**최단 경로 찾기 알고리즘(EDOC 3주차)**

#Floyd-Warshall(플로이드 워샬 알고리즘)

|  |
| --- |
| 플로이드-워샬 알고리즘은 그래프에서 모든 정점 사이의 최단 거리를 구하는 알고리즘이다.  이 알고리즘은 모든 정점에 대한 경로를 저장하기 때문에 모든 정점 사이의 최단 거리를 구하는 것이 가능하다. 모든 정점에 대한 경로를 계산하므로 거리를 저장할 자료 구조는 2차원 배열이 디면, 이를 동적계획법을 이용해 최적의 값을 계산하게 된다. 이 알고리즘은 경우지 k를 두고  dp[i][j] = min(dp[i][j], dp[i][k]+dp[k][j]) 이다. |

|  |
| --- |
| 1. 2차원 배열을 만들고 그래프의 간선의 정보를 저장한다.(두 정점이 직접적으로 연결되어 있는 것이 아니면 매우 큰 값을, 자기 자신으로 가는 거리는 0으로 한다.)  2. 경유지 1~v까지 순회하여 2차원 테이블을 업데이트 한다. |

#다익스트라 알고리즘

|  |
| --- |
| 한 노드에서 다른 노드까지의 최단 거리를 구하는 알고리즘이고, 처음에는 n개의 수만큼 배열을 초기화한 다음에, 다익스트라 알고리즘을 시작노드를 하나씩 증가시키며 진행한다.  즉, 첫 정점부터 각 노드간의 거리를 저장하는 배열을 만든 후, 첫 정점의 인접 노드간의 거리부터 먼저 계산하면서 첫 정점부터 해당 노드간의 가장 짧은 거리를 해당 배열에 업데이트하는 방법을 이용한다. |

|  |
| --- |
| ① 지도상의 모든 건물들과 집에서 각 건물들까지의 최단 거리를 나타내는 표를 만든다.  ② 집과 직접 길로 이어진 건물들까지의 최단 거리는 지도에 표시된 값으로 적고 그렇지 않은 건물들은 빈 칸으로 놓아둔다. 여기서 빈 칸의 값은 무한대를 뜻한다.  ③ 거리가 가장 짧은 건물부터 긴 건물 순으로 방문하고 방문한 건물은 색깔로 칠해 구별한다. 이때 방문한 경로도 색칠한다.  ④ 새로운 건물을 방문하면 그 건물과 이어진 건물들까지의 거리를 새로 바꾼다. 단, 이전에 이미 최단 거리가 구해졌었다면 거리를 서로 비교해 작은 것으로 바꾸거나 유지한다.  ⑤ 그래프의 모든 건물들을 방문할 때까지 ③,④의 과정을 반복한다.  여기서 from collections import deque를 해서 deque자료구조를 이용한다. |

#Heapq

|  |
| --- |
| **import** heapq  **class** ReverseLessThan(object):  **def** \_\_init\_\_(self, value):  self.value = value  **def** \_\_lt\_\_(self, other):  **return** self.value > other.value  **def** \_\_repr\_\_(self):  **return** str(self.value)  **def** heappush(heap, item): *#원소 넣기*  reverse\_item = ReverseLessThan(item)  heapq.heappush(heap, reverse\_item)  **def** heappop(heap): *#원소 꺼내기*  reverse\_item = heapq.heappop(heap)  **return** reverse\_item.value  **def** heapify(lst): *#배열로부터 힙 만들기(최소힙만 파이썬이 지원해주기 때문에 최대힙을 위해 값\*-1을 해주기도 한다.*  **for** i, ele **in** enumerate(lst):  lst[i] = ReverseLessThan(ele)  heapq.heapify(lst) |

#6118번 숨바꼭질

|  |
| --- |
| **import** sys n,m = map(int,sys.stdin.readline().split()) board = [[]**for** \_ **in** range(n)] **for** \_ **in** range(m):  a,b = map(int, sys.stdin.readline().split())  board[a-1].append(b-1)  board[b-1].append(a-1) visit = [9999999]\*n visit[0] = 0  **from** collections **import** deque file = deque() file.append(0) **while** file:  a = file.popleft()  **for** k **in** board[a]:  **if** visit[k] == 9999999:  file.append(k)  visit[k] = min(visit[k], visit[a]+1)  a,b,c = (visit.index(max(visit)))+1, max(visit), visit.count(max(visit)) print(a,b,c)  #bfs를 이용하여 해결하였고, 처음에는 그냥 for문을 이용하여 순서대로 board에 들어있는 값들의 거리만 최솟값으로 갱신해주려 했었다. 그러나 그렇게 하면 당연히 최소 경로를 제대로 찾을 수 없기 때문에 방법을 바꾸어 AC를 만들었다. |

