CS60045 Artificial Intelligence Autumn 2023

AND-OR Search

Problem Decomposition

So far we have found a single (best perhaps) path from source to goal.

At every node, we have several options dictated by the possible actions that we can take.

We follow **one** of these transitions (the best one) to go to the goal.

In some situations, we may have to follow multiple paths from a given node.

It is needed that **all** the transitions must be taken to reach the goal.

We can solve a given problem if we can solve several subproblems simultaneously.

Making only one of the transitions does not solve the given problem.

AND-OR Graphs

Suppose that we want to solve a problem *A*.

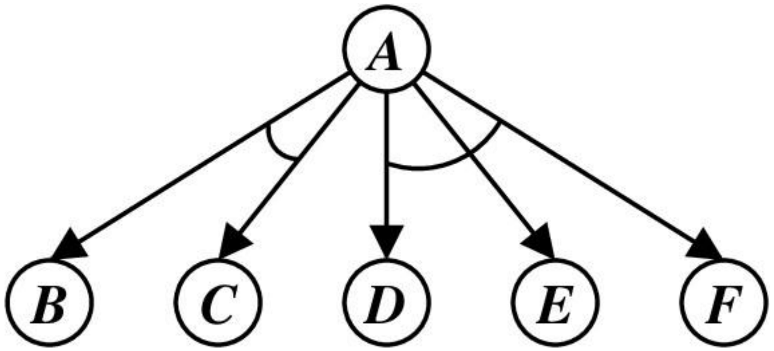
We know that *A* can be solved if either

both the two subproblems *B* and *C* can be solved

or

all of the subproblems *D*, *E*, *F* can be solved

We represent this situation like this:



Example

Problem decomposition is relevant in the context of automated proof/deduction techniques.

Suppose that we want to compute the integral $\int \big(f(x) + g(x)\big)\, h(x)\, dx$. Here are a few (not all) immediate possibilities.

- Make a substitution and continue to solve a single integration.
- Solve two integrations $\int f(x)h(x)\, dx$ and $\int g(x)h(x)\, dx$, and add them.
- Use integration by parts

  $$\int \big(f(x) + g(x)\big)\, h(x)\, dx = (f(x) + g(x))\int h(x)\, dx - \int \big(f'(x) + g'(x)\big) \left(\int h(x)\, dx\right) dx$$

  which involves three integrations, one (or two, if you will) differentations, and a few other operations.

Solving a problem by solving only a single subproblem does not necessarily work.

Search in an AND-OR graph

In an AND situation, we need to solve all the subproblems. The cost of the problem is the sum of the costs of all the subproblems plus the cost of subproblem generation.

In an OR situation, solving any one of the subproblems will do. We choose the cheapest one. If one subproblem is solved, and the other subproblems have a promise of arriving at a cheaper solution, the search should continue. For each subproblem, the cost of generating only that subproblem should be considered.

There are multiple problems to solve, so goal nodes are not meaningful. Instead we have:

- **Terminal nodes:** The solutions of these nodes are known, so too are the costs of solving them.
- **Dead ends:** No progress can be made from these nodes. These may be considered as terminal or non-terminal nodes with infinite costs.

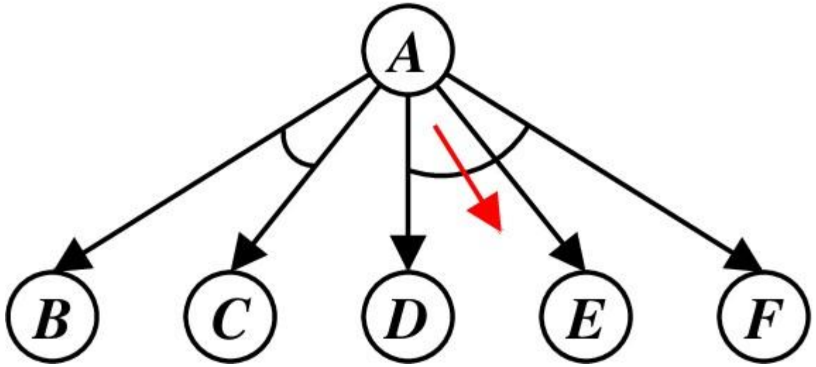
AO* search (an adaptation of A* search) works for AND-OR graphs.

AO* search: Overview

Continue until the initial problem *S* is solved (or found unsolvable by the search):

- Consider the currently most promising path *P* from *S*. This path may contain both AND and OR expansions. So *P* is actually a tree (more correctly, a graph).
- Collect all the open nodes on *P*.
- Pick the node *n* with the smallest *h*() value. (*f*() values are not used in this algorithm.) **Note:** Choosing any unexpanded node on *P* will do.
- Expand *n* to get all successor (child) nodes of *n*. Compute the *h*() values of all the successors.
- Propagate the effects of these “new” nodes up the graph to all the ancestors of *n* on *P*.
- If the propagation process discovers certain nodes as solved, mark them as solved.
- Update the most promising path as revealed by the propagation process.

Recalculating *h* ( ) values during propagation up the graph



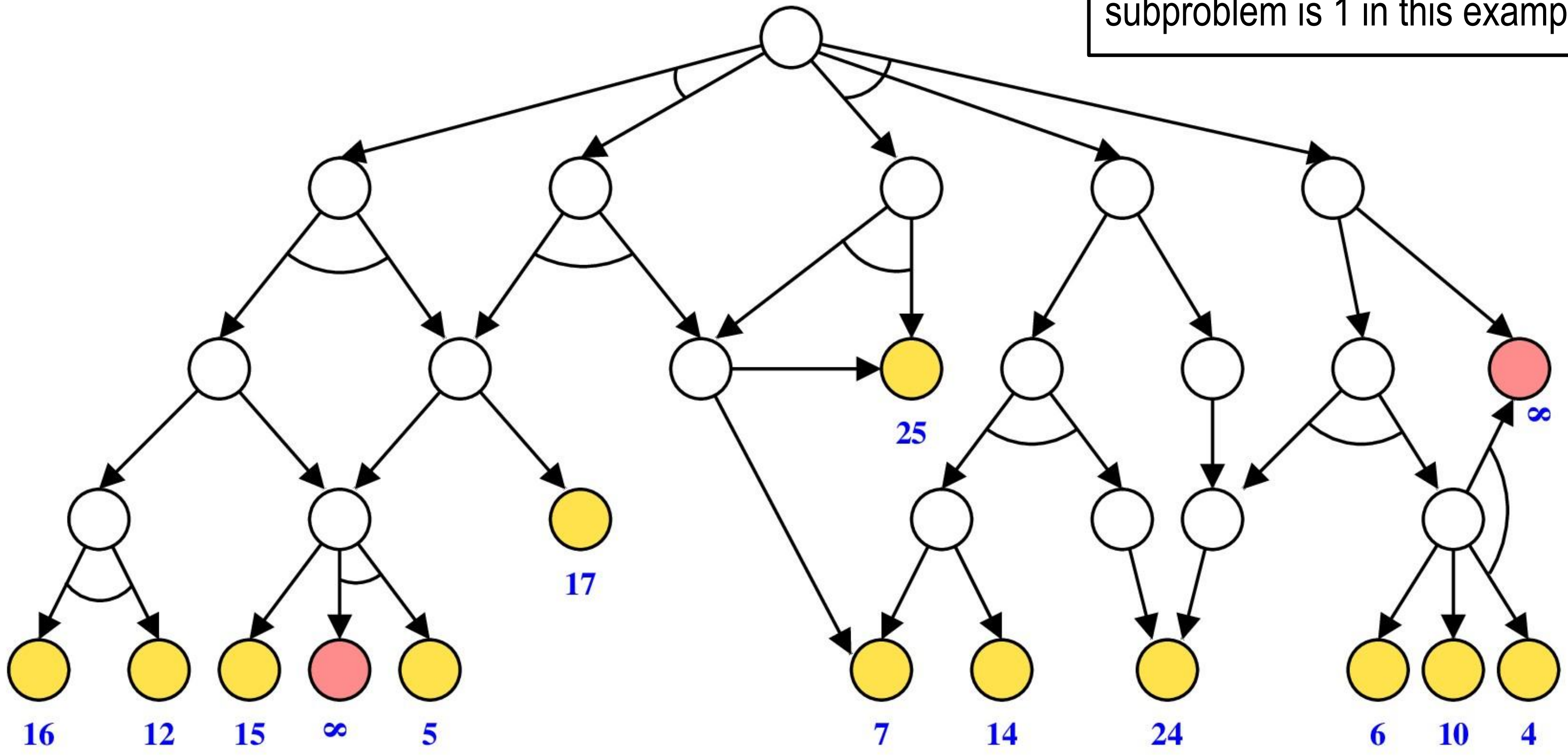
$$h(A) = \min\left(c(A,B) + c(A,C) + h(B) + h(C),\ c(A,D) + c(A,E) + c(A,F) + h(D) + h(E) + h(F)\right)$$

