Branch and Bound Depth search algorithm is normally dedicated towards a different type of problem in A.I which is the **TSP (Travelling Salesman Problem)** which is given a set of cities as a node and is returning to starting city after conducting a tour of all the cities in the shortest time it might be possible to use some of these techniques for us.

**Search Techniques**

**Lowest Cost First Search (LCFS)**

-> Expands the path with the lowest cost frontier

-> normally implemented with the data structure of a priority queue ordered by path cost

He-> similar to dijkstra's algorithm

So we involve a heuristic search which basically makes an estimate of the optimal (Cheapest) path from node n to goal node.

**Best First Search ( What we were looking into doing at the previous meeting)**

**->** Expands the path of the lowest h value at the front most node

-> Once again this can be implemented using the data structure of a priority queue

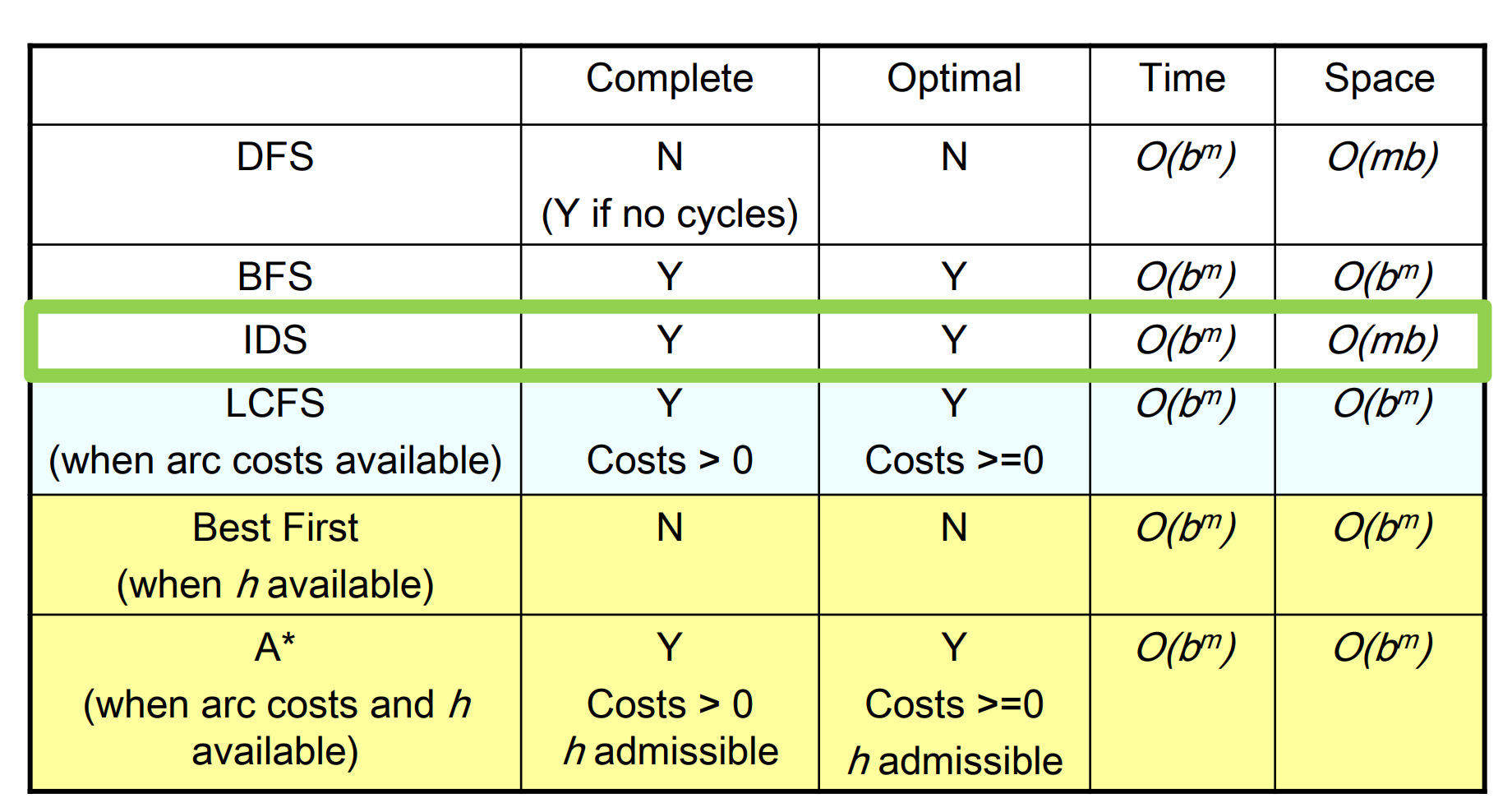
-> But this constitutes to a greedy algorithm since it expands path which leads to the goal the quickest. Problems that arise are that it can get trapped , can also yield a poor solutions but if with a great heuristic it can achieve the goal.

