**최종 프로젝트 보고서**

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1. 프로젝트 목표, 실험 환경에 대한 설명**

**1) 프로젝트 목표:** 사용자로부터 별의 개수, 별의 좌표 값을 입력 받아 크루스칼 알고리즘을 통한 최소 비용 신장 트리를 생성하고, 이것을 바탕으로 밤하늘에 MST 별자리를 그리는 프로젝트를 만들고자 했습니다.

**2) 실험 환경:** Visual studio를 기반으로 OpenFrameworks를 사용하여 프로젝트를 수행했습니다.

**2. 각 변수에 대한 설명**

* **struct Star:** 별의 좌표를 저장하는 구조체입니다. x는 별의 x 좌표, y는 별의 y 좌표를 나타냅니다.
* **struct Edge:** 연결된 두 별의 인덱스를 저장하는 구조체입니다. first는 첫 번째 별의 인덱스, second는 두 번째 별의 인덱스를 나타냅니다.
* **struct Constellation:** 별자리의 정보를 저장하는 구조체입니다. numStars는 별의 개수, stars는 별들의 좌표 정보를 담고 있는 벡터, edges는 연결된 별들의 인덱스 정보를 담고 있는 벡터, costs는 별들 간의 거리 정보를 담고 있는 벡터, parent는 크루스칼 알고리즘에서 사용되는 부모 노드 정보를 담고 있는 벡터, rank는 크루스칼 알고리즘에서 사용되는 랭크 정보를 담고 있는 벡터, minCost는 최소 비용을 나타내는 변수입니다.
* **twinklingSpeed (float):** 반짝임 효과의 속도를 나타내는 변수입니다. update 함수에서 사용됩니다.
* **twinklingAlpha (float):** 별이 은은하게 반짝이는 효과를 위해 사용되는 알파 값입니다. update 함수에서 시간에 따라 조정됩니다.
* **offsetX (float):** 별자리를 그릴 때 x축으로 이동할 거리를 나타내는 변수입니다. draw 함수에서 사용됩니다.
* **offsetY (float):** 별자리를 그릴 때 y축으로 이동할 거리를 나타내는 변수입니다. draw 함수에서 사용됩니다.
* **offsetS (int):** 별자리 그리기 후 다음 줄로 이동할 때 사용되는 간격을 나타내는 변수입니다. draw 함수에서 사용됩니다.
* **constellations (vector<Constellation>):** 생성된 별자리를 저장하는 벡터입니다. createConstellation 함수에서 별자리가 추가됩니다. draw 함수에서 별자리를 그릴 때 사용됩니다.
* **newConstellation (Constellation):** 새로 생성되는 별자리를 나타내는 객체입니다. createConstellation 함수에서 사용됩니다.
* **numStars (int):** 사용자로부터 입력받은 별의 개수입니다. createConstellation 함수에서 사용됩니다.
* **stars (vector<ofVec2f>):** 각 별의 좌표를 저장하는 벡터입니다. createConstellation 함수에서 사용됩니다.
* **distances (vector<pair<float, pair<int, int>>>):** 모든 별 좌표 쌍 간의 거리와 해당 별 좌표 쌍을 저장하는 벡터입니다. createConstellation 함수에서 사용됩니다.
* **costs (vector<float>):** 각 별 좌표 쌍 간의 거리를 저장하는 벡터입니다. createConstellation 함수에서 사용됩니다.
* **parent (vector<int>):** 각 별의 부모 노드를 나타내는 배열로, 최소 비용 신장 트리를 구성하기 위한 크루스칼 알고리즘에서 사용됩니다. createConstellation, makeSet, findSet, unionSets 함수에서 사용됩니다.
* **rank (vector<int>):** 각 별의 랭크를 나타내는 배열로, 최소 비용 신장 트리를 구성하기 위한 크루스칼 알고리즘에서 사용됩니다. createConstellation, unionSets 함수에서 사용됩니다.
* **numEdges (int):** 현재까지 추가된 간선의 수를 나타내는 변수로, 최소 비용 신장 트리를 구성하기 위한 크루스칼 알고리즘에서 사용됩니다. createConstellation 함수에서 사용됩니다.
* **idx (int):** distances 벡터의 인덱스를 나타내는 변수로, 최소 비용 신장 트리를 구성하기 위한 크루스칼 알고리즘에서 사용됩니다. createConstellation 함수에서 사용됩니다.

