UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

# Ricerca in Struttura Universitaria

*Caso di Studio di "Ingegneria della Conoscenza"*


Gruppo di lavoro

- Matteo Castano, 737890, m.castano@studenti.uniba.it
- Ugo Gabriele De Santis, 736572, u.desantis2@studenti.uniba.it


<URL repo associato, contenente il materiale completo>

AA 2022-23

# *Indice*

# Introduzione

L'obiettivo del caso di studio è quello di semplificare lo spostamento all'interno di una struttura universitaria calcolando il percorso più efficiente tra due posizioni inserite dall'utente.

Il percorso calcolato dipende dalle condizioni reali dell'edificio, perciò dalle posizioni dei nodi che lo compongono, ma anche dal loro stato: infatti vengono considerate eventuali inagibilità, immesse dall'utente allo start del programma (es. pavimenti bagnati). Viene tenuto conto, inoltre, di chi sta utilizzando il sistema (es. l'utilizzo degli ascensori o di alcuni corridoi "riservati" è limitato solo a determinati utenti privilegiati, l'accesso a determinate aule è consentito solo a determinati orari e per determinati utenti).

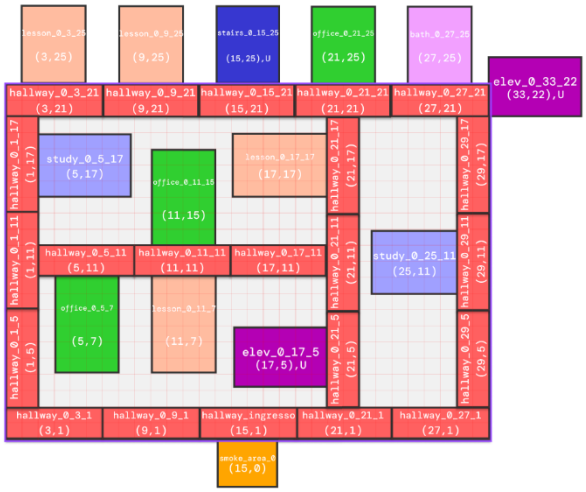
Per migliorare la ricerca è stata implementata una funzione euristica (i cui dettagli e proprietà saranno esposti successivamente) che stima quanto un nodo è vicino a un goal.

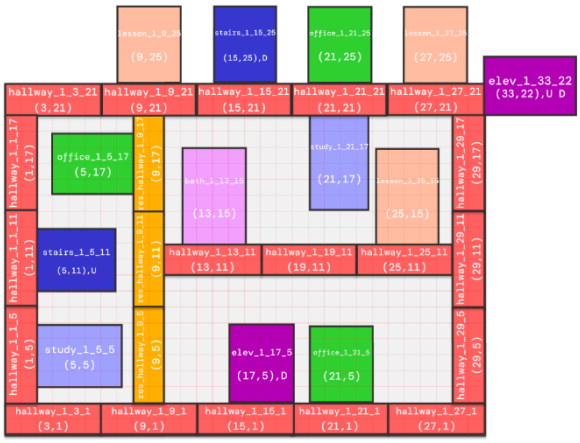
Il sistema permette di effettuare inoltre delle query di default, come la ricerca del bagno più vicino.

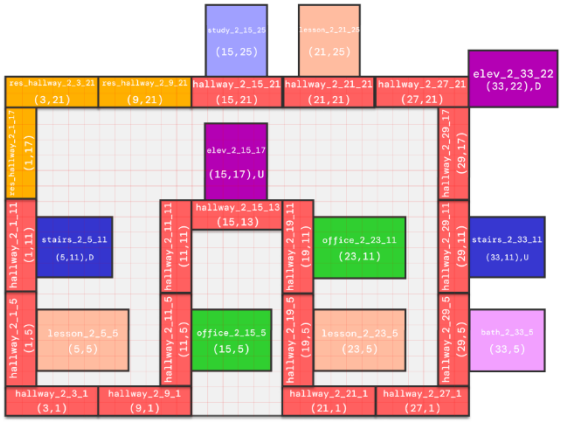
Lo step successivo all'implementazione del sistema è una analisi relativa alla computazione degli algoritmi analizzati: saranno confrontati l'algoritmo A* e l'algoritmo Lowest Cost First, entrambi con e senza Mutiple Path Pruning, per capire quanto la funzione euristica sviluppata incida sull'efficienza della ricerca nel nostro dominio.

