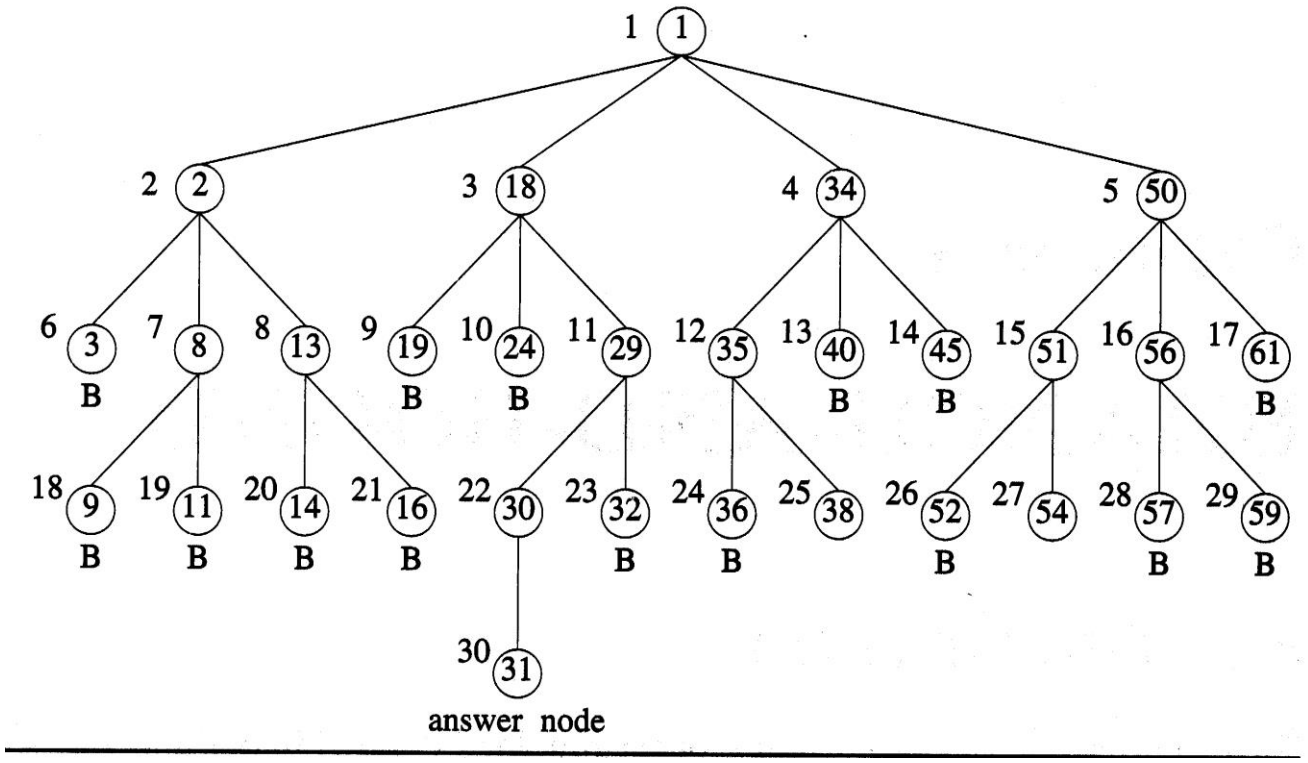
제 6 장 분기한정법 (Branch and Bound)

