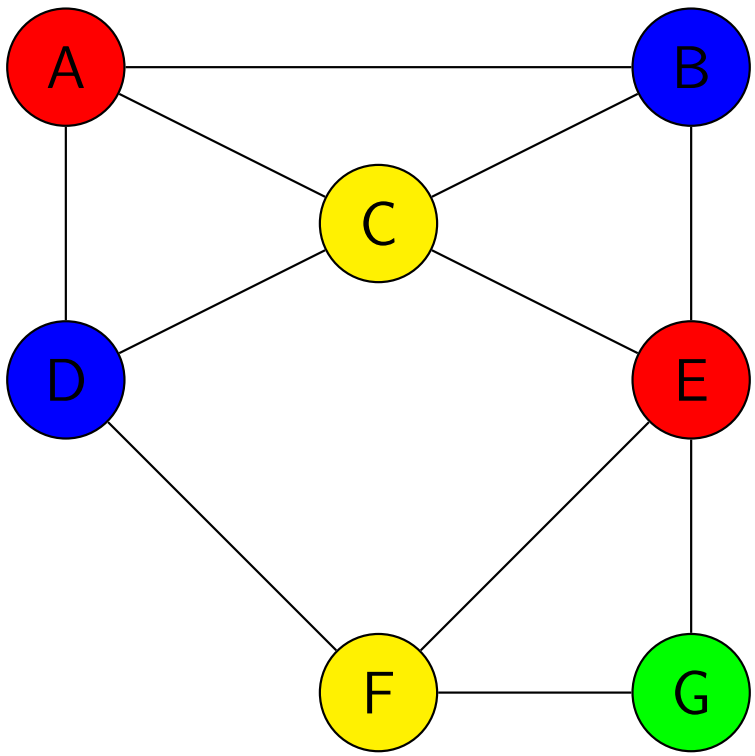
Branch and Bound

Class 40

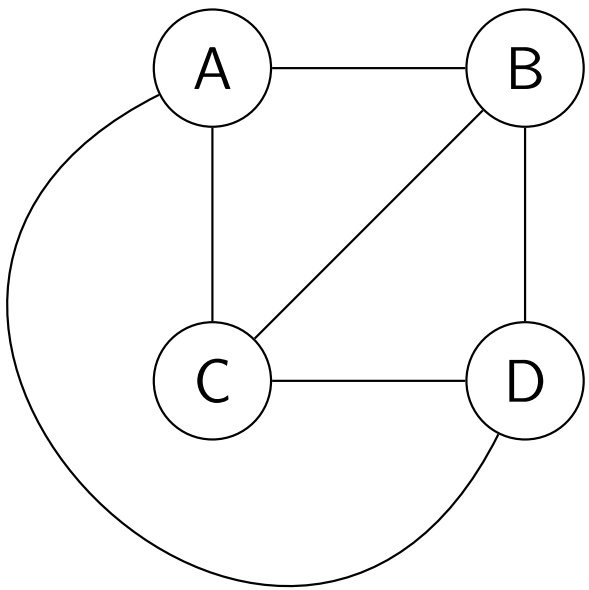
A Famous NPC Problem

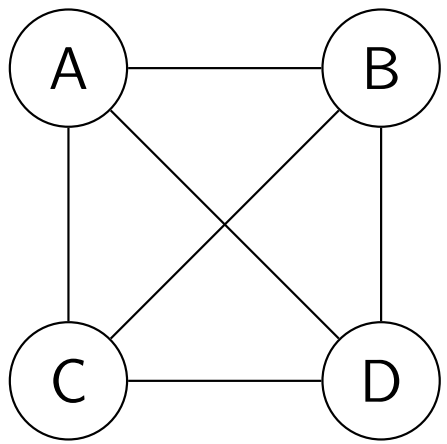
- graph coloring
- assign a color to each vertex of a graph such that no two adjacent vertices have the same color



Graphs

- a planar graph is a graph that can be drawn in a plane without any edges crossing
- this is a planar graph
- note that it can be drawn like a non-planar graph with crossing edges
- it's still planar even if it doesn't look like it





