Branch-and-Bound

Branch-and-Bound

- refers to all state space search methods in which all children of the E node are generated before any other live node can become the E node

Branch-and-Bound

- refers to all state space search methods in which all children of the E node are generated before any other live node can become the E node
- against the methods (BFS and D-search) in which the exploration of a new node cannot be begin until the node currently being explored is fully explored

Branch-and-Bound

- refers to all state space search methods in which all children of the E node are generated before any other live node can become the E node
- against the methods (BFS and D-search) in which the exploration of a new node cannot be begin until the node currently being explored is fully explored
- in branch and bound terminology,

Branch-and-Bound

- refers to all state space search methods in which all children of the E node are generated before any other live node can become the E node
- against the methods (BFS and D-search) in which the exploration of a new node cannot be begin until the node currently being explored is fully explored
- in branch and bound terminology,
- BFS-like method is called FIFO search as the list of live nodes is a first in first out list (queue)

Branch-and-Bound

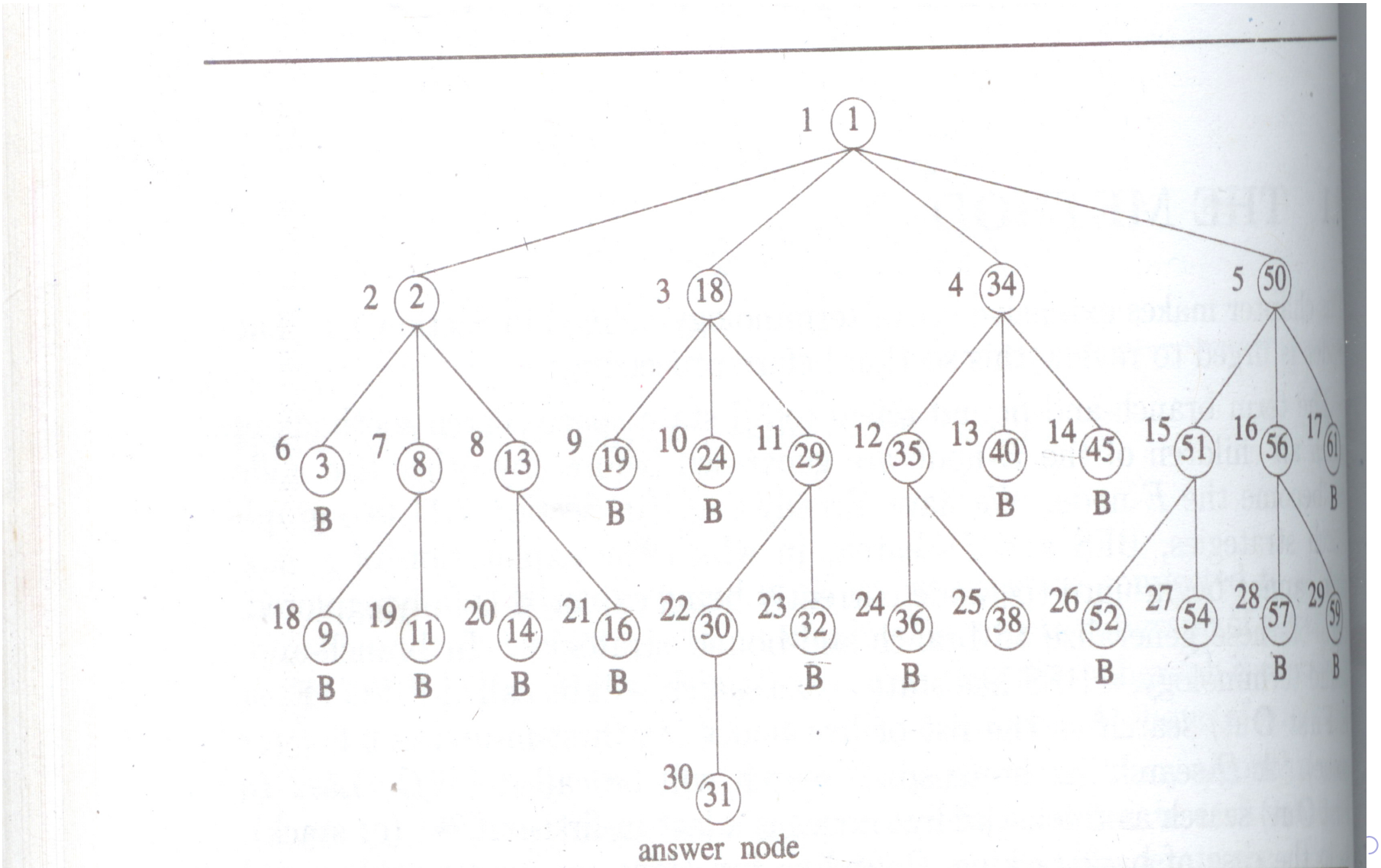
- refers to all state space search methods in which all children of the E node are generated before any other live node can become the E node
- against the methods (BFS and D-search) in which the exploration of a new node cannot be begin until the node currently being explored is fully explored
- in branch and bound terminology,
- BFS-like method is called FIFO search as the list of live nodes is a first in first out list (queue)
- D-search like state space search is called LIFO search as the list of live nodes is a last in first out list (stack)

Branch-and-Bound

- refers to all state space search methods in which all children of the E node are generated before any other live node can become the E node
- against the methods (BFS and D-search) in which the exploration of a new node cannot be begin until the node currently being explored is fully explored
- in branch and bound terminology,
- BFS-like method is called FIFO search as the list of live nodes is a first in first out list (queue)
- D-search like state space search is called LIFO search as the list of live nodes is a last in first out list (stack)
- bounding functions are used to avoid the generation of subtree that do not contain an answer node

4-queen problem

- portion of 4-queens state space tree generated by FIFO branch-and-bound



4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated
- these nodes represent a chessboard with queen 1 in row 1 and columns 1, 2, 3, and 4 respectively

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated
- these nodes represent a chessboard with queen 1 in row 1 and columns 1, 2, 3, and 4 respectively
- only live nodes now are nodes 2, 18, 34, and 50

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated
- these nodes represent a chessboard with queen 1 in row 1 and columns 1, 2, 3, and 4 respectively
- only live nodes now are nodes 2, 18, 34, and 50
- if the nodes are generated in this order then next E node is node 2

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated
- these nodes represent a chessboard with queen 1 in row 1 and columns 1, 2, 3, and 4 respectively
- only live nodes now are nodes 2, 18, 34, and 50
- if the nodes are generated in this order then next E node is node 2
- it is expanded and nodes 3, 8, and 13 are generated

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated
- these nodes represent a chessboard with queen 1 in row 1 and columns 1, 2, 3, and 4 respectively
- only live nodes now are nodes 2, 18, 34, and 50
- if the nodes are generated in this order then next E node is node 2
- it is expanded and nodes 3, 8, and 13 are generated
- node 3 is killed using bounding function and nodes 8 and 13 are added to queue of live nodes

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated
- these nodes represent a chessboard with queen 1 in row 1 and columns 1, 2, 3, and 4 respectively
- only live nodes now are nodes 2, 18, 34, and 50
- if the nodes are generated in this order then next E node is node 2
- it is expanded and nodes 3, 8, and 13 are generated
- node 3 is killed using bounding function and nodes 8 and 13 are added to queue of live nodes
- node 18 becomes the next E node and nodes 19, 24, and 29 are generated

4-queen problem

- initially, only one live node, node 1 - no queen has been placed on the chessboard
- this node becomes E node and it is expanded and its children nodes 2, 18, 34, and 50 are generated
- these nodes represent a chessboard with queen 1 in row 1 and columns 1, 2, 3, and 4 respectively
- only live nodes now are nodes 2, 18, 34, and 50
- if the nodes are generated in this order then next E node is node 2
- it is expanded and nodes 3, 8, and 13 are generated
- node 3 is killed using bounding function and nodes 8 and 13 are added to queue of live nodes
- node 18 becomes the next E node and nodes 19, 24, and 29 are generated
- nodes 19 and 24 are killed and node 29 is added to queue of live nodes and the E node is node 34

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind
- the selection rule for the next E node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind
- the selection rule for the next E node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly
- the search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind
- the selection rule for the next E node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly
- the search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes
- the next E node is selected on the basis of this ranking function

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind
- the selection rule for the next E node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly
- the search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes
- the next E node is selected on the basis of this ranking function
- ideal way to assign ranks based on the additional computational effort (cost) needed to reach an answer node from the live node

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind
- the selection rule for the next E node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly
- the search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes
- the next E node is selected on the basis of this ranking function
- ideal way to assign ranks based on the additional computational effort (cost) needed to reach an answer node from the live node
- for any node x this cost could be

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind
- the selection rule for the next E node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly
- the search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes
- the next E node is selected on the basis of this ranking function
- ideal way to assign ranks based on the additional computational effort (cost) needed to reach an answer node from the live node
- for any node x this cost could be
- the number of nodes in the subtree x that need to be generated before an answer node is generated

