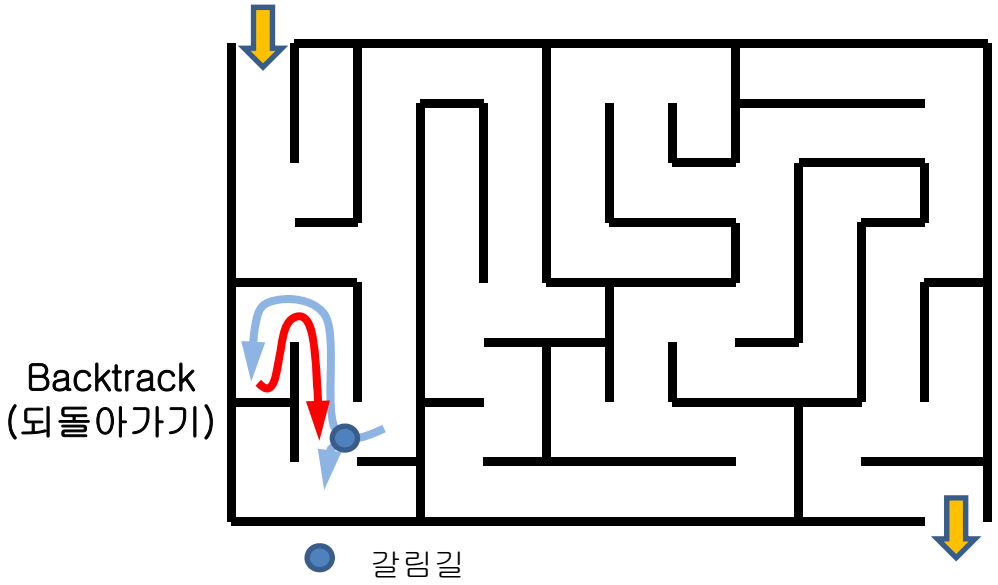
기본 알고리즘
제6장

2017-Fall

국민대학교 컴퓨터공학부 최준수

Backtracking

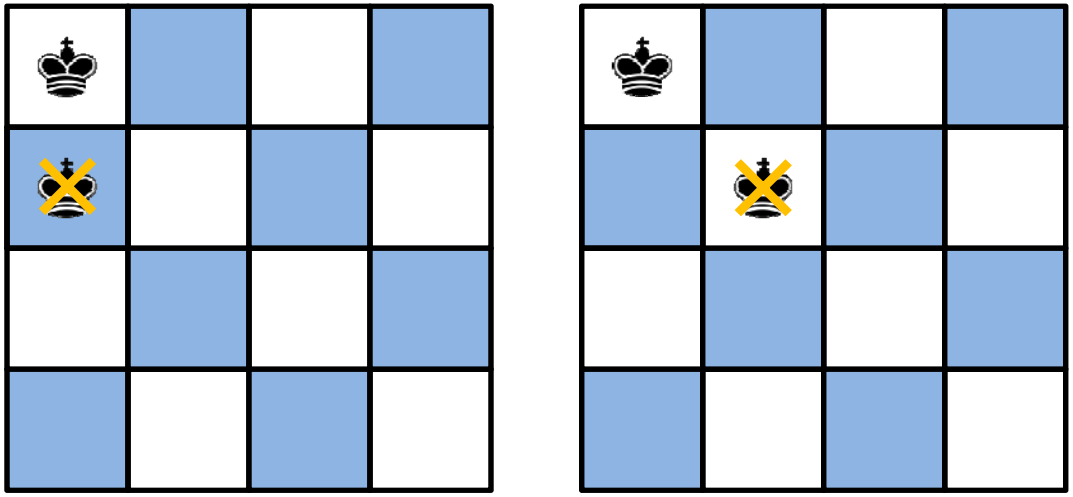
- Backtracking
  - 미로찾기
    - 막다른 골목에서는 더 이상 출구로 나아갈 수 없으므로, 이제까지 온 길을 되돌아가게 된다.



    - 마찬가지로, 지금 상태에서 앞으로 계속 진행한다고 하더라도 해답을 구할 수 없는 경우에는 앞으로 나아가는 것을 포기하고 이전 상태로 되돌아가서 다른 경우를 찾아본다.

