Parallel Programming

Parallel algorithms
Combinatorial Search

Some Combinatorial Search Methods

- Divide and conquer
- Backtrack search
- Branch and bound
- Game tree search (minimax, alpha-beta)

Terminology

- **Combinatorial algorithm: computation performed on discrete structure**

- **Combinatorial search: finding one or more optimal or suboptimal solutions in a defined problem space**

- **Kinds of combinatorial search problem**
  - **Decision problem (exists (find 1 solution); doesn't exist)**

  - **Optimization problem (the best solution)**

Examples of Combinatorial Search

- Laying out circuits in VLSI
  - Find the smallest area
- Planning motion of robot arms
  - Smallest distance to move (with or without constraints)
- Assigning crews to airline flights
- Proving theorems
- Playing games

Search Tree

- Each node represents a problem or sub-problem
- Root of tree: initial problem to be solved
- Children of a node created by adding constraints (one move from the father)
- AND node: to find solution, must solve problems represented by all children
- OR node: to find a solution, solve any of the problems represented by the children

Search Tree (cont.)

- **AND tree**
  - Contains only AND nodes
  - Divide-and-conquer algorithms
- **OR tree**
  - Contains only OR nodes
  - Backtrack search and branch and bound
- **AND/OR tree**
  - Contains both AND and OR nodes
  - Game trees

Divide and Conquer

- Divide-and-conquer methodology
  - Partition a problem into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems
- Recursive: sub-problems may be solved using the divide-and-conquer methodology
- Example: quicksort

Best for Centralized Multiprocessor

- Unsolved subproblems kept in one shared stack
- Processors needing work can access the stack
- Processors with extra work can put it on the stack
- Effective workload balancing mechanism
- Stack can become a bottleneck as number of processors increases

Multicomputer Divide and Conquer

- Subproblems must be distributed among memories of individual processors

- Two designs
  - Original problem and final solution stored in memory of a single processor
  - Both original problem and final solution distributed among memories of all processors

Design 1

- Algorithm has three phases
- Phase 1: problems divided and propagated throughout the parallel computer
- Phase 2: processors compute solutions to their subproblems
- Phase 3: partial results are combined
- Maximum speedup limited by propagation and combining overhead

Design 2

- Both original problem and final solution are distributed among processors' memories
- Eliminates starting up and winding down phases of first design
- Allows maximum problem size to increase with number of processors
- Used this approach for parallel quicksort algorithms
- Challenge: keeping workloads balanced among processors

Backtrack Search

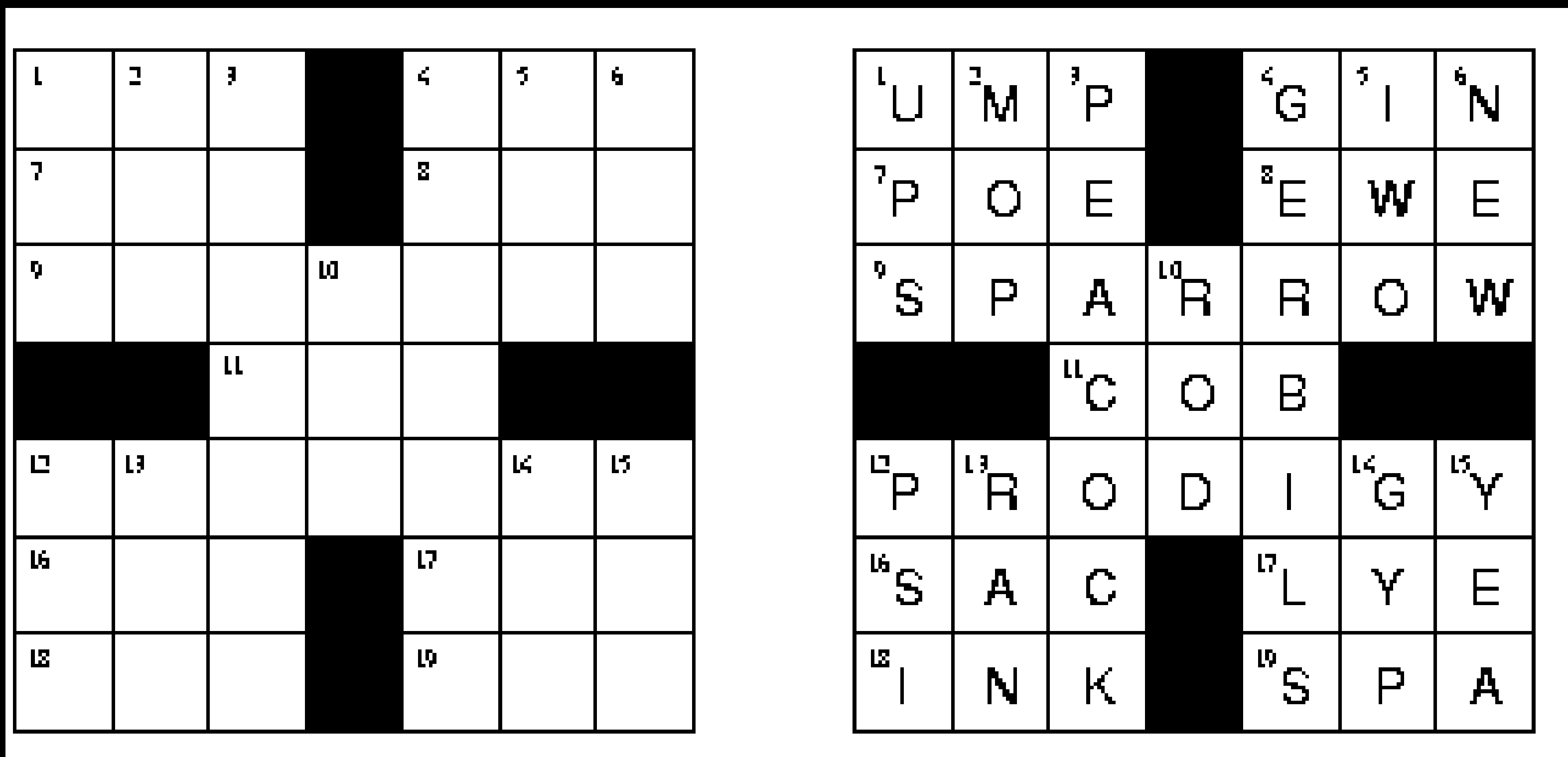
- Uses depth-first search to consider alternative solutions to a combinatorial search problem
- Recursive algorithm
- Backtrack occurs when
  - A node has no children (“dead end”)
  - All of a node’s children have been explored

Example: Crossword Puzzle Creation

- Given
  - Blank crossword puzzle
  - Dictionary of words and phrases
- Assign letters to blank spaces so that all puzzle's horizontal and vertical "words" are from the dictionary
- Halt as soon as a solution is found

Crossword Puzzle Problem

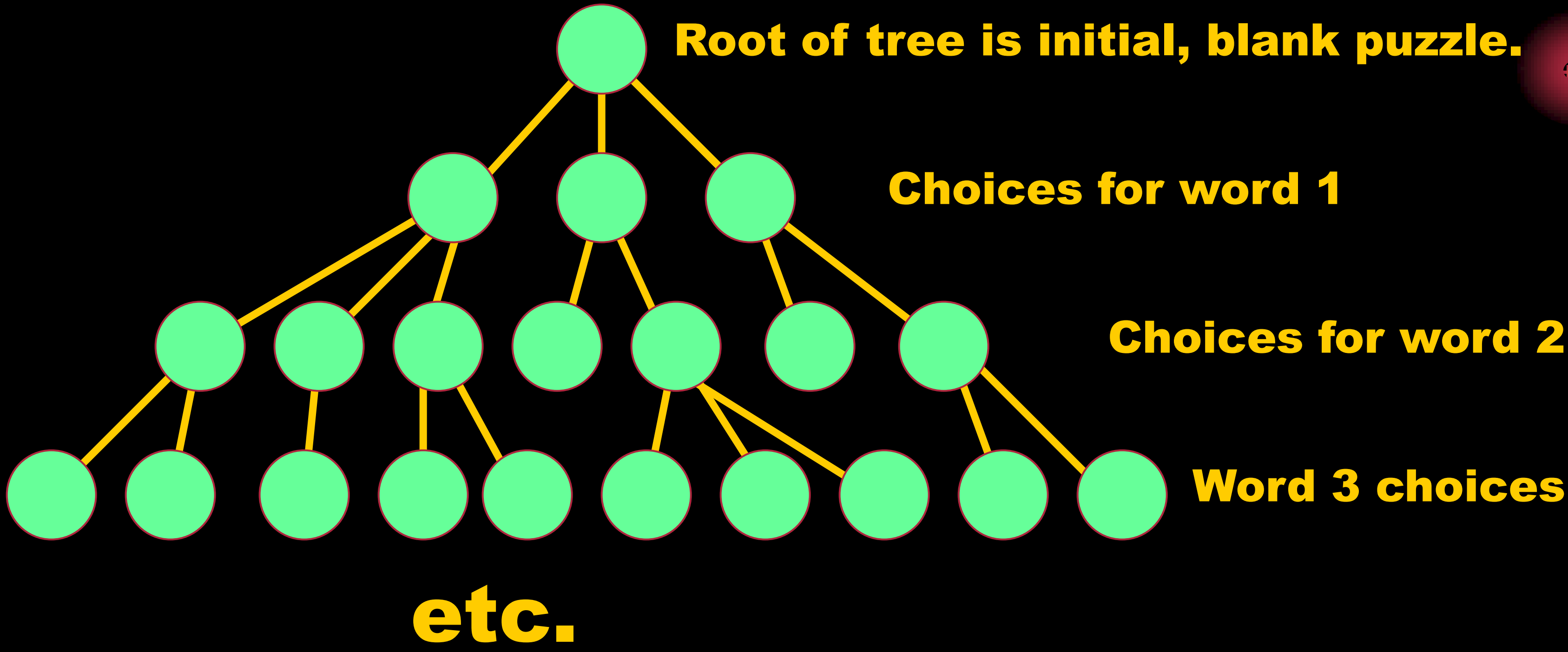
Given a blank crossword puzzle and a dictionary .... find a way to fill in the puzzle.



