Chapter 5: Tree Searching Strategies

Chin Lung Lu

Department of Computer Science

National Tsing Hua University

Tree searching problem

- The solutions of many problems may be represented by trees and solving these problems becomes a tree searching problem.

Examples:

- 8-puzzle problem
- Hamiltonian circuit problem
- Satisfiability problem

8-Puzzle problem

Definition:
Given an initial square frame which has 8 numbered tiles and an empty spot, move the numbered tiles around so that the final state is reached.

	2	3
1	8	4
7	6	5

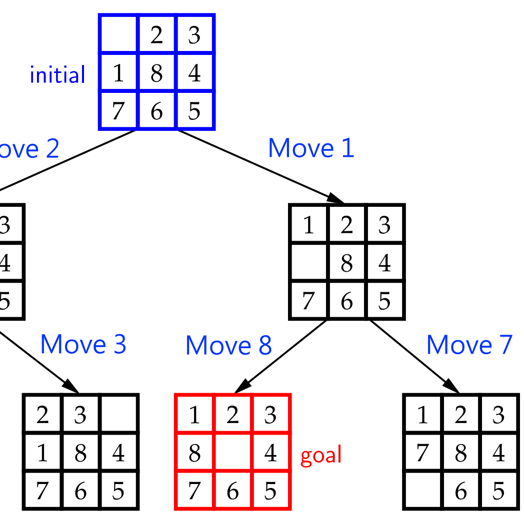
initial state

1	2	3
8		4
7	6	5

final state

- Note that the numbered tiles can be moved only horizontally or vertically to the empty spot.

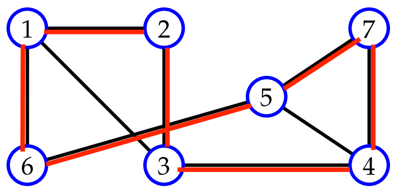
Searching tree of 8-puzzle problem



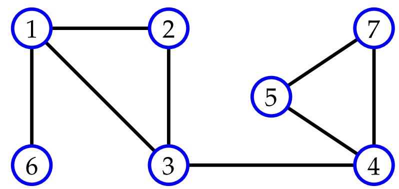
Hamiltonian circuit problem

Definition:

- Given a graph $G = (V, E)$, determine whether or not G has a Hamiltonian circuit.
- A Hamiltonian circuit is a round-trip route (cycle) of G that visits every vertex exactly once and returns to its starting position.

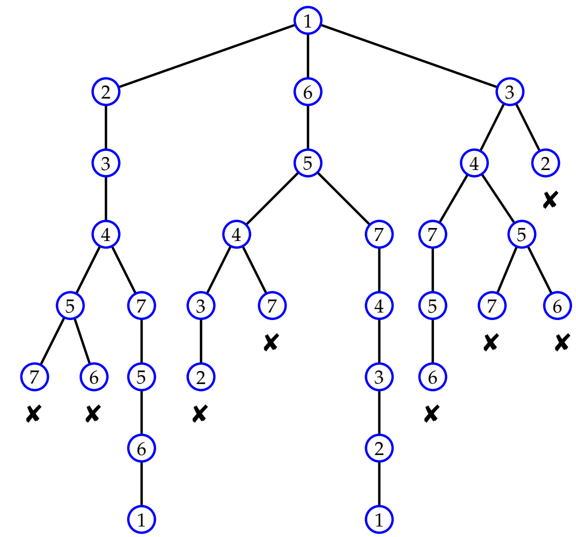


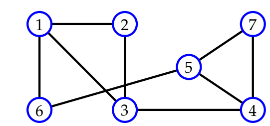
Has a Hamiltonian circuit



Has no Hamiltonian circuit

Searching tree of Hamiltonian circuit problem





Satisfibility problem

Definition:

Given a set of clauses (a logical formula), determine whether this set of clauses is satisfiable.

- Each variable is assigned either T (true) or F (false).
- If an assignment makes all the clauses true, this assignment satisfies this formula.

(1) satisfiable fomula:

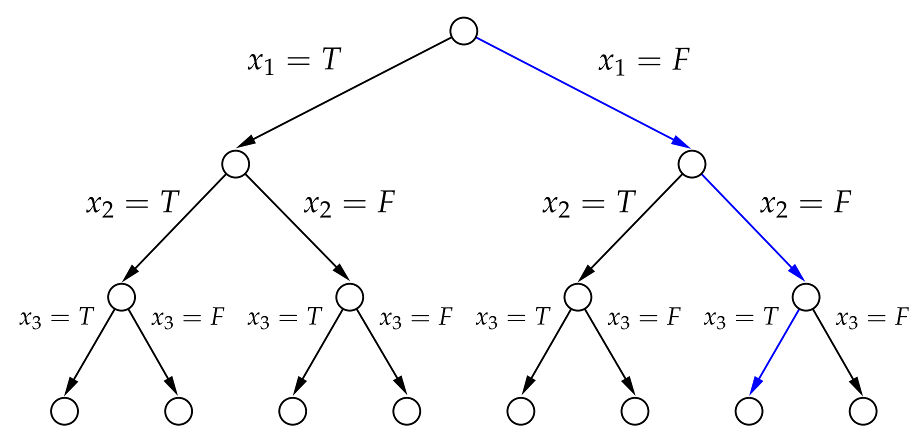
$$\begin{array}{lr} x_1 \vee x_2 \vee x_3 & (1) \\ -x_1 & (2) \\ -x_2 & (3) \end{array}$$

(2) unsatisfiable fomula:

$$\begin{array}{lr} x_1 \vee x_2 & (1) \\ -x_1 & (2) \\ -x_2 & (3) \end{array}$$

Searching tree of satisfiability problem

- Basically, there are 2^n possible assignments for n variables.
- These 2^n assignments can be represented by a tree.



- The formula $(x_1 \vee x_2 \vee x_3)(-x_1)(-x_2)$ is satisfiable.

Strategies of tree searching

1. Breadth-first search
2. Depth-first search
3. Hill climbing
4. Best-first search
5. Branch and bound strategy
6. A* algorithm

Breadth-first search

Strategy of breadth-first search:

In breadth-first search, all nodes on level i of tree are examined before any node on level $i+1$ is examined.

- Use queue to hold all of the expanded nodes.

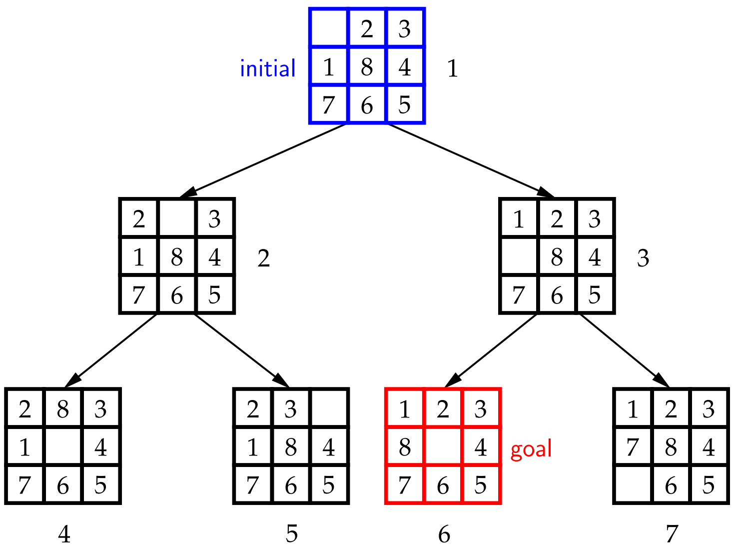
Breadth-first search (cont'd)

Breadth-first search algorithm:

---

1. Form a one-element queue Q consisting of the root.
2. **if** the first element q_1 of Q is a goal node **then** stop **else** go to step 3.
3. Remove q_1 from Q and add q_1's descendants, if any, to the end of Q.
4. **if** Q is empty **then** failure **else** go to step 2.