N-Queen Problem

- N-Queen Problem
  - N×N 크기의 체스판에 N 개의 Queen 을 같은 행, 같은 열, 같은 대각선 위에 있지 않도록 놓는다.
  - Example: 4-Queen Problem



N-Queen Problem

- 간단한 4-Queen 문제 해법
  - 4개의 queen 을 모두 다른 행에 놓으면서, 해가 되는 queen 의 열의 위치가 어디인지 계산한다.
  - Candidate 해
    - 모두 다른 행에 대한, 가능한 모든 열의 조합
      - Queen의 위치 <i, j> : i-번째 열, j-번째 행

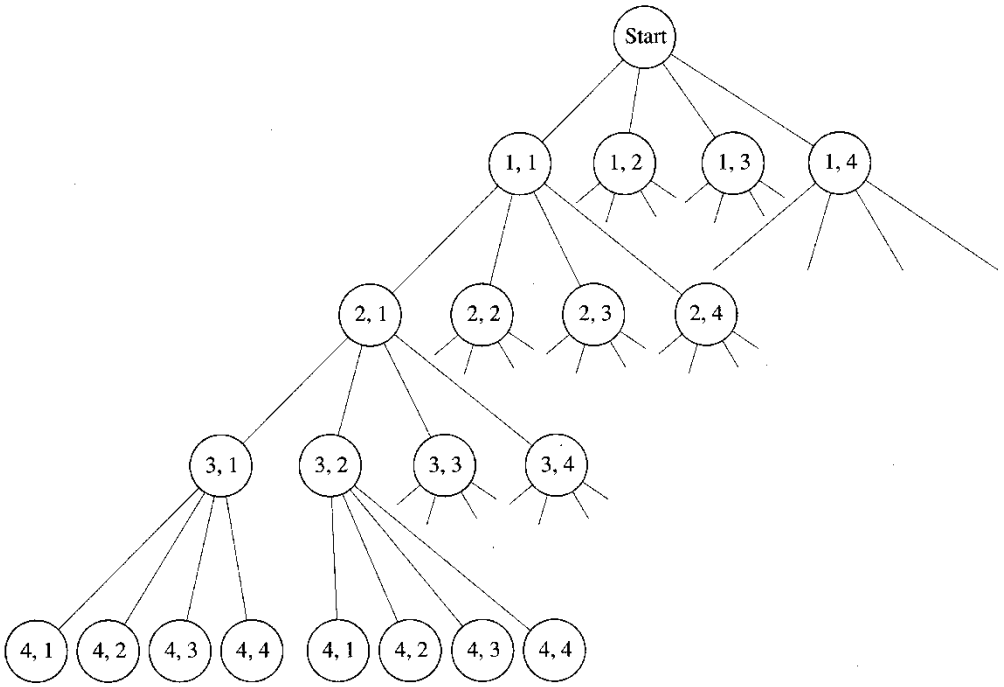
$$[<1,1>,<2,1>,<3,1>,<4,1>]$$
$$[<1,1>,<2,1>,<3,1>,<4,2>]$$
$$[<1,1>,<2,1>,<3,1>,<4,3>]$$
$$[<1,1>,<2,1>,<3,1>,<4,4>]$$
$$[<1,1>,<2,1>,<3,2>,<4,1>]$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$[<1,4>,<2,4>,<3,4>,<4,4>]$$

N-Queen Problem

- 간단한 4-Queen 문제 해법
  - 4-queen 문제에서 candidate 해의 개수
    - $4 \times 4 \times 4 \times 4 = 256$
  - N-queen 문제에서 candidate 해의 개수
    - $N \times N \times \cdots \times N = N^N$
    - 8-queen 문제 : $8^8 = 2^{24} = 16{,}777{,}216$

