【2024 PNU SW스터디그룹 최종보고서]

[멘사 알고리즘 스터디]

[팀명]
[정보컴퓨터공학부/201925111] 김건호
[정보컴퓨터공학부/202155650] 윤소현
[정보컴퓨터공학부/202155525] 김문경
[정보컴퓨터공학부/201924515] 유승훈
[정보컴퓨터공학부/201924429] 김민혁
[정보컴퓨터공학부/202155655] 정진택
[정보컴퓨터공학부/202055614] 최성민

프로젝트 개요

1.1 프로젝트 배경

- 알고리즘 문제 해결 능력 향상을 위한 스터디

1.2 기존 문제점

- 개인 학습만으로는 다양한 접근 방식을 배우기 어려움
- 체계적인 알고리즘 학습의 필요성

1.3 프로젝트 환경

- 주 1회 정기 스터디 진행
- GitHub를 통한 코드 공유 및 리뷰
- 온라인 저지 사이트를 통한 문제 풀이

2. 프로젝트 활동 주요 학습 내용

- DFS/BFS
  - 완전 탐색
  - 어떻게 그래프를 잘 만들 것인가 + 어떤 탐색 선택이 관건
- BFS
  - edge의 가중치가 동일하다면 제일 빠르게 도달하는 것은 BFS
    - 대표적인 문제
      - https://www.acmicpc.net/problem/7569
      - https://www.acmicpc.net/problem/1600
- DFS
  - 재귀적/백트래킹으로 문제를 해결할 때
  - 순열, 조합, 분할정복 등 끝까지 탐색 후 결과를 도출하는 경우
    - 대표적인 문제
      - https://www.acmicpc.net/problem/9663
- 최단거리 알고리즘
  - 다익스트라 알고리즘 사용하는 문제가 많이 나왔음
  - O(ElogV)
  - 대표적인 문제
    - https://www.acmicpc.net/problem/1238
    - 정방향이 아닌 역방향으로 해결해야 하는 문제

- 동적 계획법
  - 많이 어려워했던 부분
  - 부분합 / 문자열 알고리즘
  - 메모제이션 기반으로 해결
  - 자료구조에 저장할 상탯값을 정의하는 것이 제일 중요함.
    - 상탯값을 저장했으면 어떤 값을 저장 할 것인지도 중요함.
  - 대부분 수학적 사고력을 요구하는 경우가 많음
    - 점화식을 잘 세워야 한다.
  - 대표적인 문제
    - https://www.acmicpc.net/problem/2133
    - https://www.acmicpc.net/problem/2096
- 최소신장트리 문제(MST)
  - **크루스칼 알고리즘**
    - 간선을 가중치 순으로 정렬한 후, 최소 가중치를 가진 간선을 하나씩 추가하며 사이클을 방지하는 방식
    - 유니온-파인드(Union-Find) 자료구조를 사용하여 사이클 발생 여부를 확인.
    - 간선이 많으면 크루스칼을 사용하는 것이 유리하다.
  - 프림 알고리즘
    - 특정 정점에서 시작하여, 인접한 간선 중 가중치가 최소인 간선을 선택하며 MST를 확장
    - 우선순위 큐를 사용하여 최소 가중치 간선 효율적 선택
    - 정점이 많으면 프림
  - 대표적인 문제
    - https://www.acmicpc.net/problem/1197
    - https://www.acmicpc.net/problem/1922
- 투포인터
  - 배열이나 리스트에서 두 개의 포인터를 사용하여 문제를 효율적으로 해결하는 방법
  - 보통 정렬된 배열에서 특정 조건을 만족하는 부분 배열이나 값을 찾는 데 사용

        - 두 개의 포인터를 시작점에 놓고 움직이며 조건을 만족시키는지 확인
        - 조건에 따라 포인터를 이동시켜 탐색 범위를 조정
- 유형
    - **부분합 찾기**
        - 특정 구간의 합이 주어진 값을 만족하는 경우 찾기
        - 하나의 포인터로 구간을 확장하고, 다른 포인터로 축소하며 조건을 맞춤
    - **특정 값 합 찾기**
        - 배열에서 두 값의 합이 특정 값을 만족하는 경우 찾기
        - 시작과 끝에 포인터를 두고 합에 따라 포인터 이동
- 그리디 알고리즘
    - 특정 상황에서 제일 좋은 것을 선택하는 문제
    - 최적 해를 보장하지 않기에, 최적 해를 구하는 문제에서 잘 사용해야 함
    - 대표적인 문제
        - https://www.acmicpc.net/problem/1339
        - https://www.acmicpc.net/problem/11000
- 이분 탐색
    - 정렬된 데이터에서 특정 조건을 만족하는 값 찾기
    - 최소 O(nlogn)의 시간 복잡도가 걸림
    - 대표적인 문제
        - https://www.acmicpc.net/problem/2805
- 비트마스킹
    - 많이 어려워했던 부분
    - 정수의 비트(bit)를 각 상태나 요소에 매핑
    - 알고리즘에서 비트마스킹
        - **상태 관리**
            - 다이나믹 프로그래밍에서 방문 상태 저장
                - 외판원 순회(tsp) 대표적 예시
            - **부분 집합 생성**

        - 모든 부분 집합을 탐색할 때 사용
    - **집합의 조합**
        - 특정 조건을 만족하는 집합 찾기
  - 대표적인 문제
    - https://www.acmicpc.net/problem/2064
    - https://www.acmicpc.net/problem/2098

3. 프로젝트 내용

- 스터디 진행 방식
  - 매주 다양한 알고리즘의 문제 선정
  - 각자 문제 풀이 후 코드 리뷰 진행
  - 최적화 방안과 다양한 접근 방식 토론
  - 실전 문제 풀이를 통한 응용력 향상

4. 프로젝트 결과 분석 및 평가

- 성과
  - 기본 알고리즘의 이해와 구현 능력 향상
  - 문제 해결을 위한 최적의 알고리즘 선택 능력 습득
  - 코드 최적화와 시간 복잡도 분석 능력 향상
  - 구성원 대부분 백준 ~골드3 난이도 해결 가능
- 한계점과 개선 사항
  - 고난도 문제에 대한 접근 방식 다양화 필요
  - 알고리즘 융합 문제 해결 능력 강화 필요
  - 실전 코딩 테스트 대비 시간 관리 능력 향상 필요
- 향후 목표
  - 고난도 문제 풀이 능력 향상
  - 실전 코딩 테스트 준비와 알고리즘 최적화 능력 강화.
  - 문제 풀이에서 더 많은 알고리즘을 융합하여 효율적인 해결 방법을 찾는 연습.

