* Unordered\_map을 사용하신 이유가 뭔가요?
* Unordered map은 탐색속도가 상수시간이고 순서가 중요하지 않다는 장점이 있는데 이 경우에는 정수형인 ID를 사용하여 탐색을 하는 경우가 많았고 ID의 순서가 중요하지 않기에 사용했습니다.
* 단점은 없나요?

해시 충돌로 인해 성능이 저하될 수 있고 버킷으로 인해 메모리 사용량이 늘어난 다는 단점이 있습니다.

* Unordered map은 내부적으로 어떻게 구현되어 있는지 아시나요?

해시 테이블로 구현되어 있습니다.

해시 테이블이란 Key값을 해싱하여 나온 해쉬 값에 의해 적절한 버킷에 저장하는 것을 해시 테이블 구조라고 합니다.

* 해싱이란 ?

Key를 통해 특정 값을 얻어내는 작업을 말합니다.

그리고 그 값을 Value를 저장할 공간의 인덱스로 사용하는 것 입니다.

저장되는 공간을 버킷 이라고 부릅니다.

* Unordered\_map은 탐색에 있어 항상 map보다 우수한가요?

아니라고 알고있습니다.

자세히는 모르지만 긴 문자열이 Key로 설정 되었을 때는 키를 해싱하는 작업이 map에서 문자열을 비교하는 것 보다 성능적으로 좋지 못하다고 알고있습니다.

그리고 일정 개수를 넘어갈 경우 해시 충돌이 자주 발생하여 map이 더 좋은 경우가 있습니다.

* Unordered\_map은 시간복잡도가 항상 상수시간인가요?

최악의 경우 체이닝 방식에서 n이의 시간복잡도가 나올 수 있습니다.

체이닝이란 해시 충돌이 발생하면 해당 인덱스에 추가로 이어 저장하는 방법인데 한 인덱스에 모든 데이터가 이어서 저장되어 있고 가장 끝에 찾으려는 값이 존재한다면 n이 될 수 있습니다.

* 충돌을 해결하는 다른 방식은 없나요?

개방 번지화 라는 방식이 있습니다.

해시 충돌이 발생하면 비어 있는 인덱스를 찾아 value를 할당하는 방법입니다.

* 개방 주소법의 종류에 대해 아시나요?

선형 탐사, 제곱 탐사, 이중 해싱이 있습니다.

선형 탐사란 충돌 시 다음 인덱스를 확인해 가며 비어있는 공간을 찾는 방식이고

제곱 탐사란 이차식을 통해 인덱스를 찾는 폭을 구해 비어있는 공간을 찾는 방식입니다.

이중 해싱이란 해시 충돌 발생 시 다른 해시 함수를 사용하여 증가 폭을 구하고 그만큼 인덱스를 증가시키며 비어있는 곳을 탐색하는 방식입니다.

* Map에 대해서 설명해보세요

Key와 Value로 된 쌍을 저장하는 컨테이너이며 자료를 자동 정렬하기 때문에 정렬이 필요한 경우 사용하며 키를 새로 추가할 때마다 정렬된 위치를 찾아 추가해야 하기 때문에 대용량 데이터를 저장하는 경우 시간이 많이 소요된다는 단점이 있습니다.

* 대용량 데이터를 저장하는 경우 시간이 많이 소요된다고 하셨는데 최악의 경우 시간복잡도는 어떻게 되나요?

Log n입니다.

* Map은 내부적으로 어떻게 구현되어 있는지 아시나요?

레드블랙트리로 구현되어 있습니다.

* 레드블랙 트리란 무엇인가요?

균형 이진 탐색 트리의 한 종류로 이진 탐색 트리에서 한쪽으로 데이터가 몰려 최악의 경우 n의 탐색시간이 걸릴 수 있다는 단점을 보완하기 위한 자료구조입니다.

* 좀 더 자세히 설명하실 수 있나요?

레드 블랙 트리는 5가지 조건을 가지고 구현합니다.

모든 노드는 블랙 또는 레드 색상을 가지며

루트 노드는 검정색이여야 하고 리프 노드는 검정색이여야 합니다.

그리고 빨간 노드의 자식은 모두 검정색이지만 검정 노드의 자식이 빨강인 필요는 없고

루트 노드에서 무든 잎 노드 사이에 존재하는 검은색 노드의 수는 모두 동일하다는 조건을 가지고 있습니다.

* 이진 탐색 트리란 무엇인가요?

노드의 Key는 중복이 없어야 하며 어떤 노드의 왼쪽 자신 노드는 나보다 작고

오른쪽 자식 노드는 나보다 커야한다는 조건을 만족하는 자료구조 입니다.

단점으로는 트리의 모양이 한쪽으로 치우쳐지게 되면 탐색 성능이 떨어지게 됩니다.

* 이진 탐색이란 무엇인가요?

데이터가 정렬되어 있는 배열에서 특정값을 찾는 알고리즘 입니다.

배열의 중간에 있는 임의의 값을 선택하여 찾고자 하는 값 과 비교하고 중간 값 보다 작으면 좌측의 데이터, 크다면 우측의 데이터들을 대상으로 다시 탐색하는 알고리즘 입니다.

* 트리와 그래프에 대해 아시나요?

그래프란 노드 간을 연결하는 간선으로 구성된 자료구조 이며 계층이 없는 네트워크 모델입니다. 객체간의 관계를 표현할 수 있는 자료구조라는 특징이 있습니다.

순환 사이클 가능

그래프의 종류 중 하나로 노드와 간선으로 이루어진 자료구조 입니다.

계층적인 모델이며 노드간에 단 하나의 간선으로 연결되어 있다는 특징이 있습니다.