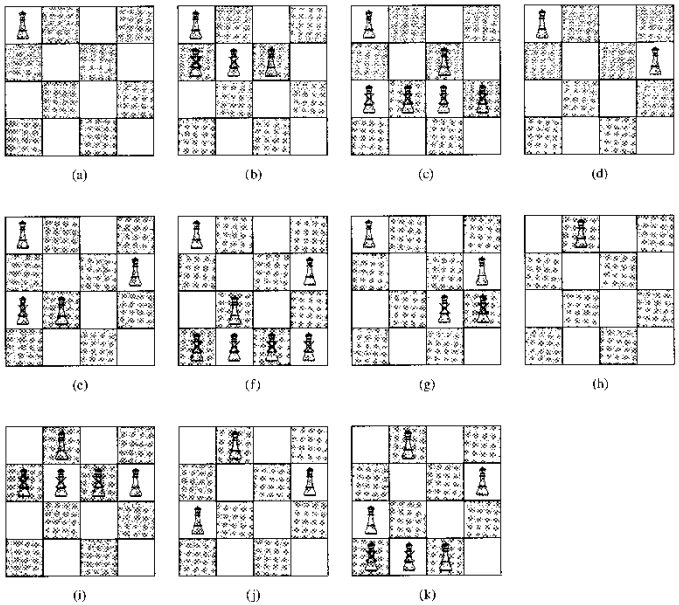
State-Space Tree

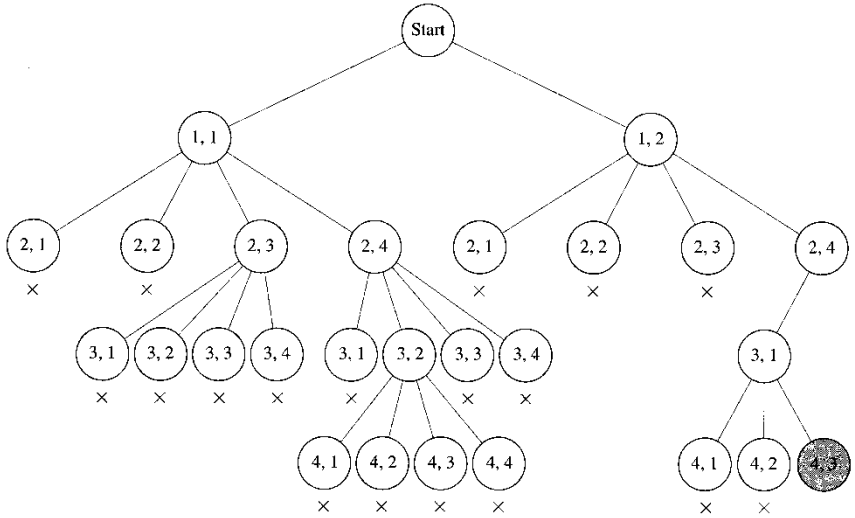
- State-Space Tree
  - 모든 candidate 해를 포함하는 tree
  - Tree의 각 노드는 한 개 queen의 위치 <i, j>를 나타냄
  - Tree의 루트노드부터 단말노드까지의 각 노드의 위치의 합이 한 개의 candidate 해를 나타냄.



State-Space Tree

- Backtracking
  - State-space tree의 어떤 노드에서는 더 이상 해를 구할 수 없는 경우(즉, 미로의 막다른 골목)라고 판단되면 그 노드의 부모노드로 되돌아가고, 그 부모노드의 다른 자식노드에서 계속 진행함.





Foundations of Algorithms Using C++ Pseudocode
R. Neapolitan, K. Naimipour, 1998

State-Space Tree

- Backtracking
  - State-space tree의 노드
    - Non-promising : 더 이상 해를 구할 수 없는 경우라고 판단되는 노드
    - Promising

  - **Pruning**
    - Backtracking 단계
      - State-space tree를 검색
      - 각 노드가 promising 인지, non-promsing 인지를 판별
        - » Promising : 계속 진행
        - » Non-promising : 부모노드로 돌아가서 다른 자식노드를 검사

Foundations of Algorithms Using C++ Pseudocode
R. Neapolitan, K. Naimipour, 1998

N-Queen Problem

```
#define N 4
int col[N];

void nQueens(int row)
{
    int i;

    if (isPromising(row))
        if (row == N-1)
            printNqueens();
        else
            for(i=0; i<N; i++)
            {
                col[row+1] = i;
                nQueens(row+1);
            }
}
```

