

# Advanced School in Artificial Intelligence

## Strategie di ricerca

Strutture dati e algoritmi

Marco Alberti

[marco.alberti@unife.it](mailto:marco.alberti@unife.it)

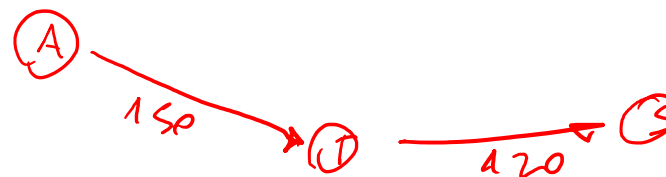
*Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021*



**Università  
degli Studi  
di Ferrara**

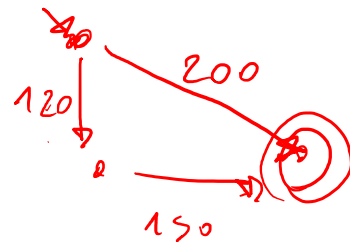
## Strutture dati

- **Node  $n$ :** uno per ogni stato esplorato
  - state: lo stato corrispondente a  $n$
  - parent: il nodo da cui è stato generato  $n$
  - action: l'azione applicata allo stato del nodo parent per ottenere lo stato di  $n$
  - path-cost: il costo del cammino dal nodo iniziale a  $n$
- **Frontier:** una coda (FIFO, LIFO o con priorità a seconda dell'algoritmo). I nodi trovati, ma non ancora espansi
- **Reached:** una lookup table (es. hash-table) con chiavi gli stati trovati e valori i nodi corrispondenti



## Gestione di loop in path e cicli sul grafo

- Siccome lo spazio degli stati è un grafo non necessariamente aciclico è possibile:
- che si producano loop all'interno di un cammino (A-B-A-B-A...)
  - si può evitare non aggiungendo alla frontiera nodi il cui cammino contiene un loop
- che lo stesso stato sia raggiungibile da più cammini
  - si può evitare tenendo in memoria gli stati raggiunti e non aggiungendo alla frontiera i nodi relativi a stati già raggiunti (aumenta memoria)
  - ma si può voler aggiungere comunque un nodo se il cammino è di costo inferiore





## Algoritmo di ricerca best first


**function** BEST-FIRST-SEARCH(*problem*, *f*) **returns** a solution node or failure

 *node*  $\leftarrow$  NODE(STATE=*problem*.INITIAL)

*frontier*  $\leftarrow$  a priority queue ordered by *f*, with *node* as an element [*node*]

*reached*  $\leftarrow$  a lookup table, with one entry with key *problem*.INITIAL and value *node*

**while not** IS-EMPTY(*frontier*) **do**

*node*  $\leftarrow$  POP(*frontier*) 


**if** *problem*.IS-GOAL(*node*.STATE) **then return** *node*

**for each** *child* **in** EXPAND(*problem*, *node*) **do** 

*s*  $\leftarrow$  *child*.STATE

**if** *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

*reached*[*s*]  $\leftarrow$  *child*

add *child* to *frontier* 

**return** *failure*

## Espansione nodo

**function** EXPAND(*problem*, *node*) **yields** nodes

$s \leftarrow node.STATE$

**for each** *action* **in** *problem.ACTIONS(s)* **do**

$s' \leftarrow problem.RESULT(s, action)$

$cost \leftarrow node.PATH-COST + problem.ACTION-COST(s, action, s')$

**yield** NODE(*STATE*= $s'$ , *PARENT*=*node*, *ACTION*=*action*, *PATH-COST*=*cost*)