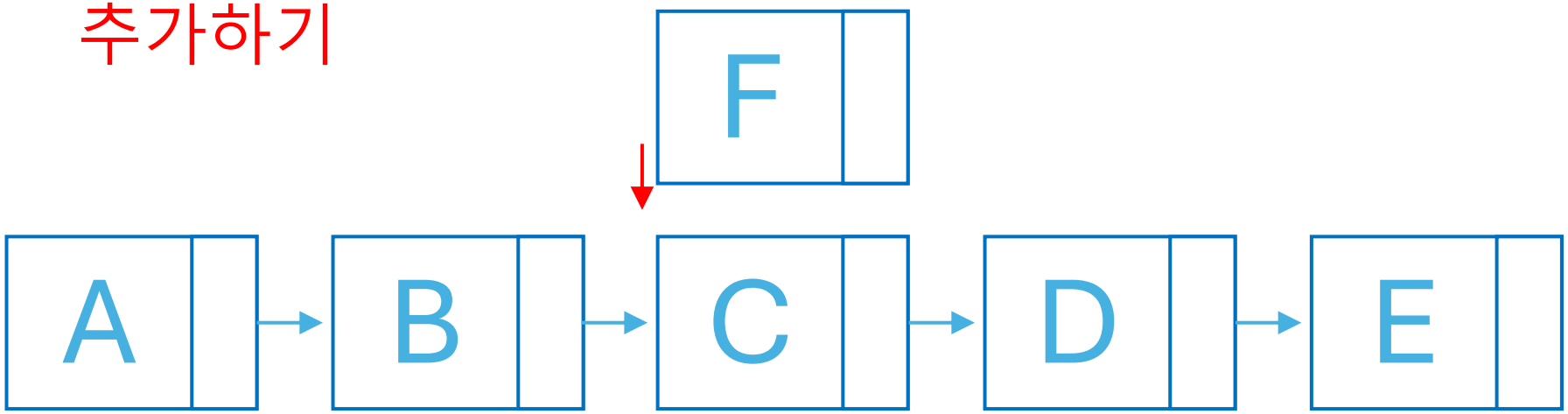
- 링크드리스트: 데이터와 다음 데이터 주소를 같이 저장해놓은 구조 (단방향)



일차원 자료구조 정리

- 링크드리스트: 데이터와 다음 데이터 주소를 같이 저장해놓은 구조 (단방향)

