컴퓨터 알고리즘 HW4 Report

202011203 정은호

A)

문제를 풀기 위한 알고리즘에 대한 생각 :

이 문제는 주어진 카드들을 더해서 만들 수 있는 모든 조합을 계산하는 문제로, 그리디 알고리즘으로 풀 수 있다. 만들 수 있는 숫자를 모두 나열해 보았을 때, 중간에 비어 있는 공간(Ex 1,( ),3)이 있다면, 즉 만들 수 있는 숫자들 사이에 빈 공간이 있을 시, 그 빈 공간 중 가장 작은 카드 한 장을 추가해야 한다. 예를 들어, 주어진 조합이 1,3,4밖에 없으면, 주어진 카드로 2는 만들 수 없다는 것이니,카드 2를 추가해야 한다. 이렇게 한 장을 추가하고, 다 만들 수 있는 카드 숫자를 다시 한 번 나열하고, 위의 과정을 반복하면 된다. 기존의 카드들과 새로 뽑은 카드들로 1~K까지의 모든 숫자를 만들 수 있을 때까지 반복하면 된다.

ChatGPT 사용 방법:

위의 알고리즘대로 실제로 만들 수 있는 모든 숫자를 나열하고 빈 공간 중 가장 작은 카드를 찾아 추가하는 과정을 반복하는 알고리즘을 naive하게 짰더니, 시간초과 문제가 발생하였다. 그리하여 내 알고리즘을 더 Time-efficient하게 짜는 방법에 대해 ChatGPT에게 조언을 구하였다. ChatGPT는 내 알고리즘을 더 효율적으로 짤 수 있는 방법을 알려주었고, 만들 수 있는 모든 숫자를 나열하는 비싼 기능을 수행하는 대신, 1부터 K까지 숫자의 범위를 점진적으로 확장해 나가며 reach_progress 변수를 도입해 현재 카드로 만들 수 있는 최대 숫자 범위를 관리하도록 하였다. 1부터 시작하여 점진적으로 숫자를 키워 나가고, Reach_progress + 1 이 숫자보다 작다면, 그 숫자를 뽑은 후 Reach_Progress를 업데이트 하는 과정을 반복한다. Reach_Progress의 값이 K보다 크거나 같으면 그때까지 추가한 카드의 숫자를 리턴하고 알고리즘을 끝내면 된다. 1부터 점진적으로 체크하였으므로, reach_progress 변수가 K보다 크다는 것은 1부터 K까지 빈 공간 없이 모든 숫자를 만들 수 있다고 생각하면 된다.

ChatGPT의 조언과 내 알고리즘에 대한 생각을 결합하여 최종 코드를 완성하였다.

B)

문제를 풀기 위한 알고리즘에 대한 생각 :

이 문제는 MST의 최소 비용을 찾는 문제이다. 이 그래프에서 찾은 MST의 비용이 메모리 최대 사용 가능량보다 크다면 -1을 리턴하고, 그렇지 않을 시, 비용을 출력하면 되는 문제이다. 이를 크루스칼 알고리즘을 이용하여 구현하였다. 크루스칼 알고리즘은 Forest를 유지하면서 MST를 찾는 알고리즘으로, 우선 각 노드를 Forest로 삼고, 엣지들을 모두 정렬해 놓아야 한다. 이후, 서로 다른 forest를 잇는 엣지 중 가장 작은 엣지를 골라서 두 Forest를 Union하는 과정을 전체 그래프를 하나의 forest가 감싸고 있을 때까지 반복하면 된다. 이를 구현하기 위해 find, union, query_single_forest, mogo_mogo_restaurant 함수를 만들었다. Find 함수는 두 forest를 연결하기 위해서 두 노드가 서로 다른 forest에 들어있는지 확인하기 위해 사용된다. Union 함수는 서로 다른 두 Forest를 Merge하는 기능을 수행한다. Merge 과정을 실제로 구현하기 위해서 각 노드마다 어느 노드가 자신을 Merge 했는지를 Parent 리스트를 통해 관리해 주어야 한다. 이는 Find를 할 때 에도 사용되며, 서로 다른 두 노드의 parent를 계속해서 거슬러 올라갔을 때 같은 parent(root)를 만나게 된다면 두 노드는 같은 forest 안에 있는 것을 알 수 있다. query_single_forest 함수를 통해 매 union이 끝날 때마다 forest가 전체 그래프를 감싸는지를 확인하고, forest가 전체 그래프를 감싼다면 이미 MST를 찾은 것으로 이해할 수 있다. mogo_mogo_restaurant는 find, union, query_single_forest 함수를 사용하여 실제로 MST를 그려나가는 역할을 수행하는 함수이다. Union을 할 때마다 mem_usage_cnt 변수에 Union할 때 사용한 엣지의 weight(cost)를 더해줌으로써 메모리 사용량을 관리한다. 최종적으로 single forest를 형성하였을 시의 mem_usage_cnt 값이 메모리 최대 사용 가능량보다 크다면 -1을 리턴하고 그렇지 않다면 해당 변수의 값을 리턴하도록 한다.

