

PART 5 재귀 알고리즘

5-1 재귀 알고리즘의 기본

5-1-1. 재귀

- 어떠한 이벤트에서 자기자신을 포함
- 다시 자기 자신을 사용하여 정의되는 경우

5-1-2. 팩토리얼

- $n!$ ($n =$ 양의 정수)
 - $0! = 1$
 - $\text{if } n > 0$
 - $n! = n \times (n-1)!$

• 재귀 호출

- '자기 자신과 똑같은 함수'를 호출

• 직접재귀와 간접재귀

- **직접재귀**
 - '자기 자신과 똑같은 함수'를 호출
- **간접재귀**
 - 다른 함수를 통해 자신과 똑같은 함수를 호출

5-1-3. 유clidean 호제법

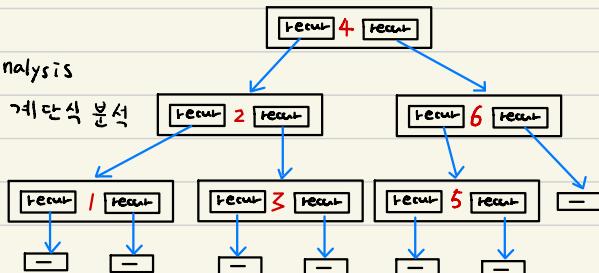
- 최대 공약수 - $\text{gcd}(x, y)$
 - $\text{if } y == 0$
 - $\text{return } x$
 - else**
 - $\text{return } (\text{gcd}(y, } x \% y))$

5-2 재귀 알고리즘 분석

5-2-1. 두 가지 분석 방법

- 하향식 분석 - **top-down analysis**

- 가장 위쪽부터 아래로 계단식 분석



· 상향식 분석 - bottom-up analysis

- 아래쪽 부터 쌓아 올리며 분석하는 방법

· ex) recur(n)

if $n > 0$:

 recur(n-1)

 print(n)

 recur(n-2)



$\Rightarrow \text{recur}(4) = ?$

$\text{recur}(-1)$: 아무것도 하지않음

$\text{recur}(0)$: 아무것도 하지않음

$\text{recur}(1)$: $\text{recur}(0) + \text{recur}(-1) \rightarrow 1$

$\text{recur}(2)$: $\text{recur}(1) + \text{recur}(0) \rightarrow 1$

$\text{recur}(3)$: $\text{recur}(2) + \text{recur}(1) \rightarrow 123 + 1$

$\therefore \text{recur}(4)$: $\text{recur}(3) + \text{recur}(2) \rightarrow 12314 + 12$

5-2-2 재귀 알고리즘의 비재귀적 표현

- 단점: 성능 저하가 심하고 메모리를 많이 소모한다

· 꼬리 재귀 제거

- 재귀 호출이 끝난 이후 현재 함수 블록 내에서 추가 연산을 할 필요가 없도록 구현하는 형태

· 재귀 제거

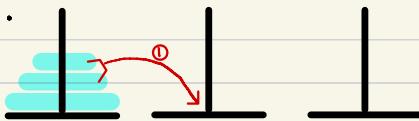
5-3 하노이의 탑

- 쌓아 놓은 원반을 최소 횟수로 옮기는 알고리즘

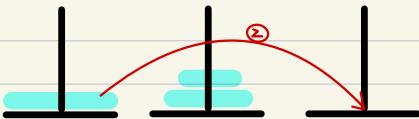
5-3-1. 하노이의 탑

- 작은 원반이 위, 큰원반인 아래에 위치하는 규칙을 유지하면서

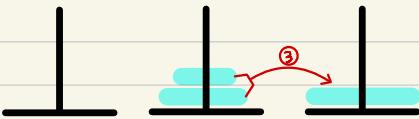
기둥 3개를 이용하여 원반을 옮기는 문제



1) 바닥에 있는 원반을 제외한 나머지 원반을
2번 기둥으로 옮김



2) 바닥에 있는 원반을 1기둥 → 3기둥으로 옮김



3) 2기둥에 있는 나머지 원반을 3기둥으로 옮김



5-4 8퀸 문제

5-4-1. 8퀸 문제

- 8개의 퀸이 서로 공격하여 잡을 수 없도록 8×8 체스판에 배치하세요.