- 일반적인 방법

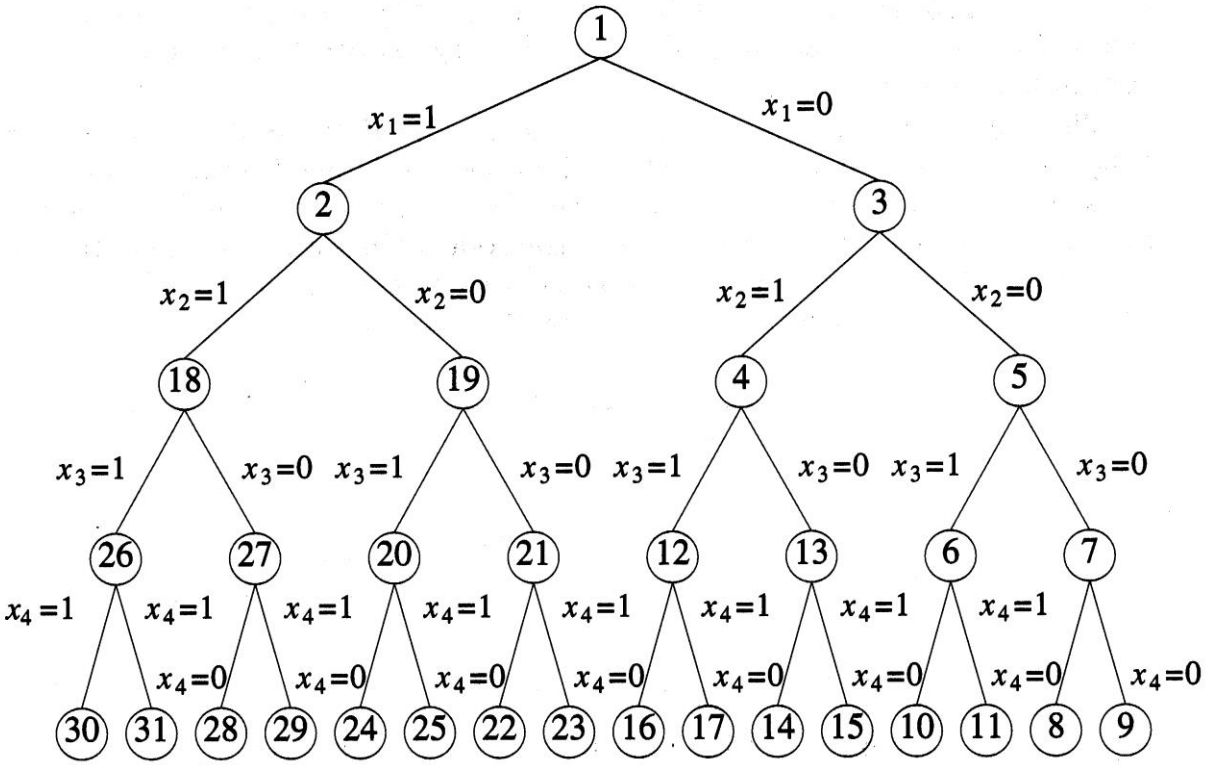
일반적인 방법

- BFS(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.
- 살아있는 노드들의 리스트에서 E-노드를 선택하는 방법에 따라 FIFO 탐색, D-탐색, LC-탐색이 있다.
- FIFO 탐색: live node들의 리스트로 큐를 사용하여 가장 먼저 삽입된 live node가 E-node로 된다.
- D-탐색: live node들의 리스트로 스택을 사용하여 가장 나중에 삽입된 live node가 E-node로 된다.
- LC(least cost)-탐색: live node들에 순위(rank)를 매겨 순위가 가장 높은 live node가 E-node로 된다.

FIFO 분기와 한정


1 1
2 2
3 18
4 34
5 50
6 3 B
7 8
8 13
9 19 B
10 24 B
11 29
12 35
13 40 B
14 45 B
15 51
16 56
17 61 B
18 9 B
19 11 B
20 14 B
21 16 B
22 30
23 32 B
24 36 B
25 38
26 52 B
27 54
28 57 B
29 59 B
30 31
answer node

부분집합의 합 문제에 대한 D-탐색



최소비용 탐색(LC-탐색)

- FIFO나 LIFO에서 다음 E-노드의 선택은 오로지 들어온 순서에 의하여 결정되므로 어떤 면에서 맹목적이다.
- 해답 노드에 가까이 있는 정도에 따라 live node에 우선 순위를 부여하여, 우선 순위가 가장 높은 live node를 E-node로 선택하여 탐색할 때 좀 더 빨리 해를 구할 수 있을 것이다. → "지능적인 탐색"
- 해답 노드에 가까이 있는 정도를 나타내기 위해 rank 함수 c(x)를 사용한다. 그러나 c(x)를 알기 어려우므로 추정 함수 $\hat{c}(x)$를 사용한다.

