제 5 장 되추적(퇴각 검색) (Backtracking)

- 일반적인 방법

일반적인 방법

- 퇴각검색 방법을 적용하기 위해서 문제에 대한 해는 n-tuple 형식, $(x_1, x_2, \ldots, x_n)$ 으로 표현될 수 있어야 한다. 이때, $x_i \in S_i$.
- 이것은 기준함수(criterion function) $P(x_1, x_2, \ldots, x_n)$을 만족하는 벡터 $(x_1, x_2, \ldots, x_n)$들 중 이익을 최대로 (또는 비용을 최소로) 하는 하나의 벡터를 찾는 것이다 (때로는 만족하는 벡터만을 찾기도 한다).
- 예: 정렬 문제

  n = 5, A[1:n] = [15, 14, 9, 20, 10]

  $(x_1, x_2, x_3, x_4, x_5) = (3, 5, 2, 1, 4)$

  여기서 x_i 는 i-번째 작은 원소의 인덱스이다.

  기준함수는 $A[x_i] \le A[x_{i+1}]$, $(1 \le i < n)$ 이다.

  위 정렬 문제에서 모든 가능한 투플의 수는 5! = 120 이다.

  n 개의 원소에 대해서 n! 개 존재한다.

  → 실제 정렬 문제는 퇴각 검색방법으로 풀지 않는다.

- $x_i \in S_i$ 가 선택될 수 있는 값의 수를 m_i라 할 때, (즉 $m_i = |S_i|$)
  모든 가능한 n-투플들의 수는 $m = m_1 \cdot m_2 \cdots m_n$ 이다.
- 최악의 경우 m 개만큼의 투플들을 생성하여 최대 또는 최소의 값을 갖는 해를 구해야 한다. 그러나 실제로 m 보다 훨씬 적은 시도를 하게 된다.
- 기준 함수를 만족할 수 없는 투플들은 제거되며, 먼저 만들어진 투플의 기준 함수값 보다 더 작은 (혹은 더 큰) 기준 함수값을 갖는 투플들은 생성할 필요가 없다. → 한정 함수(bounding function)의 사용

두 가지 제약 조건(constraint)

- 명시적 제약 조건(explicit constraint):

  x_i 를 주어진 집합 S_i 에서만 취하도록 제약하는 규칙

  예: 0/1 배낭 문제에서 $x_i = 0$ 또는 1 이다.

  n 개 원소의 정렬 문제에서 $1 \le x_i \le n$ 이다.
- 암시적 제약 조건(implicit constraint):

  투플들이 기준 함수를 만족하고 있는지의 여부를 제약하는 규칙

  예: 0/1 배낭 문제에서 $\Sigma_{l \le i \le n} w_i \cdot x_i \le m$

  n 개 원소의 정렬 문제에서 $A[x_i] \le A[x_{i+1}]$, $(1 \le i < n)$.
- 이 제약조건들이 기준 함수로 사용된다.

예제 1: 8-퀸(queens) 문제

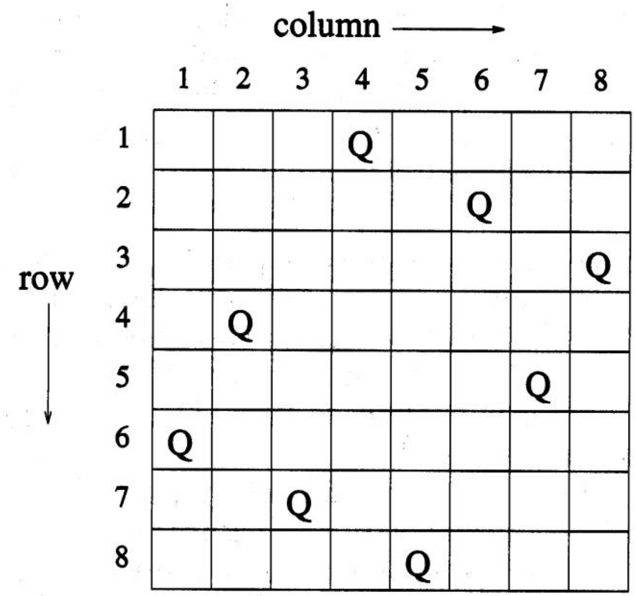
- 8 x 8 체스 판에 8 개의 퀸들이 서로 공격하지 못하도록 배치하는 문제이다. 즉, 어떤 두 퀸도 같은 행이나 같은 열, 같은 대각선 상에 있지 않도록 하는 것이다.
- 8-투플 형의 해: $(x_1, x_2, \ldots, x_8)$