#11404번 플로이드

|  |
| --- |
| 한 도시에서 출발하여 다른 도시에 도착하는 m개의 버스와 n개의 도시가 존재한다. 이때 각 버스는 한번 사용할 때 필요한 비용이 정해져 있는데, 모든 도시의 쌍(A,B)에 대해서 도시 A에서 B로 가는데 필요한 비용의 최솟값을 구하는 프로그램을 작성하시오. |

|  |
| --- |
| n = int(input()) m = int(input())  **def** DFS(i,j):  count = 0  **for** x **in** range(n):  **if** board[j][x] != 9999999 **and** j != x:  count += 1  board[i][x] = min(board[i][x], board[i][j] + board[j][x])  **return** board  board = [[9999999]\*n **for** x **in** range(n)] **for** x **in** range(m):  a,b,c= map(int, input().split())  board[a-1][b-1] = min(board[a-1][b-1], c) **for** i **in** range(n):  board[i][i] = 0 **for** i **in** range(n):  **for** j **in** range(n):  **if** i != j **and** board[i][j] != 9999999:  DFS(i,j)  **if** i != j **and** board[j][i] != 9999999:  DFS(j,i)  **for** i **in** range(n):  **for** j **in** range(n):  **if** board[i][j] == 9999999:  board[i][j] =0 **for** i **in** range(n):  **for** j **in** range(n):  print(board[i][j], end = **' '**)  print() |

#11403번 경로 찾기

|  |
| --- |
| 가중치 없는 방향 그래프 G가 주어졌을 때, 모든 정점(i,j)에 대해서 i에서 j로 가는 경로가 있는지 없는지 구하여야 한다. i번째 줄의 j번째 수가 1이면 연결 간선이 있다는 뜻이고 0이면 없다는 뜻이다. 즉, 연결 그래프를 갱신하라는 의미이다. |

|  |
| --- |
| n = int(input()) **from** collections **import** deque board = [] **for** i **in** range(n):  board.append(list(map(int, input().split()))) file = deque() **for** i **in** range(n):  **for** j **in** range(n):  **if** board[i][j] == 1:  file.append((i,j)) **while** file:  a,b = file.popleft()  **for** i **in** range(n):  **if** board[b][i] == 1 **and** board[a][i] == 0:  board[a][i] = 1  **if** a != i:  file.append((a,i))  **if** b != i:  file.append((b,i)) **for** i **in** range(n):  **for** j **in** range(n):  print(board[i][j],end = **' '**)  print() |

#3155번 터널

|  |
| --- |
| 천장과 바닥의 좌표가 주어질 때 터널을 통과하는 최단 경로의 y좌표를 순서대로 출력하는 문제인데, 자꾸만 런타임 에러가 발생해서 다시 고민해 봐야 할 것 같은 문제이다. 우선 리스트 두개를 하나의 리스트로 병합을 할 때에 +연산자를 이용해서도 가능하다는 것을 알게 되었고, 이는 dfs, bfs문제 등의 탐색 문제를 풀 때 append대신에 많이 사용 될 것 같다.  +)이후sys.recursionlimit(10\*\*9)를 해주니 런타임은 해결 되지만 메모리 초과가 발생했다. |

|  |
| --- |
| **import** sys sys.setrecursionlimit(10\*\*9)  n = int(input()) ceiling = list(map(int, input().split())) floor = list(map(int, input().split())) ans = 99999999999 answer = [] **def** check(a,b,c, res): *#a,b,c는 순서대로 현재 x죄표, 높이, 총 경로의 길이이다.* **global** ans, answer  **if** a == n-1:  c += abs(b)  **if** ans > c:  ans = c  answer = res+[0]  **return  for** i **in** range(max(floor[a-1]+1, floor[a]+1, floor[a+1]+1), min(ceiling[a-1],ceiling[a], ceiling[a+1])):  check(a+1, i, c+(abs(i-b)), res+[i]) check(1,0,0,[0]) **for** i **in** answer:  print(int(i), end = **' '**) |