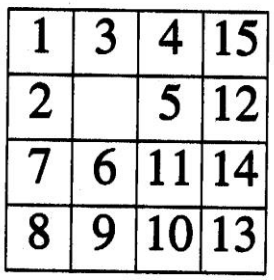
  $\hat{c}(x) = f(x) + \hat{g}(x)$

  f(x) = 루트로부터 x까지 도달하는데 걸리는 비용(증가 함수)

  $\hat{g}(x)$ = x로부터 어떤 해답 노드에 도달하는데 필요한 추가 노력에 대한 추정치
- $\hat{c}(x)$의 선택은 문제에 따라 결정되며, $\hat{c}(x)$의 값이 낮을수록 우선 순위가 높으며, 가장 작은 $\hat{c}(x)$를 갖는 노드가 다음 E-node로 선택된다.

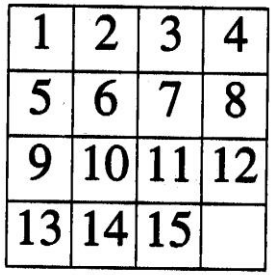
15-퍼즐 문제

- 16개의 타일들을 수용할 수 있는 정사각형의 틀위에 놓여진 15개의 타일들로 구성된다.



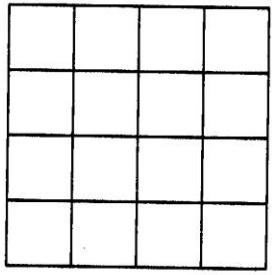
1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

