【2024 PNU SW스터디그룹 최종(중간)보고서 】

[프로젝트명]

[밥밥밥 알고리즘]
[정보컴퓨터공학부/202155547] [남원정]
[정보컴퓨터공학부/202155508] [권내현]
[정보컴퓨터공학부/201924657] [장원석]
[조선해양공학과/202229161] [임예현]

[프로젝트 보고서]

1. 프로젝트 개요

1.1 프로젝트 배경

코딩 테스트는 취업 과정에서 필수적인 요소 중 하나입니다. 많은 기업들이 취업 시에 알고리즘 문제를 출제하며(일명 코딩테스트), 이를 통해 지원자들의 능력을 검증합니다. 따라서, 알고리즘 문제를 푸는 능력이 높은 학생들은 취업 시 경쟁 우위를 확보할 수 있습니다. 또한 면접에서 코딩테스트에서 푼 문제의 접근 방법이나 코드에 대해 설명하는 경우도 다수 있으므로 동아리에서 발표 및 토론을 통해 대비하고자 스터디를 진행하게 되었습니다.

특히나 컴퓨터공학 전공에서는 알고리즘 문제 해결 능력이 필요합니다. 과제나 시험에서 알고리즘 문제가 자주 출제되며 알고리즘 문제를 풀면서 학생들은 문제를 해결하는 과정에서 다양한 접근 방식을 연습하고, 문제를 해결하기 위한 논리적인 사고력을 기를 수 있습니다. 이를 통해 학생들은 문제 해결 능력을 향상시킬 수 있습니다. 학교 수업을 따라가는 것만으로는 알고리즘 능력을 기르는 데는 한계가 있기 때문에 목표가 같은 학생들이 모여 자발적으로 모여 문제를 해결하고 토론하며 그 능력을 기르고자 스터디를 개설했습니다.

1.2 개발 목표

- 백준 티어 한 단계 상승: 실버1 티어에서 골드5 티어로 올라가는 것을 구체적인 목표로 설정했습니다. 이를 통해 알고리즘 실력을 체계적으로 향상시키고, 더 어려운 문제에 도전할 수 있는 능력을 기릅니다.
- 발표 자료 30개 이상 작성: 알고리즘 문제 풀이에 대한 발표 자료를 노션 등을 활용해 체계적으로 정리하며, 스터디 참여자 간의 의견 교환과 피드백을 통해 깊이 있는 학습을 목표로 합니다.
- 깃허브에 30개 이상의 소스 코드 업로드: 모든 문제 풀이 코드를 깃허브에 올림으로써 자신만의 코드 관리 능력과 공유 문화를 학습하고, 다른 사람들의 코드를 분석하며 성장할 기회를 만듭니다.

1.3 프로젝트 환경

C++, C, SQL 등의 언어를 사용했습니다.

