Branch and Bound Searching Strategies

Feasible Solution vs. Optimal Solution

- DFS, BFS, hill climbing and best-first search can be used to solve some searching problem for searching a feasible solution.
- However, they cannot be used to solve the optimization problems for searching an (the) optimal solution.

The branch-and-bound strategy

- This strategy can be used to solve optimization problems without an exhaustive search in the average case.

Branch-and-bound strategy

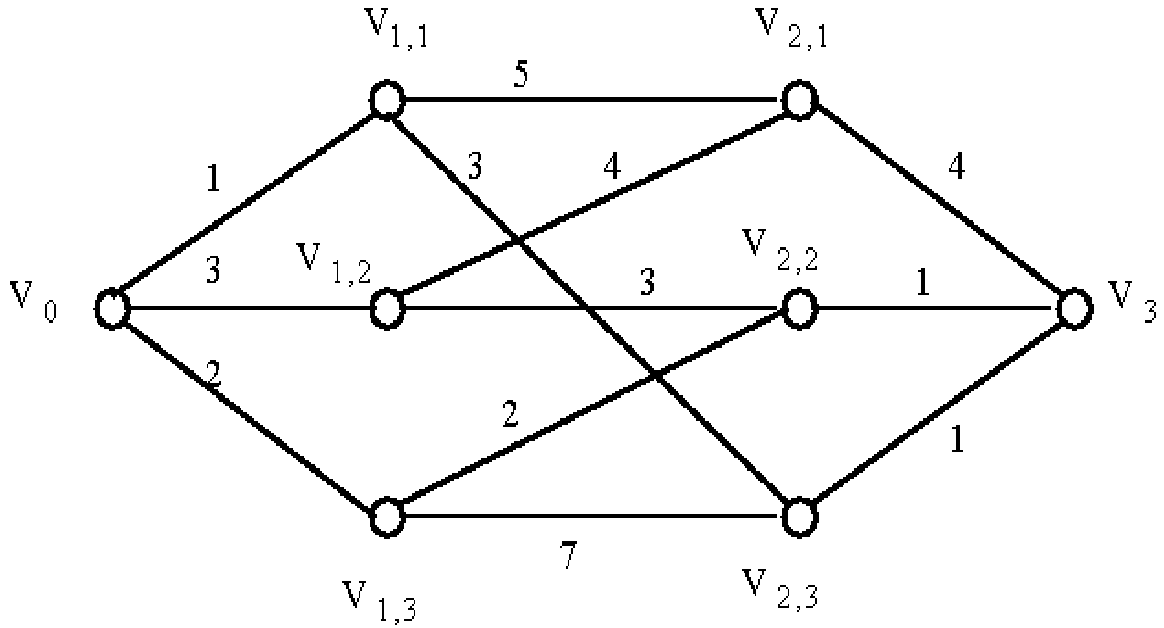
- 2 mechanisms:
  - A mechanism to generate branches when searching the solution space
  - A mechanism to generate a bound so that many braches can be terminated

Branch-and-bound strategy

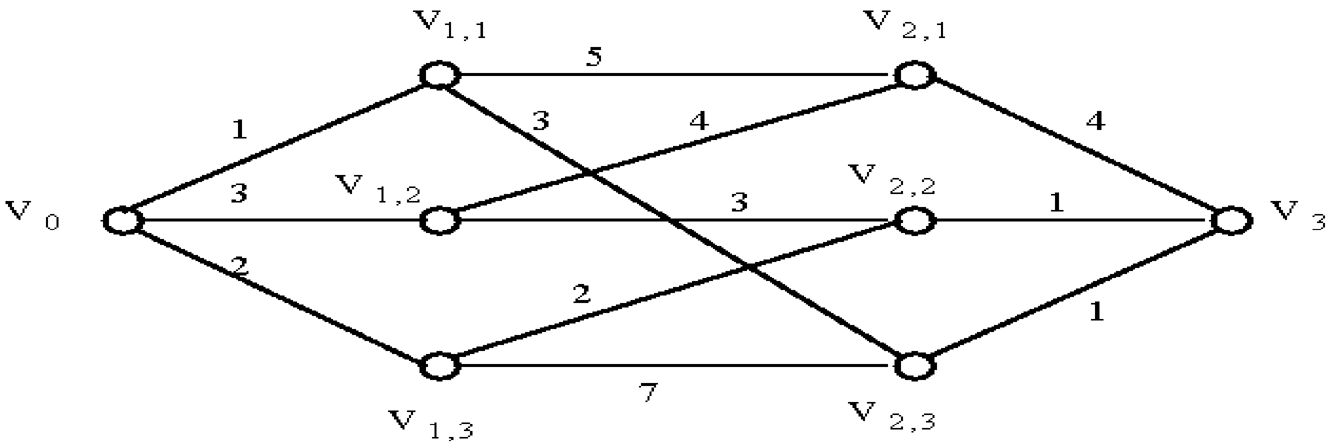
- It is efficient in the average case because many branches can be terminated very early.
- Although it is usually very efficient, a very large tree may be generated in the worst case.
- Many NP-hard problem can be solved by B&B efficiently in the average case; however, the worst case time complexity is still exponential.

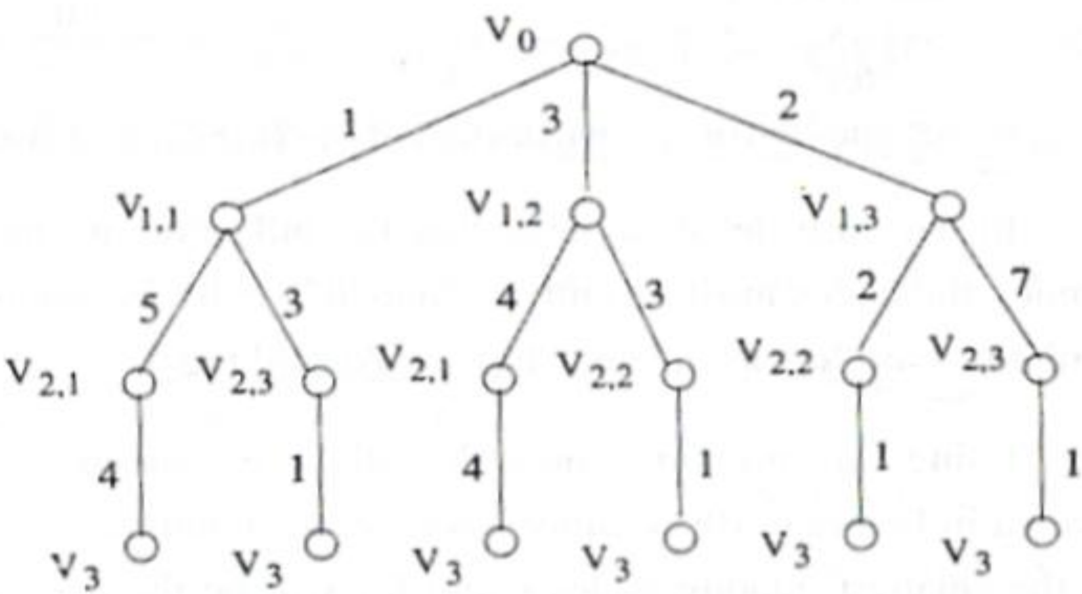
A Multi-Stage Graph Searching Problem.