**3. 각 함수에 대한 설명**

* **setup():** 애플리케이션 초기화 함수입니다. 배경색을 검정색(0)으로 설정하고, 원의 해상도를 50으로 설정합니다. twinklingSpeed 변수에 반짝임 효과의 속도를 설정하고, "Welcome to the MST constellation drawing!" 메시지를 출력한 후 createConstellation() 함수를 호출하여 별자리를 생성합니다.
* **update():** 애플리케이션의 업데이트 로직을 처리하는 함수입니다. twinklingAlpha 값을 은은하게 반짝이는 효과를 만들기 위해 조정합니다. elapsedTime 변수에 경과 시간을 저장하고, sin(elapsedTime)의 값에 따라 twinklingAlpha 값을 -1에서 1 사이로 매핑하여 알파 값의 변화를 조정합니다.
* **draw():** 애플리케이션의 렌더링 로직을 처리하는 함수입니다. 검정색으로 배경을 설정하고, constellations 벡터에 저장된 각 별자리를 그립니다. constellation.stars에 저장된 각 별에 대해 반짝이는 알파 값을 적용하여 drawStar() 함수를 호출하여 별을 그립니다. star.x와 star.y 값을 기반으로 실제 좌표를 계산하고, offsetX와 offsetY 값만큼 이동하여 그립니다. constellation.edges에 저장된 각 간선에 대해 실제 좌표를 계산하고, offsetX와 offsetY 값만큼 이동하여 간선을 그립니다. constellation.minCost 값을 출력합니다.
* **keyPressed(int key):** 사용자의 키 입력을 처리하는 함수입니다. 'a' 키를 누르면 createConstellation() 함수를 호출하여 별자리를 추가 생성합니다. 'r' 키를 누르면 resetConstellations() 함수를 호출하여 별자리를 초기화합니다. 'q' 키를 누르면 ofExit() 함수를 호출하여 애플리케이션을 종료합니다.
* **void createConstellation():** 새로운 별자리를 생성하는 함수입니다. 먼저 newConstellation 객체를 생성하여 새로운 별자리를 초기화합니다. 사용자로부터 별의 개수를 입력받습니다. 만약 입력된 값이 1보다 작거나 같으면 "Wrong input. Please enter a value greater than 1." 메시지를 출력하고 함수를 종료합니다. 사용자로부터 각 별의 좌표를 입력받습니다. 입력된 좌표는 newConstellation.stars 벡터에 추가됩니다. 모든 별의 좌표 쌍 간의 거리를 계산합니다. 이를 위해 distances 벡터에 거리와 해당 별 좌표 쌍을 저장합니다. 또한, 거리 값을 newConstellation.costs 벡터에 추가합니다. 거리를 기준으로 distances 벡터를 정렬합니다. 크루스칼 알고리즘을 사용하여 최소 비용 신장 트리를 구성합니다. 이를 위해 newConstellation.parent와 newConstellation.rank 배열을 초기화합니다. 신장 트리를 완성하기 위해 distances 벡터를 순회하면서 간선을 추가합니다. 두 정점이 다른 집합에 속해 있는 경우에만 간선을 추가하고, 두 집합을 합칩니다. 추가된 간선의 수와 최소 비용도 업데이트합니다. 생성된 별자리를 constellations 벡터에 추가합니다. "New constellation has been created!" 메시지를 출력합니다.
* **resetConstellations():** 별자리를 초기화하는 함수입니다. constellations 벡터를 비워서 저장된 모든 별자리를 삭제합니다.
* **makeSet(Constellation& constellation, int v):** 주어진 별자리 constellation에서 정점 v를 자체적인 집합으로 설정합니다. 이 함수는 초기 랭크를 0으로 설정합니다.
* **findSet(Constellation& constellation, int v):** 주어진 별자리 constellation에서 정점 v가 속한 집합의 루트(대표 정점)를 찾습니다. 재귀적으로 부모 노드를 따라가며 루트를 찾아가는 경로 압축 최적화도 수행합니다.
* **unionSets(Constellation& constellation, int a, int b):** 주어진 별자리 constellation에서 정점 a와 b가 속한 두 집합을 합칩니다. 랭크를 기반으로 한 최적화를 수행하여 짧은 트리를 긴 트리에 연결합니다.
* **drawStar(float x, float y, float radius, int numPoints):** 별을 그리는 함수입니다. 주어진 중심 좌표 (x, y)와 반지름 radius를 기반으로 별을 그립니다. numPoints는 별의 각 꼭짓점 수입니다. 별의 내접원과 외접원 크기를 번갈아 가며 설정하여 별의 형태를 만듭니다.

