Unit 7.
An overview of algorithm strategies

Data Structures and Algorithms (DSA)

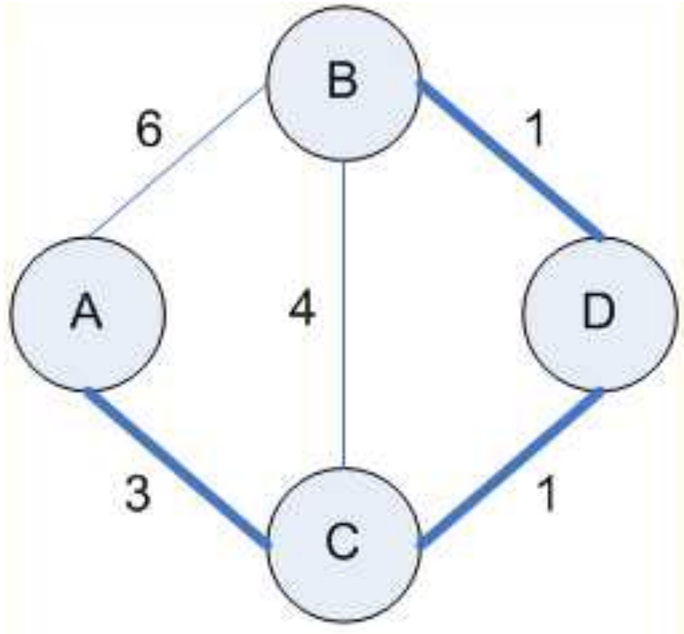
Material elaborado por los profesores de la asignatura

Some concepts

- An **algorithm** is a well-defined and finite sequence of steps used to solve a well-defined problem.
- **Algorithm strategy**
  - Approach to solving a problem
  - May combine several approaches
- **Algorithm structure:**
  - **Iterative**: uses a loop to find the solution
  - **Recursive**: a function calling itself

Problem type

- **Satisfying**
  - Find any satisfactory solution
  - Ex: Find a path from A to E
- **Optimization**
  - Find the best solution
  - Ex: Find the shortest path from A to E



This example was taken from http://cs.smu.ca/~porter/csc/common_341_342/notes/graphs_shortest_path.html

Main Algorithm Strategies

- Recursive algorithms
- Divide and Conquer algorithms
- **Backtracking algorithms**
- Dynamic programming algorithms
- Greedy algorithms
- Brute force algorithms
- Branch and bound algorithms
- Heuristic algorithms

Heuristics and Optimization, Course 3ª, Semester 1º

These slides are based on the course CMSC 132 's materials, University of Maryland

Get out of the labyrinth

- We are in a labyrinth, full of crossings, detours and somebody chasing us.
- There are brute force solutions (e.g. always follow the wall at our left) but they are inefficient.
- We have to create a path of decisions (right-left-straight-right-straight…) to reach the exit but skipping paths that, without following them, it is known that they are not driving us to the exit.
- If we take a bad decision, we can go back and try a different path.

Constraints

- Sometimes, there are no specific algorithms to solve a problem → All possible **solutions** (not decisions) must be explored.
- The solution (aprtial or global) is a vector of cases, decisions, values, etc., with a finite length.
- There is a way to know if a solution is global or partial and, hence, the algorithm must go to an end (attention to infinite loops must be paid).
- Completeness: Given a partial solution, it is possible to say if it is part of a global solution (it is "complete") or not.

Back Tracking

- General philosophy to solve problems also used in many other scopes.
  - Technique to solve problems based on exploring possible partial solutions.
  - Each partial solution is extended with more 'complete' solutions.
  - Partial solutions are evaluated using brute force approaches.
  - When a solution is 'complete', it is called 'k-promising solution' where k is the level in that execution point in the algorithm.
  - When a set of solutions is not 'complete', it is discarded..
  - Implementing these algorithms requires **recursion**.

Five core methods

1. **exploreLevel(k):** The algorithm goes one step deeper in the set of possible solutions.
2. **pendingOptions():** Checking if there are pending options to be explored at the current level.
3. **completeSolution():** Checking if the current solution is complete or global.
4. **processSolution():** to show the solution.
5. **completeness():** To evaluate if a k-promising vector can be completed.

3 stages method

Start the algorithm at k level

The algorithm analyses the next level k, starting at the first level.

Check options at k level

If there are n solutions that can be completed, the algorith goes tothe k+1 level for each option

Otherwise the path is discarded

Finalize

If there are no more options for a path, is it a global solution (return true) or not (return false)?

When to apply backtracking

- It is a brute force algorithm (discarding some options), so:
  - Used when there are no more appropriate solutions
  - The solutions tree is finite and there are no loops (or loops can be avoided)
- In general:
  - The problem can be solved taking decisions at each level, so
    - The algorithm allows for a recursive design
    - It is possible to discard candidates
  - IMPORTANT: take care of memory (each recursive call requires room for all local variables)

Designing the algorithm

- Recursive algorithm
  - At each level, the candidate solution is checked to know if it is global
  - If it is not, complete candidate solutions are evaluated and the algorithm runs over all of them
- Global solution decision
  - Evaluate if a vector is a solution to the problem
- Completeness
  - Evaluates if a solution "seems to be valid" or must be discarded. The completeness criterion must be determinist ("true" or "false"). E.g. If the solution must not have repeated numbers the candidate vector [3,5,1,1] can be discarded
- Are we looking for a solution? Or do we want all solutions?
  - We can finish the algorithm with the first solution found or explore the k-promising remaining candidate solutions

Back tracking algorithm

Pseudocode

```
VA(x: sequence, k: level) {
  exploreLevel(k)
  While pendingOptions(k){
     extend x with option vi
     if ({x,vi} is a global solution) {
            processSolution()
     }
     else {
            if(completeness(({x,vi})
                   VA({x,vi},k+1)
     }
  }
}
```

