中山大學
SUN YAT-SEN UNIVERSITY

Lecture 6
Backtracking & Branch-and-Bound

Algorithm

张子臻，中山大学计算机学院

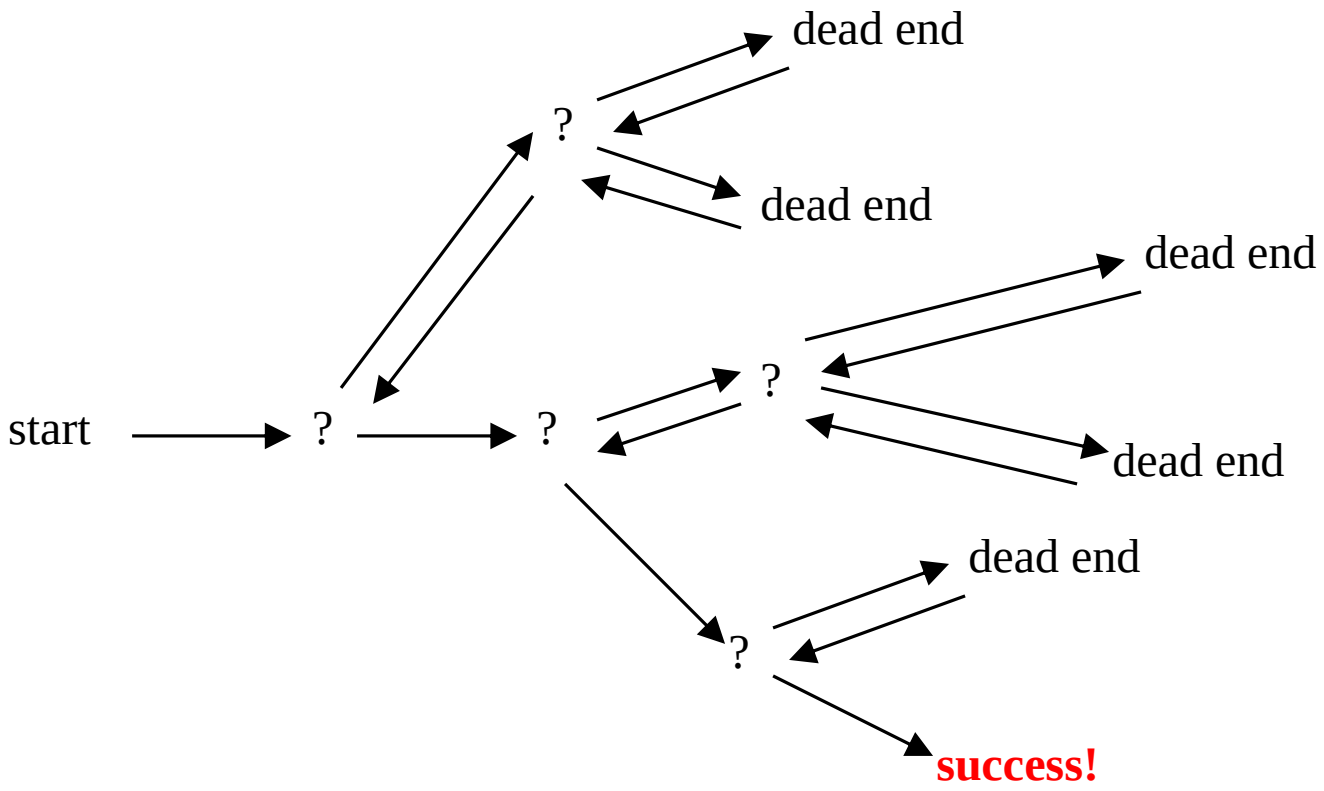
zhangzizhen@gmail.com

Backtracking

- Suppose you have to make a series of decisions, among various choices, where
  - You don’t have enough information to know what to choose
  - Each decision leads to a new set of choices
  - Some sequence of choices (possibly more than one) may be a solution to your problem
- Backtracking is a methodical way of trying out various sequences of decisions, until you find one that “works”