The red arc denotes that the expansion happened in the *DEF* subtree.

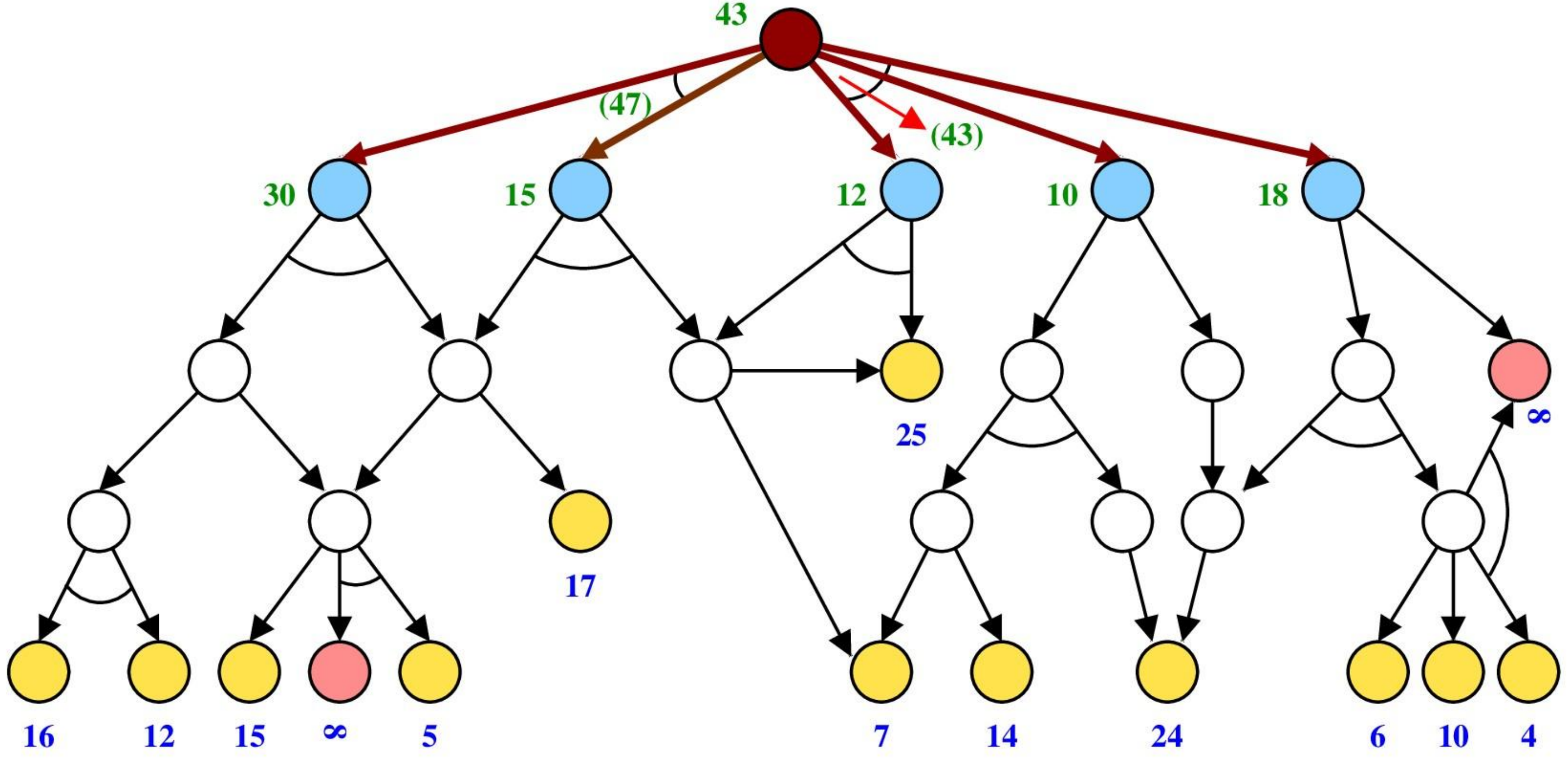
The estimates $h(B)$ and $h(C)$ may change too because of the expansion.