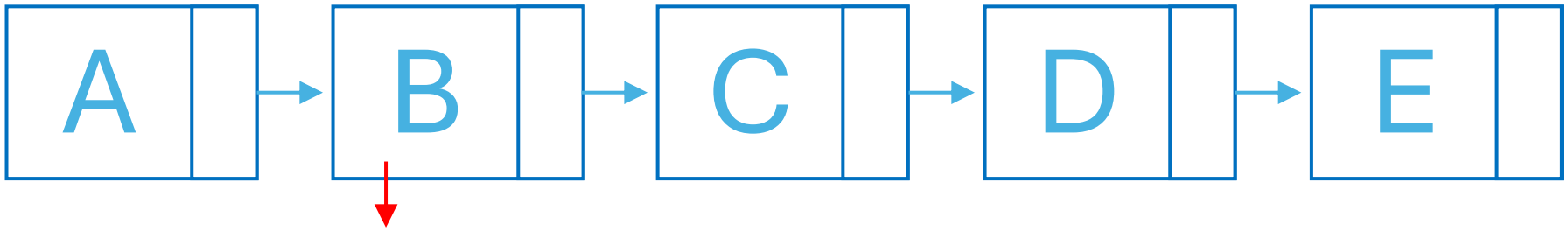
추가하기



일차원 자료구조 정리

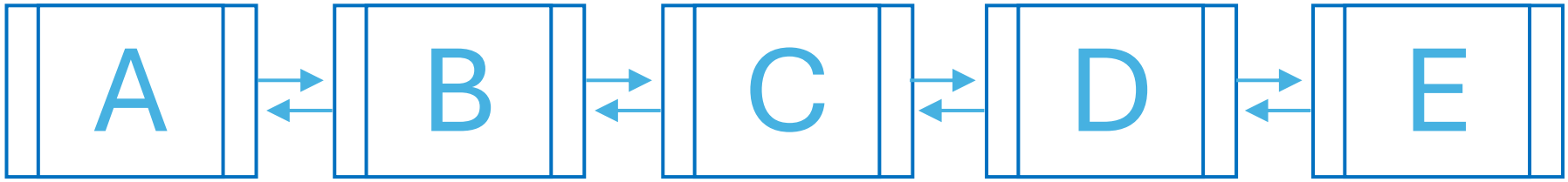
- 링크드리스트: 데이터와 다음 데이터 주소를 같이 저장해놓은 구조 (단방향)

삭제하기



일차원 자료구조 정리

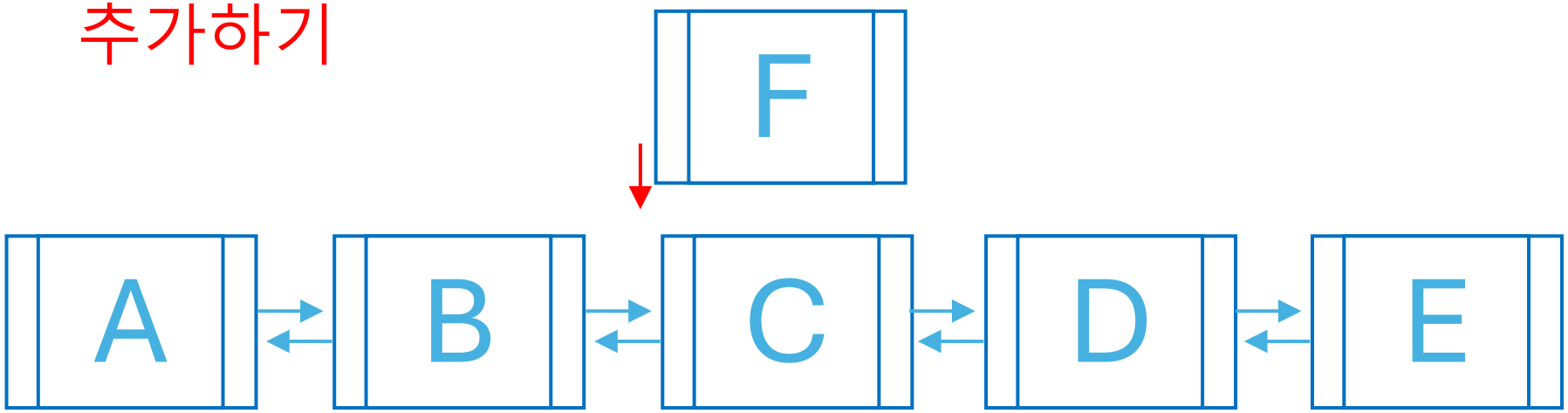
- 링크드리스트: 데이터와 이전, 다음 데이터 주소를 (양방향) 같이 저장해놓은 구조



