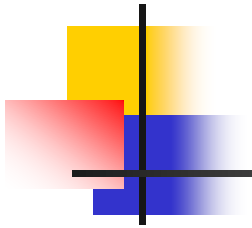
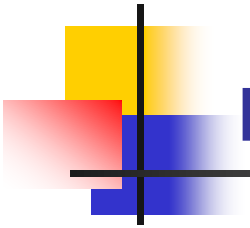


[Chapter 6]

Branch-and-Bound



Branch-and-Bound Algorithm

- Branch-and-Bound is an improvement on the backtracking
  - State space tree is used like backtracking.
  - No limit to any particular way of traversing the tree.
  - Used only for optimization problems.
- What is a bound?
  - Compute a bound at a node to determine whether the node is promising.
  - A bound is the value of the solution that could be obtained by expanding beyond the node.
  - If the bound is no better than the value of the best solution found so far, the node is non-promising.

The 0/1 Knapsack Problem Revisited (1/2)

- Algorithm sketch
  - Profit (weight) is the total profit (weight) of the items that have been included up to a node.
  - Bound is the value that could be obtained beyond this node at best.

$$totweight = weight + \sum_{j=i+1}^{k-1} w_j$$

$$bound = \left(profit + \sum_{j=i+1}^{k-1} p_j \right) + (W - totweight) \times \frac{p_k}{w_k}$$

  - Maxprofit is the value of the profit in the best solution so far.
  - So, if bound $\leq$ maxprofit, the node is non-promising.
  - We can't stop searching the state space tree as long as there are more nodes to compute since we want an optimal solution.

The 0/1 Knapsack Problem Revisited (2/2)

- Summary
  - Visit a node in DFS or BFS fashion.
  - Compute profit, weight, and bound.
  - Keep on searching as long as weight < W and bound > maxprofit.
  - Otherwise, backtrack.
- (e.g.) $n = 4$ and $W = 16$.