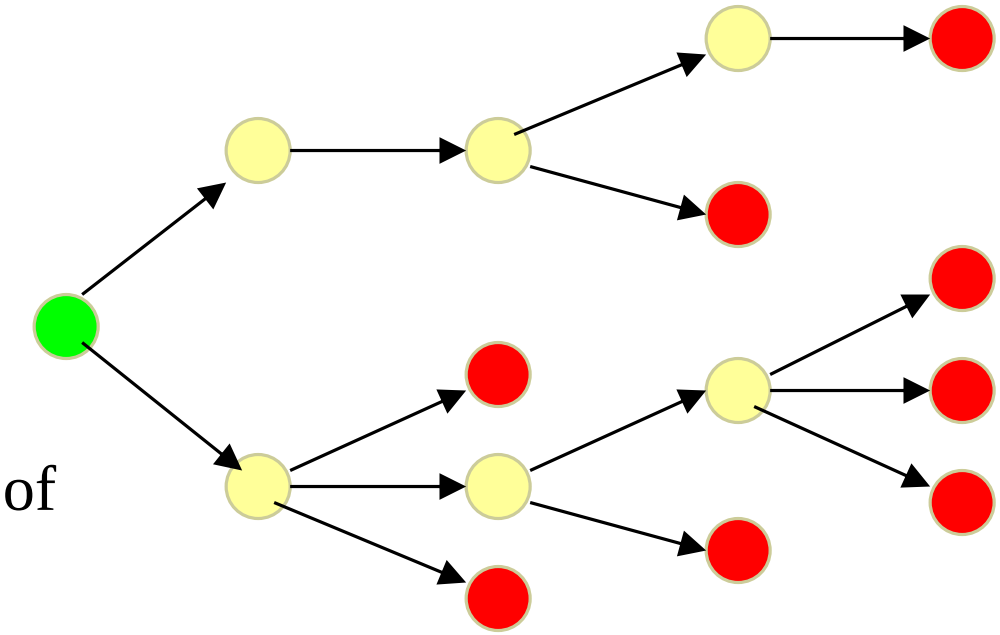
Backtracking

- Example: Decision making process.



Search Tree

- A tree is composed of nodes



There are three kinds of nodes:

- The (one) root node
- Internal nodes
- Leaf nodes

Backtracking can be thought of as searching a tree for a particular "goal" leaf node

The Backtracking Algorithm

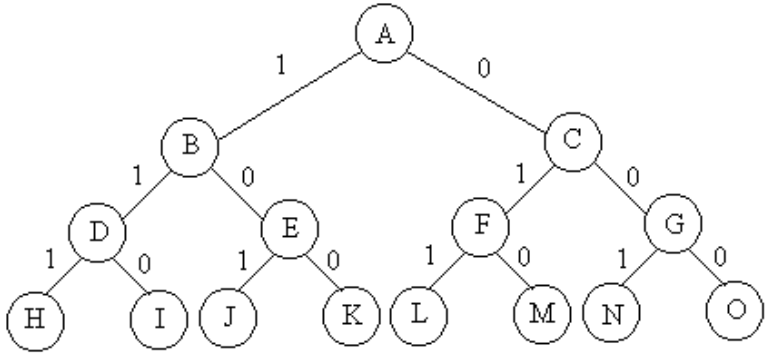
- Backtracking is really quite simple -- we **recursively** “explore” each node, as follows:
- To “explore” node N:

1. If N is a goal node, return “**success**”
2. If N is a leaf node, return “**failure**”
3. For each child C of N,

   Explore C

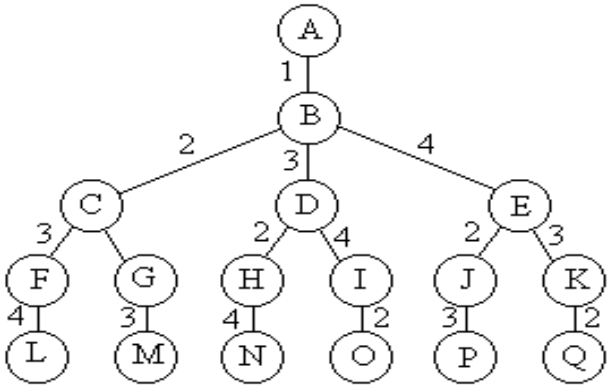
   If C was successful, return “**success**”
4. Return “**failure**”

Subset Tree and Permutation Tree



Enumerating all subsets take $O(2^n)$

```
void backtrack(int t)
{
  if (t>n) output(x);
  else
      for (int i=0;i<=1;i++) {
        x[t]=i;
        if (legal(t)) backtrack(t+1);
      }
}
```