Maps

- maps depict geographic regions
- for ease of viewing, adjacent regions are shaded in contrasting colors

4-Color Conjecture

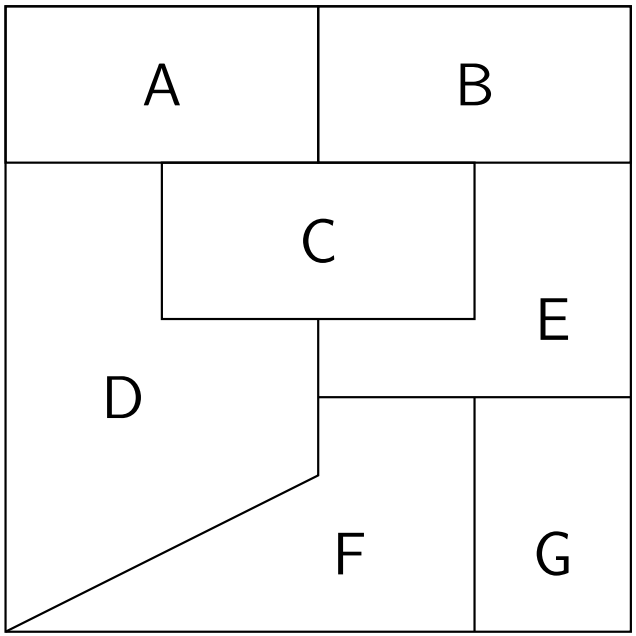
- in the late 1700's and early 1800's, mapmakers and printers noticed that no map on Earth ever required more than 4 colors
- in 1840 Möbius noted the conjecture
- in 1852 Francis Guthrie in England formally proposed the conjecture (Francis' brother Frederick was a student of De Morgan)

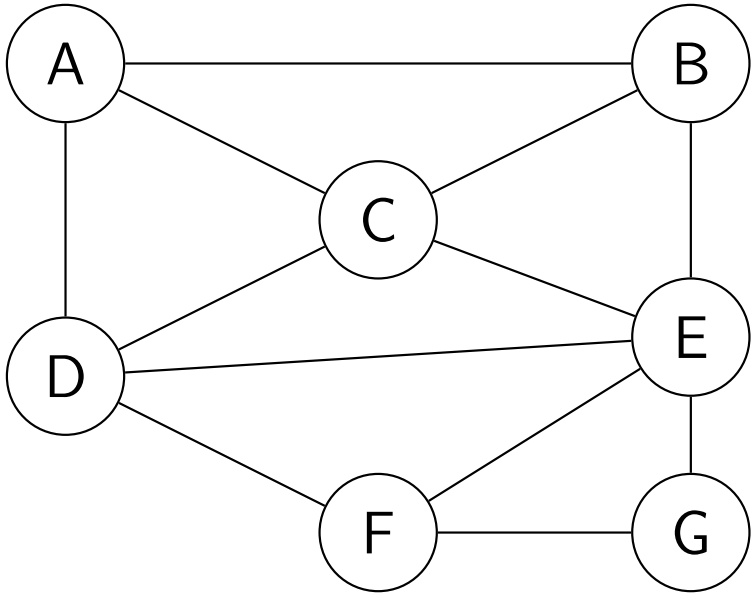
4-Color Conjecture

given any map (a separation of a plane into contiguous regions), no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color

Maps and Graphs

- a map can be represented as a graph
- a region is a vertex
- the adjacency of two regions is an edge between two vertices
- an adjacency must be a face, not a point





- thus map coloring is equivalent to graph coloring

The 4-Color Theorem

- theorem: given a planar graph, no more than four colors are required to color the vertices so that no two adjacent vertices have the same color

- proofs were written in 1879 and 1880
- each was thought valid for more than a decade before being shown to be incorrect
- journals stopped accepting proposed proofs because they were always wrong

The 4-Color Theorem Proved

- in 1976 Appel and Haken at Illinois proposed a proof based on 1,936 cases (configurations) of graphs
- each had to be checked by computation
- required more than 1,000 hours of CPU time