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

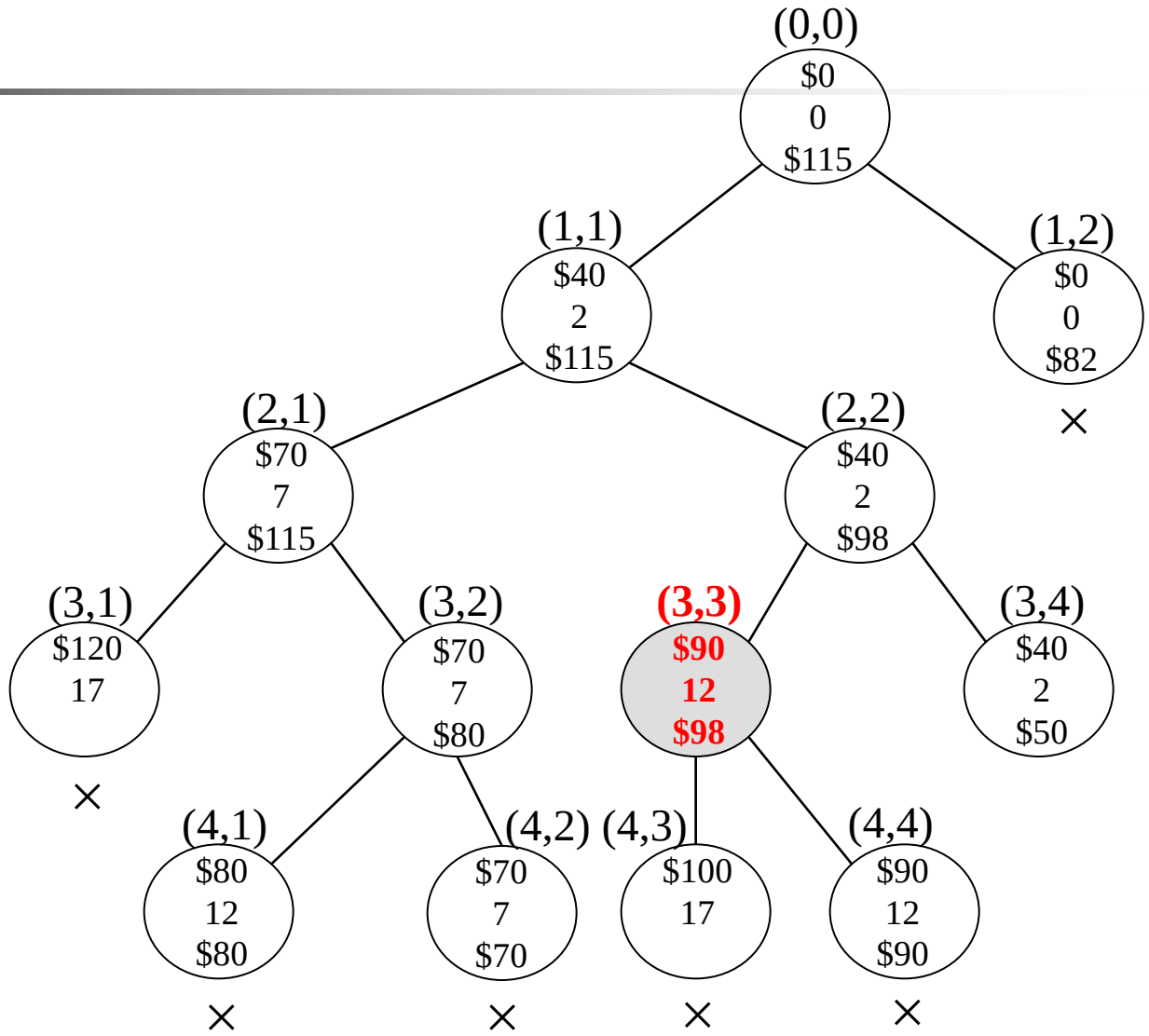
State space tree in Depth-First-Search

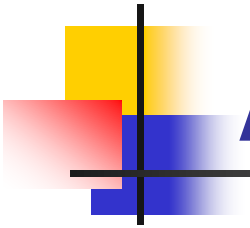
item 1 [$40, 2]

item 2 [$30, 5]

item 3 [$50, 10]

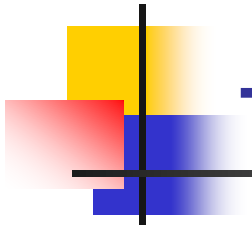
item 4 [$10, 5]