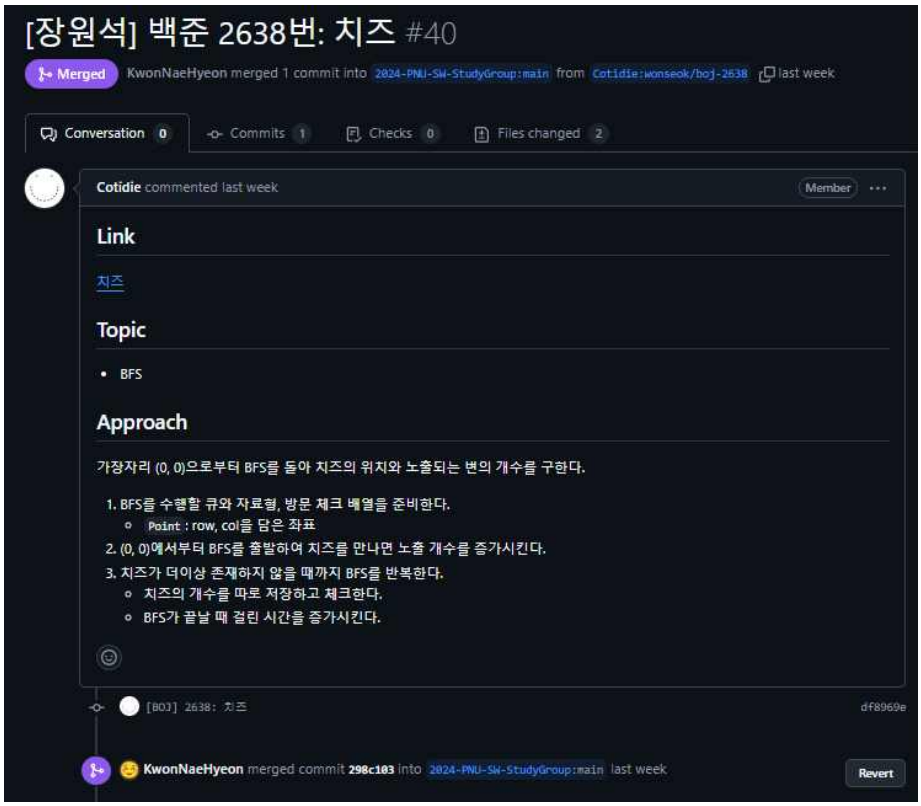
2. 프로젝트 환경

2.1 스터디 진행 방법

팀원 별로 프로그래밍 문제를 풀 수 있는 수준이 모두 다르기에, 각각 따로 수준에 맞는 문제를 풀기로 정했습니다. 책, 구글, 챗 GPT 등을 이용하여 부족한 알고리즘은 따로 정리하고, 어려웠던 문제 등은 노션에서 관리했습니다.

1) 각자 백준/프로그래머스에서 수준별 문제를 2~3개 일주일간 푼다.
2) 세미나 전까지 발표를 준비하고, 노션/깃허브에 정리한다. (선택)
3) 세미나(월요일 10시 반)에 각자 문제 풀이 방법을 공유한다. 이때, 면접관에게 설명하는 것처럼 상세하게 준비한다.

[PR 관리 현황]



[문제 풀이 내용]

3. 프로젝트 내용

3.1 스터디 진행 내용

세미나 내용 중 C++에서 유용한 함수 등을 추천하며, 서로의 팁을 공유했습니다.

예시>

```
pair<int, int> p;
p = make_pair(1,2);
p = {1,2}; // improved

int a=p.first, b=p.second;
auto [x,y] = p; // improved
```

3.1 알고리즘 설명

- 분할정복법: 주어진 문제를 작은 사례로 나누고 각각의 작은 문제들을 해결하여 정복하는 방법(Divide and Conquer)

STEP 1 - 분할과정

if (해결 가능) 그 자리에서 해결

else 적절한 크기로 쪼개기(divide)

STEP 2 - 정복과정

각각 잘려진 부분 문제 해결 (solve or conquer)
크기가 크다고 생각하면 다시 쪼갠다
→ Goto Step 1

STEP 3 - 통합과정
각 부분 해(partial solution)을 종합하여 해 구성

- 동적계획법: 하나의 큰 문제를 여러 개의 작은 문제로 나누어서 그 결과를 저장하여 다시 큰 문제를 해결할 때 사용

<조건>
1. Overlapping Subproblems(겹치는 부분 문제) : 저장된 값을 재활용할 있을 때
2. Optimal Substructure(최적 부분 구조) : 부분 문제에서 구한 최적 결과가 전체 문제에서도 동일하게 적용되어 결과가 변하지 않을 때

<복잡도>
공간 복잡도 = Table의 크기
시간 복잡도 = Table의 전체 entry를 채우는데 걸리는 시간
= (각 entry를 채우는데 걸리는 시간) x (전체 cell의 개수)

- 그리디: 최적해를 구하는데 사용되는 근사적인 방법, 미래를 생각하지 않고 매 순간마다 각자의 단계에서 최선의 해를 구해서 최종적인 답을 도출해내는 알고리즘

<조건>
1. 탐욕스러운 선택 조건
⇒ 그리디 알고리즘을 사용하면 최적해가 나올 수 있는 문제여야 한다. 반례가 없어야 함
2. 최적 부분 구조 조건
⇒ 여러 단계가 존재해야 함.

