배열

1. 정의와 성질

테이블이(가) 표시된 사진

자동 생성된 설명

배열 – 메모리 상에 원소를 연속하게 배치한 자료구조

 원래 C++에서는 예를 들어 int arr[10]; 으로 배열을 선언한 뒤에는 arr의 길이를 변경하는게 불가능하지만, 자료구조로써의 배열에서는 길이를 마음대로 늘리거나 줄일 수 있다고 생각하겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

첫번째, 배열은 메모리 상에 원소를 연속하게 배치한 자료구조라서 k번째 원소의 위치 바로 계산 가능,

시작 주소에서 k칸 만큼 오른쪽으로 가면 되기 때문입니다. 그렇기 때문에 k번째 원소를 O(1)에 확인하거나 변경할 수 있습니다.

두 번째로 메모리는 다른 자료구조와 다르게 추가적으로 소모되는 메모리의 양이 거의 없습니다.

세 번째로 메모리 상에 데이터들이 붙어있으니까 Cache hit rate가 높습니다.

네 번째로 메모리 상에 연속한 구간을 잡아야 하니 할당에서 다소 제약이 있습니다.

임의의 위치에 있는 원소 확인/변경, O(1)

원소의 끝에 추가, O(1):

끝자리에 값을 쓰고 길이를 1 증가시키면 되니 O(1)입니다.

마지막 원소를 제거, O(1):

그냥 길이를 1 줄이면 되니까 O(1)입니다.

임의의 위치에 원소를 추가, O(N):

텍스트, 탑재이(가) 표시된 사진

자동 생성된 설명

지금 보면 5번에 15를 추가했는데, 이렇게 되면 그 뒤의 원소들을 전부 한 칸씩 밀어야해서 O(N)이 필요하게 됩니다. 마치 책장에 책이 연속해서 꽂혀있을 때 중간에 책을 넣기 위해서는 나머지 책들을 밀어야하는 것과 같은 상황입니다.

임의의 위치에 원소 제거, O(N):



그리고 임의의 위치에 원소를 제거하는 것도 O(N)이 됩니다. 2번 원소를 지웠다고 치면 그 이후에 있던 모든 원소들을 한 칸씩 땡겨와야해서 그렇습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

배열을 초기화 시키는 팁

텍스트이(가) 표시된 사진

자동 생성된 설명

memset 함수를 활용하는 방식입니다. 그런데 memset 함수는 실수할 여지가 굉장히 많습니다. 예를 들어 0이나 -1이 아닌 다른 값을 넣으면 오동작한다거나, 2차원 이상의 배열을 함수의 인자로 넘겨 그 곳에서 memset을 하면 잘못 들어간다거나 하는 점들이 있습니다. 그래서 memset은 정말 비추천합니다.

두 번째로는 그냥 for문을 돌면서 값을 하나하나 다 바꾸는 방식이고, 코드가 조금 투박하긴 하지만 실수할 여지가 없기 때문에 무난하고 좋습니다.

마지막 방식은 algorithm 헤더의 fill 함수를 이용하는 것이고, fill 함수는 실수할 여지도 별로 없고 코드도 짧으니 익숙해진다면 가장 추천하는 방식입니다.

fill(a,a+21,0);

텍스트이(가) 표시된 사진

자동 생성된 설명

V2.insert(v2.begin()+1,3); // 인덱스 번호, data, O(N)

V3.earse(v3.begin()+1,3); // 인덱스 번호, 지울 data, O(N)

V4.push\_back(); // 맨뒤 원소 추가, O(1)

V4.pop\_back(); // 맨뒤 원소 삭제, O(1)

V4.clear(); // 벡터 전체 비우기, O(N)

vector에서 = 사용하면 deep copy 발생

텍스트이(가) 표시된 사진

자동 생성된 설명

금은 값을 바꾸지 않고 그냥 출력만 해서 별 상관이 없지만, 만약 int e : v1이라고 하면 복사된 값이 e에 들어가고

int& e : v1이라고 하면 원본이 e에 들어갑니다.

 3번처럼 쓰면 아주 잘못될 수 있습니다.

기본적으로 vector의 size 메소드는 시스템에 따라 unsigned int 혹은 unsigned long long을 반환합니다. 그렇기 때문에 32비트 컴퓨터 기준 3번같이 쓰면 v1이 빈 vector일 때 v1.size() - 1이 unsigned int 0에서 int 1을 빼는 식이 되고, unsigned int와 int를 연산하면 unsigned int로 자동 형변환이 발생하기 때문에 (unsigned int)0 - (int)1은 -1이 아니라 4294967295이 되어버립니다. 4294967295이라는 이상한 값은 unsigned int overflow로 인해 생기게 된 값입니다.

그래서 아무것도 출력을 하지 않고 종료되는 것이 아닌, v1[0], v1[1], v1[2], ...  이렇게 i가 계속 커지다가 어느 순간 런타임에러가 발생하게 될 것입니다.

배열은 데이터를 자주 바꾸지 않고 그냥 쌓아두고 싶을 때 활용할 수 있습니다. 그리고 사실 거의 대부분의 문제에서 일단 입력값을 저장해놓고 시작하는 일이 많기 때문에 입력을 담아두기 위해 배열을 사용하곤 합니다.

그래서 단지 데이터를 쌓아두는 용도 말고 인덱스에 해당하는 원소를 빠르게 접근하는 목적으로 배열을 사용하면 효율적인 문제를 소개하겠습니다.

#define for2(i,j,r,c) for(int i=0;i<r;++i)for(int j=0;j<c;++j)

뭔가 좋아보임