멘사 알고리즘 스터디 최종보고서

201925111 김건호

서론

- 중간 보고까진 동적 계획법 문제를 주로 풀었으나, 문제를 다양화해서 풀었음
- 그래프 탐색 문제와 최단 경로 알고리즘을 주로 학습하였음

주요 내용

- 최소신장트리 문제(MST)
  - **크루스칼 알고리즘**
    - 간선을 가중치 순으로 정렬한 후, 최소 가중치를 가진 간선을 하나씩 추가하며 사이클을 방지하는 방식
    - 유니온-파인드(Union-Find) 자료구조를 사용하여 사이클 발생 여부를 확인.
    - 간선이 많으면 크루스칼을 사용하는 것이 유리하다.
  - 프림 알고리즘
    - 특정 정점에서 시작하여, 인접한 간선 중 가중치가 최소인 간선을 선택하며 MST를 확장
    - 우선순위 큐를 사용하여 최소 가중치 간선 효율적 선택
    - 정점이 많으면 프림
  - 대표적인 문제
    - https://www.acmicpc.net/problem/1197
    - https://www.acmicpc.net/problem/1922
- 투포인터
  - 배열이나 리스트에서 두 개의 포인터를 사용하여 문제를 효율적으로 해결하는 방법
  - 보통 정렬된 배열에서 특정 조건을 만족하는 부분 배열이나 값을 찾는 데 사용
    - 두 개의 포인터를 시작점에 놓고 움직이며 조건을 만족시키는지 확인
    - 조건에 따라 포인터를 이동시켜 탐색 범위를 조정
- 유형

    - **부분합 찾기**
        - 특정 구간의 합이 주어진 값을 만족하는 경우 찾기
        - 하나의 포인터로 구간을 확장하고, 다른 포인터로 축소하며 조건을 맞춤
    - **특정 값 합 찾기**
        - 배열에서 두 값의 합이 특정 값을 만족하는 경우 찾기
        - 시작과 끝에 포인터를 두고 합에 따라 포인터 이동
- 그리디 알고리즘
    - 특정 상황에서 제일 좋은 것을 선택하는 문제
    - 최적 해를 보장하지 않기에, 최적 해를 구하는 문제에서 잘 사용해야 함
    - 대표적인 문제
        - https://www.acmicpc.net/problem/1339
        - https://www.acmicpc.net/problem/11000
- 이분 탐색
    - 정렬된 데이터에서 특정 조건을 만족하는 값 찾기
    - 최소 O(nlogn)의 시간 복잡도가 걸림
    - 대표적인 문제
        - https://www.acmicpc.net/problem/2805

결론

- 다양한 문제를 풀면서 넓은 시각을 갖게 되었음
- 또한, 문제 해설 및 생각 공유를 통해 고정된 생각을 탈피할 수 있었음
- 그래도 아직 갈 길이 멀다고 생각하기에 더 열심히 오래 스터디를 지속하려 함

멘사 알고리즘 스터디 최종 보고서

202155650 윤소현

1. 학습 개요:

1.1 학습 배경:

멘사 알고리즘 스터디는 정보컴퓨터공학부 학생 7명이 모여 일주일 동안 공부한 알고리즘에 대해 얘기하고, 문제 풀이에 대해 한 명씩 발표하며 알고리즘 문제 해결 능력을 향상시키는 스터디이다.

1.2 기존 문제점:

스터디를 만들기 전에는 혼자 공부하기에 동기가 부족하고, 공부를 계속해서 미룰 수 있다는 단점이 있었다. 이를 해결하기 위해 멘사 알고리즘 스터디를 만들어 매주 의무적으로 모여 공부한 내용을 나눔으로써 한 학기 동안 의지를 잃지 않고 알고리즘 공부를 할 수 있었다.

2. 학습 환경:

파이썬과 C++을 이용해서 알고리즘 문제를 풀고, 깃허브로 코드를 공유했다.

3. 주 차별 학습 내용:

1주 차) 복잡도

알고리즘 공부를 본격적으로 시작하기 전에, 알고리즘의 효율성을 평가하는 복잡도가 무엇인지 알아야 했다. 복잡도(complexity)란, 알고리즘의 성능을 나타내는 척도로 시간 복잡도(time complexity)와 공간 복잡도(space complexity)로 나눌 수 있다. 시간 복잡도는 특정한 크기의 입력에 대해 얼마의 알고리즘 수행 시간 걸리는지를 의미하고, 공간 복잡도는 특정한 크기의 입력에 대하여 알고리즘이 얼마나 많은 메모리 공간을 차지하는지를 의미한다. 시간 복잡도는 빅오 표기법을 사용한다. 빅오 표기법은 상수 계수나 낮은 차수 항은

생략하고 가장 높은 차수 항으로 표현한다. 예를 들어 T(n) = 3n + 2는 O(n)으로 표현된다. 공간 복잡도도 시간 복잡도와 마찬가지로 빅오 표기법을 사용한다.

```
array = [1, 2, 3]
sum = 0
for i in array:
   sum += i
print(sum)
```

위 코드에서 프로그램의 시간을 결정하는 것은 array 리스트의 크기(n)이다. 따라서 시간 복잡도는 O(n)이 된다.

```
array = [1, 2, 3]
for i in range(len(array)):
   for j in range(len(array)):
       print(f"Pair: ({array[i]}, {array[j]})")
```

이번엔 array 리스트 안의 원소로 만들 수 있는 2개의 숫자 쌍들을 모두 출력하는 코드를 살펴보자. 첫 번째 for 루프에서도 array 리스트 크기 n만큼 반복하고, 두 번째 for 루프에서도 n만큼 반복된다. 따라서 시간 복잡도는 O(n^2)이 된다.

1주 차) Union-Find 알고리즘

[1주 차 문제 1717 - 집합의 표현]

이 문제는 0, a, b 또는 1, a, b로 입력을 받는다. 첫 번째 입력으로 들어오는 숫자가 0인 경우에는 합집합(Union) 연산을, 1인 경우에는 a와 b 두 원소가 같은 집합에 포함되어 있는지 확인(Find)하는 연산을 한다. 이 문제를 해결하기 위해 Union-Find 알고리즘을 사용했다.

Union-Find 알고리즘은 이름 그대로 집합의 합집합(Union)과 Find 연산을 빠르게 처리하는 데 사용된다. Find 연산은 루트 노드에 도달할 때까지 parent 배열을 재귀적으로 호출하여 해당 원소가 속한 집합의 루트 노드를 반환한다. Union 연산은 말 그대로 두 원소가 속한 집합을 합친다. 각 집합은 트리로 표현된다.

Union-find 알고리즘에서 Tree의 깊이가 깊어질 경우에, O(N)의 시간복잡도를 가지게 된다. 이를 해결하여 Union-Find 알고리즘의 성능을 향상시키기 위해 경로 압축 기법을 사용할 수 있다. 경로 압축 기법은 Find 연산에서 트리의 깊이를 최소화하기 위해, Find 연산을 수행할 때 각 원소가 루트 노드를 가리키도록 parent를

업데이트하는 것이다. 즉, find한 값을 배열에 저장하고 값을 반환해 주는 것이다. 이를 통해 다음 find 연산에서 빠르게 루트 노드를 찾을 수 있게 된다. 약간의 코드 수정으로 시간 복잡도를 O(logN)로 줄일 수 있게 된다.

2-3, 5-6주 차) 그리디 알고리즘

그리디 알고리즘이란, 현재 단계에서 가장 최적이라고 판단되는 선택을 반복적으로 수행하여 해답을 구하는 알고리즘이다. 현재 단계에서 가장 최적인 선택을 하기 때문에 현재의 선택이 나중에 어떤 영향을 미칠지는 고려하지 않는다. 따라서 최종으로 구해진 해답이 항상 최적의 해가 된다고 보장할 수는 없으며, 그리디 선택 속성과 최적 부분 구조와 같은 문제에서만 최적해를 보장한다. 그리디 알고리즘은 최소 스패닝 트리(MST)를 찾는 문제(크루스칼, 프림 알고리즘) 또는 최단 경로 구하기(다익스트라 알고리즘) 문제 등에서 사용된다. 2주 차에 1946, 3주 차에 2170, 5주 차에 11501, 6주 차에 11000번을 풀었다.

[2주 차 문제 1946 - 신입사원]

이 문제는 다른 모든 지원자와 비교했을 때 서류 심사 성적과 면접 시험 성적 중 적어도 하나가 다른 지원자보다 떨어지지 않는 사람만 채용한다는 조건 하에서 신규 사원 채용에서 선발할 수 있는 최대 인원의 수를 구하는 문제이다. 이 문제를 그리디 알고리즘으로 해결할 수 있는 이유는, 서류 성적을 기준으로 오름차순으로 정렬하여 면접 성적을 기준으로 이전에 합격한 사람의 면접 순위보다 높은 순위의 사람만 합격자로 추가해 주면 되기 때문이다.