1. Search for promising candidates

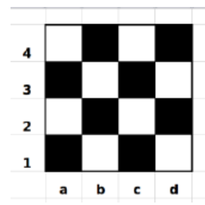
2. Detect global solution

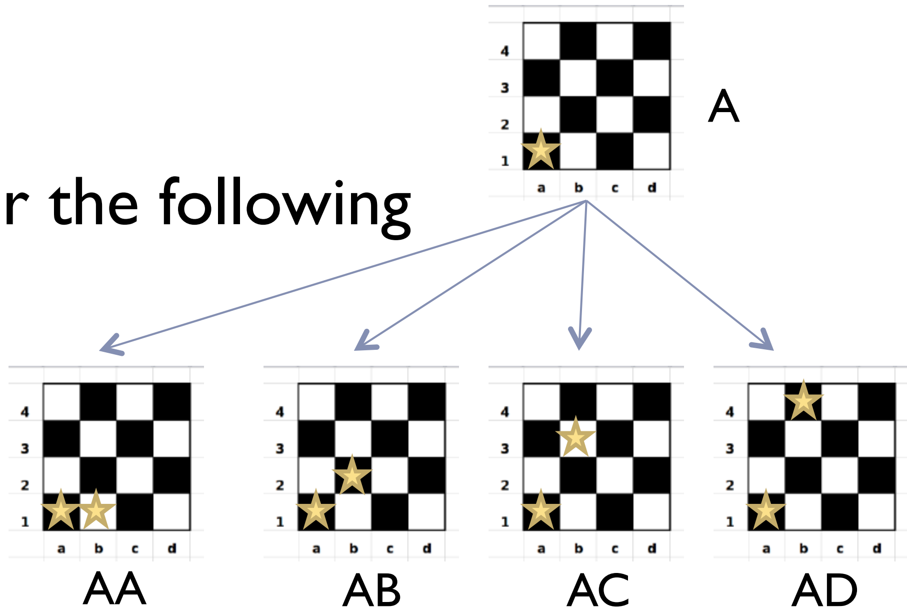
3. Detect if it is a k-promising candidate solution

4. Launch the algorithm with the next level

Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

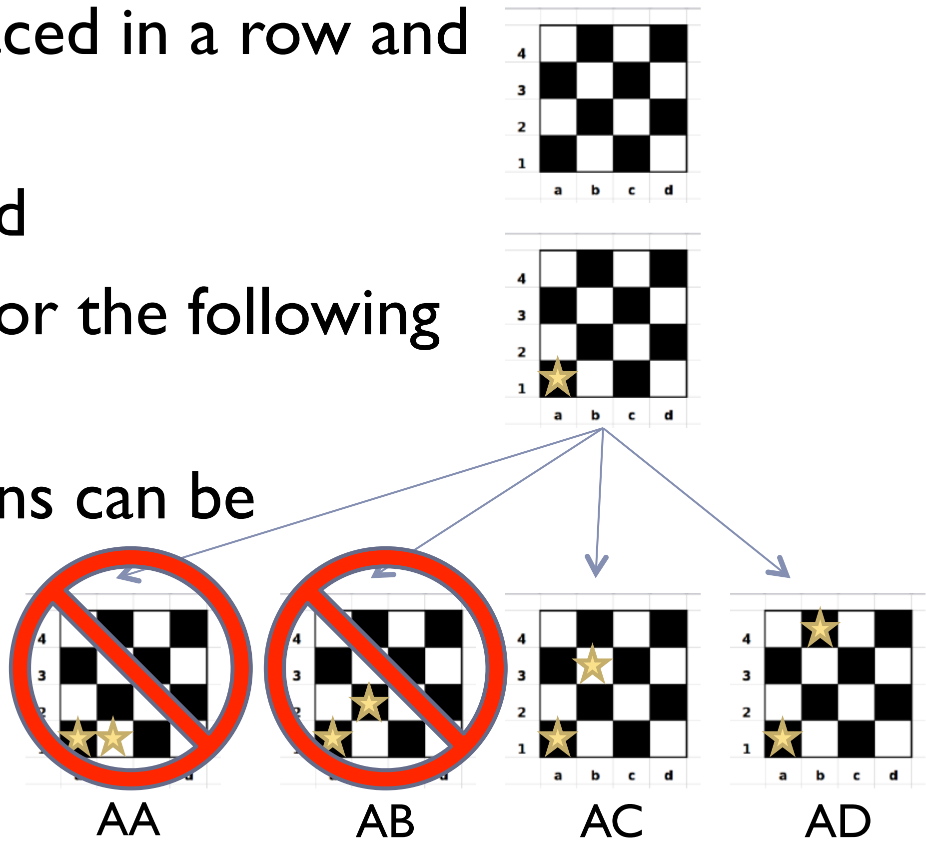
- An empty 4x4 chessboard is available
- Each queen must be placed in a row and a column
- The first queen is placed
- Four different options for the following queen





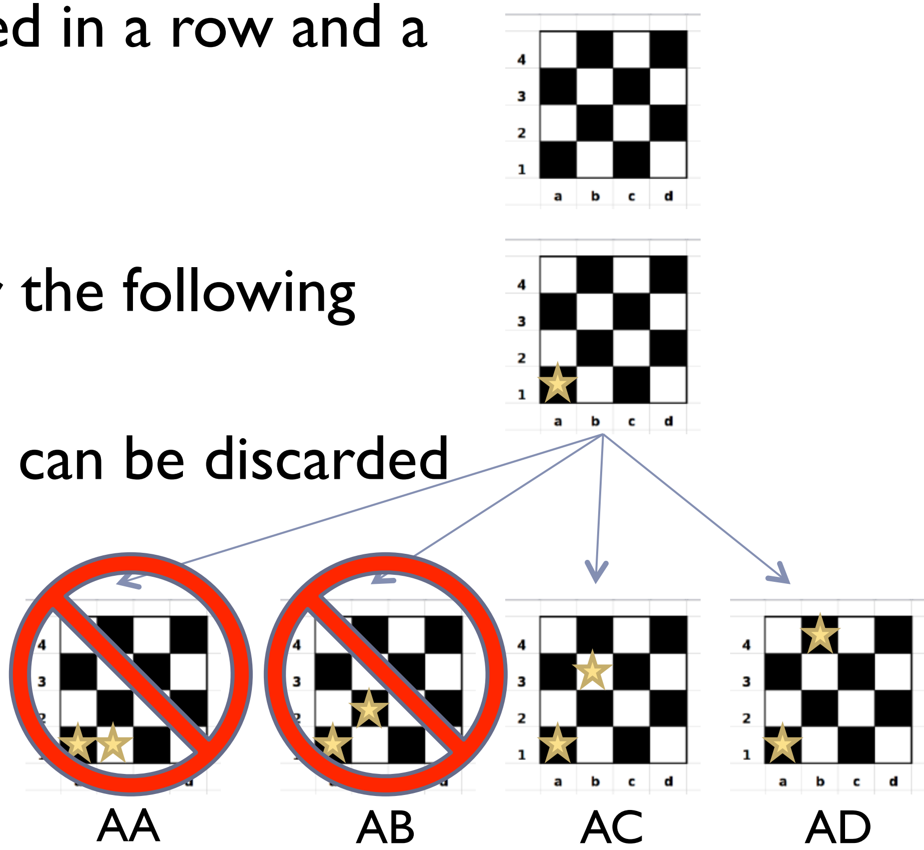
Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