Analysis: 0/1 Knapsack Problem

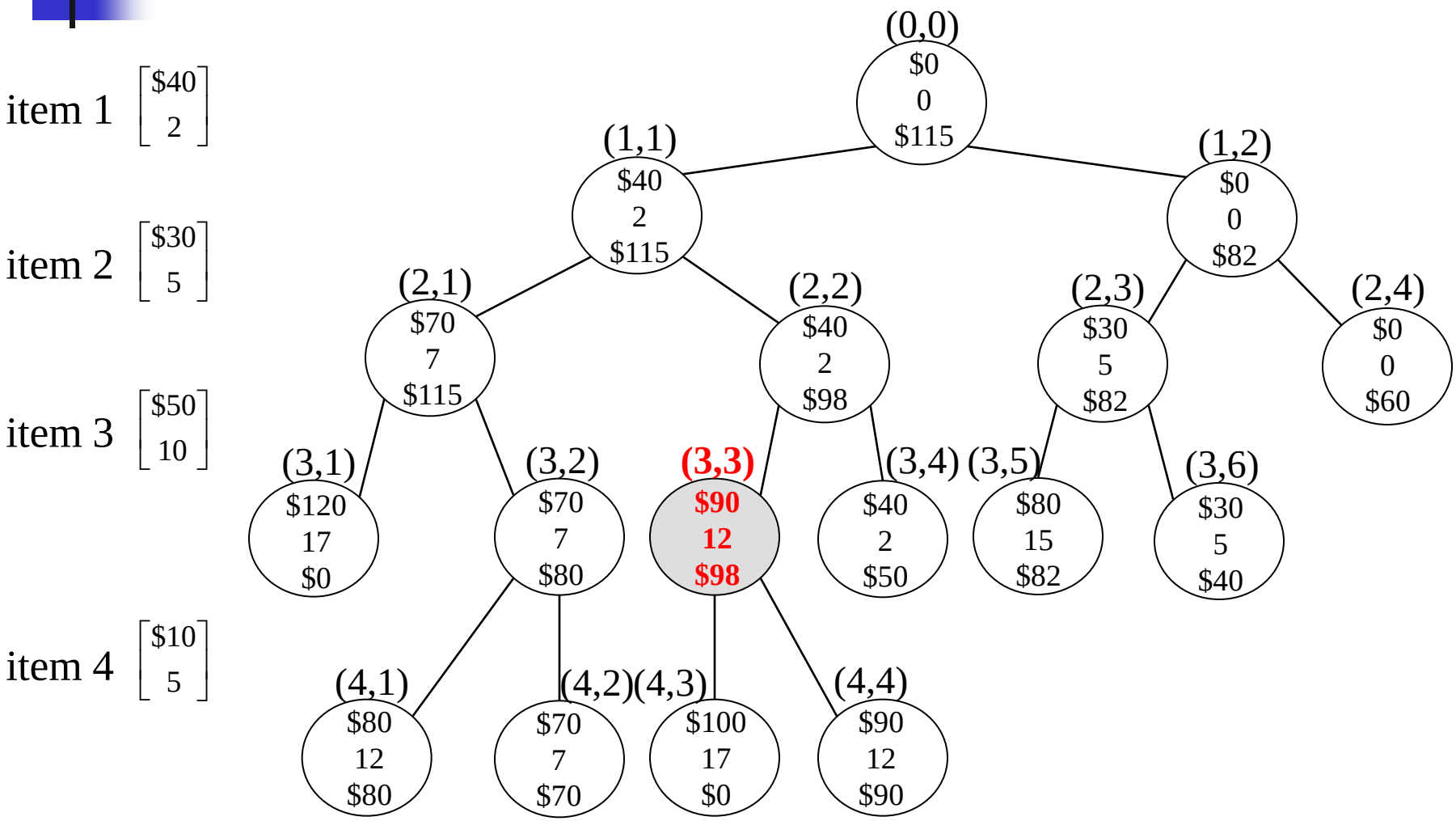
- The number of nodes to visit in this algorithm is 13 nodes. Is this better than dynamic programming version?
  - In 1978, Horowitz and Sahni showed that the branch-and-bound algorithm is better than dynamic programming for the 0/1 Knapsack problem using Monte Carlo method.
  - According to their Monte Carlo simulation, the time complexity is $O(2^{n/2})$, which is than $O(2^n)$.
- Any improvement?
  - What happens if we change the way of tree traversal from DFS to BFS?
  - Let's just do it and think about Best-First-Search with branch-and-bound pruning.

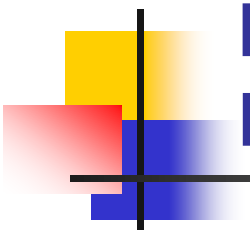


The Breadth-First-Search

```
void breadth_first_branch_and_bound(state_space_tree T,
                                    number& best) {
   queue_of_node Q; node u, v;
   initialize(Q);                   // Initialize Q to be empty.
   v = root of T;                   // Visit root.
   enqueue(Q,v);
   best = value(v);
   while(!empty(Q)) {
     dequeue(Q,v);
     for(each child u of v) {       // Visit each child.
       if(value(u) is better than best)
         best = value(u);
       if(bound(u) is better than best)
         enqueue(Q,u);
     } } }
```

Argggggh! The number of nodes to visit is increased to 17!! Worse than DFS. Hmm.