[3주 차 문제 2170 - 선 긋기]

이 문제는 자의 한 점에서 다른 한 점까지 선을 긋는데, 이미 선이 있는 위치에 겹쳐서 그릴 경우에는 차이를 구별하지 않고 여러 선을 그었을 때, 그려진 선들의 총 길이를 구하는 문제이다. 선의 시작점을 기준으로 먼저 정렬하고, 선들이 겹칠 때는 끝점이 확장된다면 이 값으로 갱신한다. 만약 겹치지 않는다면 이전 선의 길이를 더하고 새로운 선을 기준선으로 갱신하는 식으로 그리디 전략을 사용하여 문제를 풀 수 있다.

[5주 차 문제 11501 - 주식]

이 문제는 날 별로 주식의 가격을 알려주었을 때, 최대 이익이 얼마가 되는지 계산하는 문제이다. 주식 가격을 내림차순으로 정렬하고 현재 상태의 주식 가격 리스트에서 가장 큰 주식의 가격을 찾아 해당 주식을 팔아 최대 이익을 얻는다. 따라서 그리디 알고리즘을 적용하여 문제를 해결할 수 있었다.

[6주 차 문제 11000 - 강의실 배정]

이 문제는 주어진 N개의 수업을 위해 최소 강의실을 사용하기 위해 강의실을 배정하는 문제이다. 이 문제에서는 가장 빨리 종료되는 강의실을 먼저 사용하는 그리디 전략을 통해 현재 강의의 시작 시간이 가장 빨리 끝나는 강의실의 종료 시간보다 늦다면 그 강의실을 재사용할 수 있도록 한다. 현재 강의의 시작 시간이 가장 빨리 끝나는 강의실의 종료 시간보다 빠르면 새로운 강의실을 배정받아야 하므로 해당 강의의 종료 시간을 최소 힙에 추가했다. 최종적으로 최소 힙의 길이가 사용된 강의실의 수가 된다. 여기서 최소힙을 사용한 이유는 가장 빨리 끝나는 강의실을 쉽게 찾을 수 있게 하기 위해서이다.

[후기]

4주에 걸쳐 그리디 알고리즘 문제를 풀어보면서, 그리디 알고리즘으로 풀 수 있는 유형을 빠르게 식별하는 방법을 터득할 수 있었다. 문제에서 구해야 하는 답이 단계별로 최적의 선택을 해야하는 경우에는 먼저 그리디 알고리즘을 떠올려 보는 것이 중요하다는 것을 느꼈고, 앞으로 알고리즘 문제를 풀 때에는 문제를 빠르게 분석하여 어떤 알고리즘을 적용하면 좋을지 판단하는 능력을 더 키워야겠다는 생각이 들었다.

4, 8주 차) Dynamic Programming

DP(= 다이나믹 프로그래밍)는 기본적으로 하나의 큰 문제를 여러 개의 작은 문제로 나누어 그 결과를 저장하여 다시 큰 문제를 해결할 때 사용한다. 일반적으로 DP는 재귀와 유사하지만, 재귀를 사용할 때 동일한 작은 문제들이 여러번 반복되어 비효율적인 계산이 되는 것을 막아준다. 피보나치 수열을 예로 들 수 있는데, 피보나치 수열을 구할 때 f(n) = f(n-1) + f(n-2)로 호출하면 동일한 값을 2번씩 구하게 되므로 연산량이 매우 늘어난다. 이 때 DP를 사용하여 한 번 구한 작은 문제의 결과 값들을 저장해 두고 재사용할 수 있어 계산된 값을 다시 계산할 필요가 없어진다. DP로 문제에서는 점화식을 세우는 것이 핵심이다. 4주 차에는 백준 떡 먹는 호랑이 문제(2502)를 풀었다. 8주 차에는 백준 평범한 배낭 문제(12865)를 풀었다.

[4주 차 - 2502 떡 먹는 호랑이]

문제에서는 하루에 한 번 산을 넘어가는 떡 장사 할머니가 호랑이에게 어제 받은 떡의 개수와 그저께 받은 떡의 개수를 더한 만큼의 떡을 줘야만 산을 무사히 넘어갈 수 있다. 할머니가 넘어온 날(D)과 그날 호랑이에게 준 떡의 개수(K)가 주어졌을 때, 첫날에 준 떡의 개수와 둘째 날에 준 떡의 개수를 구하는 문제이다.

```
for i in range(1, D) :
   coeff.append([coeff[i - 1][0] + coeff[i][0], coeff[i - 1][1] + coeff[i][1]])
```

위와 같은 점화식을 사용해 coeff[i][0]과 coeff[i][1] 각각에 i 번째 날에서 준 떡의 값을 첫 번째 날과 두 번째 날에 준 떡의 개수의 계수를 저장하도록 한다.

[8주 차 - 12865 평범한 배낭 문제]

이 문제에는 물건의 개수(N)와 배낭의 최대 무게(K), 각 물건의 무게와 가치가 주어진다. 이 때, 배낭에 넣을 수 있는 물건들의 최대 가치의 합을 구해야 한다. 다이나믹 프로그래밍으로 풀 수 있는 대표적인 예제인 배낭 문제를 연습하기 위해 이 문제를 선택해 보았다.

```
for w, v in score.items():
       if input_W + w <= K and input_V + v > score.get(input_W + w, 0):
           tmp[input_W + w] = input_V + v
   score.update(tmp)
```

배낭 문제에서 사용되는 점화식은 위와 같다. 새로운 물건의 무게 input_W와 가치 input_V를 선택하여 기존 상태 (w, v)에 추가한다. 이때, 새로운 무게가 배낭의 최대 무게 K를 초과하지 않고, 해당 무게에서 가치가 기존 가치보다 더 크면 상태를 갱신한다. 이러한 과정을 통해 모든 가능한 경우를 탐색하여 최종적으로 배낭에 넣을 수 있는 물건들의 최대 가치를 출력하도록 할 수 있다.

[후기]

다이나믹 프로그래밍 문제를 해결하기 위해서는 큰 문제를 어떻게 작은 하위 문제로 나눌 것인지, 어떤 값을 dp 테이블에 저장할 것인지, 점화식은 어떻게 도출해야 할 것인지 결정하는 것이 핵심이다. 이 과정에 익숙해지기 위해서는 조금 더 여러 문제를 풀어 보면서 반복적인 연습이 필요하다는 것을 느꼈다.

7주 차) 이분 탐색 / 매개변수 탐색

이진 탐색(이분 탐색) 알고리즘은 정렬되어 있는 리스트에서 탐색 범위를 절반씩 좁혀가며 데이터를 탐색하는 방법이다. 반드시 리스트의 내부 데이터가 정렬되어 있어야 사용할 수 있다. 이분 탐색은 start, end, mid 3개의 변수를 사용하여 탐색을 진행한다. 정렬된 리스트에서 특정 값을 찾을 때 정중앙에 위치한 값을 활용하여 빠른 속도로 탐색을 끝낸다.

매개변수 탐색은 이진 탐색과 상당히 유사한 알고리즘으로, 답이 이미 결정되어 있다고 보고 문제를 푸는 “결정 문제”로 풀 수 있는 문제에서 사용할 수 있다. 추가로, 어떤 시점까지는 조건을 만족하지만, 그 시점 이후로는 조건을 만족하지 않는 경우에 최댓값을 찾거나 최소값을 찾는 문제에도 사용된다.

7주 차에서는 백준 2343번 기타 레슨 문제를 풀었다.