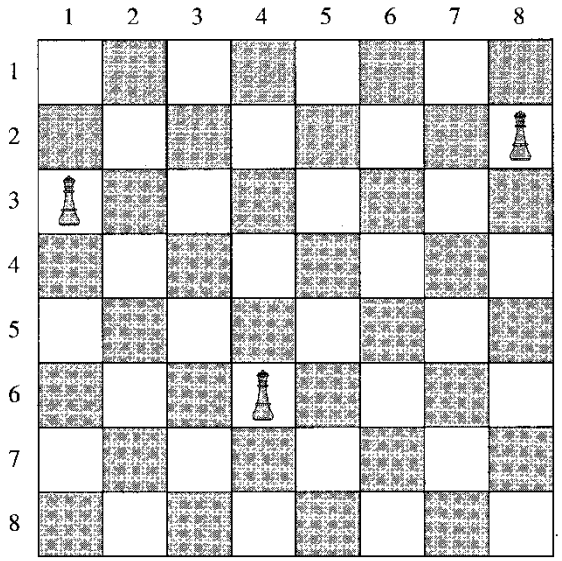
```
void main()
{
    int i;

    for(i=0; i<N; i++)
    {
        col[0] = i;
        nQueens(0);
    }
}
```

N-Queen Problem

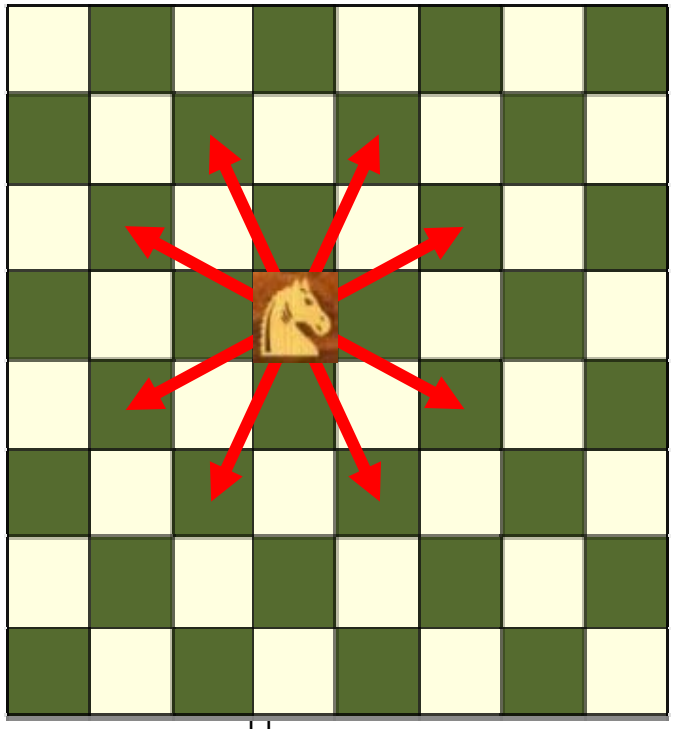
```
int isPromising(int row)
{
    int k;
    int promising;

    k = 0;
    promising = 1;
    while(k<row && promising)
    {
        if(col[row] == col[k] || abs(col[row]-col[k]) == row-k)
        promising = 0;
        k++;
    }
    return promising;
}
```