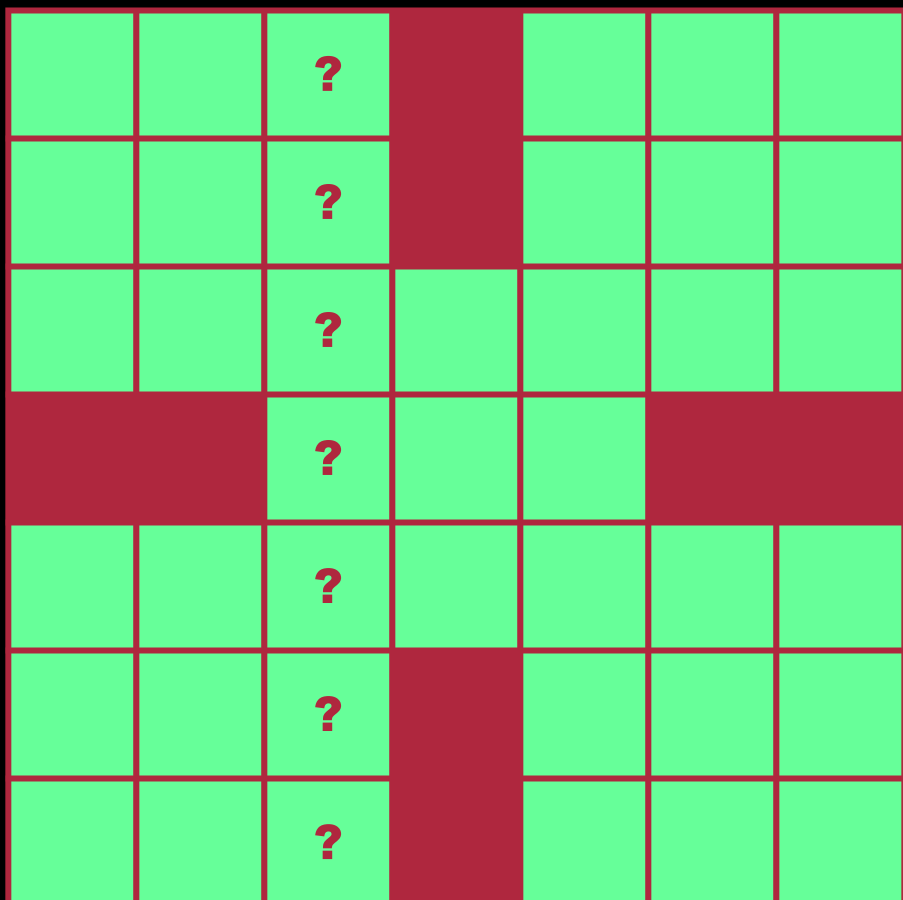
A Search Strategy

- Identify longest incomplete word in puzzle (break ties arbitrarily)
- Look for a word of that length
- If cannot find such a word, backtrack
- Otherwise, find longest incomplete word that has at least one letter assigned (break ties arbitrarily)
- Look for a word of that length
- If cannot find such a word, backtrack
- Recurse until a solution is found or all possibilities have been attempted

