

Writing: Algorithms and Definitions

Javier Larrosa

UPC Barcelona Tech

.

- In CS we often write algorithms (that we have previously implemented and tested)
- Writing the algorithm differs from implementing it
 - Writing is for communication purposes (clarity rather than efficiency)
 - Writing is done a posteriori (when key ideas are more clear)
 - Writing means coding your thoughts (helps you settle down ideas)

Writing Algorithms: example

Social Networks

Consider a Social Network (e.g. facebook). We are concerned about its users and who they follow.

Data

The set of users is S and $L(u) \subseteq S$ is the set of people followed by $u \in S$

Celebrities

In a social sub-network a **celebrity** is a person that everybody knows, but does not know anybody. Formally, a user $u \in S$ is a celebrity if $|L(u)| = 0$ and for all $v \in S$ such that $v \neq u$, we have that $u \in L(v)$

Goal

Find an efficient algorithm to find **celebrities**

Algorithm (ugly)

Assume that users are $1..n$

Function *Celebrity* **is**

```
   $i := 1;$   
  for  $j := 2..n$  do  
    | if  $j \in L(i)$  then  $i := j;$   
  end  
  if  $|L(i)| \neq 0$  then return false;  
  for  $j := 1..n$  s.t.  $j \neq i$  do  
    | if  $i \notin L(j)$  then return false;  
  end  
  return true;  
end
```

Algorithm (ugly)

Invariants:

- i may be a celebrity
- $[1..j]$ (except i) are not celebrities

Cost: $O(n)$

if $j \in L(i)$ is $O(1)$ which can be implemented with an array of booleans

Algorithm 2

Abstraction of the problem

Think of the network as a **directed graph** $G = (V, E)$

Two useful properties

- 1 There is at most one celebrity
- 2 For any pair of vertices (u, v) , if $(u, v) \in E$ then u is not a celebrity, else v is not a celebrity

Algorithm 2

Function *Celebrity* ($G = (V, E)$) **is**

$S := V$;

while $|S| > 1$ **do**

$(i, j) := S.\text{FetchPair}()$;

if $(i, j) \in E$ **then** $S.\text{Push}(j)$;

else $S.\text{Push}(i)$;

end

return $\text{IsCelebrity}(G, S)$;

end

Function *IsCelebrity* (G, v) **is**

foreach $w \in V$ s.t. $w \neq v$ **do**

if $(w, v) \notin E \vee (v, w) \in E$ **then return** false;

end

return true;

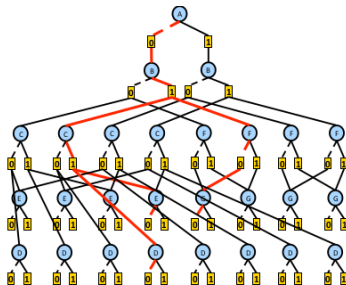
end

Invariants: S set of potential celebrities

Writing Algorithms: Example

Search on an AND/OR tree

Find the minimum cost AND/OR solution sub-tree



(b) Example AND/OR search graph. A solution tree is highlighted.

Example

Input: A graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, pseudo-tree \mathcal{T} , heuristic function $h(\cdot)$