(a) An arrangement



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

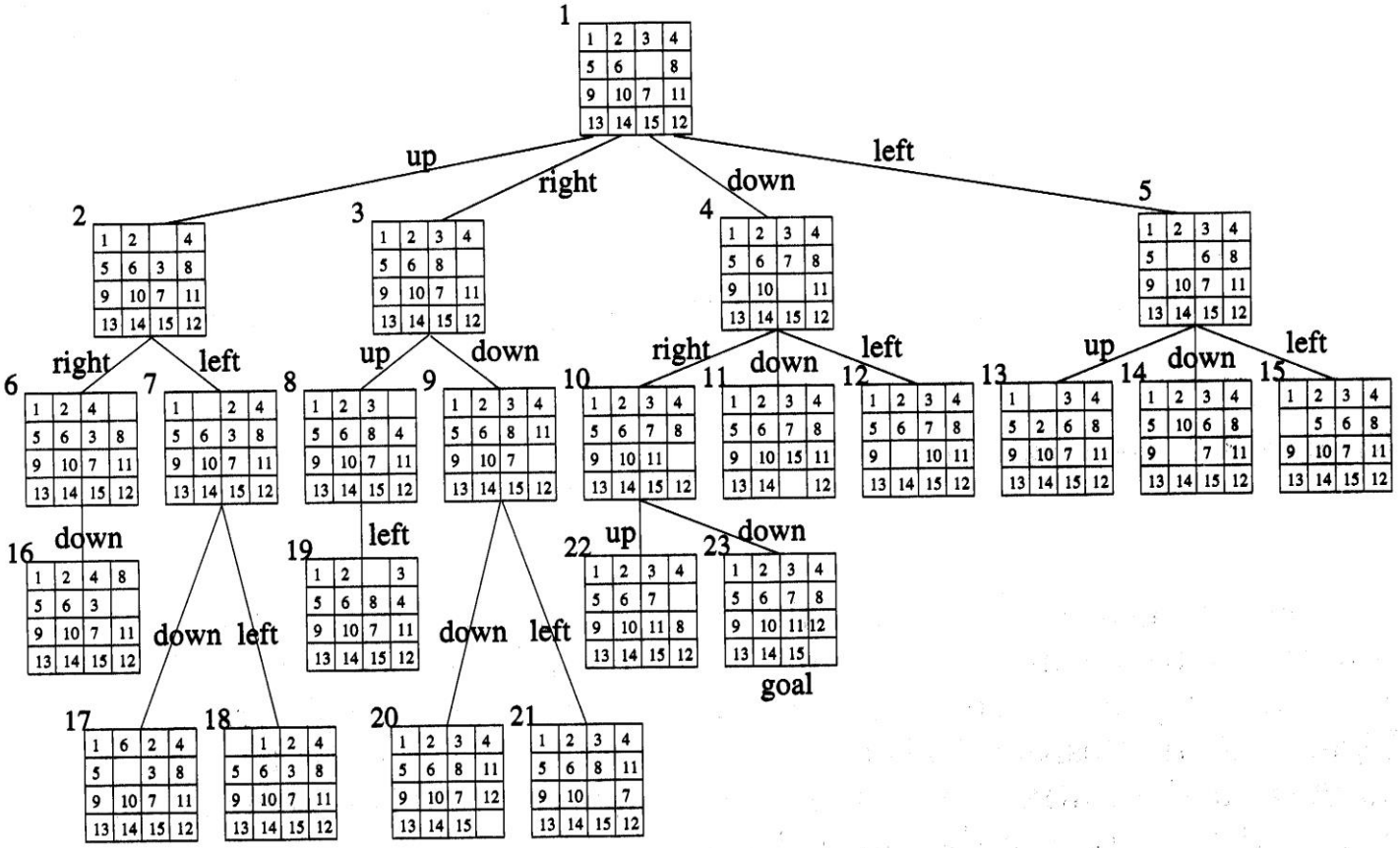
(b) Goal arrangement



(c)

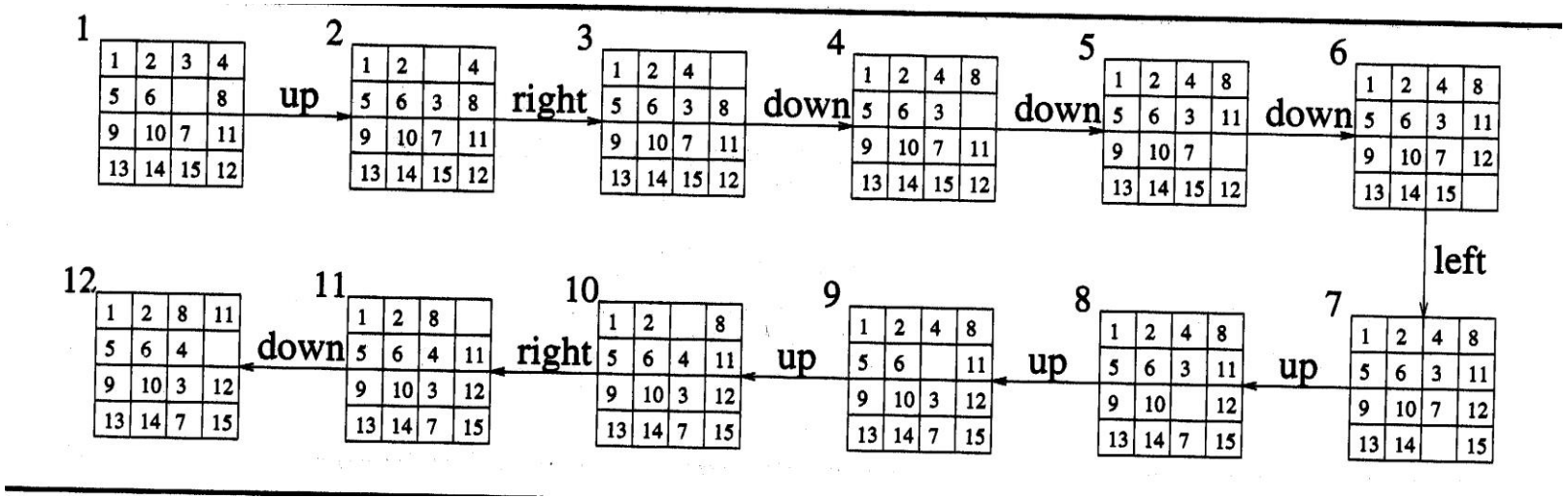
- 각 타일에 번호를 붙이며 목표 배치에 따라 타일들의 번호가 매겨진다.

FIFO-탐색



Edges are labeled according to the direction in which the empty space moves.