Controversy

- the Appel-Haken proof was very controversial
- they submitted it to multiple mathematics journals
- each rejected it because "programs are not proofs"
- finally published in-house Illinois Journal of Mathematics, with source code available as an appendix
- multiple coding errors found (and subsequently corrected), further tainting the process

- 1996 Robertson et al. created a quadratic-time algorithm that requires only 633 configurations (published in AMS journal with no problem)
- no non-computational proof has ever been found

General Problem

- the original problem was to prove every planar graph can be 4-colored
- the general problem is, given an arbitrary graph, what is the minimum number of colors required to color it
- this is the graph coloring problem, and the decision version of it is in NPC
- the related NP-hard problem is, given the graph, assign colors to each vertex so that the minimum number of colors is used

Applications

- there are many applications for graph coloring

- schedule final exams so that no student has two exams at the same time
- each course is a vertex
- draw an edge if there is at least one student in common between the courses
- color the graph
- each color then represents an exam time period

- assign radio frequencies so no two broadcasting towers are within 150 miles on the same frequency
- vertex: tower
- edge: within 150 miles
- color: assigned frequency

Coping With NPC

- you are a new hire on a software project team
- your manager gives you the assignment for solving a simple-looking problem involving graphs and numbers relating to the company's product
- what do you do?

Coping With NPC

- you are a new hire on a software project team
- your manager gives you the assignment for solving a simple-looking problem involving graphs and numbers relating to the company's product
- what do you do?
- if you are lucky, you will recognize your problem to be one of the few tractable weighted graph problems
  - the shortest path problem (Dijkstra's algorithm)
  - the minimum spanning tree problem (Prim's or Kruskal's algorithms)
  - the maximum network flow problem (a very cool problem)
- even if you're lucky, it takes practice and skill to convert a real-world problem into a "clean" version you can run a textbook algorithm on

Coping With NPC

- but typically you won't be lucky
- the majority of graph search problems are yucky NPC problems
- what do you do?

Coping With NPC

- but typically you won't be lucky
- the majority of graph search problems are yucky NPC problems
- what do you do?
- first you actually need to demonstrate your problem is NPC
- doing so rules out a simple solution
- doing so also makes you look smart
- you're no longer a newbie who can't solve the problem
- you've become a tragic hero on a hopeless quest

Coping With NPC

- but unfortunately, it's still your job
- the boss want a solution
- showing that it's NPC doesn't make the problem go away
- what do you do?

Coping With NPC

- there are four main strategies for dealing with NPC problems
  1. intelligent exhaustive search (find a solution)
  2. branch-and-bound (find an optimal solution)
  3. approximation algorithms (find an approximate solution)
  4. heuristic algorithms (find a locally optimal solution)

Intelligent Exhaustive Search

- we have already discussed this: backtracking
- one more example: SAT
- consider the set of clauses

$$(w \vee x \vee y \vee z) \wedge (w \vee \bar{x}) \wedge (x \vee \bar{y}) \wedge (y \vee \bar{z}) \wedge (z \vee \bar{w}) \wedge (\bar{w} \vee \bar{z})$$