Breadth-first search for 8-puzzle problem



Depth-first search

Strategy of depth-first search:

In depth-first search, the deepest node is selected to expand in the process.

- Use stack to hold all of the expanded nodes.

Depth-first search (cont'd)

Depth-first search algorithm:

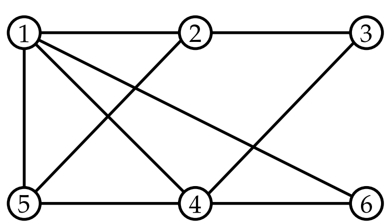
---

1. Form a one-element stack S consisting of the root.
2. **if** the top element s_1 of S is a goal node **then** stop **else** go to step 3.
3. Remove s_1 from S and add s_1's descendants, if any, to the top of S.
4. **if** S is empty **then** failure **else** go to step 2.

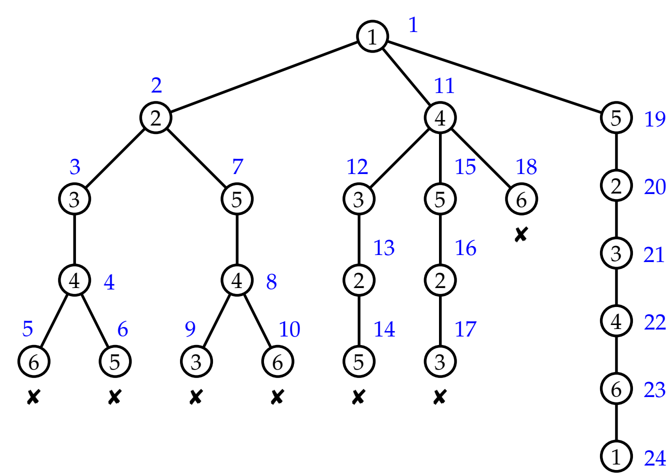
Depth-first search for Hamiltonian cycle

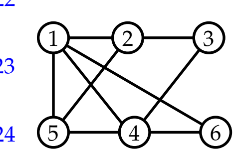
Example:

Find a Hamiltonian cycle of the following graph by the depth-first search.



Depth-first search for Hamiltonian cycle (cont'd)





Hill climbing

- After reading the depth-first search strategy, we may wonder about which node we should select to expand among all the descendants?

Strategy of hill climbing:

Hill climbing is a variant of depth-first search in which some greedy method is used to decide which direction to move in the search space.

- The better the greedy is, the better the hill climbing is.

Hill climbing for 8-puzzle problem

- For the 8-puzzle problem, the greedy method uses a evaluation function $f(n) = w(n)$ to order the choices, where $w(n)$ is the number of misplaced tiles in node n.
- According to the evaluation function, the locally optimal one is selected to expand.

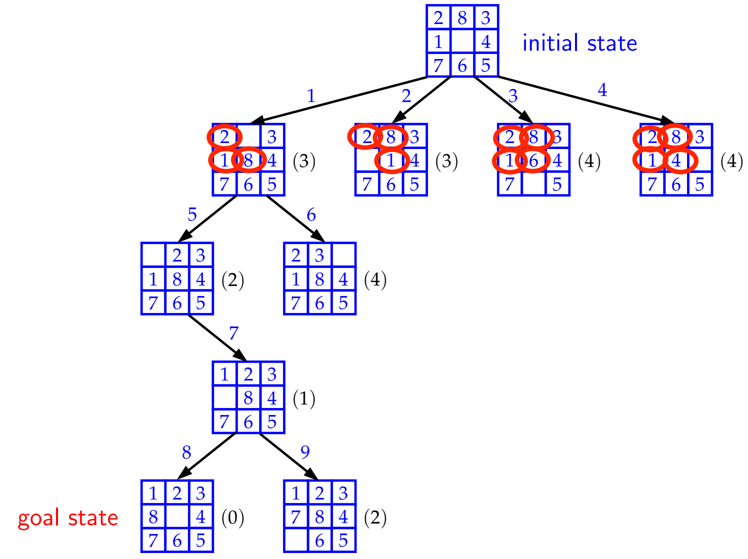
Hill climbing for 8-puzzle problem (cont'd)

Hill climbing algorithm:

---

1. Form a one-element stack S consisting of the root.
2. **if** the top element s_1 of S is a goal node **then** stop
   else go to step 3.
3. Remove s_1 from S and add s_1's descendants, ordered by the evaluation function, to the top of S.
4. **if** S is empty **then** failure **else** go to step 2.

Hill climbing for 8-puzzle problem (cont'd)



Best-first search

Strategy of best-first search:

In best-first search, there is an evaluation function and we select the node with the least cost among all nodes that have been generated so far.

- The best-first search is a way of combining the advantages of both depth-first and breadth-first searches into a single method.
- The best-first search approach has a global view, while the hill climbing has local view.
- Use heap to hold all of the expanded nodes, where the heap is constructed using the evaluation function.

f(n) = w(n), where w(n) is the number of misplaced tile in node n

Best-first search for 8-puzzle problem

Best-first search algorithm:

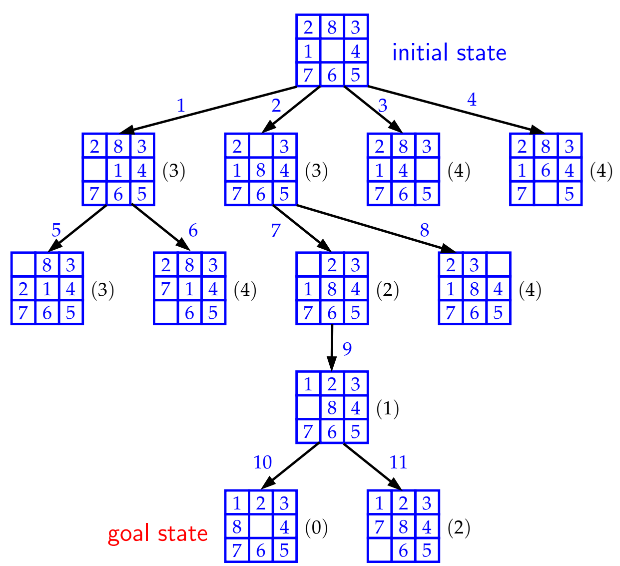
---

1. Form a one-element heap H consisting of the root.
2. **if** the root r of H is a goal node **then** stop
   else go to step 3.
3. Remove r from H and add r's descendants to H.
4. **if** H is empty **then** failure **else** go to step 2.

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems typically exponential in terms of time complexity and may require exploring all possible permutations in worst case.

