Algorithm

> https://gyoogle.dev/blog/algorithm/Selection%20Sort.html
>
> https://github.com/GimunLee/tech-refrigerator/tree/master/Algorithm/resources

1. SORT

1-1. Selection sort

```
def selectionSort(arr):
    n=len(arr)
    for i in range(n-1):
        temp=i
        for j in range(i+1, n):
            if arr[temp]>arr[j]:
                temp=j

        a[i],a[temp]=a[temp],a[i]
```

선택정렬은 우선 위치를 선정하고 그 위치에 들어올 값을 찾는 방식이다.

시간복잡도 : O(n^2)

공간복잡도 : O(n)

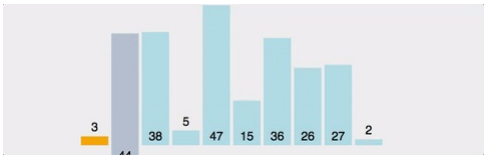
- 장점
  1. 알고리즘이 단순하다.
  2. 제자리 정렬
- 단점 :
  1. 불안정 정렬이다.
  2. 비교적 시간복잡도가 큰 편이다.

1-2. Insertion sort

```
def insertionSort(arr):
    n=len(arr)

    for i in range(1,n):
        temp=arr[i]
        for j in range(i-1,-1,-1):
            if arr[j]>temp:
                arr[j+1]=arr[j]
                arr[j]=temp

            else:
                break
```



삽입정렬은 값을 선정하고 그 값이 들어갈 위치를 선정하는 것이다.

시간복잡도 : O(n^2)

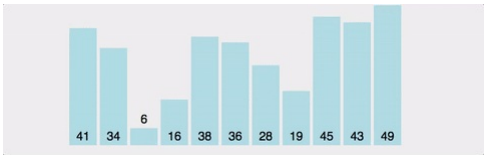
공간복잡도 : O(n)

- 장점
  1. 안정정렬이다.
  2. 이미 정렬이 되어있는 경우(최선시간복잡도) : O(n)
- 단점
  1. 정렬해야하는 배열이 커질경우 비효율적이다.

1-3. Bubble sort

```
def bubbleSort(arr):
    n=len(arr)

    for i in range(1,n):
        for j in range(i):
            if arr[j-1]>arr[j]:
                arr[j-1],arr[j]=arr[j],arr[j-1]
```



옆에있는 값이랑 비교하여 교환하는 방식으로 정렬해나가는 방법이다.

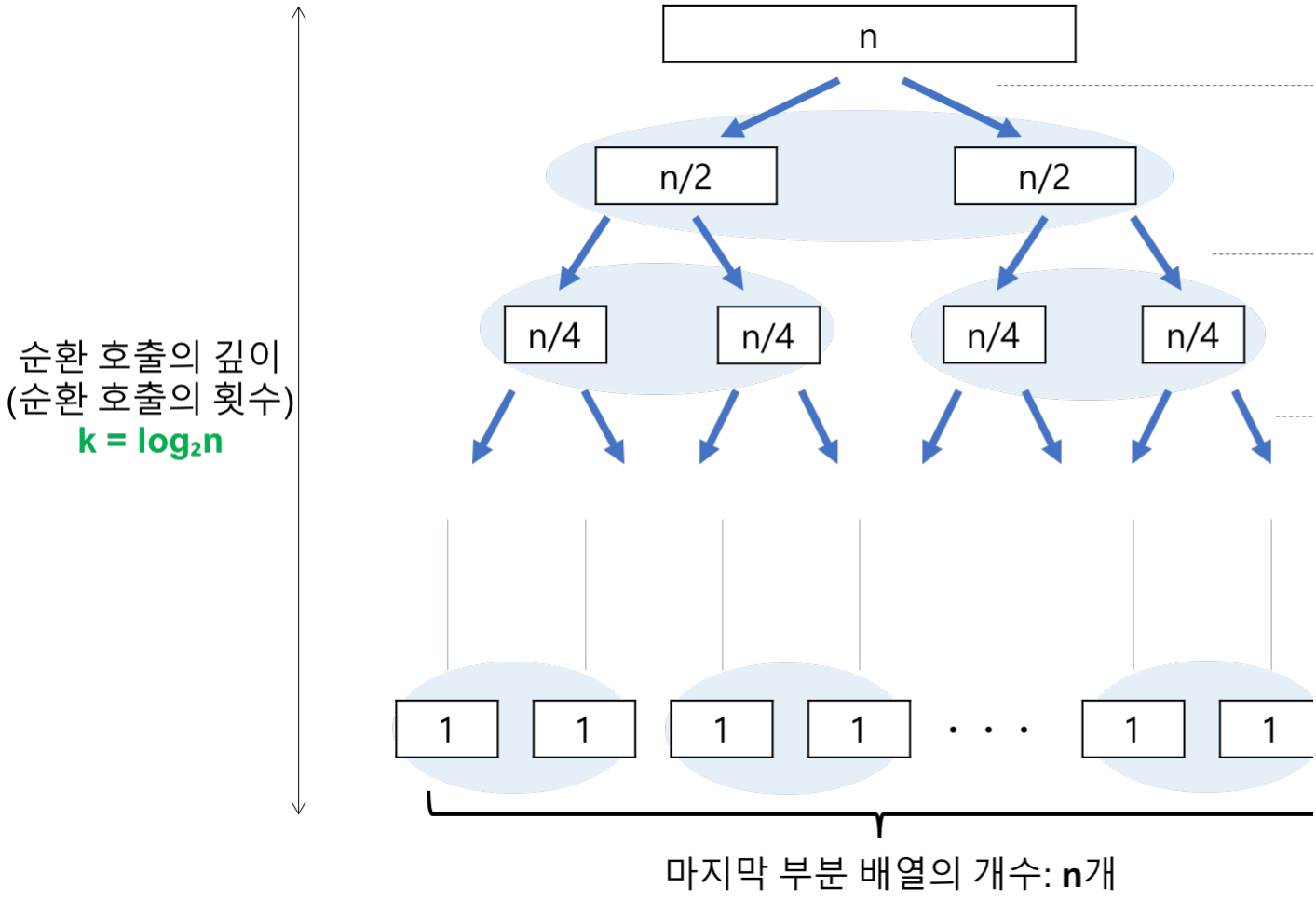
처음에 n-1번 비교하면 다음 회차때는 n-2번, n-3번... 비교횟수가 줄어든다.

