**학습활동보고서 # 5**

|  |  |  |  |
| --- | --- | --- | --- |
| 대학/학부/학과 | 엘텍공과대학 | 전공 | 컴퓨터공학과 |
| 학번 | 1871001 | 이름 | ZHU ZHAOLING |

|  |  |
| --- | --- |
| 학습활동 주제 및 목표 | 되추적(백트래킹) |

(필요한 항목 및 내용을 자유로운 양식으로 작성하세요)

1. 학습활동 –내용을 상세히 작성합니다.

**5.1 백트래킹과 n-Queens 문제**

**▪ 되추적 (backtracking)**

• 임의의 집합(set)에서 주어진 기준(criterion)대로

⁃ 원소의 순서(sequence)를 선택하는 문제를 푸는 데 적합

• 트리 자료구조의 변형된 깊이우선탐색(DFS: depth-first-search)

• 모든 문제 사례에 대해서 효율적이지 않지만,

⁃ 많은 문제 사례에 대해서 효율적이다.

⁃ 예) 부분집합의 합, 0-1 배낭문제, etc

**▪ 일반적인 백트래킹 알고리즘**

▪ 문제: 백트래킹 (backtracking)

• 백트래킹으로 문제 해결:

⁃ 임의의 집합에서 기준에 따라 원소의 순서를 선택

• 문제에 적용:

⁃ 임의의 집합(set): 체스보드에 있는가능한 위치

⁃ 기준(criterion): 새로 놓을 퀸이 다른 퀸을 위협할 수 없음

⁃ 원소의 순서(sequence): 퀸을 놓을 수 있는의 위치

**5.2 n-Queens 문제의 구현**

**▪ 문제의 해결**

• 기본 가정: 같은 행(row)에는 퀸을 놓지 않는다.

• 유망 함수의 구현

⁃ 같은 열(column)이나 같은 대각선(diagonal)에 놓이는 지를 확인

**▪ 유망의 조건 1: 같은 열 체크**

• col[i]: i번째 행(row)에서 퀸이 놓여있는 열(column)의 위치

• col[k]: k번째 행(row)에서 퀸이 놓여있는 열(column)의 위치 • col[i]==col[k]: 같은 열에 놓이게 되므로, 유망하지 않다.

**▪ 유망의 조건 2: 대각선 체크**

• 왼쪽에서 위협하는 퀸에 대해서

⁃ 열에서의 차이는 행에서의 차이와 같다.

⁃ col[i] - col[k] == i - k

• 오른쪽에서 위협하는 퀸에 대해서

⁃ 열에서의 차이는 행에서의 차이의 마이너스값과 같다.

⁃ col[i] - col[k] == k - i

• col[i]와 col[k]의 절대값으로 대각선 위협 판단

**Algorithm 5.1: Backtracking Algorithm for the n-Queens Problem**

1. **def** n\_queens (i, col):
2. n = len(col) - 1
3. **if** (promising(col, i)):
4. **if** (i == n):
5. **print**(col[1: n + 1])
6. **else**:
7. **for** j **in** range(1, n + 1):
8. col[i + 1] = j
9. n\_queens(col, i + 1)
11. **def** promising (i, col):
12. k = 1
13. flag = True
14. **while** (k < i **and** flag):
15. **if** (col[i] == col[k] **or** abs(col[i] - col[k]) == (i - k)):
16. flag = False
17. k += 1
18. **return** flag
20. n = 4
21. col = [0] \* (n + 1)
22. n\_queens(0, col)

**5.3 부분집합의 합 구하기**

**▪ 부분집합의 합 구하기: 백트래킹 (backtracking)**

• 상태공간트리를 만들어 푸는 방법

**Algorithm 5.3: Backtracking for the Sum-of-Subsets Problem**

1. **def** sum\_of\_subsets (i, weight, total):
2. n = len(w) - 1
3. **if** (promising(i, weight, total)):
4. **if** (weight == W):
5. **print**(include[1: n + 1])
6. **else**:
7. include[i + 1] = True
8. sum\_of\_subsets(i + 1, weight + w[i+1], total - w[i+1])
9. include[i + 1] = False
10. sum\_of\_subsets(i + 1, weight, total - w[i+1])
12. **def** promising (i, weight, total):
13. **if** ((weight + total >= W) **and**
14. (weight == W **or** weight + w[i+1] <= W)):
15. **return** True
16. **else**:
17. **return** False
19. n = 5
20. W = 21
21. w = [0, 5, 6, 10, 11, 16]
22. total = sum(w)
23. include = [False] \* (n + 1)
24. **print**(total)
25. sum\_of\_subsets(0, 0, total)
26. 느낀 점 - 이번 학습활동으로 배운 점 혹은 시행착오를 분석한 후, 다음 학습활동 반영합니다.

직설적으로 말하자면, 역추적 알고리즘은 철저한 방법이다. 그러나 역추적 알고리즘은 가지치기(pruning) 기능을 사용하여 최종 상태(즉, 응답 상태)에 도달할 수 없는 일부 노드를 차단하여 상태 공간 트리 노드의 생성을 줄이다.