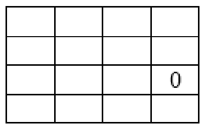
02/06/2020 **ECE 479/579 Module _3**

<u>Example</u>

4 queen problem (illustrate the efficient of good control strategy)

- Present as AI production system
  - Database (DB)
    - 4X4 board
    - Initial state no queen marks (X) on this board
    - Goal state a queen mark exactly one in each row and column of the board placed in such a way that no two (X) marks capture each other (do not have them along diagonal in same row and column)
  - Operators (Rules)
    - $Ri,j \ for\ 1 \leq i \leq 4\,, \quad 1 \leq j \leq 4$
    - preconditions


R34

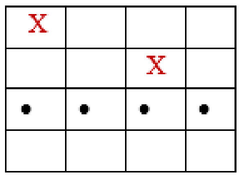
$for\ i = 1, \quad 1 \leq j \leq 4$ There is no mark on the board
Then (action) place (X) in row = 1 , column j

R1,1
R1,2
R1,3
R1,4

R1j

$for \quad 2 \leq i \leq 4\,, \ 1 \leq j \leq 4$ There is an (X) mark in row $i - 1$
Then (action) place (X) in row i

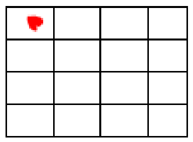
$Rij \ for \ \ 2 \leq i \leq 4 \ , \ 1 \leq j \leq 4$ if there is a mark in row $i - 1$
Then place(X) in cell (i, j)



<u>Note</u>

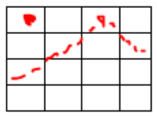
- not incorporated into the rules a way of checking Whether correct place or not

Let us introduce a metric for a cell in the 4X4 board



diag (i,j) → the length of the longest diagonal crossing cell(i,j)

What is the diagonal of cell (1, 1)? What is length of longest diagonal?

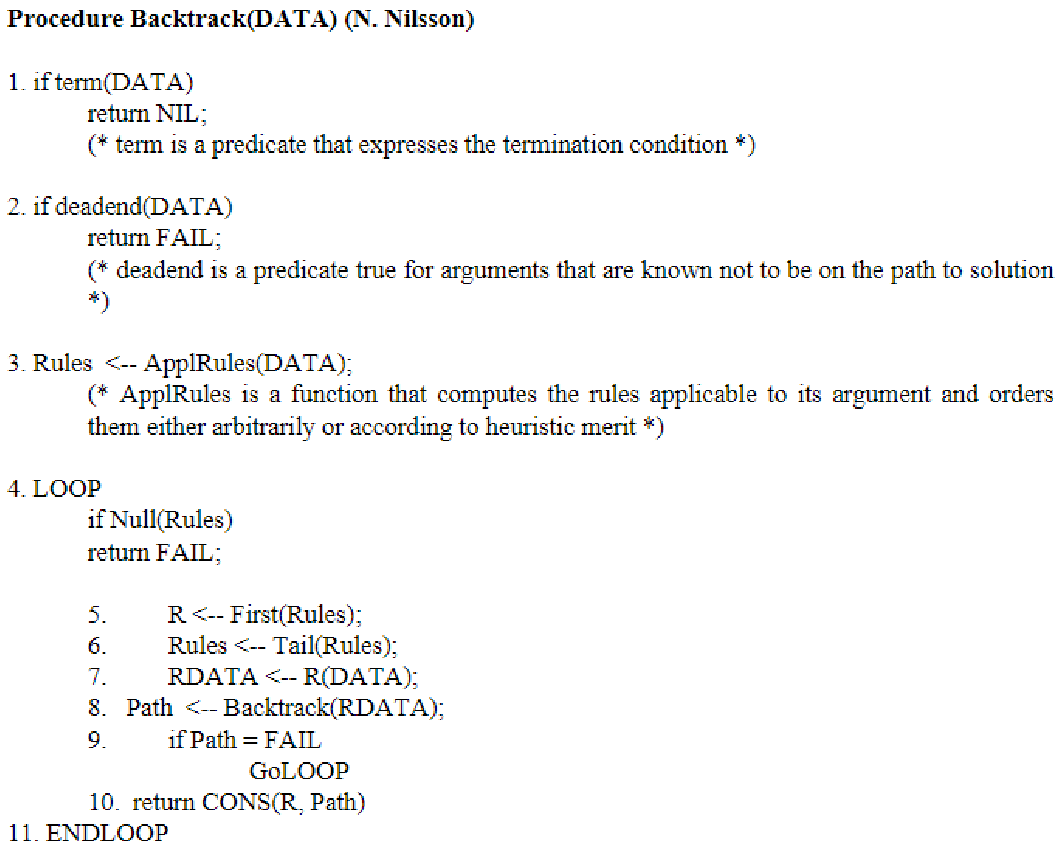


diag (1,1) = 4 dig(1,3) = 3

- Let's take the first row and no (X) mark there. It could place the X mark R_{11} , R_{12} , R_{13} , R_{14}
- Using the diagonal measure as a way of ordering the rules and select one that may give us advantage.
  $Ri,m \ < Rit \quad if \ \ diag\ (i,m) < diag(i,t)$