B & B 可以視為一種對可行解進行窮舉的演算法，但是和窮舉法所不同的是，分支定界演算法在對某一分支進行檢索之前會先算出該分支的上界或下界，如果界限不比目前最佳解更好，那麼該分支就會被捨棄，從而節約了大量的時間。分支定界演算法非常依賴合適的上界或下界，如果無法找到合適的界限，該演算法將會退化為窮舉法。

Best-first search for 8-puzzle problem (cont'd)



Branch and bound strategy

- The branch and bound is an efficient strategy to solve a large number of combinatorial problems.

Branch mechanism:

- The solution space (all feasible solutions) usually is represented by a tree.
- The branch mechanism is a way of generating the branches of the solution space (partition the solution space into smaller sub-spaces) .

Bound mechanism:

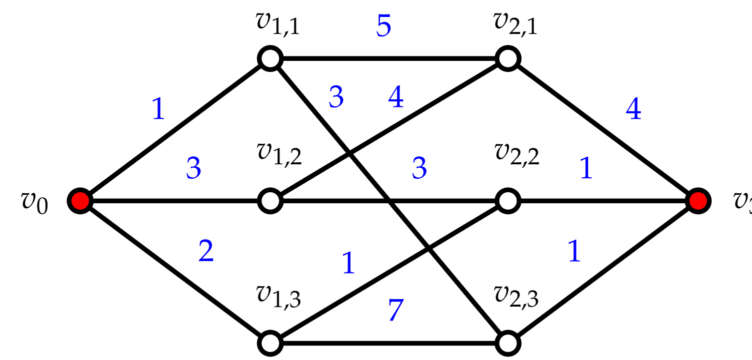
- The bound mechanism is a way of bounding the branches to avoid the exhaustive search of solution space.

How to bound a branch?

- Find an upper bound of an optimal solution.
- Predict a lower bound for a branch.
- If the lower bound of a branch is greater than the upper bound, then the branch is terminated.

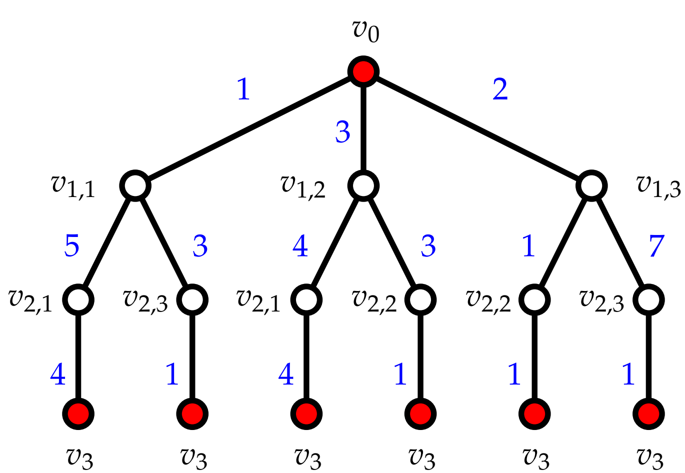
Shortest path problem

- Find a shortest path from v_0 to v_3 in the following graph.

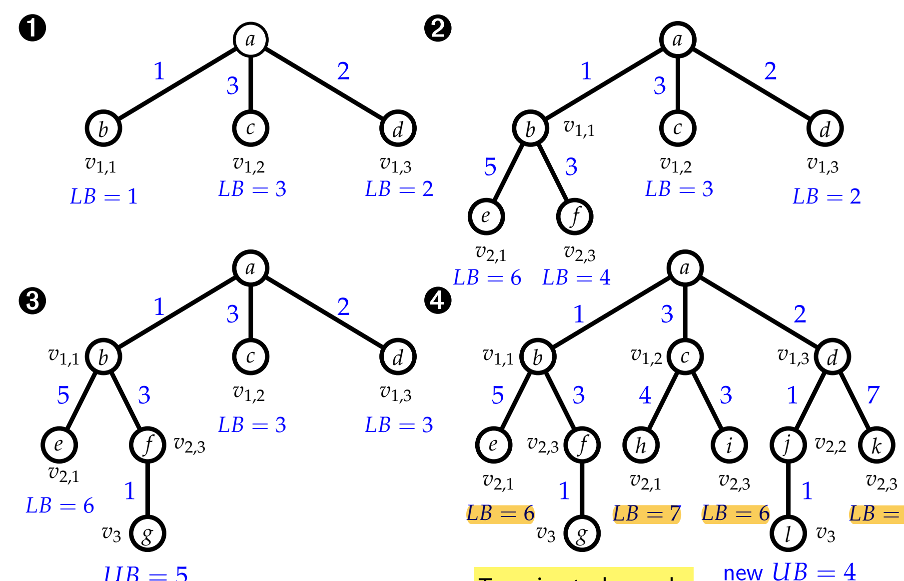


Shortest path problem (cont'd)

- A search tree of the problem with six feasible solutions.



Shortest path problem (cont'd)



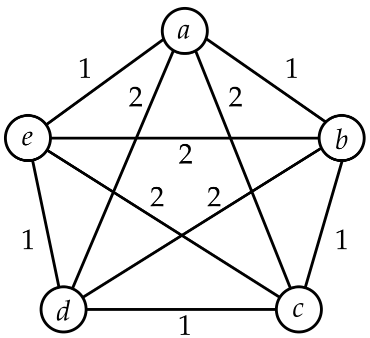
Terminate branch
if LB > current UB

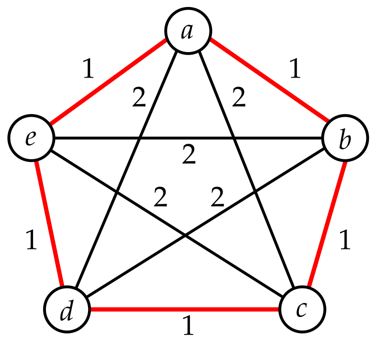
Terminate branch id LB > curre

Traveling salesperson problem (TSP)

Definition:

Given n cities and the costs of traveling from one to the other, find the cheapest round-trip route that visits each city exactly once and then returns to the starting city.





Traveling salesperson problem (TSP) (cont'd)

- The traveling salesperson problem is equivalently to search for the shortest Hamiltonian cycle in a weighted graph $G = (V, E)$.
- The brute-force method requires $\mathcal{O}(n!)$ time.
- The traveling salesperson problem is an NP-hard problem.
- It is hard to solve TSP in worst case in polynomial time.

Branch and bound method for TSP

Basic principle of the branch and bound strategy for TSP:

- Find a way to split the solution space.
- Find a way to obtain an upper bound of an optimal solution.
- Find a way to predict a low bound for a branch corresponding to a class of solutions.
- If the lower bound of a branch exceeds the upper bound, this branch can be terminated since it has no optimal solution.

Assumptions to simplify discussion:

- There is no arc between a vertex and itself.
- There is an arc between every pair of vertices that is associated with a non-negative cost (i.e., G is a complete graph).

Branch and bound method for TSP (cont'd)

- For example, consider the following cost matrix of TSP:

(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	∞

- Note that (i,j) is a directed edge from vertex i to vertex j.

Branch and bound method for TSP (cont'd)

Observation 1:
If a constant is subtracted from any row or any column of the cost matrix, an optimal solution does not change.