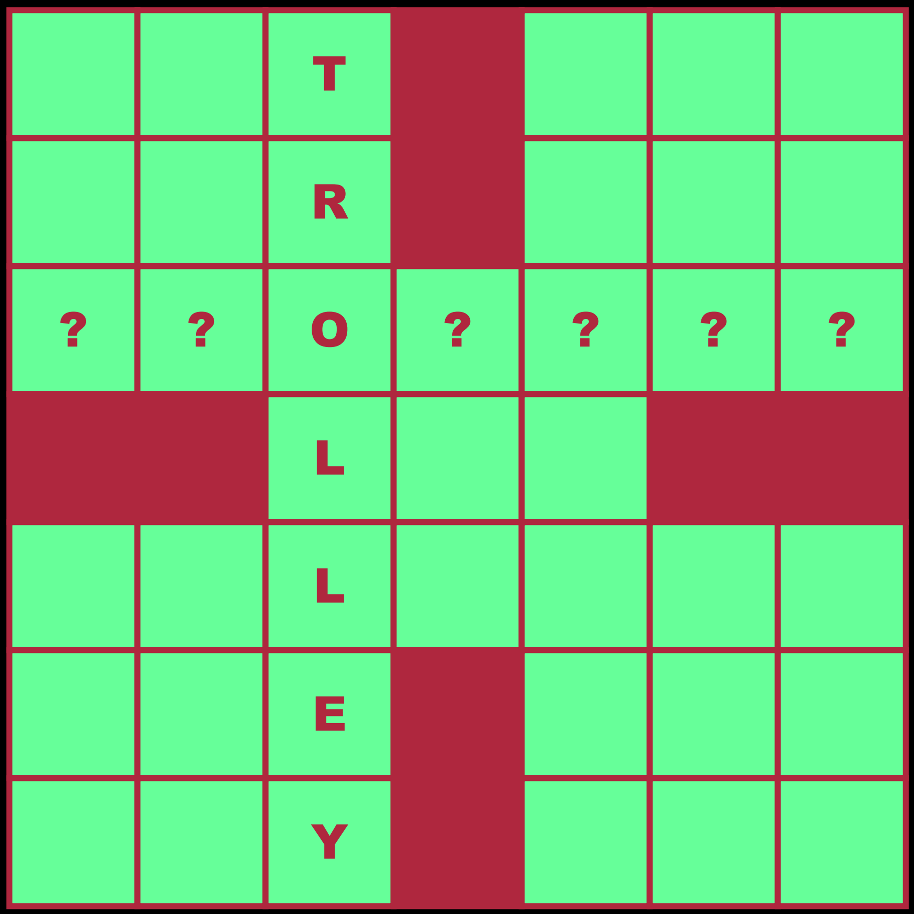
State Space Tree

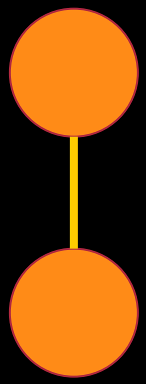


Backtrack Search

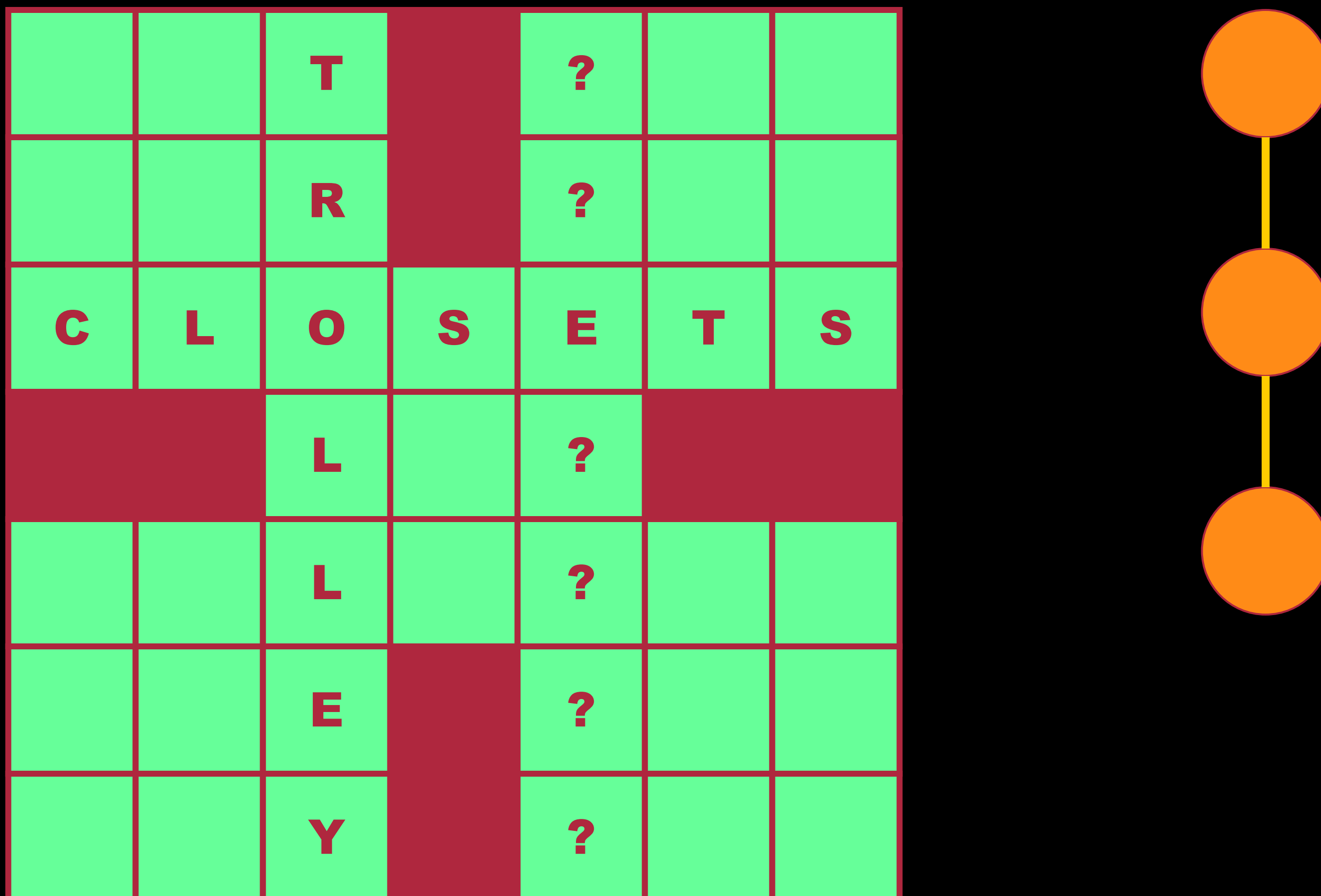


Backtrack Search

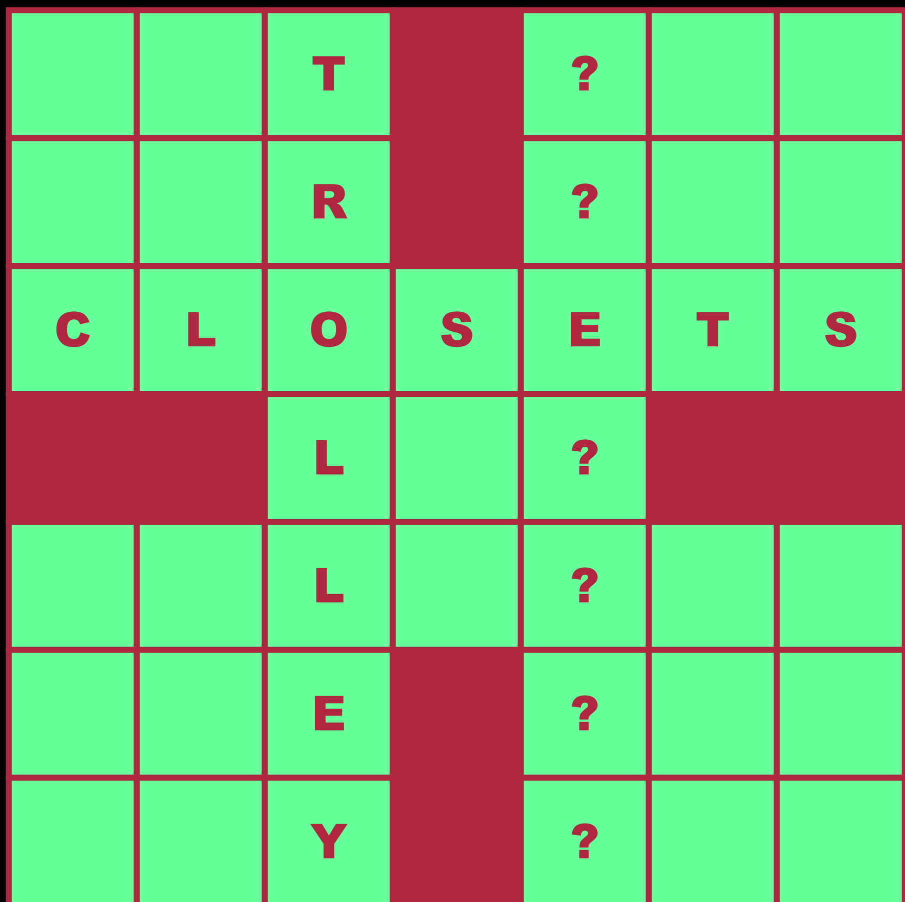


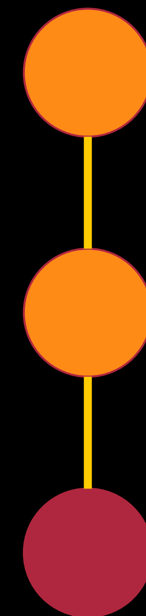


Backtrack Search



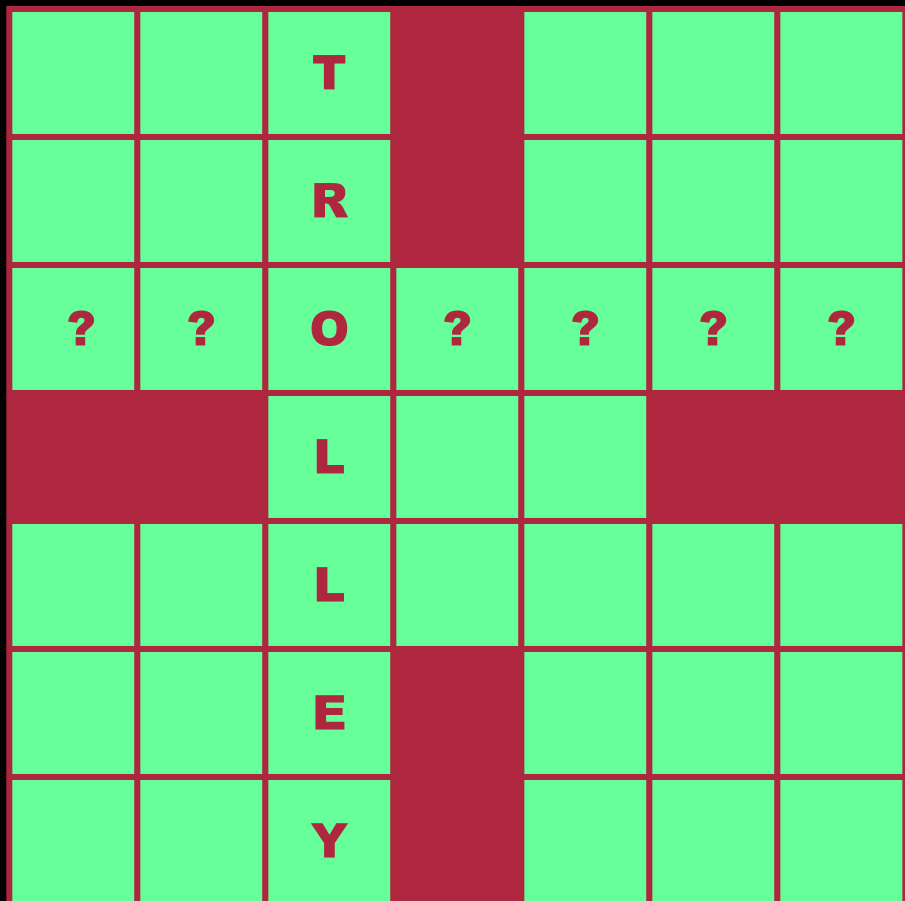
Backtrack Search

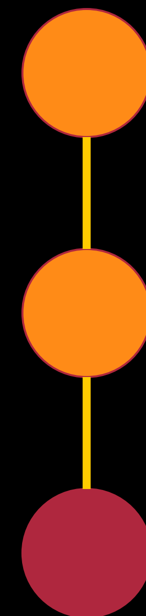




**Cannot find word.
Must backtrack.**

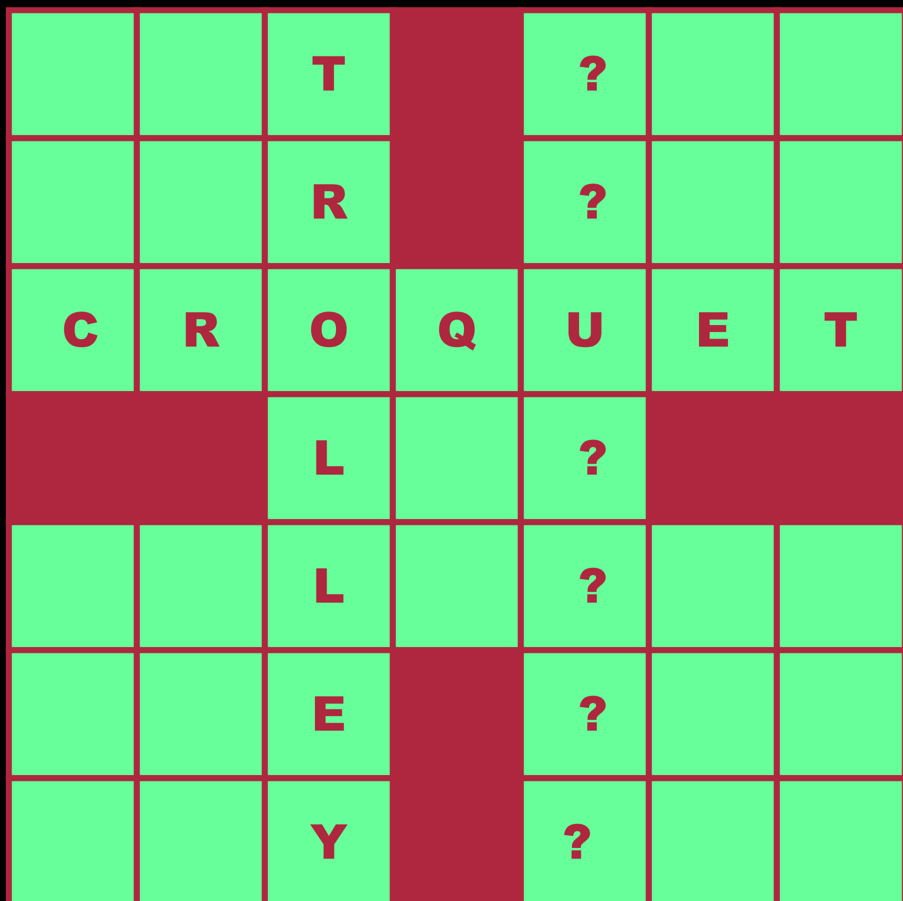
Backtrack Search

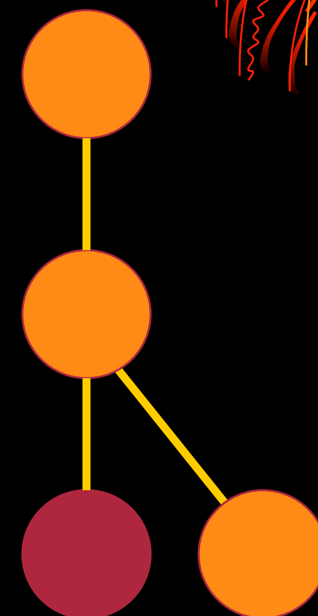




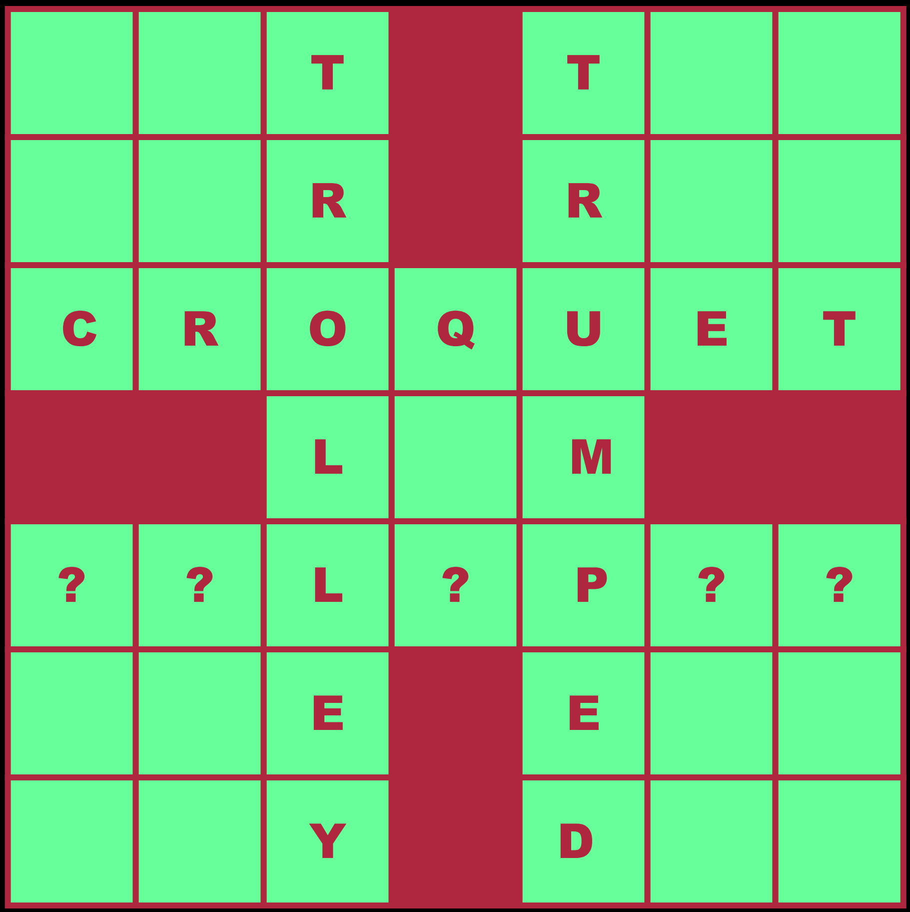
Cannot find word.
Must backtrack.