**4. 프로젝트 전체 플로우차트 + 자료구조 및 알고리즘 + 시간/공간 복잡도 상세히 설명.**

**1) 플로우 차트**

**스크린샷, 흑백, 블랙, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**2) 자료구조:**

* **struct Constellation:** Constellation은 별자리를 나타내는 구조체입니다. 다음과 같은 멤버 변수를 포함합니다:

int numStars: 별의 개수를 저장하는 변수입니다.

vector<pair<float, float>> stars: 각 별의 좌표를 저장하는 벡터입니다.

vector<pair<int, int>> edges: 최소 비용 신장 트리를 구성하는 간선들을 저장하는 벡터입니다.

vector<int> parent: Union-Find 연산에서 사용되는 부모 노드를 저장하는 배열입니다.

vector<int> rank: Union-Find 연산에서 사용되는 노드의 랭크(트리의 깊이)를 저장하는 배열입니다.

vector<float> costs: 각 별 좌표 쌍 간의 거리를 저장하는 벡터입니다.

float minCost: 최소 비용 신장 트리의 비용을 저장하는 변수입니다.

* **vector<pair<float, pair<int, int>>> distances:** distances는 각 별의 좌표 쌍 간의 거리와 해당 별 좌표 쌍을 저장하기 위한 벡터입니다. pair<float, pair<int, int>> 형식으로 구성되며, 첫 번째 요소는 거리이고 두 번째 요소는 해당 별 좌표 쌍의 인덱스를 나타냅니다.

**3) 알고리즘:**

**크루스칼 알고리즘(Kruskal's Algorithm):** 최소 신장 트리(Minimum Spanning Tree)를 찾기 위한 그래프 알고리즘 중 하나입니다. 이 알고리즘은 간선의 가중치를 기준으로 그래프의 모든 정점을 최소 비용으로 연결하는 부분 그래프를 만들어줍니다.

크루스칼 알고리즘의 동작 방식:

그래프의 간선들을 가중치 순서로 정렬합니다.

가장 가중치가 작은 간선부터 선택합니다.

선택한 간선의 양 끝 정점이 현재까지 선택한 간선들에 의해 연결되어 있지 않다면, 해당

간선을 선택하고 최소 신장 트리에 추가합니다. 그렇다면 두 정점은 같은 집합으로 묶입니다.

위의 과정을 모든 간선에 대해 반복합니다. 단, 추가할 간선의 수가 (정점의 수 - 1)개가 될

때까지 진행합니다.

**Union-Find 알고리즘:** 크루스칼 알고리즘을 효율적으로 구현하기 위해 Union-Find 알고리즘이 사용됩니다. 이 알고리즘은 서로소 집합(disjoint-set)을 표현하고 조작하는 효율적인 자료 구조입니다.

Union-Find 알고리즘의 주요 연산:

MakeSet(x): 요소 x를 포함하는 새로운 집합을 생성합니다.

Find(x): 요소 x가 속한 집합의 대표값(루트)을 반환합니다. 대표값은 각 집합의 루트 요소

를 가리킵니다.

Union(x, y): 요소 x가 속한 집합과 요소 y가 속한 집합을 합칩니다. 두 집합의 대표값을

비교하여 다른 경우 한 집합을 다른 집합의 하위 집합으로 합칩니다.

