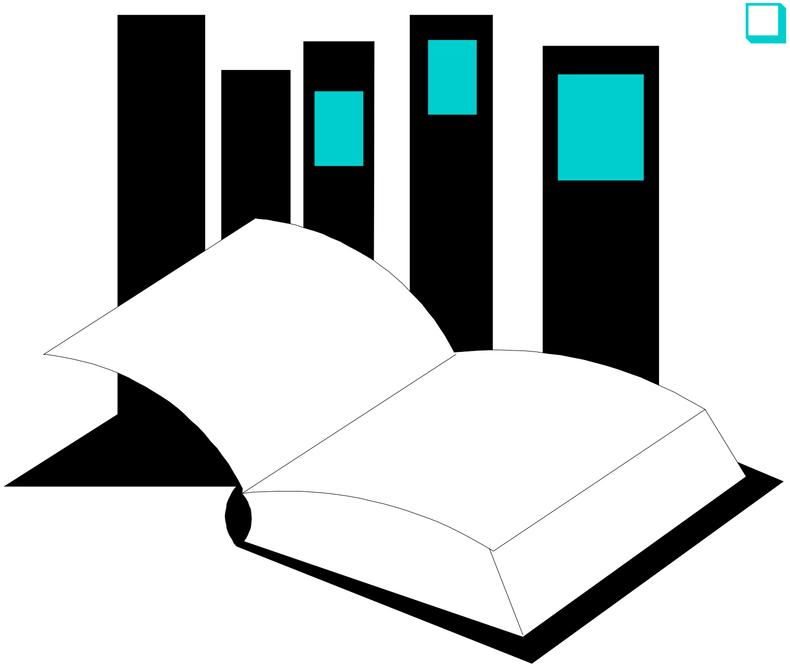
Chapter 6. Branch and Bound



- Chapter 6 introduces an algorithm design technique called “***Branch and Bound***”.

CHAPTER 6
Foundations of Algorithms

Branch and Bound

- ***Similar*** to "Backtracking"
  - a $\boxed{\text{state-space tree}}$ is used to solve a problem
- ***Different*** from "Backtracking"
  - does not limit us to any particular way of traversing a tree
  - is used ***only for optimization problems***

Branch and Bound

- Step 1:

  - computes a number (bound) at a node to determine whether the node is ***promising***

    (the number is a bound on the value of the solution that could be obtained by expanding beyond the node)

- Step 2:

  - if the bound is no better than the value of the best solution found so far, the node is ***non-promising*****.**

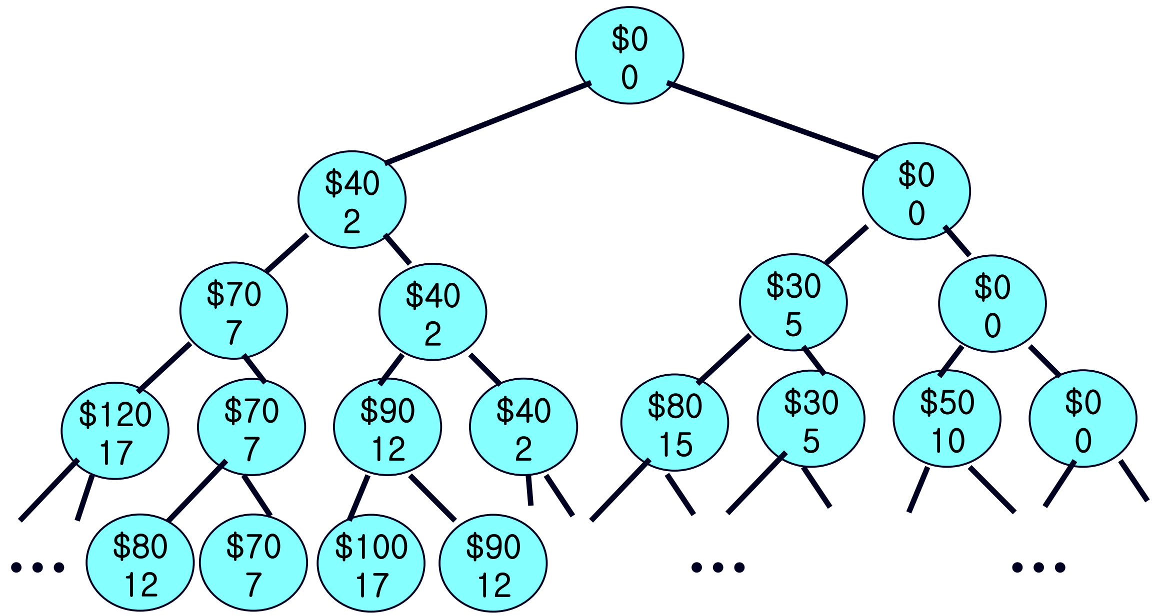
6.1 The 0-1 Knapsack Problem

- **Breadth-First Search with Branch and Bound Pruning**

Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

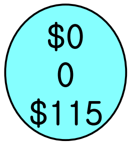
W = 16



6.1 The 0-1 Knapsack Problem

Breadth-First Search with Branch and Bound Pruning

Node$_o$



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

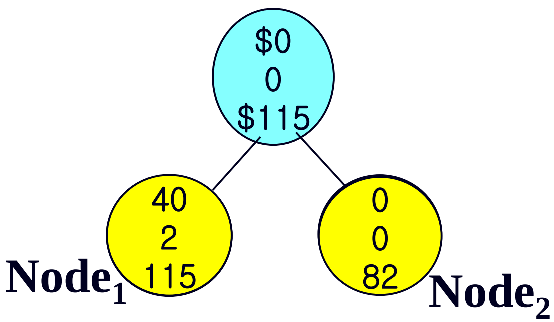
W = 16

- ➔ Bound on Maximum Possible Profit:

  Node$_1$: **40** + 30 + (50 * 9/10) = 115
- ➔ Queue: { Node$_o$ }
- ➔ Current best solution = 0

6.1 The 0-1 Knapsack Problem

Breadth-First Search with Branch and Bound Pruning



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2
W = 16			

➔ Bound on Maximum Possible Profit:

Node_1 : **40** + 30 + (50 * 9/10) = 115

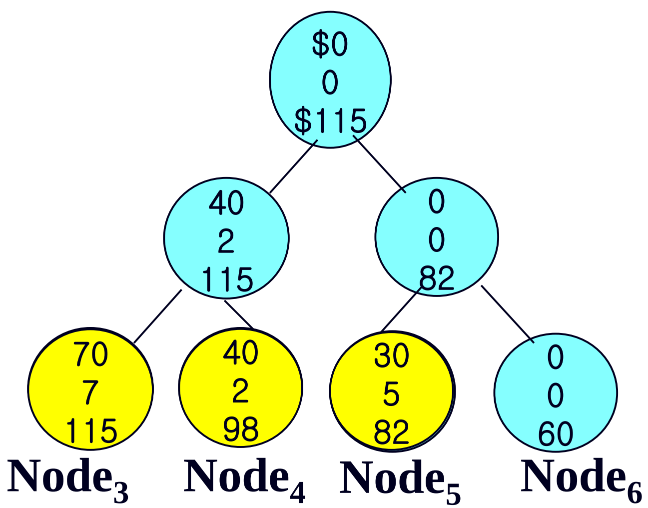
Node_2 : **0** + 30 + 50 + (10*1/5) = 82

➔ Queue: { Node_1 , Node_2 }

➔ Current best solution = 40

6.1 The 0-1 Knapsack Problem

Breadth-First Search with Branch and Bound Pruning



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2
W = 16			

➔ Bound on Maximum Possible Profit:

$Node_3$: **40** + **30** + (50 * 9/10) = 115

$Node_4$: **40** + **0** + 50 + (10*4/5) = 98

$Node_5$: **0** + **30** + 50+(10 * 1/5) = 82

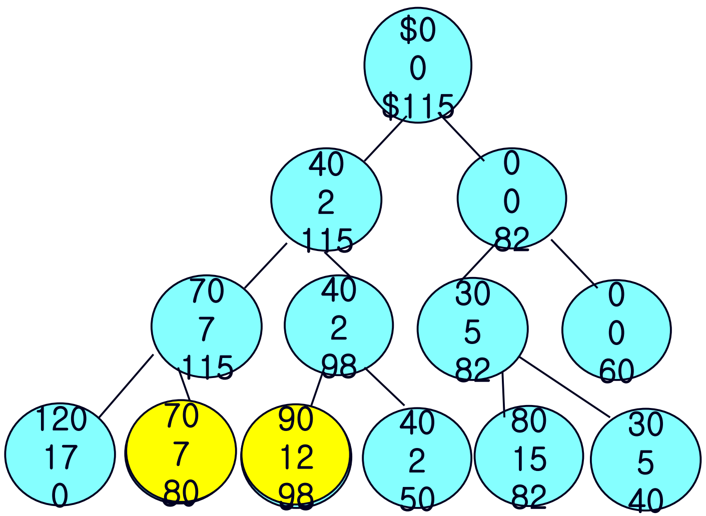
$Node_6$: **0** + **0** + 50 + 10 = 60

➔ Queue: { $Node_3$, $Node_4$, $Node_5$ }

➔ Current best solution = 70

6.1 The 0-1 Knapsack Problem

Breadth-First Search with Branch and Bound Pruning



Node$_7$ **Node$_8$** **Node$_9$** **Node$_{10}$** **Node$_{11}$** **Node$_{12}$**

➔ Queue: { Node$_8$, Node$_9$ }

➔ Current best solution = 90

Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

➔ Bound on Maximum Possible Profit:

Node$_7$: **0 (overweight)**

Node$_8$: **40** + **30** + **0** + 10 = 80

Node$_9$: **40** + **0** + **50**+(10 * 4/5) = 98

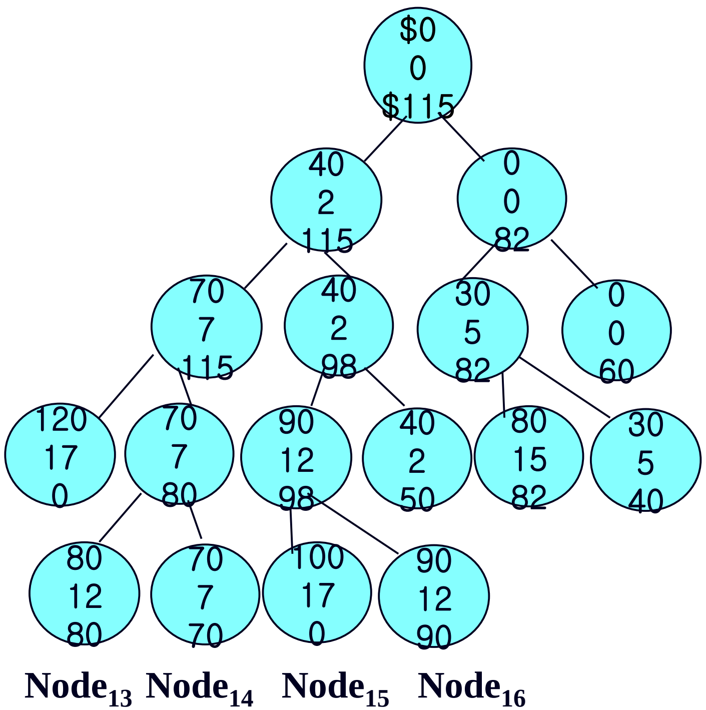
Node$_{10}$: **40** + **0** + **0** + 10 = 50

Node$_{11}$: **0** + **30** + **50**+(10 * 1/5) = 82

Node$_{12}$: **0** + **30** + **0** + 10 = 40

6.1 The 0-1 Knapsack Problem

Breadth-First Search with Branch and Bound Pruning



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

➔ Bound on Maximum Possible Profit:

Node_{13} : **40** + **30** + **0** + **10** = 80

Node_{14} : **40** + **30** + **0** + **0** = 70

Node_{15} : **0 (overweight)**

Node_{16} : **40** + **0** + **50** + **0** = 90

➔ Queue: { }

➔ **Current best solution = 90**

6.1 The 0-1 Knapsack Problem

Breadth-First Search with Branch and Bound

```
public static 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 ;
}
```