<활용 예시>
- 거스름돈 문제
- Job Scheduling
- File Merging(무거운 돌탑의 각 부분을 최소의 힘으로 옮기기)
- Partial Knapsack(물품의 일부를 잘라서 넣기) ⇒ 배낭 문제
- Minimum Spanning Tree(최소 연결 트리) 등

4. 프로젝트 결과 분석 및 평가

평균적으로 2~3문제를 풀면서, 알고리즘 실력이 향상됐으며 덕분에 코딩테스트에 통과할 수 있었습니다. 평소에 다른 스터디원들과 질의응답을 통해 임기응변 능력을 기를 수 있었습니다. 모든 팀원들이 성실하게 스터디를 진행하여, 더 높은 성과를 이룰 수 있었습니다. 하지만, 각자 알고리즘 지식 수준에 차이가 있어, 서로 다른 문제를 풀었습니다. 다음 알고리즘 스터디를 할 때는 모두 실력이 향상되어 대기업 코딩테스트 심화 수준의 문제들로만 구성한다면, 더욱 의미 있는 시간이 될 것 같습니다.

아래는 실제 대기업 코딩테스트 합격 후기 및 면접 준비 내용 입니다. 1차 면접에서 코딩테스트에 대해

질문이 나왔고, 스터디 덕분에 평소하던대로 잘 답변할 수 있었습니다. 면접에선 난이도, 1, 2번 풀이 방법과 본인만의 함수명 네이밍 기준에 대해서 물어보셨습니다.

4.1 문제 풀이 1번

<문제>
속도 일정 값만큼 올리고 내려서, 구간 통과하는 최소시간 구하기

<시간복잡도>
O(루트 n)

<접근 방법>
수학 (수열)

<구현 방식>
주어진 테스트케이스인 n=10일 때를 생각해봤을 때, 속도가 1, 2, 3, 2, 1 이렇게 규칙적인 수열이 되는 것을 보고, 수학적인 규칙성이 있을 것 같다고 생각했습니다. 규칙성을 파악하기 위해 k=0, 1일 때 각 테스트케이스별 정답을 $n \leq 20$ 일 때까지 계산했습니다. 그 결과, 답이 변화하는 지점이 2군데로 추려졌습니다. 첫 번째는1+2+3+4+3+2+1와 같이 최고 속도가 1번이고 연속되는 속도가 없는 지점, 즉 속도변화가 삼각형 형태인 유형이었습니다. 두 번째는 1+2+3+3+2+1 와 같이 최고 속도가 2번 연속으로 나오는 지점으로, 속도 변화가 사다리꼴 형태인 유형이었습니다.. 첫 번째와 두 번째 경우 속도의 전체 합을 각각 수식으로 나타내면 k*n*n와 k*n*(n+1)로 표현할 수 있습니다. i=1부터 루트 n까지 반복문을 통해 대입하여 첫 번째와 두 번째 수식을 계산 큰 값이 나오면 멈춥니다. 이때 그 전까지 첫 번째와 두 번째 지점을 만족한 개수가 정답이 됩니다.

<코드의 문제점>
첫 번째로 시간복잡도가 O(루트 n)라는 점에서 비효율적입니다. 두 번째로 반복문의 횟수를 임의로 설정했기 때문에, 이를 루트 n으로 만들고, 따로 변수로 설정하여 코드 퀄리티를 올려야 합니다. 세 번째로 해당 부분은 구현하다 보니 예외적인 케이스들이 몇몇 존재했습니다. 이를 단순 분기 처리로 해결했다는 점에서, 이후 예외적인 케이스가 충분히 발생할 수 있다고 생각듭니다.

<최적화 방법, 다른 풀이>
1+2+3+2+1 이런 형태에서 이동한 거리가 목적지까지 들어맞지 않으면, 그리디 알고리즘을 이용하여 속도를 올리거나 줄이는 과정에서 속도를 유지시켜 모자란 부분을 채우도록 설계합니다. 먼저 1+2+3+2+1 이런 형태를 수열의 합으로 표현하면 m, 즉 최대속도의 제곱 곱하기 k가 됩니다. 이때 최대 속도인 m을 구하기 위해선 m = floor (n - 1 나누기 k의 제곱근)이 됩니다. 이후에 남은 거리인 (n-1) - k 곱하기 m의 제곱은 속도 m * k ~ 0 중 조합하여 계산할 수 있습니다. 이는 잔돈 거슬러주기 문제를 활용하여 큰 값부터 0까지 for 문을 순회하여, 나머지가 0이 되도록 나누어 주면 됩니다. 이때 몫만큼 시간을 추가하면 됩니다.