⇒ 92 가지의 해

5-4-2. 퀸 배치하기

· 규칙 1

- 각 열에 퀸을 1개만 배치

· 규칙 2

- 각 행에 퀸을 1개만 배치

5-4-3. 분기 작업으로 규칙 1 해결하기

· 분기

- 나누는 작업

· 분기 작업

- 차례대로 가지가 뻗어 나가듯이 배치 작업을 열거하는 방법

· 분할정복법 (=분할 해결법)

- 1) 큰 문제를 작은 문제로 분할

- 2) 작은 문제 풀이법을 결합하여 전체 풀이법을 얻는 방법

- 주의 할점

- 2) 단계를 쉽게 할 수 있도록 문제를 분할

- ex)

· 하노이 탑 8퀸 문제

5-4-4. 한정작업과 분기 한정법 → 규칙 2 해결

· 한정작업

- 필요하지 않은 분기를 없애서 필요한 조합을 열거하지 않는 방법

- ex)

- 이전 열에 퀸이 있는 행은 제외

· 분기 한정법

- 분기 작업과 한정작업을 조합하여 문제를 풀이하는 방법

5-4-5. 8퀸 문제 해결하기

· 대각선 배치문제 ($/$, \backslash) 까지 고려하여 분기 한정법

- 92가지 조합

PART 6 정렬 알고리즘

6-1 정렬 알고리즘

6-1-1 정렬 이란?

- 항목간의 대소관계에 따라 데이터 집합을 일정한 순서로 바꾸어 늘어놓는 작업
- 오름차순 - *ascending*
 - 작은 \rightarrow 큰
- 내림차순 - *descending*
 - 큰 \rightarrow 작은
- 정렬 \leftarrow 안정적인 - 정렬후 값이 같은 원소의 순서유지
불안정적인 - 정렬후 값이 같은 원소의 순서유지 보장X
- 정렬 \leftarrow 내부정렬 - 하나의 배열에서 작업 가능
외부정렬 - 하나의 배열에서 작업 불가능
 - 내부정렬 응용

6-2 버블 정렬

- 단순 교환 정렬
- 이웃한 두 원소 비교후 필요에 따라 교환을 반복

6-2-1. 버블 정렬 알아보기

- 파스 (pass)
 - 일련의 비교교환 하는 과정
 - 원소가 n개일때, n-1번 수행
 - 원소가 n개일때, n-1번 수행

6-2-2. 알고리즘 개선

- 1) 파스에서 더 이상 교환이 이루어지지 않을 시
 \rightarrow 정렬을 완료 \rightarrow 중단 (break)
- 2) 이미 정렬된 원소를 제외한 나머지에만 파스 수행
 \rightarrow 이전 파스에서 마지막 교환한 원소들 중 오른쪽 원소까지 새로운 범위

6-2-3. 셰이커 정렬

- 홀수파스 : 가장 작은 원소 \rightarrow 맨 앞 \rightarrow 파스의 스캔방향을 앞·뒤로 바꿈
- 짝수파스 : 가장 큰 원소 \rightarrow 맨 뒤