Observation 2:
If we subtract the minimum cost of each row from the cost matrix, the total amount that we subtract will be a lower bound for the optimal solution.

Branch and bound method for TSP (cont'd)

- Based on Observation 1, we can subtract $3, 4, 16, 7, 25, 3$ and 26 from rows 1 to 7, respectively, and obtain a reduced cost matrix as shown on the next slide.

(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	∞

- Therefore, the current lower bound for the optimal solution is $3+4+16+7+25+3+26 = 84$ according to Observation 2.

Branch and bound method for TSP (cont'd)

- Since columns 3, 4 and 7 still contain no zero, we further subtract 7, 1 and 4 from columns 3, 4 and 7, respectively.

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

- Therefore, the new lower bound for the optimal solution is $84 + 7 + 1 + 4 = 96$.

Branch and bound method for TSP (cont'd)

- Now, the reduced cost matrix is shown as follows:

(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	0	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

Branch and bound method for TSP (cont'd)

Question 1:

Suppose we know that the tour does include arc $(4,6)$, whose cost is zero now. What is the lower bound of the cost of this tour?

(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	0	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

- The answer is still 96.

Branch and bound method for TSP (cont'd)

Question 2:

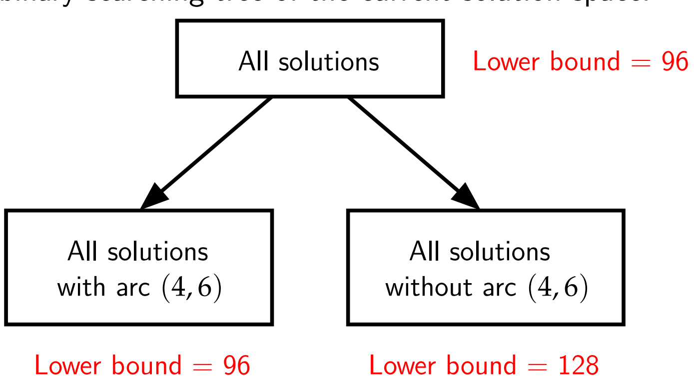
Suppose we know that the tour does not include arc $(4,6)$. What will the new lower bound be?

(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	$\boxed{0}$	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

- The tour must include some other arc from 4, where arc $(4,1)$ has the least cost 32, and some other arc to 6, where arc $(5,6)$ has cost zero ($\therefore$ the new lower bound is $96+32+0=128$).

Split of TSP solutions

- We can split a solution space into two groups: one group including arc $(4,6)$ and the other group excluding this arc.
- The binary searching tree of the current solution space:



Branch and bound method for TSP (cont'd)

Question 3:

Why did we choose arc $(4,6)$ to split the solution space?

- The reason is that arc $(4,6)$ will cause the largest increase of lower bound.
- For example, suppose we use arc $(3,5)$ to split. Then we can only increase the lower bound by $1+17=18$.

加上同個 row 中最小的以及同個 column 中最小的

(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	$\boxed{0}$	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	∞

Branch and bound method for TSP (cont'd)

- In the subtree with arc $(4,6)$ included, we must delete the 4th row and 6th column from the cost matrix, and set the cost of arc $(6,4)$ as ∞ since $(4,6)$ is used.
- The cost matrix becomes as follows:

(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	∞

- Row 5 now contains no zero.

Branch and bound method for TSP (cont'd)

- Therefore, we subtract 3 from row 5 and obtain the new cost matrix as follows:

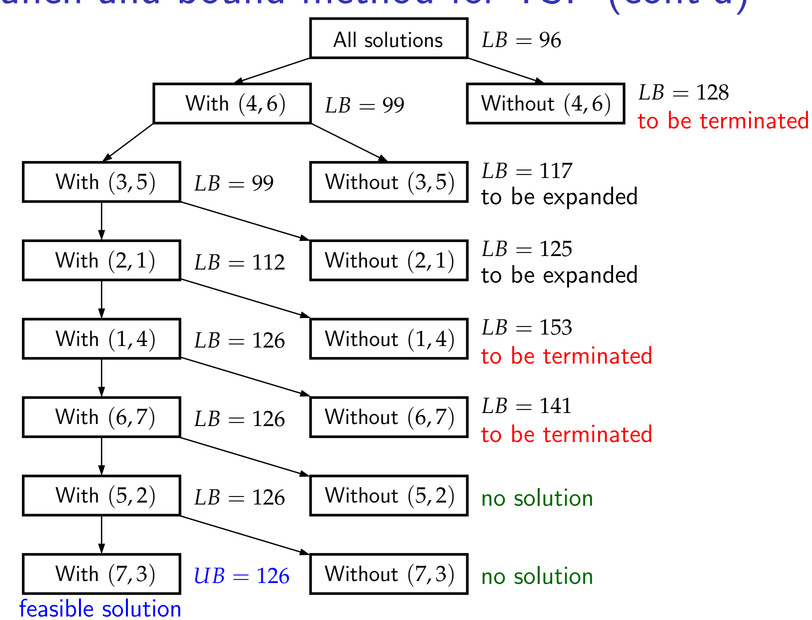
(i,j)	1	2	3	4	5	7
1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5
5	0	18	53	4	∞	25
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

- Therefore, the lower bound of the left tree is $96 + 3 = 99$.

Branch and bound method for TSP (cont'd)

- For the cost matrix of the right subtree (i.e., solutions without arc $(4,6)$), we only have to set the cost of $(4,6)$ as ∞.
- The splitting process above would continue and would produce the binary decision tree of the solution space.
- In this process, if we follow the path with the least cost, we will obtain a feasible solution with cost 126 (an upper bound).
- Any branching will be terminated if its lower bound exceeds the current upper bound or it represents an infeasible solution.

Branch and bound method for TSP (cont'd)



Branch and bound method for TSP (cont'd)

- Another point needs to be explained by considering the reduced cost matrix of all solutions with arcs $(4,6)$, $(3,5)$ and $(2,1)$.

(i,j)	2	3	4	7
1	∞	74	0	41
5	14	∞	0	21
6	85	8	∞	0
7	0	0	0	∞

- We may use arc $(1,4)$ to split and the cost matrix for the left tree will be shown as follows.

(i,j)	2	3	7
5	14	∞	21
6	85	8	0
7	0	0	∞

Branch and bound method for TSP (cont'd)

- Arcs $(4,6)$ and $(2,1)$ are already included in the solution and arc $(1,4)$ is to be added.
- We must prevent arc $(6,2)$ from being used.
- If arc $(6,2)$ is used, there will be a loop $2 \to 1 \to 4 \to 6 \to 2$ that is forbidden.
- Hence, we must set the cost of $(6,2)$ as ∞ in the cost matrix of the left tree.

(i,j)	2	3	7
5	14	∞	21
6	∞	8	0
7	0	0	∞

0/1 Knapsack problem

Definition:

Given positive integers $P_1, P_2, \ldots, P_n$, $W_1, W_2, \ldots, W_n$ and M, find $X_1, X_2, \ldots, X_n$, where $X_i = 0$ or 1 for all $i = 1, 2, \ldots, n$, such that $\sum_{i=1}^{n} P_i X_i$ is maximized subject to $\sum_{i=1}^{n} W_i X_i \le M$.