시간복잡도 : O(n^2)

공간복잡도 : O(n)

- 장점 :
  1. 제자리정렬이다.
  2. 안정정렬이다.
- 단점 :
  1. swap이 많이 일어난다.

1-4. Quick sort



퀵정렬 과정

1. 배열에서 랜덤하게 pivot을 고른다.
2. pivot을 중심으로 왼쪽에는 작은 수, 오른쪽에는 큰 수로 분할한다. (분할)
3. 분할된 배열에서 1번부터 반복한다.

퀵정렬은 분할 정복법이다.

1회차 : n-1번비교

2회차 : n-2번 비교

....

(logn)회차 : 1번 비교

시간복잡도 : O(nlogn)

공간복잡도 : O(n)

<pivot에 따른 효율성>

만약 pivot을 최소값이나 최대값으로 잡힌다면 최대 시간복잡도가 O(n^2)번이 나올 수 있다.

이를 개선하기 위한 방법

1. pivot을 정할 때 가장 앞에있는 값과 중간값을 교환후에 정한다.
2. pivot을 랜덤한 위치에서 정한다.
3. 첫인덱스, 마지막인덱스, 중간인덱스 중에서 중간값을 pivot으로 정한다.

하지만 이러한 방법들을 한다고해서 반드시 O(n^2)의 시간복잡도를 개선할 수 있는것은 아니다.

그래서 나온 베스트 방법은 삽입정렬과 함께 사용하는것이다. 삽입정렬은 배열의 크기가 작을 경우 최선으로 O(n)번의 시간복잡도를 낼 수 있다. 그래서 일정 퀵정렬을 수행하다가 일정 배열 크기 아래에서부터는 삽입정렬을 하는 것이 효율적인 방법이라고 할 수 있다.

- 장점
    1. 다른 정렬에 비해서 빠른 편이다.
- 단점
    1. pivot 값을 최대값이나 최소값으로 잡을 경우 최악의 시간복잡도가 나온다.
    2. 불안정정렬이다.

1-5. Merge sort

합병정렬은 퀵정렬과 달리 우선 분할한 뒤에 정복을하는 방법이다.

(Devide->Merge)

1. 분할(divide)
2. 정복(sort)
3. 결합(merge)
4. 복사(copy)

1회차 비교 : 1번

2회차 비교 : 2번

...

(logn)회차 비교 : n-1번

시간복잡도 : O(nlogn)

공간복잡도 : O(n)

- 장점

1. linkedlist 정렬시에 효율적이다. (퀵정렬에서는 순차탐색이 필요하기때문에 비효율적이다.)
2. 퀵정렬보다 효율적이다. pivot때문에 성능차이가 발생하지 않는다.
3. 안정정렬이다.

