Branch And Bound

6장에서 냅색 알고리즘을 사용할 때,

Breadth - First Search와 Best-First Search로 나뉜다.

1. Breadth - First Search는 Normal Queue
2. Best - First Search는 Priority Queue

Best-Frist Search는 가장 좋은 한계치(잠재 능력 = bound)를 가진 노드 순으로 큐에서 꺼내, 이를 위해 우선 순위 큐를 사용한다.

Breadth - First Search to 0-1 Knapsack

❑ Breadth-First Search with Branc

```
int  knapsack2(int n,  int[ ] p,  int[ ] w, int W)
{
      queue_of_node Q ;  node u, v ;  int maxProfit ;

      initialize(Q) ;
      v.level = 0 ; v.profit = 0; v.weight=0 ;
      maxProfit = 0 ;
      enqueue(Q,v) ;
      while(! Empty(Q) ){
            dequeue(Q,v) ;
            u.level = v.level + 1 ;
            take care of the left child ;
            take care of the right child ;
      }
      return maxProfit ;
}
```

❑ Breadth-First Search with Branch and Bound

Left Child

```
u.weight = v.weight + w[u.level] ;
u.profit = v.profit + p[u.level] ;
if (u.weight<=W && u.profit > maxProfit)
    maxProfit = u.profit ;
if (bound(u) > maxProfit)
    enqueue(Q,u) ;
```

Right Child

```
u.weight = v.weight ;
u.profit = v.profit ;
if (bound(u) > maxProfit)
    enqueue(Q,u) ;
```

```
public class node
{
    int level ;
    int profit ;
    int weight ;
}
```

Left Child는 w[u.level]과 p[u.level] 즉 child의 값을 더한 것.

Right Child는 child의 값을 더하지 않은 것.

1. 현재 차일드 값을 포함한 게 maxProfit과 비교하여 갱신한다.
2. Left와 Right 모두 유망한 지 판단 후 큐에 넣는다.

```
public static float bound(node u)
{
      index j,k ;  int totWeight ; float result ;
      if (u.weight >= W) return 0 ;
      else {
            result = u.profit ;
            j = u.level + 1 ;
            totWeight = u.weight ;
            while (j<=n && totWeight+w[j] <= W){
              totWeight = totWeight + w[j] ;
              result = result + p[j] ;
              j++ ;
            }
            k = j ;
            if (k <= n)
            result=result+(W-totWeight)*p[k]/w[k];
            return result ;
      }
}
```

Best First Search to 0-1 Knapsack Problem

```
public static int knapsack3(int n, int[ ] p, int[ ] w, int W)
{
      priority_queue_of_node PQ ;  node u, v ;
      int maxProfit ;
      v.level = 0 ; v.profit = 0; v.weight=0 ; maxProfit = 0 ;
      v.bound = bound(v) ;
      PQ.enqueue(v) ;
      while( ! PQ.Empty() ){
            v = PQ.dequeue() ;
            if (v.bound > maxProfit) {
                u.level = v.level + 1 ;
                take care of the left child ;
                take care of the right child ;
            }
      }
}
```

Left Child

```
u.weight = v.weight + w[u.level] ;
u.profit = v.profit + p[u.level] ;
if (u.weight<=W && u.profit > maxProfit)
    maxProfit = u.profit ;
u.bound = bound(u) ;
if (u.bound > maxProfit)
    PQ.enqueue(u) ;
```

Right Child

```
u.weight = v.weight ;
u.profit = v.profit ;
u.bound = bound(u) ;
if ( u.bound > maxProfit)
    PQ.enqueue(u) ;
```

```
public static float bound(node u)
{
      index j,k ;  int totWeight ; float result ;
      if (u.weight >= W) return 0 ;
      else {
            result = u.profit ;
            j = u.level + 1 ;
            totWeight = u.weight ;
            while (j<=n && totWeight+w[j] <= W){
              totWeight = totWeight + w[j] ;
              result = result + p[j] ;
              j++ ;
            }
            k = j ;
            if (k <= n)
            result=result+(W-totWeight)*p[k]/w[k];
            return result ;
      }
}
```

page 21

page 28

TSP - branch and bound O(n!)