Find the shortest path from V_0 to V_3

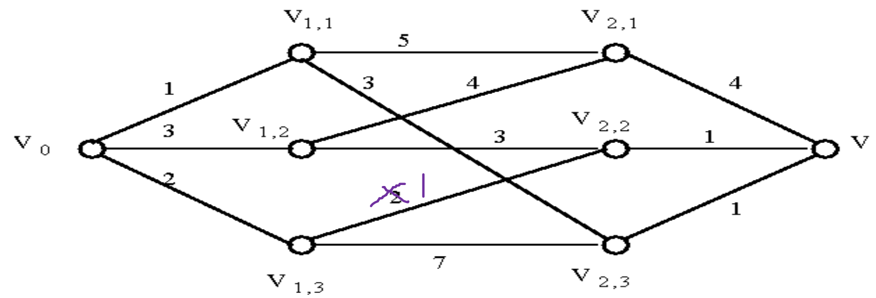


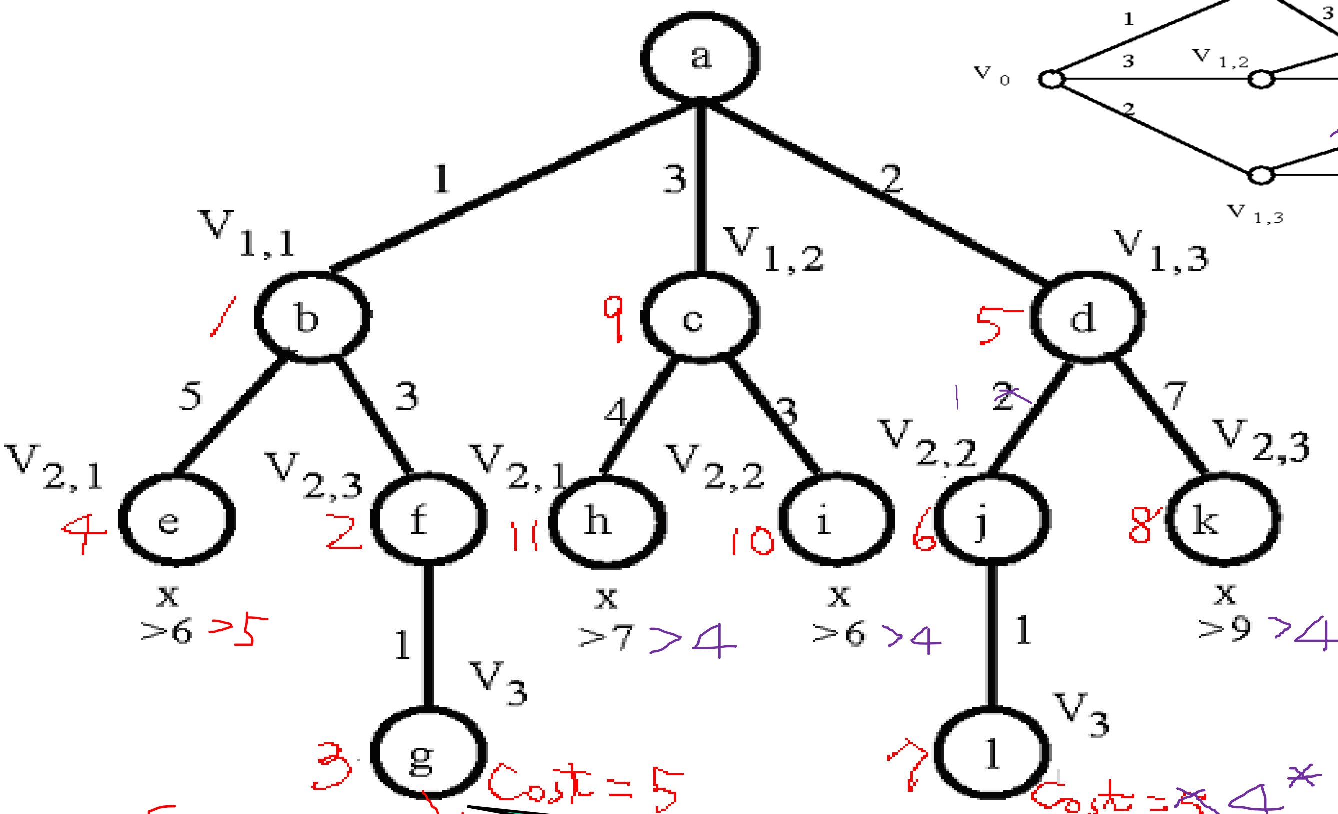
E.G.:A Multi-Stage Graph Searching Problem





Solved by branch-and-bound (hill-climbing with bounds)