Knight's Tour Problem

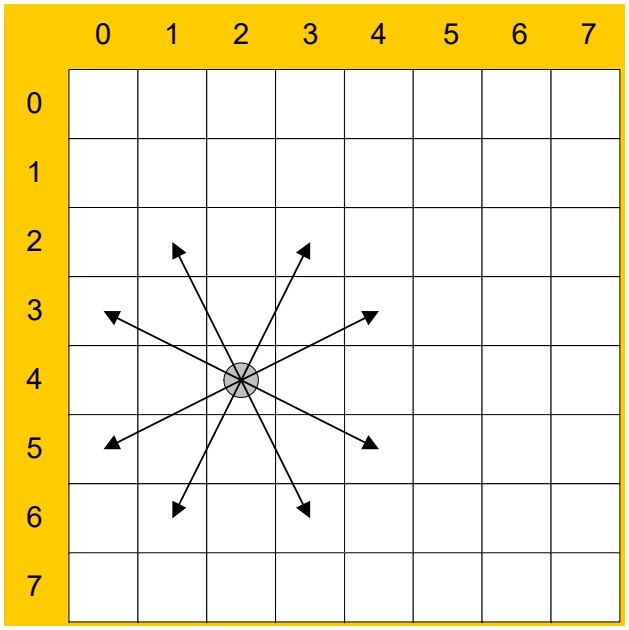
- Knight's Tour
  - 문제
    - 체스판에서 기사(Knight) 말의 움직임은 아래 그림과 같다.
    - 임의의 위치에 놓여진 기사를 움직여서 모든 64개의 격자를 모두 방문하도록 기사말을 옮기는 방법을 계산하시오. 단, 기사가 이미 방문한 격자는 다시 방문하지 않는다.



Knight's Tour Problem

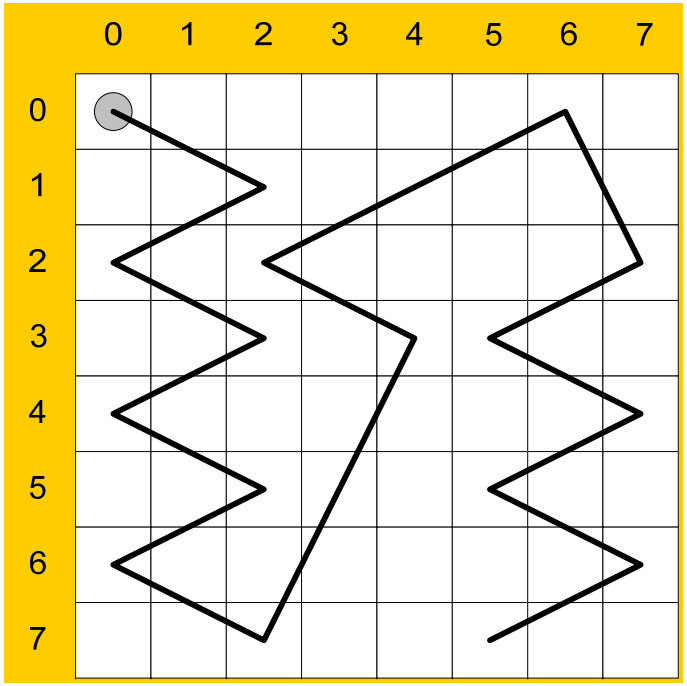
- Knight's Tour
  - <i, j> 위치에서 다음에 옮겨갈 수 있는 위치

$$<i-2, j+1> \quad <i-1, j+2> \quad <i+1, j+2> \quad <i+2, j+1>$$
$$<i+2, j-1> \quad <i+1, j-2> \quad <i-1, j-2> \quad <i-2, j-1>$$