일차원 자료구조 정리

- 링크드리스트: 데이터와 이전, 다음 데이터 주소를 (양방향) 같이 저장해놓은 구조

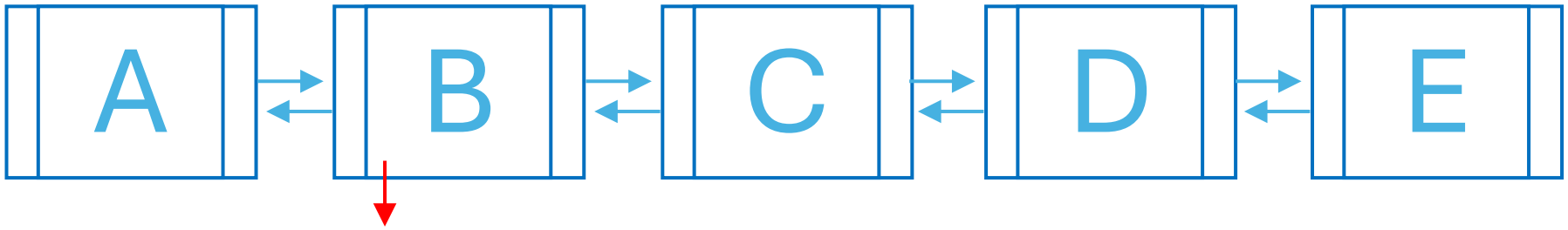
추가하기



일차원 자료구조 정리

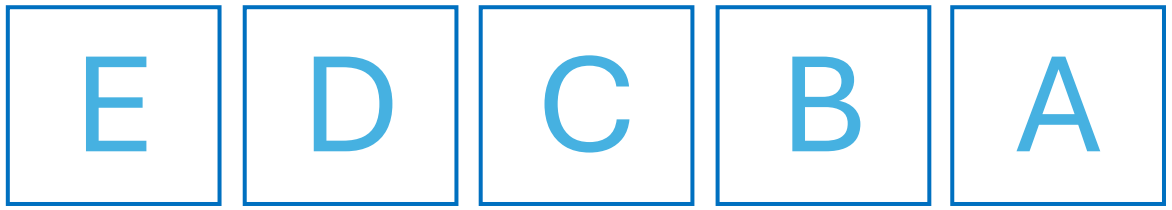
- 링크드리스트: 데이터와 이전, 다음 데이터 주소를 (양방향) 같이 저장해놓은 구조