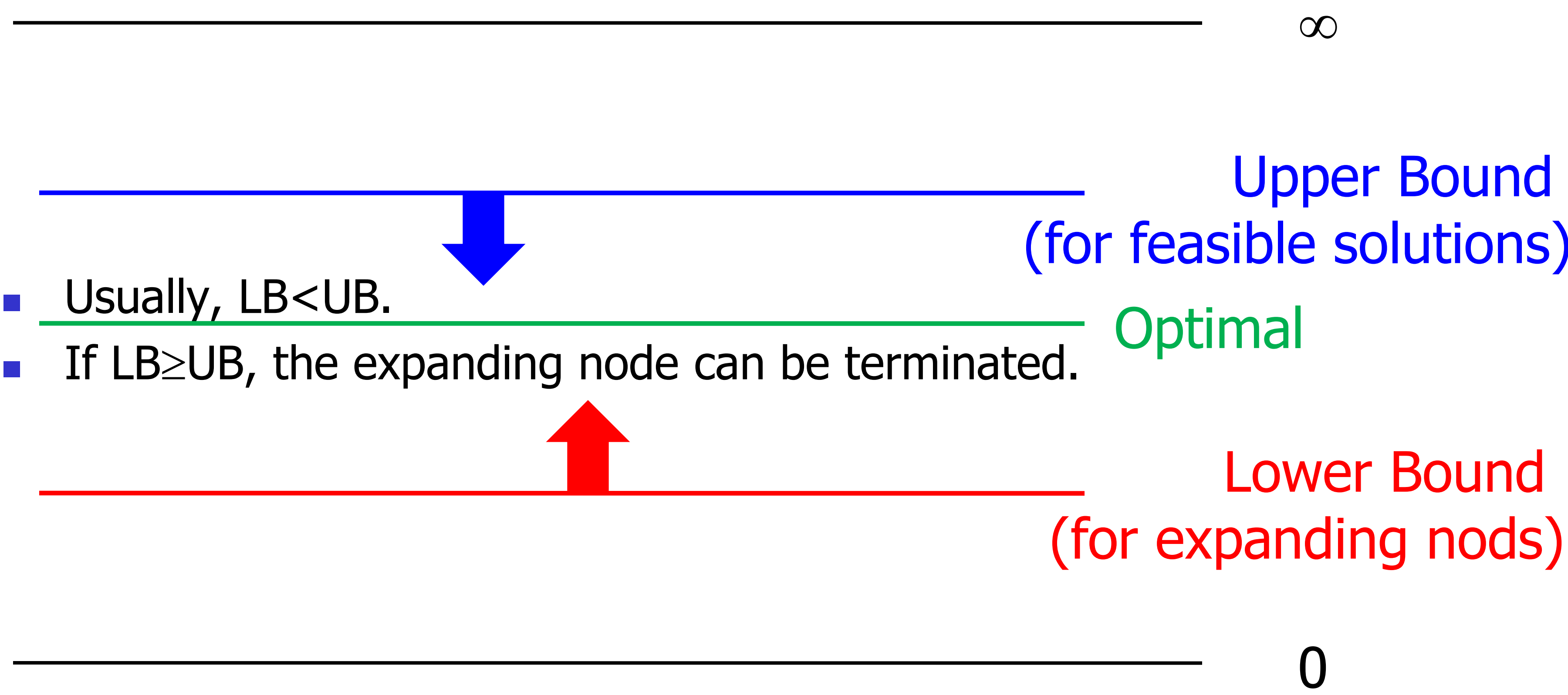


#: checking & expanding order

Feasible solution (upper bound)

A feasible solution is found whose cost is equal to 5.
An upper bound of the optimal solution is first found here.

For Minimization Problems



- Usually, LB<UB.
- If LB≥UB, the expanding node can be terminated.

For Maximization Problems

∞

Upper Bound
(for expanding nods)

- Usually, LB<UB.

Optimal

- If LB≥UB, the expanding node can be terminated.

Lower Bound
(for feasible solutions)

0

The traveling salesperson optimization problem

- Given a graph, the TSP Optimization problem is to find a tour, starting from any vertex, visiting every other vertex and returning to the starting vertex, with minimal cost.
- It is NP-hard.
- We try to avoid n! exhaustive search by the branch-and-bound technique on the average case. (Recall that $O(n!)>O(2^n)$.)

The traveling salesperson optimization problem

- E.g. A Cost Matrix for a Traveling Salesperson Problem.

i \ j	1	2	3	4	5	6	7
1	∞	3	93	13	33	9	57
2	4	∞	77	42	21	16	34
3	45	17	∞	36	16	28	25
4	39	90	80	∞	56	7	91
5	28	46	88	33	∞	25	57
6	3	88	18	46	92	∞	7
7	44	26	33	27	84	39	∞

The basic idea

- There is a way to split the solution space (branch)
- There is a way to predict a lower bound for a class of solutions. There is also a way to find a upper bound of an optimal solution. If the lower bound of a solution exceeds the upper bound, this solution cannot be optimal and thus we should terminate the branching associated with this solution.

Splitting

- We split a solution into two groups:
  - One group including a particular arc
  - The other excluding the arc
- Each splitting incurs a lower bound and we shall traverse the searching tree with the “lower” lower bound.

The traveling salesperson optimization problem

- The Cost Matrix for a Traveling Salesperson Problem.

Step 1 to reduce: Search each row for the smallest value

from i \ to j	j / i	1	2	3	4	5	6	7
	1	∞	3	93	13	33	9	57
	2	4	∞	77	42	21	16	34
	3	45	17	∞	36	16	28	25
	4	39	90	80	∞	56	7	91
	5	28	46	88	33	∞	25	57
	6	3	88	18	46	92	∞	7
	7	44	26	33	27	84	39	∞

Step 2 to reduce: Search each column for the smallest value

The traveling salesperson optimization problem

- Reduced cost matrix:

i \ j	1	2	3	4	5	6	7	
1	∞	0	90	10	30	6	54	(-3)
2	0	∞	73	38	17	12	30	(-4)
3	29	1	∞	20	0	12	9	(-16)
4	32	83	73	∞	49	0	84	(-7)
5	3	21	63	8	∞	0	32	(-25)
6	0	85	15	43	89	∞	4	(-3)
7	18	0	7	1	58	13	∞	(-26)

reduced:84

A Reduced Cost Matrix.

The traveling salesperson optimization problem

j / i	1	2	3	4	5	6	7
1	∞	0	83	9	30	6	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	32	83	66	∞	49	0	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞
			(-7)	(-1)			(-4)

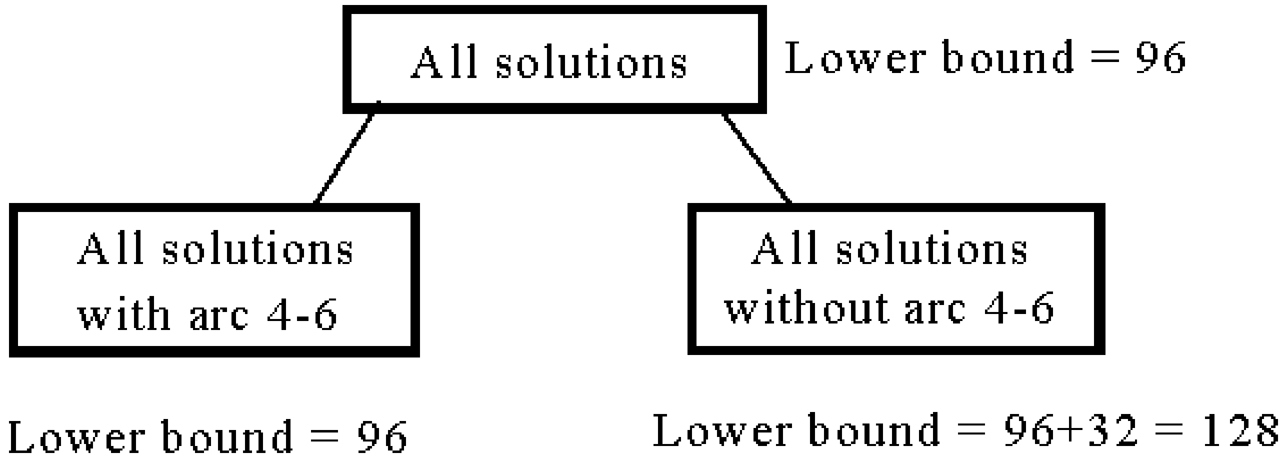
Table 6-5 Another Reduced Cost Matrix.