- An empty 4x4 chessboard is available
- Each queen must be placed in a row and a column
- The first queen is placed
- Four different options for the following queen
- Some candidate solutions can be discarded



Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

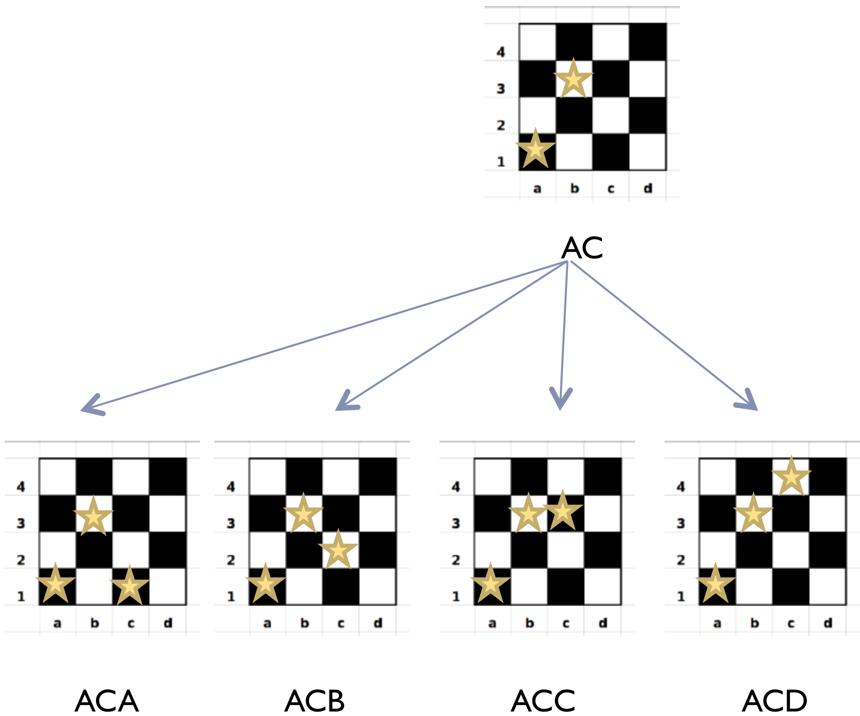
- An empty 4x4 chessboard is available
- Each queen must be placed in a row and a column
- The first queen is placed
- Four different options for the following queen
- Some candidate solutions can be discarded



- Solutions under AA and AB are not further considered, they are not valid (not complete)

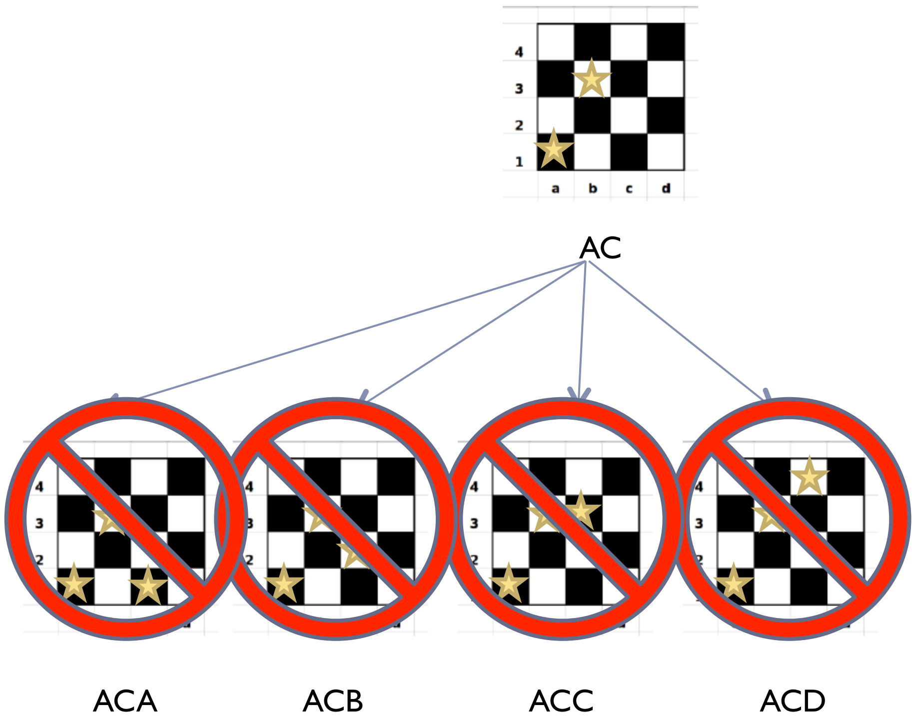
Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

- Let's develop AC candidate:



Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

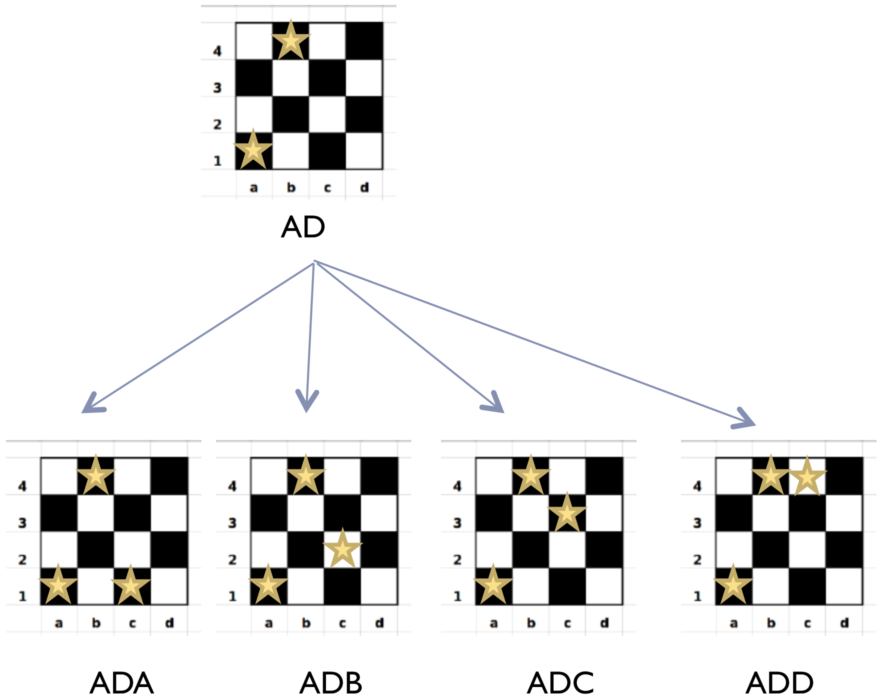
- Let's develop AC candidate:



- No valid candidates!! All options discarded … so?