Least cost (LC) search

- in both FIFO and LIFO branch and bound the selection rule for the next E node is rather rigid and in a sense blind
- the selection rule for the next E node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly
- the search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes
- the next E node is selected on the basis of this ranking function
- ideal way to assign ranks based on the additional computational effort (cost) needed to reach an answer node from the live node
- for any node x this cost could be
- the number of nodes in the subtree x that need to be generated before an answer node is generated
- the number of levels the nearest answer node is from x

LC search

- difficulty is to compute the cost of a node usually involves a search of the subtree x for an answer node

LC search

- difficulty is to compute the cost of a node usually involves a search of the subtree x for an answer node
- by the time of the cost of a node is determined, that subtree has been searched and there is no need to explore x again

LC search

- difficulty is to compute the cost of a node usually involves a search of the subtree x for an answer node
- by the time of the cost of a node is determined, that subtree has been searched and there is no need to explore x again
- search algorithms rank nodes only on the basis of an estimate $\hat{g}(\cdot)$ of their cost

LC search

- difficulty is to compute the cost of a node usually involves a search of the subtree x for an answer node
- by the time of the cost of a node is determined, that subtree has been searched and there is no need to explore x again
- search algorithms rank nodes only on the basis of an estimate $\hat{g}(\cdot)$ of their cost
- let, $\hat{g}(x)$ be an estimate of the additional effort needed to reach an answer node from x

LC search

- difficulty is to compute the cost of a node usually involves a search of the subtree x for an answer node
- by the time of the cost of a node is determined, that subtree has been searched and there is no need to explore x again
- search algorithms rank nodes only on the basis of an estimate $\hat{g}(\cdot)$ of their cost
- let, $\hat{g}(x)$ be an estimate of the additional effort needed to reach an answer node from x
- node x is assigned a rank using a function $\hat{c}(\cdot)$ such that $\hat{c}(x) = f(h(x)) + \hat{g}(x)$

LC search

- difficulty is to compute the cost of a node usually involves a search of the subtree x for an answer node
- by the time of the cost of a node is determined, that subtree has been searched and there is no need to explore x again
- search algorithms rank nodes only on the basis of an estimate $\hat{g}(\cdot)$ of their cost
- let, $\hat{g}(x)$ be an estimate of the additional effort needed to reach an answer node from x
- node x is assigned a rank using a function $\hat{c}(\cdot)$ such that $\hat{c}(x) = f(h(x)) + \hat{g}(x)$
- $h(x)$ is the cost of reaching x from the root and $f(\cdot)$ is any nondecreasing function

LC search

- difficulty is to compute the cost of a node usually involves a search of the subtree x for an answer node
- by the time of the cost of a node is determined, that subtree has been searched and there is no need to explore x again
- search algorithms rank nodes only on the basis of an estimate $\hat{g}(\cdot)$ of their cost
- let, $\hat{g}(x)$ be an estimate of the additional effort needed to reach an answer node from x
- node x is assigned a rank using a function $\hat{c}(\cdot)$ such that $\hat{c}(x) = f(h(x)) + \hat{g}(x)$
- $h(x)$ is the cost of reaching x from the root and $f(\cdot)$ is any nondecreasing function
- the effort already expended in reaching the live nodes cannot be reduced and concerned is to minimize the additional effort to find an answer node

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered
- biases the search algorithm to make deep probes into search tree

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered
- biases the search algorithm to make deep probes into search tree
- to see this, normally expect $\hat{g}(y) \leq \hat{g}(x)$ for y, a child of x

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered
- biases the search algorithm to make deep probes into search tree
- to see this, normally expect $\hat{g}(y) \leq \hat{g}(x)$ for y, a child of x
- following x, y will become the E node, then one of y's children will become E node and so on

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered
- biases the search algorithm to make deep probes into search tree
- to see this, normally expect $\hat{g}(y) \leq \hat{g}(x)$ for y, a child of x
- following x, y will become the E node, then one of y's children will become E node and so on
- nodes in subtrees other than the subtree x will not get generated until the subtree x is fully searched

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered
- biases the search algorithm to make deep probes into search tree
- to see this, normally expect $\hat{g}(y) \leq \hat{g}(x)$ for y, a child of x
- following x, y will become the E node, then one of y's children will become E node and so on
- nodes in subtrees other than the subtree x will not get generated until the subtree x is fully searched
- if $\hat{g}(x)$ were the true cost of x, no need to explore the remaining subtree $\implies$ x guaranteed to get to an answer node

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered
- biases the search algorithm to make deep probes into search tree
- to see this, normally expect $\hat{g}(y) \leq \hat{g}(x)$ for y, a child of x
- following x, y will become the E node, then one of y's children will become E node and so on
- nodes in subtrees other than the subtree x will not get generated until the subtree x is fully searched
- if $\hat{g}(x)$ were the true cost of x, no need to explore the remaining subtree $\implies$ x guaranteed to get to an answer node
- say, w and z nodes, $\hat{g}(w) < \hat{g}(z)$ and z is much close to an answer node than w

LC search

- $f(\cdot) \equiv 0 \implies$ the effort already expended need not be considered
- biases the search algorithm to make deep probes into search tree
- to see this, normally expect $\hat{g}(y) \leq \hat{g}(x)$ for y, a child of x
- following x, y will become the E node, then one of y's children will become E node and so on
- nodes in subtrees other than the subtree x will not get generated until the subtree x is fully searched
- if $\hat{g}(x)$ were the true cost of x, no need to explore the remaining subtree $\implies$ x guaranteed to get to an answer node
- say, w and z nodes, $\hat{g}(w) < \hat{g}(z)$ and z is much close to an answer node than w
- desirable not to overbias the search algorithm in favor of deep probes

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree
- search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E node

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree
- search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E node
- choose for its next E node a live node with least $\hat{c}(\cdot)$

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree
- search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E node
- choose for its next E node a live node with least $\hat{c}(\cdot)$
- a search strategy is called an LC search

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree
- search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E node
- choose for its next E node a live node with least $\hat{c}(\cdot)$
- a search strategy is called an LC search
- BFS and D-search are special cases of LC-search

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree
- search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E node
- choose for its next E node a live node with least $\hat{c}(\cdot)$
- a search strategy is called an LC search
- BFS and D-search are special cases of LC-search
- if $\hat{g}(x) = 0$ and $f(h(x)) =$ level of node x, then a LC search generates nodes by levels; same as a BFS

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree
- search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E node
- choose for its next E node a live node with least $\hat{c}(\cdot)$
- a search strategy is called an LC search
- BFS and D-search are special cases of LC-search
- if $\hat{g}(x) = 0$ and $f(h(x)) =$ level of node x, then a LC search generates nodes by levels; same as a BFS
- if $f(h(x)) = 0$ and $\hat{g}(x) \geq \hat{g}(y)$ whenever y is a child of x then the search is D search

LC search

- by using $f(\cdot) \neq 0$ the search algorithm can be forced to favor a node z close to the root over a node w which is many levels below z
- it would reduce possibility of deep and fruitless searches into tree
- search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E node
- choose for its next E node a live node with least $\hat{c}(\cdot)$
- a search strategy is called an LC search
- BFS and D-search are special cases of LC-search
- if $\hat{g}(x) = 0$ and $f(h(x)) =$ level of node x, then a LC search generates nodes by levels; same as a BFS
- if $f(h(x)) = 0$ and $\hat{g}(x) \geq \hat{g}(y)$ whenever y is a child of x then the search is D search
- LC search coupled with bounding functions is called an LC branch-and-bound search

LC search

- cost function $c(\cdot)$ for LC search

LC search

- cost function $c(\cdot)$ for LC search
- if x is an answer node, then $c(x)$ is the cost of reaching x from the root of the state space tree

LC search

- cost function $c(\cdot)$ for LC search
- if x is an answer node, then $c(x)$ is the cost of reaching x from the root of the state space tree
- if x is not an answer node, then $c(x) = \infty$ providing the subtree x contains no answer node; otherwise $c(x)$ equals the cost of minimum cost answer node in the subtree x