Lower bound

- The total cost of 84+12=96 is subtracted. Thus, we know the lower bound of feasible solutions to this TSP problem is 96.

The traveling salesperson optimization problem

- Total cost reduced: 84+7+1+4 = 96 (lower bound) decision tree:



The Highest Level of a Decision Tree.

- If we use arc 3-5 to split, the difference on the lower bounds is 17+1 = 18.

Heuristic to select an arc to split the solution space

- If an arc of cost 0 (x) is selected, then the lower bound is added by 0 (x) when the arc is included.
- If an arc <i,j> is not included, then the cost of the second smallest value (y) in row i and the second smallest value (z) in column j is added to the lower bound.
- Select the arc with the largest (y+z)-x

We only have to set c4-6 to be ∞.

For the right subtree (Arc 4-6 is excluded)

i \ j	1	2	3	4	5	6	7
1	∞	0	83	9	30	6	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	32	83	66	∞	49	∞	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

For the left subtree (Arc 4-6 is included)

i \ j	1	2	3	4	5	7
1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5
5	3	21	56	7	∞	28
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

A Reduced Cost Matrix if Arc 4-6 is included.

1. 4^{th} row is deleted.
2. 6^{th} column is deleted.
3. We must set c6-4 to be ∞. (The reason will be clear later.)

For the left subtree

- The cost matrix for all solution with arc 4-6:

j / i	1	2	3	4	5	7	
1	∞	0	83	9	30	50	
2	0	∞	66	37	17	26	
3	29	1	∞	19	0	5	
5	0	18	53	4	∞	25	(-3)
6	0	85	8	∞	89	0	
7	18	0	0	0	58	∞	

A Reduced Cost Matrix for that in Table 6-6.

- Total cost reduced: 96+3 = 99 (new lower bound)

Upper bound

- We follow the best-first search scheme and can obtain a feasible solution with cost C.
- C serves as an upper bound of the optimal solution and many branches may be terminated if their lower bounds are equal to or larger than C.

⊕ : expanding order (selecting order)

: adding order

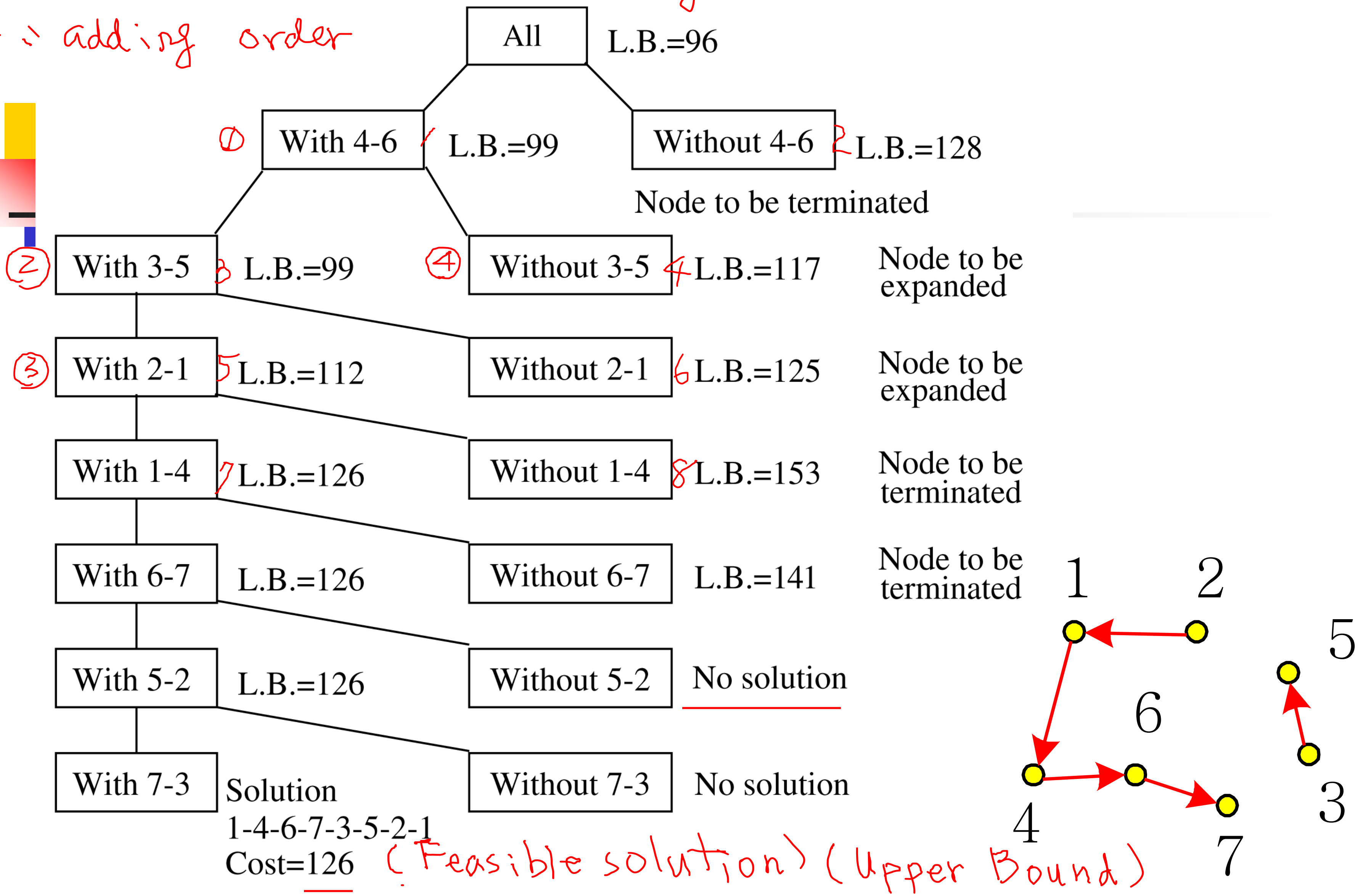


Fig 6-26 A Branch-and-Bound Solution of a Traveling Salesperson Problem.

Preventing an arc

- In general, if paths i_1-i_2-…-i_m and j_1-j_2-…-j_n have already been included and a path from i_m to j_1 is to be added, then path from j_n to i_1 must be prevented (by assigning the cost of j_n to i_1 to be ∞)
- For example, if 4-6, 2-1 are included and 1-4 is to be added, we must prevent 6-2 from being used by setting c6-2=∞. If 6-2 is used, there will be a loop which is forbidden.