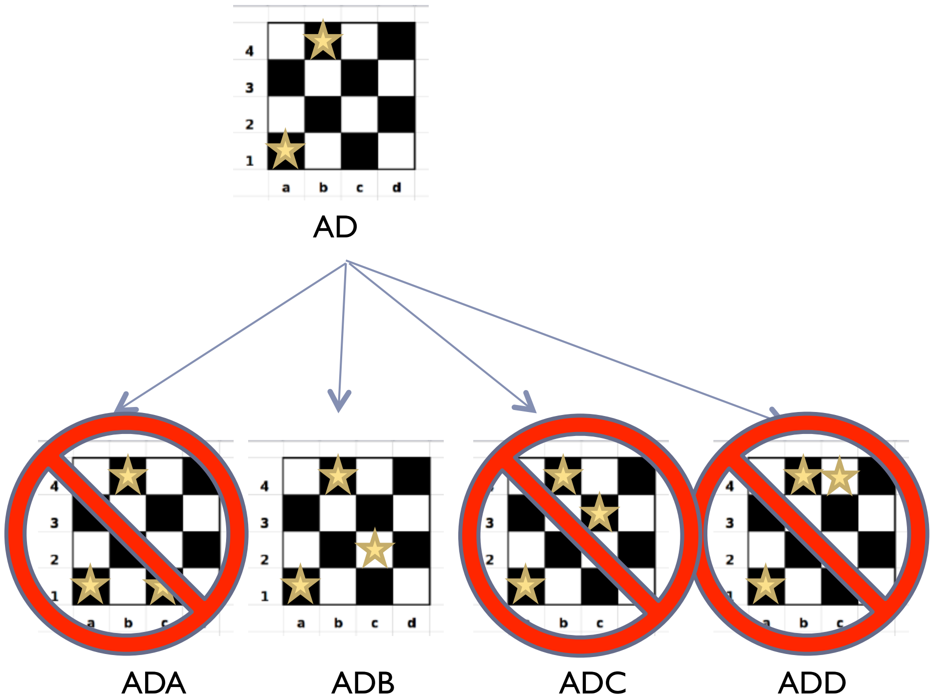
Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

- Back tracking!!, go back to AD candidate:



Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

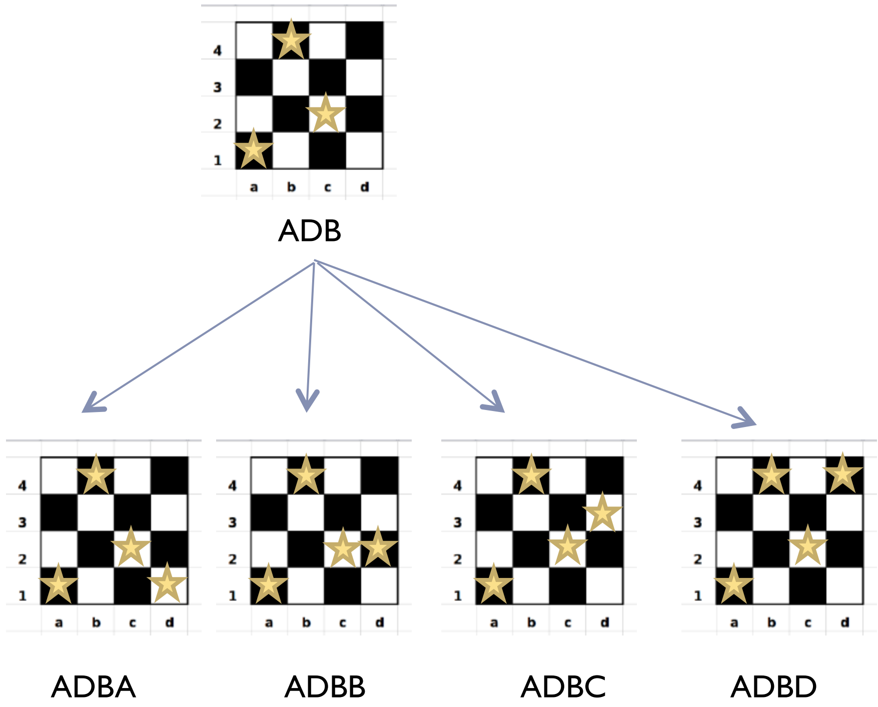
- Back tracking!!, go back to AD candidate:



- But some candidates can be discarded

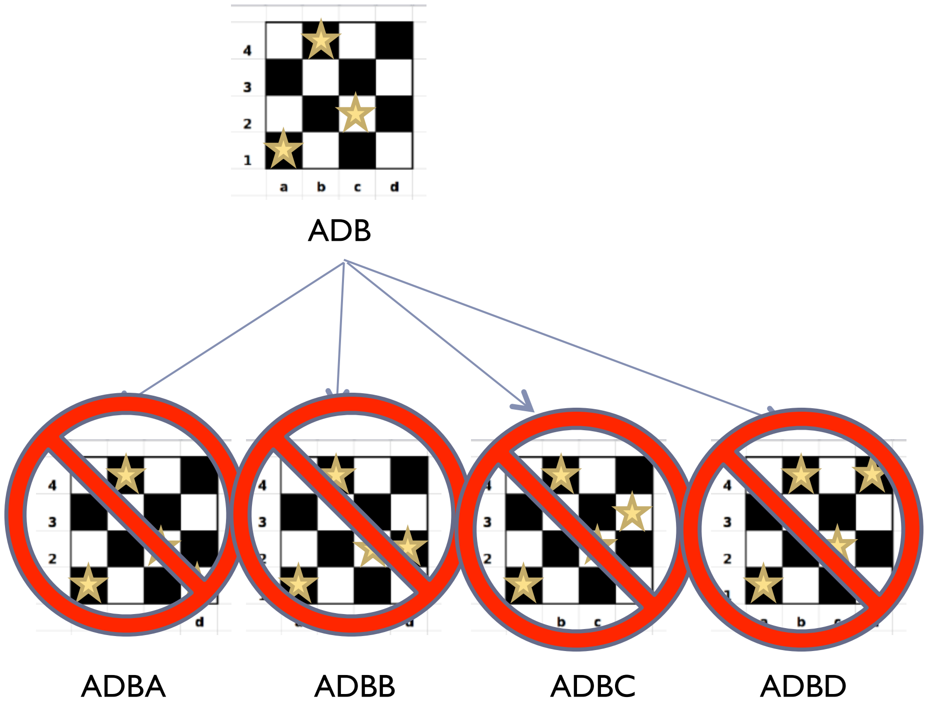
Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