Output: Optimal solution to \mathcal{M}

```
1 Create root OR node labeled by  $X_1$  and let the stack of created but not expanded
  nodes  $OPEN \leftarrow \{s\}$ 
2 Initialize  $v(s) \leftarrow \infty$  and the best partial solution tree rooted by  $s$ ,
   $T^*(s) \leftarrow \emptyset, UB \leftarrow \infty$ 
3 while  $OPEN \neq \emptyset$  do
    // Node Expansion
4   Select top node  $n$  in  $OPEN$ 
5   if  $n$  is an OR node labeled  $X_i$  then
6     foreach  $x_i \in \mathbf{D}_i$  do
7       Add AND child  $n'$  labeled  $\langle X_i, x_i \rangle$  to the list of successors of  $n$ 
8       Initialize  $v(n') \leftarrow 0$  and the best partial solution tree rooted by  $n'$ ,
         $T^*(n') \leftarrow \emptyset$ 
9   else if  $n$  is an AND node labeled  $\langle X_i, x_i \rangle$  then
10    foreach OR ancestor  $k$  of  $n$  do
11      Recursively evaluate the cost of the partial solution tree rooted by  $k$ ,
        based on the heuristic function  $h(\cdot)$ , assign its cost to  $f(k)$ 
        // See Algorithm 2
12      if evaluated partial solution is not better than the current upper bound at
         $k$  (e.g.,  $f(k) \geq v(k)$ ) then
13        Prune the subtree below the current tip node  $n$ 
14      else
15        foreach successor  $X_j$  of  $X_i \in \mathcal{T}$  do
16          Add OR child  $n'$  labeled  $X_j$  to the list of successors of  $n$ 
17          Initialize  $v(n') \leftarrow \infty$  and the best partial solution tree rooted by  $n'$ ,
             $T^*(n') \leftarrow \emptyset$ 
```

Example

```
18  Add successors of  $n$  on top of  $OPEN$ 
    // Bound Propagation
19  while list of successors of node  $n$  is not empty do
20      Let  $p$  be the parent of  $n$  if node  $n$  is the root node then
21          return solution:  $v(n), T^*(n)$ 
22      else
23          if  $p$  is an AND node then
24               $v(p) \leftarrow v(p) + v(n), T^*(p) \leftarrow T^*(p) \cup T^*(n)$ 
25          else if  $p$  is an OR node then
26              if the new value of  $v(p)$  is better than the old one (e.g.
27                   $v(p) > c(p, n) + v(n)$ ) then
28                   $v(p) \leftarrow c(p, n) + v(n), T^*(p) \leftarrow T^*(p) \cup \langle X_i, x_i \rangle$ 
29      Remove  $n$  from the list of successors of  $p$ 
      Move one level up:  $n \leftarrow p$ 
```

Algorithm 2: Recursive Computation of Heuristic Evaluation Function (evalPST)

Input: Partial solution subtree $T(n)$ rooted at node n , heuristic function $h(\cdot)$

Output: Heuristic evaluation function value $f(T(n))$

```
1 if  $n$  has no successors then
2   if  $n$  is an AND node then
3     return 0
4   else
5     return  $h(n)$ 
6 else
7   if  $n$  is an AND node then
8     Let  $k_1 \dots k_l$  be the OR children of  $n$ 
9     return  $\sum_{i=1}^l \text{evalPST}(T(k_i), h(\cdot))$ 
10  else if  $n$  is an OR node then
11    Let  $k$  be the AND child of  $n$ 
12    return  $c(n, k) + \text{evalPST}(T(k), h(\cdot))$ 
```

Example

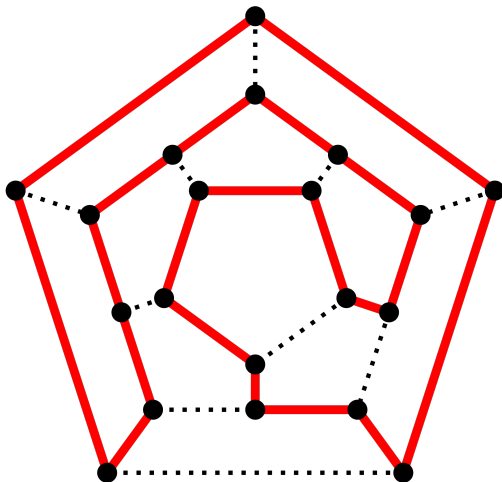
```
1  Function BBor( $n, ub$ )
2  begin
3    for  $m \in ch(m)$  do
4      if  $c(n, m) + h(m) < ub$  then
5        |  $ub := c(n, m) + BBand(m, ub - c(n, m))$ 
6      end
7    end
8    return  $ub$ ;
9  end

10 Function BBand( $n, ub$ )
11 begin
12   if  $ch(n) = \emptyset$  then return 0;
13   foreach  $m \in ch(n)$  do  $q(m) := h(m)$ ;
14   foreach  $m \in ch(n)$  do
15     | if  $\sum_{m' \in ch(n)} q(m') \geq ub$  then return  $ub$ ;
16     |  $q(m) := BBor(m, ub - \sum_{m' \in ch(n), m' \neq m} q(m'))$ ;
17   end
18   return  $\sum_{m \in ch(n)} q(m)$ ;
19 end
```