AO* Search: Example

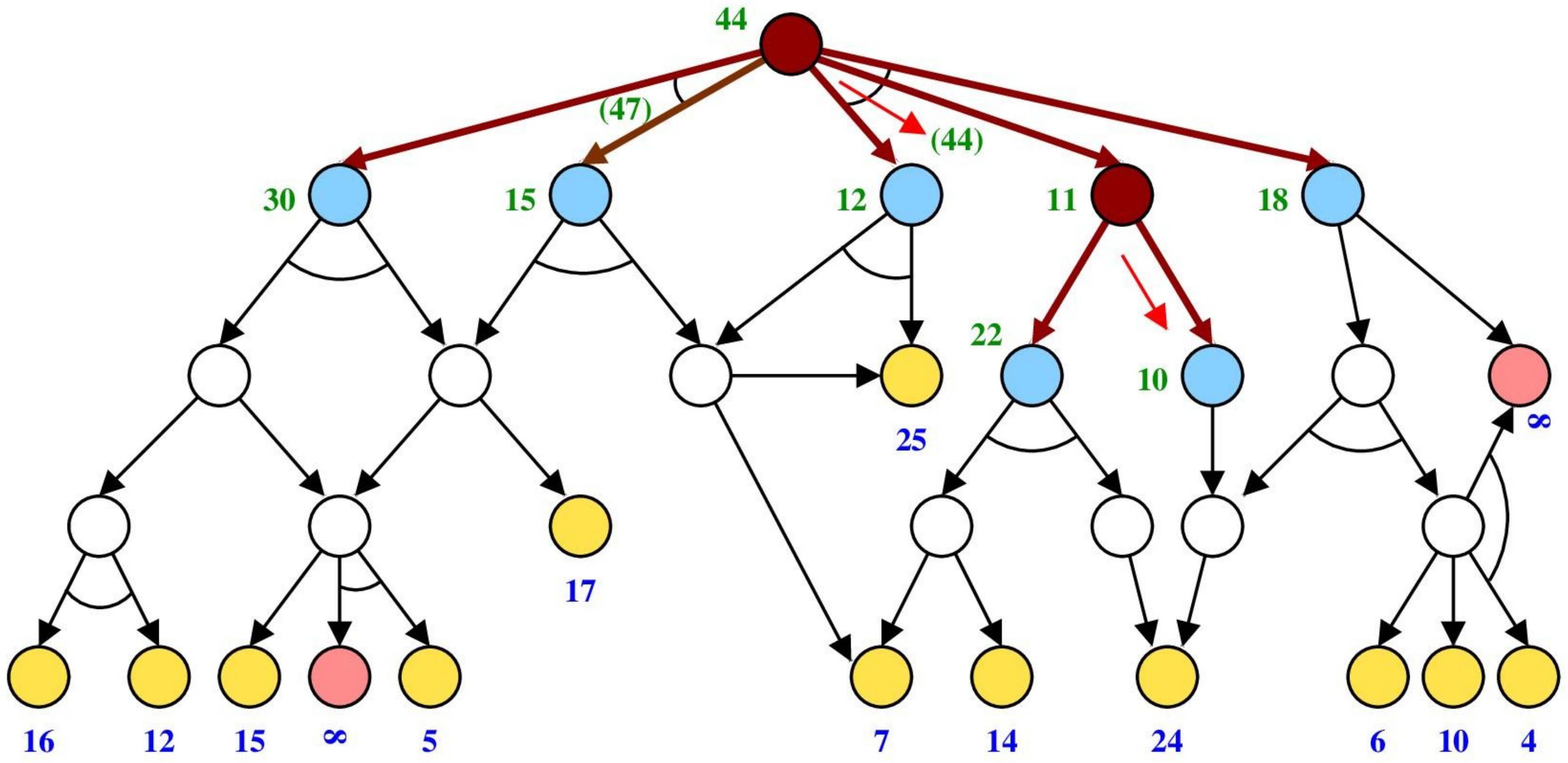
Cost of generating each subproblem is 1 in this example.