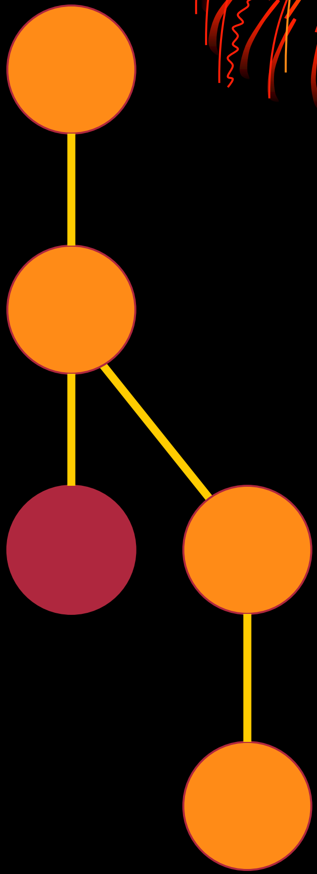
Backtrack Search





Backtrack Search





Time and Space Complexity

- Suppose average branching factor in state space tree is ***b***

- Searching a tree of depth ***k*** requires examining

$$1+b+b^2+\cdots+b^k=\frac{b^{k+1}-b}{b-1}+1=\theta(b^k)$$

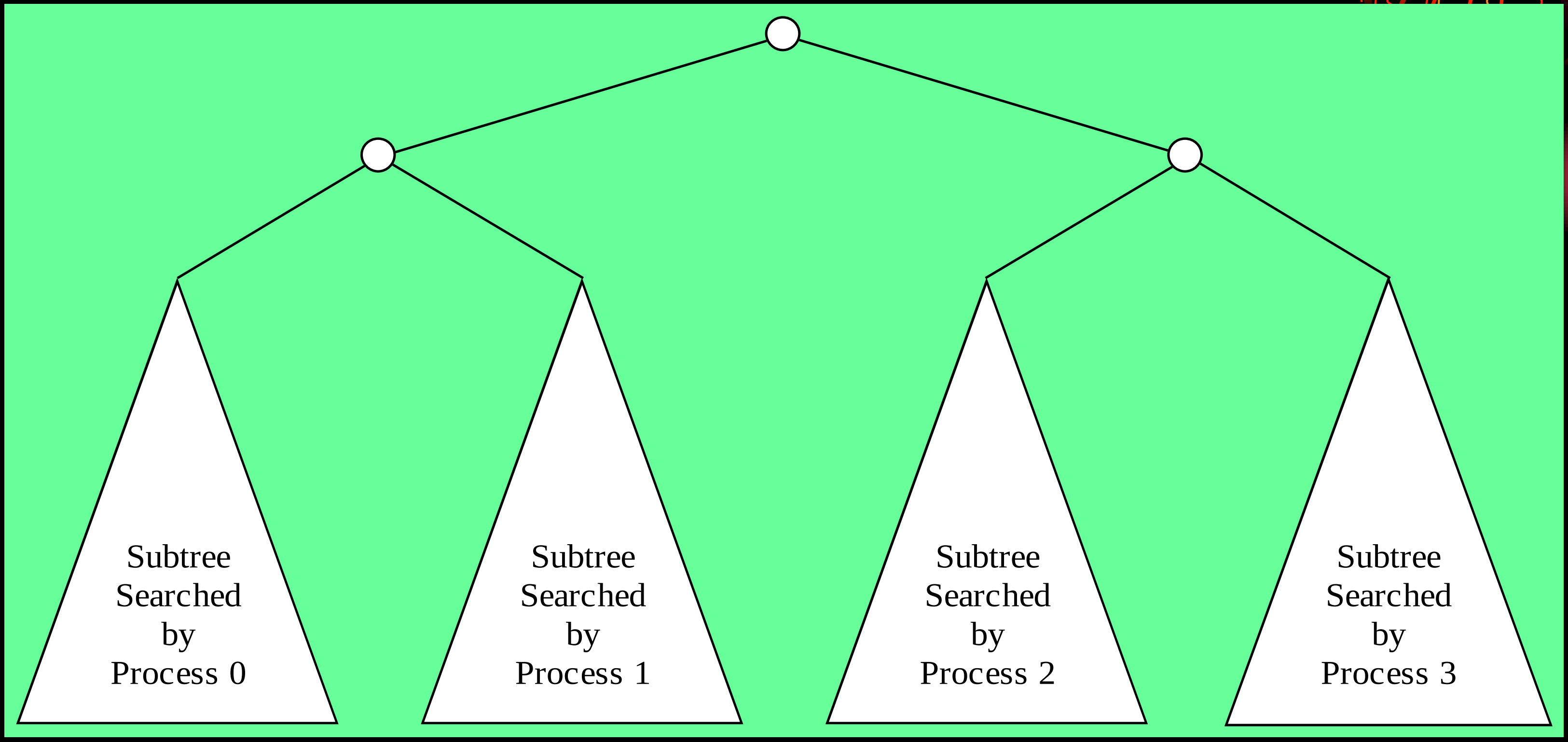
nodes in the worst case (exponential time)

- Amount of memory usually required is $\Theta(k)$

Parallel Backtrack Search

- First strategy: give each processor a subtree
- Suppose $p = b^k$
  - A process searches all nodes to depth k
  - It then explores only one of subtrees rooted at level k
  - If d (depth of search) > $2k$, time required by each process to traverse first k levels of state space tree is negligible

Parallel Backtrack when $p = b^k$

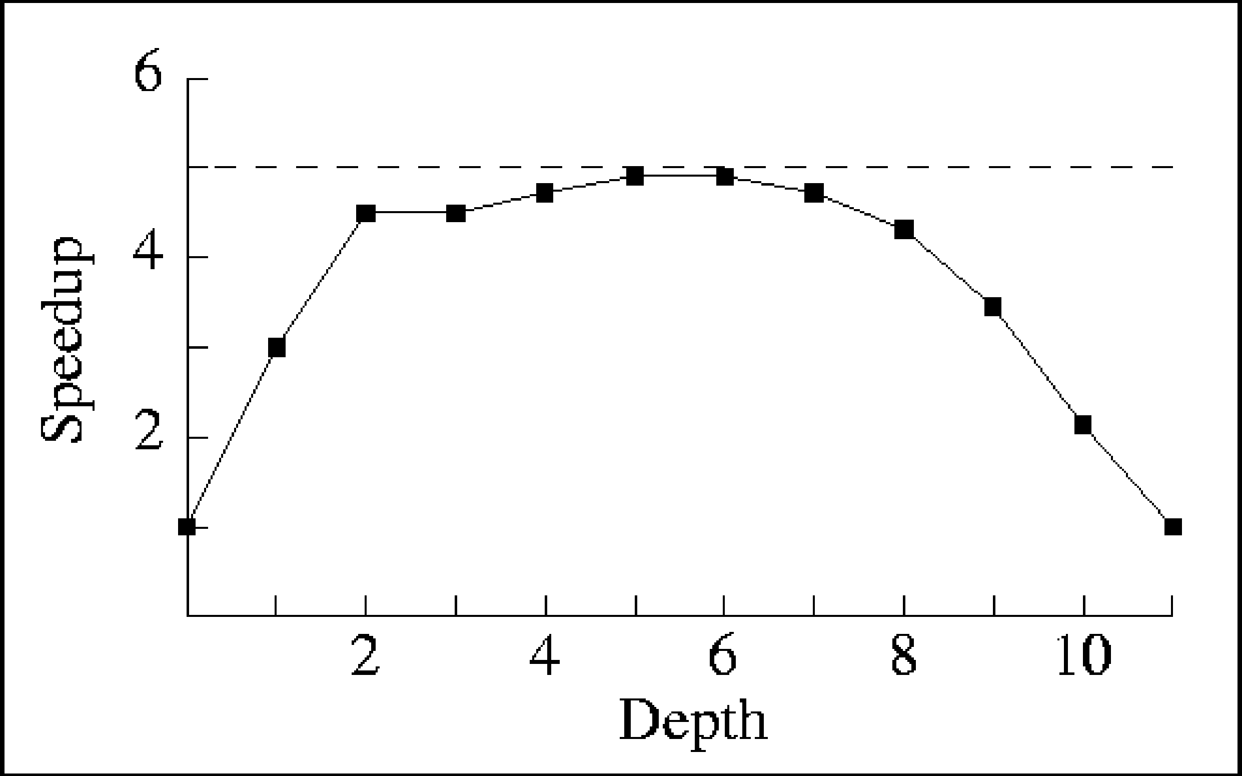


What If $p \neq b^k$?