#18352번 특정 거리의 도시 찾기

|  |
| --- |
| 그냥 도시들이 주어지고 그 사이의 ‘최단 거리’가 주어진 거리인 k와 일치하는 도시들의 번호를 출력하고 없으면 -1을 출력하는 것이다. 계속 출력초과가 발생했던 이유는 양방향 간선이 아니라 단방향 간선임을 눈치 채지 못했기 때문이다. 그것 이외에는 그냥 bfs를 이용해 탐색 안 한 노드를 순차적으로 시작 노드부터 진행해 주면 되는 것이었다. |
| **import** sys n,m,k,x = map(int, sys.stdin.readline().split()) board = [[]**for** \_ **in** range(n)] **for** \_ **in** range(m):  a,b = map(int, sys.stdin.readline().split())  board[a-1].append(b-1) res = [9999999]\*n **from** collections **import** deque file = deque() file.append(x-1) res[x-1] = 0 **while** file:  now = file.popleft()  **for** p **in** board[now]:  **if** res[p] == 9999999:  file.append(p)  res[p] = min(res[p], res[now]+1) ans = [] **for** idx, num **in** enumerate(res):  **if** num == k:  ans.append(idx+1) ans.sort() **if** ans:  **for** i **in** ans:  print(i) **else**:  print(-1) |

#1238번 파티

|  |  |
| --- | --- |
| 플로이드 와샬(python3, pypy3 모두 TLE) | 다익스트라 알고리즘 2번 사용(시간 초과 없음)  (약간 탐색 알고리즘 중 bfs와 비슷하게 deque를 이용한다.) |
| **import** sys n,m,x = map(int, sys.stdin.readline().split()) board = [[99999999]\*n **for** \_ **in** range(n)] **for** \_ **in** range(m):  a,b,c = map(int, sys.stdin.readline().split())  board[a-1][b-1] = c  **for** i **in** range(n):  **for** j **in** range(n):  **for** k **in** range(n):  **if** j == k:  board[j][k] = 0  **if** board[j][i] != 999999999 **and** board[i][k] != 999999999:  board[j][k] = min(board[j][k], board[j][i]+board[i][k])  res = 0 **for** i **in** range(n):  res = max(res, board[i][x-1]+board[x-1][i]) print(res) | **import** sys **from** collections **import** deque n,m,x = map(int, sys.stdin.readline().split()) board\_1, board\_2 = [[]**for** \_ **in** range(n)], [[]**for** \_ **in** range(n)] dp\_1, dp\_2 = [9999999]\*(n+1), [9999999]\*(n+1)  **def** check(start, dp, s):  file = deque()  dp[start] = 0  file.append((0, start))  **while** file:  a,b = file.popleft()  **if** dp[b] < a: *#dp리스트 갱신이 무의미*  **continue  for** i,j **in** s[b]:  **if** j+a < dp[i]:  dp[i] = j+a  file.append((j+a, i))  **for** \_ **in** range(m):  a,b,c = map(int, sys.stdin.readline().split())  board\_1[a-1].append((b-1, c))  board\_2[b-1].append((a-1, c)) check(x-1, dp\_1, board\_1) check(x-1, dp\_2, board\_2) ans = 0 **for** i **in** range(n):  ans = max(ans, dp\_1[i]+dp\_2[i]) print(ans) |

#10282번 해킹

|  |
| --- |
| **for** \_ **in** range(t):  n,d,c = map(int, sys.stdin.readline().split()) *#컴퓨터 개수, 의존성 개수, 해킹당한 컴퓨터의 번호* board = [[]**for** \_ **in** range(n)]  **for** \_ **in** range(d):  a,b,s = map(int, sys.stdin.readline().split())  board[b-1].append((a-1, s))  res = check(c-1)  num, time = 0,0  **for** i **in** res:  **if** i != 999999999:  time = max(time, i)  num += 1  print(num, time) |

|  |  |
| --- | --- |
| *#다익스트라 알고리즘 적용(전형적인 최단 거리를 구하는 알고리즘)(맞)*  **def** check(start):  file = deque()  visit = [999999999]\*n  file.append(start)  visit[start] = 0  **while** file:  a = file.popleft()  **for** i,j **in** board[a]:  now = j+visit[a]  **if** visit[i] > now:  visit[i] = now  file.append(i)  **return** visit | *#bfs를 이용하여 해결(틀)*  **def** check(start):  file = deque()  visit = [999999999]\*n  file.append(start)  visit[start] = 0  **while** file:  a = file.popleft()  **for** i,j **in** board[a]:  **if** visit[i] == 999999999:  file.append(i)  visit[i] = min(visit[i], visit[a]+j)  **return** visit  *#이렇게 푸니 50%에서 계속 틀렸다고 나옴*  *#그 이유는 이렇게 풀면 계속해서 최솟값으로 갱신이 되는데 따라서 이미 방문한 상태인 값을 계속 최솟값으로 갱신해 처음의 시작 노드가 일치하지 않고 중간에 끊긴 경로로 이동할 수도 있기 때문이다.* |