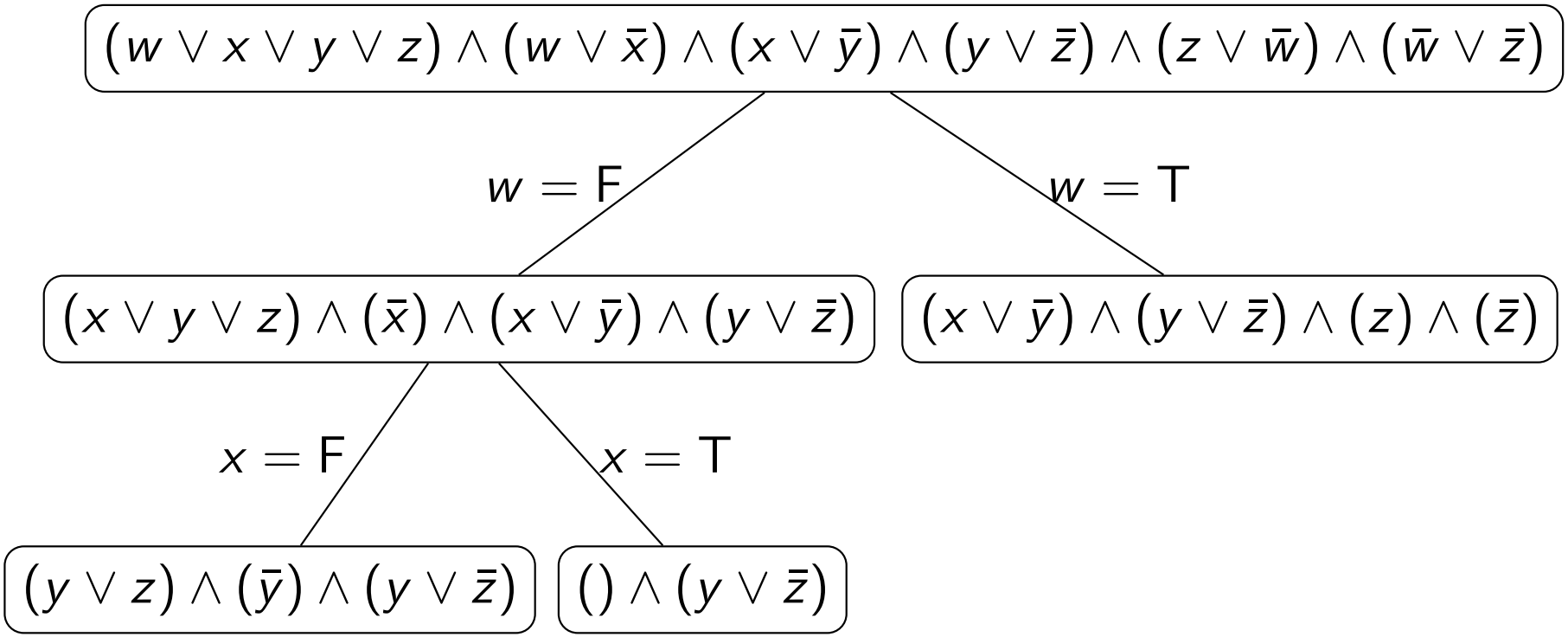
Intelligent Exhaustive Search

- we have already discussed this: backtracking
- one more example: SAT
- consider the set of clauses

$$(w \vee x \vee y \vee z) \wedge (w \vee \bar{x}) \wedge (x \vee \bar{y}) \wedge (y \vee \bar{z}) \wedge (z \vee \bar{w}) \wedge (\bar{w} \vee \bar{z})$$

- to solve this by backtracking
  - a level consists of an assignment of one variable
  - the order of the branches are the assignments of the variable, false then true

SAT via Backtracking



- an empty clause denotes failure (no solution)
- a pair of directly contradictory clauses denotes failure
- success is when no clauses are left

Backtracking

- uses a search space tree
- searches for a solution in a fixed depth-first search pattern
- seeks to prune unpromising branches
  - a contradiction, or conflict
  - a state in which intelligence can guarantee no solution is possible
- if a level-n node is reached, halt with success

Backtracking for Optimization

- backtracking is particularly unsuitable for an optimization problem — why?

Backtracking for Optimization

- backtracking is particularly unsuitable for an optimization problem — why?

- you must find all solutions, then choose the optimal one
- this is full brute force — as bad as it gets

Branch and Bound

- for optimization problems, with small changes, we can do much better
- for branch and bound we need additional things
  - a cost function that gives a bound on the best possible value, given current assignments (must be very efficient)
  - a modification of the strict DFS pattern

Branch and Bound

- branch and bound strategy backtracks (prunes) in three cases
  - a contradiction is reached (just like backtracking)
  - a node is reached whose value is worse than the value of the best solution seen so far
  - a node is reached whose bound function indicates that no solution on this path can be better than the value of the best solution seen so far
- and proceeds forward not by blind DFS but by choosing the most promising node to elaborate next

Job Assignment Problem

Person	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

- we have n workers and n jobs to be performed
- every worker can do every job, but with different efficiencies
- jobs and workers are atomic, so we must assign exactly one worker to exactly one job