- A process can perform sequential search to level *m* (where $b^m > p$) of state space tree
- Each process explores its share of the subtrees rooted by nodes at level *m*
- As *m* increases, there are more subtrees to divide among processes, which can make workloads more balanced
- Increasing *m* also increases number of redundant computations

Maximum Speedup when $p \neq b^k$

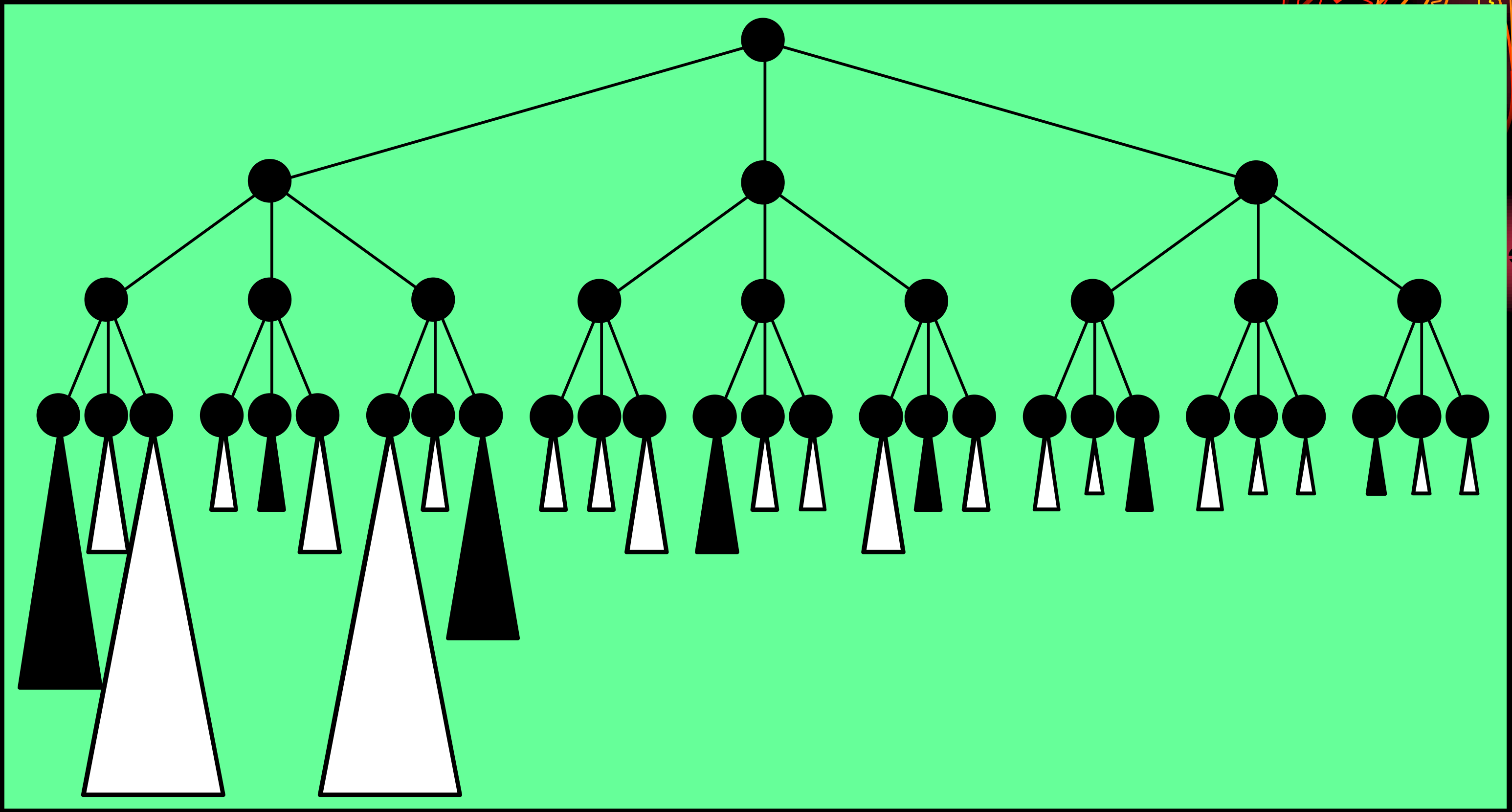
In this example 5 processors are exploring a state space tree with branching factor 3 and depth 10.



Disadvantage of Allocating One Subtree per Process

- In most cases state space tree is not balanced
- Example: in crossword puzzle problem, some word choices lead to dead ends quicker than others
- Alternative: make sequential search go deeper, so that each process handles many subtrees (cyclic allocation)

Allocating Many Subtrees per Process



b = 3; p = 4; m = 3; allocation rule → (subtree nr) % p == rank

Backtrack Algorithm

cutoff_count – nr of nodes at cutoff_depth
cutoff_depth – depth at which subtrees are divided among processes
depth – maximum search depth in the state space tree
moves – records the path to the current node (moves made so far)
p, id – number of processes, process rank

```
Parallel_Backtrack(node, level)
   if (level == depth)
      if (node is a solution)
         Print_Solution(moves)
   else
      if (level == cutoff_depth)
         cutoff_count ++
         if (cutoff_count % p  !=  id)
            return
      possible_moves = Count_Moves(node)       // nr of possible moves from current node
      for i = 1 to possible_moves
         node = Make_Move(node, i)
         moves[ level ] = i
         Parallel_Backtrack(node, level+1)
         node = Unmake_Move(node, i)
   return
```

Distributed Termination Detection

- Suppose we only want to print one solution
- We want all processes to halt as soon as one process finds a solution
- This means processes must periodically check for messages
  - Every process calls MPI_Iprobe every time search reaches a particular level (such as the cutoff depth)
  - A process sends a message after it has found a solution

Simple (Incorrect) Algorithm

- A process halts after one of the following events has happened:
  - It has found a solution and sent a message to all of the other processes
  - It has received a message from another process
  - It has completely searched its portion of the state space tree

Why Algorithm Fails

- If a process calls MPI_Finalize before another active process attempts to send it a message, we get a run-time error

- How this could happen?
  - A process finds a solution after another process has finished searching its share of the subtrees

  OR

  - A process finds a solution after another process has found a solution

Distributed Termination Problem

- Distributed termination problem: Ensuring that
  - all processes are inactive AND
  - no messages are en route

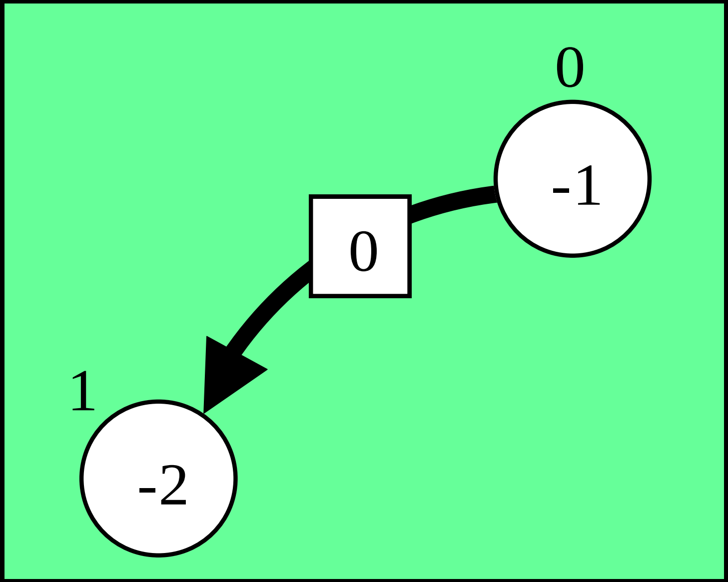
- Solution developed by Dijkstra, Seijen, and Gasteren in early 1980s

Dijkstra et al.'s Algorithm

- Each process has a color and a message count
  - Initial color is white
  - Initial message count is 0
- A process that sends a message turns black and increments its message count
- A process that receives a message turns black and decrements its message count
- If all processes are white and sum of all their message counts are 0, there are no pending messages and we can terminate the processes

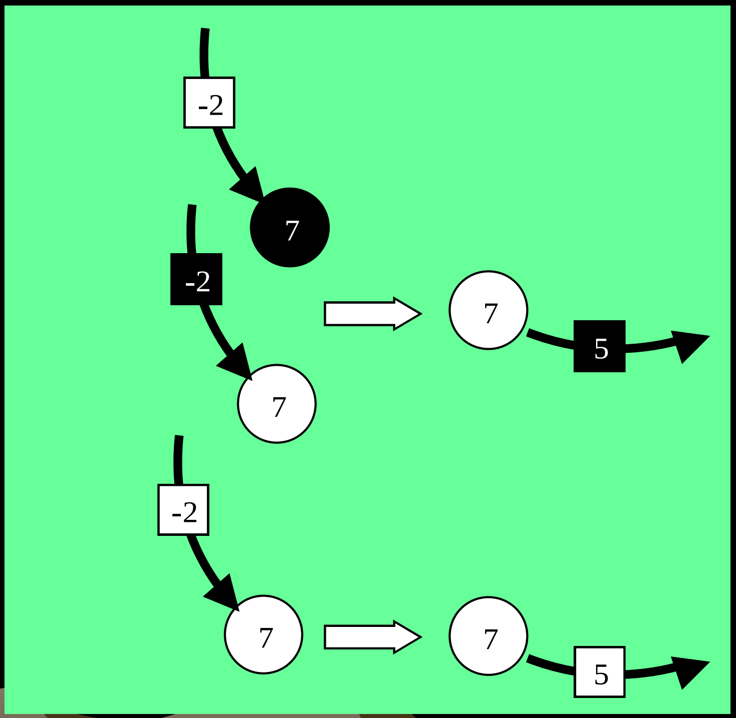
Dijkstra et al.'s Algorithm (cont.)

- Organize processes into a logical ring
- Process 0 passes a token around the ring
- Token also has a color (initially white) and count (initially 0)



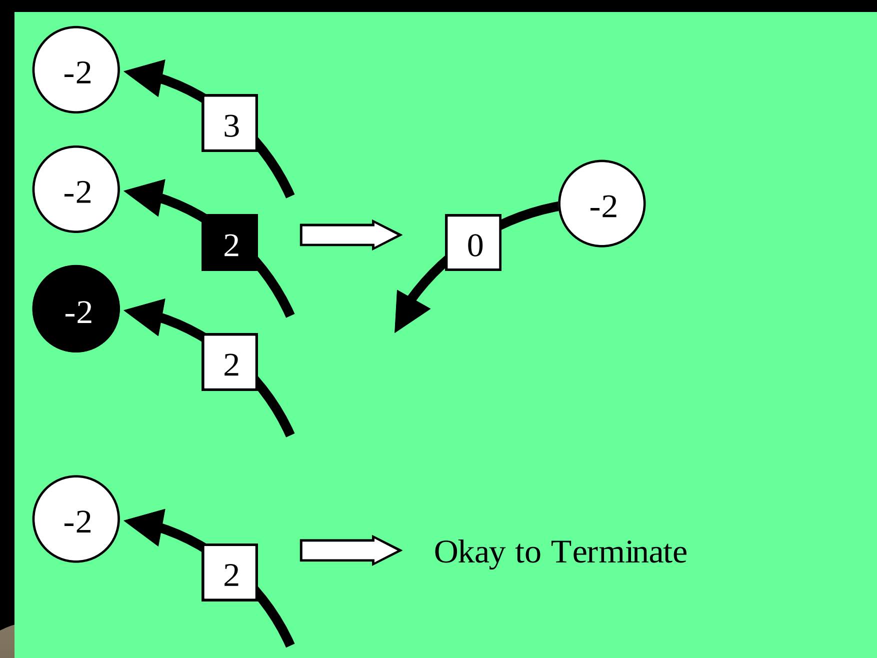
Dijkstra et al.’s Algorithm (cont.)