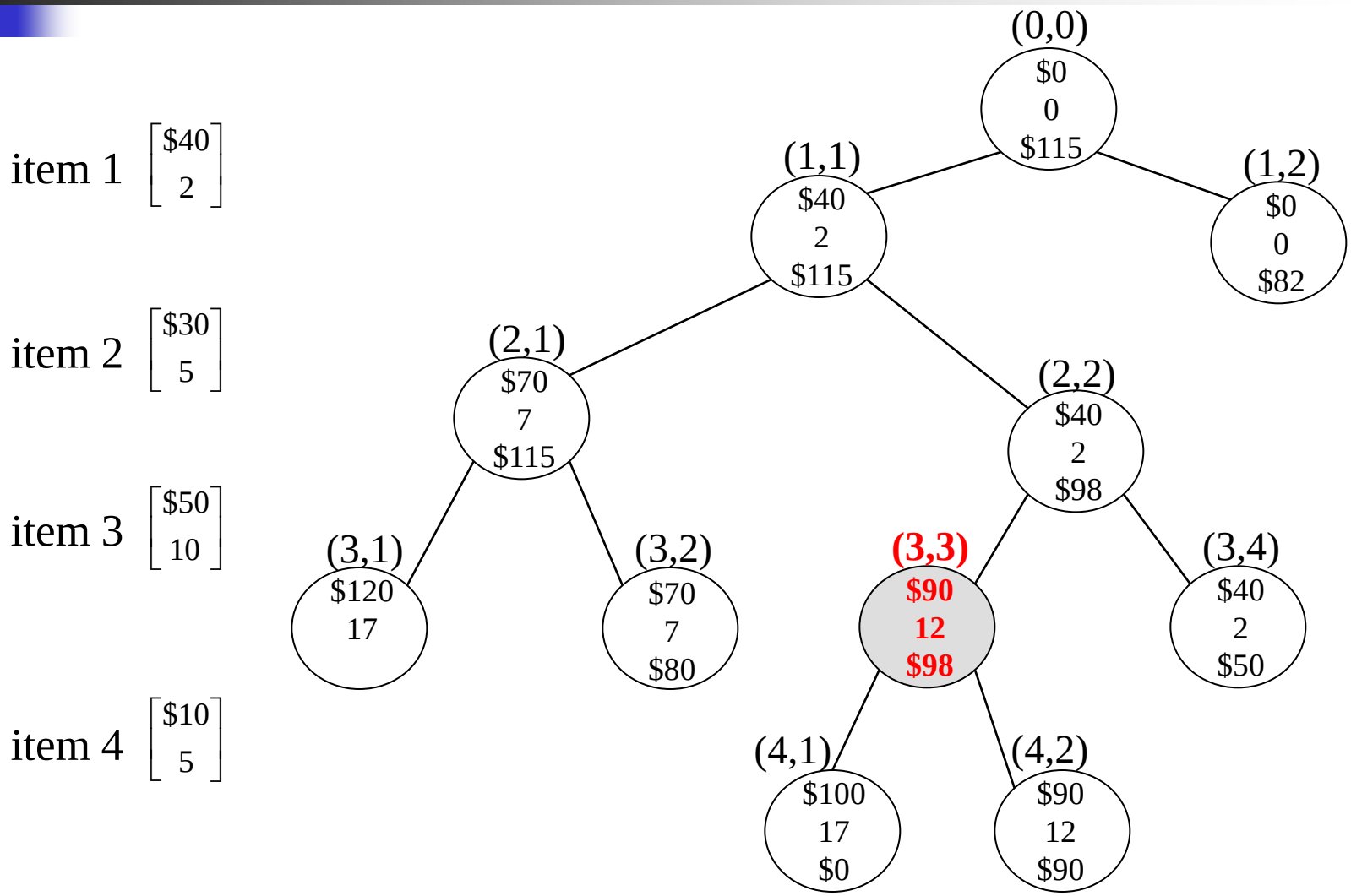
Best-First-Search with Branch-and-Bound Pruning.