- 단점
  1. 추가적인 메모리가 필요할 수 있다. (병합과정에서 생김) 이럴때는 퀵정렬을 선택하는 편이다.

1-6. Heap sort

완전 이진트리를 기본으로하여 정렬한다.

1. 배열 순서대로 완전 이진트리를 생성한다.
2. 힙트리를 구성한다. (부모노드가 자식노드보다 값이 크거나 같다)
3. 힙의 마지막 노드와 최대값(루트)와 교환하면서 힙의 사이즈를 줄인다. (정렬)

다음 유튜브 영상이 힙정렬 과정을 자세히 보여주고있다.

> https://www.youtube.com/watch?v=aOP81IhPOmw

<인덱스 참조>

left child = parent * 2 +1

right child = left + 1

시간복잡도 : O(nlogn)

- 장점
  1. 최대값이나 최소값을 구하기 쉽다.(최대힙, 최소힙의 루트)
- 단점
  1. 불안정 정렬이다.

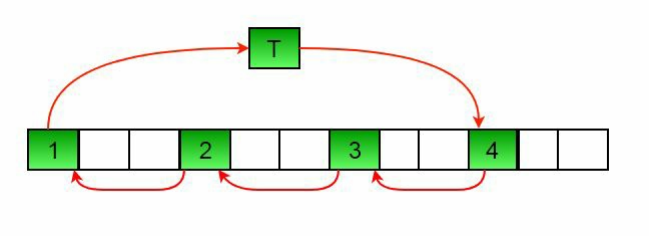
2. Array

정적으로 할당한 공간

- 장점
  1. 인덱스로 탐색이 빠르다.
- 단점
  1. 삽입,삭제가 느리다.
  2. 고정된 공간이라 메모리 효율성은 linked list보다 떨어진다.

3. 배열 회전

3-1. 저글링 알고리즘



배열 크기의 최대 공약수로 집합을 묶어서 회전시킨다.

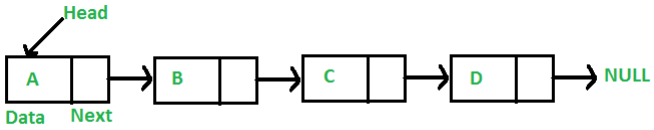
3-2. 역전 알고리즘

배열의 크기가 n이고 회전 횟수가 d일경우

(1,2,3,4,5)

1. d/n-d로 쪼갠다. : (1,2 / 3,4,5)
2. reverse(d), reverse(n-d) : (2,1/ 5,4,3)
3. merge : (2,1,5,4,3)
4. revers() : (3,4,5,1,2)

4. Linked List



- 장점
  1. 동적으로 생성할 수 있다.
  2. 삽입, 삭제가 편하다.
- 단점
  1. 배열의 인덱스 검색보다 느리다. (순차적 검색 필요)
  2. 포인터를 저장할 추가적인 공간이 필요하다.

5. Stack / Queue

- Stack : LIFO (Last In First Out)
- Queue : FIFO (First In First Out)

- Stack : DFS 탐색
- Queue : BFS 탐색

- Stack : push(), pop()
- Queue : enqueue(), dequeue()

- Stack : 함수 콜스택, 역순 출력
- Queue : 버퍼

5-1. Queue 구현

큐를 array로 구현하면 문제가 생긴다. 왜냐하면 front/rear(dequeue 위치/enqueue 위치)는 -1로 초기화가 된다.

만약 enqueue가 생기면 rear를 +1하고 값을 삽입한다.

dequeue가 생기면 front를 +1하고 값을 얻은뒤 그 공간을 삭제한다.

isEmpty : front=rear이면 비어졌다.

isFull : rear=queueSize-1이면 가득찼다.

하지만, front를 +1해왔기때문에 사실상 앞에 공간이 생겼음에도 불구하고 queue가 꽉찼다고 착각할 수 있다. 이를 개선한 것이 원형 큐이다.

5-2. 원형 Queue

논리적으로 배열의 처음과 끝이 같다고 생각한다.

front/rear의 초기값 = 0

front/rear를 배열 순환시키면서 탐색한다.

- 장점 :
    1. 메모리 공간 효율이 좋다.
- 단점 :
    1. 배열이기때문에 어쩔수없이 큐의 크기가 한정되어있다.