LC search

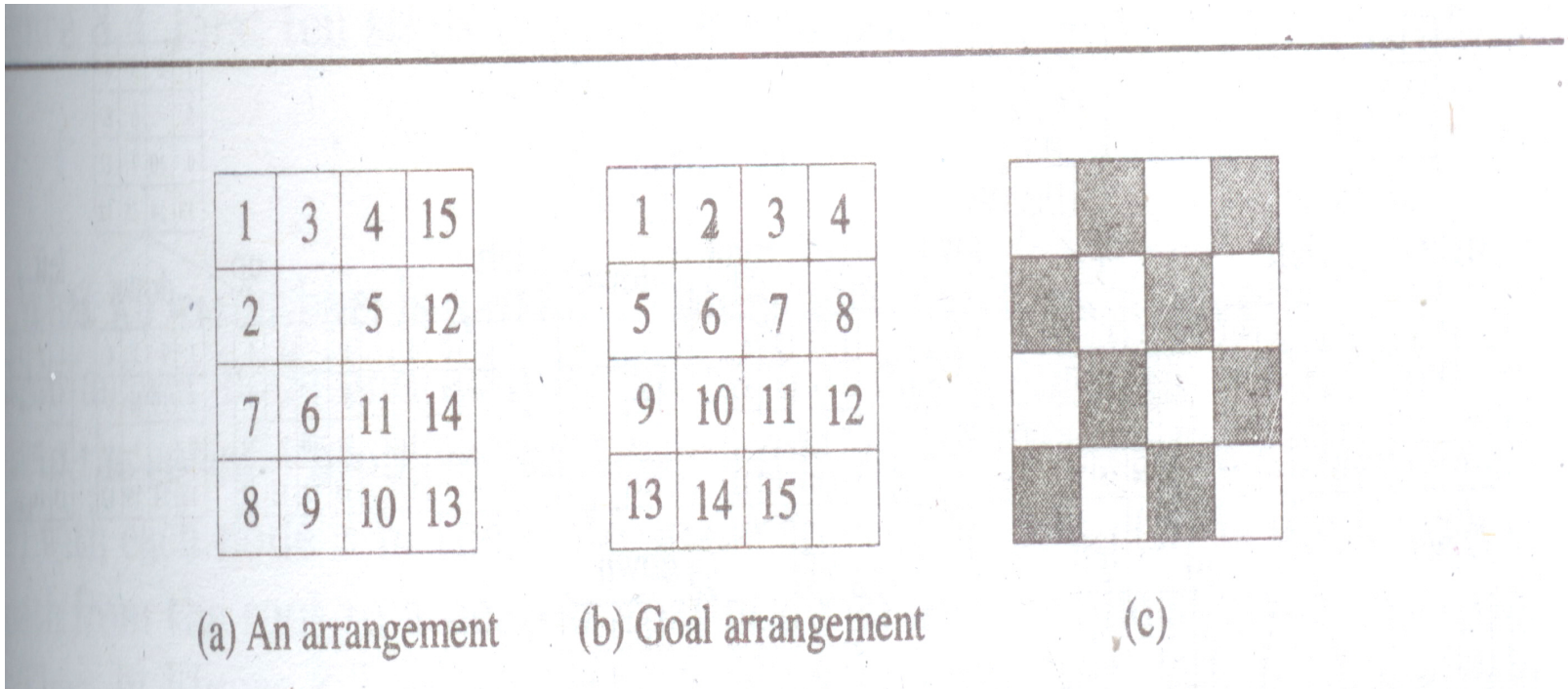
- cost function $c(\cdot)$ for LC search
- if x is an answer node, then $c(x)$ is the cost of reaching x from the root of the state space tree
- if x is not an answer node, then $c(x) = \infty$ providing the subtree x contains no answer node; otherwise $c(x)$ equals the cost of minimum cost answer node in the subtree x
- 15 puzzle example

LC search

- cost function $c(\cdot)$ for LC search
- if x is an answer node, then $c(x)$ is the cost of reaching x from the root of the state space tree
- if x is not an answer node, then $c(x) = \infty$ providing the subtree x contains no answer node; otherwise $c(x)$ equals the cost of minimum cost answer node in the subtree x
- 15 puzzle example
- 15 numbered tiles on a square frame with a capacity of 16 tiles

LC search

- cost function $c(\cdot)$ for LC search
- if x is an answer node, then $c(x)$ is the cost of reaching x from the root of the state space tree
- if x is not an answer node, then $c(x) = \infty$ providing the subtree x contains no answer node; otherwise $c(x)$ equals the cost of minimum cost answer node in the subtree x
- 15 puzzle example
- 15 numbered tiles on a square frame with a capacity of 16 tiles



15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move

15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move
- legal move - tile adjacent to the empty spot (ES) is moved to ES

15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move
- legal move - tile adjacent to the empty spot (ES) is moved to ES
- each move creates a new arrangement of the tiles

15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move
- legal move - tile adjacent to the empty spot (ES) is moved to ES
- each move creates a new arrangement of the tiles
- these arrangements are called the states of the puzzle

15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move
- legal move - tile adjacent to the empty spot (ES) is moved to ES
- each move creates a new arrangement of the tiles
- these arrangements are called the states of the puzzle
- initial and goal arrangements are initial and goal states

15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move
- legal move - tile adjacent to the empty spot (ES) is moved to ES
- each move creates a new arrangement of the tiles
- these arrangements are called the states of the puzzle
- initial and goal arrangements are initial and goal states
- a state is reachable from the initial state iff there is a sequence of legal moves from the initial state to this state

15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move
- legal move - tile adjacent to the empty spot (ES) is moved to ES
- each move creates a new arrangement of the tiles
- these arrangements are called the states of the puzzle
- initial and goal arrangements are initial and goal states
- a state is reachable from the initial state iff there is a sequence of legal moves from the initial state to this state
- the state space of an initial state consists of all states that can be reached from the initial state

15-puzzle Example

- given an initial arrangement of the tiles - transform into goal arrangement through a series of legal move
- legal move - tile adjacent to the empty spot (ES) is moved to ES
- each move creates a new arrangement of the tiles
- these arrangements are called the states of the puzzle
- initial and goal arrangements are initial and goal states
- a state is reachable from the initial state iff there is a sequence of legal moves from the initial state to this state
- the state space of an initial state consists of all states that can be reached from the initial state
- solution - search the state space for the goal state and use the path from the initial state to the goal state

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame
- given initial state only one-half are reachable

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame
- given initial state only one-half are reachable
- state space for the problem is very large

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame
- given initial state only one-half are reachable
- state space for the problem is very large
- before attempting to search this state space for the goal state, it is worthwhile to determine whether the goal state is reachable from the initial state

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame
- given initial state only one-half are reachable
- state space for the problem is very large
- before attempting to search this state space for the goal state, it is worthwhile to determine whether the goal state is reachable from the initial state
- number the frame positions 1 to 16

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame
- given initial state only one-half are reachable
- state space for the problem is very large
- before attempting to search this state space for the goal state, it is worthwhile to determine whether the goal state is reachable from the initial state
- number the frame positions 1 to 16
- position i is the frame position containing tile numbered i in the goal arrangement

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame
- given initial state only one-half are reachable
- state space for the problem is very large
- before attempting to search this state space for the goal state, it is worthwhile to determine whether the goal state is reachable from the initial state
- number the frame positions 1 to 16
- position i is the frame position containing tile numbered i in the goal arrangement
- $position(i)$ be the position number in the initial state of the tile numbered i

15-puzzle Example

- there are 16! ($\approx 20.9 \times 10^{12}$) different arrangements of the tiles on the frame
- given initial state only one-half are reachable
- state space for the problem is very large
- before attempting to search this state space for the goal state, it is worthwhile to determine whether the goal state is reachable from the initial state
- number the frame positions 1 to 16
- position i is the frame position containing tile numbered i in the goal arrangement
- $position(i)$ be the position number in the initial state of the tile numbered i
- given figure position 16 is the empty spot and $position(16)$ will denote the position of the empty spot

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$
- let $x = 1$ if in the initial state the empty spot is at one of the shaded positions

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$
- let $x = 1$ if in the initial state the empty spot is at one of the shaded positions
- $x = 0$ if it is at one of the remaining positions

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$
- let $x = 1$ if in the initial state the empty spot is at one of the shaded positions
- $x = 0$ if it is at one of the remaining positions
- theorem: the goal state is reachable from the initial state iff $\sum_{i=1}^{16} less(i) + x$ is even

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$
- let $x = 1$ if in the initial state the empty spot is at one of the shaded positions
- $x = 0$ if it is at one of the remaining positions
- theorem: the goal state is reachable from the initial state iff $\sum_{i=1}^{16} less(i) + x$ is even
- determine whether the goal state is in the state space of the initial state

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$
- let $x = 1$ if in the initial state the empty spot is at one of the shaded positions
- $x = 0$ if it is at one of the remaining positions
- theorem: the goal state is reachable from the initial state iff $\sum_{i=1}^{16} less(i) + x$ is even
- determine whether the goal state is in the state space of the initial state
- if it is, proceed to determine a sequence of moves leading to the goal state

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$
- let $x = 1$ if in the initial state the empty spot is at one of the shaded positions
- $x = 0$ if it is at one of the remaining positions
- theorem: the goal state is reachable from the initial state iff $\sum_{i=1}^{16} less(i) + x$ is even
- determine whether the goal state is in the state space of the initial state
- if it is, proceed to determine a sequence of moves leading to the goal state
- to search this, the state space organized into tree

15-puzzle Example

- for any state, $less(i)$ be the number of tiles j such that $j < i$ and $position(j) > position(i)$
- $less(1) = 0$ $less(4) = 1$ $less(12) = 6$
- let $x = 1$ if in the initial state the empty spot is at one of the shaded positions
- $x = 0$ if it is at one of the remaining positions
- theorem: the goal state is reachable from the initial state iff $\sum_{i=1}^{16} less(i) + x$ is even
- determine whether the goal state is in the state space of the initial state
- if it is, proceed to determine a sequence of moves leading to the goal state
- to search this, the state space organized into tree
- children of each node x in this tree represent states reachable from state x by one legal move