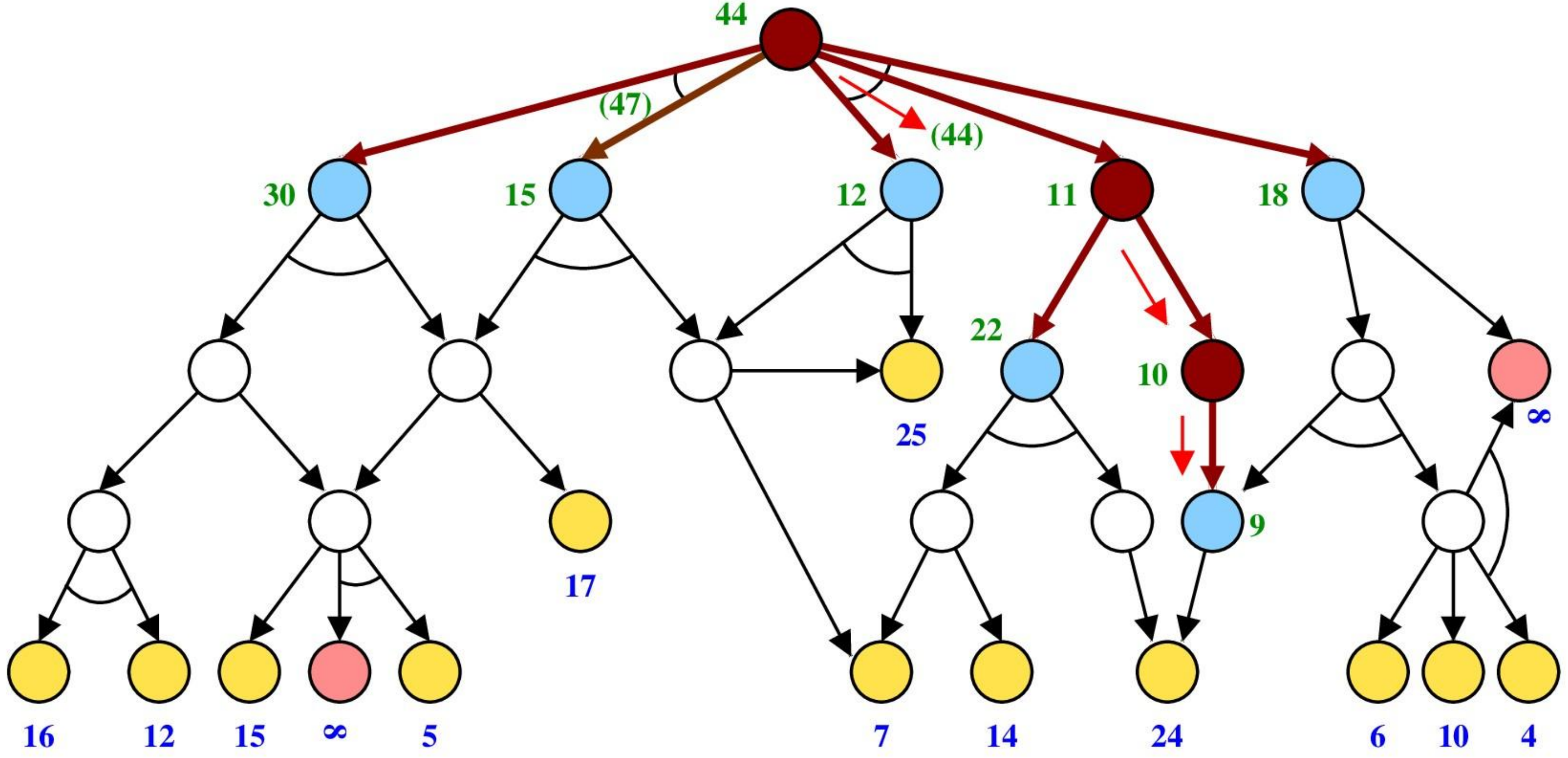
AO* Search: Example



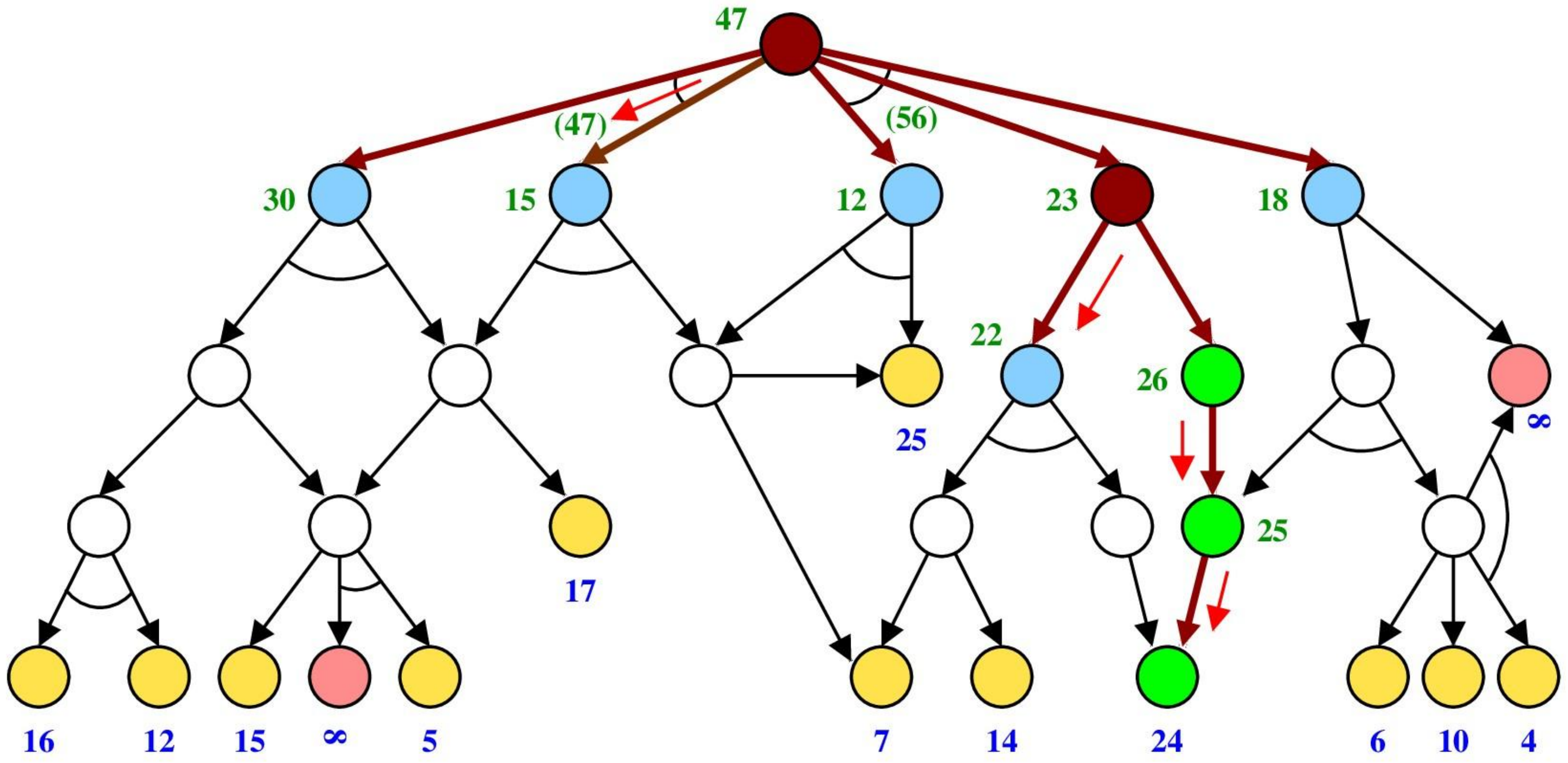
AO* Search: Example



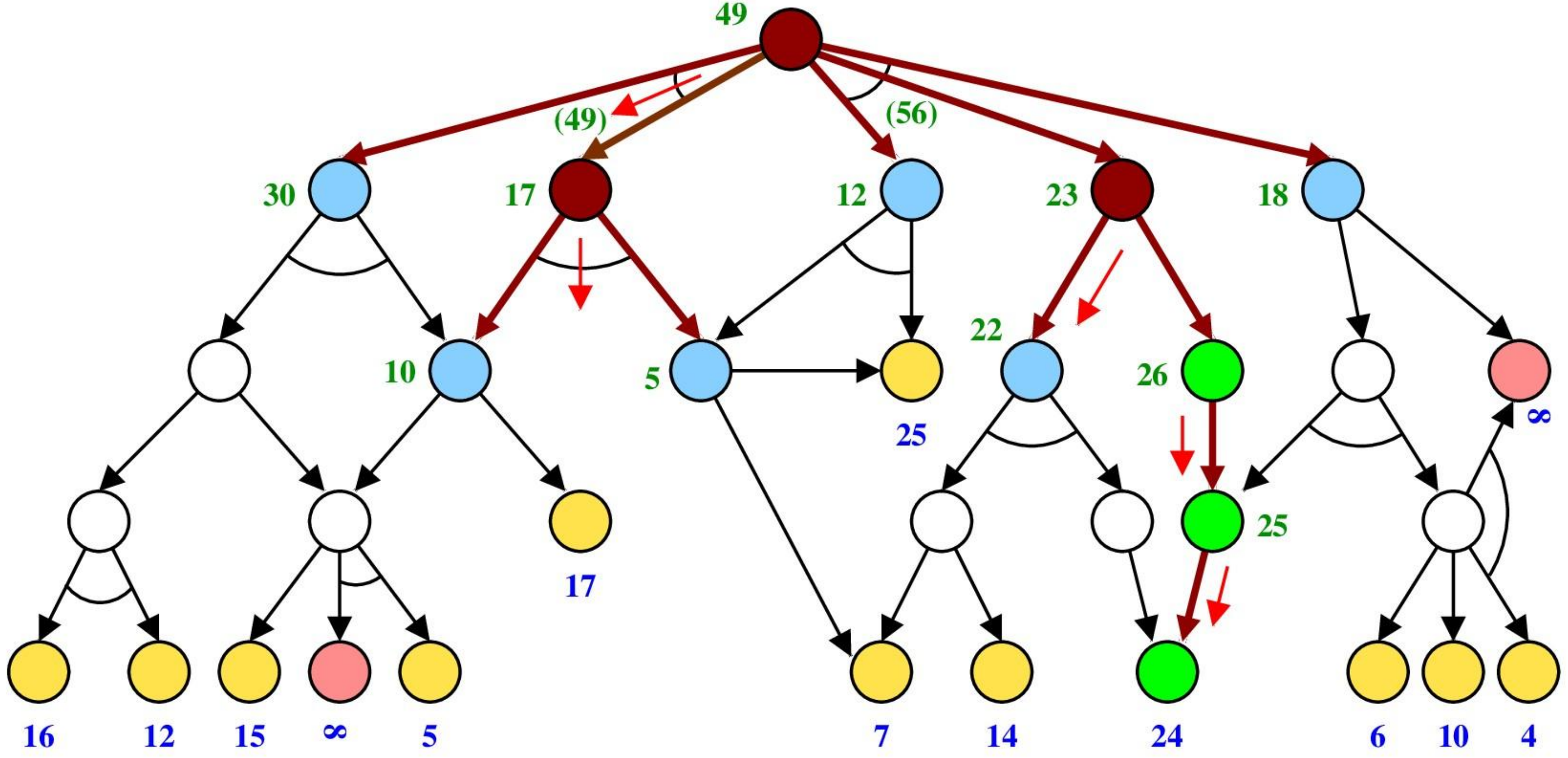
AO* Search: Example



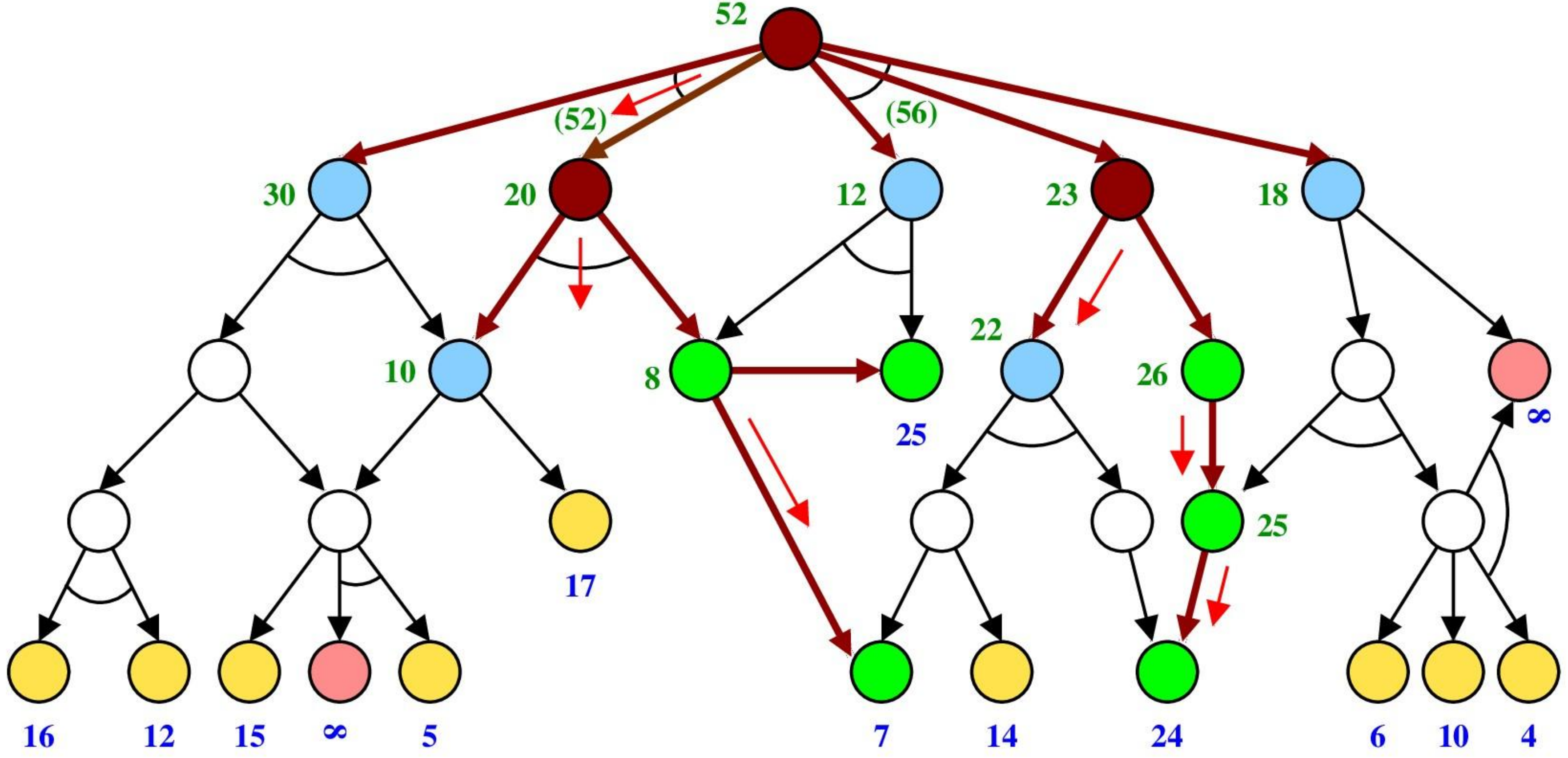
AO* Search: Example



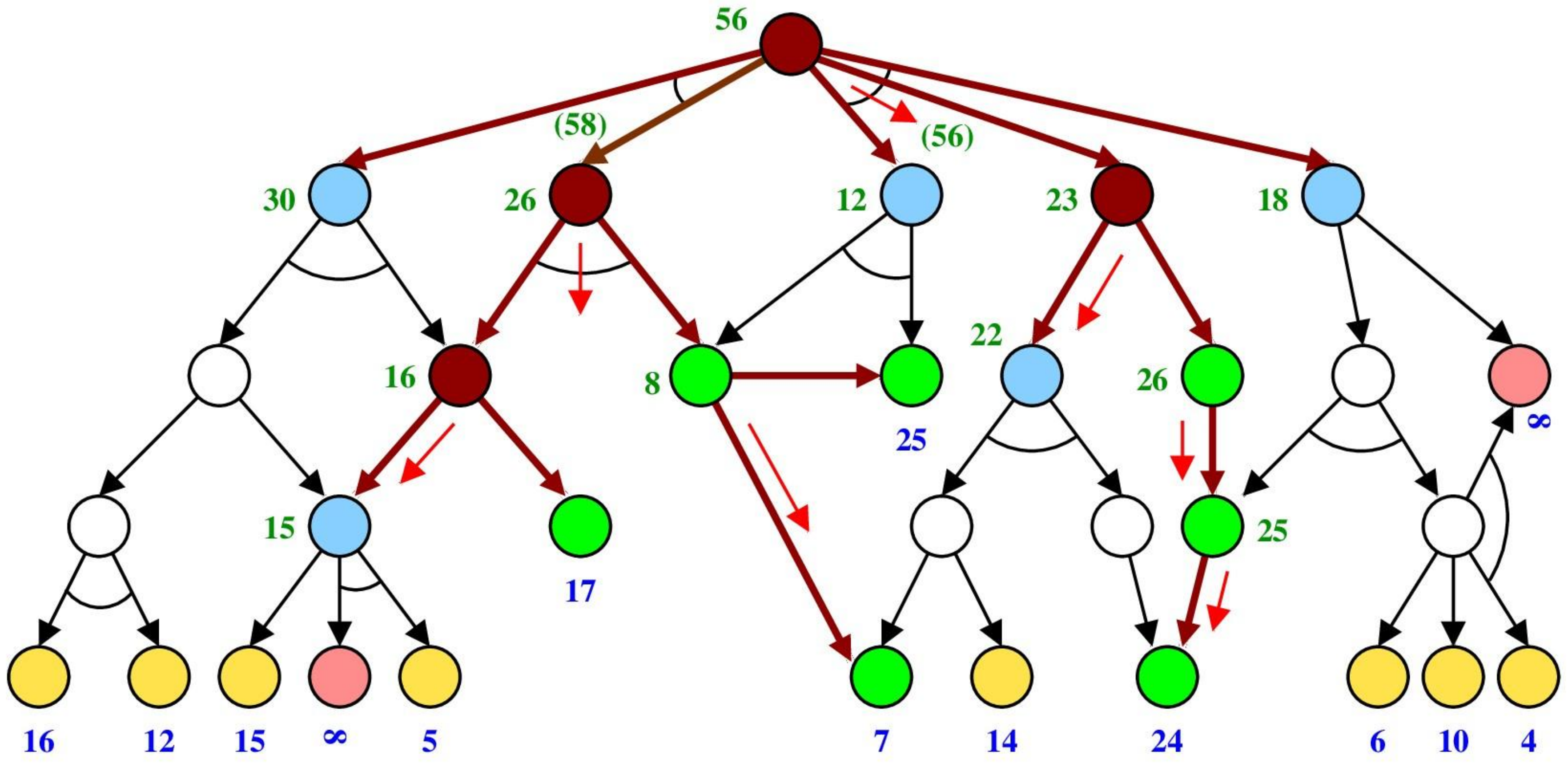
AO* Search: Example



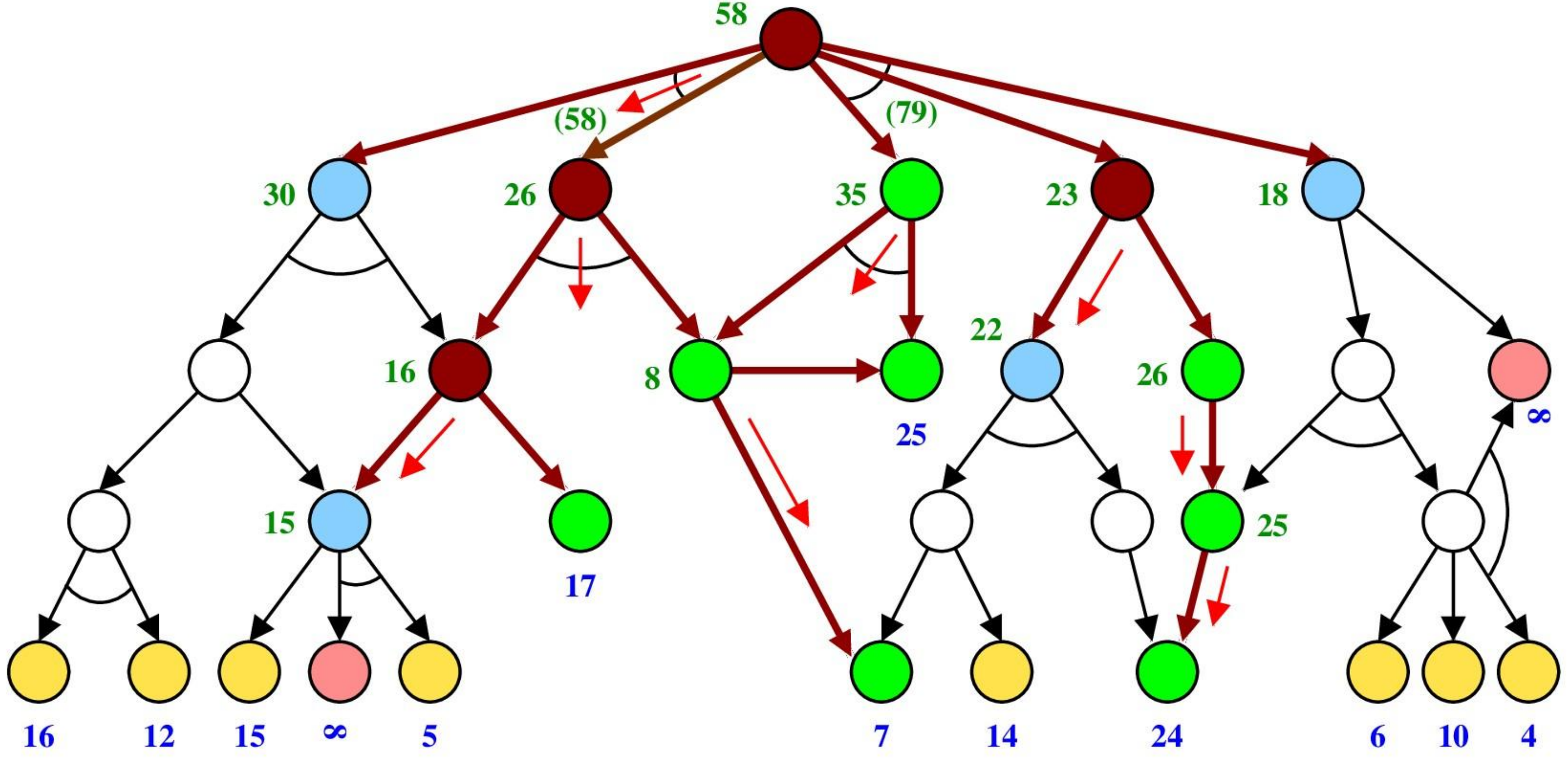
AO* Search: Example



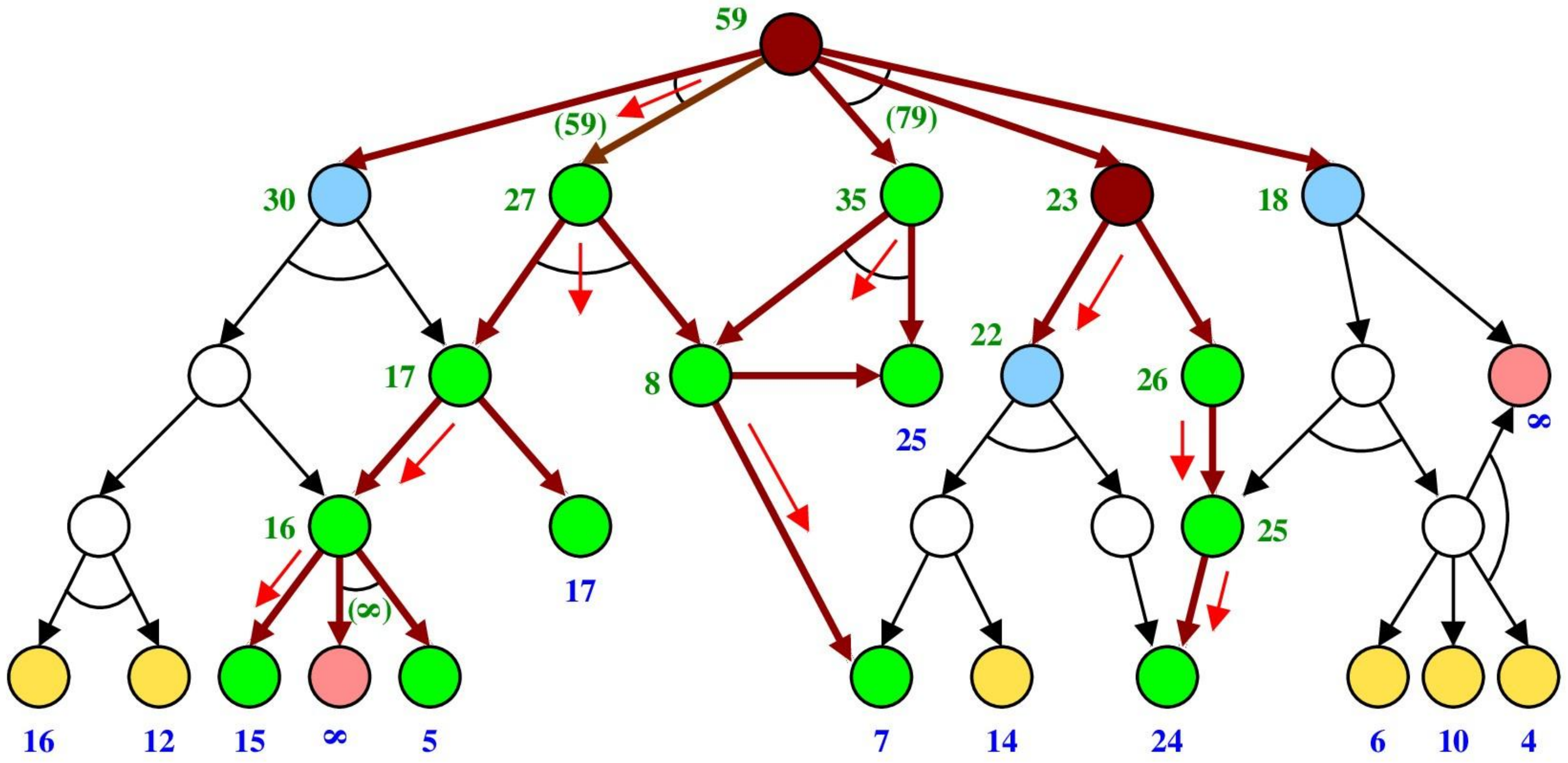
AO* Search: Example