깊이 우선 생성



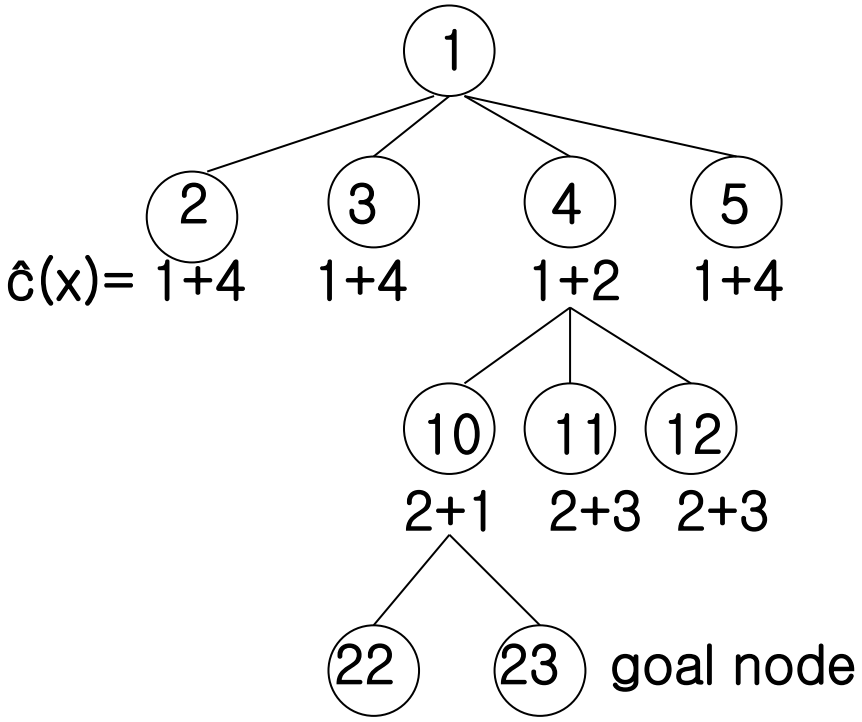
- 해답 상태로 가지 못한다.

LC-탐색

- $\hat{c}(x) = f(x) + \hat{g}(x)$

  f(x) = 루트로부터 노드 x 까지의 경로의 길이

  $\hat{g}(x)$ = 자신들의 목표 위치에 있지 않은 타일들의 수



분기한정 알고리즘 형식

- 순환호출 함수로 작정하지 않는다.

```
void breadth_first_search(tree T) {
    queue Q;
    node u, v;
    initialize(Q);
    v = root of T;
    visit v;
    enqueue(Q,v);   // 큐에 삽입
    while(!empty(Q)) {   // 빈 큐가 될때까지 반복
        dequeue(Q,v);       // 큐에서 삭제
        for(each child u of v) {    // 모든 가능한 자식노드 생성
            visit u;                // 한정함수 적용
            enqueue(Q,u);   // 큐에 삽입
      }
    }
}
```

N-queen 문제의 예(lec12-1)

```
private void queens()   {
       int j, k ;
       int level = 0 ;
      que Q = new que();
       Q.Add(level) ;
       while( !Q.Empty() ) {       // 큐가 빌 때까지 반복
            level = Q.Delete() ;   // 큐에서 레벨 삭제
            for (j = 1; j <= level ; j++ ) col[j] = Q.Delete() ;   // 레벨까지 값 삭제
            // 자식 노드들의 생성
            level++ ;
           for (j = 1; j <= n ; j++) {   // 각각의 자식을 생성한다.
                  col[level] = j ;          // 하나의 자식 생성
                  if (promising(level)) {    // 제약조건 확인
                       if ( level == n ) {
                            for ( k = 1; k <= n ; k++ )    // 해답 노드인 경우
                                  System.out.print(col[k] + "  ") ;
                            System.out.println() ;
                       }
                       else {
                             Q.Add(level) ;       // 해답 노드는 아니나 살아있는 노드인 경우
                             for (k = 1; k <= level ; k++ ) Q.Add(col[k]) ;   // 큐에 삽입
            }     }  // end of if   }  // end of for
      }   // end of while
}
```