15-puzzle Example

- the empty space moves either up, right, down or left

15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent

15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent
- search of state space is blind

15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent
- search of state space is blind
- what the initial configuration, the algorithm attempts to make the same sequences of moves

15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent
- search of state space is blind
- what the initial configuration, the algorithm attempts to make the same sequences of moves
- FIFO search always generates the state space tree by levels

15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent
- search of state space is blind
- what the initial configuration, the algorithm attempts to make the same sequences of moves
- FIFO search always generates the state space tree by levels
- intelligent search method - seeks out an answer node and adapts the path in takes through the state space tree to the specific problem instance being solved

15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent
- search of state space is blind
- what the initial configuration, the algorithm attempts to make the same sequences of moves
- FIFO search always generates the state space tree by levels
- intelligent search method - seeks out an answer node and adapts the path in takes through the state space tree to the specific problem instance being solved
- associate cost $c(x)$ with each node x in the state space tree

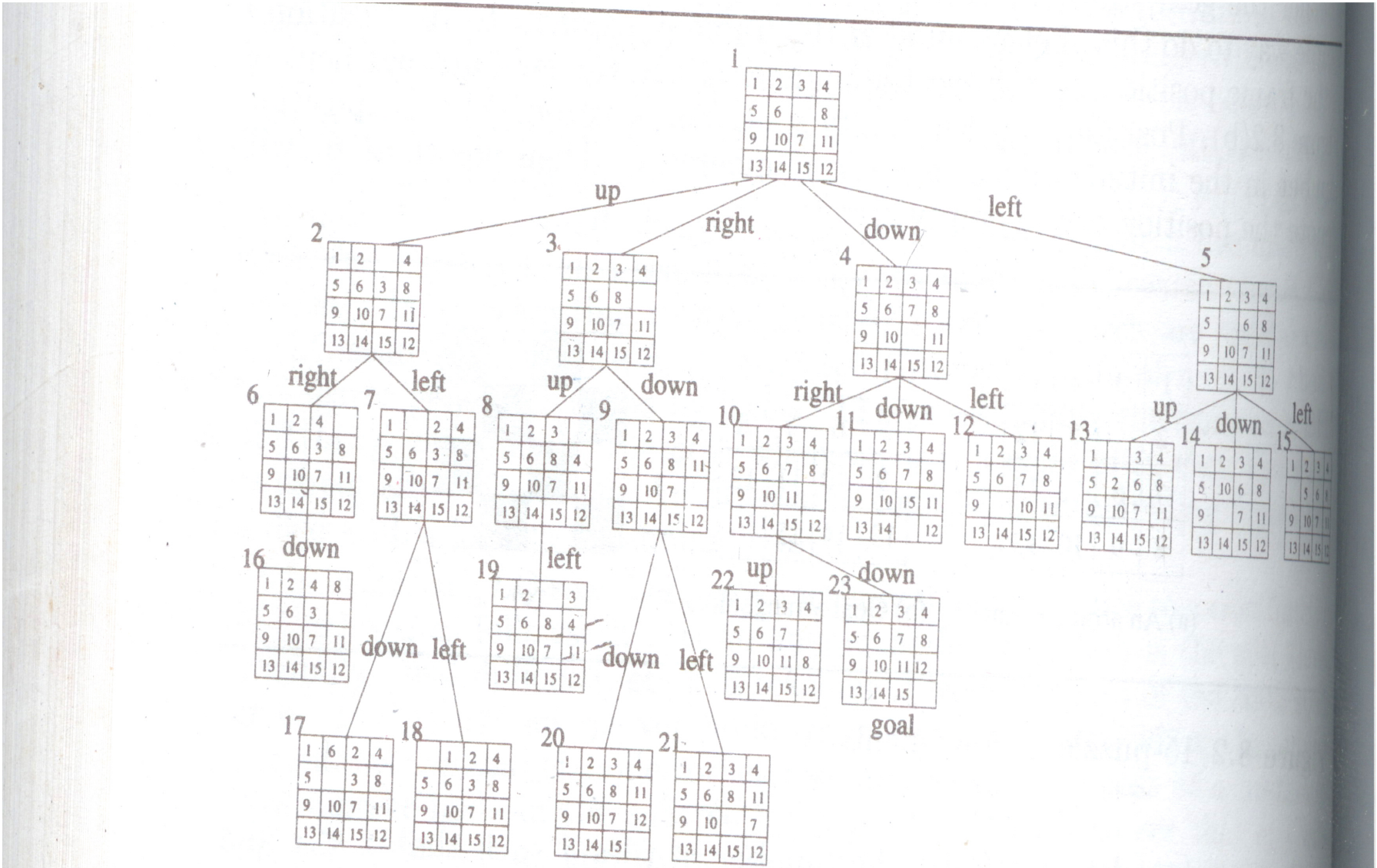
15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent
- search of state space is blind
- what the initial configuration, the algorithm attempts to make the same sequences of moves
- FIFO search always generates the state space tree by levels
- intelligent search method - seeks out an answer node and adapts the path in takes through the state space tree to the specific problem instance being solved
- associate cost $c(x)$ with each node x in the state space tree
- cost $c(x)$ is the length of a path from the root to a nearest goal node, if any, in the subtree with root x

15-puzzle Example

- the empty space moves either up, right, down or left
- no node p has a child state that is the same as p's parent
- search of state space is blind
- what the initial configuration, the algorithm attempts to make the same sequences of moves
- FIFO search always generates the state space tree by levels
- intelligent search method - seeks out an answer node and adapts the path in takes through the state space tree to the specific problem instance being solved
- associate cost $c(x)$ with each node x in the state space tree
- cost $c(x)$ is the length of a path from the root to a nearest goal node, if any, in the subtree with root x
- begin with the root as E node and generate a child node with $c()$ value the same as root

15-puzzle Example



Edges are labeled according to the direction in which the empty space moves

15-puzzle Example

- $c(1) = c(4) = c(10) = c(23) = 3$

15-puzzle Example

- $c(1) = c(4) = c(10) = c(23) = 3$
- children nodes 2, 3, and 5 are eliminated and node 4 becomes a live node, becomes next E node

15-puzzle Example

- $c(1) = c(4) = c(10) = c(23) = 3$
- children nodes 2, 3, and 5 are eliminated and node 4 becomes a live node, becomes next E node
- first child is node 10 with $c(10) = c(4) = 3$

15-puzzle Example

- $c(1) = c(4) = c(10) = c(23) = 3$
- children nodes 2, 3, and 5 are eliminated and node 4 becomes a live node, becomes next E node
- first child is node 10 with $c(10) = c(4) = 3$
- remaining children are not generated
- node 4 dies and node 10 becomes E node

15-puzzle Example

- $c(1) = c(4) = c(10) = c(23) = 3$
- children nodes 2, 3, and 5 are eliminated and node 4 becomes a live node, becomes next E node
- first child is node 10 with $c(10) = c(4) = 3$
- remaining children are not generated
- node 4 dies and node 10 becomes E node
- node 22 killed immediately as $c(22) > 3$ and node 23 is generated next

15-puzzle Example

- $c(1) = c(4) = c(10) = c(23) = 3$
- children nodes 2, 3, and 5 are eliminated and node 4 becomes a live node, becomes next E node
- first child is node 10 with $c(10) = c(4) = 3$
- remaining children are not generated
- node 4 dies and node 10 becomes E node
- node 22 killed immediately as $c(22) > 3$ and node 23 is generated next
- it is a goal node and search terminates

15-puzzle Example

- $c(1) = c(4) = c(10) = c(23) = 3$
- children nodes 2, 3, and 5 are eliminated and node 4 becomes a live node, becomes next E node
- first child is node 10 with $c(10) = c(4) = 3$
- remaining children are not generated
- node 4 dies and node 10 becomes E node
- node 22 killed immediately as $c(22) > 3$ and node 23 is generated next
- it is a goal node and search terminates
- here, nodes to become E nodes are nodes on the path from the root to a nearest goal node
- impractical as not possible to computer $c(\cdot)$ specified here

15-puzzle Example

- possible to compute estimate $\hat{c}(x) = f(x) + \hat{g}(x)$ of $c(x)$

15-puzzle Example

- possible to compute estimate $\hat{c}(x) = f(x) + \hat{g}(x)$ of $c(x)$
- $f(x)$ is the length of the path from the root to node x and $\hat{g}(x)$ is an estimate of the length of a shortest path from x to a goal node

15-puzzle Example

- possible to compute estimate $\hat{c}(x) = f(x) + \hat{g}(x)$ of $c(x)$
- $f(x)$ is the length of the path from the root to node x and $\hat{g}(x)$ is an estimate of the length of a shortest path from x to a goal node
- $\hat{g}(x)$ may be number of nonblank tiles not in their goal position