[7주 차 문제 - 2343 기타 레슨]

이 문제에서는 기타 강의의 길이가 강의 순서대로 분 단위로 주어질 때, 녹화 가능한 블루레이의 크기 중 최소를 출력해야 한다.

```
if blu_ray_counting <= blu_ray:
    end = mid - 1
else:
    start = mid + 1
```

이분 탐색을 진행하고 있는 부분이다.

[후기]

이 문제는 풀이에 사용하고 있는 이분 탐색뿐만 아니라, 매개변수 탐색으로도 풀 수 있다고 한다. 이분 탐색 문제에는 이미 익숙해져 있는 상태였지만, 매개변수 탐색은 처음 들어보는 알고리즘 유형이었다. 이 문제를 풀어봄으로써 이분 탐색 알고리즘을 복습하고, 매개변수 탐색이라는 새로운 알고리즘을 학습할 수 있었다.

9, 10주 차) BFS / DFS

BFS는 너비 우선 탐색 알고리즘으로, 큐를 이용하여 구현할 수 있다. BFS는 시작 노드에서부터 인접한 노드를 모두 탐색한 후, 다음 노드로 이동한다. BFS는 두 노드 사이의 최단 경로 혹은 임의의 경로를 찾고 싶을 때 이 방법을 선택한다.

DFS는 깊이 우선 탐색 알고리즘으로, 스택을 이용하여 구현한다. DFS는 탐색 시작 노드를 스택에 삽입하고, 방문 처리한다. 스택의 최상단 노드에 방문하지 않은 인접한 노드가 하나라도 있으면 그 노드를 스택에 넣고 방문 처리한다. 방문하지 않은 인접 노드가 없으면 스택 최상단 노드를 꺼낸다. 더이상 이 과정을 수행할 수 없을 때까지 반복하여 그래프를 탐색한다. DFS는 트리 순회 등의 문제를 해결하기 위해 사용한다.

9주 차에선 백준 2606, 10주 차에선 2178번 문제를 풀었다.

[9주 차 - 2606 바이러스]

이 문제는 여러 대의 컴퓨터가 네트워크 상으로 서로 연결되어 있을 때, 1번 컴퓨터가 바이러스에 걸릴 경우 해당 컴퓨터에 의해 바이러스에 걸리게 되는 컴퓨터의 수를 구하는 문제이다. 1번 컴퓨터에서 부터 시작하여 연결된 모든 컴퓨터의 수를 출력함으로써 풀 수 있는 문제라고 생각이 들었고 깊이 우선 탐색 알고리즘을 사용하여 쉽게 풀 수 있었다.

[10주 차 - 2178 미로 탐색]

이 문제는 (N, M)과 미로가 입력될 때, (1, 1) 에서 미로를 통과하여 (N, M)으로 가기 위해 지나야 하는 최소 칸의 수를 구해야 하는 문제이다. 가중치가 없는 그래프에서 최단 경로를 구하는 경우에는 너비 우선 탐색을 사용하여 해결할 수 있다는 것을 알게되었고, BFS를 사용해서 미로를 통과하는 알고리즘을 구현하여 문제를 풀 수 있었다.

[후기]

깊이 우선 탐색과 너비 우선 탐색 문제를 풀어보면서 두 알고리즘의 차이를 명확하게 알 수 있었고, 어떤 문제 상황에서 어떤 알고리즘을 선택하여 풀면 좋은지 학습할 수 있었다. 깊이 우선 탐색의 경우 모든 경로를 탐색하게 되므로 경로가 많아지면 공간과 시간이 급증할 수 있기 때문에 종료 조건을 적절히 설정하거나, 입력의 크기를 유의해야 할 것 같다. 너비 우선 탐색도 큐에 노드를 넣고 꺼내는 방식으로 동작하게 되므로, 모든 경로를 탐색할 필요가 없는 문제에서 불필요한 탐색을 줄이기 위해 유의해야겠다는 생각이 들었다.

11주 차) 다익스트라

다익스트라 알고리즘은 대표적인 최단 경로 탐색 알고리즘이다. 다익스트라 알고리즘을 사용하여 최단 경로를 탐색할 때에는 음의 가중치를 가지는 간선을 포함할 수 없다. 모든 간선의 가중치가 음이 아닐 때, 한 정점에서 다른 모든 정점으로 가는 최단 경로를 알려준다. 다익스트라 알고리즘의 동작은 다음과 같다.

출발 정점과 도착 정점을 선택하고, 최단 거리를 저장할 배열을 만들어 초기화해 둔다. 이때, 배열의 값을 계속해서 최단 경로로 갱신할 것이기 때문에 초기값은 무한대로 설정해 두고, 출발 정점의 거리는 0으로 설정한다. 추가로 정점의 방문 여부를 구별할 배열도 하나 준비해 둔다.

그리고 아래와 같은 탐색 과정을 반복해야 한다.

현재까지 최단 거리 배열 중에서 방문하지 않은 정점들 중 최단 거리 값이 가장 작은 정점을 선택하여 방문 처리한다. 선택된 정점과 인접한 정점들에 대해 간선을 따라 이동했을 때의 거리(=현재 정점의 최단 거리 + 간선의 가중치)를 계산한다. 계산된 거리와 인접한 정점의 기존 최단 거리 값을 비교하여, 더 작은 값으로 최단 거리 배열을 갱신한다. 방문하지 않은 정점이 없거나, 도착 정점까지 최단 경로가 확보되면 반복을 종료한다. 이때 최단 거리 배열에 저장된 값들이 시작 정점에서 각 정점까지 도달할 수 있는 최단 거리이다.

11주 차에는 백준 1238 문제를 풀었다.

[11주 차 - 1238 파티] 이 문제에서는 N개의 숫자로 구분된 각각의 마을에 한 명의 학생들이 살고 있다. 총 M개의 단방향 도로가 있고, i 번째 길을 지나는데 T_i의 시간이 소비되는 각 마을이 있을 때, N명의 학생이 파티에 참석하기 위해 파티 장소 X 마을에 갔다가 다시 그들의 마을로 돌아가는데 가장 많은 시간을 소비하는 학생이 누구인지 구하는 문제이다. 따라서 각 학생이 살고있는 마을에서 출발하여 파티가 열리는 X 마을까지의 최단 경로를 구하기 위해 학생별로 다익스트라를 실행하고, 각 학생이 파티에 참석하고 돌아오는 시간을 추가로 계산했다.

```
def dijkstra(start):
    queue = []
    heapq.heappush(queue, (0, start))
    distance[start] = 0
    while queue:
        dist, now = heapq.heappop(queue)
        if distance[now] < dist:
            continue
        for next in graph[now]:
            cost = dist + next[1]
            if cost < distance[next[0]]:
                distance[next[0]] = cost
                heapq.heappush(queue, (cost, next[0]))
    return distance
```

다익스트라 알고리즘은 위와 같다.

```
for i in range(1, students + 1):
    distance = [INF] * (students + 1)
    result.append(dijkstra(i))
for i in range(1, students + 1):
    cost_lst.append(result[i][time] + result[time][i])
```

학생별로 다익스트라를 실행하고, 마을 X 까지 갔다가 다시 돌아오는 비용을 더하고 있다.

