|  |
| --- |
| ***Knapsack\_Tree*** |
|  |

Node -> 트리에 관한 정보를 가지고 있는 구조체를 선언해 준다. 트리의 높이를 의미하는 level

*Level : 트리의 높이를 나타내는 변수*

*Profit : 루트에서 현재 노드까지 오는 데에 필요한 경로 값 가중치*

*Bound : 현재 노드의 서브 트리에서 최대 이익의 경계*

*Item -> 아이템의 상응하는 가치와 무게를 저장하는 구조체*

*Bound 함수 : 매개변수로 넘어온 노드의 레벨에서 최대 이익의 경계 값을 구한다*

1. *Queue가 처음 노드일 경우 v노드의 level을 0으로 초기화 시켜주고 level이 마지막 노드일 경우 지나간다*
2. *차례대로 현재 노드의 level과 해당 층에 속하는 아이템 값과 가치를 더하여 갱신한다.*
3. *현재 노드 cost가 최대 무게보다 작고 profit이 현재 최대 가성비보다 크다면 max\_profit을 현재노드(v)의 profit으로 갱신한다.*
4. *현재 노드 최대 경계 값을 bound 함수 값을 이용하여 저장한 후 현재 노드가 가지고 있는 profit보다 크다면 queue에 현재 노드를 넣는다.*
5. *과거 노드의 최대 경계 값을 bound 함수 값을 이용하여 저장한 후 과거 노드가 가지고 있는 profit보다 크다면 queue에 현재 노드를 넣는다.*

***TSP***

|  |
| --- |
| ***TSP\_brute*** |
|  |

*모든 경우의 수를 출력하는 Brute를 사용한 TSP*

1. *방문할 노드를 넣을 path와 방문한 곳을 표시할 visited 배열, 현재까지 획득한 값을 저장할 cost를 매개변수로 선언해 준다.*
2. *만약 현재 방문하려는 노드가 이미 방문한 곳이면 다음으로 넘어가고 만약 방문하지 않은 곳이라면 cur에 마지막으로 들어가 있던 노드를 임시 저장한 후 새 노드를 넣어주고 방문한 노드라고 표시*
3. *현재 방문상태와 방문할 path 그리고 cost값을 갱신하여 재귀함수 호출. 모든 경로를 방문했을 때 다시 0으로 돌아오는 수까지 더하여 반환하고 반환된 수들 중에서 최소값을 t\_cost로 갱신한다. 반환이 완료되면 마지막으로 방문했던 노드 방문 여부를 false로 바꿔주며 모든 방문 경우의 수를 돌며 최소값을 t\_cost에 갱신하여 마지막에 반환한다.*

|  |
| --- |
| ***TSP\_bit*** |
| Visited[] 배열을 bit 연산자로 바꾸면 된다. |

*Brute를 사용한 TSP에서 배열 대신 bit를 사용한 알고리즘*

|  |
| --- |
| ***TSP\_dp*** |
|  |

*DP를 사용한 TSP*

1. *비트마스크를 통해 이미 방문했거나 자기 값이면 다른 노드를 찾는다.*
2. *현재 방문한 도시와 방문할 노드를 갱신하여 재귀 함수를 호출하고 이를 통해 방문한 도시들과 현재 있는 도시 번호가 같은 지 확인하여 (memorization) 된 cost 값을 받는다. 반환된 cost 값과 현재 노드에서 다음 노드로 가려는 경로 값을 더하여 가장 최소 값으로 cost 값을 갱신시켜준다.*

*기억해 둬야 하는 비트 마스크 연산자*

|  |  |  |  |
| --- | --- | --- | --- |
| *공집합* | *set = 0* | *p번 원소 추가* | *set |= (1<<p)* |
| *p번 원소 삭제* | *set &= ~(1<<p)* | *p번 원소 토글* | *set ^= (1<<p)* |
| *전체집합* | *set = (1<<20) -1* | *p번 원소 확인* | *if(set&(1<<p))* |

***BFS & DFS & Recursive\_DFS***

|  |
| --- |
| ***BFS*** |
|  |

*queue을 이용하여 형제를 먼저 탐색하는 너비 우선 탐색 알고리즘*

1. *Queue에서 가장 앞에 있는 노드를 빼낸 후 뺀 노드는 queue에서 제거해 준다.*
2. *해당 노드와 인접해 있는 노드를 모두 빼내어 큐에 넣어 준다. 여기서 만약 이미 넣었던 노드라면 큐에서 제거만 해준 후 다음 노드로 넘어간다.*
3. *queue의 크기가 0이 될 때까지 반복*

|  |
| --- |
| ***DFS*** |
|  |

*stack을 이용하여 자식 노드를 먼저 탐색하는 깊이 우선 탐색 알고리즘*

1. *stack에서 가장 앞에 있는 노드를 빼낸 후 뺀 노드는 stack에서 제거해 준다.*
2. *해당 노드와 인접해 있는 노드를 모두 빼내어 큐에 넣어 준다. 여기서 만약 이미 넣었던 노드라면 큐에서 제거만 해준 후 다음 노드로 넘어간다.*
3. *Stack의 크기가 0이 될 때까지 반복*