15-puzzle Example

- possible to compute estimate $\hat{c}(x) = f(x) + \hat{g}(x)$ of $c(x)$
- $f(x)$ is the length of the path from the root to node x and $\hat{g}(x)$ is an estimate of the length of a shortest path from x to a goal node
- $\hat{g}(x)$ may be number of nonblank tiles not in their goal position
- atleast $\hat{g}(x)$ moves have to be made to transform state x to a goal state

15-puzzle Example

- possible to compute estimate $\hat{c}(x) = f(x) + \hat{g}(x)$ of $c(x)$
- $f(x)$ is the length of the path from the root to node x and $\hat{g}(x)$ is an estimate of the length of a shortest path from x to a goal node
- $\hat{g}(x)$ may be number of nonblank tiles not in their goal position
- atleast $\hat{g}(x)$ moves have to be made to transform state x to a goal state
- more than $\hat{g}(x)$ moves may be needed to achieve this

1	2	3	4
5	6		8
9	10	11	12
13	14	15	7

- $\hat{g}(x) = 1$ only tile 7 not in its final position

15-puzzle Example

- possible to compute estimate $\hat{c}(x) = f(x) + \hat{g}(x)$ of $c(x)$
- $f(x)$ is the length of the path from the root to node x and $\hat{g}(x)$ is an estimate of the length of a shortest path from x to a goal node
- $\hat{g}(x)$ may be number of nonblank tiles not in their goal position
- atleast $\hat{g}(x)$ moves have to be made to transform state x to a goal state
- more than $\hat{g}(x)$ moves may be needed to achieve this

1	2	3	4
5	6		8
9	10	11	12
13	14	15	7

- $\hat{g}(x) = 1$ only tile 7 not in its final position
- $\hat{c}(x)$ is a lower bound on the value of $c(x)$

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$
- $\hat{c}(2) = 1 + 4$ $\hat{c}(3) = 1 + 4$ $\hat{c}(4) = 1 + 2$ $\hat{c}(5) = 1 + 4$

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$
- $\hat{c}(2) = 1 + 4$ $\hat{c}(3) = 1 + 4$ $\hat{c}(4) = 1 + 2$ $\hat{c}(5) = 1 + 4$
- node 4 becomes E node and its children are generated

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$
- $\hat{c}(2) = 1 + 4$ $\hat{c}(3) = 1 + 4$ $\hat{c}(4) = 1 + 2$ $\hat{c}(5) = 1 + 4$
- node 4 becomes E node and its children are generated
- live nodes at this time 2, 3, 5, 10, 11, and 12

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$
- $\hat{c}(2) = 1 + 4$ $\hat{c}(3) = 1 + 4$ $\hat{c}(4) = 1 + 2$ $\hat{c}(5) = 1 + 4$
- node 4 becomes E node and its children are generated
- live nodes at this time 2, 3, 5, 10, 11, and 12
- $\hat{c}(10) = 2 + 1$ $\hat{c}(11) = 2 + 3$ $\hat{c}(12) = 2 + 3$

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$
- $\hat{c}(2) = 1 + 4$ $\hat{c}(3) = 1 + 4$ $\hat{c}(4) = 1 + 2$ $\hat{c}(5) = 1 + 4$
- node 4 becomes E node and its children are generated
- live nodes at this time 2, 3, 5, 10, 11, and 12
- $\hat{c}(10) = 2 + 1$ $\hat{c}(11) = 2 + 3$ $\hat{c}(12) = 2 + 3$
- live node with least $\hat{c}$ is node 10 and it becomes next E node
- nodes 22 and 23 are generated next

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$
- $\hat{c}(2) = 1 + 4$ $\hat{c}(3) = 1 + 4$ $\hat{c}(4) = 1 + 2$ $\hat{c}(5) = 1 + 4$
- node 4 becomes E node and its children are generated
- live nodes at this time 2, 3, 5, 10, 11, and 12
- $\hat{c}(10) = 2 + 1$ $\hat{c}(11) = 2 + 3$ $\hat{c}(12) = 2 + 3$
- live node with least $\hat{c}$ is node 10 and it becomes next E node
- nodes 22 and 23 are generated next
- node 23 is determined to be a goal node and search terminates

15-puzzle Example

- LC-search begins using node 1 as the E node and all children are generated
- node 1 dies and leaves live nodes 2, 3, 4, and 5
- the next node to become E node is a live node with least $\hat{c}(x)$
- $\hat{c}(2) = 1 + 4$ $\hat{c}(3) = 1 + 4$ $\hat{c}(4) = 1 + 2$ $\hat{c}(5) = 1 + 4$
- node 4 becomes E node and its children are generated
- live nodes at this time 2, 3, 5, 10, 11, and 12
- $\hat{c}(10) = 2 + 1$ $\hat{c}(11) = 2 + 3$ $\hat{c}(12) = 2 + 3$
- live node with least $\hat{c}$ is node 10 and it becomes next E node
- nodes 22 and 23 are generated next
- node 23 is determined to be a goal node and search terminates
- suitable choice for $\hat{c}()$, LC search more selective than any of the other search methods

LC Search(t)

- listnode consists of point to next, its parent and cost associated

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return
3. $E = t$ // E node, initialize the list of live nodes to be empty

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return
3. $E = t$ // E node, initialize the list of live nodes to be empty
4. repeat $\{$
5. for each child x of E do

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return
3. $E = t$ // E node, initialize the list of live nodes to be empty
4. repeat $\{$
5. for each child x of E do
6. $\{$
7. if x is an answer node then output the path from x to t and return

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return
3. $E = t$ // E node, initialize the list of live nodes to be empty
4. repeat $\{$
5. for each child x of E do
6. $\{$
7. if x is an answer node then output the path from x to t and return
8. Add(x) // x is a new live node

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return
3. $E = t$ // E node, initialize the list of live nodes to be empty
4. repeat $\{$
5. for each child x of E do
6. $\{$
7. if x is an answer node then output the path from x to t and return
8. Add(x) // x is a new live node
9. $(x \rightarrow parent) = E$ // pointer for path to root
10. $\}$
11. if there are no more live nodes then write ("No answer node") return

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return
3. $E = t$ // E node, initialize the list of live nodes to be empty
4. repeat $\{$
5. for each child x of E do
6. $\{$
7. if x is an answer node then output the path from x to t and return
8. Add(x) // x is a new live node
9. $(x \rightarrow parent) = E$ // pointer for path to root
10. $\}$
11. if there are no more live nodes then write ("No answer node") return
12. $E = Least()$

LC Search(t)

- listnode consists of point to next, its parent and cost associated

1. // search t for an answer node
2. if t is an answer node then output t and return
3. $E = t$ // E node, initialize the list of live nodes to be empty
4. repeat {
5. for each child x of E do
6. {
7. if x is an answer node then output the path from x to t and return
8. Add(x) // x is a new live node
9. $(x \rightarrow parent) = E$ // pointer for path to root
10. }
11. if there are no more live nodes then write ("No answer node") return
12. $E = Least()$
13. } until (false)

Bounding

- LC, FIFO and LIFO search are essentially the same

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes
- differ only in the selection rule used to obtain the next E node

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes
- differ only in the selection rule used to obtain the next E node
- branch and bound method searches a state space tree using any search mechanism in which all the children of the E node are generated before another node becomes the E node

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes
- differ only in the selection rule used to obtain the next E node
- branch and bound method searches a state space tree using any search mechanism in which all the children of the E node are generated before another node becomes the E node
- each answer node x has a cost $c(x)$ associated with it

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes
- differ only in the selection rule used to obtain the next E node
- branch and bound method searches a state space tree using any search mechanism in which all the children of the E node are generated before another node becomes the E node
- each answer node x has a cost $c(x)$ associated with it
- minimum cost answer node is to be found

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes
- differ only in the selection rule used to obtain the next E node
- branch and bound method searches a state space tree using any search mechanism in which all the children of the E node are generated before another node becomes the E node
- each answer node x has a cost $c(x)$ associated with it
- minimum cost answer node is to be found
- cost function $\hat{c}(\cdot)$ such that $\hat{c}(x) \leq c(x)$ is used to provide lower bounds on solutions obtainable from any node x

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes
- differ only in the selection rule used to obtain the next E node
- branch and bound method searches a state space tree using any search mechanism in which all the children of the E node are generated before another node becomes the E node
- each answer node x has a cost $c(x)$ associated with it
- minimum cost answer node is to be found
- cost function $\hat{c}(\cdot)$ such that $\hat{c}(x) \leq c(x)$ is used to provide lower bounds on solutions obtainable from any node x
- if *upper* is an upper bound on the cost of a minimum cost solution, then all live nodes x with $\hat{c}(x) > upper$ may be killed as all answer nodes reachable from x have cost $c(x) \geq \hat{c}(x) > upper$