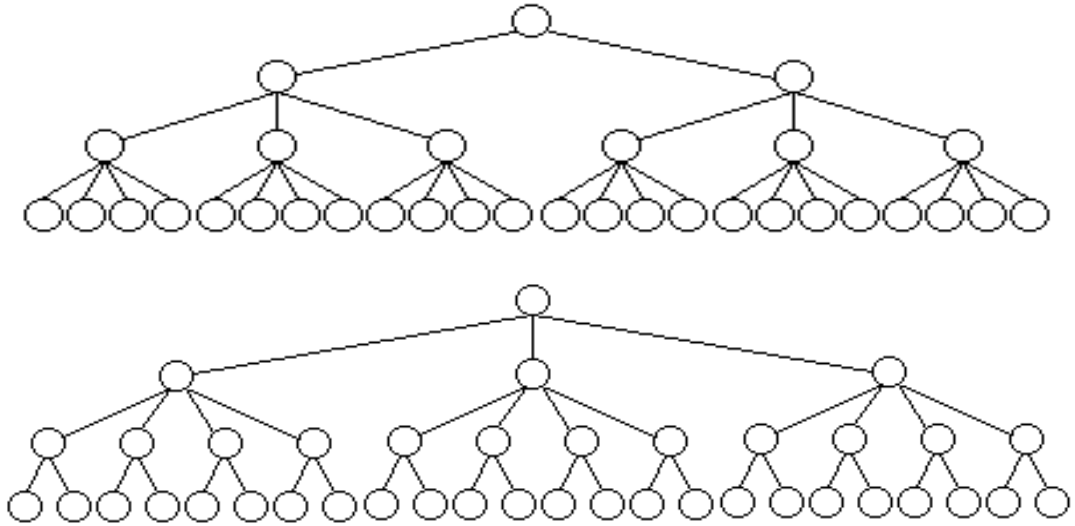


Enumerating all permutations take O(n!)

```
void backtrack(int t)
{
  if (t>n) output(x);
    else
      for (int i=t;i<=n;i++) {
        swap(x[t], x[i]);
        if (legal(t)) backtrack(t+1);
        swap(x[t], x[i]);
      }
}
```

重排原理

- 对于许多问题而言，在搜索试探时选取x[i]的值顺序是任意的。**在其它条件相当的前提下，让可取值最少的x[i]优先**。从图中关于同一问题的2棵不同解空间树，可以体会到这种策略的潜力。



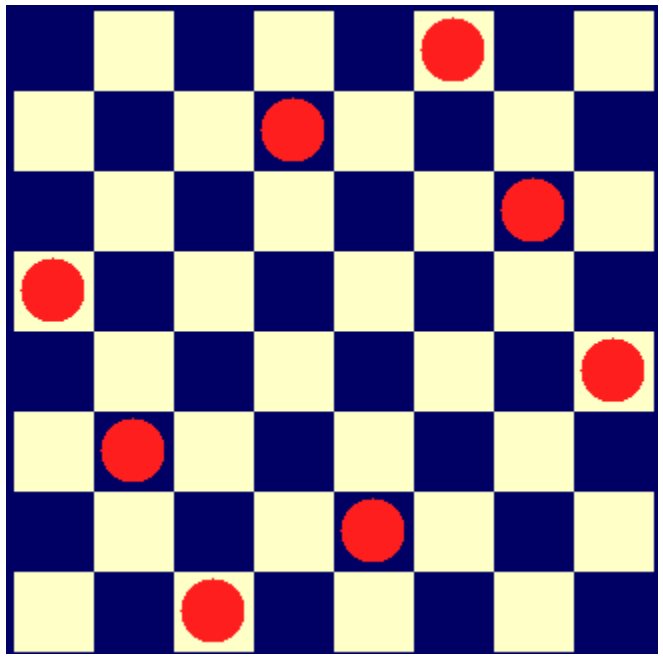
(a)

(b)

图(a)中，从第1层剪去1棵子树，则从所有应当考虑的3元组中一次消去12个3元组。对于图(b)，虽然同样从第1层剪去1棵子树，却只从应当考虑的3元组中消去8个3元组。前者的效果明显比后者好。