단점을 개선하기 위해서 연결리스트 큐를 사용한다. 공간의 제약이 없으며 삽입,삭제가 편리하다. (O(1))

6. B Tree

이진트리는 좌우 균형이 맞지 않는 경우 탐색에 비효율적일 수 있다. B Tree는 삽입과 삭제가 발생하면 트리의 균형을 조절한다.

이진트리와 달리 하나의 노드에 자식노드가 여러개일 수 있다.

관계형 데이터베이스에서 사용되는 가장 일반적인 인덱스이다.

6-1. B Tree 차수

B tree의 차수는 노드가 최대 가질 수 있는 자식의 개수이다. 차수가 n인 B tree에서 각 노드는 다음 조건을 만족시켜야한다.

- max child : n
- min child : (올림)(n/2)
- max key : n-1
- min key : (올림)(n/2)-1

6-2. B Tree 규칙

1. 노드의 차수가 n이라면, 그 노드의 자식의 수는 n+1이어야한다.
2. 각 노드의 자료는 정렬되어있어야한다.
3. 각 노드의 자료 D(K)는 왼쪽 서브트리는 D(k)보다 작아야하고 오른쪽 서브트리는 D(k)보다 커야한다.
4. root는 적어도 2개이상의 자식을 가져야한다.
5. root노드를 제외한 모든 노드는 적어도 m/2개의 자료를 가지고 있어야한다.
6. 말단 노드로 가는 경로의 길이는 모두 같다.
7. 입력자료는 중복될 수 없다.

6-3. B+ Tree 규칙

B Tree와 max, min의 조건은 같다. 하지만 B+ Tree는 외부 노드에 모든 데이터가 포함이 되어있어야한다는 점이 B Tree와 다른 점이다.

6-4. B+ Tree insertion

> https://www.youtube.com/watch?v=DqcZLulVJ0M

6-5. B+ Tree deletion

> https://www.youtube.com/watch?v=pGOdeCpuwpI

7. Hash

데이터를 해시함수를 거쳐 key(index)와 value(해시값)으로 변환하여 저장한다. 저장한 key,value 테이블을 해시테이블이라고한다.

버켓 : 인덱스 값(key)

엔트리 : 해시값 (vlaue)

- 장점 :
    - 해시함수를 이용하여 저장하면 나중에 데이터를 search할때 O(1)이라는 시간복잡도를 가지므로 매우 빠르다고 할 수 있다. (이진트리는 O(logn), 배열의 경우도 O(1)이지만 메모리를 할당해 두어야 하기 때문에 공간효율성이 떨어진다.)
    - 적은 리소스로 많은 데이터를 효율적으로 관리할 수 있다. 예를들어 작은 크기의 캐시 메모리에서도 프로세스를 관리할 수 있다.
- 단점 :
    - 해시값이 같을 경우 테이블 충돌일 날 수 있다.
    - 부하율이 커질 경우 검색,삽입,삭제 효율이 낮아진다. (자바 해시맵에서는 75%로 제한함)

7-1. Chaining

해시값 충돌을 방지하는 방법으로 체이닝 기법이있다. 이는 충돌이 일어나면 데이터를 포인터를 이용하여 리스트 형식으로 연결하는 것이다. 주기억장치에서 효율적인 방법이다.

- 단점 :

  최악의 경우 모든 데이터가 같은 해시값을 가지면 O(n)의 시간복잡도로 search가 된다.

  메모리가 낭비될 수 있다.

- JDK 라이브러리에 구현된 HashMap은 체이닝 방식을 사용하며 버킷의 길이에따라 Tree 형태로 변경된다. (슬롯이 8개 이하이면 연결리스트, 그 이상의 경우에는 트리 구조)

7-2. Open Addressing

해시 함수로 얻은 키값이 충돌이 날 경우 다른 공간에 데이터를 저장한다. 디스크에서 데이터 저장시 효율적인 방법이다.

7-3. 선형 탐사

충돌이나면 다음 빈칸을 찾아서 저장한다. 데이터가 한곳에 몰리는 데이터 클러스터에 취약하다. 이를 해결하는 방법은 빈 버킷을 일정한 간격으로 키우며 할당하는 방법과 이중 해싱으로 해시코드를 두번하여 저장하는 방식이 있다.

7-4. 제곱 탐사

정해진 고정 폭을 제곱수로 옮겨 해시값의 중복을 피하는 방법이다.

7-5. 배열과 해시의 차이

배열은 논리적인 저장 순서와 물리적인 저장 순서가 일치한다.