Example:

- Let $(P_1, P_2, P_3) = (25, 24, 15)$, $(W_1, W_2, W_3) = (18, 15, 10)$ and $M = 20$.
- Then the optimal solution is $(X_1, X_2, X_3) = (1, 0, 0)$.

- Note that the 0/1 knapsack problem is NP-hard.

0/1 Knapsack problem (cont'd)

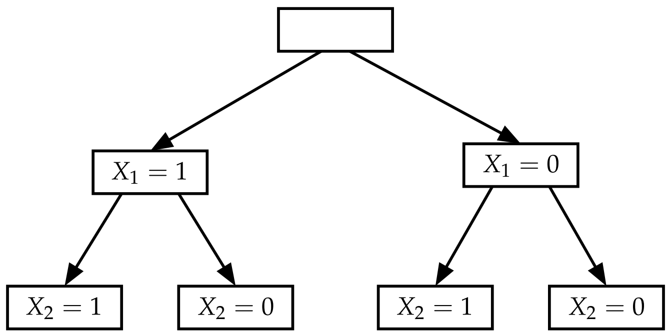
- The original 0/1 knapsack problem is a maximization problem.
- For designing a branch and bound algorithm, we modify it into a minimization problem as follows.

Minimization version of the 0/1 knapsack problem:

Given positive integers $P_1, P_2, \ldots, P_n$, $W_1, W_2, \ldots, W_n$ and M, find $X_1, X_2, \ldots, X_n$, where $X_i = 0$ or 1 for all $i = 1, 2, \ldots, n$, such that $-\sum_{i=1}^{n} P_i X_i$ is minimized subject to $\sum_{i=1}^{n} W_i X_i \le M$.

Braching mechanism for 0/1 Knapsack problem

- The first branching splits all the solutions into two groups: the solutions with $X_1 = 1$ and the solutions with $X_1 = 0$.
- For each group, X_2 is used to split the solutions.



- After all $X_1, X_2, \ldots, X_n$ are enumerated, a feasible solution will be found.

Basic principle of branch and bound strategy

Traveling salesperson problem

- We split the solutions of the traveling salesperson problem into two groups.
- For each group, a lower bound is found.
- At the same time, we try to search for a feasible solution.
- Whenever a feasible solution is found, an upper bound is found.
- The expansion of a node is terminated if and only if one of the following conditions is satisfied.
  1. The node itself represents an infeasible solution.
  2. The lower bound of this node is greater than or equal to the presently found lowest upper bound.

Basic principle of branch and bound strategy

0/1 Knapsack problem

- We split all the solutions of the 0/1 knapsack problem into two groups.
- For each such group, not only a lower bound is found, but also an upper bound is found by finding a feasible solution.
- We terminate the branching if and only if one of the following conditions is satisfied:
  1. The node itself represents an infeasible solution.
  2. The lower bound of this node is higher than or equal to the presently found lowest upper bound.
  3. The lower bound of this node is equal to its upper bound (the upper bound cannot be lowered).

Basic principle of branch and bound strategy

0/1 Knapsack problem (cont'd)

- How can we find an upper bound and a lower bound of a node?
- A node of the branch and bound tree corresponds to a partially constructed solution.
- A lower bound of this node corresponds to the highest possible profit associated with this partially constructed solution.
- In addition, the upper bound of this node means the cost of a feasible solution that corresponds to the partially constructed solution.

B&B method for 0/1 knapsack problem

- Below, we illustrate the branch and bound method for the 0/1 knapsack problem through an example as follows.

Example:

Consider the following instance of the 0/1 knapsack problem:

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

and $M = 34$.

- Note that $\boxed{\frac{P_i}{W_i} \geq \frac{P_{i+1}}{W_{i+1}}}$ for $i = 1, 2, \ldots, 6$, which is required as we shall see later.

B&B method for 0/1 knapsack problem

How can we find a lower bound of a node?

- In the 0/1 knapsack problem, X_i is restricted to 0 or 1.
- If we relax this restriction, we shall obtain a better result and this better result will be used as our lower bound.
- That is, we may let X_i be between 0 and 1.
- If we do this, the 0/1 knapsack problem becomes the knapsack problem.

B&B method for 0/1 knapsack problem

How can we find a lower bound of a node? (cont'd)

Knapsack problem:

Given positive integers $P_1, P_2, \ldots, P_n$, $W_1, W_2, \ldots, W_n$ and M, find $X_1, X_2, \ldots, X_n$, where $0 \leq X_i \leq 1$ for $i = 1, 2, \ldots, n$, such that

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

is minimized subject to

$$\sum_{i=1}^{n} W_i X_i \leq M.$$

B&B method for 0/1 knapsack problem

How can we find a lower bound of a node? (cont'd)

- Let $Y = -\sum_{i=1}^{n} P_i X_i$ be an optimal solution for $\boxed{0/1}$ knapsack problem and let $Y' = -\sum_{i=1}^{n} P_i X'_i$ be an optimal solution for knapsack problem.
- It is clear that $Y' \leq Y$.
- That is, a solution of the knapsack problem can be served as a lower bound of the solution of the 0/1 knapsack problem.
- Recall that the greedy method can be used to find an optimal solution of the knapsack problem (see Chapter 3).

B&B method for 0/1 knapsack problem

How can we find a lower bound of a node? (cont'd)

Example:

Consider the following instance:

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

and $M = 34$.

- Assume that we have already set $X_1 = X_2 = 1$.
- We cannot let $X_3 = 1$ since $W_1 + W_2 + W_3 = 37 > M = 34$.
- However, if we let X_3 be somewhat between 0 and 1, we can obtain an optimal solution for the knapsack problem.

B&B method for 0/1 knapsack problem

How can we find a lower bound of a node? (cont'd)

- We can set X_3 a value between 0 and 1 such that:

$$W_1 + W_2 + W_3X_3 = 10 + 19 + 8X_3 = M = 34$$

- Hence, $X_3 = \frac{5}{8}$.
- Note that the above method of finding an optimal solution of knapsack problem is correct, since $\frac{P_i}{W_i} \geq \frac{P_{i+1}}{W_{i+1}}$.
- With this value, a lower bound of the 0/1 knapsack problem is found to be $-(6 + 10 + \frac{5}{8} \times 4) = -18.5$.
- In our instance (in which profits are integers), we can use the higher limit -18 to serve as the lower bound.

B&B method for 0/1 knapsack problem

How can we find an upper bound of a node?