Bounding

- LC, FIFO and LIFO search are essentially the same
- difference in implementation of the list of live nodes
- differ only in the selection rule used to obtain the next E node
- branch and bound method searches a state space tree using any search mechanism in which all the children of the E node are generated before another node becomes the E node
- each answer node x has a cost $c(x)$ associated with it
- minimum cost answer node is to be found
- cost function $\hat{c}(\cdot)$ such that $\hat{c}(x) \leq c(x)$ is used to provide lower bounds on solutions obtainable from any node x
- if *upper* is an upper bound on the cost of a minimum cost solution, then all live nodes x with $\hat{c}(x) > upper$ may be killed as all answer nodes reachable from x have cost $c(x) \geq \hat{c}(x) > upper$
- starting value for *upper* can be set to ∞ and each time a new answer node is found the value of *upper* can be updated

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems
- formulate the search for an optimal solution as a search for a least-cost answer node in a state space tree

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems
- formulate the search for an optimal solution as a search for a least-cost answer node in a state space tree
- define the cost function $c(\cdot)$ such that $c(x)$ is minimum for all nodes representing an optimal solution

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems
- formulate the search for an optimal solution as a search for a least-cost answer node in a state space tree
- define the cost function $c(\cdot)$ such that $c(x)$ is minimum for all nodes representing an optimal solution
- objective function may be used as $c(\cdot)$

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems
- formulate the search for an optimal solution as a search for a least-cost answer node in a state space tree
- define the cost function $c(\cdot)$ such that $c(x)$ is minimum for all nodes representing an optimal solution
- objective function may be used as $c(\cdot)$
- for nodes representing feasible solutions, $c(x)$ is the value of objective function for that feasible solution

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems
- formulate the search for an optimal solution as a search for a least-cost answer node in a state space tree
- define the cost function $c(\cdot)$ such that $c(x)$ is minimum for all nodes representing an optimal solution
- objective function may be used as $c(\cdot)$
- for nodes representing feasible solutions, $c(x)$ is the value of objective function for that feasible solution
- for nodes representing infeasible solutions $c(x) = \infty$

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems
- formulate the search for an optimal solution as a search for a least-cost answer node in a state space tree
- define the cost function $c(\cdot)$ such that $c(x)$ is minimum for all nodes representing an optimal solution
- objective function may be used as $c(\cdot)$
- for nodes representing feasible solutions, $c(x)$ is the value of objective function for that feasible solution
- for nodes representing infeasible solutions $c(x) = \infty$
- for nodes representing partial solutions, $c(x)$ is the cost of the minimum cost node in the subtree with root x

Branch and Bound Algorithm

- used for optimization, minimization or maximization problems
- formulate the search for an optimal solution as a search for a least-cost answer node in a state space tree
- define the cost function $c(\cdot)$ such that $c(x)$ is minimum for all nodes representing an optimal solution
- objective function may be used as $c(\cdot)$
- for nodes representing feasible solutions, $c(x)$ is the value of objective function for that feasible solution
- for nodes representing infeasible solutions $c(x) = \infty$
- for nodes representing partial solutions, $c(x)$ is the cost of the minimum cost node in the subtree with root x
- $c(x)$ is hard to compute, the $\hat{c}(x)$ is used

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$
- assume that every tour starts and ends at vertex 1

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$
- assume that every tour starts and ends at vertex 1
- solution space $S = \{1, \pi, 1 | \pi \text{ is a permutation of} (2, 3, \ldots, n)\}$
  $|S| = (n-1)!$

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$
- assume that every tour starts and ends at vertex 1
- solution space $S = \{1, \pi, 1 | \pi \text{ is a permutation of} (2, 3, \ldots, n)\}$ $|S| = (n-1)!$
- size of S can be reduced by restricting S so that $(1, i_1, i_2, \ldots, i_{n-1}, 1) \in S$ iff $\langle i_j, i_{j+1} \rangle \in E$ $0 \leq j \leq n-1$ and $i_0 = i_n = 1$

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$
- assume that every tour starts and ends at vertex 1
- solution space $S = \{1, \pi, 1 | \pi \text{ is a permutation of} (2, 3, \ldots, n)\}$ $|S| = (n-1)!$
- size of S can be reduced by restricting S so that $(1, i_1, i_2, \ldots, i_{n-1}, 1) \in S$ iff $\langle i_j, i_{j+1} \rangle \in E$ $0 \leq j \leq n-1$ and $i_0 = i_n = 1$
- S can be organized into a state space tree

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$
- assume that every tour starts and ends at vertex 1
- solution space $S = \{1, \pi, 1 | \pi \text{ is a permutation of} (2, 3, \ldots, n)\}$ $|S| = (n-1)!$
- size of S can be reduced by restricting S so that $(1, i_1, i_2, \ldots, i_{n-1}, 1) \in S$ iff $\langle i_j, i_{j+1} \rangle \in E$ $0 \leq j \leq n-1$ and $i_0 = i_n = 1$
- S can be organized into a state space tree
- use LCBB to search the TSP state space tree

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$
- assume that every tour starts and ends at vertex 1
- solution space $S = \{1, \pi, 1 | \pi \text{ is a permutation of} (2, 3, \ldots, n)\}$ $|S| = (n-1)!$
- size of S can be reduced by restricting S so that $(1, i_1, i_2, \ldots, i_{n-1}, 1) \in S$ iff $\langle i_j, i_{j+1} \rangle \in E$ $0 \leq j \leq n-1$ and $i_0 = i_n = 1$
- S can be organized into a state space tree
- use LCBB to search the TSP state space tree
- define cost function $c(\cdot)$ and $\hat{c}(\cdot)$, $u(\cdot)$ such that $\hat{c}(r) \leq c(r) \leq u(r)$ for all nodes r

Traveling Salesperson Problem

- permutation problem $n!$ different permutations of n objects
- $G = (V, E)$ directed graph defining an instance of TSP
- c_{ij} cost of edge $\langle i, j \rangle$ $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$ and $|V| = n$
- assume that every tour starts and ends at vertex 1
- solution space $S = \{1, \pi, 1 | \pi \text{ is a permutation of} (2, 3, \ldots, n)\}$ $|S| = (n-1)!$
- size of S can be reduced by restricting S so that $(1, i_1, i_2, \ldots, i_{n-1}, 1) \in S$ iff $\langle i_j, i_{j+1} \rangle \in E$ $0 \leq j \leq n-1$ and $i_0 = i_n = 1$
- S can be organized into a state space tree
- use LCBB to search the TSP state space tree
- define cost function $c(\cdot)$ and $\hat{c}(\cdot)$, $u(\cdot)$ such that $\hat{c}(r) \leq c(r) \leq u(r)$ for all nodes r
- cost $c(\cdot)$ is such that the solution node with least $c(\cdot)$ corresponds to a shortest tour in G

TSP problem

- one choice for $c(\cdot)$ is

TSP problem

- one choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \\ \qquad \text{if } A \text{ is a leaf} \\ \text{cost of a minimum-cost leaf in the subtree } A, \\ \qquad \text{if } A \text{ is not a leaf} \end{cases}$$

TSP problem

- one choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \\ \qquad \text{if } A \text{ is a leaf} \\ \text{cost of a minimum-cost leaf in the subtree } A, \\ \qquad \text{if } A \text{ is not a leaf} \end{cases}$$

- simple $\hat{c}(\cdot)$ for all A - length of the path defined at node A

TSP problem

- one choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \\ \qquad \text{if } A \text{ is a leaf} \\ \text{cost of a minimum-cost leaf in the subtree } A, \\ \qquad \text{if } A \text{ is not a leaf} \end{cases}$$

- simple $\hat{c}(\cdot)$ for all A - length of the path defined at node A
- path defined at node 6 is $i_0, i_1, i_2 = 1, 2, 4$ consists of edges $\langle 1, 2 \rangle$ $\langle 2, 4 \rangle$

TSP problem

- one choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \\ \qquad \text{if } A \text{ is a leaf} \\ \text{cost of a minimum-cost leaf in the subtree } A, \\ \qquad \text{if } A \text{ is not a leaf} \end{cases}$$

- simple $\hat{c}(\cdot)$ for all A - length of the path defined at node A
- path defined at node 6 is $i_0, i_1, i_2 = 1, 2, 4$ consists of edges $\langle 1, 2 \rangle$ $\langle 2, 4 \rangle$
- better $\hat{c}(\cdot)$ can be obtained by using the reduced cost matrix corresponding to G

TSP problem

- one choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \\ \qquad \text{if } A \text{ is a leaf} \\ \text{cost of a minimum-cost leaf in the subtree } A, \\ \qquad \text{if } A \text{ is not a leaf} \end{cases}$$

- simple $\hat{c}(\cdot)$ for all A - length of the path defined at node A
- path defined at node 6 is $i_0, i_1, i_2 = 1, 2, 4$ consists of edges $\langle 1, 2 \rangle$ $\langle 2, 4 \rangle$
- better $\hat{c}(\cdot)$ can be obtained by using the reduced cost matrix corresponding to G
- a row (column) is said to be reduced iff it contains at least one zero and all remaining entries are non-negative