  x_i 는 i 번째 퀸이 위치하는 i 번째의 열 번호이다.

  예: 오른쪽 그림에서 (4, 6, 8, 2, 7, 1, 3, 5)
- 명시적 제약조건:

  $x_i \in S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- 암시적 제약조건:

  어떤 두 퀸도 같은 행이나 같은 열, 같은 대각선 상에 있지 않아야 한다.



예제 2 부분집합의 합 (sum of subsets)

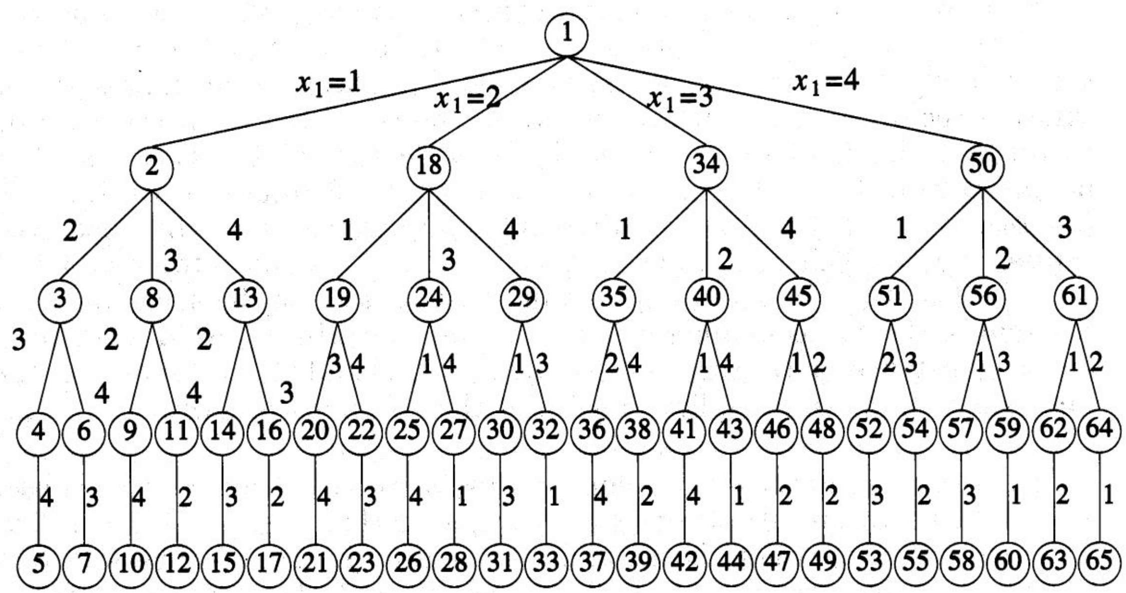
- 주어진 양수 w_i, $1 \le i \le n$ 들과 m 에 대하여 합이 m 이 되는 모든 부분 집합들을 찾는 문제이다.
- 예: n = 4, (w_1, w_2, w_3, w_4) = (11, 13, 24, 7), m = 31
  원하는 부분 집합들은 (11, 13, 7) 과 (24, 7) 이다.
  w_i 의 인덱스로 표시하는 두 가지 방법이 있다.
  첫째, (1, 2, 4) 와 (3, 4) 이다. → 가변 길이
  둘째, (1, 1, 0, 1) 과 (0, 0, 1, 1) 이다. → 고정 길이
- 가변 길이 표시: k-투플 $(x_1, x_2, \ldots, x_k)$, $1 \le k \le n$
  명시적 제약조건: $x_i \in \{j \mid j$ 는 $1 \le j \le n$ 인 하나의 정수 $\}$
  암시적 제약조건: 어느 두 인덱스도 같지 않고, 대응하는 w_i 들의 합이 m 이다.
- 고정 길이 표시: n-투플 $(x_1, x_2, \ldots, x_n)$
  명시적 제약조건: $x_i \in \{0, 1\}$, w_i 가 선택되면 1, 그렇지 않으면 0.
  암시적 제약조건: $\sum_{1 \le i \le n} w_i \cdot x_i = m$