- Let's consider a node corresponding to the case where:

$$X_1 = 1, X_2 = X_3 = X_4 = 0$$

- In this case, an upper bound corresponds to a feasible solution that is obtained by starting the smallest available $i = 5$ and scanning towards the larger i's untill M is exceeded.

$$X_1 = 1, X_2 = X_3 = X_4 = 0, X_5 = X_6 = 1$$

- The profit of this upper bound is:

$$-(P_1 + P_5 + P_6) = -(6 + 6 + 4) = -16$$

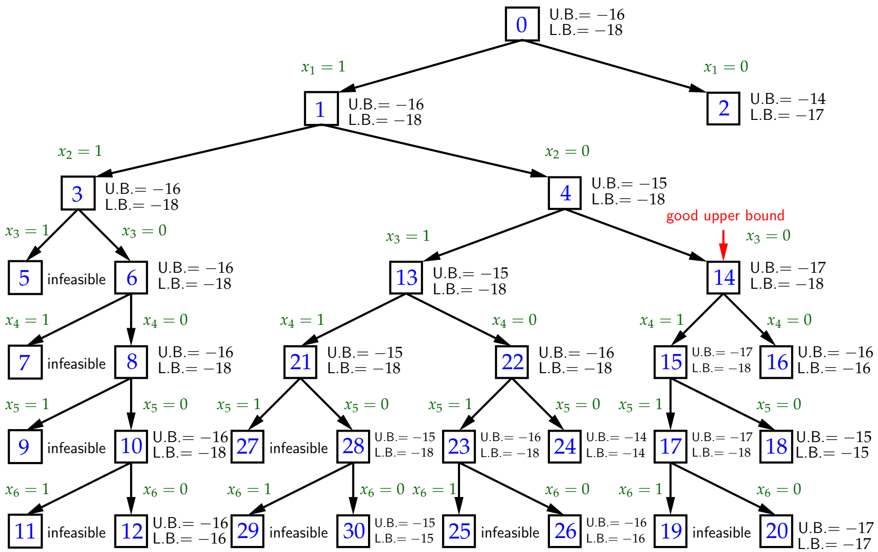
- This means that when the node corresponding this case is concerned, if we further expand it, we shall obtain a feasible solution with cost -16.
- This is why we call this -16 the upper bound of this node.

B&B method for 0/1 knapsack problem (cont'd)

The 0/1 knapsack problem can be solved by the branch and bound strategy with a searching tree as shown on the next slide.

- The number in each node indicates the sequence in which the node is expanded.
- We use the best-first search rule to expand the node.
- That is, we expand the node with the best lower bound.
- If two nodes have the same lower bound, then we expand the node with the lower upper bound.

B&B strategy for 0/1 knapsack problem (cont'd)



B&B strategy for 0/1 knapsack problem (cont'd)

- Note that node 2 in the branch and bound tree is terminated since its lower bound is equal to the upper bound of node 14.
- Moreover, all other nodes are terminated because each of them is infeasible or its lower bound is equal to its upper bound.

A^* algorithm

- A^* is a tree searching strategy favored by artificial intelligence researchers.
- Recall that in the branch and bound strategy, our effort is to make sure that many solutions need not be further probed because they will not lead to optimal solutions.
- The A^* algorithm emphasizes another viewpoint: it will tell us that under certain situations, a feasible solution that we have obtained must be optimal one and therefore we can stop the algorithm.

Strategy of A^* algorithm

Tree searching strategy:

The A^* algorithm uses the best-first strategy to select the next node to be expanded.

- The critical element of the A^* algorithm is the cost function.
- It'll compute the cost of each expanded node and choose the expanded node with the smallest cost for the next expansion.

Termination rule:

If a selected node is a goal node, then this selected node represents an optimal solution and the process of A^* algorithm is terminated.

Cost function $f^*(n)$

- Suppose we use a tree searching algorithm to solve a problem.
- Let $g(n)$ denote the path length from the root of the searching tree to node n.
- Let $h^*(n)$ denote the optimal path length from node n to a goal node.
- The cost of node n is $f^*(n) = g(n) + h^*(n)$.
- Note that $f^*(n)$ is generally unknown, since $h^*(n)$ is unknown.

Estimated cost function $f(n)$

- But, we can estimate $h^*(n)$, although it is generally unknown.
- There are many ways to estimate $h^*(n)$, but the A^* algorithm always uses a conservative estimation $h(n)$ of $h^*(n)$.
- That is, $h(n) \leq h^*(n)$ for node n.
- We let $f(n) = g(n) + h(n)$ and use it as the cost of node n.
- In this case, we have $f(n) \leq f^*(n)$ for node n.

Correctness of A^* algorithm

- Let t be the selected goal node with cost $f(t)$.
- Let n be an arbitrary expanded node with cost $f(n)$.
- We then have $f(t) \leq f(n)$ for all n (all expanded nodes), since the A^* algorithm uses the best-first search (least cost rule).
- We also have $f(n) \leq f^*(n)$ for all n, because the A^* algorithm uses conservative estimation of $h^*(n)$ (i.e., $h(n) \leq h^*(n)$).
- But, one of the $f^*(n)$'s must be an optimal solution.
- Let s denote such an expanded node.
- That is, $f^*(s)$ is the value of an optimal solution.
- By the above discussion, we have $f(t) \leq f^*(s)$.

Correctness of A^* algorithm (cont'd)

- Since t is a goal node, we have $h(t) = 0$.

$$\begin{aligned} \therefore\ & f(t) = g(t) + h(t) = g(t) \\ \because\ & f(t) \leq f^*(s) \\ \therefore\ & g(t) = f(t) \leq f^*(s) \qquad (1) \end{aligned}$$

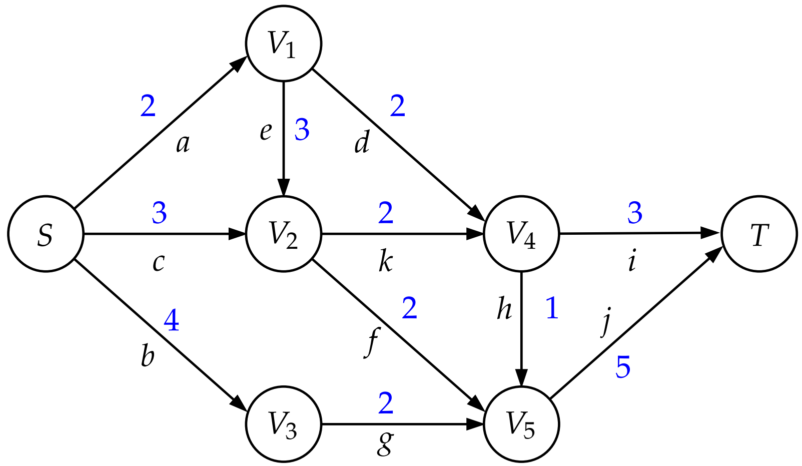
- Moreover, $f(t) = g(t)$ is the value of a feasible solution (since reaching a goal node).

$$\therefore\ g(t) = f(t) \geq f^*(s) \qquad (2)$$

- By (1) and (2), therefore, we have $g(t) = f^*(s)$.

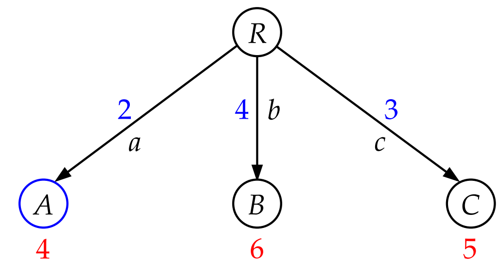
A^* algorithm

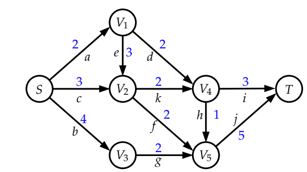
Finding the shortest path $S \to T$



A^* algorithm of shortest path $S \to T$ (cont'd)