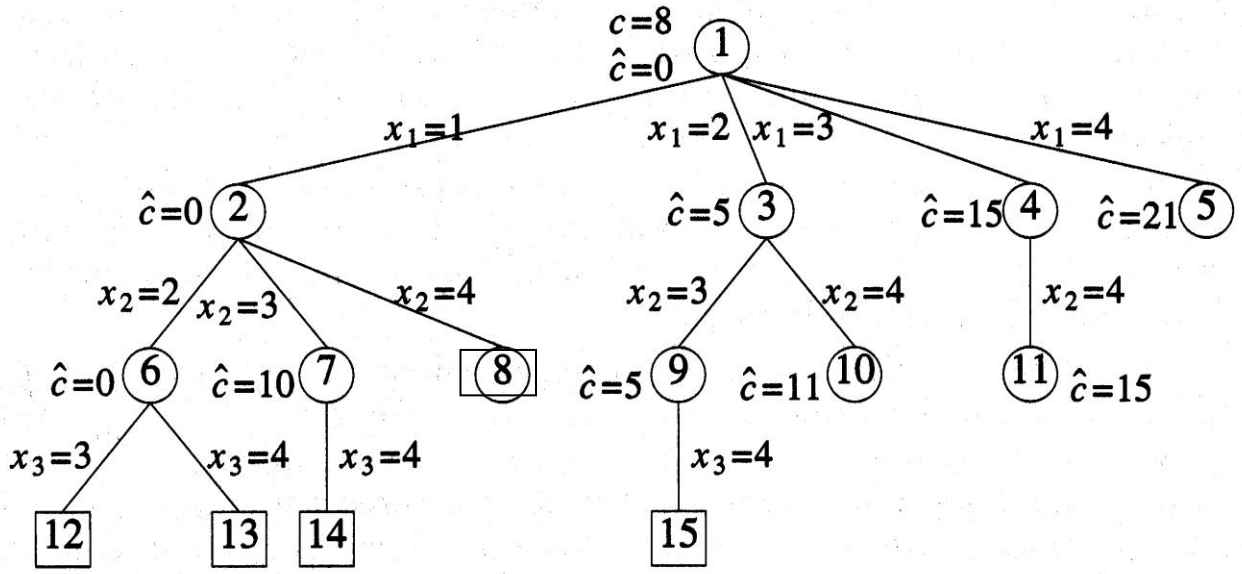
ĉ(x)의 한정(bounding)기능

- 최소-비용 해답 노드를 찾는 문제에서 임의의 노드 x 에서
  ĉ(x) ≤ c(x) 을 만족하도록 ĉ(x)를 결정한다.
  u(x) = 노드 x의 비용의 상한계(upper bound)라 할때,
  upper가 현재까지 확장된 노드의 u 값들 중 최소값일 때,
  ĉ(x) ≥ upper 인 모든 live node x 들을 제거할 수 있다.
- 처음 upper 는 ∞ 또는 가질 수 있는 최대값으로 결정되며, 제거되지 않은 해답 노드(answer node)에서 업데이트된다.
- 예: 마감시간을 갖는 작업 순서화 문제
  → 작업 $i = (p_i, d_i, t_i)$
  t_i 는 작업 i의 처리 시간이며, 이 처리가 마감시간 d_i 까지 완료되지 않는다면 p_i 의 벌금을 물게 된다.
  이때 최소의 벌금을 물게되는 작업의 순서를 구하라.

마감 시간을 갖는 최적의 작업 순서의 예

- n =4, $(p_1,d_1, t_1) = (5, 1, 1)$, $(p_2,d_2, t_2) = (10, 3, 2)$
  $(p_3,d_3, t_3) = (6, 2, 1)$, $(p_4,d_4, t_4) = (3, 1, 1)$
- 이 문제의 해 공간은 집합 {1, 2, 3, 4}의 모든 부분집합이다.
- 가변 길이의 투플