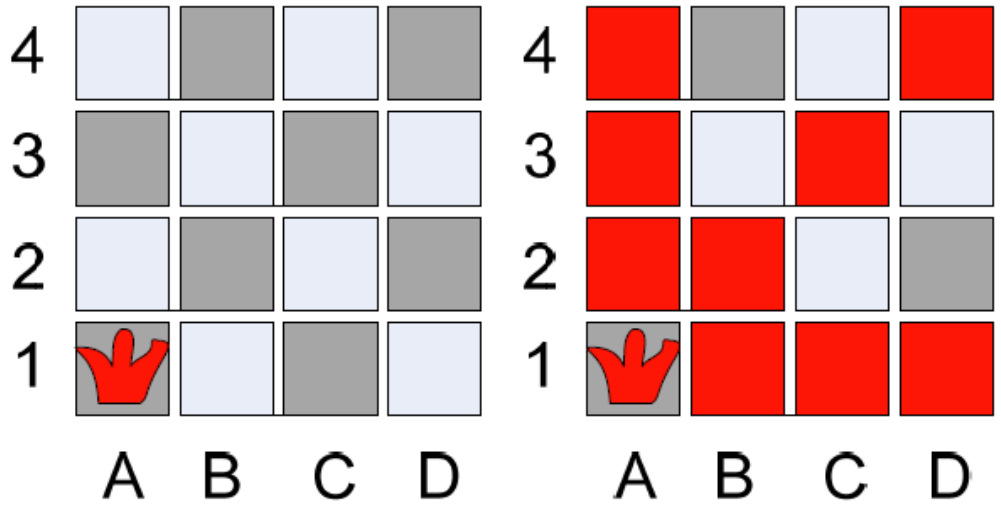
N Queen Problem

- In chess, a queen can move as far as she pleases, horizontally, vertically, or diagonally. A chess board has 8 rows and 8 columns. The standard 8 by 8 Queen's problem asks how to place 8 queens on an ordinary chess board so that none of them can hit any other in one move.