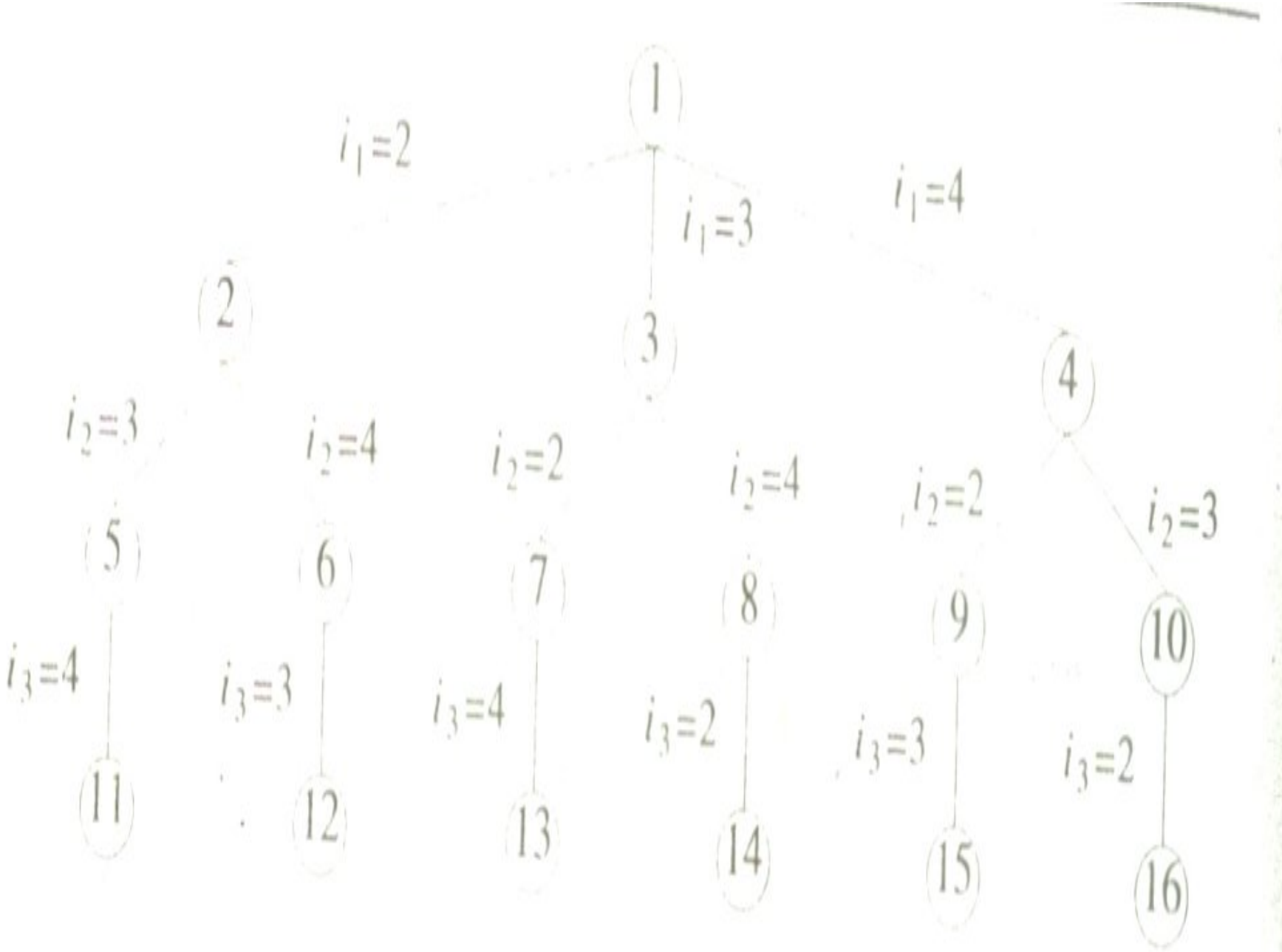
TSP problem

- one choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \\ \qquad \text{if } A \text{ is a leaf} \\ \text{cost of a minimum-cost leaf in the subtree } A, \\ \qquad \text{if } A \text{ is not a leaf} \end{cases}$$

- simple $\hat{c}(\cdot)$ for all A - length of the path defined at node A
- path defined at node 6 is $i_0, i_1, i_2 = 1, 2, 4$ consists of edges $\langle 1, 2 \rangle$ $\langle 2, 4 \rangle$
- better $\hat{c}(\cdot)$ can be obtained by using the reduced cost matrix corresponding to G
- a row (column) is said to be reduced iff it contains at least one zero and all remaining entries are non-negative
- a matrix is reduced iff every row and column is reduced

TSP problem


1
$i_1=2$
$i_1=3$
$i_1=4$
2
3
4
$i_2=3$
$i_2=4$
$i_2=2$
$i_2=4$
$i_2=2$
$i_2=3$
5
6
7
8
9
10
$i_3=4$
$i_3=3$
$i_3=4$
$i_3=2$
$i_3=3$
$i_3=2$
11
12
13
14
15
16

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t
- minimum cost tour remains a minimum cost tour following the subtraction operation

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t
- minimum cost tour remains a minimum cost tour following the subtraction operation
- t is chosen to be the minimum entry in row (column)

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t
- minimum cost tour remains a minimum cost tour following the subtraction operation
- t is chosen to be the minimum entry in row (column)
- repeating this subtraction, the cost matrix can be reduced

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t
- minimum cost tour remains a minimum cost tour following the subtraction operation
- t is chosen to be the minimum entry in row (column)
- repeating this subtraction, the cost matrix can be reduced
- total amount subtracted from the columns and rows is a lower bound on the length of a minimum cost tour and can be used as $\hat{c}$ value for the root of the state space tree

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t
- minimum cost tour remains a minimum cost tour following the subtraction operation
- t is chosen to be the minimum entry in row (column)
- repeating this subtraction, the cost matrix can be reduced
- total amount subtracted from the columns and rows is a lower bound on the length of a minimum cost tour and can be used as $\hat{c}$ value for the root of the state space tree
- subtracting 10, 2, 2, 3, 4 from rows 1, 2, 3, 4, and 5

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t
- minimum cost tour remains a minimum cost tour following the subtraction operation
- t is chosen to be the minimum entry in row (column)
- repeating this subtraction, the cost matrix can be reduced
- total amount subtracted from the columns and rows is a lower bound on the length of a minimum cost tour and can be used as $\hat{c}$ value for the root of the state space tree
- subtracting 10, 2, 2, 3, 4 from rows 1, 2, 3, 4, and 5
- similarly for columns and total amount subtracted is 25

TSP problem

- graph with five vertices - edge $\langle i, j \rangle$
- subtracting a constant t from every entry in one column or one row of the cost matrix reduces the length of every tour by exactly t
- minimum cost tour remains a minimum cost tour following the subtraction operation
- t is chosen to be the minimum entry in row (column)
- repeating this subtraction, the cost matrix can be reduced
- total amount subtracted from the columns and rows is a lower bound on the length of a minimum cost tour and can be used as $\hat{c}$ value for the root of the state space tree
- subtracting 10, 2, 2, 3, 4 from rows 1, 2, 3, 4, and 5
- similarly for columns and total amount subtracted is 25
- all tours in the original graph have a length at least 25

TSP problem

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

- initial cost matrix and reduced cost matrix $L = 25$

TSP problem

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

- initial cost matrix and reduced cost matrix $L = 25$
- associate a reduced cost matrix with every node in the TSP state space tree

TSP problem

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

- initial cost matrix and reduced cost matrix $L = 25$
- associate a reduced cost matrix with every node in the TSP state space tree
- let A reduced cost matrix for node R, S child of R such that the tree edge (R, S) corresponds to including edge $\langle i, j \rangle$ in the tour

TSP problem

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

- initial cost matrix and reduced cost matrix $L = 25$
- associate a reduced cost matrix with every node in the TSP state space tree
- let A reduced cost matrix for node R, S child of R such that the tree edge (R, S) corresponds to including edge $\langle i, j \rangle$ in the tour
- if S is not a leaf, then the reduced cost matrix for S obtained using

TSP problem

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

- initial cost matrix and reduced cost matrix $L = 25$
- associate a reduced cost matrix with every node in the TSP state space tree
- let A reduced cost matrix for node R, S child of R such that the tree edge (R, S) corresponds to including edge $\langle i, j \rangle$ in the tour
- if S is not a leaf, then the reduced cost matrix for S obtained using
  1. change all entries in row i and column j of A to ∞ - prevents the use of any more edges leaving vertex i or entering vertex j

TSP problem

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

- initial cost matrix and reduced cost matrix $L = 25$
- associate a reduced cost matrix with every node in the TSP state space tree
- let A reduced cost matrix for node R, S child of R such that the tree edge (R, S) corresponds to including edge $\langle i, j \rangle$ in the tour
- if S is not a leaf, then the reduced cost matrix for S obtained using
  1. change all entries in row i and column j of A to ∞ - prevents the use of any more edges leaving vertex i or entering vertex j
  2. set $A(j, 1)$ to ∞ - prevents the use of edge $\langle j, 1 \rangle$