|  |
| --- |
| ***Recursive\_DFS*** |
|  |

*너비 우선 탐색 DFS를 재귀로 푼 방법*

***N\_Queen***

|  |
| --- |
| ***N\_Queen*** |
|  |

*Promising한지 아닌지를 판별하여 체스판에 들어갈 수 있는 Queen의 수를 구하는 알고리즘*

1. *Promising 함수 : 해당 노드가 유망한 지 판단하는 함수*

*K 는 예전 자리 I 는 현재 자리이며 array배열은 현재 queen이 존재하는 좌표를 나타낸다.*

*(x: i , y : array[i]) 매개변수를 통해 전달된 자리가 아래 식을 만족할 때 true를 반환하고 아니라면 false를 반환한다.*

***array[i]==array[k]| AND abs(array[i]-array[k])==i-k***

1. *Promising 함수에 의해 해당 자리가 유망하다고 판명이 나면 해당 자리가 마지막 자리인지 확인 후 맞으면 출력을 아니라면 x 좌표를 증가시키며 queens를 재귀 호출하여 최종 자리를 결정한다.*

***Problem***

|  |
| --- |
| ***Problem*** |
| ***문제 1 . Greedy***    ***문제 1. Brute*** |

*Greedy를 이용한 cow 문제 풀기*

1. *무게가 큰 순서대로 cow를 정렬*
2. *정렬 된 순서대로 최대 값에서 빼주고 만약 이미 옮겼다면 다음 소로 넘어간다.*

***Prims & Kruskal’s & Dijkstra***

|  |
| --- |
| ***Prims 알고리즘*** |
|  |

*인접한 노드들 중 최소 간선으로 연결된 노드를 선택하여 트리를 확장하는 알고리즘*

*Vertex\_min 함수 : 최소값을 가지는 정점을 반환하는 하는 함수*

1. *Dist[]는 해당 노드가 가진 최소 가중치를 담은 노드로 만약 이 노드의 값이 INF면 연결되어 있는 노드가 아니기 때문에 다음 노드로 넘어가고 해당 출발 노드에서 가는 더 작은 가중치가 있다면 값을 다시 갱신해 준다.*
2. *Vertex\_min 함수를 통해 이미 방문한 곳이 아니며 연결된 노드 중 가장 최소 값을 가지는 노드를 반환 받아 해당 노드와 연결 된 인접 노드들의 최단 거리를 다시 갱신하여 준다.*

|  |
| --- |
| ***Prims 알고리즘*** |
|  |

*Vertex\_min 함수 대신 우선순위 큐를 사용한 알고리즘*

|  |
| --- |
| ***Kruskal 알고리즘*** |
|  |

*최소 간선 거리를 선택하는 알고리즘*

*getSet 함수 : 가장 뿌리 노드를 찾는 함수로써 재귀 함수를 통해 뿌리 노드 값을 반환 한다.*

*UnionSet 함수 : 두 노드를 연결하는 함수로써 작은 노드가 뿌리가 되도록 큰 수의 노드가 작은 수의 노드를 값으로 갖게 한다.*

*Find 함수 : 서로의 뿌리 노드가 같은 지 확인 하는 함수 getSet 함수를 사용하여 확인한다.*

1. *노드 값을 넣은 후 노드 값이 작은 순 대로 정렬 한다.*
2. *반복문을 순서대로 돌면서 사이클이 생기지 않으면 해당 간선을 선택 사이클이 생긴다면 해당 간선을 선택하지 않고 다음으로 작은 간선으로 넘어간다.*

|  |
| --- |
| ***Dijkstra 알고리즘*** |
|  |

1. *Priority\_queue에 pair로 연결된 정점번호 간선 가중치 쌍을 넣어주는 queue를 선언하여 준다. 거리 값을 넣을 때 우선순위 큐에는 큰 순서대로 정렬이 되므로 음수로 넣어 정렬해 주고 출력 역시 음수를 붙여 출력한다.*
2. *우선순위 큐에 담겨 있는 노드와 해당 출발 노드와의 간선 거리를 cost와 here에 저장 한 후 우선순위 큐에서 해당 노드를 삭제해 준다.*
3. *Here에 저장된 노드와 인접한 노드들을 검사하여 출발 노드에서 해당 노드까지 가는 거리 최소값을 구한다. Dist[i]는 출발 노드에서 해당 노드(i)까지 가는 최단 거리를 담는 변수로써 현재 최소값을 담고 있다 이 Dist[i]의 값과 새로 경유해서 가는 값인 nextDist를 비교하여 nextDist 이 더 작다면 Dist[i]값을 갱신하여 준다.*
4. *해당 과정을 우선순위 큐에 노드가 없을 때까지 반복한다.*