**A\***

-> Expands the with the lowest cost plus the h value on the frontier node

-> Once again A\* can be implemented using the priority queue ordered by **f(p) = cost(p) + h(p)**

**Various different search methods**

****

Looking at the various other methods it might be worth it looking at others that might be effective since a DFS is rated as the worst in doing the search.

**Heuristic DFS**

-> when expanding a node it would be available to put all the neighbours of that node into a stack, normally the order would use heuristic guidance and with a perfect heuristic it wouldn’t require an backtracking.

-> great for finding just a solution but it becomes a little difficult to find an optimal solution.

**Branch and Bound DFS**

**->** follows the same search path as depth first search , but to ensure its optimality it doesn’t stop at the first solution and keeps going.

-> continues onwards after finding the upper bound on the solution cost , (Upper bound = cost of the best solution so far) , initially that is infinite.

-----------------------------------------------------------------------------------------------------------------

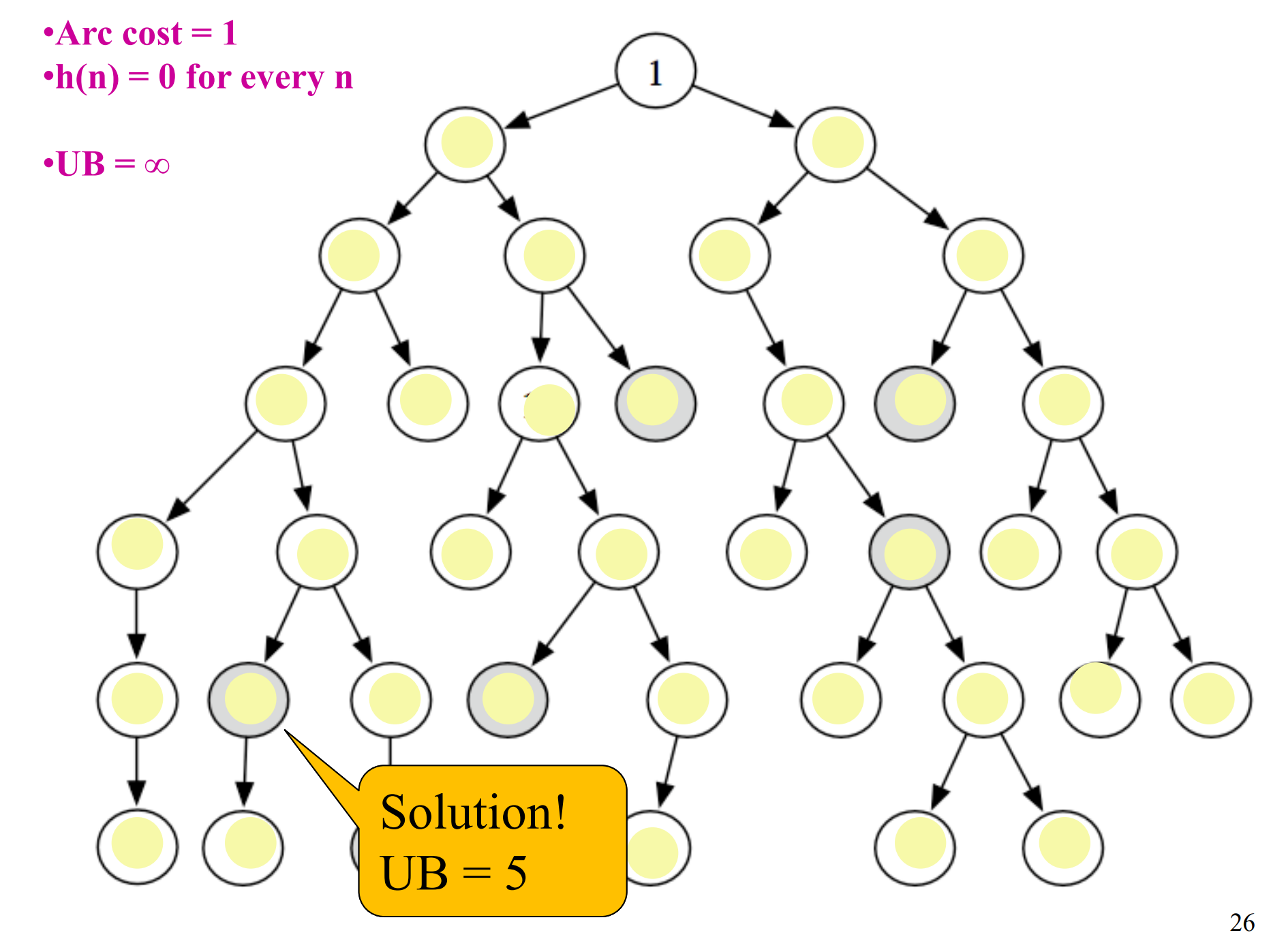
When the path p is selected for expansion, compute the Lowerbound (p) = f(p) = cost(p) + h(p)