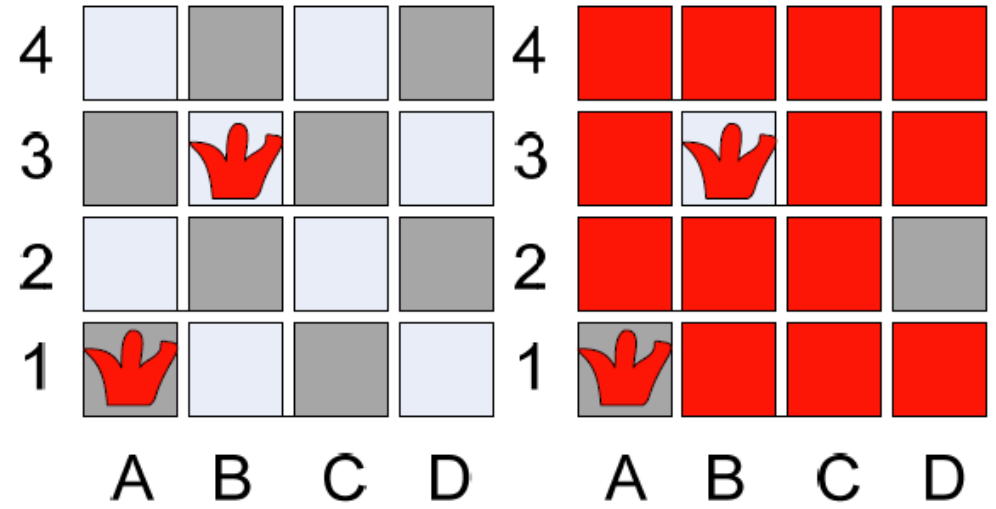


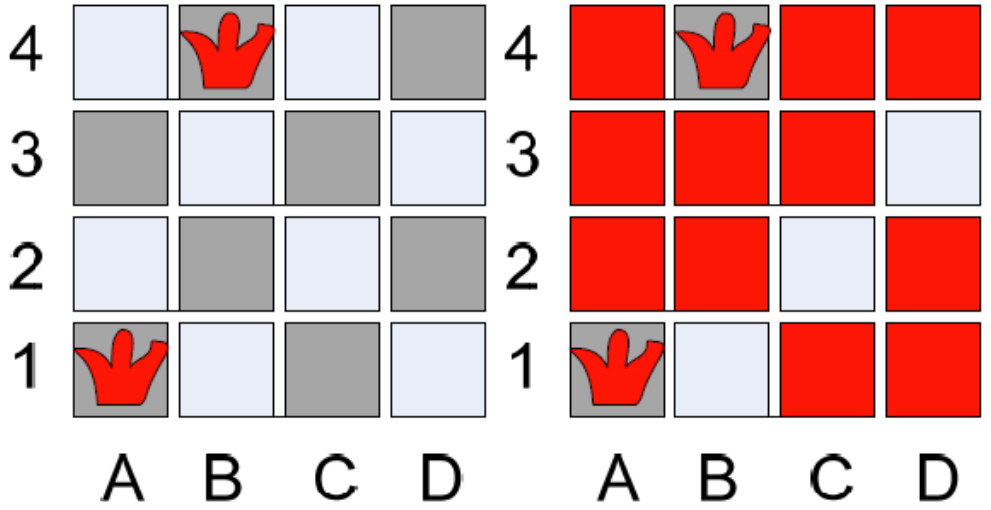
N Queen Problem

- Algorithm:
  - Start with one queen at the first column first row
  - Continue with second queen from the second column first row
  - Go up until find a permissible situation
  - Continue with next queen