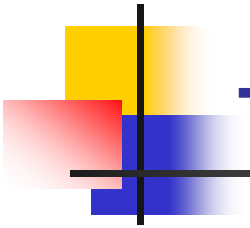
- We want to reach the solution node as fast as possible. What strategy are you going to try?
  - We CAN compute the bounds for all children nodes.
  - The more bound, the better.
    - DFS treats them all equal.
    - BFS treats them all equal.
    - Alas, they are blind to the value of bound!!!
  - We will see what happens if we try the node with best-bound-first strategy as a node to expand next.

The Best-First-Search

```
void best_first_branch_and_bound(state_space_tree T,
                                 number best) {
  priority_queue_of_node PQ;
  node u,v;
  initialize(PQ);                  // Initialize PQ to be empty.
  v = root of T;
  best = value(v);
  insert(PQ,v);
  while(!empty(PQ)) {
    remove(PQ,v);                  // Remove bide with best bound.
    if(bound(v) is better than best)// Check if node is promising.
      for(each child u of v) {
        if(value(u) is better than best)
          best = value(u);
        if(bound(u) is better than best)
          insert(PQ,u);
      }
  }
}
```

Wow! The number of nodes to visit is reduced to 11!! Truly the best.

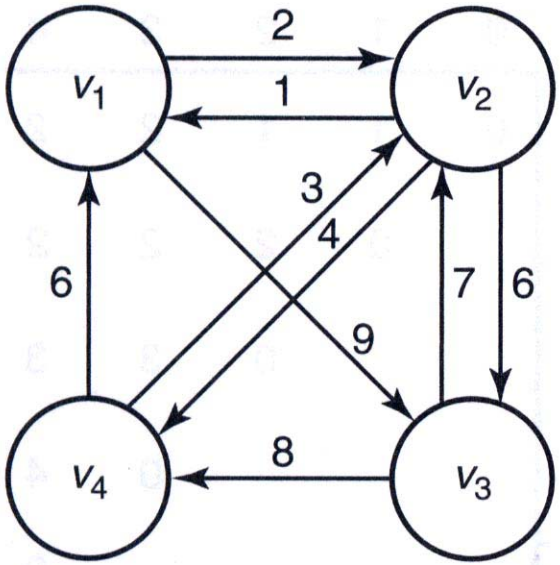




The Traveling Salesperson Problem

- Problem
  - Determine an optimal tour in a weighted, directed graph.
  - The weights are non-negative numbers.
- Inputs
  - A weighted, directed graph, and n, the number of vertices in the graph. The graph is represented by a 2-dim array W, which has both its rows and columns indexed from 1 to n, where $W[i][j]$ is the weight on the edge from the i-th vertex to the j-th vertex.
- Outputs
  - A variable, $minlength$, whose value is the length of an optimal tour, and a matrix P from which an optimal tour can be constructed.

Figure 3.16 The optimal tour is [v_1, v_3, v_4, v_2, v_1].



	1	2	3	4
1	0	2	9	∞
2	1	0	6	4
3	∞	7	0	8
4	6	3	∞	0

Dynamic Programming for TSP

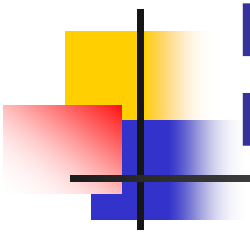
- Brute-force algorithm
  - $(n$ -1)!
- The length of the optimal path
  - $D[v_1][V-\{v_1\}] = min_{2 \le j \le n}(W[1][j] + D[v_j][V-\{v_1, v_j\}])$
  - In general, $i \neq 1$ and v_i is not in A,

$$D[v_i][A] = min_{v_j \in A}(W[i][j] + D[v_j][A-\{v_j\}]) \text{ if } A \neq 0$$

$$D[v_i][\phi] = W[i][1]$$

DP for TSP

```
void travel(int n,const number W[][],index P[][],
            number& minlength) {
  index i,j,k;
  number D[1..n][subset of V-{v1}];
  for(i=2; i<=n; i++) D[i][emptyset] := W[i][1];
  for(k=1; k<=n-2; k++)
    for all subsets A in(V-{v1}) containing k vertices
      for(i != 1 and vi is not in A){
        D[i][A] = minimum_{vj∈A}(W[i][j] + D[vj][A-{vj}]);
        P[i][A] = value of j that gave the minimum;
      }
  D[1][V-{v1}] = minimum_{2≤j≤n}(W[1][j] + D[vj][A-{v1}]);
  P[1][V-{v1}] = value of j that gave the minimum;
  minilength = D[1][V-{v1}];
}
```