- Let’s develop ADB:



Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

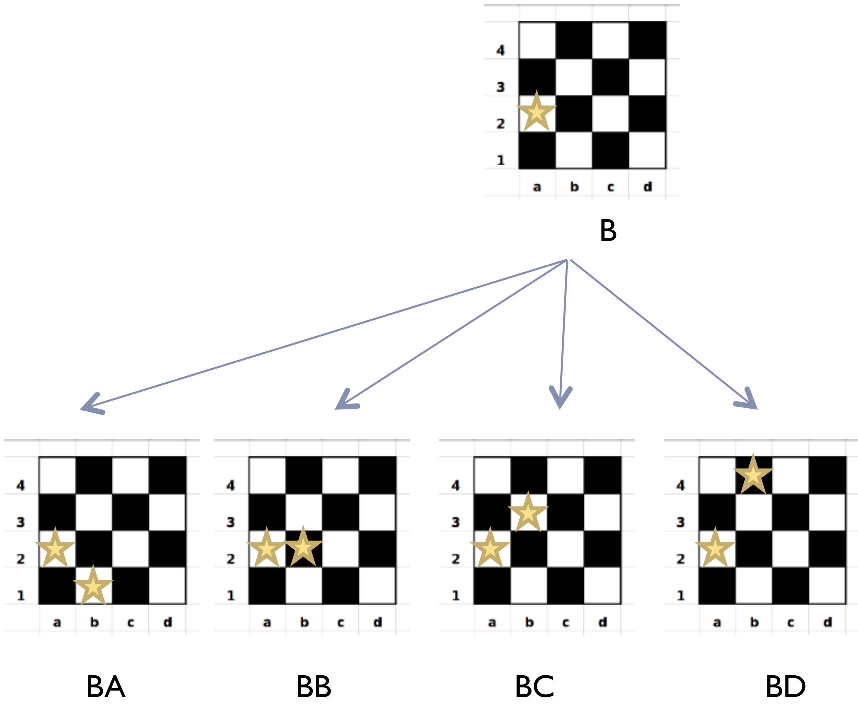
- Let's develop ADB:



- All candidates discarded!! Is there a solution?

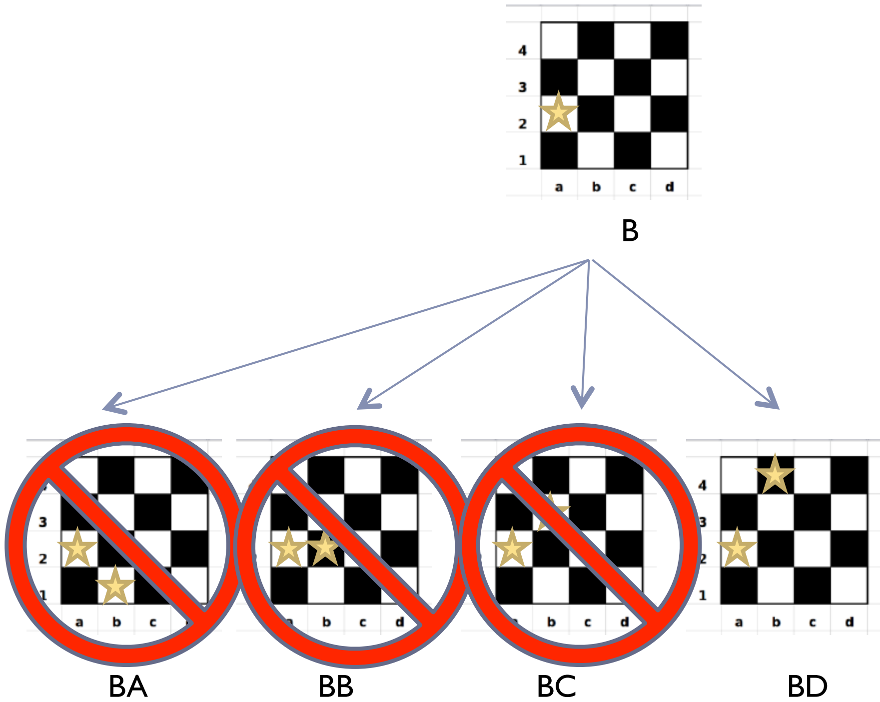
Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

- Back tracking!! Try another position for first queen:



Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

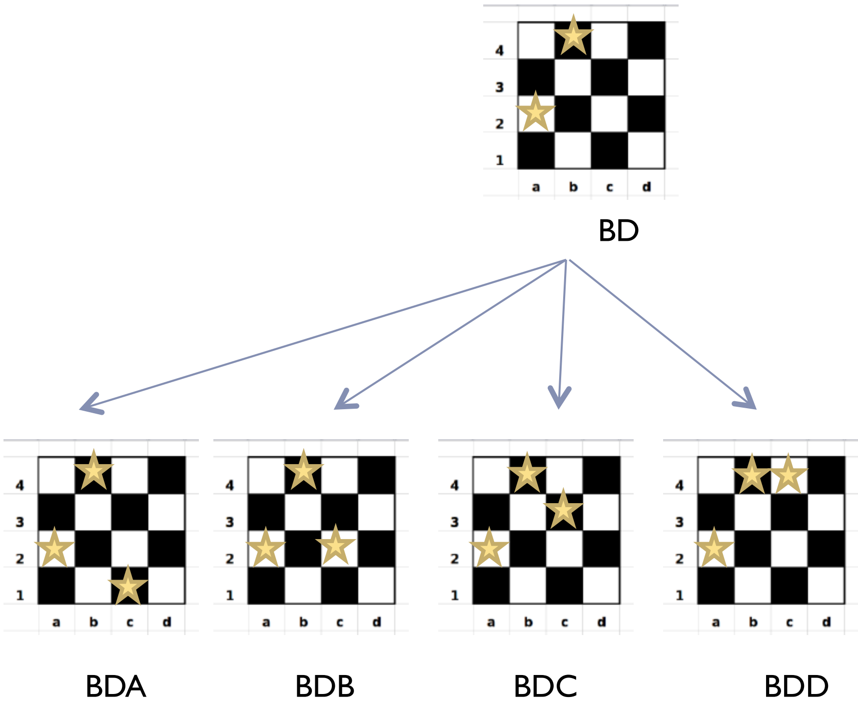
- Back tracking!! Try another position for first queen:



- Not valid candidates are discarded

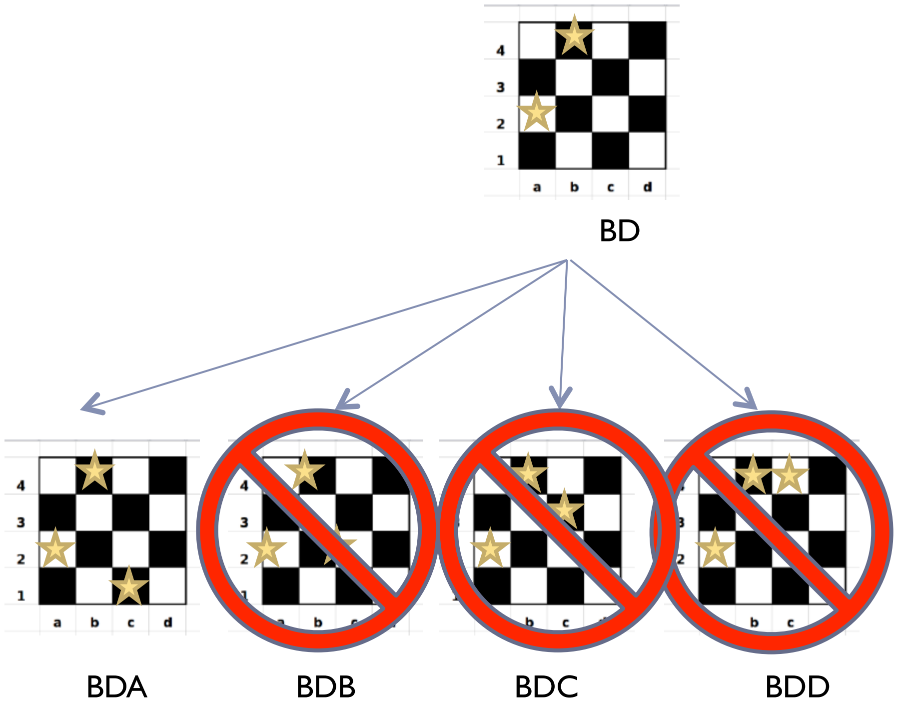
Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

- **Let's evaluate BD candidate:**



Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

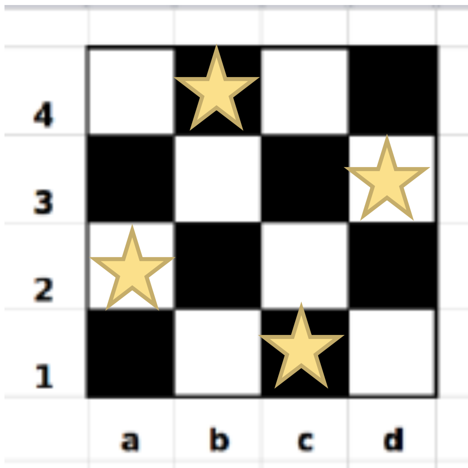
- Let's evaluate BD candidate:



- Not valid candidates are discarded

Example. Placing 4 queens in a 4x4 chessboard so that no queen attacks any other

- And so on …



BDAC

Example. Placing 30 queens in a 30x30 chessboard so that no queen attacks any other

```
Queens resuelto en 19237575 iteraciones. Fin
Mostrando tablero...
Q . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . Q . . . . . . . . . . . . . . . . . . . . . . . . . .
. Q . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . Q . . . . . . . . . . . . . . . . . . . . . . . . .
. . Q . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . Q . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . Q . . . . . . .
. . . . . . . . . . . . . . . . . . . . Q . . . . . . . . .
. . . . . Q . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . Q . . . . .
. . . . . . Q . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . Q . . . . . . . .
. . . . . . . Q . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . Q . . . . . . . . . . .
. . . . . . . . Q . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . Q . .
. . . . . . . . . Q . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . Q . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . Q
. . . . . . . . . . . . . . . . . . . . . . . . . . Q . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . Q .
. . . . . . . . . . . . . . . . . . . . . . . . . Q . . . .
. . . . . . . . . . . . . . Q . . . . . . . . . . . . . . .
. . . . . . . . . . . Q . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . Q . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . Q . . . . . . . . . . . . .
. . . . . . . . . . . . Q . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . Q . . . . . . . . . . . .
. . . . . . . . . . . . . . . Q . . . . . . . . . . . . . .
. . . . . . . . . . Q . . . . . . . . . . . . . . . . . . .
```

Summary Backtracking

- For problems where the solution can be seen as a 'data sequence'
  - The solution is a set of decissions among several possible candidates
  - All possible cases constitute a set that can be seen as a decision tree (a tree with conditions in internal nodes and candidates at leaves)
- At each decission, a recursive call is done and every k-promising candidate is tested
- It is possible to have clear criteria to assign to tree nodes
  - K-promising partial solutions must be evaluated as true or false
- Clear criteria to finalize the algorithm
  - Do we want all solutions or only one?
  - Identify global solution
  - Avoid infinite loops (of course)

Backtracking applications

- Graphs painting
- Hamiltonian path
- Combinatorial optimizations (knapsack problem…)
- Labyrinth resolution
- Prisoner's dilemma
- Resources management,
- Chess, dominoes, cards games, scrabble, … and **SUDOKUS** (the next weekly work)

Main Algorithm Strategies

- Recursive algorithms
- Divide and Conquer algorithms
- Backtracking algorithms
- **Dynamic programming algorithms**
- Greedy algorithms
- Brute force algorithms
- Branch and bound algorithms
- Heuristic algorithms

Heuristics and Optimization, Course 3ª, Semester 1º

Dynamic Programming Algorithm

- Based on remembering past results
- Approach:
  - Divide problem into smaller subproblems
    - Subproblems **must be of same type**
    - Subproblems **must overlap**
  - Solve each subproblem recursively
    - May simply look up solution (if previously solved)
  - Combine solutions to solve original problem
  - Store solution to problem
- For optimization problems.

```java
// Fibonacci Series using Dynamic Programming
class fibonacci
{
   static int fib(int n)
    {
        /* Declare an array to store Fibonacci numbers.
    int f[] = new int[n+1];
    int i;

    /* 0th and 1st number of the series are 0 and 1*/
    f[0] = 0;
    f[1] = 1;

    for (i = 2; i <= n; i++)
    {
       /* Add the previous 2 numbers in the series
         and store it */
        f[i] = f[i-1] + f[i-2];
    }

    return f[n];
    }

    public static void main (String args[])
    {
        int n = 9;
        System.out.println(fib(n));
    }
}
```

Divide and conquer vs Dynamic Programming

- Both paradigms divide the problem into subproblems, recursively solve them and combine their solutions.
- Choose Divide and Conquer when subproblems must be solved only once. For example: binary search o mergesort.
- Other wise, Dynamic Programming. For example: fibonacci.