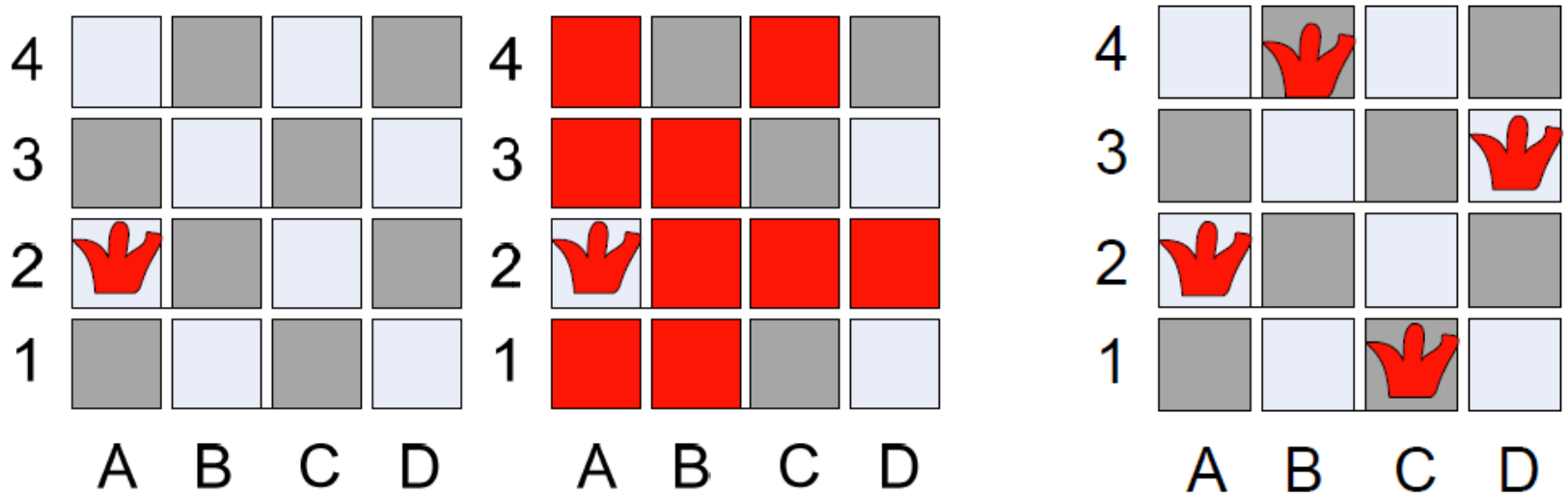


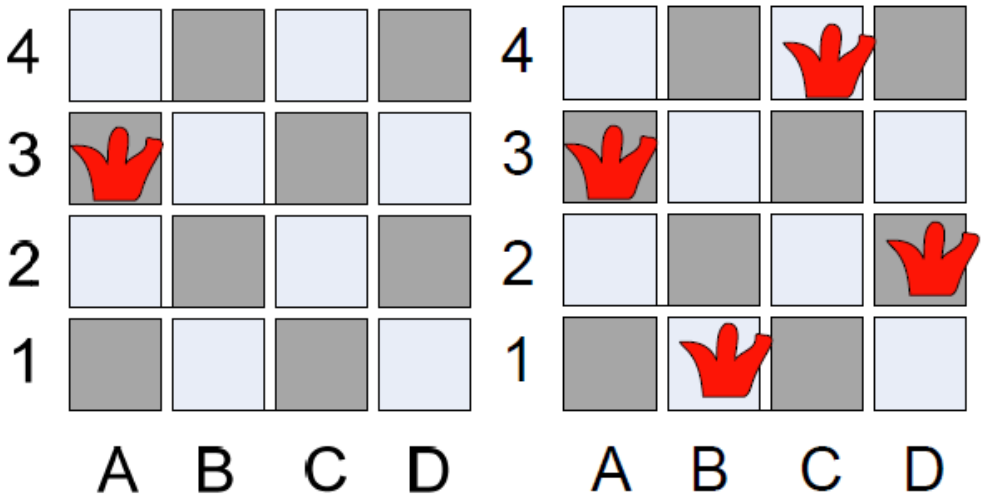
N Queen Problem





N Queen Problem





N Queen Problem

- Different column: $x_i \neq x_j$
- Different diagonal: $|i-j| \neq |x_i - x_j|$

```
bool Queen::Place(int k)
{
  for (int j=1;j<k;j++)
    if ((abs(k-j)==abs(x[j]-x[k]))||(x[j]==x[k])) return false;
  return true;
}
void Queen::Backtrack(int t)
{
  if (t>n) sum++;
    else
      for (int i=1;i<=n;i++) {
        x[t]=i;
        if (Place(t)) Backtrack(t+1);
      }
 }
```

Branch-and-Bound

- The branch-and-bound design strategy is very similar to **backtracking** in that a state space tree is used to solve a problem.
- The differences are that the branch-and-bound method 1) does not limit us to any particular way of traversing the tree, and 2) is used for optimization problems.
- A branch-and-bound algorithm computes a number (**bound**) at a node to determine whether the node is promising.

Branch-and-Bound

- The number is a bound on the value of the solution that could be obtained by expanding beyond the node.
- If that bound is no better than the value of the best solution found so far, the node is **nonpromising**. Otherwise, it is **promising**.
- Besides using the bound to determine whether a node is promising, we can compare the bounds of promising nodes and visit the children of the one with the best bound.
- This approach is called **best-first search with branch-and-bound pruning.**

Branch-and-Bound

- An enhancement of backtracking
- Applicable to optimization problems
- Uses a bound for the value of the objective function for each node (partial solution) so as to:
  - guide the search through state-space
  - rule out certain branches as “unpromising”
- Many Branch-and-bound methods are adopted using the Depth-First Search (DFS) manner, since it generally consumes little memories.