트리 구성(tree organization)

- 퇴각 검색 알고리즘들은 주어진 문제에 대해 해 공간을 체계적으로 탐색함으로써 문제의 해를 구한다.
- 이러한 탐색은 해 공간을 트리 구성으로 나타냄으로써 쉽게 나타낼 수 있다.
- 트리 구성은 여러 형태로 나타낼 수 있다.

예제 3 n-퀸

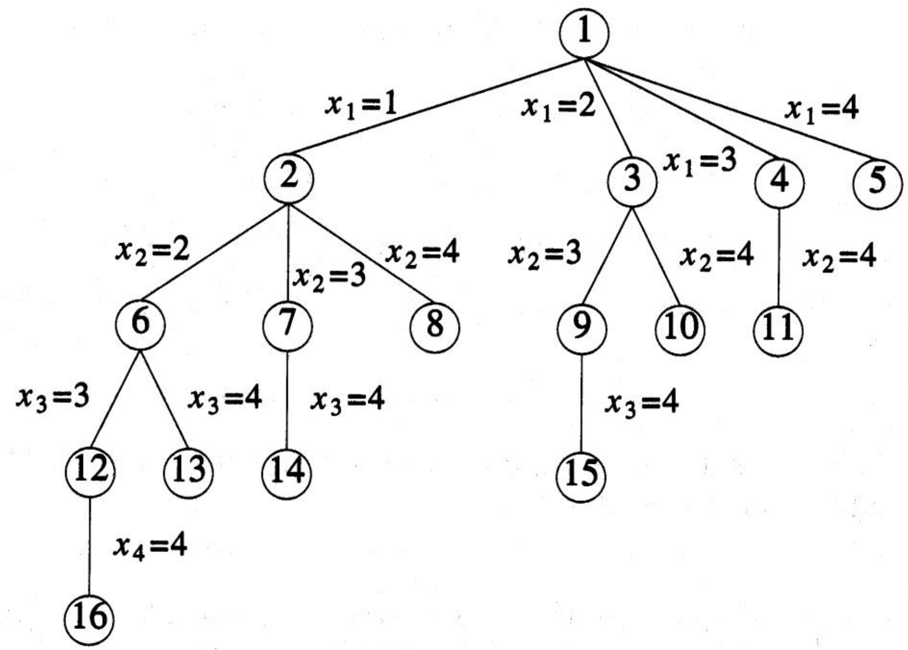
- 예제 1의 8-퀸 문제의 일반화된 문제이다.
- n 개의 퀸들은 서로 공격할 수 없도록 n x n 체스판에 놓여져야 한다.
  즉 어떤 두 퀸도 같은 행, 같은 열, 같은 대각선 상에 놓여서는 안된다.
- 해 공간은 n-투플 (1, 2, …, n)의 n! 개의 모든 순열들로 구성된다.
- 예: n = 4인 4-퀸 문제의 트리 구성(깊이 우선 탐색 순서, 순열트리)



해는 단말 노드에 존재

예제 4 부분집합의 합(n=4)