Assignment Cost Matrix

Person	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

- the problem is to select n pairs (p, j) such that the ps are unique and the js are also unique
- and to select the n pairs so that their corresponding entries in the cost matrix sum to the minimum
- this is a perfect problem for branch and bound

Lower Bound

- branch and bound depends on having both an exact cost function and a running notion of a bound (lower bound for minimization, upper bound for maximization)
- what is a lower bound on any possible solution for the problem?
- it is impossible for any solution to have a lower cost than the sum of the smallest element in each row
- in this case, lower bound $= 2 + 3 + 1 + 4 = 10$

Person	**Job 1**	**Job 2**	**Job 3**	**Job 4**
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

Lower Bound

- further, if we happened to choose $(a, 1)$ as the first assignment, then a lower bound for any solution in this path is $9 + 3 + 1 + 4 = 17$

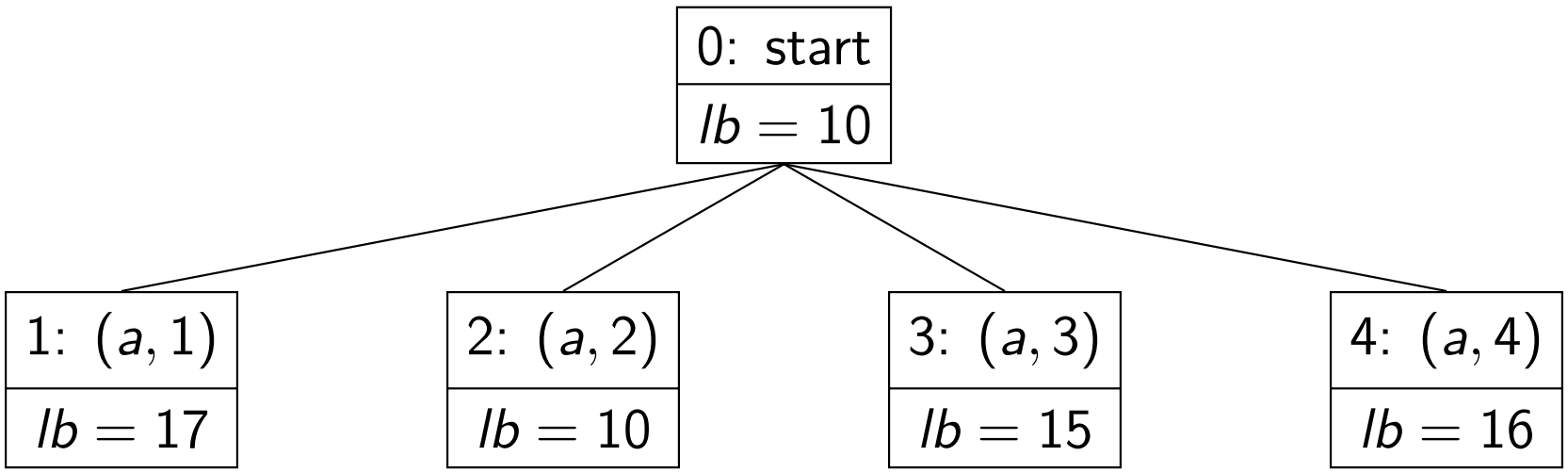
Person	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