6-3 단순 선택정렬

- 가장 작은 원소 부터 선택해 알맞은 위치로 옮기는 작업 반복

6-3-1. 단순 선택정렬 알아보기

- 교환 과정

- 1) 아직 정렬하지 않은 부분에서 가장 작은 원소 선택

- 2) 선택한 원소 \leftrightarrow 아직 정렬하지 않은 부분에서 가장 앞의 원소

- 비교횟수

$$-(n^2-n)/2$$

- 불안정적인 정렬

- 서로 이웃하지 않는 떨어져 있는 원소를 교환

6-4 단순 삽입정렬

- 주목한 원소보다 더 앞쪽에서 알맞은 위치로 삽입하여 정렬

6-4-1. 단순 삽입정렬 알아보기

- 정렬되지 않은 부분의 맨 앞원소를 정렬된 부분의 알맞은 위치에 삽입

6-4-2. 선택한 원소를 알맞은 위치에 삽입하는 과정

- 반복제어

- 종료조건 1. 정렬된 배열의 맨앞에 도달했을 경우

- or (종료조건 2. 선택한 원소보다 작거나 같은 한칸앞의 원소를 발견할 경우

- \rightarrow 계속조건 1. 정렬된 배열의 현재 인덱스 > 0

- and (계속조건 2. 한칸앞의 원소 $>$ 선택한 원소

6-4-3. 단순정렬 알고리즘의 시간 복잡도

- 버블, 선택, 삽입 $\rightarrow \bigcirc(n^2)$

- 효율이 좋지 않음

— 이진 삽입정렬

- 이미 정렬을 마친 배열을 제외하고 원소를 삽입해야 할 위치를 검사

6-5 셀 정렬

- 단순 삽입 정렬 보완

6-5-1. 단순 삽입정렬의 문제점

- 장점

- 정렬 완료된 또는 거의 끝나가는 상태에서 속도가 빠름

- 단점

- 삽입할 위치가 멀리 있으면 이동 횟수가 많아짐

6-5-2. 셀 정렬 알아보기

- 1) 배열의 원소들을 그룹으로 나눠 각 그룹별로 정렬을 수행

- 그룹 안의 원소의 개수는 단계별로 서로 배수가 되지 않도록 결정

- 2) 정렬된 그룹을 합침

- 1, 2) 작업을 반복

- 전체적으로 원소의 이동 횟수 감소

- 정렬 횟수는 늘어남

6-6 퀵 정렬

- 가장 빠른 정렬 알고리즘

- 널리 사용됨

6-6-1. 퀵 정렬 알아보기

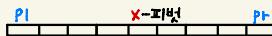
- 기준(피벗)에 의해서 그룹을 나눔

- 피벗 - 중심축

- 임의 선택 가능

- 선택된 피벗은 2개로 나눈 그룹 어디에 넣어도 상관X

6-6-2. 배열을 두 그룹으로 나누기



- $p_i \geq X$ 인 원소를 찾을 때 까지 오른쪽으로 스캔

- $p_i < X$ 인 원소를 찾을 때 까지 왼쪽으로 스캔

- 조건을 만족하는 원소 발견 시 정지 후 p_i 과 p_r 값 교환

- p_i 과 p_r 이 교차

- 그룹을 나누는 과정 끝



- 피벗 이하인 그룹 : $0, 1, \dots, p_i - 1$

- 피벗 이상인 그룹 : $p_i + 1, \dots, n-1$

- if 배열 나누기 후 $p_i > p_{i+1}$ 일 때,

- 피벗과 일치하는 그룹

6-6-3. 퀵정렬 만들기

- 분할정복 알고리즘
→ 재귀 호출 사용

- `qSort (배열, left, right)`
 - 배열을 두개의 그룹으로 나누는 함수
- `quick-sort (배열)`

`qSort (배열, 0, len(배열)-1)`

6-6-4. 비재귀적인 퀵 정렬 만들기

- 스택을 사용하여 구현
— `qSort (배열, left, right)`

• 스택 생성

— 크기: 배열의 크기

• 스택 푸시

— 나눌 범위의 맨 앞, 맨 끝 인덱스를 조합한 듀플 스택

• 반복으로 배열 나누기 — 스택이 비어있지 않은 동안

— 기존 스택에 있는 범위로 배열을 나눔

— If 나누기 전 맨 앞 < 나누기 후 맨 끝

 • 푸시(↓, ↓)

— If 나누기 후 맨 앞 < 나누기 전 맨 앞

 • 푸시(↓, ↓)

• 스택의 크기

— 배열을 스택에 푸시하는 순서에 따라서 다름

• 규칙 1

— 원소수가 많은 쪽의 그룹을 먼저 푸시

 • 스택에 쌓이는 데이터의 최대 개수 < $10\% n$

— 원소수가 적은 쪽의 그룹을 먼저 푸시

 • 스택에 넣고, 꺼내는 횟수는 같지만, 동시에 쌓이는 최대 데이터 개수는 다르다

 • 스택의 크기

• 규칙 2 > 규칙 1

6-6-5. 피벗 선택하기

- 전체에서 중앙값으로 하는게 이상적

— But, 중앙값을 구하기 위해 많은 계산시간이 걸림

→ 피벗을 선택하는 의미 X → 다른 방법 필요

방법 1.

발견

- I^f 원소수 ≥ 3

· 임의의 원소 3개를 꺼내 중앙값 → 피벗

방법 2.