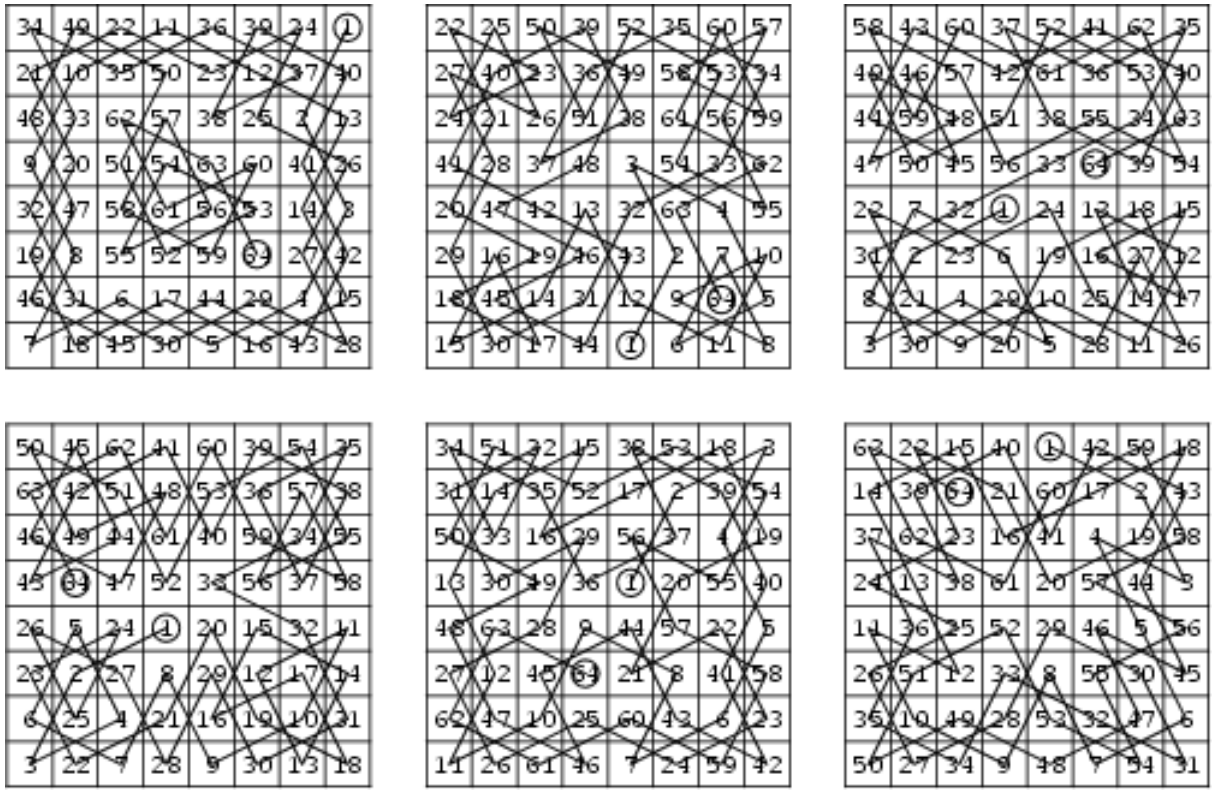
Knight's Tour Problem

- Knight's Tour
  - <0, 0> 위치에서 출발한 knight movement



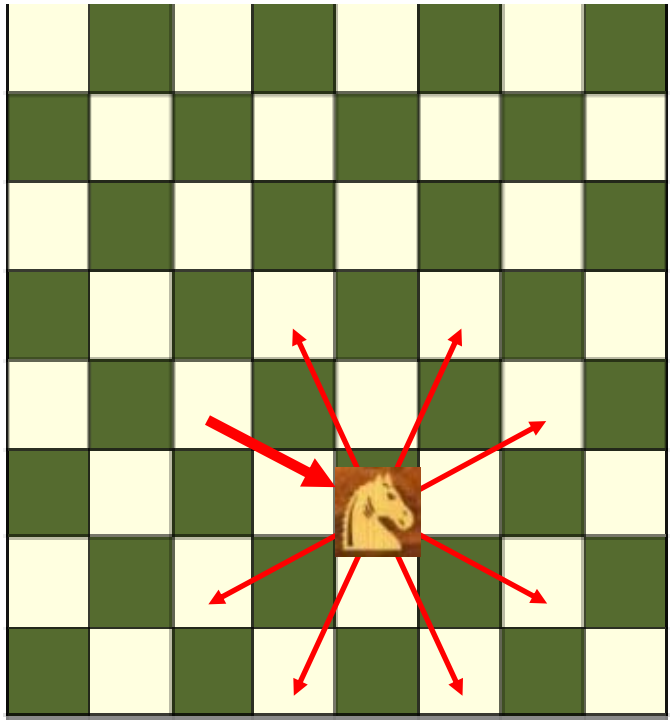
Knight's Tour Problem

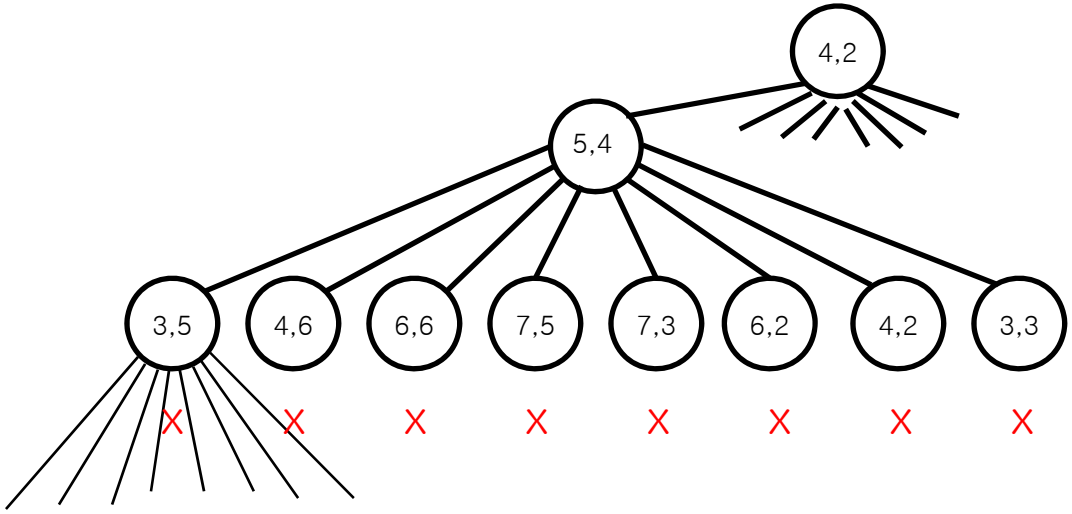
- Knight's Tour
  - 예



Knight's Tour Problem

- Knight's Tour
  - Test promising(v) at state-space tree





Knight's Tour Problem

- Knight's Tour Solution (recursive)

```
#define MAXSIZE 9

#define MARK 1
#define UNMARK 0

typedef struct Point {int x, y;} point;
point direction[8] = {{1, -2}, {2, -1}, {2, 1}, {1, 2},
                      {-1, 2}, {-2, 1}, {-2, -1}, {-1, -2}};
int board[MAXSIZE][MAXSIZE], path[MAXSIZE][MAXSIZE];

int knightTour (int m, int n, point pos, int counter)
{
    int i;
    point next;

    if (counter == m * n)
        return 1;

    for (i=0; i<8; i++)
    {
        {
            next.x = pos.x + direction[i].x;
            next.y = pos.y + direction[i].y;

            if ( next.x > 0 && next.x <= n &&
                 next.y > 0 && next.y <= m &&
                 board[next.y][next.x] != MARK )
            {
                board[next.y][next.x] = MARK;
                path[next.y][next.x] = counter+1;