#1753번 최단 경로

|  |
| --- |
| 다익스트라 알고리즘을 이용하면 되는 문제였는데, 처음에는 그냥 dfs문제처럼 deque를 이용했었는데 그렇게 하니까 시간 초과가 발생했다. 그 이유는 다익스트라 알고리즘이 ‘시작하는 노드에서부터 연결된 노드들의 가중치 중에서 제일 작은 가중치를 선택하여 노드의 값을 갱신하여야 하기 때문’이다. 무엇보다 이 알고리즘은 그냥 탐색이 아닌 제일 짧은 거리를 찾는 것이기 때문에 최소인 가중치를 찾는 것이고, 파이썬에는 heapq라는 내장 모듈이 존재한다.  이는 우선 순위 큐인데, 이는 우선순위가 가장 높은 자료를 먼저 꺼낼 수 있는 자료구조이다. 즉, 자료에 입력된 값이 오름차순으로(작은 수일 때 우선순위가 높은 것으로 간주) 정렬되는 구조이다. |
| **import** sys node,line = map(int, sys.stdin.readline().split()) k = int(input()) board = [[]**for** \_ **in** range(node)] **for** \_ **in** range(line):  u,v,w = map(int, sys.stdin.readline().split())  board[u-1].append((v-1, w)) **from** heapq **import** heappush, heappop file = [] heappush(file, [0,k-1]) visit = [99999999]\*node visit[k-1] = 0 **while** file:  b,a = heappop(file)  **for** i,j **in** board[a]:  now = j+b  **if** visit[i] > now:  visit[i] = now  heappush(file, (now, i)) **for** i **in** visit:  **if** i == 99999999:  print(**'INF'**)  **else**:  print(i) |

#1916번 최소 비용 구하기

|  |
| --- |
| **import** sys n = int(sys.stdin.readline()) m = int(sys.stdin.readline()) board = [[]**for** \_ **in** range(n)] **for** \_ **in** range(m):  a,b,c = map(int, sys.stdin.readline().split())  board[a-1].append((b-1, c)) start, end = map(int, sys.stdin.readline().split()) **from** heapq **import** heappop, heappush file = [] visit = [99999999]\*n heappush(file, (0, start-1)) **while** file:  a,b = heappop(file)  **for** i,j **in** board[b]:  now = a+j  **if** visit[i] > now:  visit[i] = now  heappush(file,(now,i)) print(visit[end-1])  *#기본적인 다익스트라 알고리즘 구현만 가능했다면 풀 수 있는 문제였다. 다만 파이썬의 리스트 구현에 있어서 자꾸만 헷갈리는 부분들이 생기는 것 같다. 일단 []\*3을 하면 그냥 []이렇게 되고 [[]for \_ in range(3)]을 해야만 [[],[],[]]이렇게 된다.* |

#1504번 특정한 최단 경로

|  |
| --- |
| 다익스트라 알고리즘을 3번씩 2번 이용해야 하는 문제였기 때문에 다익스트라 알고리즘을 함수로 정의하고 문제를 풀었다. 그렇게 한 이유는 ‘반드시 지나야 하는 정점’이 존재하였고 ‘최단 경로’로 이동해야 한다는 조건이 있었기 때문이다. 시작 노드와 끝 노드는 반드시 0번째와 n-1번쨰여야 했고, 이때 반드시 거쳐야 하는 노드를 각각 first, second 라 한다면  0 -> first +first -> second + second -> n-1 이나  0 -> second + second -> first + first -> n-1로 해결하면 된다.  이 값이 min으로 해도 갱신이 안되면 -1을 출력하도록 했다. |
| **import** sys n,e = map(int, sys.stdin.readline().split()) board = [[]**for** \_ **in** range(n)] **for** \_ **in** range(e):  a,b,c = map(int, sys.stdin.readline().split())  board[a-1].append((b-1, c))  board[b-1].append((a-1, c)) no\_1, no\_2 = map(int, sys.stdin.readline().split())  **from** heapq **import** heappop, heappush **def** move(v,x):  file = []  visit = [99999999]\*n  heappush(file, (0,v))  visit[v] = 0  **while** file:  i,j = heappop(file)  **for** p,q **in** board[j]:  now = q+i  **if** visit[p] > now:  visit[p] = now  heappush(file, (now, p))  **return** visit[x]  ans = 99999999 ans = min(ans, move(0,no\_1-1)+move(no\_1-1, no\_2-1)+move(no\_2-1, n-1)) ans = min(ans, move(0, no\_2-1)+move(no\_2-1, no\_1-1)+move(no\_1-1, n-1))  **if** ans == 99999999:  print(-1) **else**:  print(ans) |