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

6.1 The 0-1 Knapsack Problem

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 ;
}
```

6.1 The 0-1 Knapsack Problem

- ***Best-First* Search with Branch and Bound Pruning**
  - **Basic Idea**

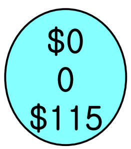
    - uses ***bound*** to ***select a node to expand next*****,** rather than just determine whether a node is promising

    - uses a ***priority queue*** of nodes where the priority is determined by the bound value of a node

6.1 The 0-1 Knapsack Problem

❑ ***Best-First*** **Search with Branch and Bound Pruning**

Node$_o$



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

➔ Bound on Maximum Possible Profit:

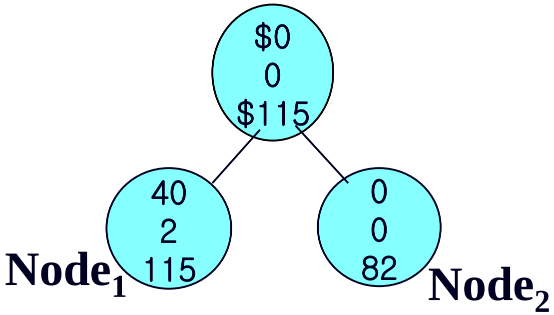
Node$_0$: **40** + 30 + (50 * 9/10) = 115

➔ Queue: { Node$_o$ }

➔ Current best solution = 0

6.1 The 0-1 Knapsack Problem

❑ ***Best-First*** **Search with Branch and Bound Pruning**



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2
W = 16			

➔ Bound on Maximum Possible Profit:

Node$_1$: **40** + 30 + (50 * 9/10) = 115

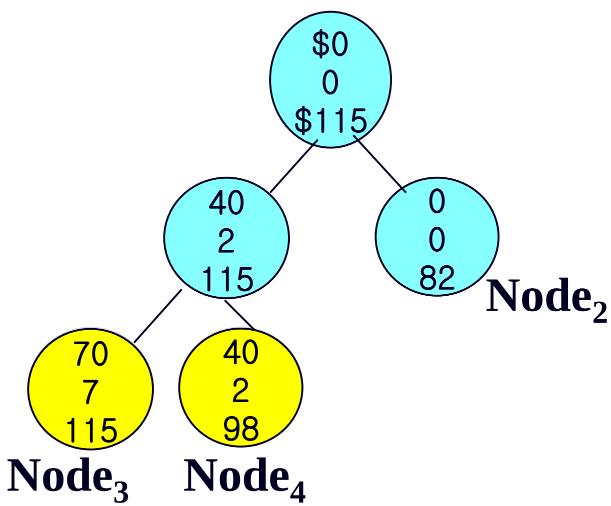
Node$_2$: **0** + 30 + 50 + (10*1/5) = 82

➔ Queue: { Node$_1$, Node$_2$ }

➔ Current best solution = 40

6.1 The 0-1 Knapsack Problem

❑ *Best-First* Search with Branch and Bound Pruning



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

➔ Bound on Maximum Possible Profit:

$Node_3$: **40** + **30** + (50 * 9/10) = 115

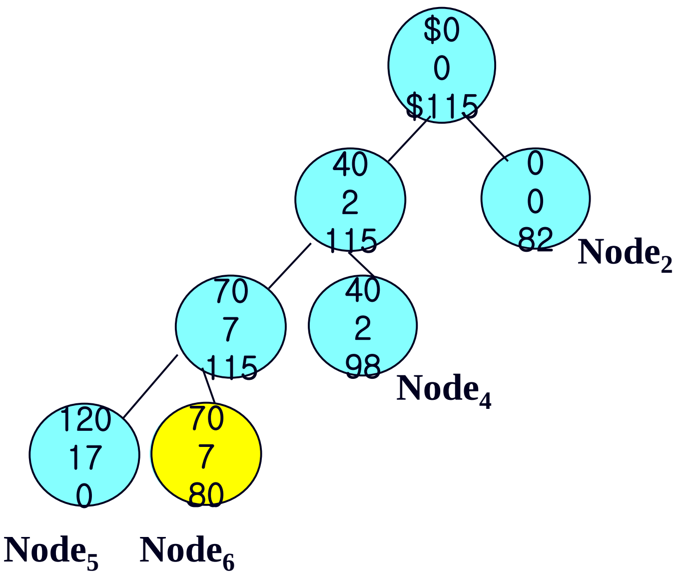
$Node_4$: **40** + **0** + 50 + (10*4/5) = 98