순환 사이클 불가능

* 트리와 그래프의 차이점에 대해 아시나요?

트리와 다르게 그래프는 루트 노드나 부모 자식 관계라는 개념이 없고 노드 간에 2개 이상의 경로가 가능하다는 차이점이 있습니다.

공통점으로는 노드 사이를 연결하는 간선으로 구성된 자료구조라는 점이 있습니다.

* 깊이 우선탐색

루트 노드에서 시작하여 다음 분기로 넘어가기 전에 해당 분기를 완벽하게 탐색하는 방법으로 모든 노드를 방문하고자 하는 경우에 이 방법을 사용합니다.

특징으로는 자기 자신을 호출하는 순환 알고리즘의 형태를 지닌 다는 것과

방문 여부를 반드시 검사해야 한다는 점이 있습니다.

* 왜 검사해야 하나요?

방문 여부를 검사하지 않는다면 무한루프에 빠질 위험이 있기 때문입니다.

* 시간복잡도는 어떻게 되나요?

인접 리스트로 표현된 그래프의 경우 n+e

인접 행렬로 표현된 그래프는 n2입니다.

* 왜 그런지 아시나요?

인접 행렬은 이차원 배열로 구현 되는데 행과 열이 노드의 개수이기 때문입니다.

인접 리스트는 벡터의 배열로 나타내는 방식이며 노드의 개수 만큼의 벡터에 간선의 정보를 저장하므로 n+ 간선의 개수 입니다.

* 너비 우선 탐색에 대해 아시나요?

루트 노드에서 시작하여 인접한 노드를 먼저 탐색하는 방법입니다.

보통 두 노드 사이의 최단 경로 혹은 임의의 경로를 찾고 싶을 때 이 방법을 선택 합니다.

특징으로는 재귀적으로 동작하지 않으며 그래프 탐색의 경우 방문여부를 검사해야하고

방문한 노드를 차례로 저장한 후 꺼낼 수 있는 자료 구조인 큐를 사용합니다.

* 시간 복잡도에 대해 아시나요?

인접 행렬의 경우 n2

인접 리스트로 구현한 경우 n + 간선의 개수입니다.

* 두가지는 어떨 때 사용하나요?

경로의 특징을 저장해 줘야하는 문제는 DFS를 사용하고

최단거리를 구하거나 검색 시작 지점부터 원하는 대상 사이의 거리가 멀지 않다면

BFS를 사용합니다.

* Mmo Rpg 개발을 위해 IOCP를 사용하셨다고 했는데 어떤 점에서 좋다고 생각했나요?

IOCP 객체 내부의 스레드 풀을 사용하기 때문에 스레드 생성 파괴 오버헤드를 줄여 효율적인 스레드 관리가 가능하고 그렇기 때문에 적은 수의 스레드로 많은 연결을 할 수 있기 때문입니다.

스레드가 적다는 것은 Context switching 비용이 적다는 것을 의미하기도 합니다.

* GQCS에서 어떤 내용이 있는지에 대해 어떻게 판단하셨나요?

오버렙드 구조체를 확장하여 사용하였습니다.

Send recv를 판단하는 정보와 buffer의 위치를 추가하였습니다.

* ID는 필요하지 않았나요?

아니요 필요했지만 확장하여 사용하지 않았습니다.

IOCP에서 오고 가는 completion\_key에 id를 담아 사용했습니다.

* Buffer의 위치는 왜 알아야 하나요?

SEND일 경우 어떤 버퍼에서 완료되었는지 알 수 없는데 재사용을 위해서는 위치를 알아야 합니다.

* 좀더 자세히 설명해 주실 수 있나요?

기본적으로 Overlapped IO (callback)으로 동작합니다.

그리고 IOCP 객체를 통해 관리되는데 이는 커널에서 관리하며 소켓을 IOCP에 등록하여 사용합니다.

전반적인 실행의 순서는

비동기 함수(AcceptEX, WSASend, WSARecv)를 실행하여 비동기 입출력이 종료되면 완료된 내용을 IOCP객체에 쌓게 됩니다.

그러면 작업이 끝난 스레드에서 GetQueuedCompletionStatus를 호출하여 완료된 내용을 가져와 사용하게 됩니다.

* 데이터가 잘리는 경우가 있는가?

네 있습니다 (부하가 적으면 데이터가 띄엄띄엄 가고 부하가 커지면 잘려오는 등 제대로 동작하지 않는 경우가 있습니다.)

* 그럴 땐 어떻게 처리했는가?

패킷 재조립 방식을 사용했습니다.

* 패킷 재조립이란 무엇인가요?

패킷이 잘못 올 경우를 대비하여 클라이언트 각 객체는 보관용 버퍼와 받은 패킷의 사이즈를 저장하는 변수를 가지고 있으며 패킷이 도착했을 때

받은 패킷의 사이즈와 버퍼의 내용을 담아 놓고 패킷을 완성할 수 있다면 완성 후 패킷을 처리하는 함수로 넘어가는 방식입니다.

완성할 수 없다면 버퍼에 복사해 두고 다음 처리할 데이터에 접근합니다.

* Overlapped io 모델에 대해 아시나요?
* 다른 모델에 대해 아시는 게 있나요?
* Iocp쪽 코드에 모르는 내용들 정리
* 효림이한테 usb 받아서 추가 작성
* 왜 스레드의 개수가 이렇게 많나요?

거의 개발이 완료되었을 때 스레드의 개수를 달리하여 성능 테스트를 진행했고 가장 많은 동시접속자를 확보했기 때문에 사용했습니다.

* 성능 테스트는 어떻게 진행했나요?

더미 클라이언트를 만들어 테스트를 진행 했는데 8코어 PC