Obviously, no having choice between R_{11}, R_{12} !

- Control Strategy (CS)
    - <u>*Ordering*</u>
        - **A.** Ordering
        - ***B.*** *If the diag(i,m)= diag(i,t) then use index based ordering that is* $R_{im} < R_{it}$ *if* $m<t$



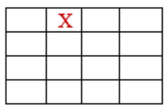
Procedure Backtrack(DATA) (N. Nilsson)

```
1. if term(DATA)
        return NIL;
        (* term is a predicate that expresses the termination condition *)

2. if deadend(DATA)
        return FAIL;
        (* deadend is a predicate true for arguments that are known not to be on the path to solution
        *)

3. Rules  <-- ApplRules(DATA);
        (* ApplRules is a function that computes the rules applicable to its argument and orders
        them either arbitrarily or according to heuristic merit *)

4. LOOP
        if Null(Rules)
        return FAIL;

        5.      R <-- First(Rules);
        6.      Rules <-- Tail(Rules);
        7.      RDATA <-- R(DATA);
        8.   Path  <-- Backtrack(RDATA);
        9.      if Path = FAIL
                        GoLOOP
        10.  return CONS(R, Path)
11. ENDLOOP
```

<u>Procedure backtracking</u>

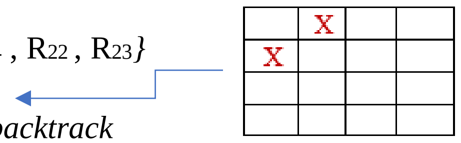
1) *Term(data)* $\rightarrow$ *No*
2) *Deadend* $\rightarrow$ *No*
3) *Rule* $\rightarrow$ *apply rule*

    $\hookrightarrow$ R_{11} , R_{12} , R_{13} , R_{14}
4) Loop
5) R $\leftarrow$ R_{12} (arrange according to the diagonal R_{12} , R_{13} , R_{11} , R_{14})
6) Rules –{ R_{11} , R_{13} , R_{14}}
7) RData $\dashrightarrow$
8) Path $\leftarrow$ backtracking (data)

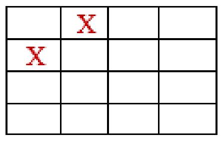


Rewrite

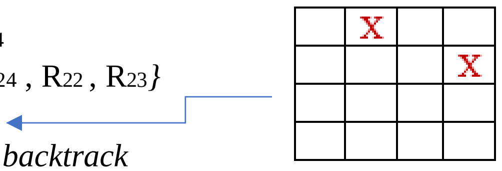
1) *No*
2) *No*
3) *Rules* –>{ R_{21} , R_{22} , R_{23} , R_{24}} $\rightarrow$ { R_{21} , R_{24} , R_{22} , R_{23}}
4) *Loop*
5) *R* $\leftarrow R_{21}$
6) *R* $\leftarrow$ { R_{24} , R_{22} , R_{23} *}*
7) *Rdate*
8) *Path* $\leftarrow$ *backtrack*



Rewrite



1) *No*
2) *Yes exit this call with fail*
3) *Rules* –>{ R_{21} , R_{22} , R_{23} , R_{24}} $\rightarrow$ { R_{21} , R_{24} , R_{22} , R_{23}}
4) *Loop*
5) *R* $\leftarrow R_{24}$
6) *R* $\leftarrow$ { R_{24} , R_{22} , R_{23} *}*
7) *Rdate*
8) *Path* $\leftarrow$ *backtrack*



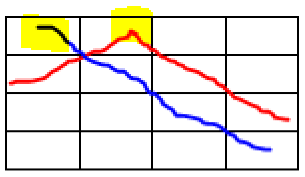
Making Recursive call

How to pick the order

Rule order

1) Diagonal (i,j)