Practice to solve TSP using Dynamic Programming

- Step 1
- Step 2
- Step 3

Theorem: $\sum_{k=1}^{n} k\begin{bmatrix} n \\ k \end{bmatrix} = n2^{n-1}$

$$\begin{bmatrix} n \\ k \end{bmatrix} = \frac{n!}{k!(n-k)!} = \frac{n}{k}\frac{(n-1)!}{(k-1)!(n-k)!}$$

$$= \frac{n}{k}\frac{(n-1)!}{(k-1)!(n-1-(k-1))!}$$

$$= \frac{n}{k}\begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

$$k\begin{bmatrix} n \\ k \end{bmatrix} = k\frac{n}{k}\begin{bmatrix} n-1 \\ k-1 \end{bmatrix} = n\begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

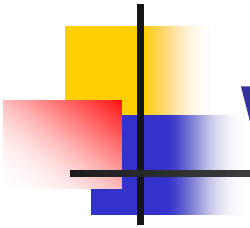
$$\sum_{k=1}^{n} k\begin{bmatrix} n \\ k \end{bmatrix} = \sum_{k=1}^{n} n\begin{bmatrix} n-1 \\ k-1 \end{bmatrix} = n\sum_{k=0}^{n-1}\begin{bmatrix} n-1 \\ k \end{bmatrix} 1^k 1^{n-1-k} = n(1+1)^{n-1} = n2^{n-1}$$

Time complexity analysis for TSP in DP

- $T(n) = \sum_{k=1}^{n-2} (n-k-1)k \begin{bmatrix} n-1 \\ k \end{bmatrix}$

Ouch, exponential again!!!

$= (n-1)(n-2)\, 2^{n-3} \in \Theta(n^2 2^n)$

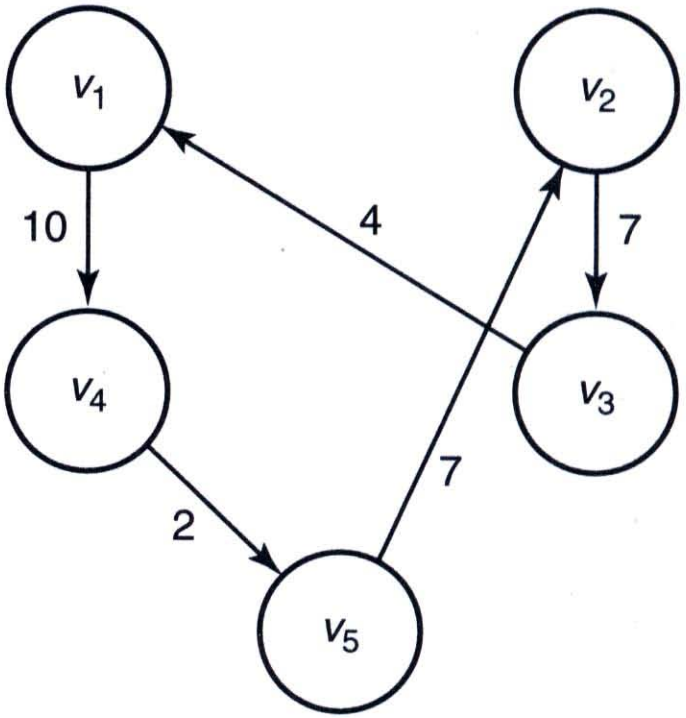


What does $\Theta(n2^n)$ mean?