  원형 노드: 해 노드, 사각형 노드: 해 노드가 아님



비용함수와 추정함수

- 비용 함수: c(x) = x를 루트로 하는 부분 트리에 속한 노드들에 대응하는 최소의 벌금
  사각형 노드의 c(x) = ∞,
  c(1)=8, c(2)=9, c(3)=8, …
  그러나 처음부터 계산할 수 없다.
- 추정함수: $\hat{c}(x) = \sum_{i<m} p_i$, i 는 S_x 에 포함 안된 작업
  S_x = 노드 x가 포함하는 작업들의 집합
  $m = \max \{i \mid i \in S_x\}$
- 이때 $\hat{c}(x) \leq c(x)$ 이 성립된다.
- 상한 함수 u(x) = 노드 x 에 포함 안된 작업들의 벌금 합

FIFO 분기와 한정

- (앞의 그림)
- upper = ∞ 또는 $\sum_{1 \le i \le 4} p_i = 24$
- 처음 E-노드: 노드 1 → 노드 2, 3, 4, 5 가 차례로 생성된다.

  (세부 단계)

  노드 2: u(2)=19 , ĉ(x)=0, upper=19로 update

  노드 3: u(3)=14 , ĉ(x)=5, upper=14로 update

  노드 4: u(4)=18, ĉ(x)=15 ≥ upper 이므로 제거됨

  노드 5: u(5)=21, ĉ(x)=21 ≥ upper 이므로 제거됨
- 다음 E-노드: 노드 2 (FIFO 순) → 노드 6, 7, 8 이 차례로 생성된다.

  노드 6: u(6)=9, ĉ(x)=0, upper=9로 update

  노드 7: u(7)=13, ĉ(x)=10 ≥ upper 이므로 제거됨

  노드 8: 해답 노드가 될 수 없으므로 제거됨

- 다음 E-노드: 노드 3 → 노드 9, 10 이 차례로 생성된다.
  노드 9: u(9)=8 , ĉ(x)=5, upper=8로 update
  노드10: u(10)=11, ĉ(x)=11 ≥ upper 이므로 제거됨
- 다음 E-노드: 노드 6 → 자식 노드들은 모두 해답 노드가 될 수 없다.
- 다음 E-노드: 노드 9 → 자식 노드는 해답 노드가 될 수 없다.
- 더 이상 live node 가 없으므로 종료한다.
- 여기서 최소-비용 해답 노드는 노드 9이고, 최소 비용(벌금)은 8이 된다.

LC 분기와 한정

- (앞의 그림)
- upper = ∞ 또는 $\sum_{1 \le i \le 4} p_i = 24$
- 처음 E-노드: 노드 1 → 노드 2, 3, 4, 5 가 차례로 생성된다.

  (세부 단계)

  노드 2: u(2)=19 , ĉ(x)=0, upper=19로 update

  노드 3: u(3)=14 , ĉ(x)=5, upper=14로 update

  노드 4: u(4)=18, ĉ(x)=15 ≥ upper 이므로 제거됨

  노드 5: u(5)=21, ĉ(x)=21 ≥ upper 이므로 제거됨
- 다음 E-노드: 노드 2 (ĉ(2)=0 < ĉ(3)=5 이므로)

  → 노드 6, 7, 8 이 생성됨

  노드 6: u(6)=9 , ĉ(x)=0, upper=9로 update

  노드 7: u(7)=13, ĉ(x)=10 ≥ upper 이므로 제거됨

  노드 8: 해답 노드가 될 수 없으므로 제거됨

- 다음 E-노드: 노드 6 (ĉ(6)=0 < ĉ(3)=5 이므로)
  → 자식 노드들은 모두 해답노드가 될 수 없다.
- 다음 E-노드: 노드 3 → 노드 9, 10 이 차례로 생성된다.
  노드 9: u(9)=8 , ĉ(x)=5, upper=8로 update
  노드10: u(10)=11, ĉ(x)=11 ≥ upper 이므로 제거됨
- 다음 E-노드: 노드 9 → 자식 노드는 해답노드가 될 수 없다.
- 더 이상 live node가 없으므로 종료한다.
- 여기서 최소-비용 해답 노드는 노드 9이고, 최소 비용(벌금)은 8이 된다.

고정길이 투플 방식

- FIFO 분기와 한정
- LC 분기와 한정