In Scientific Writing, we often have to write **definitions**

- Definitions can be given in **words** or in **mathematical writing**
- Sometimes we give both
- Sometimes we add an example

Hamiltonian Graph



Hamiltonian

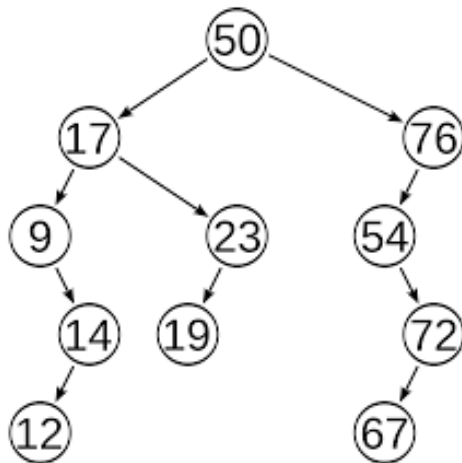
Hamiltonian

A graph is Hamiltonian if it contains a **loop** such that every vertex in the graph is **visited** exactly once

Hamiltonian (more formal)

Let $G = (V, E)$ be an undirected graph with $V = \{1, 2, \dots, n\}$. A *path* of length k is a sequence of vertices v_1, v_2, \dots, v_k such that (v_i, v_{i+1}) is in E . A *cycle* is a path that starts and ends at the same vertex (i.e, $v_1 = v_k$). A cycle is *proper* if the only repetition is the starting/ending vertex. We say that G is *Hamiltonian* if there is a proper cycle of size $n + 1$

Binary Search Tree



Assume knowledge on trees (node, left child, right child,...)

Binary Search Tree

Consider binary trees where each vertex has an associated label. In a *binary search tree* every vertex satisfies that all the labels on its left (respectively, its right) are smaller than (respectively, greater than) or equal to its label.

Binary Search Tree

Notation

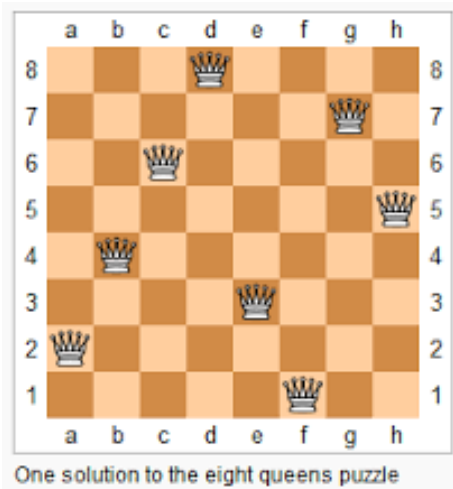
Consider a binary tree $T = (V, E)$ with *root* $r \in V$. Let $v \in V$ be an arbitrary vertex. The *sub-tree* on the *left* and *right* of v are noted $left(v)$ and $right(v)$, respectively. Let's assume that each vertex has associated a *numerical label* noted $label(v)$.