Step 1: Expand the root R

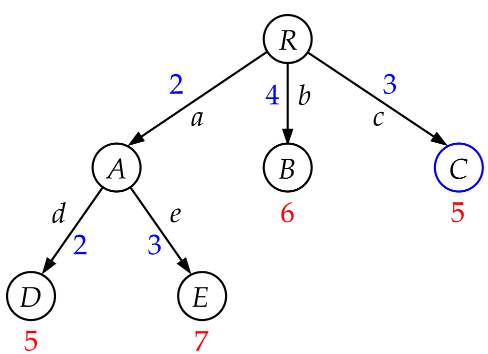


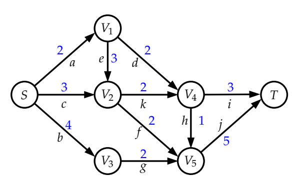


- $g(A) = 2$, $h(A) = \min\{2, 3\} = 2$ and $f(A) = 4$
- $g(B) = 4$, $h(B) = \min\{2\} = 2$ and $f(B) = 6$
- $g(C) = 3$, $h(C) = \min\{2, 2\} = 2$ and $f(C) = 5$

A^* algorithm of shortest path $S \to T$ (cont'd)

Step 2: Expand A

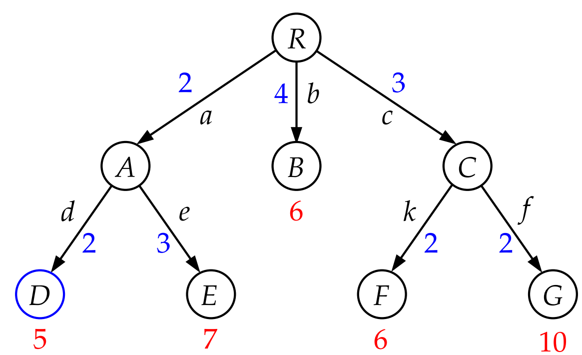


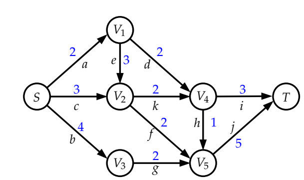


- $g(D) = 4$, $h(D) = \min\{3, 1\} = 1$ and $f(D) = 5$
- $g(E) = 5$, $h(E) = \min\{2, 2\} = 2$ and $f(E) = 7$

A^* algorithm of shortest path $S \to T$ (cont'd)

Step 3: Expand C

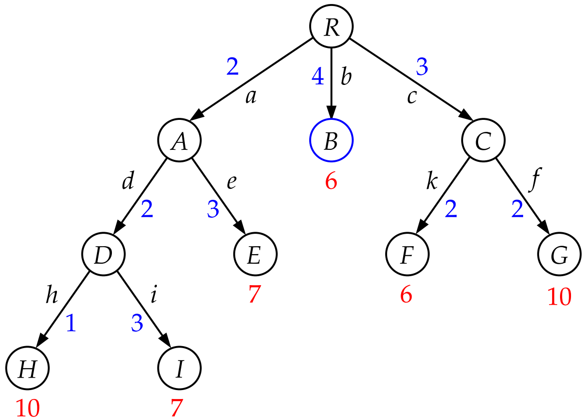


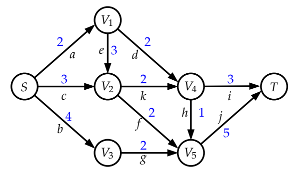


- $g(F) = 5$, $h(F) = \min\{3, 1\} = 1$ and $f(F) = 6$
- $g(G) = 5$, $h(G) = \min\{5\} = 5$ and $f(G) = 10$

A^* algorithm of shortest path $S \to T$ (cont'd)

Step 4: Expand D

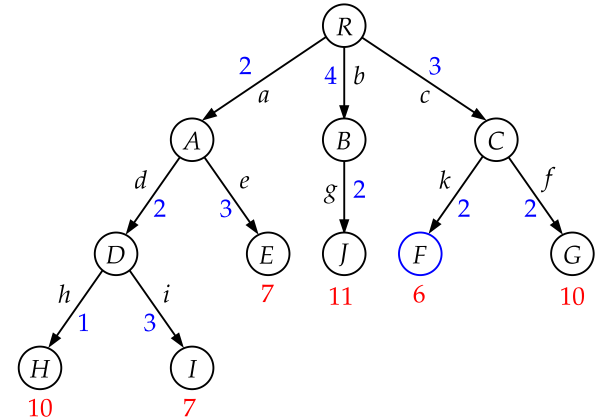


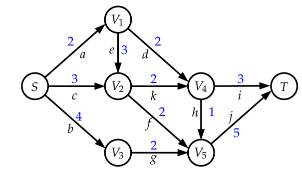


- $g(H) = 5$, $h(H) = \min\{5\} = 5$ and $f(H) = 10$
- $g(I) = 7$, $h(I) = 0$ and $f(I) = 7$

A^* algorithm of shortest path $S \to T$ (cont'd)

Step 5: Expand B

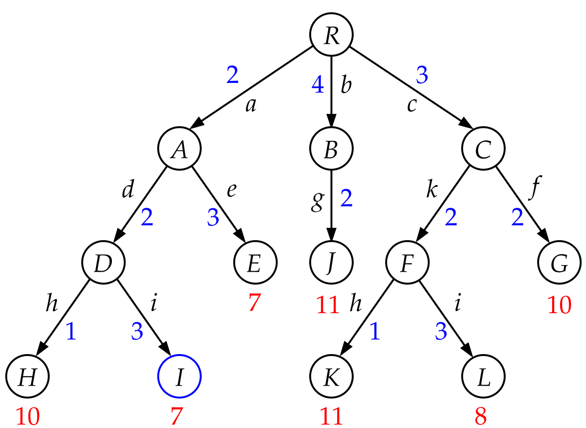


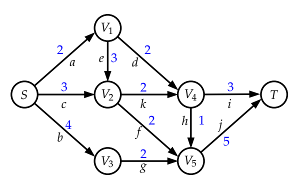


- $g(J) = 6$, $h(J) = \min\{5\} = 5$ and $f(J) = 11$

A^* algorithm of shortest path $S \to T$ (cont'd)

Step 6: Expand F



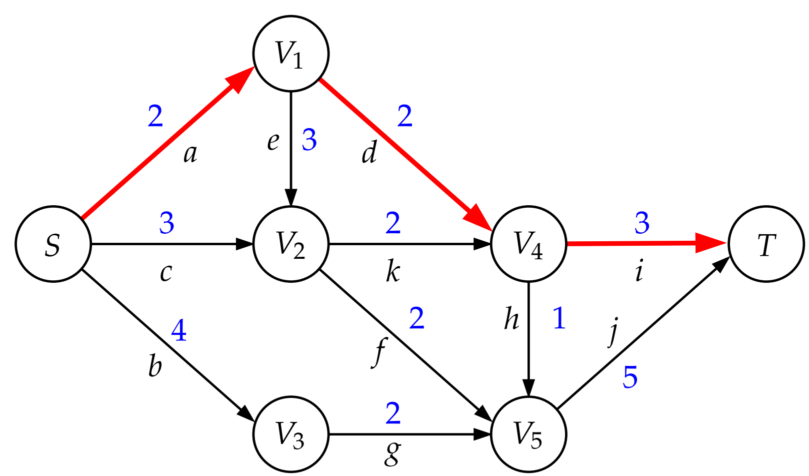


- $g(K) = 6$, $h(K) = \min\{5\} = 5$ and $f(K) = 11$
- $g(L) = 8$, $h(K) = 0$ and $f(K) = 8$

A^* algorithm of shortest path $S \to T$ (cont'd)

Step 7: Expand I

- Since I is a goal node, we stop and return $S \to V_1 \to V_4 \to T$ as an optimal solution with cost $= 2 + 2 + 3 = 7$.