The 0/1 knapsack problem

- Positive integer $P_1, P_2, \ldots, P_n$ (profit)
  $W_1, W_2, \ldots, W_n$ (weight)
  M (capacity)

$$\text{maximize} \quad \sum_{i=1}^{n} P_i X_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} W_i X_i \leq M \quad X_i = 0 \text{ or } 1, \; i = 1, \ldots, n.$$

The problem is modified:

$$\text{minimize} \quad -\sum_{i=1}^{n} P_i X_i$$

The 0/1 knapsack problem

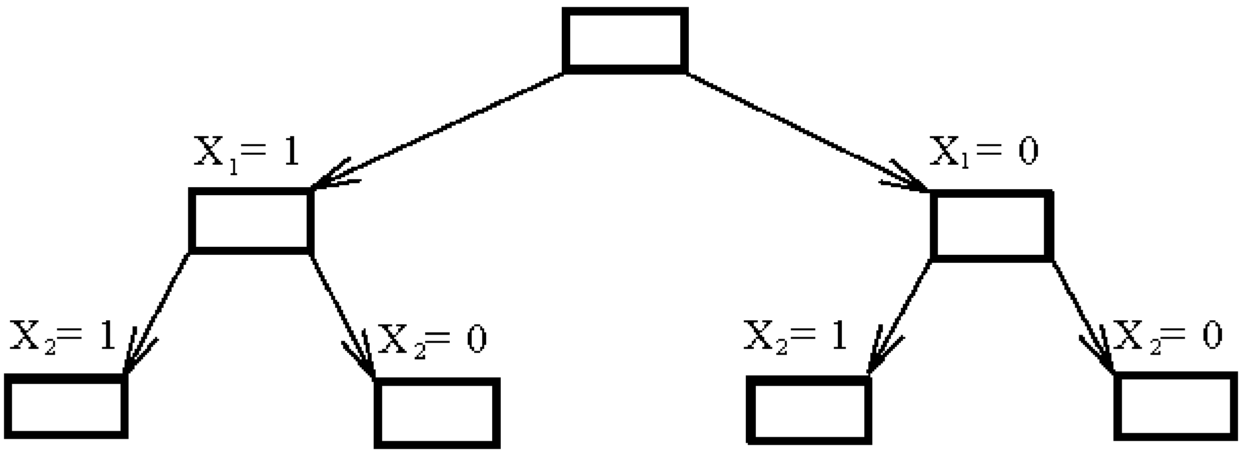


Fig. 6-27 The Branching Mechanism in the Branch-and-Bound Strategy to Solve 0/1 Knapsack Problem.

How to find the upper bound?

- Ans: by quickly finding a feasible solution in a greedy manner: starting from the smallest available i, scanning towards the largest i's until M is exceeded. The upper bound can be calculated.

The 0/1 knapsack problem

- E.g. n = 6, M = 34

i	1	2	3	4	5	6
P_i	6	10	4	5	6	4
W_i	10	19	8	10	12	8

$$(P_i/W_i \ge P_{i+1}/W_{i+1})$$

- A feasible solution: $X_1 = 1$, $X_2 = 1$, $X_3 = 0$, $X_4 = 0$, $X_5 = 0$, $X_6 = 0$
  $-(P_1+P_2) = -16$ (upper bound)
  Any solution higher than -16 can not be an optimal solution.

How to find the lower bound?

- Ans: by relaxing our restriction from $X_i = 0$ or 1 to $0 \leq X_i \leq 1$ (knapsack problem)

Let $-\sum_{i=1}^{n} P_i X_i$ be an optimal solution for 0/1 knapsack problem and $-\sum_{i=1}^{n} P_i X_i'$ be an optimal solution for fractional knapsack problem. Let $Y = -\sum_{i=1}^{n} P_i X_i$, $Y' = -\sum_{i=1}^{n} P_i X_i'$.

$\Rightarrow Y' \leq Y$

The knapsack problem

- We can use the greedy method to find an optimal solution for knapsack problem.
- For example, for the state of $X_1=1$ and $X_2=1$, we have

  $X_1 = 1, X_2 =1, X_3 = (34-6-10)/8=5/8, X_4 = 0, X_5 = 0, X_6 =0$

  $-(P_1+P_2+5/8P_3) = -18.5$ (lower bound)

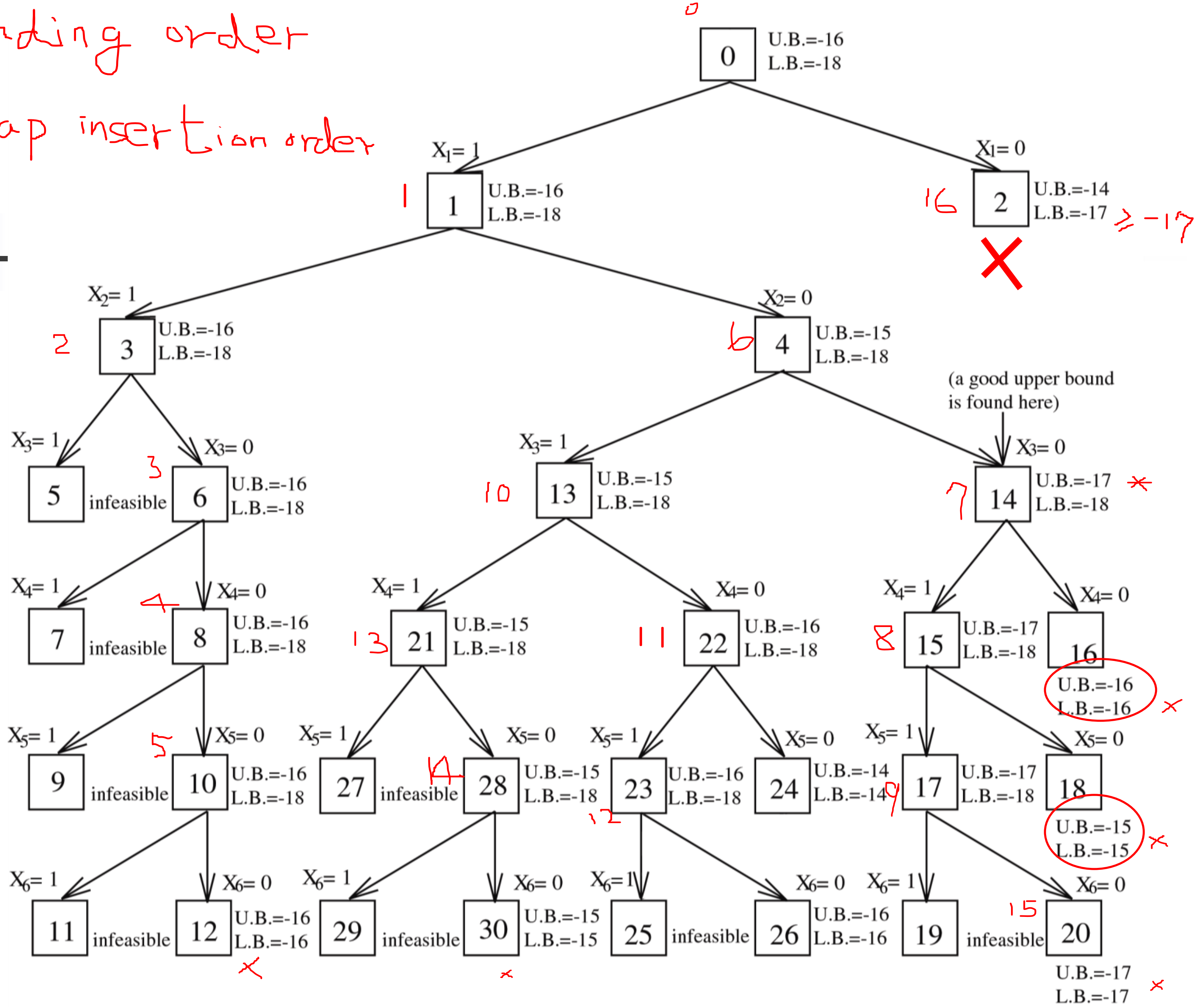
  -18 is our lower bound. (We only consider integers, since the benefits of a 0/1 knapsack problem will be integers.)