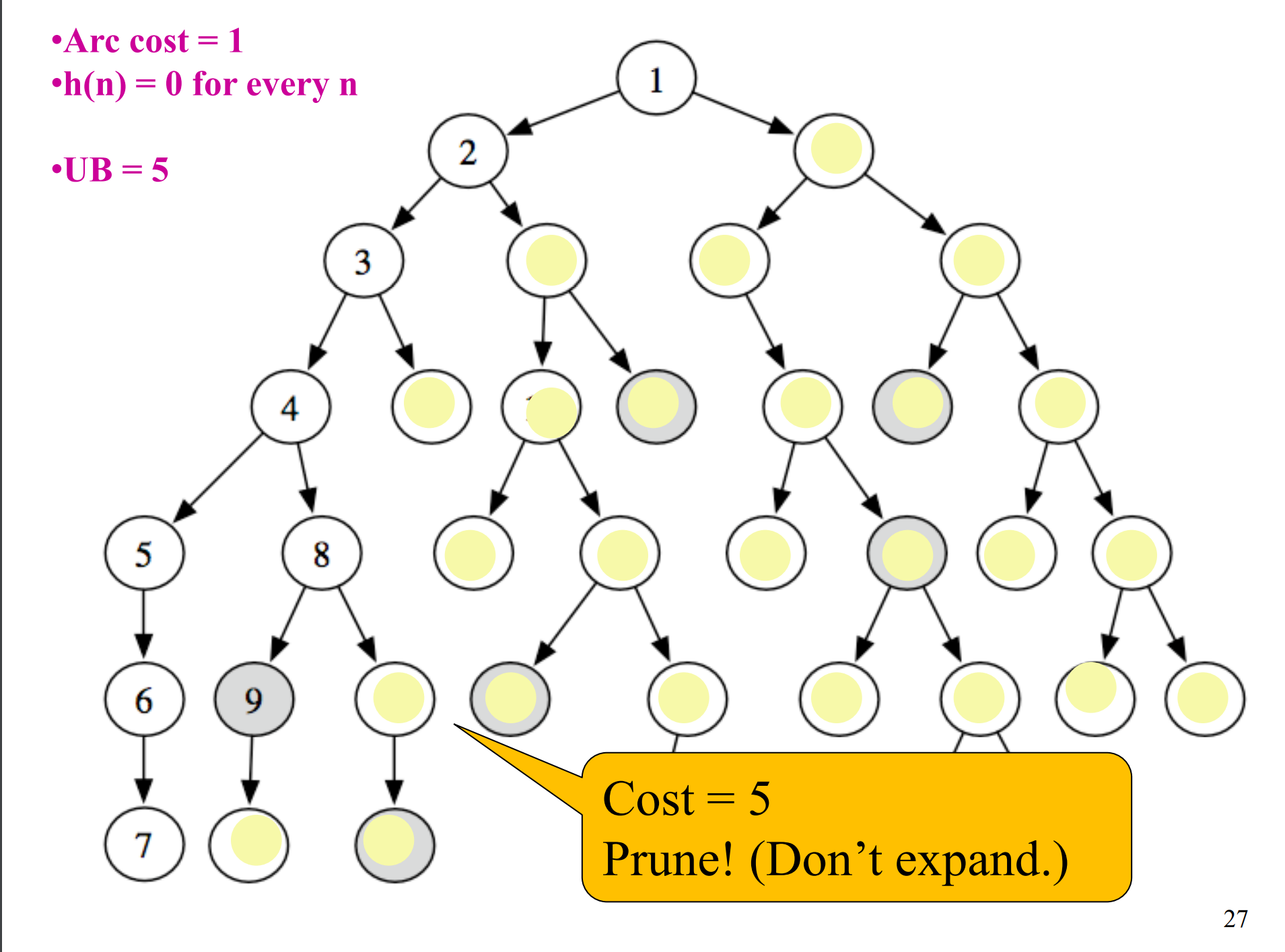
-> if the LB >= UB , remove p from the frontier without expanding it (pruning)

