Busan science high school

2023 Ocean ICT Festival
2023 BOIF

B
11

QR 코드 영역
QR 삽입 후
테두리 삭제

Youtube 영상 QR

해저 자원 채굴 로봇의 효율적인 구동 방법 탐구

CY : 1314 양성수 1317 최현민

주제 선정 동기 및 목적

현재 해저 채굴을 통한 망간 단괴 확보는 전세계적인 배터리 소재 부족 문제를 해결할 수 있는 방안으로 주목받고 있다.

하지만
"수만 킬로미터에 걸쳐 생태적 피해를 입히지 않고 상업적으로 망간단괴를 채굴하는 것은 불가능"
-크레이그 스미스 미국 마노아 하와이대 해양학 교수

"해저 광물 채굴의 안정성이 입증되기 전까지
자원 이용 중단"
-삼성 SDI 등 여러 기업

"지상 채굴과 비교하면 해저 채굴이 환경 파괴가 적다"
-해저 채굴 기업
등 과 같이 상반된 의견을 보이고 있다.

우리는 탐구를 통해 생태적 피해를 입히지 않고 망간단괴를 채굴할 수 있는지, 또 해저 자원 채굴 로봇을 효율적으로 구동시킬 수 있는 방법에는 어떤 것들이 있는지를 알아보고자 한다.

기대 효과

① 망간 단괴 : 심해저에서 발견되는 망가니즈를 주성분으로 하는 금속덩어리. 동그란 구 모양의 외관을 보이며 흑갈색, 흑색을 띠는데, 주로 망가니즈, 철, 구리, 니켈, 코발트의 금속 원소로 구성된다.

시작지점에서 모든 망간단괴를 최적의 방법으로 지나는 것을 통해서 망간단괴 채굴을 비롯한 이와 비슷한 상황에서 에너지 효율을 높일 수 있을 것 입니다.

탐구 내용 - 정보과학

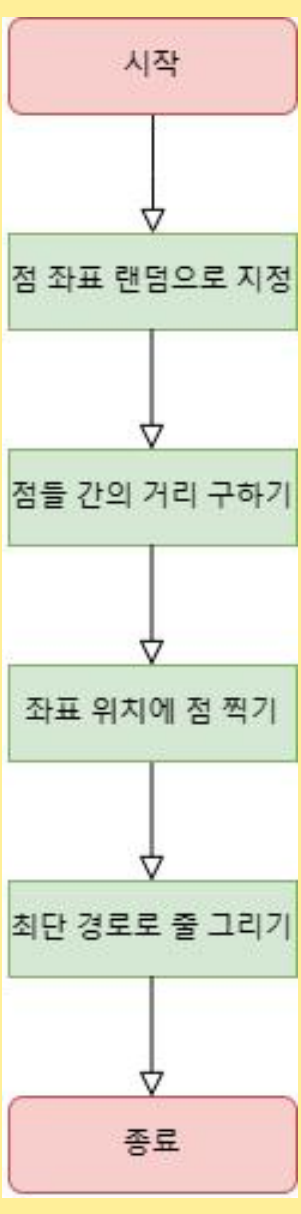
⊛ 동적 계획법(dynamic programming)이란 복잡한 문제를 간단한 여러 개의 문제로 나누어 푸는 방법을 말한다. 이것은 부분 문제 반복과 최적 부분 구조를 가지고 있는 알고리즘을 일반적인 방법에 비해 더욱 적은 시간 내에 풀 때 사용한다.

⊛ 메모이제이션 : 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술이며, 동적계획법의 핵심이 되는 기술이다.