(1,1) = 4
(1,2) = 3

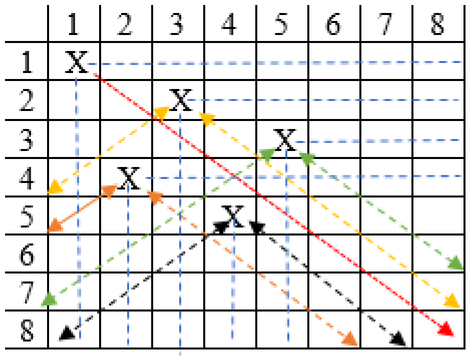


If you have choices between R_{11} and R_{12} then R_{12} has high priority

2) Index based $\rightarrow$ R_{11} , R_{12} , R_{13} , R_{14}

Continue with back tracking

4 queen problem

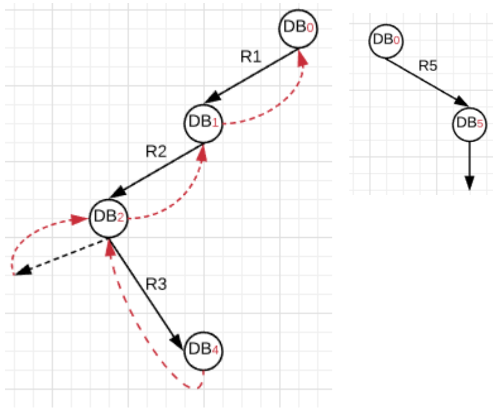


- A few comments
  - A good control strategy (CS) helps
  - consider the following situations
    - play queen mark in the row 6

      does not depend on the previous placement in row 5 but it is impacted with all other placements and that indicate that may be it is necessary to backtrack more than one step.

General concept of graph search

- Consider "this contrast" to classical backtracking



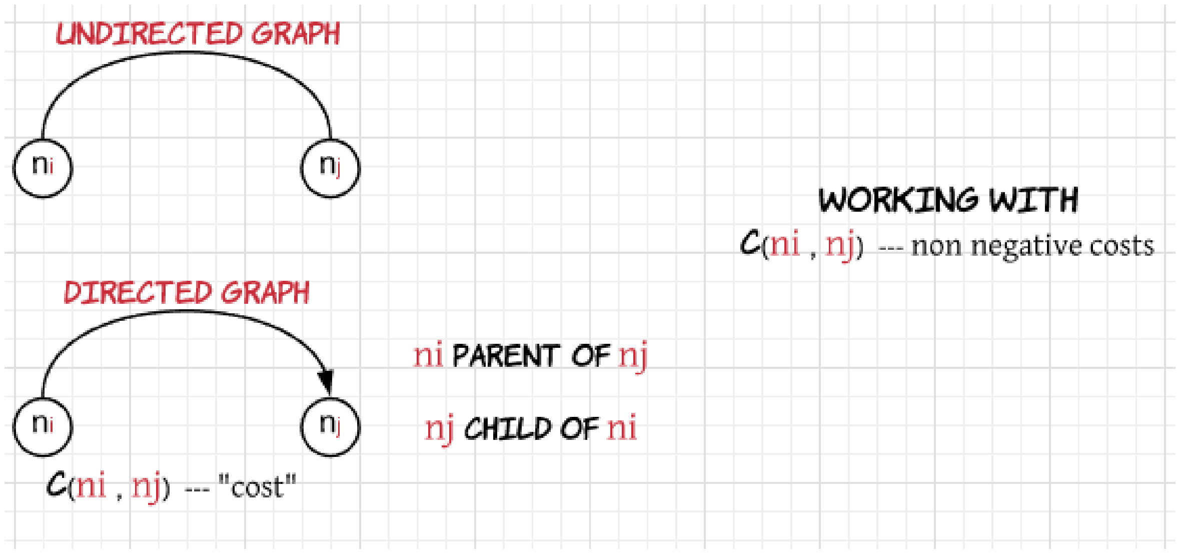
Note

There is no provision for keeping track of partial search tree or partially expand search space "not remember these routes".(cannot jump back to them to keep exploring them in a different way)

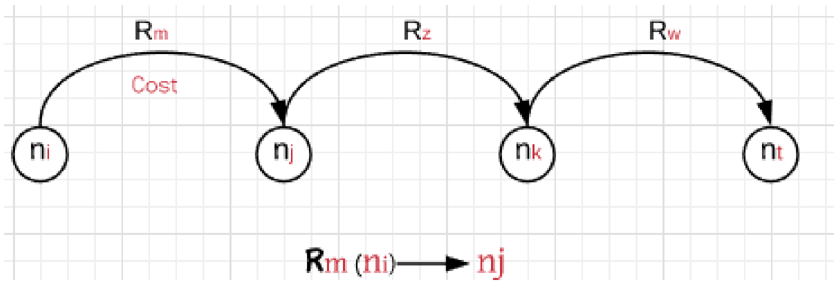
- In graph search, we preserve partially expanded search space so that we can return to it should we need to.