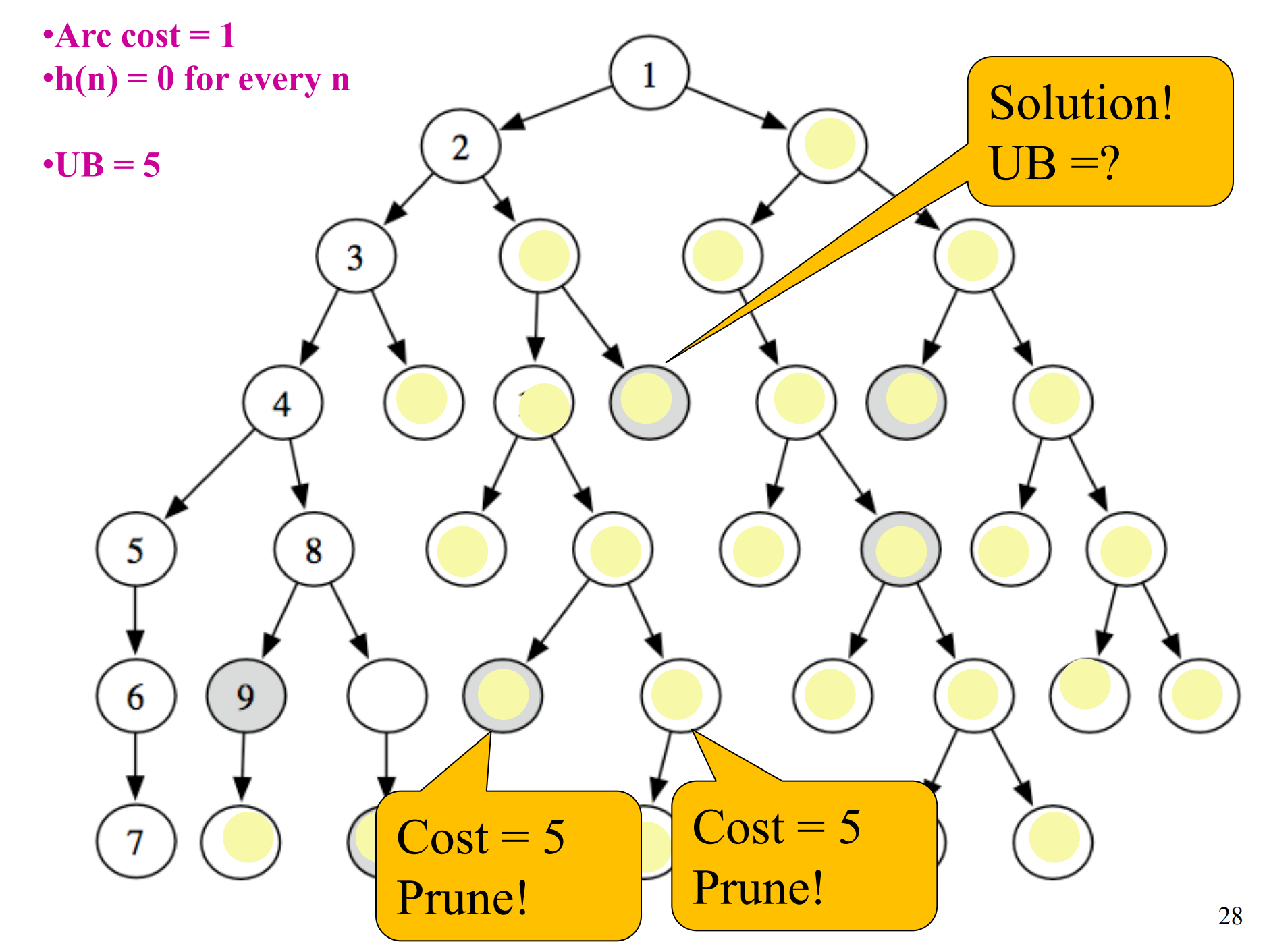
-> otherwise expand p adding all of its neighbours to the frontier.