How to expand the tree?

- By the best-first search scheme
- That is, by expanding the node with the best lower bound. If two nodes have the same lower bounds, expand the node with the lower upper bound.

#: expanding order

$\boxed{\#}$: heap insertion order



0/1 Knapsack Problem Solved by Branch-and-Bound Strategy

- Node 2 is terminated because its lower bound is equal to the upper bound of node 14.
- Nodes 16, 18 and others are terminated because the local lower bound is equal to the local upper bound.
(lower bound $\leq$ optimal solution $\leq$ upper bound)

The A* algorithm

- Used to solve optimization problems.
- Using the best-first strategy.
- If a feasible solution (goal node) is selected to expand, then it is optimal and we can stop.
- Estimated cost function of a node n : f(n)

$f(n) = g(n) + h(n)$

g(n): cost from root to node n.

h(n): estimated cost from node n to a goal node.

$h^*(n)$: "real" cost from node n to a goal node.

$f^*(n)$: "real" cost of node n

$h(n) \leq h^*(n)$

Estimated further cost should never exceed the real further cost.

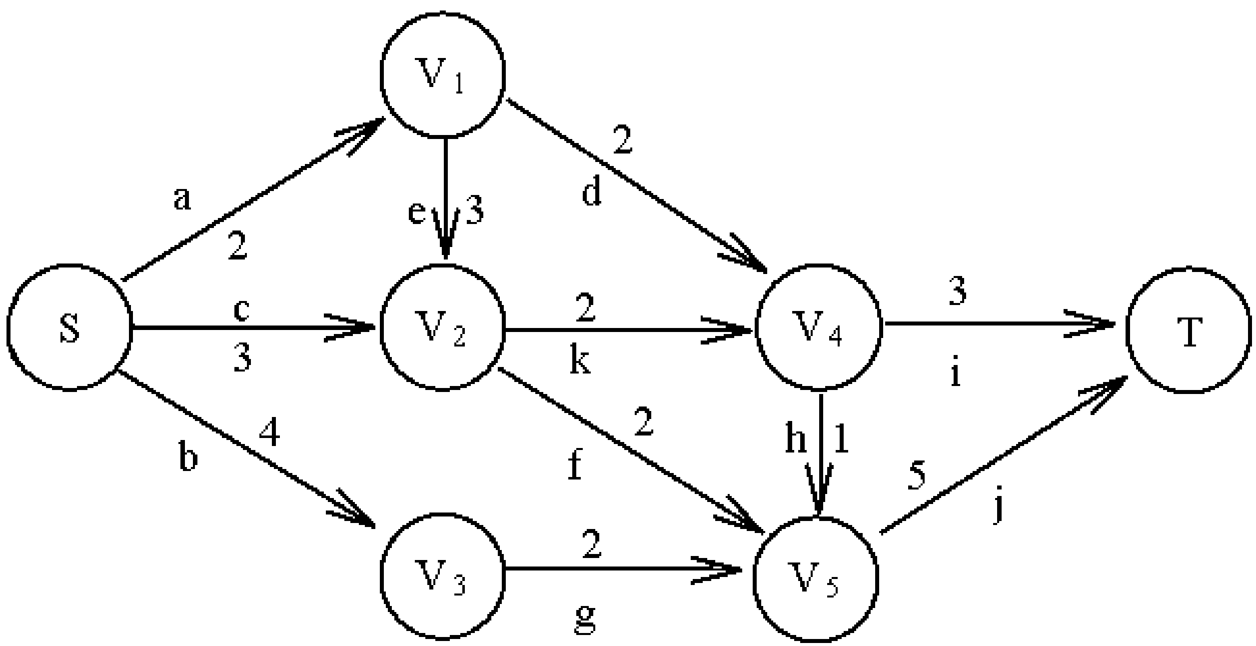
$\Rightarrow f(n) = g(n) + h(n) \leq g(n)+h^*(n) = f^*(n)$ .... (1)

Reasoning

- Let t be the selected goal node. We have $f^*(t)=f(t)+h(t)=f(t)+0=f(t)$.....(2)
- Assume that t is not the optimal node. There must exist one node, say s, that has been generated but not selected and that will lead to the optimal node.
- Since we take the best first search strategy, we have $f(t) \leq f(s)$......(3).
- We have $f^*(t)=f(t) \leq f(s) \leq f^*(s)$ by Eqs. (1), (2) and (3), which means that s is not the node leading to the optimal node. Contradiction occurs.
- Therefore, t is the optimal node.

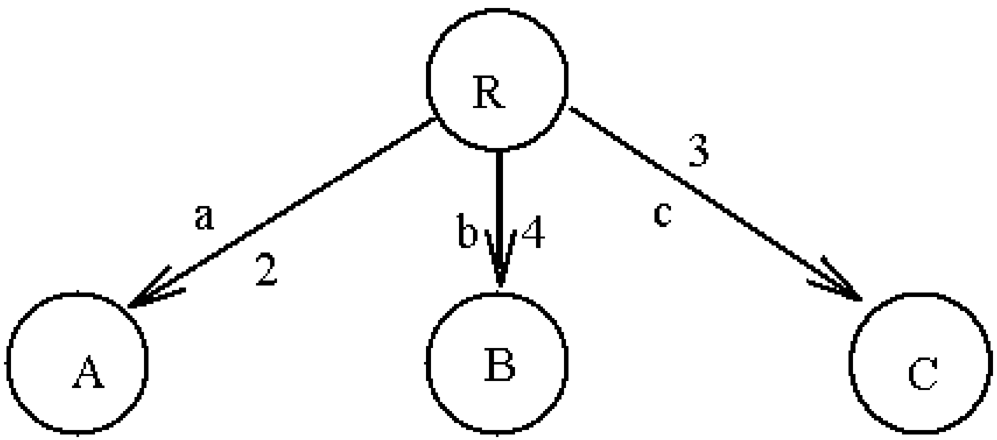
The A* algorithm

- Stop when the selected node is also a goal node. It is optimal iff h(n)≤h*(n)
- E.g.: To find a shortest path from node s to node t