Main Algorithm Strategies

- Recursive algorithms
- Divide and Conquer algorithms
- Backtracking algorithms
- Dynamic programming algorithms
- **Greedy algorithms**
- Brute force algorithms
- Branch and bound algorithms
- Heuristic algorithms

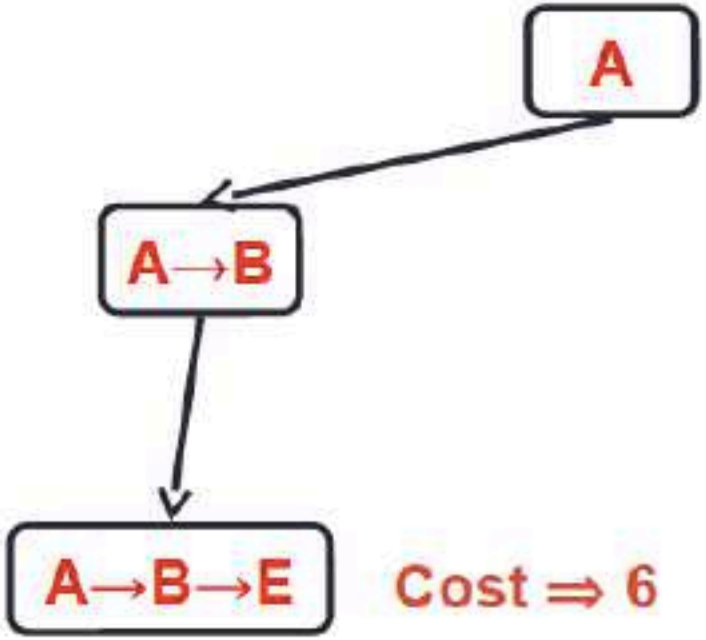
Heuristics and Optimization, Course 3ª, Semester 1º

Greedy Algorithm

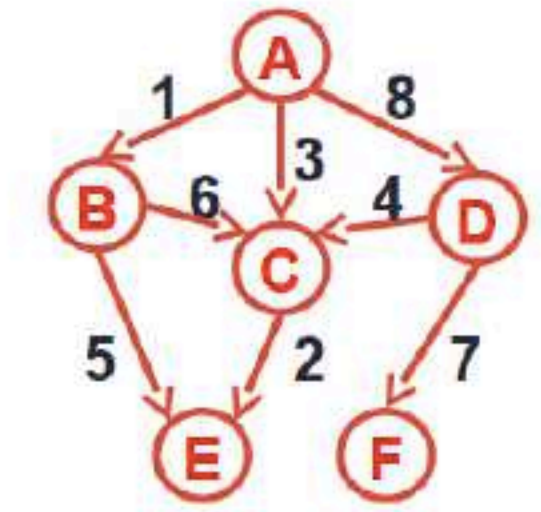
- Based on trying best current (local) choice results
- Approach:
  - At each step of algorithm
  - Choose best local solution
- Avoid backtracking, exponential time $O(2^n)$.
- Hope local optimum lead to global optimum

Greedy Algorithm

- Example (Shortest Path from A to E)
  - Choose lowest-cost neighbor



Cost ⇒ 6



Does not obtain the global shortest path!!!

Main Algorithm Strategies

- Recursive algorithms
- Divide and Conquer algorithms
- Backtracking algorithms
- Dynamic programming algorithms
- Greedy algorithms
- **Brute force algorithms**
- Branch and bound algorithms
- Heuristic algorithms

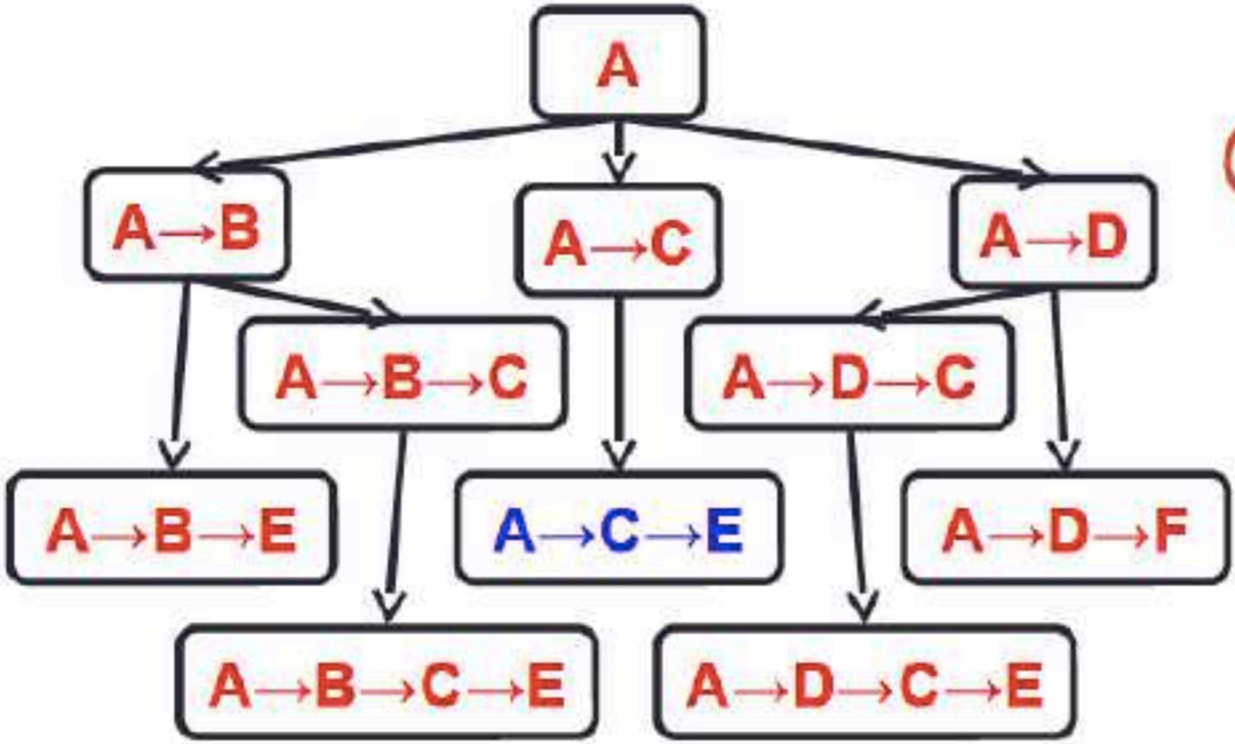
Heuristics and Optimization, Course 3ª, Semester 1º