Binary Search Tree (more formally)

We say that T is a *binary search tree* if for every vertex v we have that:

- every label in $left(v)$ is smaller than or equal to $label(v)$
- every label in $right(v)$ is larger than or equal to $label(v)$

n-queens



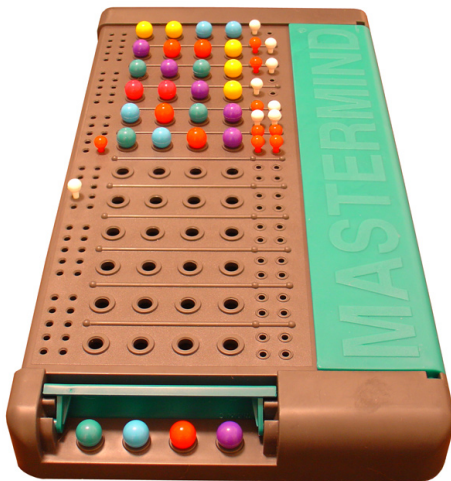
n -queens

The n -queens problem consists on placing n **queens** in an $n \times n$ chess board in such a way that no pair of queens **attack** each other. Recall that, according to chess rules, queens can **move** any number of positions along rows, columns and diagonals

n -queens (more formally)

Let (x_1, x_2, \dots, x_n) be a set of variables taking values in the range $1..n$. The n -queens problem consists on assigning one value to each variable in such a way that for all $i, j = 1..n$, with $i \neq j$:

- $x_i \neq x_j$ (not in the same column)
- $|x_i - x_j| \neq |i - j|$ (not in the same diagonal)



Mastermind

Mastermind is a *code-breaking* game for two players: the *codemaker* and the *codebreaker*. The codemaker makes up a code which keeps secret from the codebreaker. A *code* is a sequence of four colors out of 6 candidate colors. The same color can appear more than once in the code. The goal of the codebreaker is to discover the code as soon as possible during a sequence of guesses. After each guess the codemaker gives some feedback. In particular it tells the codebreaker how many colors have been guessed (irrespective of the position) and how many colors have been guessed in the right position. For example, if the code is (r, b, r, p) and the guess is (g, p, r, g) the feedback will be *you got two right colors, one of them in the right position*. The feedback will be used to refine the codebreaker guess in the subsequent iteration.

Mastermind (more formally)

A *code* is a sequence $C = (c_1, c_2, c_3, c_4)$ where each element is a number from 1 to 6. The game of *Mastermind* consists of discovering an unknown code C in a minimum number of guesses. After each guess $G = (g_1, g_2, g_3, g_4)$ the codebreaker gets the following feedback, which will be used to refine subsequent guesses:

- Number of guessed numbers irrespective of the position. Formally, for each number $i = 1..6$, let C_i be the number of times i occurs in C , and G_i the number of times i occurs in G . The feedback is,

$$\sum_{i=1}^6 \min\{C_i, G_i\}$$

- Number of positions where the guess matches with the code. Formally, number of positions $i = 1..4$ such that $g_i = c_i$

Round Robin Schedule

Microsoft Excel - IanBalancedRoundRobin_v3.2.xls

File Edit View Insert Format Tools Data Window Help Adobe PDF

9

Generate

☐ Court or Field Format

☒ Home/Away Format

☒ Use Letters for upto 26 teams

Help

	round 1	round 2	round 3	round 4	round 5	round 6	round 7	round 8	round 9
team A		C	E	G	I	B	D	F	H
team B	I		D	F	H	A	C	E	G
team C	H	A		E	G	I	B	D	F
team D	G	I	B		F	H	A	C	E
team E	F	H	A	C		G	I	B	D
team F	E	G	I	B	D		H	A	C
team G	D	F	H	A	C	E		I	B
team H	C	E	G	I	B	D	F		A
team I	B	D	F	H	A	C	E	G	

Round Robin Schedule

Round Robin

Consider a two-teams sport for which a tournament with n teams has to be scheduled. A round robin schedule is the pairing of the teams during $n - 1$ weeks in such a way that any pair of teams are paired exactly once. We can assume n being an even number, since otherwise we can add a dummy team. Playing against the dummy team represents a resting week