- A process receives the token
  - If process is black
    - Process changes token color to black
    - Process changes its color to white
  - Process adds its message count to token’s message count
- A process sends the token to its successor in the logical ring



Dijkstra et al.'s Algorithm (cont.)

- Process 0 receives the token
  - Safe to terminate processes if
    - Token is white
    - Process 0 is white
    - Token count + process 0 message count = 0
  - Otherwise, process 0 must probe ring of processes again



Branch and Bound

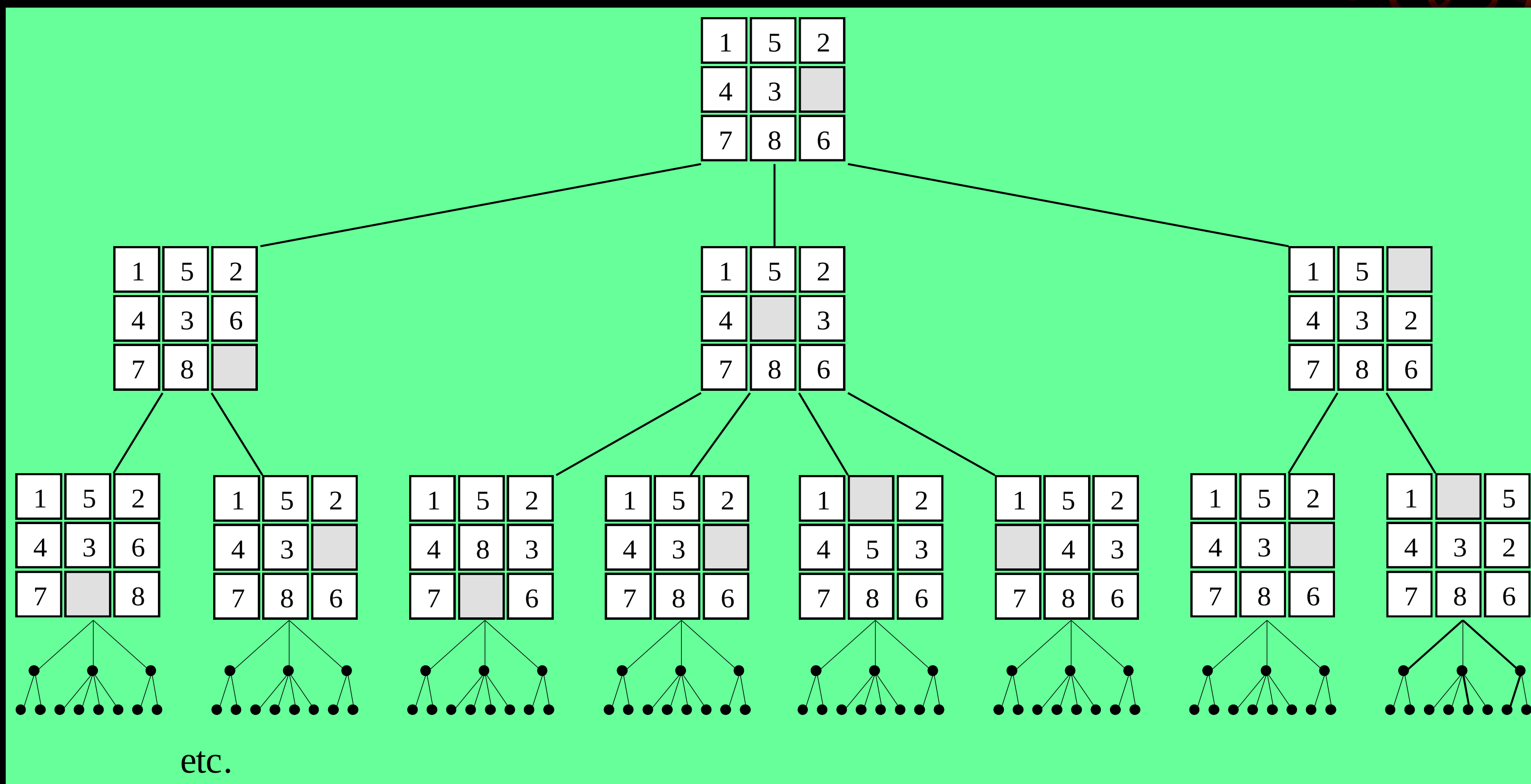
- Variant of backtrack search
- Takes advantage of information about optimality of partial solutions to avoid considering solutions that cannot be optimal

Example: 8-puzzle

1	2	3
4	5	6
7	8	Hole

This is the solution state. Tiles slide up, down, or sideways into hole.

State Space Tree Represents Possible Moves



Branch-and-bound Methodology

- Could solve puzzle by pursuing breadth-first search of state space tree
- We want to examine as few nodes as possible
- Can speed search if we associate with each node an estimate of minimum number of tile moves needed to solve the puzzle, given moves made so far

Manhattan (or City Block) Distance

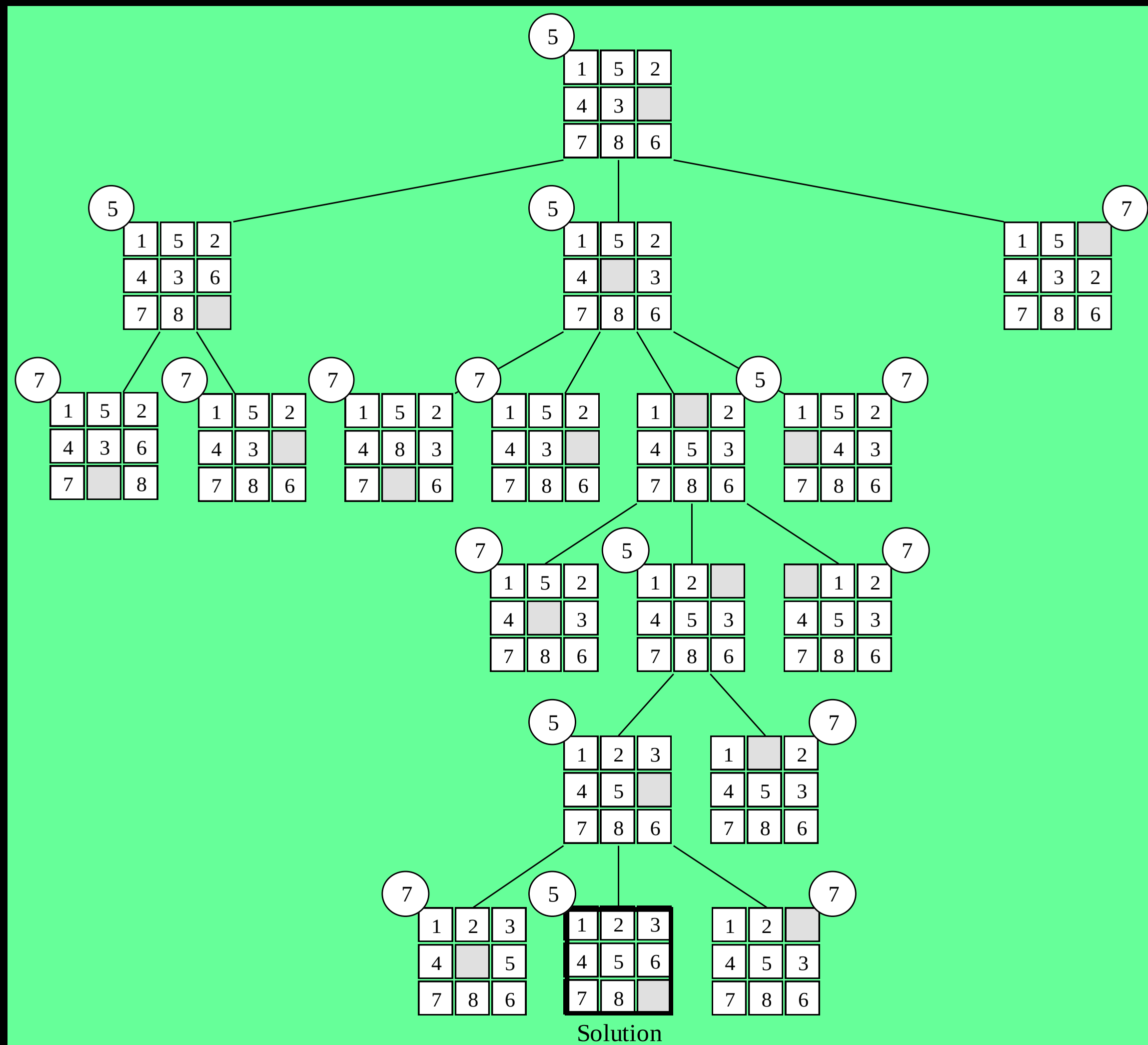
4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

Manhattan distance from the yellow intersection.