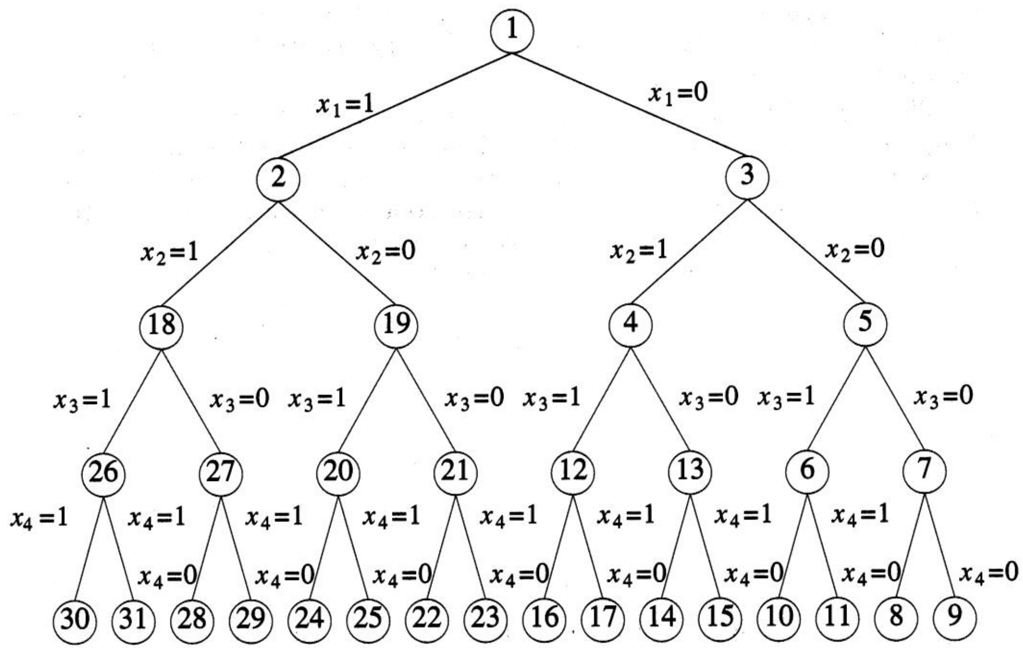
- 가변 길이의 해 공간



- 해는 어느 노드에서나 가능하며, 어떤 노드가 제약조건을 만족할 때(즉, 기준함수를 만족할 때), 루트로부터 그 노드까지의 경로가 해를 나타낸다.

  예: 13번 노드가 조건을 만족하면, 해는 (1, 2, 4)가 된다.

- 고정 길이의 해 공간



- 해는 단말 노드에서만 가능하며, 어떤 단말 노드가 제약조건을 만족할 때, 루트로부터 그 노드까지의 경로가 해를 나타낸다.
  예: 28번 노드가 조건을 만족하면, 해는 (1, 1, 0, 1)이 된다.

해 공간의 트리 구성과 관련된 용어

- 문제 상태(problem state): 트리에 포함된 각 노드
- 해 상태(solution state) : 문제 상태들 중, 루트로부터 그 노드까지의 경로가 해가 될 수 있는 투플을 정의하는 문제 상태

  예: 부분집합의 합 문제에서 가변 길이의 경우에는 <u>모든 노드가</u> 해 상태, 고정 길이의 경우에는 <u>단말 노드들만이</u> 해 상태가 된다.
- 해답 상태(answer state) : 해 상태들 중, 루트로부터 그 노드까지의 경로가 문제에 대한 해가 되는 경우(즉 제약조건들을 만족하는 해 상태)
- 이러한 해 공간에 대한 트리 구성을 상태 공간 트리(state space tree)라 부른다.