- **Example:**

$W =$

0	14	4	10	20	→ min : 4
14	0	7	8	7	→ min : 7
4	5	0	7	16	→ min : 4
11	7	9	0	2	→ min : 2
18	7	17	4	0	→ min. : 4

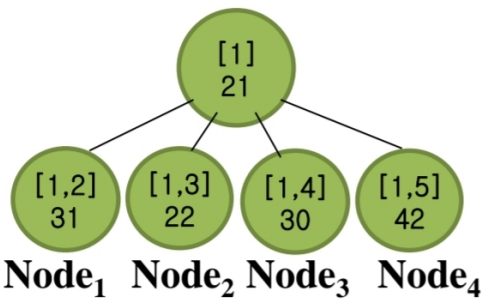
V_1 에서 lower bound

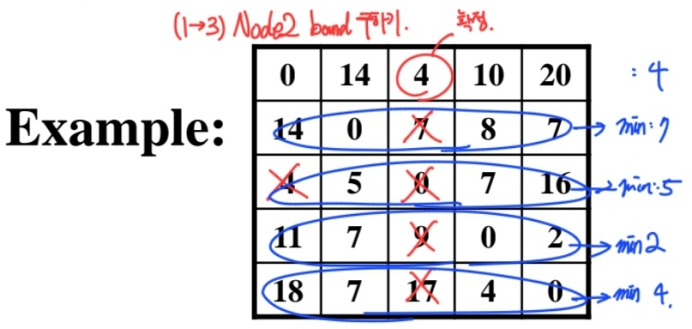
The start node is V_1.

➔ the lower bound on the *root* node

$$= \sum_{v_m \in V} (lowest\ weight\ of\ edge\ leaving\ v_m) = 4 + 7 + 4 + 2 + 4$$