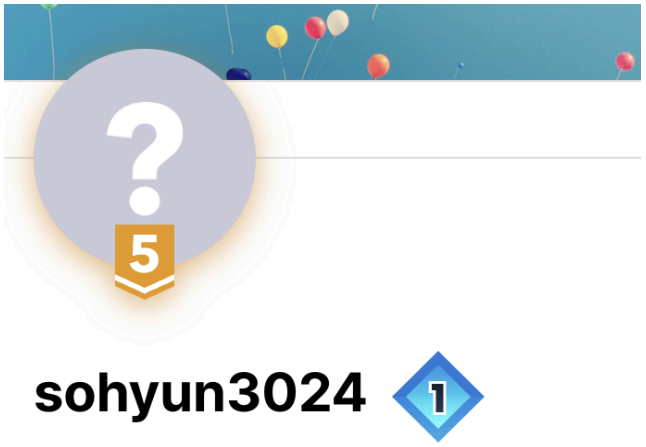
[후기]

학생의 수만큼 다익스트라를 실행하는 알고리즘으로 문제를 해결하긴 했으나, 학생의 수가 더 많아지면 해당 풀이로는 문제를 주어진 시간 안에 해결할 수 없을 것 같다는 생각이 든다. 조금 더 효율적으로 해결할 수 있는 방법을 추가적으로 생각해 보아야 할 필요성이 있음을 느꼈다. 다익스트라 알고리즘에 대해 이론적으로 알고 있었지만, 직접 구현해 보면서 다익스트라 알고리즘의 동작 방식을 더 잘 이해할 수 있게 되었다.

4. 알고리즘 스터디 활동 결과:

매주 조원들과 함께 한 주 동안 공부한 알고리즘에 대해 논의하고, 각자의 풀이법을 공유하며 학습했다. 이를 통해 여러 알고리즘에 대한 다양한 접근법을 익힐 수 있었다.

한 학기 동안 조원들과 공부 내용을 꾸준히 공유하고 함께 학습한 덕분에 지속적으로 알고리즘 공부를 이어갈 수 있었고, 백준에서 골드 5를 달성할 수 있었다.



알고리즘 스터디 최종 보고서

202155525 김문경

---

1. 어떤 파트를 어떻게 공부했는지

스터디 진행 방식

- **플랫폼**: 노션을 활용해 문제 풀이 기록
- **목표**: 매주 최소 1 문제 이상 풀기
- **형식**:
    - 문제 풀이 후 코드를 공유
    - 관련 알고리즘 공부 및 발표 (5~10 분)
    - 발표 후 팀원들과 토의 진행

학습 방식

스터디를 통해 주로 동적 계획법(DP), 깊이 우선 탐색(DFS), 너비 우선 탐색(BFS)와 같은 알고리즘 문제 풀이에 집중했습니다. 특히 개인적으로 부족하다고 느꼈던 DP 영역을 보완하기 위해 다양한 유형의 문제를 시도하며 실력을 키웠습니다. 또한, 스터디에서는 각자 문제 풀이를 준비해 발표했는데, 다른 스터디 구성원들의 발표를 통해 새로운 접근법이나 알고리즘 활용 방법을 배울 수 있었습니다. 발표와 토론을 통해 기존에 알지 못했던 아이디어를 얻고, 알고리즘에 대한 이해도를 더 넓힐 수 있었습니다.

학교 수업과의 연계

스터디 활동을 하면서 학교에서 듣고 있는 알고리즘 수업과 많은 연관성을 느꼈습니다. 수업에서 배운 개념과 기법들을 스터디에서 복습하며 실제 문제에 적용해볼 수 있는 기회가 되어 정말 좋았습니다. 덕분에 수업에서 배운 내용이 더욱 명확하게 정리되었고, 많은 도움이 되었습니다. 또한 다양한 유형의 문제를 접하면서 단순히 알고리즘을 외우는 것을 넘어 문제 해결을 위한 응용 능력을 키울 수 있었습니다.

---

2. 공부한 내용 : 문제 요약 및 알고리즘 정리

1. **백준 2240 번 - 자두나무 (DP)**

문제 요약:
이 문제는 자두가 1 번 혹은 2 번 나무에서 떨어지는 상황에서 제한된 이동 횟수 내에 최대 몇 개의 자두를 받을 수 있는지 계산하는 문제입니다.

알고리즘:
문제를 해결하기 위해 동적 계획법(DP)을 활용했습니다. DP 테이블을 `dp[t][w]`로 정의했으며, 이는 `t` 초에 `w` 번 이동한 후 받을 수 있는 자두의 최대 개수를 의미합니다. 점화식은 두 가지로 나뉘는데, 첫 번째는 현재 위치를 유지하는 경우 `dp[t][w] = dp[t-1][w] + (`현재 위치 자두 여부`)`이며, 두 번째는 이동한 경우로 `dp[t][w] = dp[t-1][w-1] + (`이동 후 위치 자두 여부`)` 입니다.

느낀 점:
DP 의 상태 정의가 문제 해결의 핵심임을 다시 깨달음. 초기화와 테이블의 정확한 해석이 중요.

2. 백준 5052 번 - 전화번호 목록 (트라이 자료구조)

문제 요약:
전화번호 목록에서 어떤 번호가 다른 번호의 접두사가 되는지 확인하는 문제입니다.

알고리즘:
트라이(Trie) 자료구조를 사용했습니다. 전화번호를 트라이에 삽입하면서, 삽입 도중 접두사 조건이 위배되면 과정을 종료하는 방식으로 구현했습니다.

느낀 점:
문자열 문제를 해결할 때 트라이 자료구조가 접두사 문제에 적합하다는 것을 경험했습니다.

-

3. 백준 16928 번 - 뱀과 사다리 게임 (BFS)

문제 요약:
뱀과 사다리 게임은 주사위를 던져 가장 빠르게 100 번 칸에 도달하는 최소 이동 횟수를 구하는 문제입니다. 사다리는 플레이어를 앞으로 보내고, 뱀은 뒤로 보내는 역할을 합니다.

알고리즘:
이 문제는 BFS(너비 우선 탐색)를 활용해 해결했습니다. 주사위를 던진 결과를 큐에 추가하고, 방문 배열을 사용해 이미 방문한 칸을 체크하여 중복 계산을 방지했습니다.

느낀 점:
BFS 가 최단 거리 문제에 적합하다는 점과, 게임 규칙에 맞게 그래프 형태를 구성하는 것이 중요하다는 점을 배울 수 있었습니다.

---

3. 배울 수 있었던 점

다양한 알고리즘을 문제에 맞게 선택하고 적용하는 능력을 키울 수 있었습니다. 동적 계획법(DP), 트라이(Trie), 그리고 너비 우선 탐색(BFS)과 같은 알고리즘을 사용하여 각 문제를 해결하면서 알고리즘 선택의 중요성과 효율적인 적용 방법을 배웠습니다. 또한, 알고리즘 최적화의 중요성에 대해서도 깊이 고민하게 되었습니다. 문제를 해결하는 과정에서 불필요한 계산을 줄이고, 더 효율적인 방법으로 접근할 수 있는 방안을 찾으며, 실행 속도와 자원 사용을 크게 개선할 수 있음을 경험했습니다.
특히, 발표를 하면서 다른 사람들의 피드백을 통해 나도 모르게 놓쳤던 다양한 접근법이나 최적화 방법을 배우는 기회가 되어 매우 유익했습니다. 각자가 준비한 문제가 달라서 늘 비슷한 유형의 문제를 풀었던 저에게는 다양한 문제를 보고 듣는 경험이 정말 귀중한 경험이 되었습니다.

---

4. 보완점

이번 학습을 통해 보완해야 할 점은 리드미 작성 습관화와 꾸준한 문제 풀기입니다. 다른 팀원들에 비해 높은 수준의 문제를 많이 풀어보지 못한 점이 아쉬움으로 남습니다. 더 많은 문제를 풀어보며 문제 해결 능력을 기르고 싶습니다. 앞으로는 다양한 알고리즘을 경험하고, 기존에 익숙한 알고리즘 외에도 새로운 알고리즘을 공부해 안목을 넓힐 계획입니다. 이를 통해 문제 해결 능력을 더욱 향상시키고, 리드미 작성 습관을 통해 학습 내용을 체계적으로 정리하면서 지속적으로 발전해 나가겠습니다.