The A* algorithm

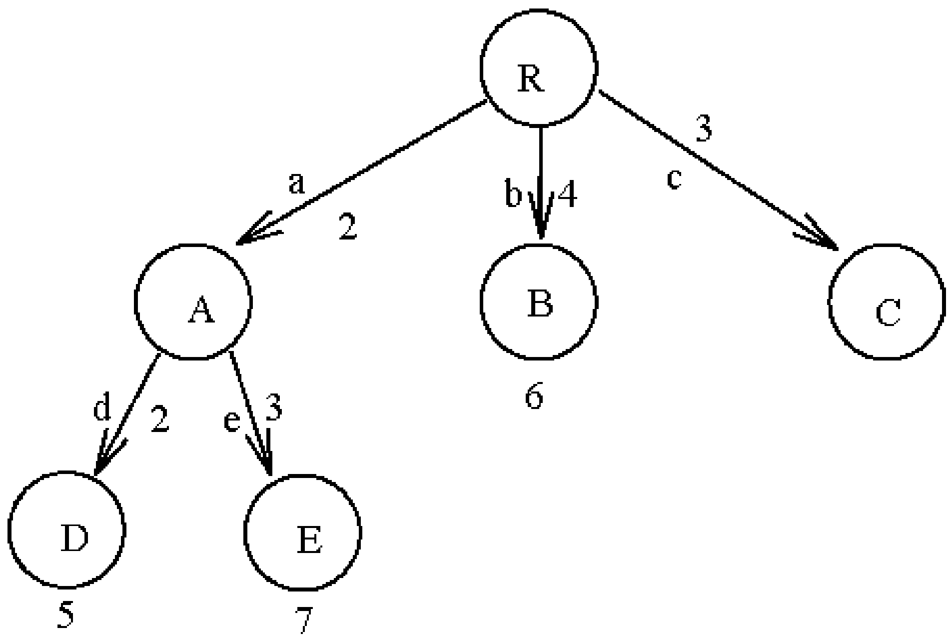
- **Step 1.**



g(A)=2 h(A)=min{2,3}=2 f(A)=2+2=4
g(B)=4 h(B)=min{2}=2 f(B)=4+2=6
g(C)=3 h(C)=min{2,2}=2 f(C)= 3+2=5