입출력 설계 & 알고리즘

입력 : 랜덤한 점의 좌표
출력 : 점을 찍고 최단거리로 줄 그리기

< 입출력 설계 >



< 알고리즘 >

코드 설명

```python
import math
import random
import turtle
INF = 999999                # 처음 초기 거리값을 infinity 무한으로 잡는다. 후에 거리의 최솟값을 구할 때 이용
def get_distance(point1, point2):
    # 두 점 사이의 유클리드 거리 계산
    x1, y1 = point1
    x2, y2 = point2
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def find_shortest_path(points):
    n = len(points)  # 점의 개수
    INF = 999999

    # 모든 점들 사이의 거리 계산하여 graph에 저장
    graph = [[0] * n for _ in range(n)]    # 점들 간 거리 table 만들기
    for i in range(n):
        for j in range(i + 1, n):
            dist = get_distance(points[i], points[j])   # i점과 j점 사이의 거리 구하기
            graph[i][j] = dist
            graph[j][i] = dist

    dp = [[INF] * (1 << n) for _ in range(n)]
    # dp[i][j]: i = 현재 위치, j = 아직 방문하지 않은 노드 정보
    # dp[i][j] = 남은 노드들을 최적 경로로 돌았을 때의 총 거리
```

```python
def tsp(start, visited):        # 여기 : tsp(Traveling Salesman Problem) 알고리즘 에서 따옴
    # 모든 점점을 방문한 경우                    # 기본 아이디어:  visited :  해당 비트가 0이면 방문하지 않은 것, 1이면 방문한 것을 나타냅니다.
    if visited == (1 << n) - 1:
        return graph[start][0]

    # 이미 계산한 값이 있는 경우
    if dp[start][visited] != INF:
        return dp[start][visited]

    # 다음 방문할 노드 선택
    min_dist = INF              # 최소거리 지정 초기값: 무한으로 잡기
    for next_node in range(1, n):
        if visited & (1 << next_node) == 0:
            dist = graph[start][next_node] + tsp(next_node, visited | (1 << next_node))
            min_dist = min(min_dist, dist)

    dp[start][visited] = min_dist
    return min_dist

# 시작 위치는 0, 아직 방문하지 않은 노드는 모든 노드로 초기화
start_node = 0
initial_visited = (1 << n) - 1

shortest_distance = tsp(start_node, initial_visited)
return shortest_distance, graph
```

이 때 우리는 인덱스로 접근하기 쉽게 하기 위해서 두 번째 visited 인자로 숫자를 사용할 것이고, 각각의 비트를 각각의 도시에 매핑해서 0과 1로 방문 여부를 표시할 것이다.

즉, 5개의 도시 중 visit 이 00110 이라면 2, 3번 도시를 방문한 것이고 00001 이라면 1번 도시만 방문한 것이고 11111 이라면 모두 방문한 것이다.

< 점들 사이의 거리 구하기 >

코드 설명

```python
def generate_random_points(num_points):
    points = []
    for _ in range(num_points):
        x = random.randint(-300, 300)
        y = random.randint(-300, 300)
        points.append((x, y))
    return points
```

< 점 랜덤으로 지정 >

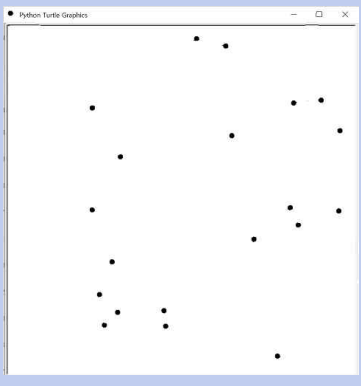
실행 코드

```python
# 점의 개수를 랜덤하게 결정
num_points = 20 #random.randint(3, 15)

# 랜덤 좌표로 점들 생성
points = generate_random_points(num_points)

# 최단 경로 계산 및 그래프 반환
shortest_distance, graph = find_shortest_path(points)

# 최단 경로 찾기
path = [0]  # 시작점은 항상 0
visited = [False] * num_points
visited[0] = True

current = 0
for _ in range(num_points - 1):
    next_node = -1
    min_dist = INF
    for i in range(num_points):
        if not visited[i] and graph[current][i] < min_dist:
            next_node = i
            min_dist = graph[current][i]
    path.append(next_node)
    visited[next_node] = True
    current = next_node

draw_dot(points, path)
# 그래픽으로 최단 경로 표시
draw_path(points, path)

print("Points:", points)
print("Shortest Distance:", shortest_distance)
```

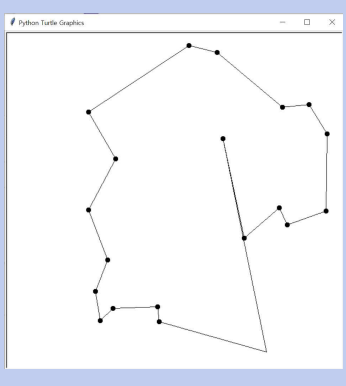
코드 설명

```python
def draw_dot(points, path):
    turtle.speed(2)
    turtle.penup()
    for i in range(len(path) - 1):
        start = path[i]
        end = path[i + 1]
        x1, y1 = points[start]
        x2, y2 = points[end]
        turtle.goto(x1, y1)
        turtle.dot(10)
        turtle.goto(x2, y2)
    turtle.goto(points[path[-1]])
    turtle.goto(points[path[0]])
```

```python
def draw_path(points, path):
    turtle.speed(2)
    turtle.penup()
    for i in range(len(path) - 1):
        start = path[i]
        end = path[i + 1]
        x1, y1 = points[start]
        x2, y2 = points[end]
        turtle.pendown
        turtle.goto(x1, y1)
        turtle.pendown()
        turtle.goto(x2, y2)
        turtle.penup()
    turtle.pendown()
    turtle.goto(points[path[-1]])
    turtle.goto(points[path[0]])
    turtle.hideturtle()
    turtle.done()
```



< 좌표 위치에 점 찍기 >



< 최단 경로로 줄 그리기 >