배열과 해시테이블 둘다 인덱스로 바로 데이터를 찾을 수 있다. (O(1)) 하지만, 배열의 경우에는 삽입,삭제의 경우가 조금 다르다.

배열은 삭제를 할 경우 O(1)의 시간이 걸리기는 하지만 데이터의 연속성이 깨진다. 그렇기 때문에 뒤에있는 데이터들을 shift해주어야 한다. 결국 O(n)의 시간이 걸린다.

또한, 메모리 주소가 연속되어야하기때문에 배열의 크기를 늘리는 것이 어려운일이다.

7-6. 자바에서의 해시

> https://devlog-wjdrbs96.tistory.com/88

자바에서는 객체의 주소를 가지고 해싱하여 힙영역에 객체를 생성한다.

- String str1 = new String("abc") : 힙메모리에 객체 저장
- String str2 = "abc" : 상수 풀에 저장

- Strign str3 = "abc" : 상수 풀에 저장

ex1)

- str1==str2 : False
- str2==str3 : True
- str1.equals(str2) : True
- str2.equals(str3) : True

이 이유는 ==는 주소값을 비교한다. str1만 힙 영역에 할당 되어있으며 str2와 str3은 같은 상수풀에 있는 "abc"를 가리키기 때문이다.

하지만, equals는 참조하고 있는 값의 동일 여부를 비교하기 때문에 True로 나오는 것이다.

ex2)

- StudentTest s1 = new StudentTest(2016, 100);
- StudentTest s2 = new StudentTest(2016, 100);
- s1==s2 : False
- s1.equals(s2) : False

== 는 False가 나오는 것이 당연하지만 왜 equals는 내용이 같은데도 False가 나오는가 ?

ex1의 equals는 String Class의 equals()메소드이지만 ex2의 equals는 Object Class의 equals()메소드이다.

Object Class의 equals()는 내부적으로 == 로 비교하기때문에 False가 나온다. String Class의 equals()는 오버라이딩하여 참조값이 같으면 true를 반환하도록 바뀌어져있다.

(자바의 해시코드란!!)

Object hashCode()는 메모리주소값을 이용하여 해시코드를 만들기 때문에 객체마다 다른 값을 가진다. 그렇기때문에 해시 코드값이 다르면 다른 객체로 판단한다. 만약 해시코드가 같으면 equals() 메소드로 다시 비교하며 두 값이 모두 동등해야 같은 객체로 인지한다.

8. DFS

깊이 탐색이다. 루트노드에서 시작하여 갈 수있는 edge를 모두 탐색하고 더이상 갈 수 있는 edge가없을 경우 backtracking하여 돌아와 탐색하는 과정이다. 스택과 재귀함수로 구현할 수 있다.

모든 경로를 방문해야하는 경우 적합한 방식이다. 또한, 경로에 특징을 가질때에는 DFS 방식이 좋다. 예를들어 노드에 숫자가 적혀있고 같은 숫자의 노드를 방문하면 안된다는 조건을 가질때, BFS로 할 경우에는 이전 방문 경로를 모두 각기 저장해야하기때문에 공간효율성이 떨어진다.

- 인접행렬 : O(V^2)
- 인접리스트 : O(V+E)

9. BFS

너비 탐색이다. root에서 가까운 노드 순서대로 방문하는 방법이다. 큐를 통해 구현한다.

최소 비용을 구할때 적합한 방식이다.

- 인접행렬 : O(V^2)
- 인접리스트 : O(V+E)

10. 인접행렬

모든 노드를 이차행렬로 구현하여 연결된 i노드가 j노드에 연결되어있을 경우 adj행렬의 i행j열의 값을 1로 바꾸어 연결표시를 한다.

- 장점
  - 구현이 쉽다.
  - 연결여부 확인에 O(1)시간을 갖는다.
- 단점
  - 노드는 많지만 브랜치가 적을 경우에는 공간 효율성이 낮아진다.
  - 하나의 노드에 연결된 모든 노드를 구하고 싶을 때에 O(V)라는 시간 복잡도를 갖는다.

> https://sarah950716.tistory.com/12

11. 인접리스트

벡터를 사용하여 각 노드마다 연결된 노드르 모두 linkedlist형식으로 연결하는 것이다.

- 장점
  - 모든 벡터 원소의 개수가 E의 수와 같다.
  - 하나의 노드에 연결된 모든 노드를 구하고 싶을때 효율이 좋다.
- 단점
  - 연결 여부를 확인할때 O(V)의 시간을 갖는다.

> https://sarah950716.tistory.com/12