TSP problem

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

- initial cost matrix and reduced cost matrix $L = 25$
- associate a reduced cost matrix with every node in the TSP state space tree
- let A reduced cost matrix for node R, S child of R such that the tree edge (R, S) corresponds to including edge $\langle i, j \rangle$ in the tour
- if S is not a leaf, then the reduced cost matrix for S obtained using
  1. change all entries in row i and column j of A to ∞ - prevents the use of any more edges leaving vertex i or entering vertex j
  2. set $A(j, 1)$ to ∞ - prevents the use of edge $\langle j, 1 \rangle$
  3. reduce all rows and columns in the resulting matrix except for rows and columns containing only ∞

TSP problem

- let resulting matrix B

TSP problem

- let resulting matrix B
- if r is the total amount subtracted in step 3 then $\hat{c}(S) = \hat{c}(R) + A(i, j) + r$

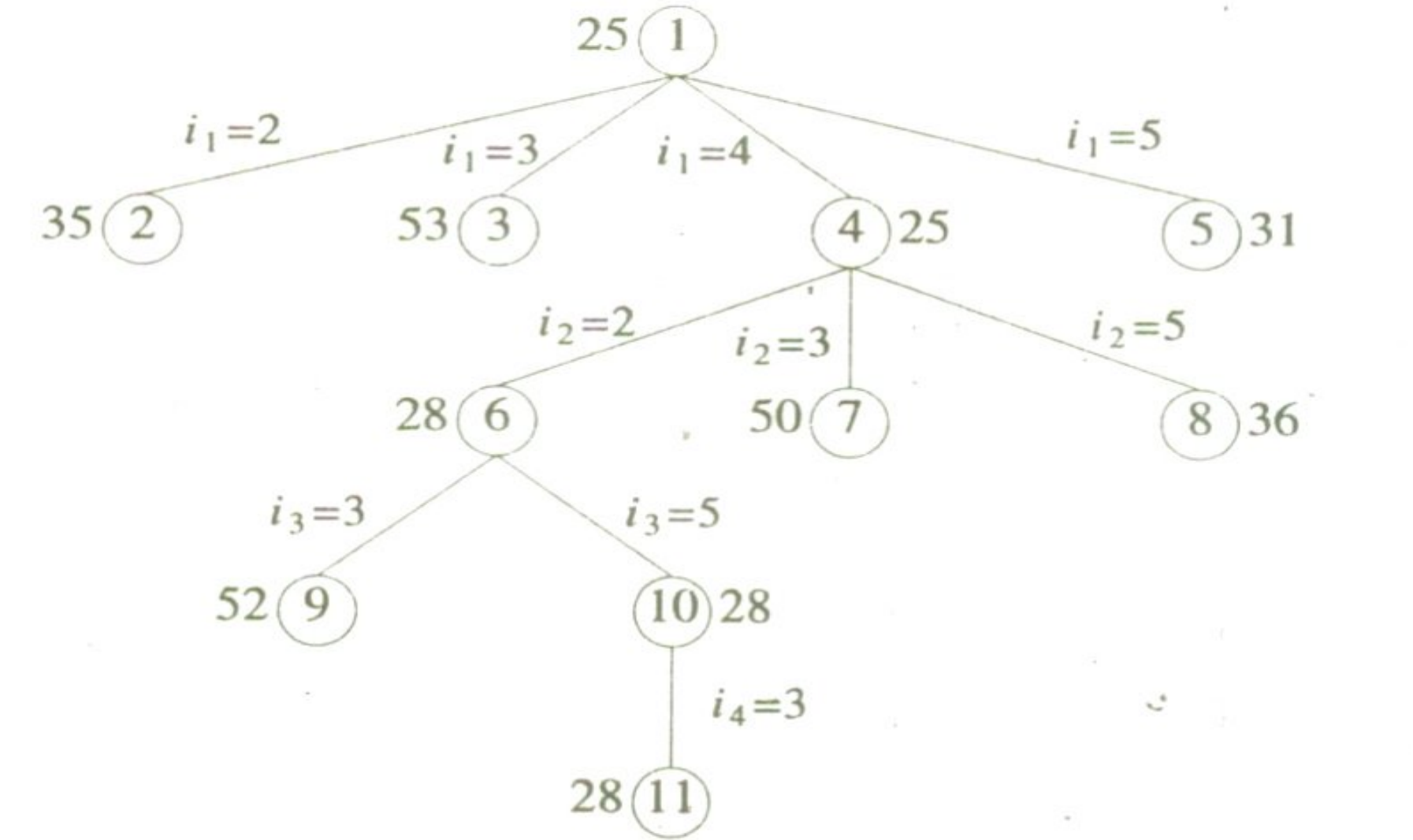
TSP problem

- let resulting matrix B
- if r is the total amount subtracted in step 3 then $\hat{c}(S) = \hat{c}(R) + A(i, j) + r$
- for leaf nodes, $\hat{c}(\cdot)$ is easily computed as each leaf defines a unique tour

TSP problem

- let resulting matrix B
- if r is the total amount subtracted in step 3 then $\hat{c}(S) = \hat{c}(R) + A(i, j) + r$
- for leaf nodes, $\hat{c}(\cdot)$ is easily computed as each leaf defines a unique tour
- for the upper bound function u use $u(R) = \infty$ for all nodes R
- use LCBB algorithm for the problem instance defined by initial matrix
- start with initial reduced matrix and $upper = \infty$
- starting with root node as E node, nodes 2, 3, 4, and 5 are generated

TSP problem


25 1
$i_1=2$
$i_1=3$
$i_1=4$
$i_1=5$
35 2
53 3
4 25
5 31
$i_2=2$
$i_2=3$
$i_2=5$
28 6
50 7
8 36
$i_3=3$
$i_3=5$
52 9
10 28
$i_4=3$
28 11

TSP problem

- reduced matrices corresponding to different nodes

TSP problem

- reduced matrices corresponding to different nodes

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix} \qquad \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

(a) Path 1,2; for node 2 (b) Path 1,3; node 3

TSP problem

- reduced matrices corresponding to different nodes

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

(a) Path 1,2; for node 2

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

(b) Path 1,3; node 3

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

(c) Path 1,4; for node 4

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Path 1,5; node 5

TSP problem

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

(e) Path 1,4,2; for node 6 (f) Path 1,4,3; node 7

TSP problem

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

(e) Path 1,4,2; for node 6

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

(f) Path 1,4,3; node 7

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

(g) Path 1,4,5; for node 8

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

(h) Path 1,4,2,3; node 9

TSP problem

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

(i) Path 1,4,2,5; for node 10

TSP problem

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

(i) Path 1,4,2,5; for node 10

- for (b) all entries in row 1 and column 3 to ∞

TSP problem

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

(i) Path 1,4,2,5; for node 10

- for (b) all entries in row 1 and column 3 to ∞
- setting the element (3,1) to ∞, reducing column 1 by subtracting by 11

TSP problem

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

(i) Path 1,4,2,5; for node 10

- for (b) all entries in row 1 and column 3 to ∞
- setting the element (3,1) to ∞, reducing column 1 by subtracting by 11
- $\hat{c}$ for node 3 is 25 + 17 (the cost of edge $\langle 1, 3 \rangle$ in the reduced matrix) + 11 = 53

TSP problem

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

(i) Path 1,4,2,5; for node 10

- for (b) all entries in row 1 and column 3 to ∞
- setting the element (3,1) to ∞, reducing column 1 by subtracting by 11
- $\hat{c}$ for node 3 is $25 + 17$ (the cost of edge $\langle 1, 3 \rangle$ in the reduced matrix) $+ 11 = 53$
- calculate $\hat{c}$ values for other nodes similarly
- value of *upper* is unchanged and node 4 becomes the next E node

TSP problem

- children 6, 7, and 8 are generated, live nodes 2, 3, 5, 6, 7, and 8

TSP problem

- children 6, 7, and 8 are generated, live nodes 2, 3, 5, 6, 7, and 8
- node 6 has least $\hat{c}$ becomes the next E node and nodes 9 and 10 are generated

TSP problem

- children 6, 7, and 8 are generated, live nodes 2, 3, 5, 6, 7, and 8
- node 6 has least $\hat{c}$ becomes the next E node and nodes 9 and 10 are generated
- node 10 is next E node and solution node 11 is generated

TSP problem

- children 6, 7, and 8 are generated, live nodes 2, 3, 5, 6, 7, and 8
- node 6 has least $\hat{c}$ becomes the next E node and nodes 9 and 10 are generated
- node 10 is next E node and solution node 11 is generated
- tour length for this node $\hat{c}(11) = 28$ and *upper* is updated to 28
- next E node, node 5 having $\hat{c}(5) = 31 > upper$
- LCBB terminates with 1, 4, 2, 5, 3, 1 as the shortest length tour