알고리즘 스터디 프로젝트 최종 보고서

201924515 유승훈

1. 프로젝트 개요

진행기간 동안 프로그래머스와 백준, 그리고 LeetCode 문항들을 중점적으로 학습하였습니다.

프로그래머스와 백준에서는 단순 구현부터, 그래프 탐색 문제들을 위주로 학습하였으며, LeetCode 플랫폼을 통해선 알고리즘의 기초적인 자료구조나 방법론을 다루는 문제들 위주로 풀었습니다.

2. 프로젝트 환경

사용언어는 Python3으로, 파이썬 친화적인 Pycharm을 사용하였습니다. 또한, 리트코드나 프로그래머스와 같은 문제를 풀 때는 해당 플랫폼에서 제공하는 웹 IDE를 사용하였습니다. 푼 문제를 깃허브 이슈화하고, 해당 문제에 대한 Pull Request를 올리는 방식으로 학습을 진행했습니다.

3. 프로젝트 내용

먼저 기초적인 자료구조나 방법론을 공부하였습니다. 자세하게는 Hashing, Two pointer, Tree, BFS, DFS에 대해서 학습하였고, 각각 Leetcode에서 해당 자료구조나 방법론과 연계된 문제들을 풀며 기초를 닦았습니다.

이후 백준에서 제공하는 클래스 3, 4 문제를 풀며 모든 유형의 문제에 대해서 풀며 유형화된 학습을 하였고, 유형 중 가장 어려웠던 Dynamic Programming 문제들을 추가적으로 풀이하였습니다.

프로젝트 진행 기간 중, 삼성 코딩테스트를 볼 기회가 주어져, 코드트리라는 플랫폼에서 삼성전자 소프트웨어 직무 기출문제 5문제를 해결하기도 하였습니다. 주로 시뮬레이션과 그래프 탐색이 결합된 유형으로, 호흡이 긴 문제를 풀며 해결법에 대해 설계하고 깊게 사고하는 능력을 길렀습니다.

4. 프로젝트 결과 분석 및 평가

기초부터 차근차근 문제를 풀고, 그 문제를 풀면서 얻은 지식과 방법론을 블로그에 정리하면서 코딩테스트에 대비하는 방법을 터득하는 시간이 되었습니다. 또한 그런 방법론을 실제 삼성 코딩테스트 시험장에 가서도 사용해보면서, 실전적 감각을 익힐 수 있었습니다.

앞으로 더 많은 기업들의 코딩테스트에 참여해보며, 실전 감각을 유지할 예정입니다.

201924429 김민혁

1. 프로젝트 개요

1.1 프로젝트 배경

- 알고리즘 문제 해결 능력 향상을 위한 스터디 구성

1.2 기존 문제점

- 개인 학습만으로는 다양한 접근 방식을 배우기 어려움
- 체계적인 알고리즘 학습의 필요성

1.3 프로젝트 환경

- 주 1회 정기 스터디 진행
- GitHub를 통한 코드 공유 및 리뷰
- 온라인 저지 사이트를 통한 문제 풀이

2. 프로젝트 활동 주요 학습 내용:

1. 그래프 탐색 (DFS/BFS)
    - 깊이/너비 우선 탐색의 구현과 활용
    - 최단 경로 탐색과 연결 요소 확인 문제 해결
    - 백트래킹을 활용한 완전 탐색 구현
2. 최단 경로 알고리즘
    - 다익스트라 알고리즘 구현 및 활용
    - 플로이드-워셜 알고리즘을 통한 모든 정점 간 최단 거리 계산
3. 동적 계획법
    - Top-down과 Bottom-up 접근 방식의 이해
    - Memoization 기법을 통한 최적화
    - LIS, LCS 등 다양한 DP 문제 패턴 학습
4. 그리디 알고리즘
    - 최적해를 보장하는 조건 학습

        - 회의실 배정, 동전 거스름돈 등 실생활 문제 해결
    5. 투 포인터
        - 배열에서의 구간 합과 부분 수열 탐색 구현
        - 슬라이딩 윈도우 기법과의 차이점 이해
    6. 시뮬레이션
        - 주어진 문제 조건을 정확히 구현하는 능력 향상
        - 복잡한 구현 문제에서의 예외 처리 방법 학습
3. 프로젝트 내용
    - 스터디 진행 방식
        - 매주 다양한 알고리즘의 문제 선정
        - 각자 문제 풀이 후 코드 리뷰 진행
        - 최적화 방안과 다양한 접근 방식 토론
        - 실전 문제 풀이를 통한 응용력 향상
4. 프로젝트 결과 분석 및 평가
    - 성과
        - 기본 알고리즘의 이해와 구현 능력 향상
        - 문제 해결을 위한 최적의 알고리즘 선택 능력 습득
        - 코드 최적화와 시간 복잡도 분석 능력 향상
    - 한계점과 개선 사항
        - 고난도 문제에 대한 접근 방식 다양화 필요
        - 알고리즘 융합 문제 해결 능력 강화 필요
        - 실전 코딩 테스트 대비 시간 관리 능력 향상 필요
    - 향후 목표
        - 고난도 문제 풀이 능력 향상.
        - 실전 코딩 테스트 준비와 알고리즘 최적화 능력 강화.
        - 문제 풀이에서 더 많은 알고리즘을 융합하여 효율적인 해결 방법을 찾는

연습.

202155655 정진택

1. 학습 내용

1.1 그래프 탐색 (DFS, BFS)

- 학습 방법
  그래프 탐색 알고리즘인 **DFS(깊이 우선 탐색)**와 **BFS(너비 우선 탐색)**를 먼저 이론적으로 학습한 후, 각각의 알고리즘을 구현하고, 백준 문제에서 이를 적용해 풀이했습니다.

- 느낀 점
  DFS와 BFS는 기본적인 탐색 알고리즘이지만, 각각의 활용도와 구현 방법이 다르기 때문에 처음에는 어려움을 겪었습니다. 특히 BFS는 큐를 사용하여 순차적으로 탐색하므로, 최단 경로 문제에서 유용하다는 점을 깨달았습니다.
  또한, DFS는 스택을 사용하여 깊이를 우선적으로 탐색하기 때문에 그래프의 모든 경로를 빠짐없이 탐색하는 데 적합함을 느꼈습니다.

- 향후 개선 사항
  DFS와 BFS를 더 다양한 문제에 적용하여 이해도를 높이고, 탐색 효율성을 고려한 최적화 방법을 고민할 필요가 있습니다. 또한, 그래프 문제에서 다익스트라나 플로이드-워셜과 같은 더 복잡한 알고리즘을 연계해 훈련할 계획입니다.

1.2 동적 계획법 (DP)

- 학습 방법
  동적 계획법(DP)을 처음 접했을 때는 상태 전이의 개념이 낯설었으나, 이를 점화식으로 풀어보면서 점차 익숙해졌습니다.

    - 11053번 (가장 긴 증가하는 부분 수열) 문제에서는 **O(N^2)**의 시간 복잡도를 가진 이중 루프 방식을 사용한 후, **O(N log N)**으로 최적화된 이분 탐색을 적용하여 풀었습니다.