Basic Definitions (in graph)

- G=< V, E > where V is the set of nodes (nodes correspond to states)
  E is the set of edges (edges correspond to rules)

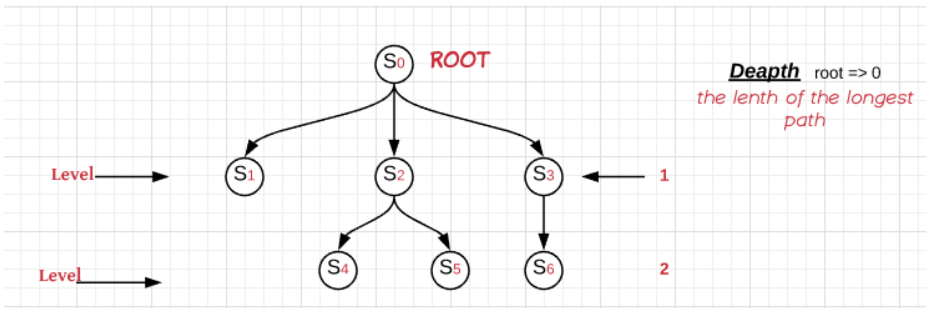


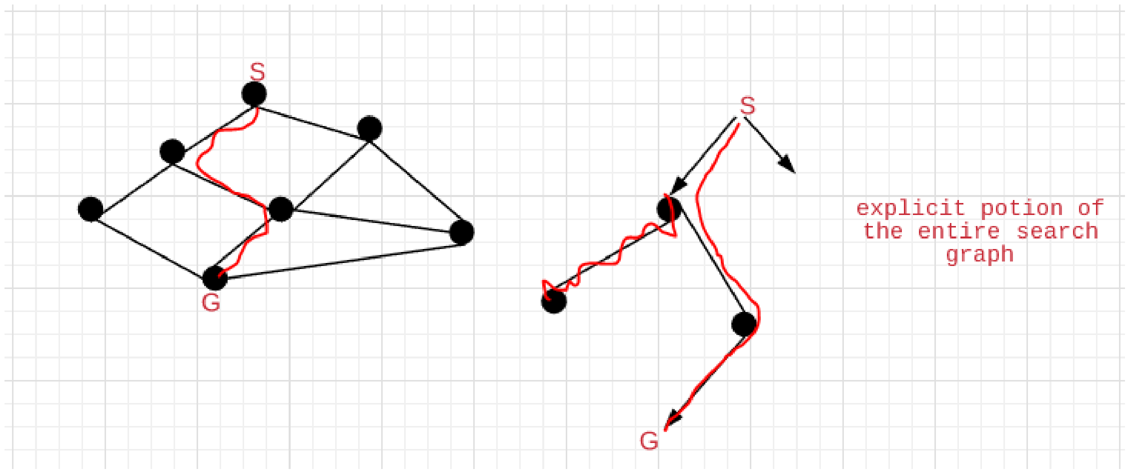
- We will work with finite graph
  - Each node, will have a finite number of children



A tree definition

- A graph in which each node (with the exception of one called the root) has exactly one parent





Procedure GRAPHSEARCH (Nils Nilsson)

1. Create a search graph, G, which consists only of the start node, s. s is placed on a list called OPEN.

2. Create a list called CLOSED. This list is initially empty.

3. LOOP: if Empty(OPEN), then exit with failure.

    4. n <-- First(OPEN)
    OPEN = OPEN - {n}
    CLOSED = CLOSED + {n}

    5. If Goal(n), exit successfully with the solution obtained by tracing a path along the pointers from n to s in G (pointers are established in step 7).

    6. Expand node n, generating the set, M, of its successors and install them as successors of n in G

    7. Establish a pointer to n from those members of M that were not already in G, i.e., on either CLOSED or OPEN. Add these members of M to OPEN.
    For each member of M that was already on OPEN or CLOSED, decide whether or not to redirect its pointer to n.
    For each member of M already on CLOSED, decide for each of its descendants in G whether or not to redirect its pointer.

    8. Reorder the list OPEN, either according to an arbitrary scheme or according to heuristic merit.

    9. End LOOP