                if ( knightTour(m, n, next, counter+1) )
                    return 1;

                board[next.y][next.x] = UNMARK;
            }
        }
    }

    return 0;
}
```

Sudoku

- Sudoku
  - Solve Sudoku problem with Backtracking Algorithm

	1		6	3	5	4		7
2				8			9	
			2			1		
		1		7		8	5	
6	5		8		1		3	2
	7	8		2		9		
		2			4			
	6			5				9
5		7	9	1	2		6	

					3		8	5
		1		2				
			5		7			
		4				1		
	9							
5							7	3
		2		1				
				4				9

Term Project

- Term Project
  - 진행방안
    - 중간고사 이전부터 시작
      - 제안서 :
      - 중간보고서 :
      - 최종보고서 및 발표 :
    - 일정 및 장소
      - 최종발표: 12월21일(월) 오후 6:00 – 오후 9:00
      - 장소 : 7호관 114호
    - 내용
      - Sudoku + Crosswords
    - 조 구성
      - 6명/팀 (프로그래밍 경시대회 참가 2팀을 1개 조로 구성)

Term Project

- Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

  - Implementation of Sudoku solving algorithm
    - Ver1 : Backtracking algorithm (or Brute-force algorithm)
      - Reference: wiki "Sudoku solving algorithms",
    - Ver2 : Algorithm with "Dancing Links" data structure
      - Read a paper, "Dancking Links", D. Knuth, Stanford Univ.
      - Can use an open source code
    - Computation time?