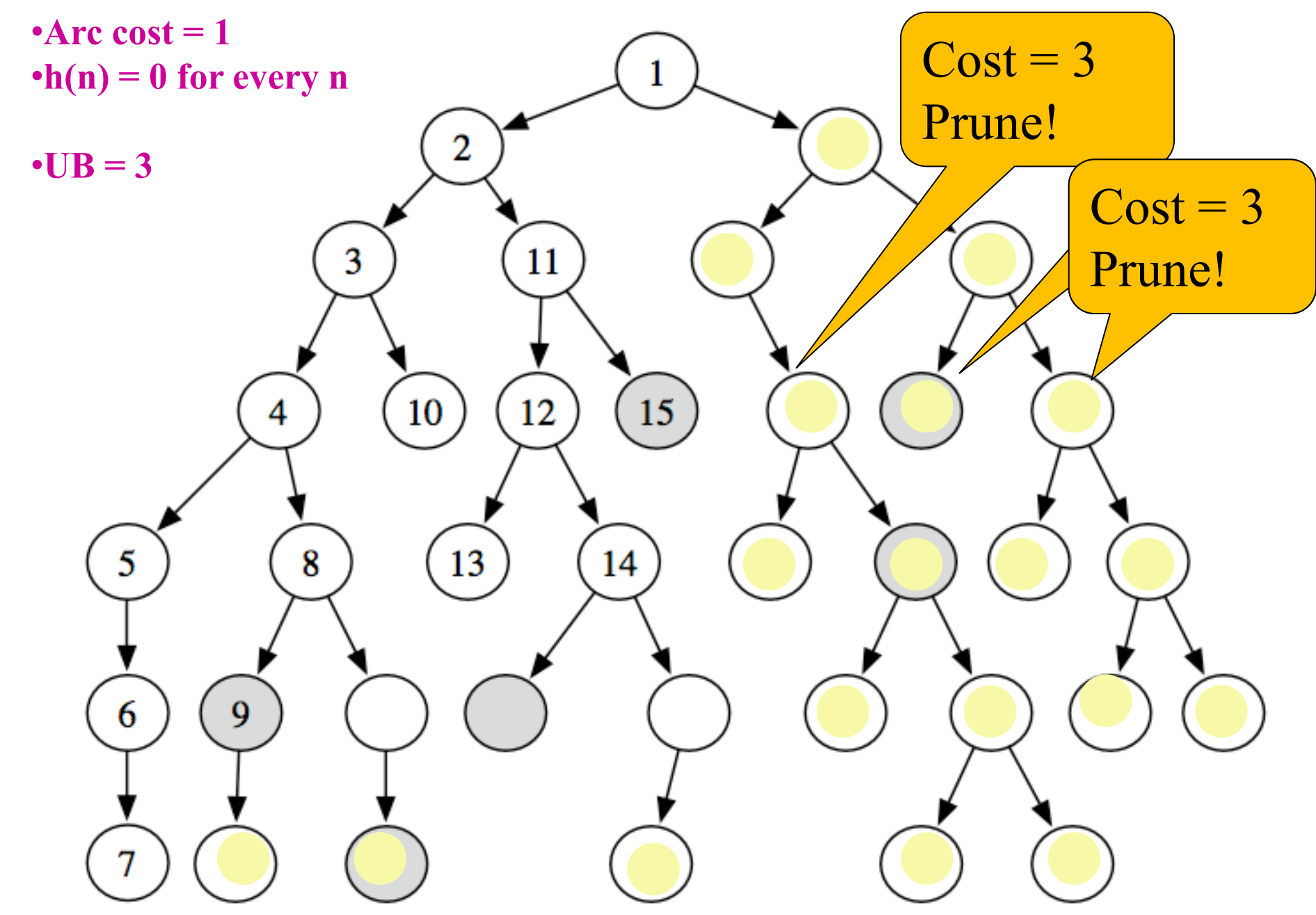
-----------------------------------------------------------------------------------------------------------------