ChatGPT 사용 방법:

크루스칼 알고리즘과 이를 구현하기 위한 find, union, query_single_Forest 등의 알고리즘은 생각해내었으나, 어떠한 과정을 통해서 서로 다른 두 노드가 같은 Forest에 있는지 query 하는 마땅한 방법이 떠오르지 않아, ChatGPT에게 조언을 구했다. ChatGPT가 parent List를 사용하여 이를 find, union의 함수에서 관리함으로써 적절히 서로 다른 두 Forest를 union하고, 각 노드의 forest를 Find 할 수 있는 세부알고리즘을 제시해주었다.

ChatGPT의 조언과 내 알고리즘에 대한 생각을 결합하여 최종 코드를 완성하였다.

C)

문제를 풀기 위한 알고리즘에 대한 생각 :

이 문제는 기본적으로 포드 퍼거슨 알고리즘을 사용하여 Min-cut, max-flow를 찾는 문제이다. 다만, Min cut을 찾았을 때, 해당 컷이 K 길이 이상의 브릿지(엣지)를 지날 경우, cost가 그 다음으로 작은 컷을 찾아야 할 것이다. 이 과정을 반복하여 K 길이 이상의 엣지를 지나지 않으면서 동시에 상대 부족과 자기 부족을 분리시키는 Minimum cut을 찾게 된다면, 그 때의 flow 를 반환하면 될 것이고, 만일 그런 엣지를 찾는 데에 실패했다면, -1을 리턴하도록 구현하면 된다.

포드 퍼거슨 알고리즘에서 augmenting path를 찾는 과정은 여러 가지이지만, 그중 bfs를 이용하여 찾을 때, 이 알고리즘을 edmonds-karp 알고리즘이라고 한다. bfs를 사용하여 augmenting path를 찾을 수 있었고, ford_fulkerson 함수에서는 bfs로 찾은 path에 대해서 가장 작은 edge weight 값을 바탕으로 기존의 그래프의 Flow를 업데이트 해 주면서 Minimum cut을 찾았다. 찾은 Minimum cut을 기준으로 상대 부족이 reach할 수 있는 포인트들을 찾는 함수 find_reachable을 통해 reachable 한 Point와 unreachable한 Point를 잇는 엣지 중 K보다 큰 엣지가 있다면, 이는 K 길이 이상의 엣지가 컷을 지난다는 뜻이므로 -1을 반환하도록 하고, 그렇지 않다면 그때의 Max flow를 리턴하도록 코드를 작성하였다.

하지만 이렇게 작성하니 92점밖에 받지 못하였다. 이후 이 알고리즘을 어떻게 개선할지에 대해 ChatGPT로부터 조언을 받았다.