노드 2 lower bound 구하기





노드 5 lower bound 구하기

노드 5는 1→3→2 로 간다고 가정하고, 이 뒤에 일어나는 경로들의 lower bound를 계산

1 → 3 간선 값 4

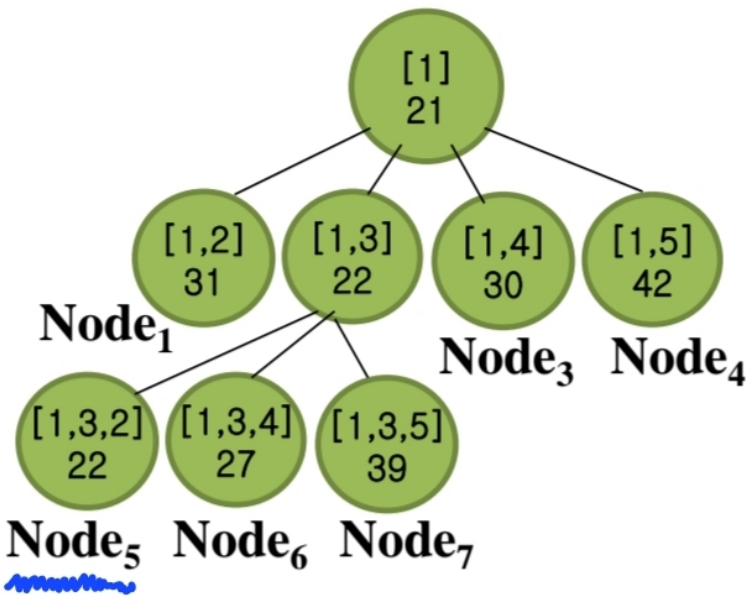
3 → 2 간선 값 5

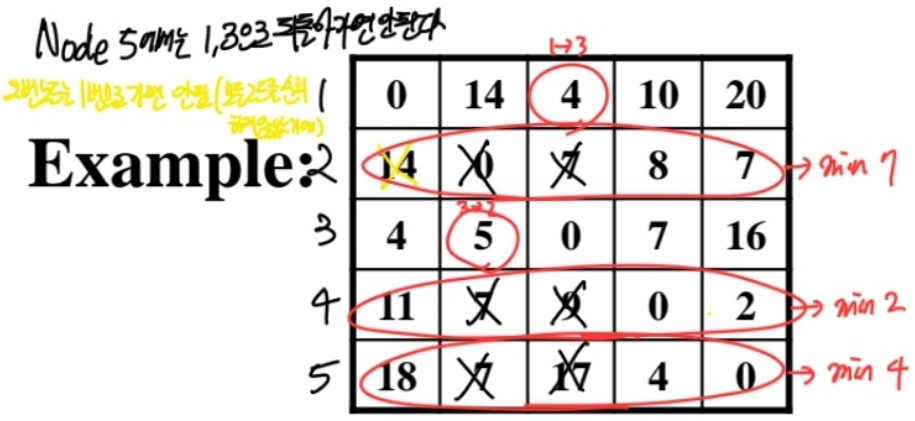
위 두개는 확정이다.

갔던 곳을 다시 가거나, 모든 노드를 순회하지 않았는데 1로 돌아가는 오류를 제외하기 위해서 배열에 X 표시한다.

1. 2, 4, 5번 노드에서는 3, 2 번 노드로 다시는 가지 않는다. - 검은색 x
2. 노드 2에서는 노드 1로 바로 가지 않는다. (1→3→2→1 의 경우는 모든 노드를 순회하지 않았기에) - 노란색 x

이러고 나서 2, 4, 5 번 노드들이 간선을 선택한다. (min)





➔ Lower Bound on Minimum Cost Tour

$Node_5 : 4+5 + (7 + 2 + 4) = 22$

$Node_6 : 4+7 + (7 + 2 + 7) = 27$

$Node_7 : 4+16 + (8 + 7 + 4) = 39$

노드 6 lower bound 구하기

노드 6은 1→3→4로 간다고 가정,

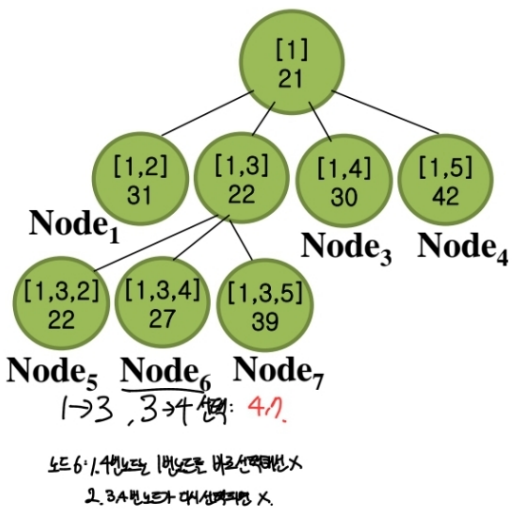
1 → 3 간선 값 4

3 → 4 간선 값 7

위 두개는 확정

1. 2 ,4 ,5번 노드에서는 3, 4번 노드로 다시는 가지 않는다.
2. 1→3→4 번노드에서 다음 노드로 1번 노드를 선택하지 않는다. arr[4][1]은 선택 X

현재 PQ 순서 (5, 6, 3, 1, 7, 4)



Example:

~~0~~	~~14~~	~~4~~	~~10~~	~~20~~
14	0	~~7~~	~~8~~	7
~~4~~	~~5~~	~~0~~	7	~~16~~
~~11~~	7	~~9~~	~~0~~	2
18	7	~~17~~	~~4~~	0