- 느낀 점
  DP는 문제를 푸는 데 있어서 큰 그림을 그리기 어렵다는 점에서 처음엔 난이도가 높았지만, 재귀적 문제 해결을 이해하고 메모이제이션과 타뷸레이션 기법을 적용하면서 점차 해결할 수 있었습니다.
  특히 배낭 문제는 DP의 전형적인 예제로, 반복되는 계산을 줄이는 기법이 얼마나 중요한지 느꼈습니다. DP 문제는 문제를 작은 단위로 분할하여 해결하

는 방식이므로, 이를 최적화하는 것이 중요하다는 점을 깨달았습니다.

- 향후 개선 사항
  더 많은 동적 계획법 문제를 풀고, 문제에서 요구하는 최적화 방식을 찾는 연습이 필요합니다. 특히 중복 계산을 피하는 방법이나 상태 공간을 줄이는 기법에 대한 이해를 더욱 깊게 해야 합니다.

1.3 그리디 알고리즘

- 학습 방법
  그리디 알고리즘은 매번 최적이라고 생각되는 선택을 하는 방식으로, 이를 처음 배울 때는 최적해를 보장하는지에 대한 확신이 부족했습니다. 그러나 여러 문제를 풀어보면서, 그리디 알고리즘이 문제 조건에 맞는 경우 매우 효율적이고 직관적인 방식임을 알게 되었습니다.
    - 11399번 (ATM) 문제는 그리디 알고리즘을 활용하여, 시간을 최소화하는 방법을 배웠습니다. 각 사람의 ATM 사용 시간이 짧은 순서대로 처리하는 방식입니다.
    - 1931번 (회의실 배정) 문제는 회의실을 효율적으로 배치하기 위한 그리디 알고리즘으로, 종료 시간이 빠른 회의부터 배치하여 회의실을 최대한 활용하는 방식입니다.
- 느낀 점
  그리디 알고리즘은 직관적이고 코드 구현이 간단하지만, 해가 최적임을 보장할 수 있는 조건을 정확히 파악해야 한다는 점에서 어려움을 겪었습니다. 여러 문제를 풀면서 최적 조건을 찾는 눈을 키울 수 있었습니다.
- 향후 개선 사항
  그리디 알고리즘은 항상 최적해를 보장하는지 검토해야 하므로, 조건을 정확히 파악하는 연습을 더 해야 합니다. 다양한 그리디 알고리즘 문제를 풀어보며 문제 해결 패턴을 익히고, 그리디 알고리즘을 적절히 적용하는 능력을 키워야 합니다.

1.4 이분 탐색

- 학습 방법
  이분 탐색은 정렬된 데이터에서 특정 조건을 만족하는 값을 찾는 알고리즘입니다. 이를 처음에는 단순히 정렬된 리스트에서의 탐색만으로 이해했지만, 문제를 풀어가면서 응용 방식이 중요하다는 것을 알게 되었습니다.

        - 2805번 (나무 자르기) 문제는 이분 탐색을 사용하여, 최소 높이를 찾는 문제를 해결했습니다.

- 느낀 점
  이분 탐색은 문제를 해결하는 데 있어서 매우 빠르고 효율적인 방법이지만, 시작과 끝을 어떻게 설정할지에 대한 고민이 필요했습니다. 특히 이분 탐색의 경계 설정을 어떻게 할지에 대해 고민하면서 많은 도움을 얻었습니다.

- 향후 개선 사항
  이분 탐색은 문제의 조건을 정확히 파악한 후, 적절한 경계 설정을 해야 하므로, 여러 이분 탐색 문제를 풀어 보며 경계 설정의 패턴을 익히는 것이 중요합니다.

2. 활동에서 느낀 점

- 다양한 접근법의 중요성
  같은 문제를 풀이하더라도 사람마다 다른 접근법을 사용했는데, 이를 통해 더 효율적인 풀이 방법을 배우고 사고의 폭을 넓힐 수 있었습니다.

- 협업의 가치
  어려운 문제를 해결할 때, 스터디원과의 의견 교환을 통해 새로운 통찰을 얻었습니다.

- 기본기 강화 필요성
  초기에 놓쳤던 알고리즘 이론이나 기초적인 부분에서 종종 막히는 경우가 있었는데, 이는 기초를 탄탄히 다질 필요성을 느끼게 했습니다.

3. 향후 계획

- 심화 문제 풀이
  기존에 다룬 알고리즘을 응용한 난이도 높은 문제 풀이 시도.

- 최적화에 집중
  단순히 문제를 푸는 데 그치지 않고, 코드의 시간 복잡도와 공간 복잡도를 분석하고 개선.

- 코딩 테스트 준비
  실전 대비를 위해 기업별 코딩 테스트 기출 문제 풀이.

4. 리드미 요약

- 스터디 목적

알고리즘 문제 풀이를 통해 컴퓨터 과학적 사고력을 기르고, 팀원들과의 협업을 통해 새로운 풀이 방법과 통찰을 얻기.

- 학습한 알고리즘 및 주요 문제

  1. 그래프 탐색 (DFS, BFS)

     - DFS(깊이 우선 탐색)와 BFS(너비 우선 탐색)를 활용하여 그래프에서의 경로를 찾는 문제를 풀었습니다.

  2. 동적 계획법 (DP)

     - 재귀적 접근을 통한 최적화 문제 해결 기법인 동적 계획법을 학습했습니다.
     - 문제 예시: 백준 11053번 (가장 긴 증가하는 부분 수열)

  3. 그리디 알고리즘

     - 최적의 선택을 하여 문제를 해결하는 그리디 알고리즘을 통해 최적화된 해결 방법을 배웠습니다.
     - 문제 예시: 백준 11399번 (ATM), 1931번 (회의실 배정).

  4. 이분 탐색

     - 정렬된 데이터에서 조건에 맞는 값을 빠르게 찾는 알고리즘인 이분 탐색을 활용한 문제 해결을 학습했습니다.
     - 문제 예시: 2805번 (나무 자르기).

  5. 기타 알고리즘

     - 알고리즘 이론을 토대로 다양한 문제를 풀면서 알고리즘의 실용성과 효율성을 체득했습니다.

- 성과
  - 다익스트라, 플로이드-워셜 등 주요 알고리즘 완전 이해.
  - 시간 복잡도 및 최적화의 중요성 인식.
  - 협업을 통한 새로운 문제 접근 방식 발견.

- 향후 목표
    - 고난도 문제 풀이 능력 향상.
    - 실전 코딩 테스트 준비와 알고리즘 최적화 능력 강화.
    - 문제 풀이에서 더 많은 알고리즘을 융합하여 효율적인 해결 방법을 찾는 연습.

최종보고서

멘사스터디 202055614 최성민

1. 공부한 내용:

2024학년도 2학기 멘사 스터디 활동을 하면서 목표했던 문제 유형들에 대한 공부와 문제 풀이를 달성할 수 있었다. 목표했던 유형은 다음과 같다.

- 그래프 탐색(+다익스트라, 플로이드 워셜), 다이나믹 프로그래밍, 이분 탐색, 구현, 자료구조, 백트래킹, 유니온 파인드, 비트마스킹

각 유형별로 이 알고리즘이 어떨 때에 사용되는지, 이 문제에 어떤 방식으로 사용됐는지에 초점을 두어 공부해서 암기식으로 문제를 푸는 것이 아니라 이해를 하고 적절한 알고리즘을 대입하여 푸는 연습을 하였다. 총 30문제를 풀이하였으며 난이도는 백준 기준 실버, 골드 난이도 문제를 풀었다.