➔ Queue: { $Node_3$, $Node_4$, $Node_2$ }

➔ Current best solution = 70

6.1 The 0-1 Knapsack Problem

- ***Best-First* Search with Branch and Bound Pruning**



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

→ Bound on Maximum Possible Profit:

$Node_5$: **0 (overweight)**

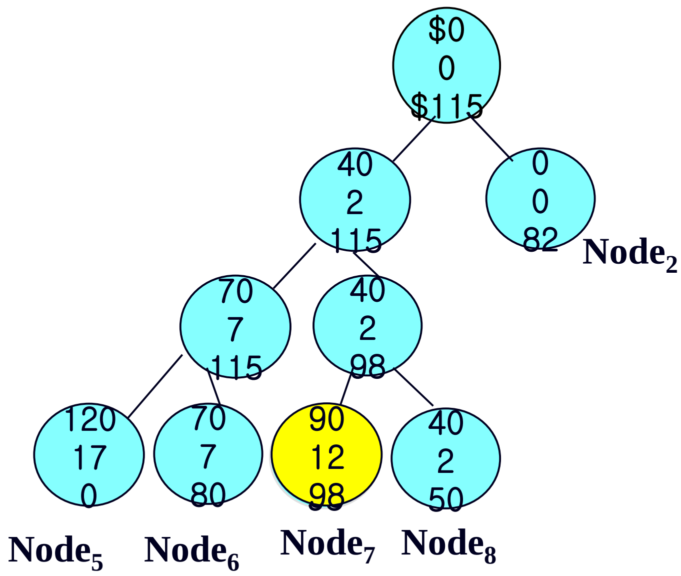
$Node_6$: **40** + **30** + **0** + 10 = 80

→ Queue: { $Node_4$, $Node_6$, $Node_2$ }

→ Current best solution = 70

6.1 The 0-1 Knapsack Problem

❑ *Best-First* Search with Branch and Bound Pruning



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

➔ Bound on Maximum Possible Profit:

$Node_7$: **40** + **0** + **50** + (10 * 4/5) = 98

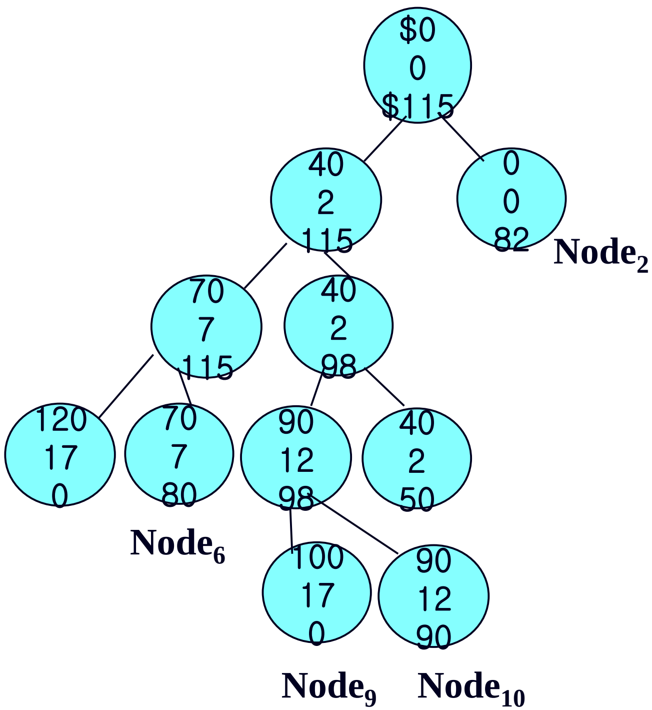
$Node_8$: **40** + **0** + **0** + 10 = 50

➔ Queue: { $Node_7$, $Node_2$, $Node_6$ }

➔ Current best solution = 90

6.1 The 0-1 Knapsack Problem

❑ *Best-First* Search with Branch and Bound Pruning



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

➔ Bound on Maximum Possible Profit:

Node$_9$: **0 (overweight)**

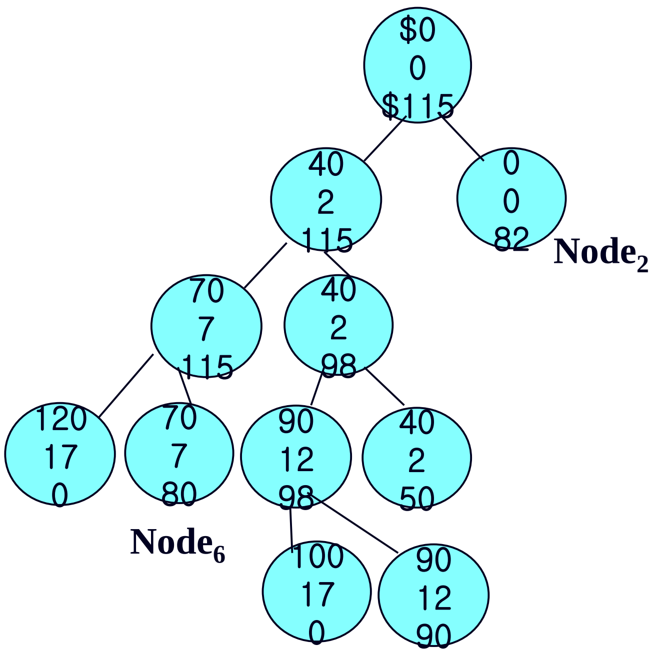
Node$_{10}$: **40** + **0** + **50** + **0** = 90

➔ Queue: { Node$_2$, Node$_6$ }

➔ **Current best solution = 90**

6.1 The 0-1 Knapsack Problem

❑ ***Best-First*** **Search with Branch and Bound Pruning**



Example:

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16

➔ Since both of Node$_2$ and Node$_4$ have bound values less than 90, they will ***not be expanded further.***

➔ Queue: { }

➔ **Final best solution = 90**

6.1 The 0-1 Knapsack Problem

❑ *Best-First* Search with Branch and Bound