ChatGPT 사용 방법:

위의 내 알고리즘의 문제는 내가 찾은 cut이 K 길이 이상의 엣지를 지나면 그 다음으로 작은 cut을 찾으려는 시도를 해야 하지만, 그런 시도조차 없이 바로 -1을 리턴하도록 한 것이었다. ChatGPT는 애초에 문제에서 입력을 받아 그래프를 형성할 때, K 길이 이상의 엣지에 대해서는 capacity를 무한대(INF = 10**9) 로 설정하는 방향을 제시해 주었다. 이렇게만 해준다면, 만일 내가 찾은 Min cut의 cost가 INF이상이라면, K 길이 이상의 엣지를 지나지 않고는 상대 부족으로부터 우리 부족을 분리하는 cut을 찾지 못했다는 것이므로 -1을 리턴하면 되고, 그렇지 않을 경우, 찾은 Min cut(max flow)를 반환해 주면 될 것이다.

ChatGPT의 조언과 내 알고리즘에 대한 생각을 결합하여 최종 코드를 완성하였다.

D)

문제를 풀기 위한 알고리즘에 대한 내 생각 : (간선기반 알고리즘)

이 문제는 장애물이 있는 상태에서의 Min-cut/max flow를 찾는 문제라고 생각하였다. 격자 위의 모든 점들을 하나의 노드로 생각하고 문제에 접근하였다. 각 점들 사이의 선분들을 엣지, 점들을 노드라고 생각하면 된다. 장애물이 있는 경우, 이 장애물을 통과할 수 없어야 하므로 해당 장애물이 위치한 노드와 연결된 엣지들의 capacity를 0으로 설정하여 flow가 0이상이 될 수 없게끔 하여야겠다고 생각하였다. 그 외의 엣지들은 모두 capacity를 1로 설정하였다. 이후, C번 문제처럼, bfs와 edmonds-karp 알고리즘을 사용하여 min-cut(max-flow)를 구해 주었다. 이 때 구한 Min cut이 결국 우리가 추가적으로 설치해야 하는 장애물 개수라고 생각하였다. 그 이유는, 장애물과 연결된 엣지들의 flow는 0이 될 수밖에 없기 때문에, 결국 Flow는 설치해야 하는 장애물의 개수와 같을 것이라고 생각했기 때문이다.

ChatGPT 사용 방법 : (노드분리 기반 알고리즘)

하지만 내 알고리즘은 입력 예시에 대해서 올바른 답을 도출하지 못했다. 이에 대해서 ChatGPT에 조언을 구하였다. 내 알고리즘이 올바른 답을 도출하지 못한 이유는 노드를 물리적으로 차단하기 위해서는 해당 노드를 둘러싼 모든 간선을 막아야 하지만, 그렇게 한다면 결국 설치해야 하는 장애물의 개수와 max flow의 값 자체가 같지 않을 수 있기 때문이었다. 이에 대해 Chat GPT는 노드 분리 기반 알고리즘을 추천해 주었다. Chat GPT는 각 격자를 In 격자와 Out 격자로 나누었고, In 격자와 Out 격자 사이의 간선 용량을 1로 연결하고, 노드 차단은 그 용량을 0으로 설정함으로써 구현할 수 있게끔 추천해 주었다. 이후의 과정은 기존의 내 알고리즘과 동일하였다. BFS를 활용하여 시작점부터 종료점까지의 경로를 찾고, 해당 경로의 최소 용량만큼 유량을 증가시키는 edmonds-karp 알고리즘을 활용하였다. 더 이상 증가 경로가 없을 때까지 반복하고, 이후 Max flow를 반환하도록 하였다.

이와 같이 노드를 분리해 주면, 기존의 간선 기반 알고리즘과 달리 노드 자체를 차단해 줄 수 있기 때문에 노드를 차단하기 위해 모든 인접 간선을 차단해야 하는 문제를 해결할 수 있다. 때문에 노드분리 기반 알고리즘을 사용하면 max flow를 반화하면 기존 알고리즘보다 추가해야 하는 장애물의 개수를 올바르게 알아 낼 수 있을 것으로 기대된다.

나의 기본적인 아이디어와 ChatGPT의 노드 분리 아이디어를 결합하여 코드를 구현하여 최종 점수 84점을 기록하였다.