정적 트리와 동적 트리

- 정적 트리(static tree): 문제의 사례에 관계없이 항상 같은 형태로 만들어지는 상태 공간 트리

  예: 부분집합의 합문제에서 n = 4, (11, 13, 24, 7), m = 31

  n = 4, (25, 10, 15, 5), m = 35

  위 두 사례에 대해 항상 같은 상태 공간 트리를 만든다.
- 동적 트리(dynamic tree): 문제의 사례에 따라 서로 다른 형태로 만들어지는 상태 공간 트리

  → 문제의 사례에 따라 왼쪽 자식 노드를 만들 때, $x_i = 1$ 이 되거나 $x_i = 0$ 이 된다. 이것은 부모 노드의 상태에 따라 결정된다.
- 지금까지의 예들은 모두 정적 트리들이다.

살아있는 노드와 죽은 노드

- 상태 공간 트리는 처음부터 모든 노드들이 만들어져 있는 것이 아니라 루트 노드에서부터 체계적으로 다른 노드(즉, 문제 상태)들을 만들어 간다.
- 노드에는 두 가지 종류의 노드가 있다.
- <u>살아있는 노드(live node)</u>: 이미 생성되어졌으나 아직 자식 노드들 모두가 생성되지는 않은 노드

  그 중, 현재 자식을 생성하고 있는 노드를 <u>E-노드</u> 라 한다.
- <u>죽은 노드(dead node)</u>: 이미 생성된 노드로서 확장될 수 없거나 자식들이 모두 생성된 노드
- 노드들을 생성할 때 살아있는 노드들의 리스트를 간직한다.
- 이처럼 체계적으로 노드(문제 상태)를 생성하며, 노드들 중 어느 것이 해 상태인가를 결정하고, 해 상태인 노드들 중 해답 상태인 노드를 결정해 간다.

깊이 우선 생성

- 깊이 우선 생성:

  현재 E-노드인 R의 새로운 자식 C를 생성하자마자, 이 자식 노드가 새로운 E-노드가 된다. 그리고 R은 live node로 살아 있게 되며, C의 부분트리가 모두 생성되고 난 후에 다시 E-노드가 되어 다른 자식을 생성하게 된다.

  예: 슬라이드 8번의 4-퀸 문제의 상태 공간 트리의 노드 생성 순서

- 한정 함수(bounding function)를 사용하여 live node들 중 해답 상태로 갈 수 없는 노드들을 제거하는 방법을 사용할 때, 이것을 퇴각 검색(backtracking) 이라 한다.

너비 우선 생성

- 너비 우선 생성:

  현재 E-노드는 모든 자식 노드들을 생성하고 죽은 노드가 된다. 이때 생성된 자식 노드들은 모두 live node 가 된다. 다음의 E-노드는 live node들의 리스트에서 선택된다. 주로 live node들의 리스트로 큐(queue)를 사용한다.

  예: 슬라이드 9번의 가변 길이 상태 공간 트리의 노드 생성 순서
- 한정 함수(bounding function)를 사용하여 live node들 중 해답 상태로 갈 수 없는 노드들을 제거하는 방법을 사용할 때, 이것을 분기와 한정(branch and bound) 이라 한다.
- live node들의 리스트를 큐 대신 스택(stack)을 사용할 수 있다.

  → D-탐색(depth search) 라 한다.