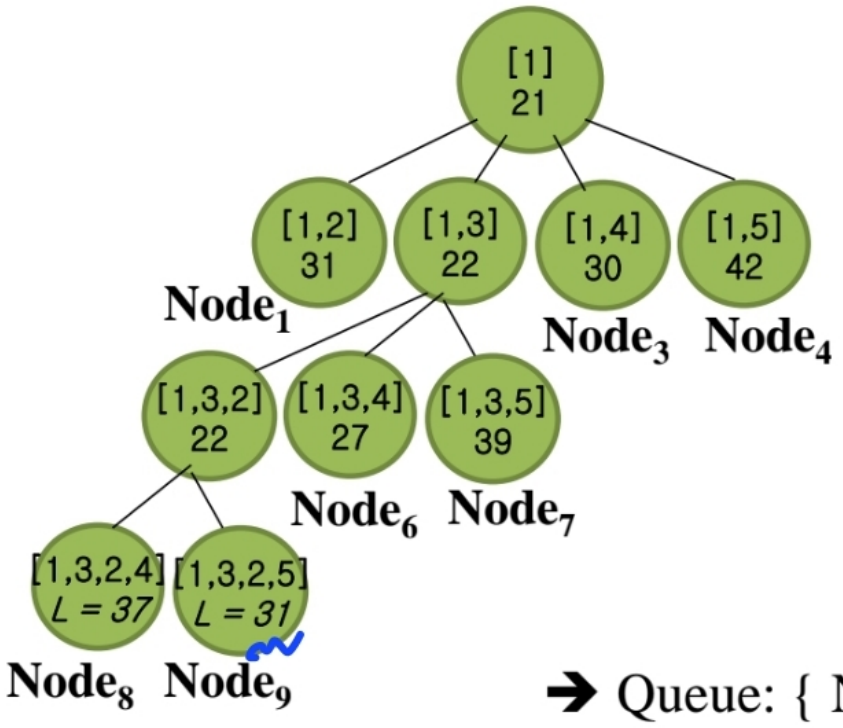
➔ Lower Bound on Minimum Cost Tour

$Node_5$: 4+5 + (7 + 2 + 4) = 22

$Node_6$: 4+7 + (7 + 2 + 7) = 27

$Node_7$: 4+16 + (8 +7 + 4) = 39

이 다음 PQ에서 꺼낸 노드(노드 5)의 자녀 노드들의 bound를 계산한다.
어라라? 이녀석들은 level이 n-2니까 최종적인 계산이 되는 놈들이구만 → 최종 계산을 한다.



Example:

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

➔ Lower Bound on Minimum Cost Tour

$Node_8$: 4+5+8 + (2+18) = 37

$Node_9$: 4+5+7 + (4+11) = 31

➔ Queue: { $Node_6$, $Node_3$, $Node_1$, $Node_7$, $Node_4$}

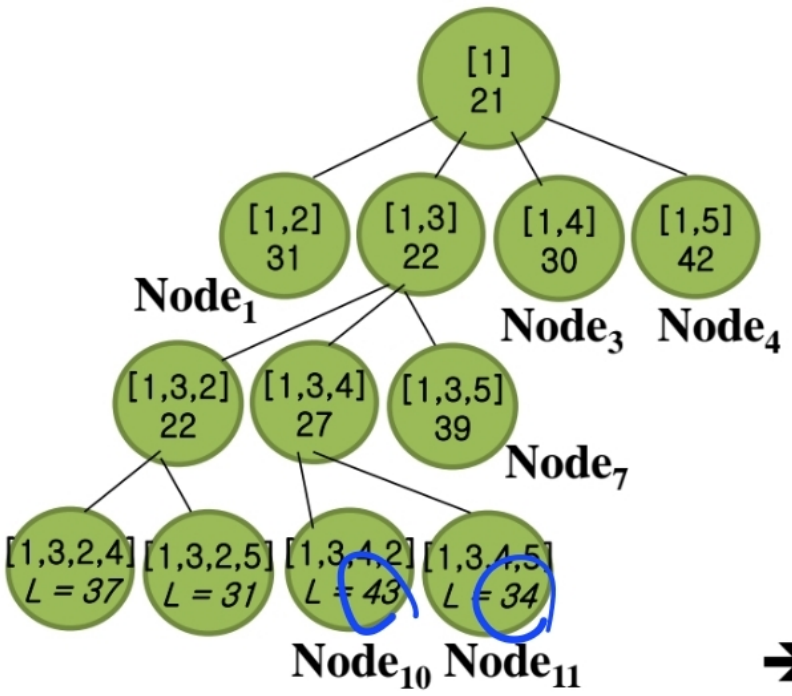
➔ Current best solution = 31

34

노드8과 9의 경로를 따랐을 때 최종 경로의 weight합은 37과 31이다.
31이 더 작으니, Current best solution = 31로 저장한다. 이 다음 수행으로는 PQ에서 값을 꺼내 자녀노드들을 검색할 준비를 한다.

(노드8, 9는 최종 경로 계산이 끝난 노드이므로 PQ에 넣지 않는건 당연!)

PQ에는 (6, 3, 1, 7, 4)가 들어있으니 노드 6의 자식 노드들의 bound를 계산하자.



Example:

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

➔ Lower Bound on Minimum Cost Tour

$Node_{10}$: *4+7+7* + (7 + 18) = 43

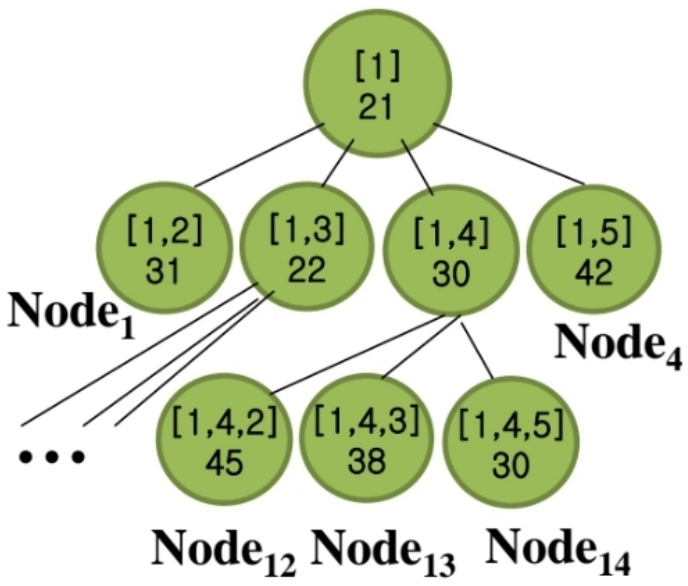
$Node_{11}$: 4+7+2 + (7 + 14) = 34

➔ Queue: { $Node_3$, $Node_1$, $Node_7$, $Node_4$}

➔ Current best solution = 31

노드 6의 자녀 노드들의 bound를 구하려고 보니 level == n-2 로써 최종 경로로 weight 합을 구할 수 있다. 노드 10은 43, 노드 11은 34이다. 근데 Current best solution에 저장된 값보다 크므로 의미없다. 불합격!

현재 PQ는 (3, 1, 7, 4)이므로 노드 3의 자식 노드들을 꺼낸다.



Example:

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

➔ Lower Bound on Minimum Cost Tour

$Node_{12}$: *10+7+* (7 +4 + 17) = 45

$Node_{13}$: *10+9+* (7 +5 + 7) = 38

$Node_{14}$: *10+2*+ (7 +4 + 7) = 30

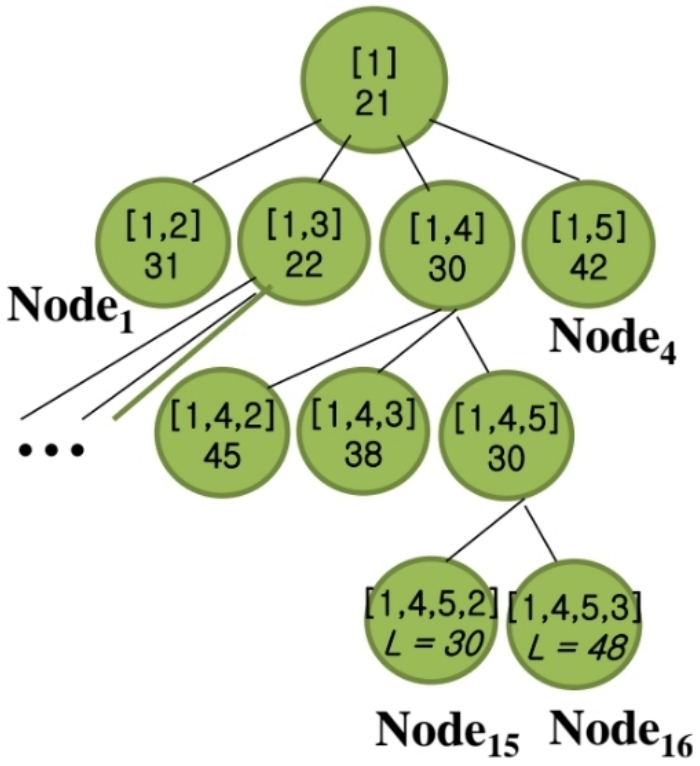
➔ Queue: { $Node_{14}$, $Node_1$,$Node_7$, $Node_4$}

➔ Current best solution = 31

노드 3의 자식노드는 노드 12, 13, 14로서 level == n-2 가 아니므로 bound를 계산한다.

여기서 바운드(한계치)가 Current best solution보다 높으면 잠재성이 부족한 (유망하지 않은) 경로이므로 PQ에 저장하지 않는다. 따라서 노드 12와 노드 13은 무시하고 노드 14만 PQ에 담긴다.

현재 PQ는 우선순위 큐이므로 (14, 1, 7, 4) 순으로 담겨있다.



Example:

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

➔ Lower Bound on Minimum Cost Tour

$Node_{15}$: *10+2+7* + (7 + 4) = 30

$Node_{16}$: 10+2+17 + (5 + 14) = 48

➔ Queue: { $Node_1$, $Node_7$, $Node_4$}

➔ Current best solution = 30

노드 14의 자식노드 노드 15, 16의 bound를 계산하려고 보니 level == n-2에 해당하므로 최종 경로를 통한 weight의 합을 계산한다. 어라랏~ 노드 15를 통한 weight합이 30으로써 Current best solution보다 낮다. 갱신해주자.

위 과정을 반복해 current best solution 값을 갱신해 최종적인 값을 사용한다.

TSP - 슈도코드

```
public static number travel2(int n, number[ ] W,
                             node optTour)
{
   priority_queue_of_node PQ;    node u, v ;
   number minLength ;

   PQ.initialize() ;
   v.level = 0; v.path =[1]; minLength= ∞;
   v.bound=bound(v);
   PQ.enqueue(v) ;
   while(! PQ.Empty() ) {
        v = PQ.dequeue() ;
        if v is promising    // the bound of v < minLength

   }
}
```

take_care_of_children

```
u.level = v.level + 1;
for (all i such that 2 ≤ i ≤ n && i not in v.path) {
    u.path = v.path ; put i at the end of u.path;
    if ( u.level == n-2 ) {
        put index of only vertex not in u.path at the end of u.path ;
        put 1 at the end of u.path;
        if (length(u) < minLength) {
          minLength = length(u) ;  optTour = u.path ;
        }
    }
    else {
        u.bound = bound(u) ;
        if (u.bound < minLength )
            PQ.enqueue(u) ;
    }
```

경로가 정해졌을 경우

u경로에 선택X 정점들 Put

u의 마지막 point까지 경로.

u의 마지막 정점 다음으로 1로 되돌아간다.

아직 경로가 정해지지 (트리로 갈 수 있음)

최소

한 개라도 현재까지 계산된 경로의 값보다 작으면 PQ에 넣는다.

https://develop247.tistory.com/71