너비 우선 탐색(BFS)

큐를 이용해서 순환적 형태로 구성하자.

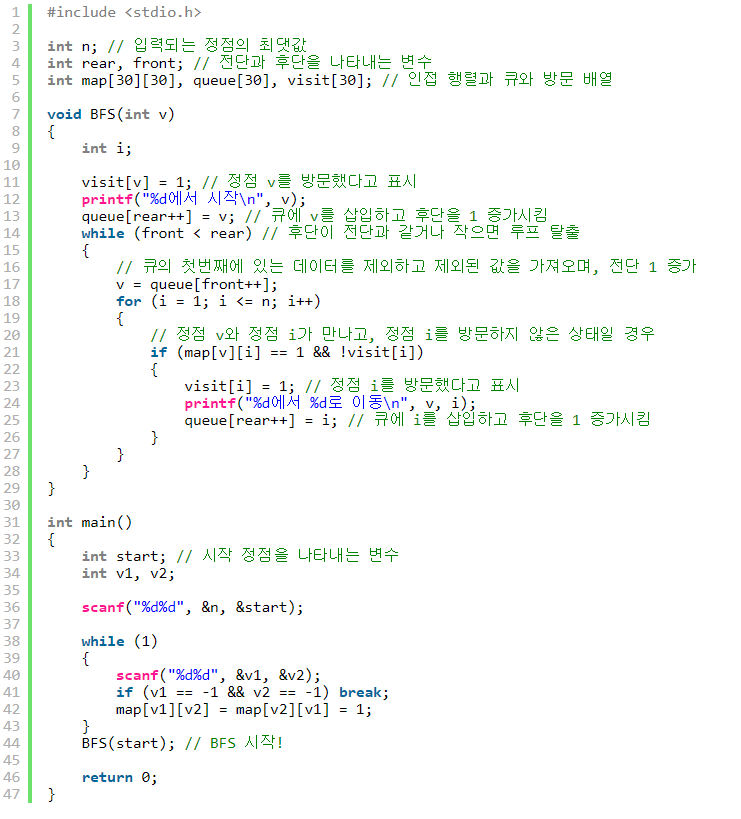
1) 시작 정점을 '방문함' 으로 표시하고 큐에 Enqueue합니다.

2) 큐로부터 정점을 Dequeue 합니다. 반환된 정점이 인접한 정점중 아직 방문하지 않은 곳들을 '방문함'으로 표시하고 큐에 Enqueue합니다.

3) 큐에 정점이 없을 때까지 2) 를 반복하고, 큐에 정점이 없다면 탐색을 종료합니다.

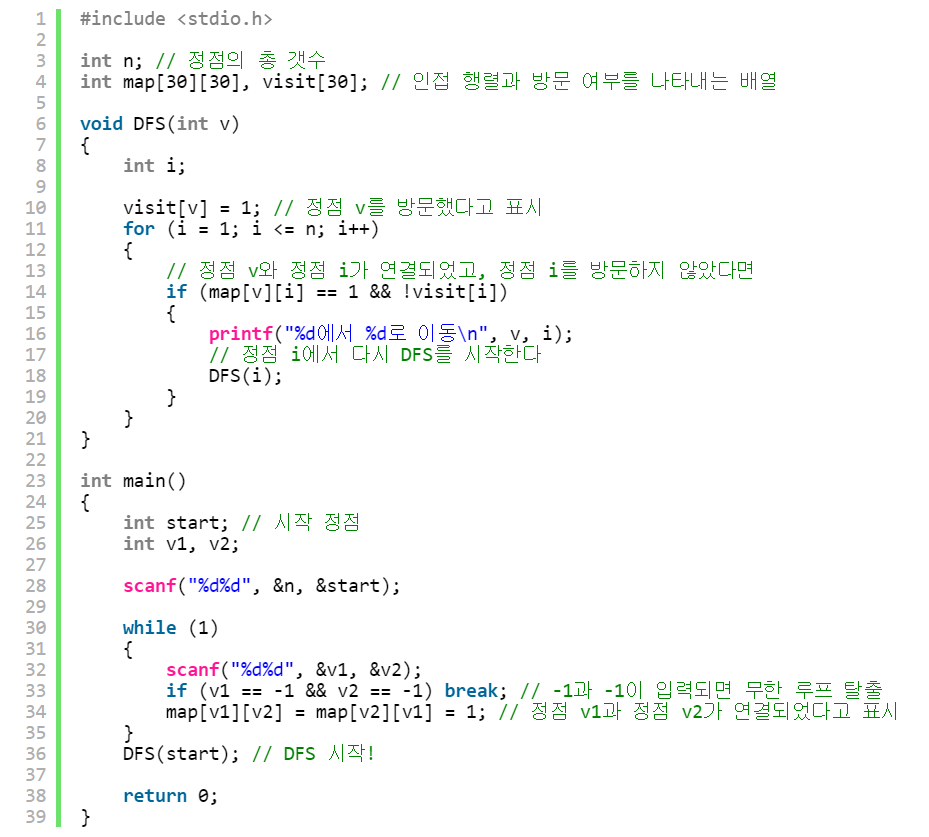
이 내용을 토대로 예를 한번 따라가 보시기 바랍니다.