A Lower Bound Function

- A lower bound on number of moves needed to solve puzzle is sum of Manhattan distance of each tile's current position from its correct position
- Depth of node in state space tree indicates number of moves made so far
- Adding two values gives lower bound on number of moves needed for any solution, given moves made so far
- We always search from node having smallest value of this function (best-first search)

Best-first Search of 8-puzzle



Pseudocode: Sequential Algorithm

```
// initial – initial problem
// q – priority queue
// u, v – nodes of the search tree

Intialize (q)
Insert (q, initial)
repeat
   u ← Delete_Min (q)
   if u is a solution then
      Print_solution (u)
      Halt
   else
      for i ← 1 to Possible_Constraints (u) do
         Add constraint i to u, creating v
         Insert (q, v)
```

Time and Space Complexity

- In **worst case**, lower bound function causes function to perform breadth-first search
- Suppose branching factor is b and optimum solution is at depth k of state space tree
- **Worst-case** time complexity is $\Theta(b^k)$
- On average, b nodes inserted into priority queue every time a node is deleted
- **Worst-case** space complexity is $\Theta(b^k)$
- Memory limitations often put an upper bound on the size of the problem that can be solved

Parallel Branch and Bound

- We will develop a parallel algorithm suitable for implementation on a multicomputer or distributed multiprocessor
- Conflicting goals
  - Want to maximize ratio of local to non-local memory references
  - Want to ensure processors searching worthwhile portions of state space tree

Single Priority Queue

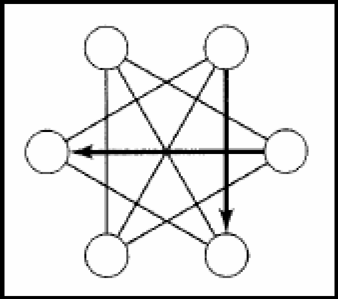
- Maintaining a single priority queue not a good idea
- Communication overhead too great
- Accessing queue is a performance bottleneck
- Does not allow problem size to scale with number of processors

Multiple Priority Queues

- Each process maintains separate priority queue of unexamined subproblems
- Each process retrieves subproblem with smallest lower bound to continue search
- Occasionally processes send unexamined subproblems to other processes

Start-up Mode

- Process 0 contains original problem in its priority queue
- Other processes have no work
- After process 0 distributes an unexamined subproblem, 2 processes have work
- A logarithmic number of distribution steps are sufficient to get all processes engaged



Efficiency

- Conditions for solution to be found and guaranteed optimal
  - At least one solution node must be found
  - All nodes in state space tree with smaller lower bounds must be explored
- Execution time dictated by which of these events occurs last
- This depends on number of processes, shape of state space tree, communication pattern

Efficiency (cont.)

- **Sequential algorithm searches minimum number of nodes (never explores nodes with lower bounds greater than cost of optimal solution)**
- **Parallel algorithm may examine unnecessary nodes because each process searching *locally best* nodes**
- **Exchanging subproblems**
  - **promotes distribute of subproblems with good lower bounds, reducing amount of wasted work**
  - **increases communication overhead**

Halting Conditions

- Distributed termination detection more complicated than for backtrack search
- Can only halt when
  - Have found a solution
  - Verified no better solutions exist

Modifications to DTP Algorithm

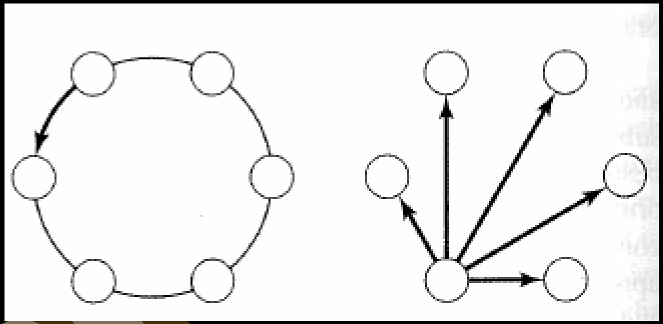
- **Process turns black if it manipulates an unexamined subproblem with lower bound less than cost of best solution found so far**

- **Add additional fields to termination token**
  - Cost of best solution found so far
  - Solution itself (i.e., moves made to reach solution)

Actions When Process Gets Token

- Updates token's color, count fields
- If locally found solution better than one carried by token, updates token
- If lower bound of first unexamined problem in priority queue ≥ best solution found so far, empties priority queue

View Algorithm



Searching Game Trees

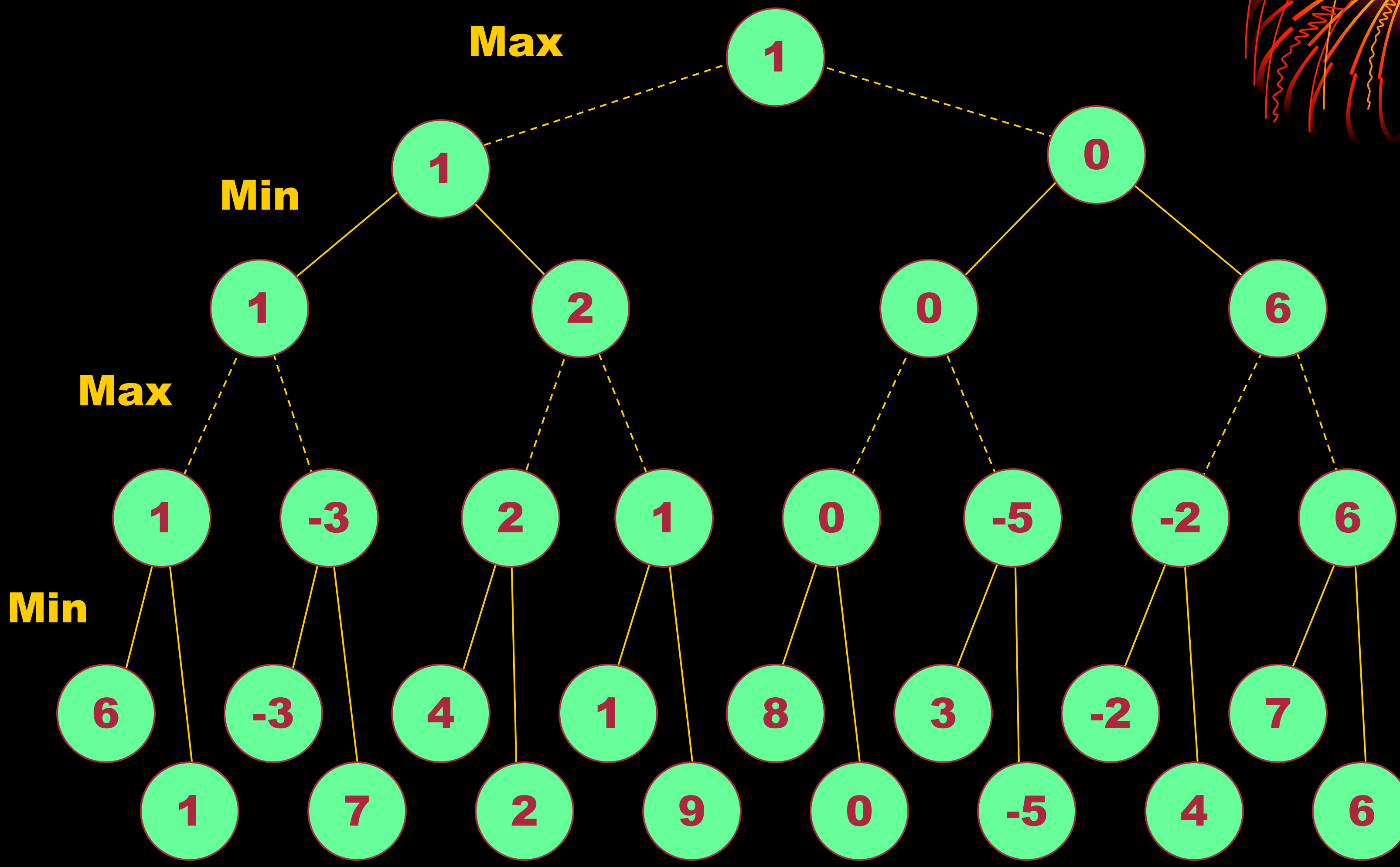
- **Best programs for chess, checkers based on exhaustive search**

- **Algorithms consider series of moves and responses, evaluate desirability of resulting positions, and work their way back up search tree to determine best initial move**

Minimax Algorithm

- A form of **depth-first search**
- Value node = value of position from point of view of player 1
- Player 1 wants to maximize value of node
- Player 2 want to minimize value of node

Illustration of Minimax



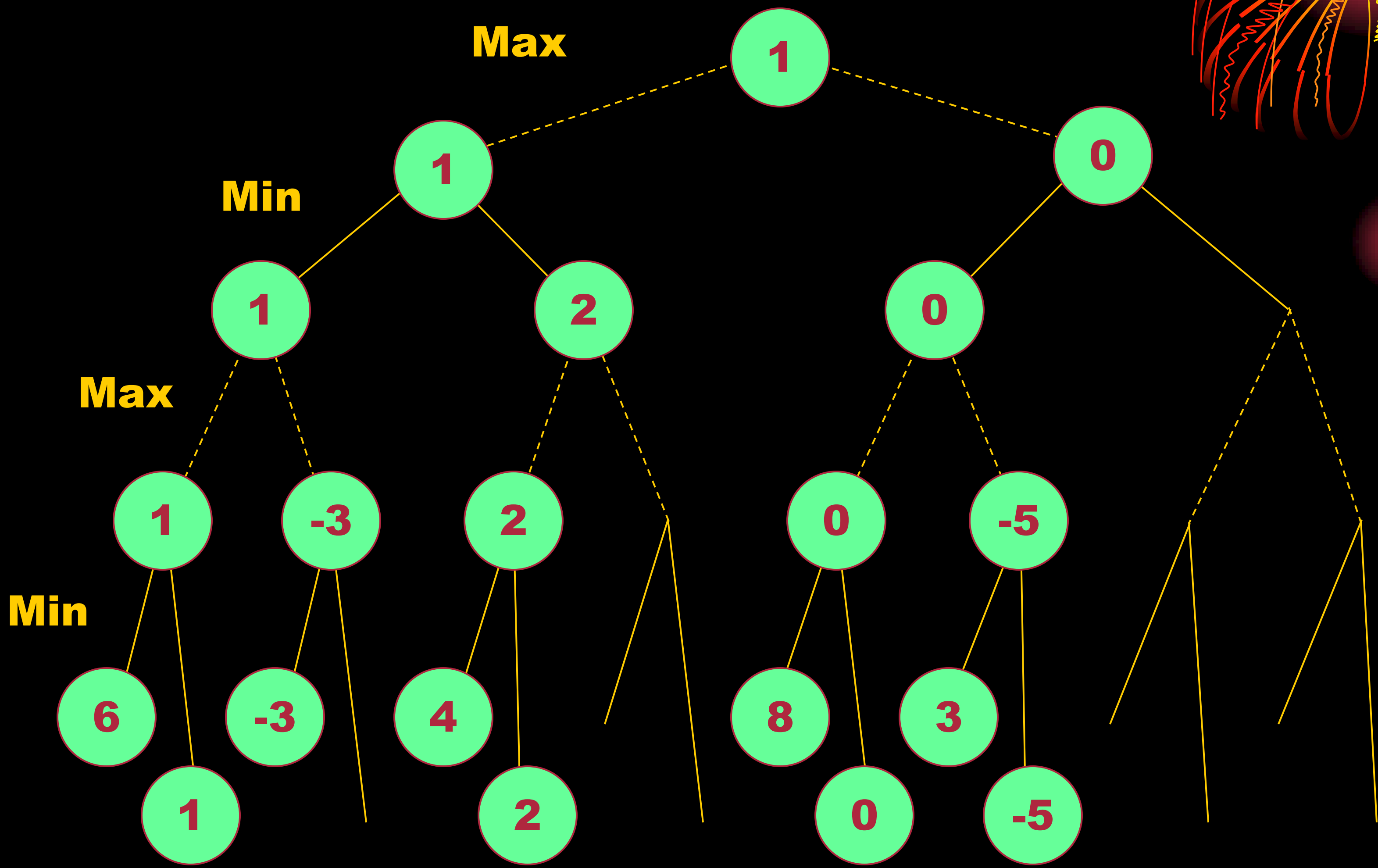
Complexity of Minimax

- Branching factor *b*
- Depth of search *d*
- Examination of b^d leaves
- Exponential time in depth of search
- Hence frequently cannot search entire tree to final positions
- Must rely on evaluation function to determine value of non-final position
- Space required = linear in depth of search