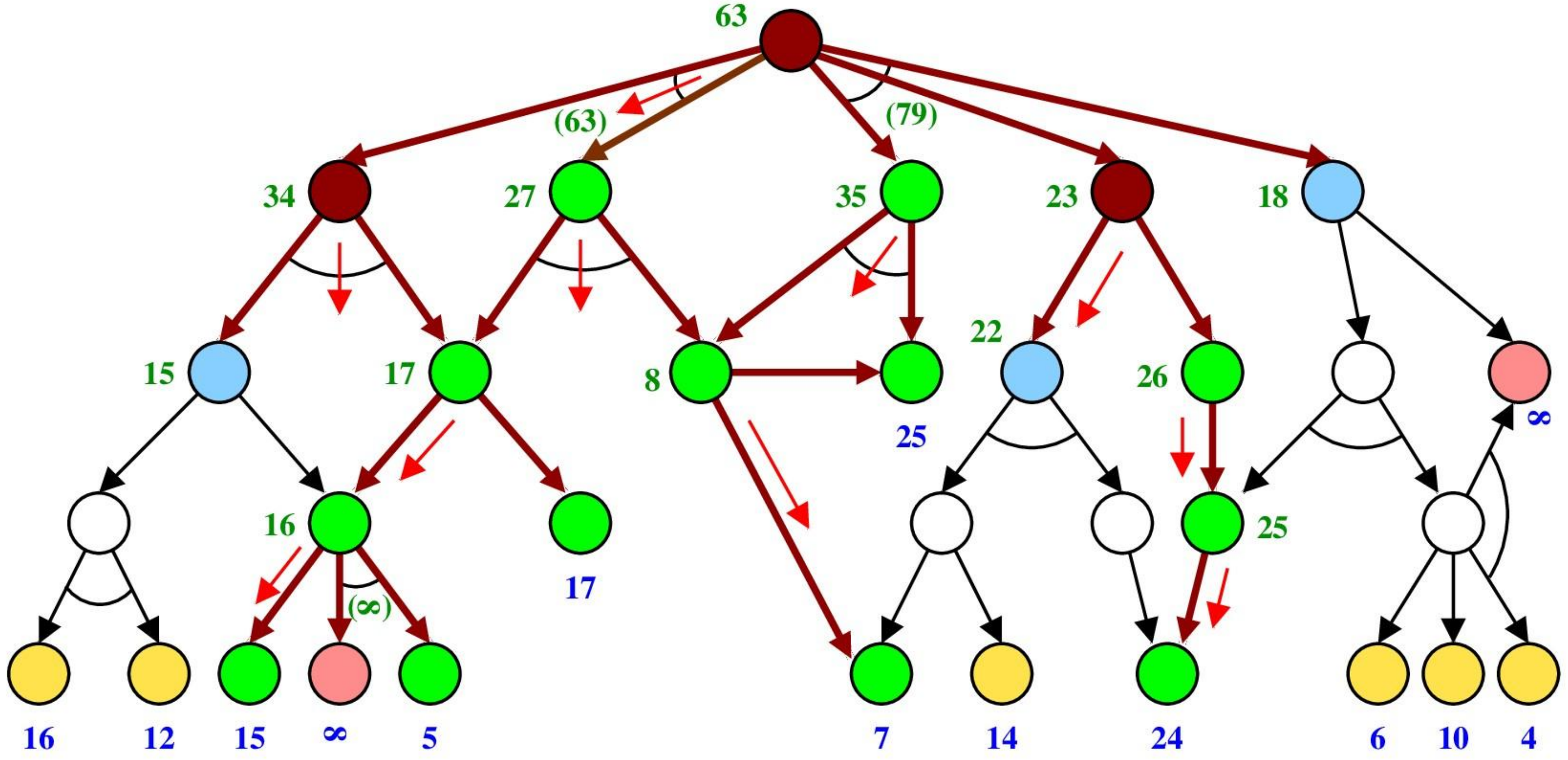
AO* Search: Example



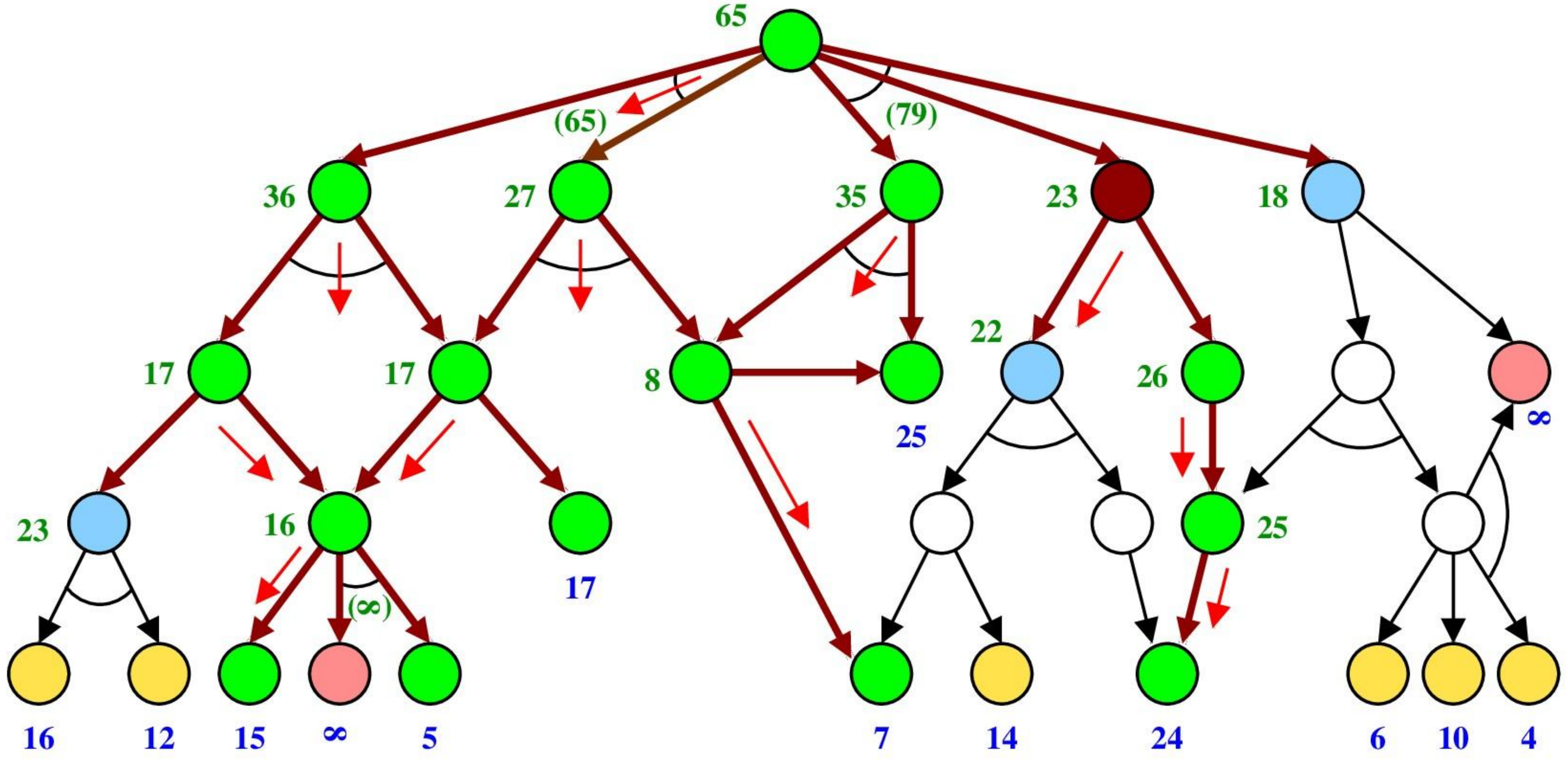
AO* Search: Example



AO* Search: Example



AO* Search: Example



AO* Search: Detailed Algorithm

1. Initialize the explored graph *G* with only the initial node *S*.
2. If *S* is marked ***solved***, return.
3. Collect all the unexpanded nodes on the currently preferred (marked) path from *S* in *G*.
4. Pick any node *n* (like the one with the smallest *h* value) from the collection.