출처: <http://pooh-explorer.tistory.com/43> [푸의 개발 이야기]



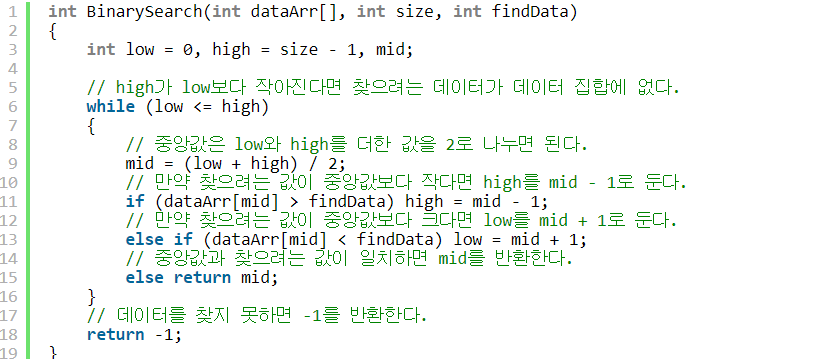
깊이 우선 탐색

스택을 이용한다

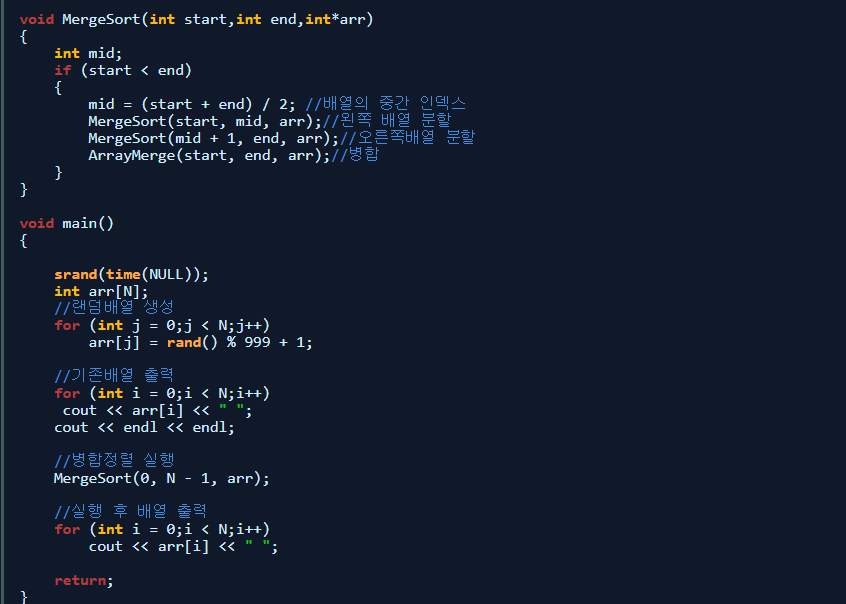
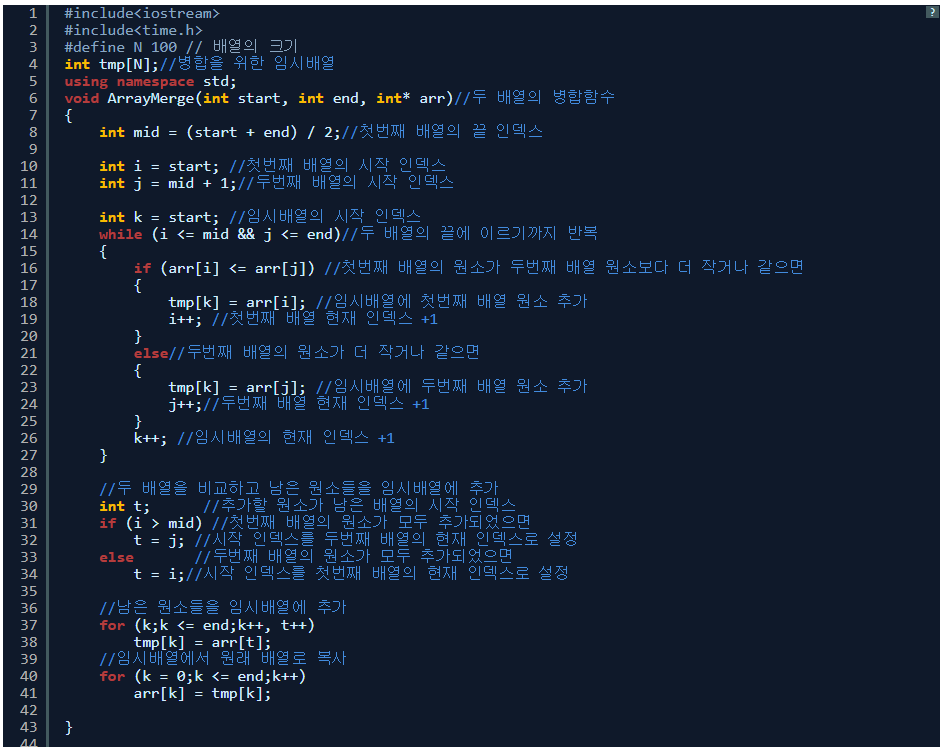