```
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 ;
                }
        }
}
```

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

6.1 The 0-1 Knapsack Problem

- ***Best-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 ;
u.bound = bound(u) ;
if (u.bound > maxProfit)
    PQ.enqueue(u) ;
```

Right Child

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

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

6.1 The 0-1 Knapsack Problem

Best-First Search with Branch and Bound

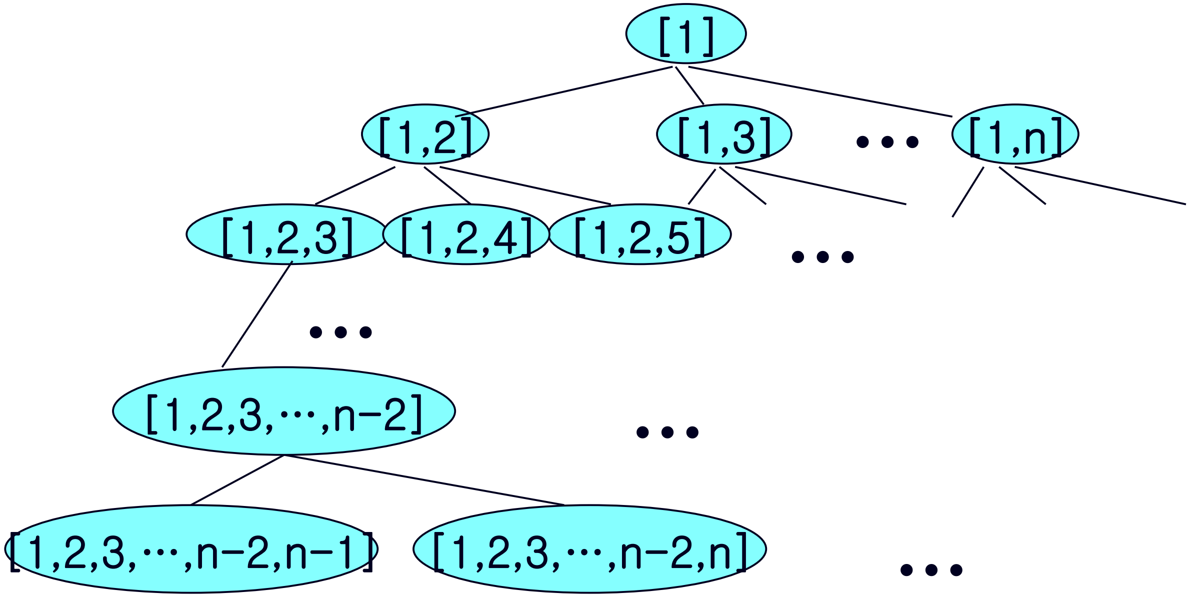
```
public static float bound(node u)
{
      index j,k ;  int totWeight ; float result ;
      if (u.weight >= W) return 0 ;
      else {
            result = u.profit ;
            j = u.level + 1 ;
            totWeight = u.weight ;
            while (j<=n && totWeight+w[j] <= W){
               totWeight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

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

6.2 The Traveling SalesPerson Problem

- **The Branch and Bound Approach to T.S.P.**

Given a directed graph with n nodes, let $[i_1, i_2, \ldots, i_k]$ be a path from i_1 to i_k passing through $i_2, i_3, \ldots,$ and i_{k-1}



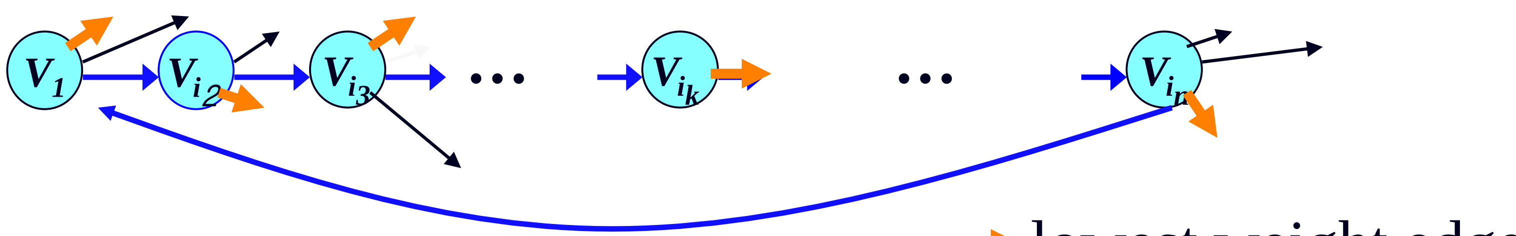
6.2 The Traveling SalesPerson Problem

- **The Branch and Bound Approach to T.S.P.**
  - **How to compute the *bound* on each node?**

    - At the level k of the state space tree, each node corresponds to a state where (k+1) vertices have been visited.

    - **lower bound on the *root* node**

      $$= \sum_{v_m \in V} (\textit{lowest weight of edge leaving } v_m)$$



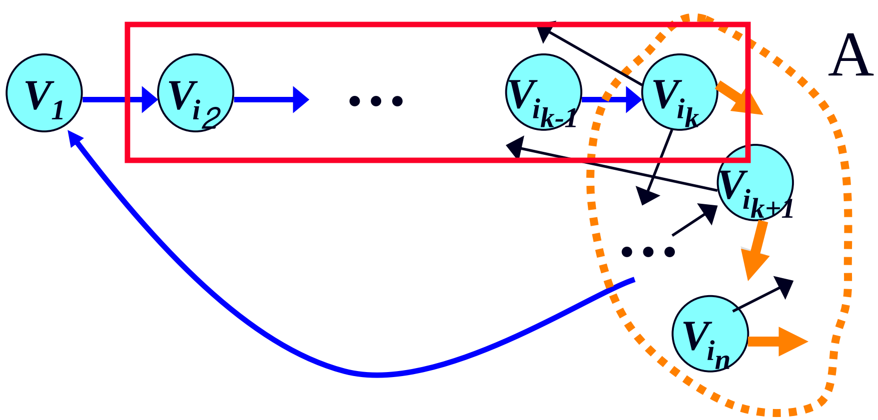
6.2 The Traveling SalesPerson Problem