5. Expand *n* to get all its successors. Update *G*.
6. Compute the *h* values of all the successors of *n*.
7. Update the cost estimates in *G* as follows:
   (a) Initialize *Q* to contain the single node *n*.
   (b) Pick any node *m* from *Q* such that *m* has no descendents in *Q*. Remove *m* from *Q*.
   (c) Update *h*(*m*) (and the path marking from *m* if necessary). If a complete best solution from *m* is found, mark *m* as ***solved***.
   (d) If *h*(*m*) changes or the status of *m* changes to ***solved***, add all immediate ancestors of *m*, of which *m* is the best current option.
   (e) Go to Step (b) if *Q* is not empty.
8. Go to Step 2.

AO* Search: Properties

Theorem: AO* search is complete in a finite AND-OR graph.

Two properties of the heuristic function *h*

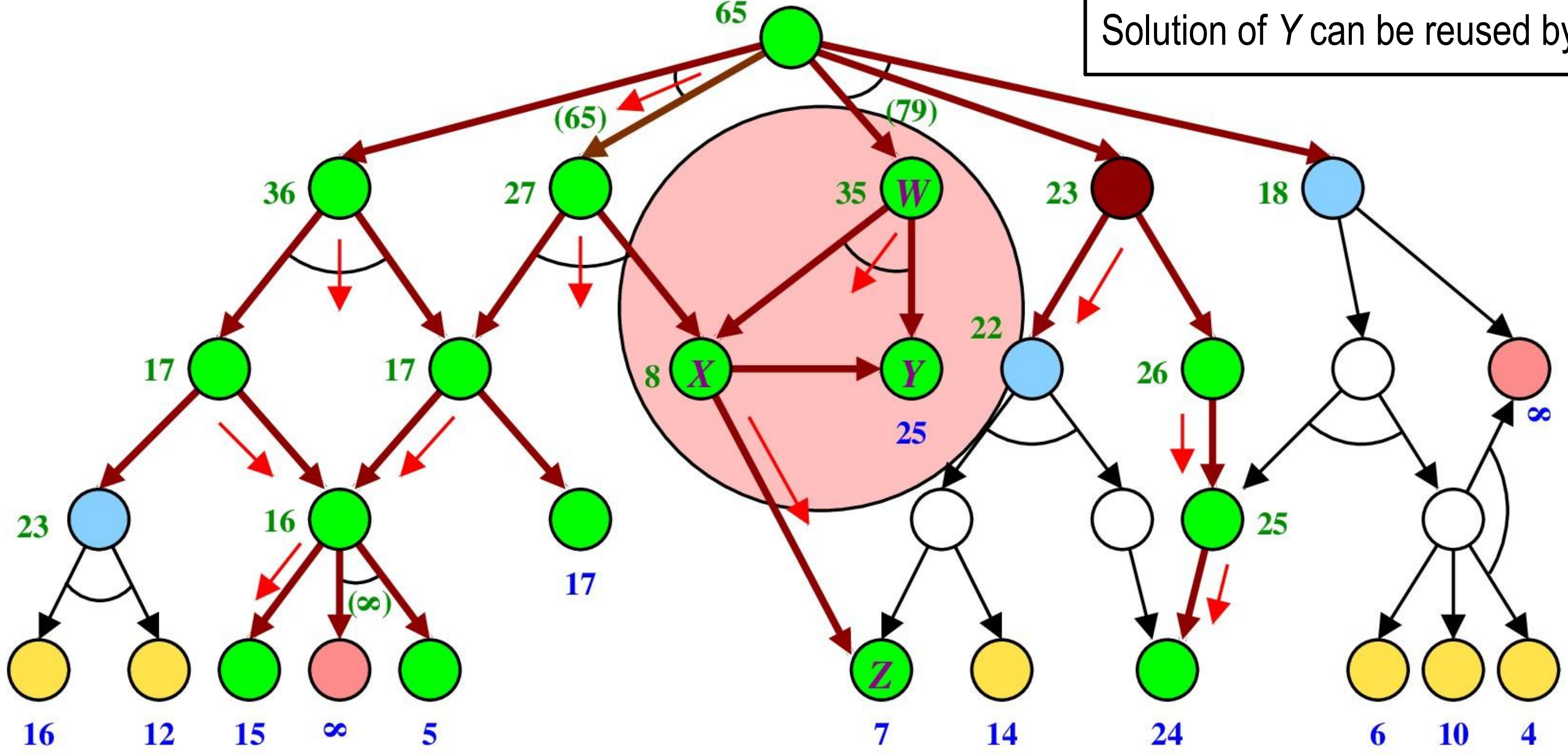
- **Monotonicity:** $h(n)$ never decreases during the expansion process.
- **Admissibility:** $h(n) \leqslant h^*(n)$

Theorem: If *h*() is monotonic and admissible, then AO* search produces a cost-optimal solution.

Proofs are involved, and are omitted.

Limitation of AO* Search: Interacting Subproblems

No need to solve *Z* to solve *X*.
Solution of *Y* can be reused by *X*.


65
(65)
(79)
36
27
35
W
23
18
17
17
8
X
Y
25
22
26
∞
23
16
17
25
(∞)
Z
16
12
15
∞
5
7
14
24
6
10
4