Alpha-Beta Pruning

- As a rule, deeper search leads to a higher quality of play
- Alpha-beta pruning allows game tree searches to go much deeper (twice as deep in best case)
- Pruning occurs when it is in the interests of one of the players to allow play to reach that position

Illustration of Alpha-Beta Pruning



Alpha-Beta Pruning Algorithm

max_c – Maximum possible moves (children) of a position (node)
pos – position or node of the game tree
α, β – lower and upper values of cutoff; cutoff – flag set when is OK to prune
depth – maximum search depth in the game tree
c[1 .. max.c] – children of current position (node)
val – value of each position (point of view of the root player),
width - nr. of legal moves from the current position

```
Alpha_Beta(pos, α, β, depth)                  // initialy called with α = - ∞  and  β = + ∞
  if (depth <= 0)  return (Evaluate(pos))     // Evaluate terminal node  (point of view of the root player)
  width = Generate_Moves(pos)                 // Fills array c [  ]
  if (width == 0)  return (Evaluate(pos))     // No more legal moves from this position
  cutoff = FALSE;
  i = 1
  while (i <= width)  and  (cutoff == FALSE)
    val = Alpha_Beta(c[ i ], α, β, depth-1)
    if (Max_Node(pos) and val > α)            // Root moves
      α = val
    if (Min_Node(pos) and val < β)            // Opponent moves
      β = val
    if (α > β)
      cutoff = TRUE
    i ++
  if (Max_Node(pos))  return α
  else  return β
```

Enhancement Aspiration Search

- Ordinary alpha-beta algorithm begins with pruning window $(-\infty, \infty)$ (worst value, best value)
- Pruning increases as window shrinks
- Goal of aspiration search is to start pruning sooner
- Make estimate of value v of board position
- Figure probable error e of that estimate
- Call alpha-beta with initial pruning window $(v-e, v+e)$
- If search fails, re-do with $(-\infty, v-e)$ or $(v+e, \infty)$

Enhancement Iterative Deepening

- Ply: level of a game tree
- Iterative deepening: use a (*d*–1)-ply search to prepare for a *d*-ply search
- Allows time spent in a search to be controlled: can iterate deeper and deeper until allotted time has expired
- Can use results of (*d*–*1*)-ply search to help order nodes for *d*-ply search, improving pruning
- Can use value returned from (*d*–*1*)-ply search as center of window for *d*-ply aspiration search

Parallel Alpha-Beta Search

- Perform move generation in parallel and position evaluation
  - CMU's custom chess machine
- Search the tree in parallel
  - IBM's Deep Blue
  - Capable of searching more than 100 millions positions per second
  - Defeated Gary Kasparov in a six-game match in 1997 by a score of 3.5 - 2.5

Parallel Aspiration Search

- Create multiple windows, one per processor
- Allows narrower windows than with a single processor, increasing pruning
- Chess experiments: maximum expected speedup usually not more than 5 or 6
- This is because there is a lower bound on the number of nodes that will be searched, even with optimal search window

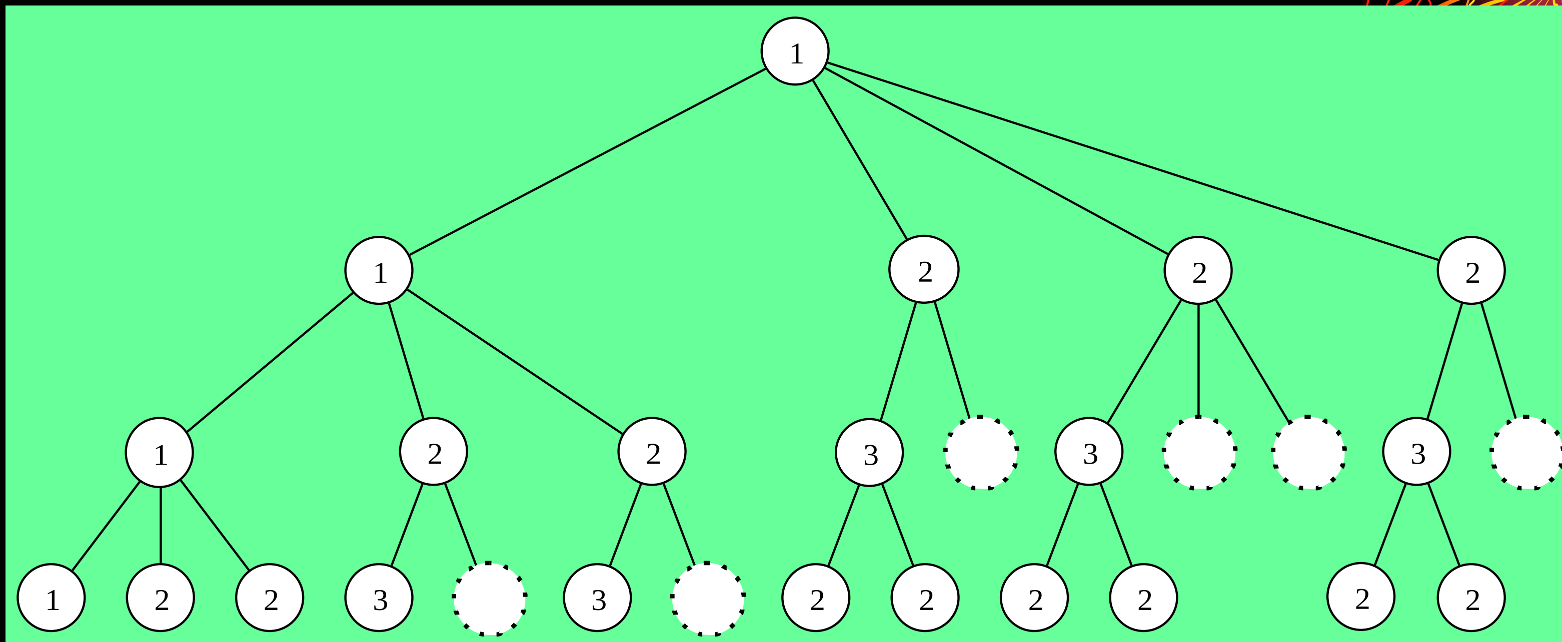
Parallel Subtree Evaluation

- Processes examine independent subtrees in parallel
- Search overhead: increase in number of nodes examined through introduction of parallelism
- Communication overhead: time spent coordinating processes performing the search
- Reducing one kind of overhead is usually at expense of increasing other kind of overhead

Game Trees Are Skewed

- In a perfectly ordered game tree the best move is always the first move considered from a node
- In practice, search trees are often not too far from perfectly ordered
- Such trees are highly skewed: the first branch takes a disproportionate share of the computation time

Alpha-beta Pruning of a Perfectly Ordered Game Tree



1 – type 1 nodes: root and first child of type 1 nodes
2 – type 2 nodes: other children of type1 nodes and children of type 3 nodes
3 – type 3 nodes: first child of type 2 nodes
The other than the first child of a type 2 node can be pruned

Distributed Tree Search

- Processes control groups of processors
- At beginning of algorithm, root process is assigned root node of tree and controls all processors
- Allocation of processors depends on location in search tree

Distributed Tree Search (cont.)

- Type 1 node
  - All processors initially allocated to search leftmost child of node
  - When search returns, processors assigned to remaining children in breadth-first manner
- Type 2 or 3 node: processes assigned to children in breadth-first manner
- When a process completes searching a subtree, it returns its allocated processors to its parent and terminates
- Parents reallocate returned processors to children that are still active

Performance of Distributed Tree Search

- Given a uniform game tree with branching factor *b*
- If alpha-beta algorithm searches tree with effective branching factor b^x, then DTS with p processors will achieve a speedup of $O(p^x)$
- Usually *x* is between 0.5 and 1

Summary (1/5)

- Combinatorial search used to find solutions to a variety of discrete decision and optimization problems
- Can categorize problems by type of state space tree they traverse
- Divide-and-conquer algorithms traverse AND trees
- Backtrack search and Branch-and-Bound search traverse OR trees
- Minimax and alpha-beta pruning search AND/OR trees

Summary (2/5)

- Parallel divide and conquer
  - If problem starts on a single process and solution resides on a single process, then speedup limited by propagation and combining overhead
  - If problem and solution distributed among processors, efficiency can be much higher, but balancing workloads can still be a challenge

Summary (3/5)

- **Backtrack search**
  - Depth-first search applied to state space trees
  - Can be used to find a single solution or every solution
  - Does not take advantage of knowledge about the problem to avoid exploring subtrees that cannot lead to a solution
  - Requires space linear in depth of search (good)
  - Challenge: balancing work of exploring subtrees among processors
  - Need to implement distributed termination detection

Summary (4/5)

- **Branch-and-bound search**
  - Able to use lower bound information to avoid exploration of subtrees that cannot lead to optimal solution
  - Need to avoid search overhead without introducing too much communication overhead
  - Also need distributed termination detection

Search (5/5)

- **Alpha-beta pruning:**
  - **Preferred method for searching game trees**
  - **Only parallel search of independent subtrees seems to have enough parallelism to scale to massively parallel machines**
  - **Distributed tree search algorithm a way to allocate processors so that both search overhead and communication overhead are kept to a reasonable level**