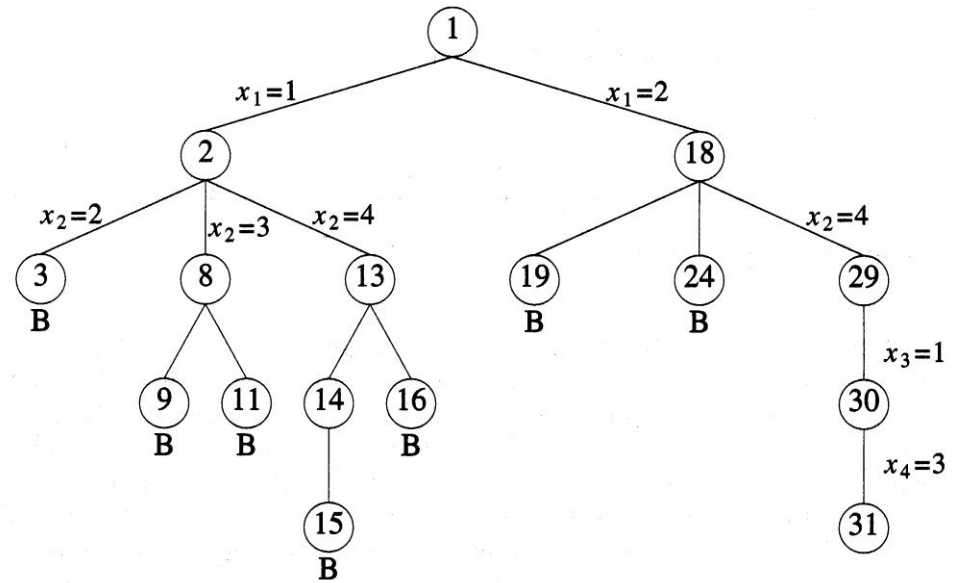
  예: 슬라이드 10번의 고정 길이 상태 공간 트리의 노드 생성 순서

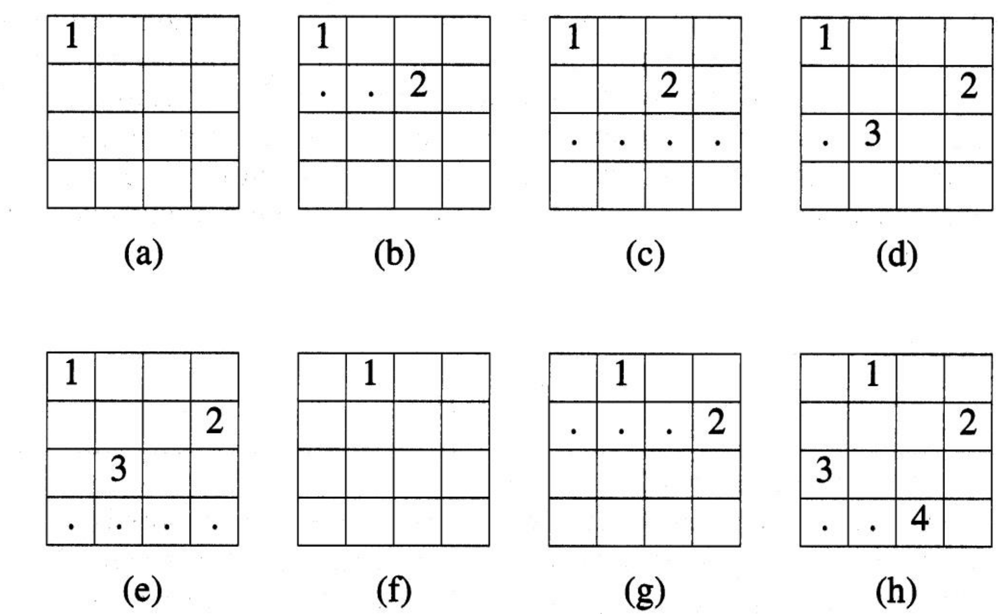
예제 5. 4-퀸 문제

- 한정 함수:

  $(x_1, x_2, \ldots, x_i)$가 현재의 E-노드까지의 경로일 때, 이 노드의 자식 노드는 $(x_1, x_2, \ldots, x_{i+1})$의 경로로 나타낼 수 있으며 x_{i+1} 의 위치에 있는 퀸이 다른 퀸을 공격할 수 없어야만 한다.
- 루트 노드: () → E-노드
- 노드 2번: (1) → E-노드
- 노드 3번: (1, 2) → 한정 함수에 의해 삭제된다.
- 노드 8번: (1, 3) → E-노드
- 노드 9번: (1, 3, 2) → 삭제
- 노드 11번: (1, 3, 4) → 삭제
- 노드 8은 이제 dead node가 되며, 노드 2로 퇴각하고 노드 2가 다시 E-노드로 된다.
- 노드 13번: (1, 4) → E-노드