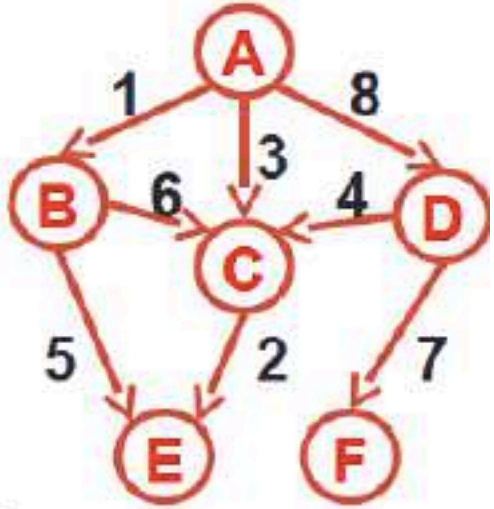
Brute force Algorithm

- Based on trying all possible solutions
- Most expensive approach
- Approach:
  - Generate and evaluate possible solutions until
  - Best solution is found (if can be determined)
  - All possible solutions found
    - Return best solution
    - Return failure if no satisfactory solution

Brute force Algorithm

- Example (From A to E)





Main Algorithm Strategies

- Recursive algorithms
- Divide and Conquer algorithms
- Backtracking algorithms
- Dynamic programming algorithms
- Greedy algorithms
- Brute force algorithms
- **Branch and bound algorithms**
- Heuristic algorithms

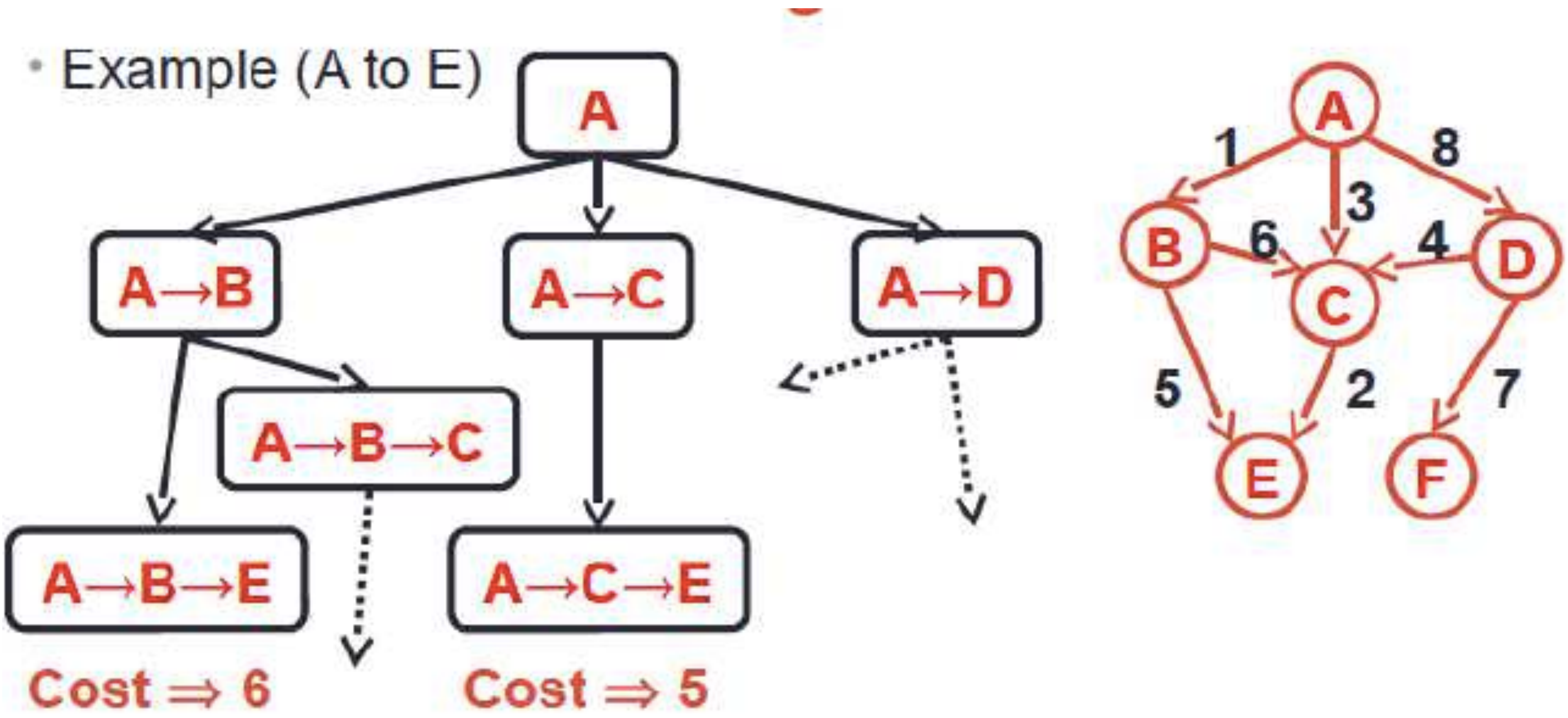
Heuristics and Optimization, Course 3ª, Semester 1º

Branch and bound Algorithm

- Based on limiting search using current solution
- Approach
  - Track best current solution found
  - Eliminate (prune) partial solutions that can not improve upon best current solution
- Reduces amount of backtracking
- Not guaranteed to avoid exponential time

Branch and bound Algorithm

- Example (A to E)



Main Algorithm Strategies

- Recursive algorithms
- Divide and Conquer algorithms
- Backtracking algorithms
- Dynamic programming algorithms
- Greedy algorithms
- Brute force algorithms
- Branch and bound algorithms
- **Heuristic algorithms**

Heuristics and Optimization, Course 3ª, Semester 1°

Heuristic Algorithm

- Based on trying to guide search for solution
- Heuristic => “rule of thumb”
- Approach
  - Generate and evaluate possible solutions
    - Using “rule of thumb”
    - Stop if satisfactory solution is found
- Can reduce complexity
- Not guaranteed to yield best solution

Heuristic Algorithm

- Example (A to E)
  - Try only edges with cost < 5