Breadth-first Search

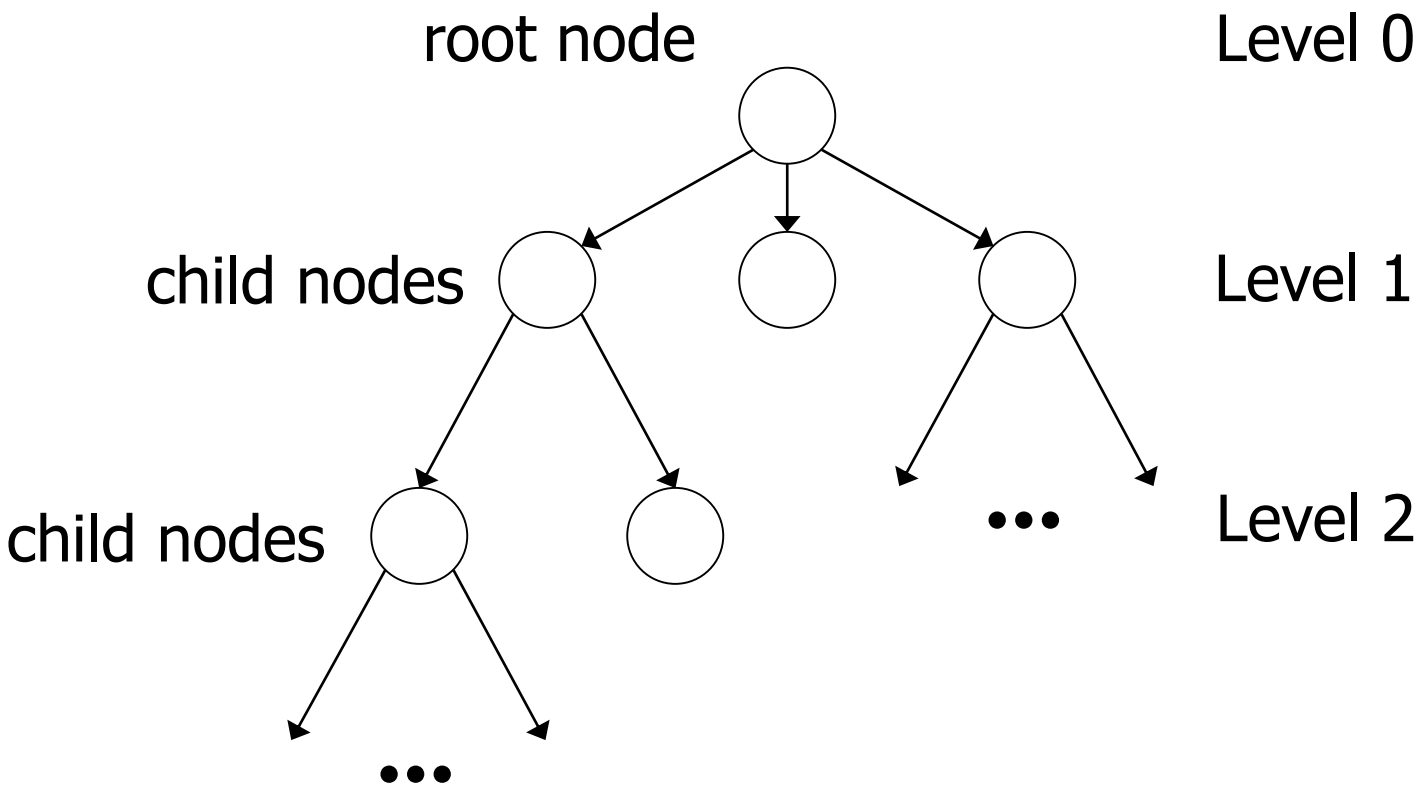
- The Branch-and-bound can also be applied within the Breadth-First Search (BFS) framework.
- We can implement the BFS using a queue.
- All child nodes are placed in the queue for later processing if they are promising.
- Consider the **minimization** problems.
- Calculate the value for each node that represents the minimum possible value if we pick that node.
- If the minimum possible value is greater than the global best value, don’t expand the branch.

Best-first Search

- The breadth-first search strategy has no advantage over a depth-first search (backtracking).
- However, we can improve our search by using our bound to do more than just determine whether a node is promising.
- Best-first search expands the node with the best bounds next. (A* ?)
- How would you implement a best-first search?
  - Depth-first is a stack
  - Breadth-first is a queue
  - Best-first is a ???

Enumeration in a Search Tree

- Each node is a partial solution, i.e., a part of the solution space



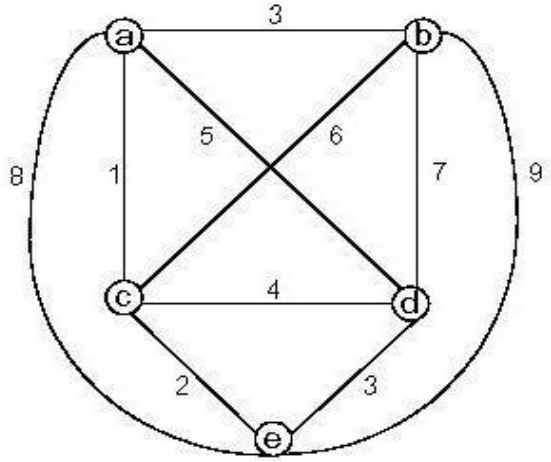
Bounding

- A bound on a node is a guarantee that any solution obtained from expanding the node will be:
  - Greater than some number (lower bound)
  - Or less than some number (upper bound)
- If we are looking for a maximal optimal (knapsack problem), then we need an upper bound.
- If we are looking for a minimal optimal (traveling salesman problem), then we need a lower bound.

Bounding

- We prune (via bounding) when:
  - (nodeBound >= currentBestSolutionCost) for minimization problem
  - (nodeBound <= currentBestSolutionCost) for maximization problem
- We want to find a globally good solution quickly.
- We want to find a good bound for each node.
- Initially, a global solution can be obtained by a greedy or an efficient heuristic.
- A bound can be obtained by relaxing some constraints of the subproblem related to the node.