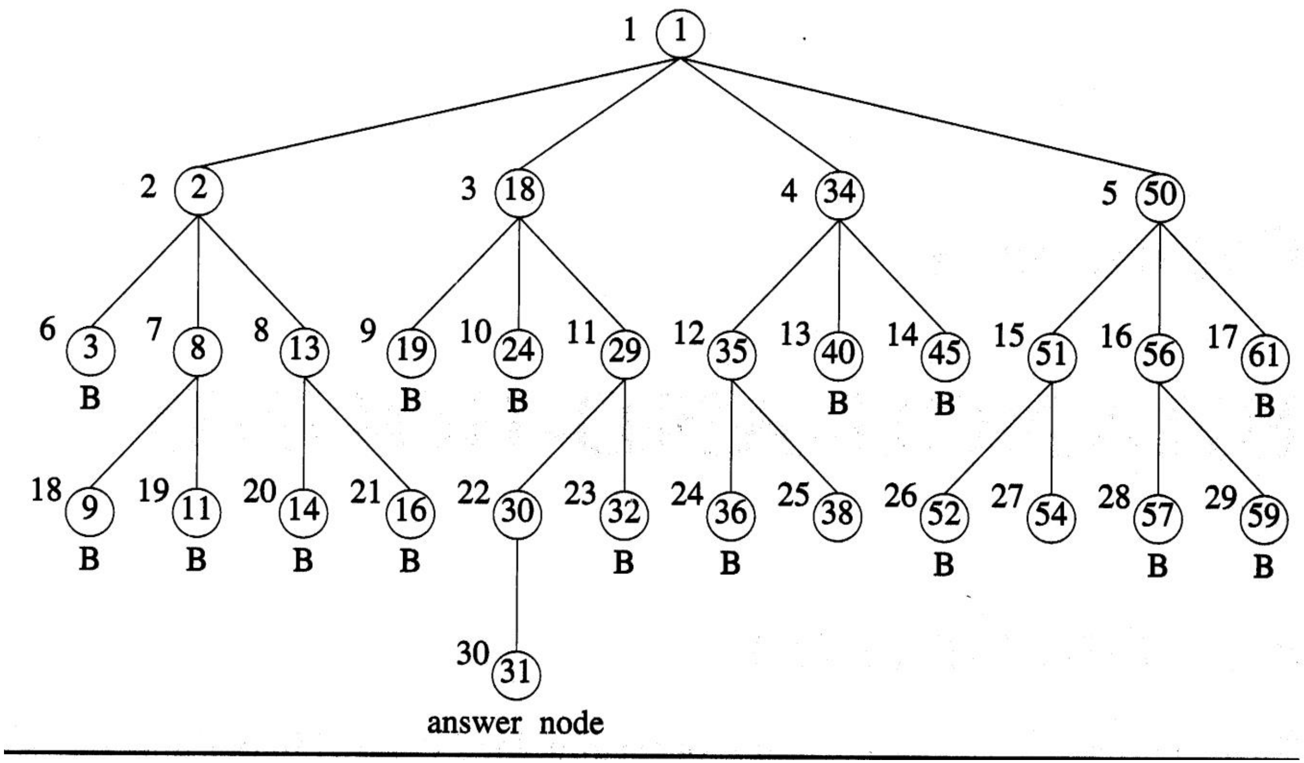
  : (다음 슬라이드의 그림 참조)


1
$x_1=1$
$x_1=2$
2
18
$x_2=2$
$x_2=3$
$x_2=4$
$x_2=4$
3
B
8
13
19
B
24
B
29
9
B
11
B
14
16
B
$x_3=1$
30
$x_4=3$
15
B
31


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

FIFO 분기와 한정(6장)

- 같은 한정 함수를 사용하며 생성되는 노드의 순서가 다르다.