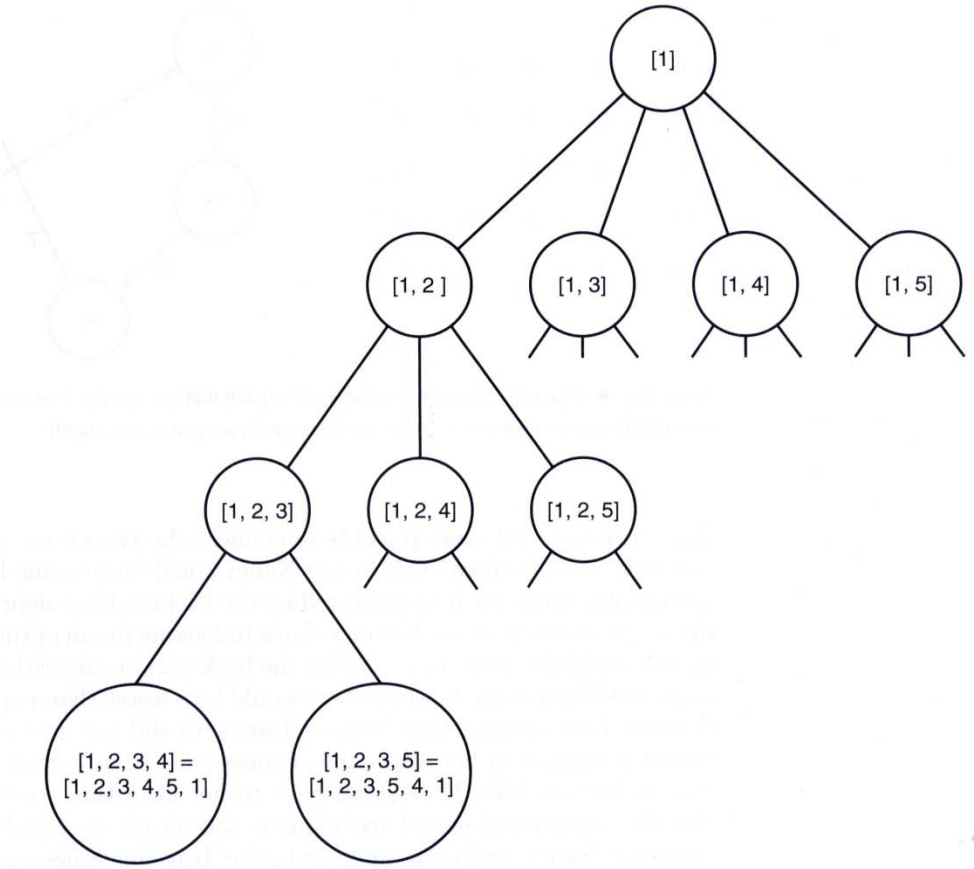
- Suppose n = 20 and unit computing time is $1\mu sec$
  - The brute-force algorithm requires you to compute (20-1)! = 19! μsec = 3857 years.
  - In DP, $T(n) = (n\text{-}1)(n\text{-}2)\,2^{n-3}$
  - Therefore, $T(20) = (20 - 1)(20 - 2)2^{20-3}\mu sec = 45$ sec
  - Fair enough.
- What happens if we apply the branch-and-bound algorithm to solve TSP?

Example: Adjacency matrix representation of a graph that has an edge from every vertex to every other vertex(left), and the nodes in the graph and the edges in an optimal tour (right).