Discussion of A^* algorithm

Question:

Can we consider the A^* algorithm as a special kind of branch and bound strategy in which the cost function is cleverly designed?

- The answer is yes.
- When the A^* algorithm stops (a goal node is selected), all of the other expanded nodes are simultaneously bounded by the found feasible solution corresponding to the goal node.

Linear block code decoding problem

- Suppose we use binary codes to send 8 numbers from 0 to 7.
- We then need 3 bits for each number.
- Example 0 is sent by 000 and 4 is sent by 100.
- The problem is that if there is any error, the received signal will be decoded wrongly.
- Example If 100 is sent and received as 000, then it will cause an error, since the received signal will be decoded as 0, instead of the original sent number 4.

Linear block code decoding problem (cont'd)

Code words

- Below, we use 6 bits, instead of 3 bits, to code numbers.

A table of code words for numbers 1–7:

Number	Code word	Number	Code word
000 (0)	000000	100 (4)	100110
010 (2)	010101	001 (1)	001011
110 (6)	110011	101 (5)	101101
011 (3)	011110	111 (7)	111000

- Each number is now sent by its code word.
- Example We send 100110 for number 4, instead of 100.

Linear block code decoding problem (cont'd)

- The advantage is that we decode a received vector to a code word whose Hamming distance is the smallest among all code words.

Example:

- Suppose that the code word 000000 is sent as 000001.
- Then we can see that the Hamming distance between 000000 and 000001 is the smallest.
- Hence, the decoding process will decode 000001 as 000000.
- We can tolerate more errors by enlarging the number of digits.

Linear block code decoding problem (cont'd)

- Assume that 1 is sent as -1 and 0 is sent as 1.
- Let $c = (c_1, \ldots, c_n)$ be a code word and $r = (r_1, \ldots, r_n)$ be a received vector.
- The distance between r and c is then defined as:

$$d(r,c) = \sum_{i=1}^{n} (r_i - (-1)^{c_i})^2$$

Examples:

- If $c = 111000$ and $r = (-1,-1,-1,1,1,1)$, $d(r,c) = 0$.
- If $c = 111000$ and $r = (-2,-2,-2,-1,-1,0)$, $d(r,c) = 12$.

Linear block code decoding problem (cont'd)

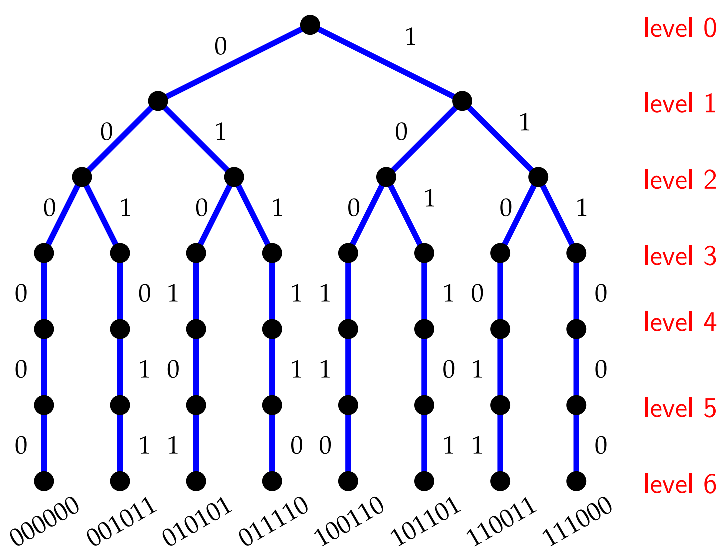
A simple method to decode a received vector:

1. Calculate the distance between the received vector and all code words.
2. Decode this received vector as the code word with the smallest distance.

- In fact, this exhaustive searching through all of the code words is not practical, because in practice, the number of code words is more than 10^7, which is extremely large.
- We can use A^* algorithm to efficiently conquer this problem by a code tree that represents all the code words.

Linear block code decoding problem (cont'd)

Code tree



Linear block code decoding problem (cont'd)

- Decoding a received vector (finding a code word closest to the received vector) now becomes a tree searching problem defined as follows.

Tree searching problem for decoding a received vector:

Find the path from the root of code tree to a goal node such that the cost of the path is minimum among all paths from the root to a goal node.

- The cost of a path is the summation of the costs of branches in the path.
- The cost of the branch from a node at level $t-1$ to level t is $(r_t - (-1)^{c_t})^2$.

Linear block code decoding problem (cont'd)

- Let the level of the root of the code tree be 0.
- Let x be a node at level t.
- The function $g(x)$ is defined as:

$$g(x) = \sum_{i=1}^{t}(r_i - (-1)^{c_i})^2$$

where $c_1, c_2, \ldots, c_t$ are the labels of branches associated with the path from the root to node x.

Linear block code decoding problem (cont'd)

- Define $h(x)$ as follows:

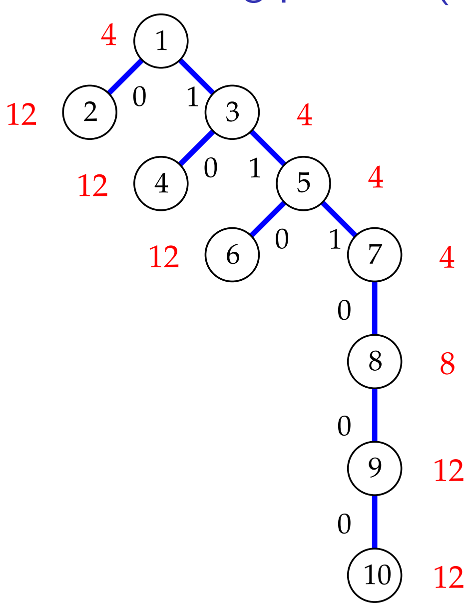
$$h(x) = \sum_{i=t+1}^{n}(|r_i| - 1)^2$$

- Then $h(x) \leq h^*(x)$ for every node x.

Example:

- Let $(-2, -2, -2, -1, -1, 0)$ be the received vector.
- Its decoding process by the A^* algorithm is illustrated on the next slide.
- When node 10 (a goal node) is selected to expand, the process is terminated and the closest code word is 111000.

Linear block code decoding problem (cont'd)



Linear block code decoding problem (cont'd)

- Recall that the received vector is $(-2, -2, -2, -1, -1, 0)$.
- The value of $f(1)$ is calculated as follows.

$$\begin{aligned} f(1) &= g(1) + h(1) \\ &= 0 + \sum_{i=1}^{6}(|r_i| - 1)^2 \\ &= 0 + (1 + 1 + 1 + 0 + 0 + 1) \\ &= 4 \end{aligned}$$

- The value of $f(2)$ is calculated as follows.

$$\begin{aligned} f(2) &= g(2) + h(2) \\ &= (-2 - (-1)^0)^2 + \sum_{i=2}^{6}(|r_i| - 1)^2 \\ &= 9 + (1 + 1 + 0 + 0 + 1) \\ &= 12 \end{aligned}$$