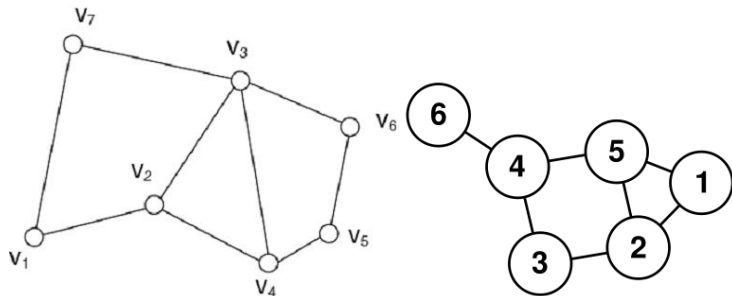
Round Robin Schedule

Round Robin (more formal)

Let n be an even natural number. A round robin schedule is a matrix $M_{n \times n-1}$ such that for all $1 \leq i \leq n, 1 \leq j < n$:

- Teams play teams: $M(i, j) \in [1..n]$ and $M(i, j) \neq i$
- Teams are paired: if $M(i, j) = k$, then $M(k, j) = i$
- Same two teams only play once: $\forall 1 \leq j < j' < n, M(i, j) \neq M(i, j')$

Induced Width



Induced Width

Let $G = (V, E)$ be an undirected graph with $V = \{1, 2, \dots, n\}$. The **neighbors** of vertex $v \in V$ is the set of vertices adjacent to v . Its **higher neighbors** are those whose index is larger than v . The **width** of v is the number of higher neighbors that it has. The **width** of G is the maximum width among all its vertices.

The **induced graph** $G^* = (V, E^*)$ is obtained from G as follows: take its vertices in increasing order. When considering vertex v connect with a new edge any pair of higher neighbors that are not already connected.

The **induced width** of G , is the width of its induced graph.

Width

Let $G = (V, E)$ be an undirected graph with $V = \{1, 2, \dots, n\}$. The **neighbors** of vertex $v \in V$, noted $N(v)$ is the set of vertices adjacent to v ,

$$N(v) = \{u \in V \mid (v, u) \in G\}$$

The **higher neighbors** of v , noted $HN(v)$ are those whose index is larger than v ,

$$HN(v) = \{u \in V \mid u > v, (v, u) \in G\}$$

The **width** of v , noted $w(v)$ is the number of higher neighbors (i.e, $W(v) = |HN(v)|$). The **width** of G is the maximum width among all its vertices,

$$W(G) = \max_{v \in V} W(v)$$

Induced Width

The **induced graph** $G^* = (V, E^*)$ is obtained from G as follows (see Algorithm 3): take its vertices in increasing order. When considering vertex v connect with a new edge any pair of higher neighbors that are not already connected.

The **induced width** of G is the width of its induced graph G^* .

Function *InducedGraph* (G) is

```
 $G^* = (V^*, E^*)$  is a graph;  
 $G^* \leftarrow G$ ;  
foreach  $v \in V$  in increasing order  
  do  
    for  $u, w \in HN(v)$  do  
      if  $(u, w) \notin E^*$  then  
         $E^* \leftarrow E^* \cup \{(u, w)\}$ ;  
      end  
    end  
  end  
return  $G^*$ ;
```

end

Algorithm 1: Computation of the induced graph G^* of a graph G . $HN(v)$ denotes the set of neighbors of v with index higher than v in the graph that is being computed G^*

Examples

- <https://www.solitairenetwork.com/solitaire/accordion-solitaire-game.html>
- <https://www.solitairenetwork.com/solitaire/black-hole-solitaire-game.html>
- <https://dominoes.playdrift.com/>
- <http://www.playzgame.com/online-flash-games/Eternity-II.php>

2 Formal:

Game is based on a matrix M that contains number of cells C . Each cell C can have two sets, CE cell empty or CM cell with mine. Player one sets the matrix that $CE+CM=M$. Player two selects the cell C that transforms to CE or CM . If it is CM then the game is over. If it is CE game continues, Some CE have a hint on them that belongs to set 1,2,3. Hint corresponds to number of CM in 8 fields around. Strategy for the first player can be to

set matrix M that $CM > CE$. Tactics for the second player can be to reveal hints with a higher number from given set.