4.2 문제 풀이 2번

<문제>
1, 1, 1에서 시작해 N, M, L로 이동했을 때 최댓값, 벽은 T번 지날 수 있음

<시간복잡도>
O(N * M * L * T)

<접근 방식>
구현(BFS)

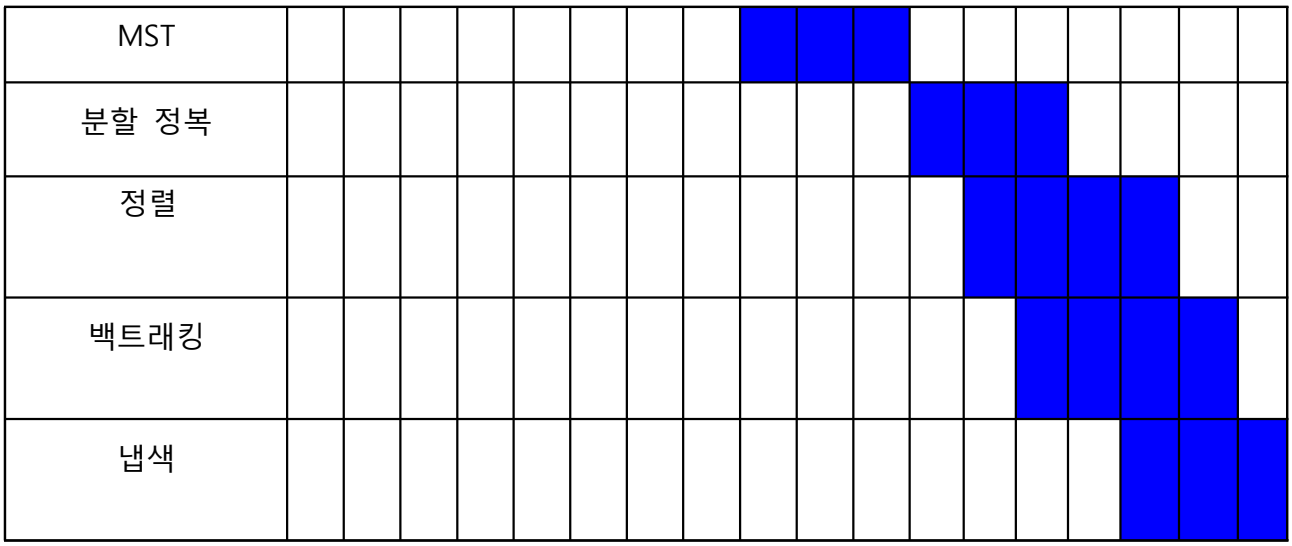
<구현방식>
BFS를 사용해서 해결할 수 있습니다. (1, 1, 1)에서 (N, M, L) 까지 도착할 때까지 아이템의 최대값을 구해야 하기 때문에, 깊이 우선 탐색보다는 넓이 우선 탐색의 특성을 활용하기 위해 BFS를 사용했습니다. 임의의 지점에서 벽을 동일한 횟수로 통과했을 때 다음 도착지 간 값을 비교하여 최댓값을 갱신합니다. 이때 시간 복잡도를 줄이고, 무한 반복이 되는 경우를 방지하기 위해 3차원 내용에 벽을 통과한 횟수를 추가하여 4차원 배열인 visit에 최댓값을 갱신했습니다.

5. 구성원별 역할 및 개발일정

공통적으로 알고리즘 문제 풀이 및 세미나에 참여했습니다.

- 남원정: 스터디 보고서 작성
- 권내현: Github PR관리
- 임예현: 알고리즘 문제 추천
- 장원석: 회의 시간 관리

구분	작업 일정																		
	9월			10월				11월				12월					1월	
	3	4	5	1	2	3	4	1	2	3	4	1	2	3	4	5	1	2
다익스트라																		
그리디 알고리즘																		
구현																		
DP																		
문자열																		
힙																		
그래프																		
스택																		



6. 참고 문헌

- 이것이 취업을 위한 코딩 테스트다 with 파이썬(나동빈 저)
- Do it! 알고리즘 코딩테스트 with C++ (김종빈 저)