The A* algorithm

- **Step 2. Expand A**

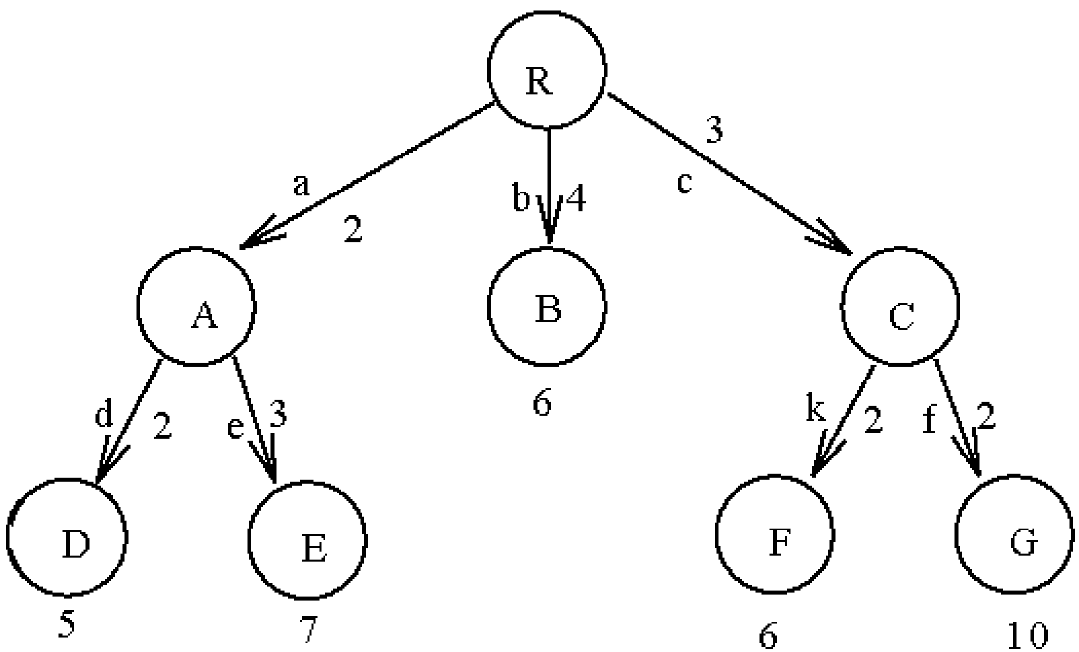


g(D)=2+2=4 h(D)=min{3,1}=1 f(D)=4+1=5

g(E)=2+3=5 h(E)=min{2,2}=2 f(E)=5+2=7

The A* algorithm

- **Step 3. Expand C**

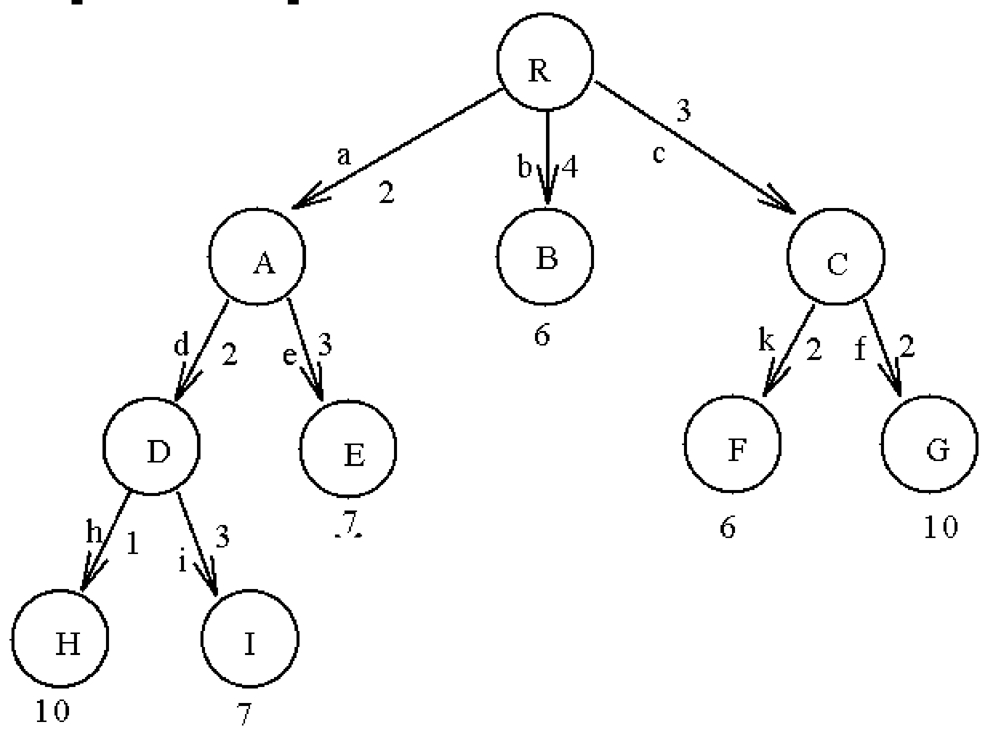


g(F)=3+2=5 h(F)=min{3,1}=1 f(F)=5+1=6

g(G) =3+2=5 h(G)=min{5}=5 f(G) =5+5=10

The A* algorithm

- **Step 4. Expand D**

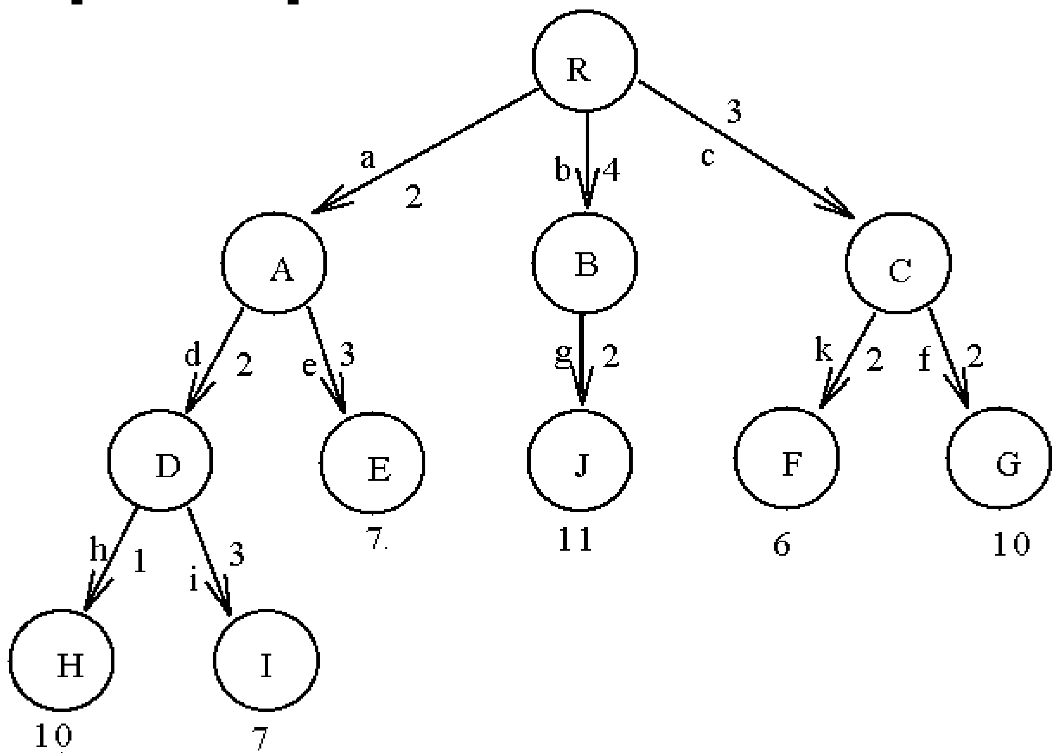


$g(H)=2+2+1=5$ $h(H)=\min\{5\}=5$ $f(H)=5+5=10$

$g(I)=2+2+3=7$ $h(I)=0$ $f(I)=7+0=7$

The A* algorithm

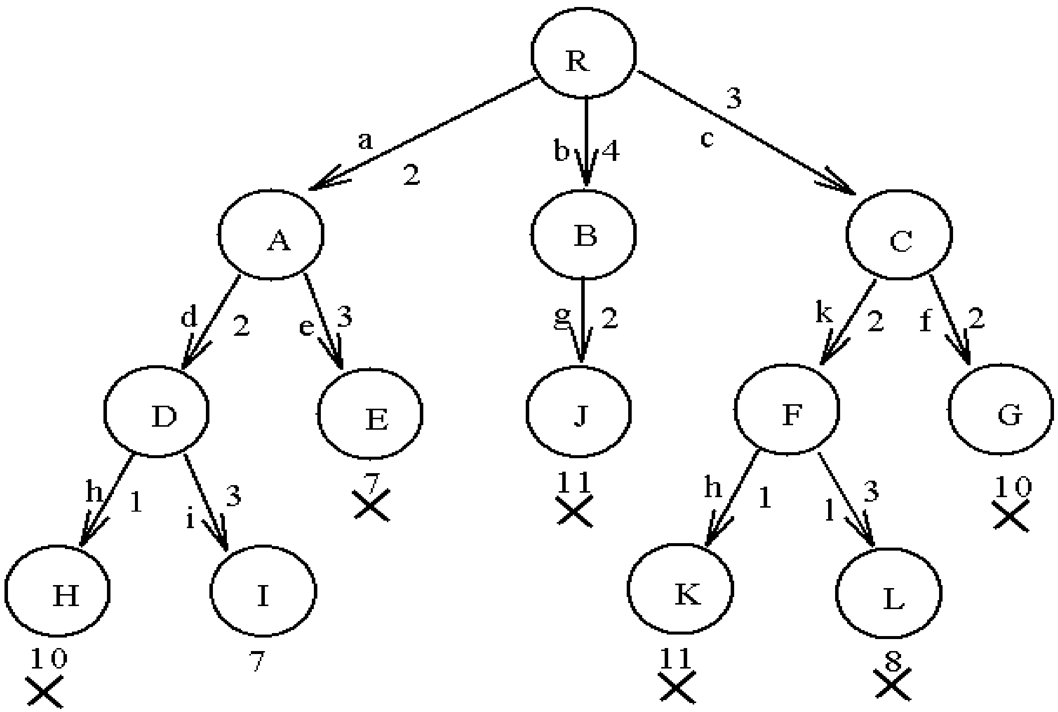
- **Step 5. Expand B**



$g(J)=4+2=6$ $h(J)=\min\{5\}=5$ $f(J)=6+5=11$

The A* algorithm

- **Step 6. Expand F**



I is selected to expand. The A* algorithm stops, since I is a goal node.

$g(K)=3+2+1=6$ $h(K)=\min\{5\}=5$ $f(K)=6+5=11$

$g(L)=3+2+3=8$ $h(L)=0$ $f(L)=8+0=8$

The A* Algorithm

- Can be considered as a special type of branch-and-bound algorithm.
- When the first feasible solution is found, all nodes in the heap (priority queue) are terminated.
- * stands for “real”
- “A* algorithm” stands for “real good algorithm”

Q&A