Example with pictures is shown below:









**Memory Bounded A\***

**->** the main problem with A\* is that it cause the system to run out of memory , therefore their must be adequate methods to work around the memory problem here are some methods that were found online:

-> when a particular memory threshold is passed , delete the worst path (highest f value from the frontier) and back the f value up with a common ancestor but that subtree is only regenerated when all other paths have been shown to be “worse” than the path which was just forgotten.

Do we have enough time to conduct parallelization and pruning together? Should we focus on one or the other first so we can focus on parallelization first and then move on to pruning or look at pruning first and then move on to parallelization?

-> If wanting only one optimal path pruning is a must :P , other pruning for optimality we can use what was shown above in a simpler context:

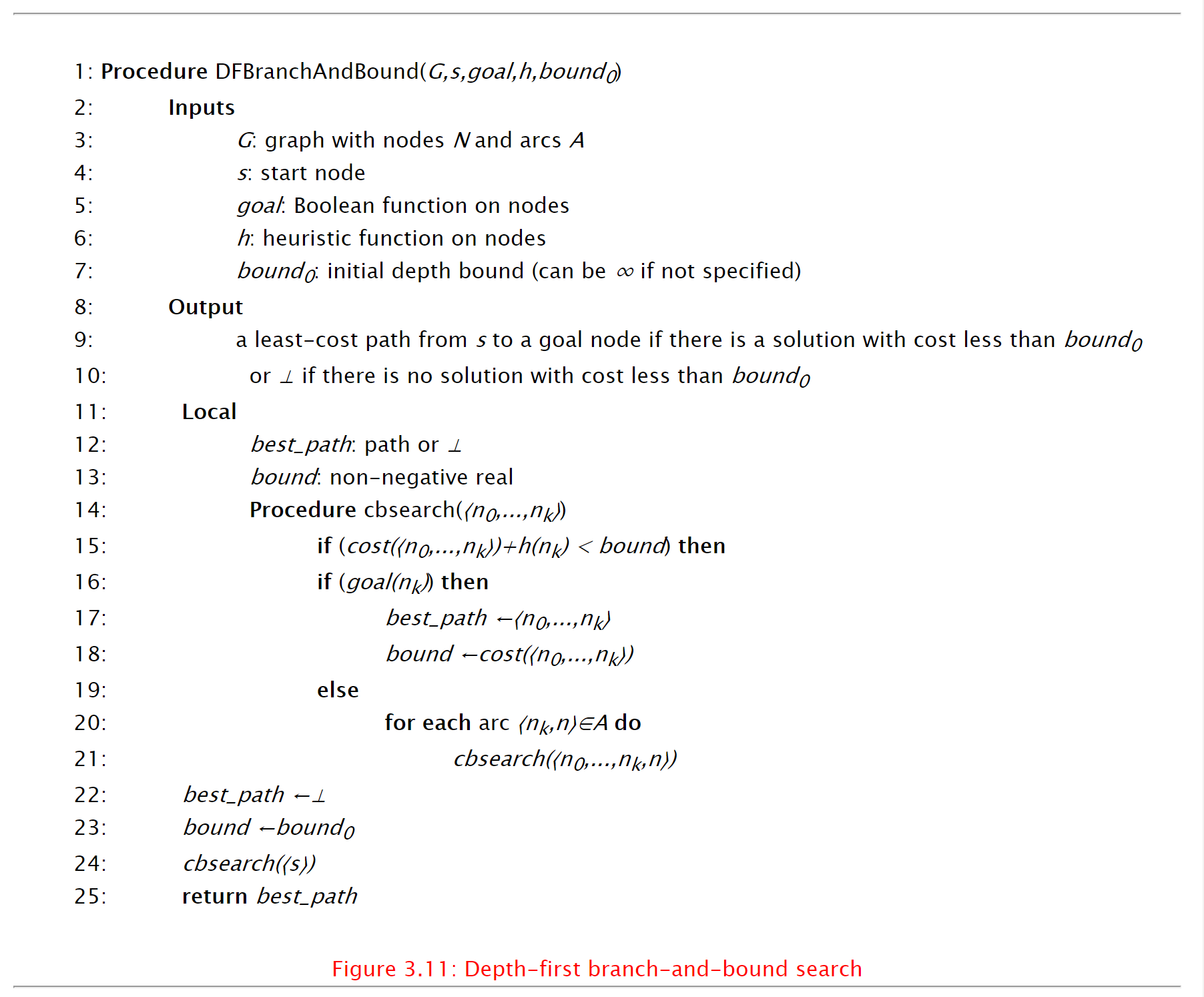
-----------------------------------------------------------------------------------------------------------------

If cost (p) < cost(p’)

* Remove paths from the frontier with prefix p’ or replace those paths with the prefix of p’ to be p

-----------------------------------------------------------------------------------------------------------------

In general (LCSF - least cost search first) can be used to prove that the first path which is found is optimal.