Order of Elaboration

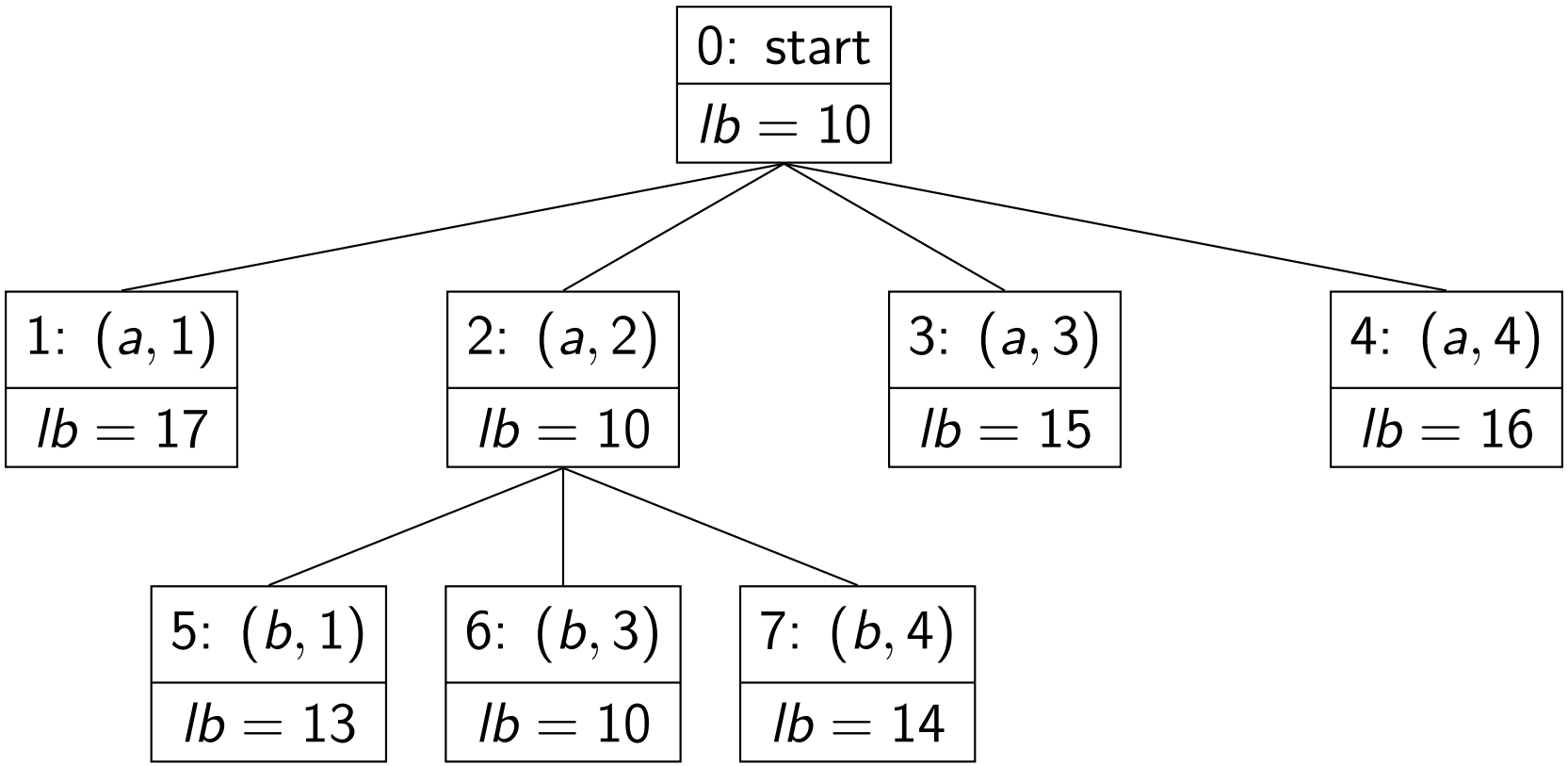
- pure backtracking goes strictly in DFS order
  - down a level to a new node
  - left to right for the children of that node
- branch and bound is smarter about which path to choose next:

  1. elaborate all children of the most promising node
  2. choose the best of all current leaf nodes to elaborate next

- to start, no assignments have been made
- person a can be given either of four different jobs
- we elaborate all four and calculate the lower bound of each



- the most promising node is 2; elaborate its three children



0-1 Knapsack

- you're a thief
- there are n items with weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$ (all positive integers)
- items are indivisible
- you have a knapsack with weight capacity W; more than that and it will break

Let $W = 10$

item	wt	value	value/wt
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

Setup

- we order the items in descending value to wt ratio
- we need a way to calculate the upper bound of each node in the search tree

- a simple ub function is the value of everything already in the knapsack plus the knapsack's remaining weight capacity filled (possibly fractionally) with the unused item of best payoff