삭제하기



일차원 자료구조 정리

- 스택: 마지막에 추가한 데이터가 삭제되는 구조



일차원 자료구조 정리

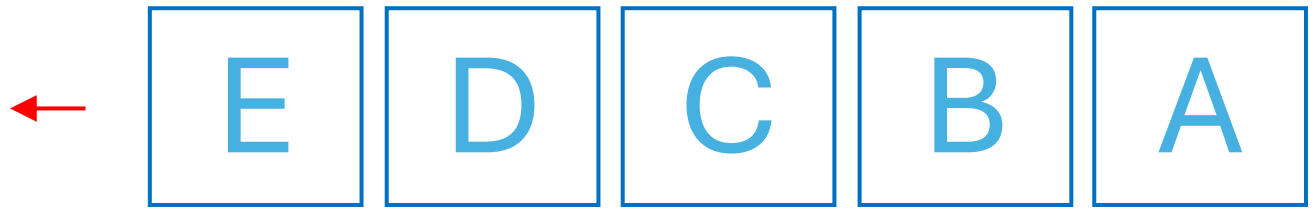
- 스택: 마지막에 추가한 데이터가 삭제되는 구조

추가하기

일차원 자료구조 정리

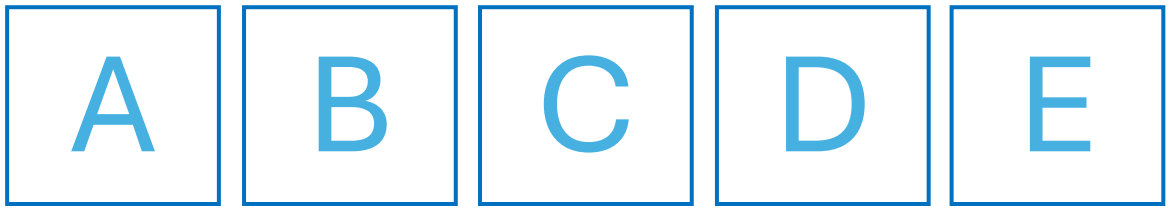
- 스택: 마지막에 추가한 데이터가 삭제되는 구조

삭제하기



일차원 자료구조 정리

- 큐: 처음에 추가한 데이터가 삭제되는 구조



일차원 자료구조 정리

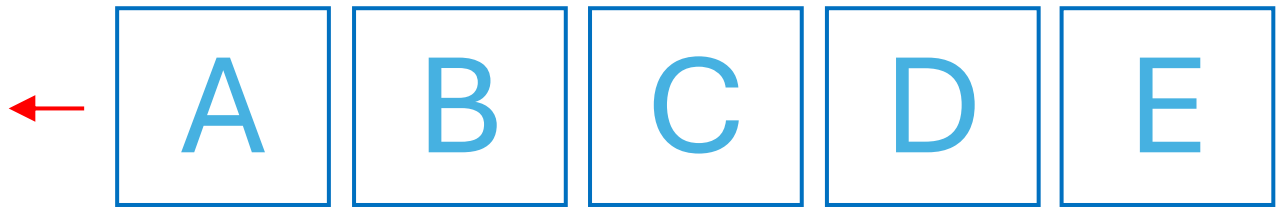
- 큐: 처음에 추가한 데이터가 삭제되는 구조

추가하기

일차원 자료구조 정리

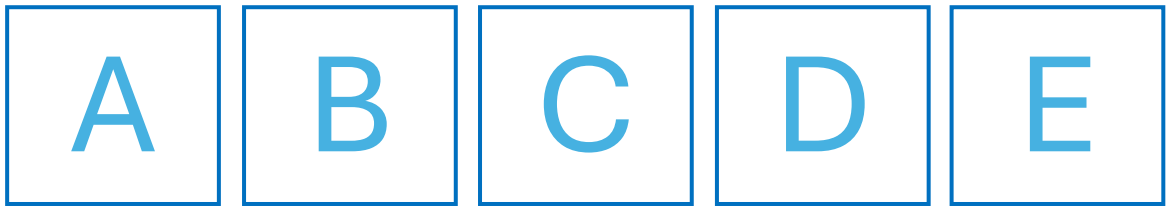
- 큐: 처음에 추가한 데이터가 삭제되는 구조

삭제하기



일차원 자료구조 정리

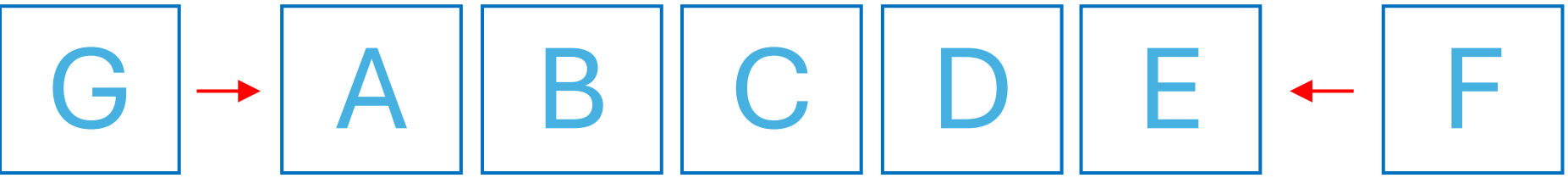
- 덱: 양방향에 데이터 추가/삭제가 가능한 구조



일차원 자료구조 정리

- 덱: 양방향에 데이터 추가/삭제가 가능한 구조

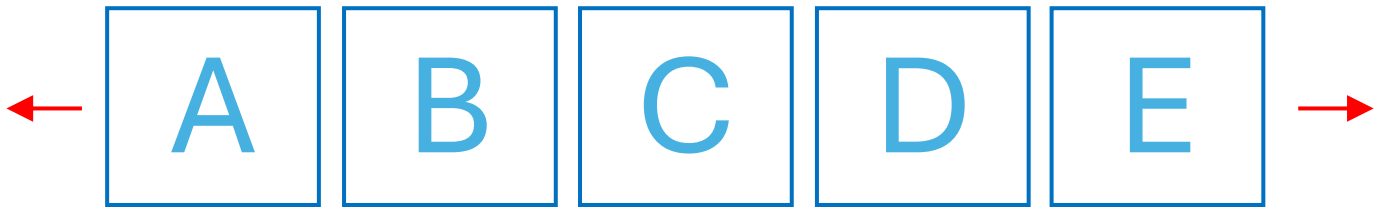
추가하기



일차원 자료구조 정리

- 덱: 양방향에 데이터 추가/삭제가 가능한 구조

삭제하기



트리 정리

- 트리: 루트에서 시작. 부모 자식 관계로 연결.
- 이진 트리: 자식 노드가 두개 이하