The is a pseudo code for a branch and bound algorithm DFS Algorithm. I think for everyone to get a go at back end and understand the algorithms thoroughly we can do split so we can have our main backend people being Oli, Yvi and Ray working on the DFS algorithm and making sure we have solid working algorithm for submission while Me and Kevin not really doing much coding this week based on planning visualization and looking into graph stream can start on the A\* equivalent alongside you guys on a different branch and package.

<https://artint.info/html/ArtInt_63.html>

**Visualizations:**

* We want the node value , path value and the node number on the graph
* Zoom function?
* Able to visualize the path , works concurrently with the algorithm as it runs
* Different types of visualizations.

<http://www.cs.umsl.edu/~sanjiv/classes/cs5130/lectures/bb.pdf>

Extensive information on Parallel Branch and Bound algorithms:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.2645&rep=rep1&type=pdf>

**Graphstream visualization:**

The Graphstream comes with a normal viewer and any special viewers would have to be downloaded separately

* The viewer is considered to be stable and is made in pure Java but it has a customizability which means you can change the size and colours. Edge colours and width can be changed on decisions. Labels can be printed beside the nodes and edges.
* Can have 2D and 3D viewers , the 2D is more aesthetic with different shapes and images while 3D space is used to show very large data sets. (idea can be traversal with different colours for different processors so we have different colours for them)