- **The Branch and Bound Approach to T.S.P.**
  - **lower bound on node** ***[1, i_2, ... , i_k]*** (1 < k < n)

= sum of actual weight from V_1 to V_{ik}

$+ \sum_{v_m \in A}$ (*lowest weight of edge leaving V_m excluding those to vertices i_2, ..,i_k and the edge from Vi_k to V_1*)

where $A = V - \{V_1, Vi_2, .., Vi_{k-1}\}$



6.2 The Traveling SalesPerson Problem

- **The Best-First Search with Branch and Bound**
  - **Example:**

W =

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

The start node is V_1.

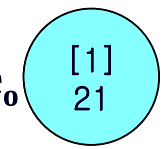
➔ the lower bound on the *root* node

$$= \sum_{v_m \in V} (\textit{lowest weight of edge leaving } v_m) = 4 + 7 + 4 + 2 + 4 = \mathbf{21}$$

6.2 The Traveling SalesPerson Problem

The Best-First Search with Branch and Bound

Node_0



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**

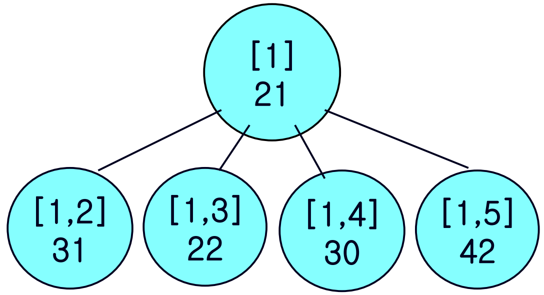
➔ Lower Bound on Minimum Cost Tour

Node_0 : $4 + 7 + 4 + 2 + 4 = 21$

➔ Queue: { Node_0 }

6.2 The Traveling SalesPerson Problem

- **The Best-First Search with Branch and Bound**



Node$_1$ Node$_2$ Node$_3$ Node$_4$

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$_1$: $\mathbf{\mathit{14}} + (7 + 4 + 2 + 4) = 31$

Node$_2$: $\mathbf{\mathit{4}} + (7 + 5 + 2 + 4) = 22$

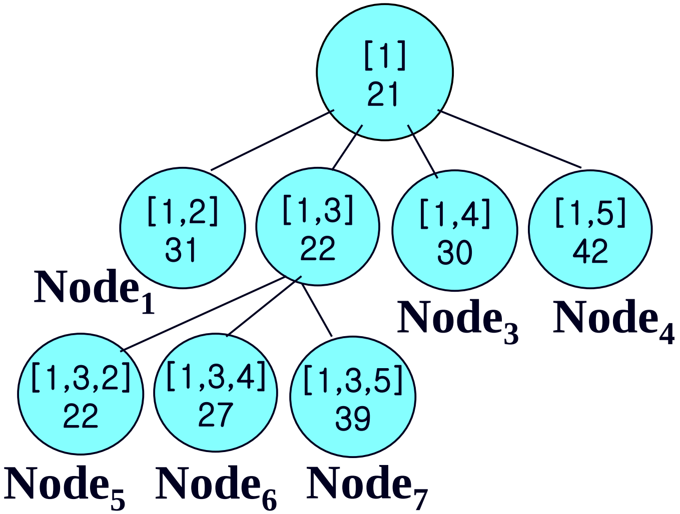
Node$_3$: $\mathbf{\mathit{10}} + (7 + 4 + 2 + 7) = 30$

Node$_4$: $\mathbf{\mathit{20}} + (7 + 4 + 7 + 4) = 42$

➔ Queue: { Node$_2$, Node$_3$, Node$_1$, Node$_4$}

6.2 The Traveling SalesPerson Problem

❑ The Best-First Search with Branch and Bound



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$_5$: $\mathbf{4+5 + (7 + 2 + 4) = 22}$