2. 그래프 탐색

내가 항상 그래프 탐색이 어렵다고 느낀 이유가 반복문을 통해서 노드를 방문할지 방문해서는 안될지 직관적으로 생각하기 어려웠기 때문이었다. 또한, 어느 부분에서 너비 우선 탐색을 해야 할지, 깊이 우선 탐색을 해야 할지 결정하기 어려웠다. 그러나 유형별로 문제를 접하면서 14500번 테트로미노 문제와 같이 모든 경로를 거쳐 완전 탐색이 필요할 때에는 dfs, 7569번 토마토 문제와 같이 최적의 경로를 구할 때에는 bfs를 써야 한다는 것을 알 수 있었다. 더 나아가 1238번 파티 같이 시작 지점부터 도착 지점까지의 모든 경로 중에서 가장 최단거리를 구하는 문제 같은 경우는 다익스트라 알고리즘을 사용하여 문제를 해결할 수 있다는 점을 배웠다. 여기서 11403번 경로찾기 문제 같은 모든 경로를 구하는 문제는 플로이드 워셜 알고리즘을 이용해 풀이가 가능하다는 점을 알 수 있었다.

3. 다이나믹 프로그래밍(동적 계획법)

다이나믹 프로그래밍은 문제도 다양하고 다른 유형들과 접목되어 출제되는 문제들이 많았어서 처음에 접하기 굉장히 어려웠다. 1463번 1로만들기 문제와 같이 간단한 동적 프계획법 문제를 풀면서 동적 계획법의 핵심 아이디어인 최적 부분 구조와 중복되는 하위 문제, 메모이제이션을 이해하는데 도움이 되었다. 1912번 연속합과 같은 문제는 Bottom-up 방식으로 localMax 부분합을 구하는 것이 키 포인트였고, 12865번 평범한 배낭 문제는 2D DP를 구현함으로써 한정된 무게 내에서 가치의 합이 최대가 되도록 할 수 있었다. 다이나믹 프로그래밍은 조금 더 자주 풀어보면서 몸에 익히는 것이 중요하다는 점을 알 수 있었다.

4. 이분 탐색

이분 탐색은 정렬된 배열이나 리스트에서 값을 빠르게 찾을 수 있는 알고리즘이다. 이분 탐색이라는 개념은 잘 알고 있었으나 어느 부분에서 사용해야 될지는 잘 몰랐었다. 1654번 랜선 자르기 문제에서도 그랬는데 처음에는 반복문을 통해서 1부터 랜선의 최대 길이까지 찾는 방식으로 구현을 했는데 시간이 초과될 뿐만 아니라 복잡도도 많이 높았다. 여기서 사용되는 것이 이분 탐색으로 일일이 탐색하는 것이 아니라 탐색 범위를 절반씩 줄여나가기 때문에 효율적이었다. 2512번 예산 문제가 제한 시간 짧았지만 이분 탐색을 사용해 예산 요청의 최댓값을 빠르게 구할 수 있었다.

5. 자료구조

자료구조 같은 부분에서는 대개 정렬 문제에서 유용하게 사용할 수 있었다. 특히, 최소힙을 이용해서 정렬 혹은 최소값을 구하는 문제를 손쉽게 해결할 수 있었다. 1655번 가운데를 말해요 문제는 시간 제한이 0.1초로 굉장히 짧지만 heapRight는 최소힙, heapLeft는 최대힙으로 만들어 heapLeft의 root노드의 값이 중간값으로 만들어 지게끔 구현하여 해결할 수 있었다.

6. 백트래킹

백트래킹은 내게는 익숙하지 않은 문제였다. 모든 경우의 수를 탐색하면서도 불필요한 탐색을 가지치기하여 효율적으로 문제를 해결하는 알고리즘 설계 기법으로 재귀를 기반으로 최적의 해를 찾거나 가능한 해를 모두 탐색할 때 사용된다. 1759번 암호 만들기 문제가 백트래킹으로 해결하였는데 암호를 만드는 조건이 있고, 이 조건을 만족하는 가능성 있는 모든 암호를 구하는 문제이다. 백트래킹을 통해서 문자로 암호를 만듦과 동시에 valid할 시 return하는 방식으로 구현하여 해결할 수 있었다.

7. 구현

구현 문제는 다른 알고리즘을 사용하기 보다는 말그대로 직접 문제를 구현하여서 해결하는 방식이다. 17144번 미세먼지 안녕! 문제는 미세먼지가 공기청정기의 바람에 따라 변하는 것을 구현해야 하는 문제이다. 미세먼지의 움직임은 기본적으로 bfs를 이용하여 다음 번에 미세먼지가 어떻게 변하는지 구현하였다. 다만, 그 변화를 바로 원래 배열에 적용시키면 값이 변해버리기 때문에 1시간이 지난 후에 변화량을 한 번에 업데이트하는 방식을 사용했다. 미세먼지가 확산되는 방식이 시계, 반시계 방향 두 가지 이므로 이를 고려해서 정밀하게 구현하는 것이 어려웠던 점이었다.

8. 유니온 파인드

유니온 파인드는 서로소 집합을 표현하고 효율적으로 관리하기 위한 자료구조로, 주로

그래프 알고리즘에서 사이클 검출, 최소 신장 트리 생성 등에 사용 된다고 한다. 여기서 Find는 특정 요소가 속한 집합의 대표(루트) 요소를 찾는 연산이고, Union은 두 집합을 하나로 합치는 연산이다. 이 부분에서 어느 한 집합의 부모 집합을 찾을 때에는 유니온 파인드를 이용하면 손쉽게 해결할 수 있음을 알 수 있었다. 2606번 바이러스 문제는 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터에 연결된 네트워크 상의 모든 컴퓨터에게 옮기게 되는데 1번 컴퓨터가 바이러스에 걸리면 바이러스에 옮는 컴퓨터가 몇 개인지 구하는 문제이다. 여기서 네트워크를 구현하는 부분에서 Union방식으로 집합을 하나로 합치고, Find를 통해 부모가 1번인 컴퓨터의 개수를 구하여서 문제를 해결할 수 있었다.

9. 비트마스킹

비트마스킹은 문제 유형 중에서도 생소하고 어려웠던 부분인데 주로 집합을 표현하거나 특정 조건을 확인 및 변경할 때 사용되며, 비트 연산을 활용해 높은 성능을 제공한다는 이점이 있다. 9527번 1의 개수 세기 문제는 시간 제한이 1초로 굉장히 짧다. A, B 두 자연수가 주어지고 둘 사이의 자연수를 이진수로 바꿨을 때 1의 개수를 모두 세는 문제로 일일이 더해서 1의 개수를 세면 시간초과가 된다. 이 때 비트마스킹 방식을 사용하여서 2의 60제곱까지의 누적합을 구해놓고 B까지의 1의 누적합과 A까지의 1의 누적합을 빼는 식으로 구현하면 문제를 해결할 수 있다.

10. 결론 및 느낀점

스터디를 통해서 나에게 특히 어려웠던 그래프 탐색 문제를 잘 해결할 수 있었고, 유니온 파인드와 비트마스킹, 백트래킹 등 익숙하지 않았던 유형들도 공부할 수 있는 기회가 되었다. 예전에는 문제를 보면 이해하고 방식을 찾는 것이 아니라 생각 나는대로 문제를 풀어서 효율적이지 못하고 정답률도 낮았는데, 문제의 유형을 찾고 그 유형에 맞는 알고리즘을 대입하는 연습을 통해서 정답률을 높일 수 있었다. 아직 백트래킹과 비트 마스킹과 같이 익숙하지 않은 부분은 부족하여서 문제를 더 많이 접해봐야 겠다고 느꼈고, 동적 계획법과 그래프 탐색 문제는 꾸준히 문제를 풀어보며 더 자연스러워지게끔 노력해야 겠다고 생각이 든다. 이번 활동을 통해 알고리즘에 대한 접근성을 높일 수 있어 유익했고 이 활동을 통해 배운 점을 잊지 말고 계속 공부하기로 다짐했다.