MineSweeper: 2nd ex

Minesweeper is a two players game in which one player has the role of a **mine layer** and the other of a **mine sweeper**. The game board has a $n \times n$ size, dividing it in n^2 **cells**. The set of cells is C and a cell is represented by c_{ij} , where i is the row and j the column, it has a state $StateC_{ij}$ and a content $ContentC_{ij}$, where:

- $StateC_{ij} = \text{discovered or unknow or flagged}$
- $ContentC_{ij} = \text{number or mine; number} \in [0..8]$

The number indicates how many mined cells are around, in case the cell doesn't have a mine:

```
number  $\leftarrow$  0;
for ( $x \leftarrow i - 1$ ;  $x \leq i + 1$ ;  $x++$ ;) do
  for ( $y \leftarrow j - 1$ ;  $y \leq j + 1$ ;  $y++$ ;) do
    if  $ContentC_{xy} = \text{mined}$  then
      | number ++;
    end
  end
end
end
```

MineSweeper: 2nd ex

The mine sweeper can only see $ContentC_{ij}$ if $StateC_{ij} = discovered$, when the game starts the mine layer hides an x number of mines, and sets all the cells to unknown: $\forall c \in C, State_c = unknown$. The mine sweeper has to guess where the mines are, and set $StateC_{ij} = discovered$. If he reveals one mine **he loses immediately**. Mine swapper can also **flag** as many cells as x , number of mines, if he thinks there is a mine. **Mine sweeper wins if:** $\forall c \in C, State_c = discovered$ or flagged.

MineSweeper: 3rd ex

Minesweeper is a game of two players (p_1, p_2) . Given a matrix of size $N \times M$ where $N, M \in \mathbb{Z}^+$ where each of the cells $c_{i,j}$ can contain:

- Mine m
- Number of mines nearby (n) $n \in \mathbb{Z}^+$ where $n = \sum_{i,j}^{N,M} (Neight(c_{i,j})) \in m$.
We define a cell neighbourhood as $Neight(c_{i,j}) = c_{i-1,j-1}, c_{i-1,j}, c_{i-1,j+1}, c_{i+1,j}, c_{i+1,j+1}, c_{i+1,j-1}, c_{i-1,j+1}, c_{i,j+1} \parallel 0 \leq i, j \leq N, M$.

When is p_1 turn a player can:

- Reveal a cell $reveal_{p_1}(c_{i,j})$ where $0 \leq i, j \leq N, M$. This makes propagation when $\sum_{i,j=0}^{N,M} costCell(c_{i,j}) = 0 \implies reveal(c_{i,j})$.

MineSweeper: 3rd ex

- Flag a cell $flag_{p_1}(c_{i,j})$ where $0 \leq i, j \leq N, M$ and $flag(c_{i,j}) \neq 1$
- Unflag a cell $unflag_{p_1}(c_{i,j})$ where $0 \leq i, j \leq N, M$ and $flag(c_{i,j}) = 1$

The game ends when p_1 :

- Uses $reveal_{p_1}(c_{i,j})$ and p_2 detects that $c_{i,j} \in m$.
- $flaggedCells = \sum_{i,j=0}^{N,M} c_{i,j} \in m$ and p_2 verifies $c_{i,j}$ are correct.

Otherwise the game continues.

Note that if $c_{i,j}$ is surrounded by $costCell = 1$ then $c_{i,j} \in m$. Similarly per bigger costs.