퇴각 검색의 일반적인 방법

- $(x_1, x_2, \ldots, x_{i-1})$: 루트 노드로부터 어떤 노드까지의 경로
- $T(x_1, x_2, \ldots, x_{i-1})$: $(x_1, x_2, \ldots x_{i-1}, x_i)$이 역시 하나의 문제 상태까지의 경로가 될 수 있는 모든 가능한 x_i 의 집합

  예: 슬라이드 17번에서 T(1) = {2, 3, 4}, 즉 (1, 2) (1, 3) (1, 4) 가 모두 가능한 다음 경로들이다.

  그리고 T(1, 3) = {2, 4}, 즉 (1, 3, 2) 와 (1, 3, 4)가 가능한 다음 경로들이다.
- $B(x_1, x_2, \ldots x_{i-1}, x_i)$: 한정 함수로서 거짓이면 그 경로가 해답 노드까지 확장될 수 없음을 뜻한다.
- 따라서 $(x_1, x_2, \ldots, x_{i-1})$의 i 번째 위치에 대한 후보는 (즉, x_i) T에 의해 생성되고 한정 함수 B를 만족하는 값들이다.
- 실제 알고리즘에서는 트리의 노드는 생성되지 않으며 벡터 값 $(x_1, x_2, \ldots, x_i)$ 만을 저장한다. 이 벡터 값은 하나의 노드를 의미한다.

순환적인 퇴각 검색 알고리즘

- 해 벡터 (x_1, x_2, ..., x_n)은 전역 일차원 배열 x[1:n]에 저장된다.

```
void Backtrack(int k)
{
    for (each x[k] such that x[k] ∈ T(x[1], ⋯, x[k−1]) {
            if (B(x[1], ⋯, x[k]) {
                if (x[1], ⋯, x[k] is a path to an answer node)
                        output x[1:k] ;  // 만약 하나의 해만 구한다면 여기서
                                         // 플래그를 반환하여 종료시킨다.
                if (k < n) Backtrack(k+1) ;
            }
     }
}
```

- 처음에 Backtrack(1) 을 호출한다.

일반적인 퇴각검색 알고리즘(반복적 구조)

```
void Ibacktrack(int n)
{
    int k=1;
    while (k) {
          if (there remains an untried x[k] such that
               x[k] is in t(x[1], …, x[k−1]) and
               Bi(x[1], …, x[k]) is true ) {
                   if (x[1], …, x[k] is a path to an answer node)
                       output x[1:k] ;  // 만약 하나의 해만 구한다면 여기서
                                        //return; 문을 추가한다.
                   k++;  // 자식 노드로 내려간다.
               }
               else  k-- ;   // 부모 노드로 퇴각한다.
     }  // end of while
}
```

퇴각검색 알고리즘의 효율성

- 다음의 네 가지 요인들에 의해 달라진다

(1) 다음 x_i 를 생성하는데 필요한 시간
(2) 명시적 제약조건들을 만족하는 x_i 들의 수
(3) 한정함수 B_k를 계산하는데 필요한 시간
(4) B_k를 만족하는 x_i 들의 수

- 좋은 한정함수는 생성되는 노드의 수를 가능한 한 많이 줄이는 함수이다. 그러나 한정함수를 계산하는데 너무 많은 시간이 걸려서는 안 된다.

N-queen 문제의 예(lec10-1)

```
void queens(int k)
{
    int j ;
    int i ;
    for ( i = 1; i <= n ; i++) {
        col[k] = i ;
        if (promising(k)) {    // 제약조건 확인
            if ( k == n ) {
                for ( j = 1; j <= n ; j++ )
                    System.out.print(col[j} + "  ") ;
                System.out.println() ;
            }
            else
                queens(k+1)  ;    // 다음 행을 순환 호출
        }
    }
}
```