— 1) 배열의 맨앞, 가운데, 맨끝 원소를 정렬

2) 가운데 원소와 맨끝-i 원소를 교환

3) 맨끝-i ⇒ 피벗, 맨앞+i ~ 맨끝-2 ⇒ 나눌 대상

6-6-6. 쿠적정렬의 시간복잡도

- $\bigcirc(n \log n)$

— 배열의 초기값 아 피벗을 선택하는 방법에 따라 달라짐

— 원소수가 적은 경우에는 빠른 알고리즘이 아님

6-7

병합정렬

- 배열을 앞 뒷 부분의 두 그룹으로 나누어 정렬 후 병합

6-7-1. 정렬을 마친 배열의 병합

- 각 배열에서 주목하는 원소의 값을 비교하여 작은쪽의 원소를 꺼내 새로운 배열에 저장

— 저장 후 두 배열 중 하나에 남은 원소를 새로운 배열에 저장

- 시간복잡도 — 병합

— $\bigcirc(n)$

6-7-2. 병합정렬 만들기

· 병합정렬

— 정렬을 마친 배열의 병합을 이용하여 분할정복법에 따라 정렬하는 알고리즘

- 시간복잡도 — 병합정렬

— $\bigcirc(n \log n)$

6-8 힙 정렬

- 선택정렬 응용

6-8-1. 힙 정렬 알아보기

• 힙

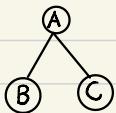
- 완전 이진트리

- 부모의 값 \geq 자식의 값
 - 부모의 값 \leq 자식의 값
- 두 조건 중 하나를 항상 만족

- 부분 순서 트리

- 형제간의 대소관계 일정 X

• 트리



- (A) - 루트, (B), (C)의 부모노드
- (B), (C) - (A)의 자식노드
- (B), (C)는 서로 형제노드

- 완전 이진트리

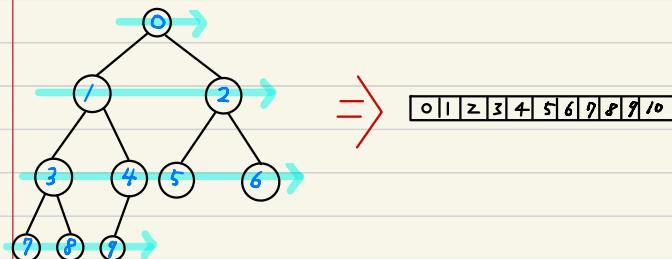
- 완전

- 부모는 왼쪽 자식부터 추가하여 모양유지

- 이진

- 자식의 최대개수 = 2

• 힙 \rightarrow 배열



- 자기 자신 $\alpha[i]$ 에 대해서

- 부모노드 - $\alpha[(i-1)//2]$

- 자식노드 ↗ 왼쪽 - $\alpha[i*2 + 1]$

오른쪽 - $\alpha[i*2 + 2]$

6-8-2. 힙 정렬의 특징

- '힙에서 최댓값 - 루트' 특성을 이용

— 1) 힙에서 최댓값인 루트를 꺼냄 → 새로운 정렬 배열의 맨 끝에 위치

— 2) 남은 부분을 힙으로 만듬

— 위의 과정을 반복

- 루트를 삭제한 힙의 재구성

— 1) 마지막 원소를 루트로 이동

— 2) 루트의 자식 노드 중 큰 값과 위치를 바꿈

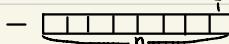
• 2) 과정을 반복

— 자식의 값이 각을 대까지

— 자식이 없는 리프가 될 때 까지

6-8-3. 힙 정렬 알아보기

- 힙으로 만든 배열 α



- 방법

— 1) $i = n-1$ 로 초기화

— 2) $\alpha[i] = \text{루트} \text{ 와 } \alpha[i] \text{ 교환} \rightarrow \text{힙의 최댓값} \Rightarrow \alpha[i]$

— 3) $\alpha[0] \sim \alpha[i-1]$ 로 힙 재구성

— 4) i 값을 1 감소시켜 0이 되면 종료

• 그렇지 않으면 2)로 돌아감

6-8-4. 배열 \rightarrow 힙

- 가장 아래 부분의 오른쪽 서브트리부터 상향식으로 힙 변환을 진행하여

전체 배열 \rightarrow 힙으로 변경

6-8-5. 힙 정렬의 시간 복잡도

- $\mathcal{O}(n \log n)$

— $\mathcal{O}(n)$ - 최댓값인 원소를 선택

$\mathcal{O}(\log n)$ - $\alpha[0] \sim \alpha[i-1] \rightarrow$ 힙 재구성

6-9 도수 정렬

- 분포수 세기 정렬
- 원소의 대소 관계 판단 X

6-9-1. 도수 정렬 알아보기

- 단계

- 1) 도수 분포표 만들기

- 배열 =

--	--	--	--	--	--

 $\rightarrow \min, \max$

→ 도수분포표

- 배열

- 크기: $\max - \min + 1$

- 도수분포표 [원소] + 1

2) 누적 도수분포표 만들기

- 누적 도수분포표 [i] = 누적 도수분포표 [i] + 도수분포표 [i-1]

3) 작업용 배열 만들기

- 배열 [i] = 작업용 배열 [누적 도수분포표 [배열[i]-1]]

4) 배열 복사하기

- 배열 [i] = 작업용 배열 [i]

PART 7 문자열 검색

- 어떤 문자열 안에 찾고자 하는 문자열이 포함되어 있는지 검사, 만약 포함되어 있다면 위치 반환

- 텍스트

- 검색되는 쪽의 문자열

- 파악

- 찾아내는 문자열

7-1 브루트 포스법

7-1-1. 브루트 포스법 알아보기

- 단순법

- 선형검색을 단순하게 확장