Traveling Salesman Problem



- Generate permutations

TSP: 1-Tree Bound

- Can apply branch & bound if we come up with a reasonable lower bound on tour lengths.
- Simple lower bound = finding smallest element in the intercity distance matrix D and multiplying it by number of cities *n*
- Another bound:

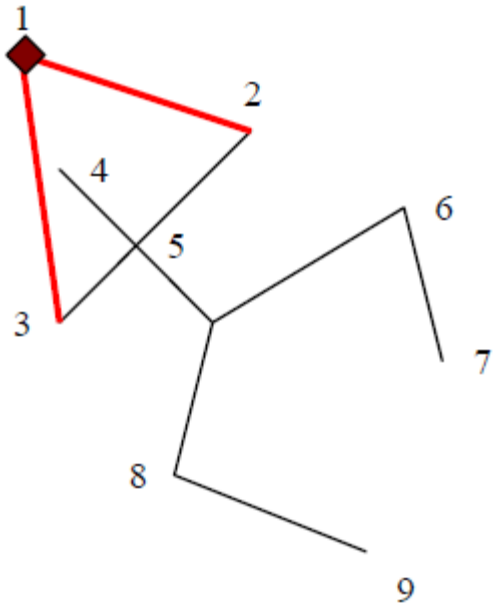
$$\frac{1}{2}\sum_{v\in V}\begin{pmatrix}\text{Sum of the costs of the two tour}\\ \text{edges adjacent to } v\end{pmatrix}$$

- 1-tree bound: a tour consists of a special *spanning tree* (namely a path) on the remaining *N*-1 nodes plus two edges connecting node 1 to this spanning tree.

1-Tree

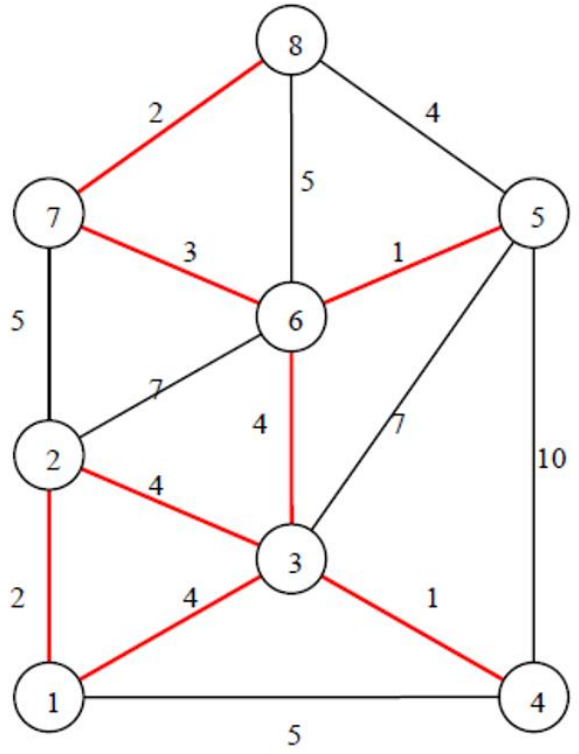
- Definition: for a given vertex, say vertex 1, a 1-Tree is a tree of {2,3,…,n} plus 2 distinct edges connected to vertex 1.

Example of 1-Tree



Minimum Weight 1-Tree

- Definition: Min cost 1-tree of all possible 1-Trees.
- To find minimum weight 1-Tree, first find minimum spanning tree of {2,3,…,n} vertices, and add two lowest cost edges incident to vertex 1.
- Any TSP tour is 1-Tree tour (with arbitrary starting node 1) in which each vertex has a degree of 2.
- If minimum weight 1-Tree is a tour, it is the optimal TSP tour.
- Thus, the minimum 1-Tree provides a lower bound on the length of the optimal TSP tour.



Summary: Branch-and-bound

- If an optimal solution is found to a subproblem, it is a feasible solution to the full problem, but not necessarily globally optimal.
- Since it is feasible, it can be used to prune the rest of the tree.
- The search proceeds until all nodes have been solved or pruned, or until some specified threshold is meet between the best solution found and the lower bounds on all unsolved subproblems.

Thank you!