자료구조 과제

#1 : N-Queen Problem

컴퓨터 공학부

20151523 김동현

1. Backtracking에 대한 설명

Backtracking이란 퇴각검색으로 조건 검색을 해야 하는 알고리즘에서 작업시간을 줄이는데 효과적인 방법이다. 더 이상 조건에 맞지 않아 진행 불가능한 노드에서 더 나아가지 않고, 되돌아가서 다시 탐색하는 것이 그 개념이다. 4-Queen Problem에서 그 개념을 알아보자면, 퀸이 모든 체스 좌좌표에 존재할 때 가능한지, 불가능한지를 판단하기 위해서라면 원래라면 모든 위치에서 가능성을 따져 봐야한다. 그러면 각 열마다 1개의 퀸이 위치할 수 있다하면 4\*4\*4\*4 = 256개의 경우의 수를 고려해야한다. N이 증가할수록 그 양은 아주 커지게 된다. Backtracking을 적용하면 만일 (0,0)에 퀸이 놓였을 경우 (n, 1)을 탐색할 때 (1, 1), (2, 1)은 퀸이라는 체스말의 조건상 불가능한 위치가 된다. 따라서 이 경우에 도달하면 다시 n의 결정 노드로 돌아가서 그 다음 위치에서 탐색을 하게 되어서 탐색을 하는 양을 크게 줄여준다.

1. 코드

#include <stdio.h>

#include <stdlib.h>

int backtracking(int n, int q\_col[]) {

int i;

for (i = 1; i < n; i++) {

if (q\_col[i] == q\_col[n] || abs(q\_col[i] - q\_col[n]) == abs(i - n))

return 0;

}

return 1;

}

int searching(int n, int i, int q\_col[]) {

int j = 0;

int flag = 0;

if (!backtracking(i, q\_col))

return;

if (i == n) {

for (j = 1; j <= n; j++)

printf("%d ", q\_col[j] - 1);

flag = 1;

}

else {

for (j = 1; j < n + 1; j++) {

q\_col[i + 1] = j;

flag = searching(n, i + 1, q\_col);

if (flag)

return;

}

}

return flag;

}

int main() {

int\* q\_col;

int n, i, result;

i = 0;

result = 0;

printf("Input N : ");

scanf("%d", &n);

q\_col = (int\*)malloc(sizeof(int) \* n + 5);

result = searching(n, i, q\_col);

if (!result)

printf("No solution");

free(q\_col);

return 0;

}

1. 자료구조 및 공간 복잡도

이 문제를 해결하기위해 동적할당한 배열을 사용하였다. 입력으로 n을 받아서

q\_col = (int\*)malloc(sizeof(int) \* n) 해주었다.

Searching 함수를 살펴보면, 한번의 call 당 j, flag, n-1번의 두번의 반복문 속에서 q\_col이 참조되고, backtracking 함수의 호출과 searching 함수의 재귀 호출이 있다. Backtracking 함수 내에는 i, n-1번의 반복문 속에서 q\_col이 참조되므로 S(n) = (n-1) + 1 = n이고, 다시 searching 함수를 계산해보면 call 당 S(n) = 2 + (n-1) \* 2 + n + 1= 3n + 1의 공간 복잡도가 필요하고, 재귀호출의 depth는 발견할 때까지의 횟수이다.

1. 알고리즘 및 시간 복잡도

Backtracking 함수를 통해 그 위치에서 진행할지 검사를 해주고, searching 함수에서 인덱스를 늘리며 재귀호출을 통하여 i가 n에 도달한 경우 solution을 발견했으므로 출력 후 종료하고, 아직 n에 도달하지 못한 경우 해당 g\_col 배열원소에서 1칸씩 증가시키며 재귀 호출을 하는 알고리즘이다. Backtracking 함수의 시간복잡도는 O(n)이고, searching 함수의 시간복잡도는 n인 경우를 예로 들어서 생각을 해보면, 첫번째 column에 첫 퀸이 오는 경우는 n가지, 그 다음 두번째 column에 퀸이 오는 경우는 첫 퀸의 같은 행과 대각줄을 제외한 n-2, 그 다음 세번째 column에 퀸이 오는 경우는 n-4…. 마지막은 1가지 이런 식으로 되므로 이 값들을 곱한 것을 f(n) (f(n) <= n!) 이라고 하면 f(n)번 search 함수가 호출되므로 searching 함수의 시간복잡도는 O(n) + O(n!)이므로 이 프로그램의 시간복잡도는 O(n!)이라고 할 수 있다.

1. 프로그램 실행 모습

시계이(가) 표시된 사진

자동 생성된 설명

<n = 4>

시계이(가) 표시된 사진

자동 생성된 설명

<n = 3>

개체, 시계이(가) 표시된 사진

자동 생성된 설명

<n = 10>