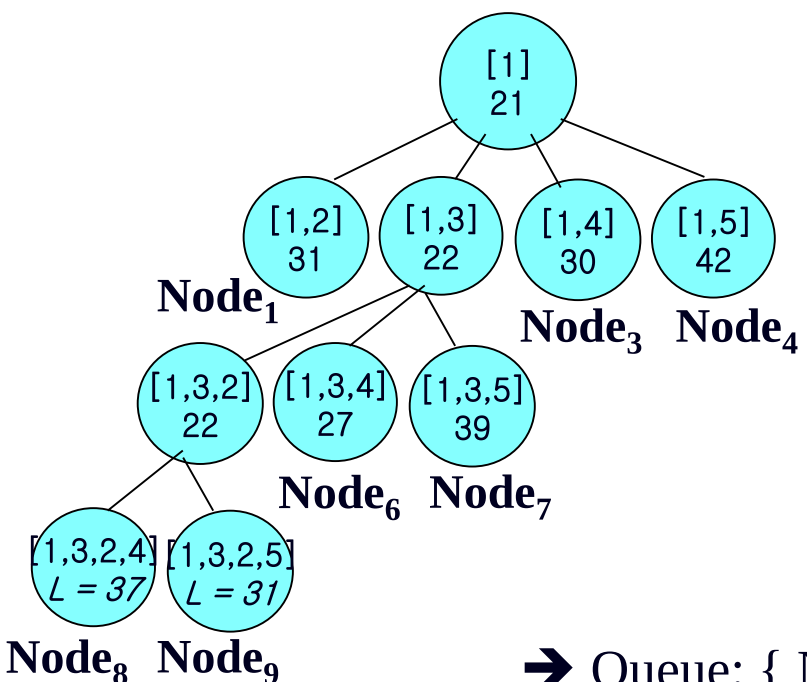
Node$_6$: $\mathbf{4+7 + (7 + 2 + 7) = 27}$

Node$_7$: $\mathbf{4+16 + (8 +7 + 4) = 39}$

➔ Queue: { Node$_5$, Node$_6$, Node$_3$, Node$_1$, Node$_7$, Node$_4$}

6.2 The Traveling SalesPerson Problem

❑ The Best-First Search with Branch and Bound



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**

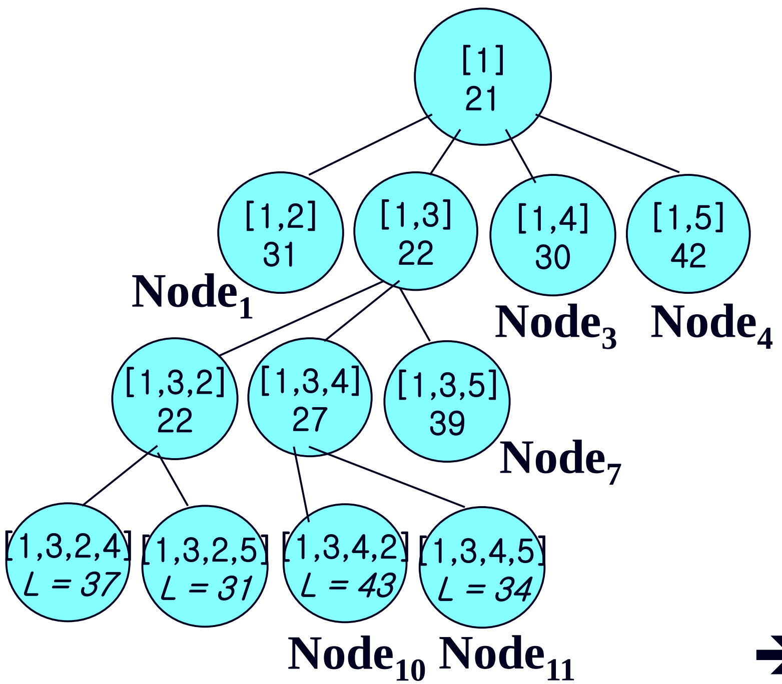
Node$_9$: ***4+5+7*** **+ (4+11) = 31**

➔ Queue: { Node$_6$, Node$_3$, Node$_1$, Node$_7$, Node$_4$}

➔ Current best solution = 31

6.2 The Traveling SalesPerson Problem

The Best-First Search with Branch and Bound



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**

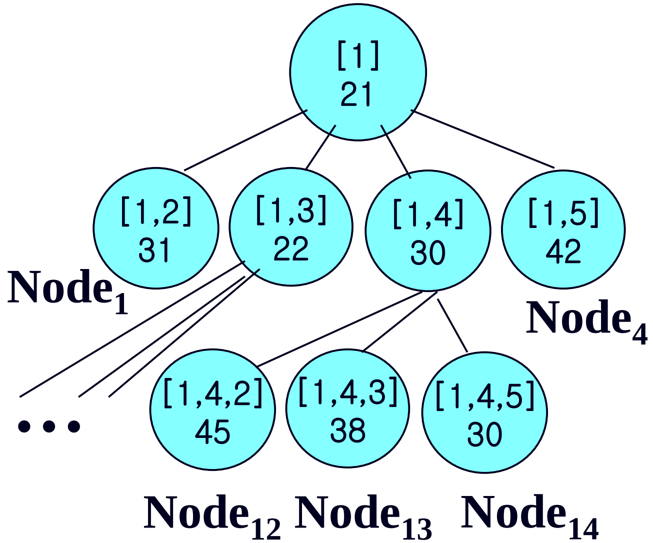
Node$_{11}$: $4+7+2$ + **(7 + 14) = 34**

➔ Queue: { Node$_3$, Node$_1$, Node$_7$, Node$_4$}

➔ Current best solution = 31

6.2 The Traveling SalesPerson Problem

- **The Best-First Search with Branch and Bound**



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**

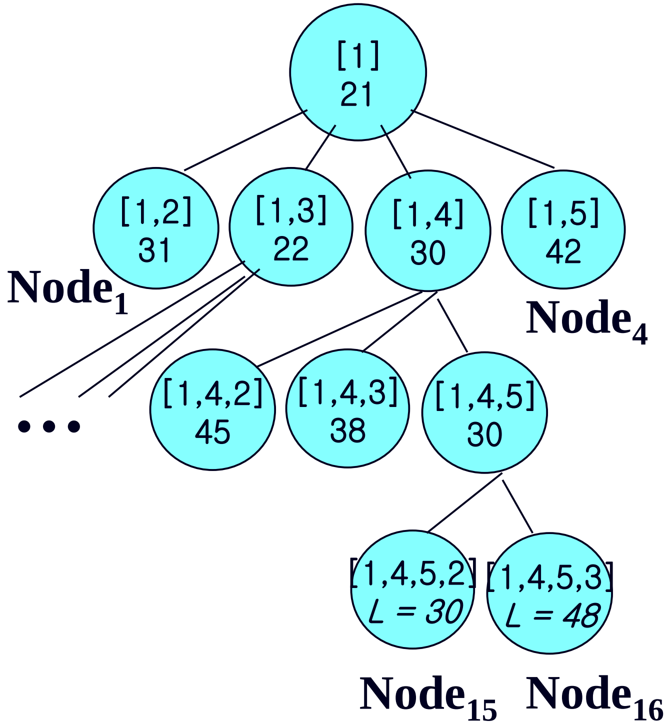
Node$_{14}$: ***10+2*** **+ (7 +4 + 7) = 30**

➔ Queue: { Node$_{14}$, Node$_1$,Node$_7$, Node$_4$}

➔ Current best solution = 31

6.2 The Traveling SalesPerson Problem

- **The Best-First Search with Branch and Bound**



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

6.2 The Traveling SalesPerson Problem

The Best-First Search with Branch and Bound

```
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 ;
   }
}
```

```
public class node
{
      int level ;
      ordered_set path;
      number bound ;
}
```

6.2 The Traveling SalesPerson Problem

The Best-First Search with Branch and Bound

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) ;
    }
```