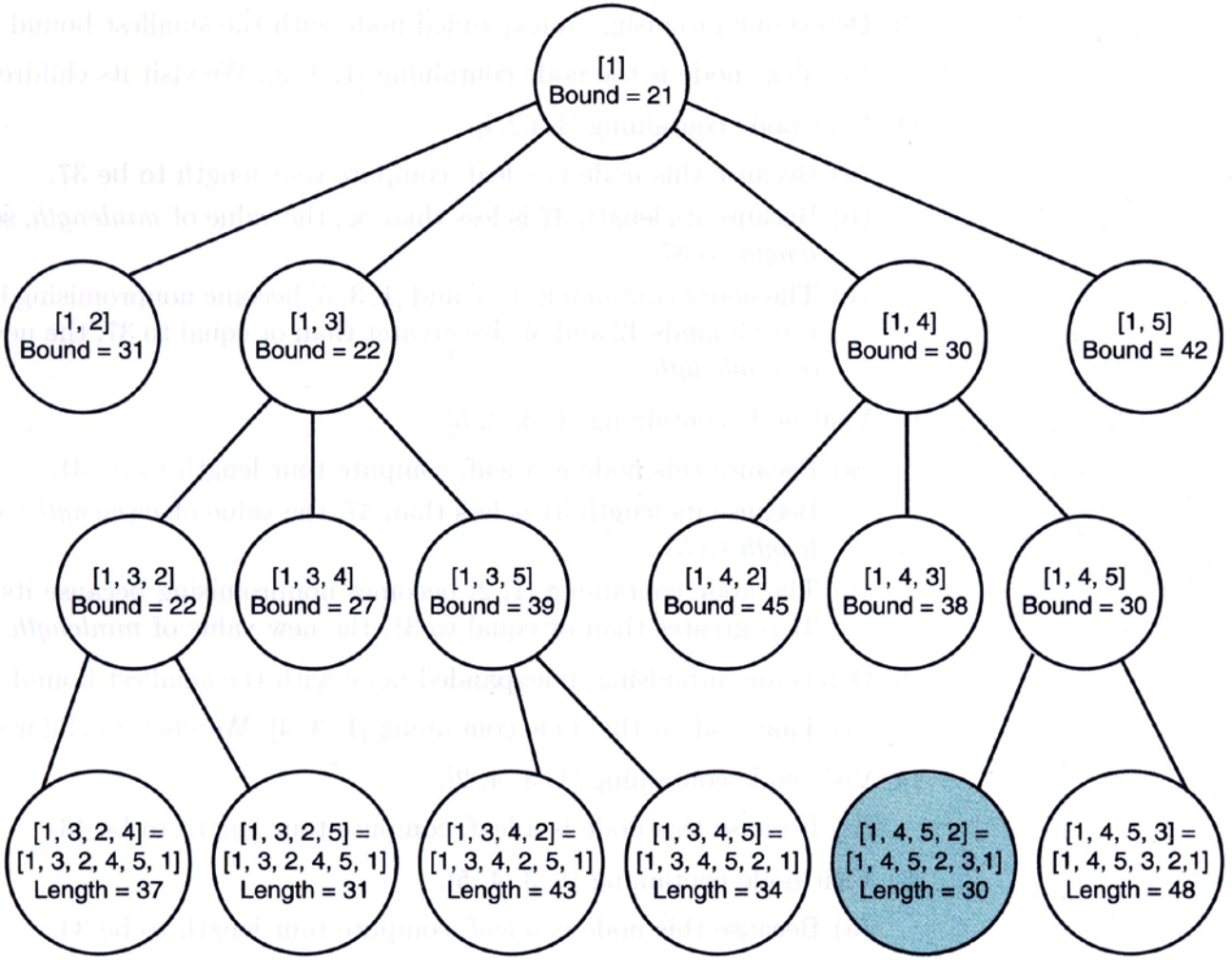
$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



A state space tree for an instance of the Traveling Salesperson problem in which there are five vertices. The indices of the vertices in the partial tour are stored at each node.



The pruned state space tree produced using best-first search with branch-and-bound pruning in Example 6.3. At each node that is not a leaf in the state space tree, the partial tour is at top and the bound on the length of any tour that could be obtained by expanding beyond the node shaded in color is the one at which an optimal tour is found.



Exercise #1. Use Algorithm 6.1 (The Breadth-First Search with Branch-and-Bound Pruning algorithm for the 0/1 Knapsack problem) to maximize the profit for the following problem instance. Show the action step by step.

W=13

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$20	2	10
2	\$30	5	6
3	\$35	7	5
4	\$12	3	4
5	\$3	1	3

Exercise #2. Use Algorithm 6.3 (The Best-first Search with Branch-and-Bound Pruning Algorithm for the Traveling Salesperson problem) to find an optimal tour and the length of the optimal tour for the graph below.