이진 탐색

정렬된 행렬을 기본으로 수행한다.



병합 정렬



퀵 정렬

void quickSort(int numbers[], int array\_size);

void q\_sort(int numbers[], int left, int right);

int main(int argc, char \*\*argv)

{

const int itemSize = 6;

int array[itemSize] = {3,8,0,2,1,4};

quickSort(data, itemSize);

}

void quickSort(int numbers[], int array\_size)

{

q\_sort(numbers, 0, array\_size -1);

}

void q\_sort(int numbers[], int left, int right)

{

int pivot, l\_hold, r\_hold;

l\_hold = left;

r\_hold = right;

pivot = numbers[left]; // 0번째 원소를 피봇으로 선택

while (left < right)

{

// 값이 선택한 피봇과 같거나 크다면, 이동할 필요가 없다

while ((numbers[right] >= pivot) && (left < right))

right --;

// 그렇지 않고 값이 피봇보다 작다면,

// 피봇의 위치에 현재 값을 넣는다.

if (left != right)

{

numbers[left] = numbers[right];

}

// 왼쪽부터 현재 위치까지 값을 읽어들이면서

// 피봇보다 큰 값이 있다면, 값을 이동한다.

while ((numbers[left] <= pivot) && (left < right))

left ++;

if (left != right)

{

numbers[right] = numbers[left];

right --;

}

}

// 모든 스캔이 끝났다면, 피봇값을 현재 위치에 입력한다.

// 이제 피봇을 기준으로 왼쪽에는 피봇보다 작거나 같은 값만 남았다.

numbers[left] = pivot;

pivot = left;

left = l\_hold;

right = r\_hold;

// 재귀호출을 수행한다.

if (left < pivot)

q\_sort(numbers, left, pivot - 1);

if (right > pivot)

q\_sort(numbers, pivot+1, right);

}

트리 삽입/탐색

이진 트리의 경우

노드를 삭제할 때

1. 노드가 단말 노드인 경우

* 부모노드를 찾아서 부모 노드의 연결을 NULL로 만들어주면 된다.

1. 노드의 자식노드가 1개 있는 경우

* 자식 노드를 부모노드에 붙여주면 된다.

1. 두개 서브트리 모두 가지고 있는 경우

* 왼쪽에서 제일 큰 노드 탐색 or 오른쪽에서 제일 작은 노드 탐색해서

원래 부모노드와의 연결은 끊고 삭제된 노드 위치에 삽입