크루스칼 알고리즘에서 Union-Find 알고리즘을 활용하여 간선을 선택하고 추가할 때, Find 연산을 사용하여 두 정점이 이미 같은 집합에 속해 있는지 확인하고 Union 연산을 통해 두 정점을 같은 집합으로 합칩니다. 이를 통해 최소 신장 트리를 구성하면서 사이클을 방지할 수 있습니다. 크루스칼 알고리즘과 Union-Find 알고리즘은 각각 독립적으로 구현될 수 있으며, 크루스칼 알고리즘에 Union-Find 알고리즘을 활용하는 방식으로 효율적인 실행이 가능합니다.

**4) 시간 복잡도:**

별자리 생성(createConstellation() 함수): 주요 부분은 크루스칼 알고리즘입니다. 별의 개수를 N이라고 할 때, 간선을 정렬하는 시간 복잡도는 O(NlogN)입니다. 크루스칼 알고리즘을 적용하는 시간 복잡도는 O(ElogE)입니다. 별의 개수에 비해 간선의 개수는 대체로 작을 것으로 예상되므로, 간선을 정렬하는 시간이 주요 요소가 될 것입니다. 따라서 전체적인 시간 복잡도는 **O(NlogN)**입니다.

**5) 공간 복잡도:**

별자리 생성(createConstellation() 함수): 주요 요소는 stars 벡터와 edges 벡터입니다. 별의 개수에 비례하여 stars 벡터의 공간이 할당되며, 간선의 개수에 비례하여 edges 벡터의 공간이 할당됩니다. 따라서 공간 복잡도는 **O(N + E)**입니다.

**5. 창의적 구현 항목에 대한 설명**

maze 수업 시간에는 가중치가 없는 그래프에서 DFS, BFS 알고리즘의 적용을 배웠기에 개인 프로젝트에서는 가중치가 있는 그래프에 적용될 수 있는 알고리즘을 구현해보고 싶었습니다. 그중 Kruskal 알고리즘을 이용해 그래프의 MST를 구하는 코드를 작성하고자 했습니다. min heap대신 distances 벡터를 정렬하는 방식을 사용했고 disjoint set의 union, find 함수를 이용했습니다. MST를 시각적으로 표현했을 때 어떤 주제가 보기도 좋고 구현하는 재미가 있을지 생각해보다가 별자리를 떠올렸습니다. 이러한 아이디어들을 조합해 사용자로부터 별의 개수, 별의 좌표 값을 입력 받아 각 좌표의 거리를 계산하고 크루스칼 알고리즘을 통해 MST를 구한 후, 이것을 바탕으로 밤하늘에 MST 별자리를 그리는 프로젝트를 만들고자 했습니다.

**6. 프로젝트 실행 결과 캡쳐**

스크린샷, 텍스트, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

[초기 실행 화면]

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[사용자 입력]

스크린샷, 텍스트, 컴퓨터, 블랙이(가) 표시된 사진

자동 생성된 설명

[사용자 입력에 따른 MST 별자리 생성]

텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

[‘a’ key 입력 후 별자리 여러 개 생성]

스크린샷, 블랙이(가) 표시된 사진

자동 생성된 설명

[‘r’ key 입력 시 밤하늘 초기화]

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[‘q’ key 입력 시 프로그램 종료]

**7. 느낀 점 및 개선 사항**

코드를 작성하며 예비보고서에서 pseudo 코드를 작성하는 절차가 매우 중요했다는 것을 깨달았다. pseudo 코드 없이 머리만 굴려서 0부터 코드를 작성하려니까 매우 힘들어서 중간에 다시 알고리즘과 자료구조를 공부하고 pseudo 코드를 완성한 후 구현을 시작했다. 또한 크루스칼 알고리즘을 구현하는 것보다. 별 모양 그리는 함수를 구현할 때 좌표와 각도 계산하는 코드를 작성하는게 더 힘들었다는 점에서 오픈프레임워크에 대한 공부가 아직 부족하다는 것을 느꼈다. 이제까지 실습에서는 짜여진 템플릿에 코드를 채워넣는 느낌이라 그렇게 막막하지 않았는데 이번 최종 프로젝트에서는 직접 프로젝트를 기획하고, 알고리즘을 구현한다는 점에서 힘들기도 했지만 재미있었다. 개선해야할 점은 사용자와 상호작용하는 기능을 좀더 추가하면 더욱 재밌는 프로젝트가 될 것 같다.