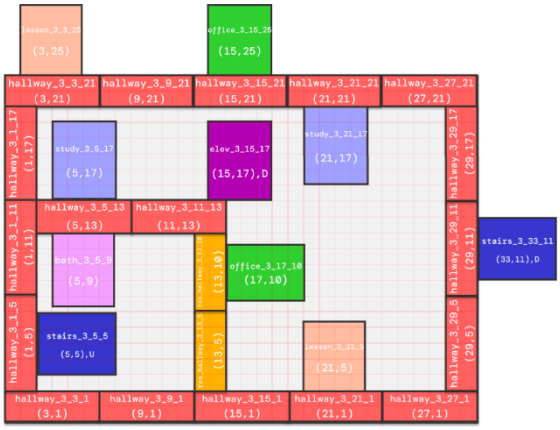
# Struttura dell'edificio

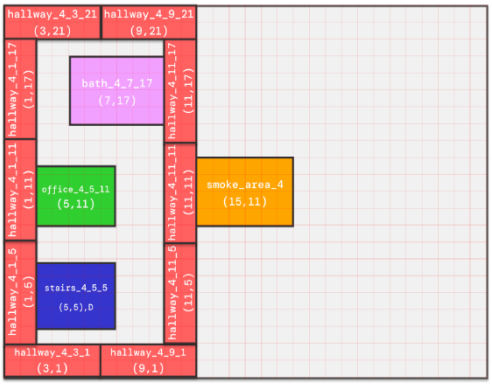
La planimetria dell'edificio è stata codificata nella base di conoscenza tramite i predicati che saranno illustrati nella parte dedicata alla KB ed è strutturata nella seguente maniera:











L'edificio è composto da cinque piani nei quali sono presenti diversi ambienti; essi si distinguono in:

- Aule per Lezioni (Rosa): esse sono accessibili in un determinato momento solo dai soggetti interessati alla lezione che si sta tenendo: dato lo scheduling delle lezioni e delle aule, un soggetto può quindi entrare solo se è il

professore della lezione che si sta tenendo oppure uno studente che segue il relativo corso.

- Aule Studio (Azzurro): esse sono accessibili in qualsiasi momento, ma solamente dagli studenti.
- Bagni (Violetto): essi sono accessibili da tutti in qualsiasi momento.
- Uffici dei professori (Verde): essi sono accessibili in qualsiasi momento, ma solo dal professore che possiede l'ufficio oppure dagli studenti che seguono un corso tenuto dallo stesso professore.
- Aree Fumatori (Arancione): esse sono accessibili da tutti in qualsiasi momento.
- Scale (Blu): esse collegano due piani e possono essere utilizzate da qualsiasi individuo.
- Ascensori (Viola): essi collegano due (o più, in alcuni casi) piani ma sono accessibili solo dai professori o da alcuni studenti che possiedono un permesso speciale.
- Corridoi: essi collegano tutti gli ambienti dello stesso piano, perciò sono collegati a altri corridoi e alle stanze dell'edificio; essi sono divisi in:
    - Corridoi Normali (Rosso): accessibili a tutti.
    - Corridoi Riservati (Oro): accessibili solo da professori oppure da studenti che hanno un determinato permesso.

Ogni ambiente è accessibile solo da chi ne ha autorizzazione ammesso che non ci siano condizioni che precludano il corretto utilizzo (nel nostro caso sono state previste solo le situazioni anormali di 'pavimento bagnato' o di 'problema generico').

## Utenti

Il KBS è stato progettato per essere utilizzato da due categorie utente: studenti e professori. All'avvio verrà chiesto all'utente di autentificarsi con il suo '*id_utente*', legato nella KB a una delle due categorie e a specifici permessi.

Il sistema potrebbe essere agilmente esteso per aggiungere ulteriori categorie, caratterizzate da specifici permessi sui relativi accessi (es. custode, guest, ecc).

# Sommario

Il sistema è implementato mediante l'impiego della libreria pyswip, che consente l'utilizzo del relativo interprete Prolog, per interrogare una base di conoscenza costruita in maniera semi-automatica (mediante script python per assiomatizzare i fatti e in maniera manuale per assiomatizzare le regole che descrivono le caratteristiche del nostro dominio).

Le interrogazioni sono fatte per costruire un apposito grafo di ricerca in cui i nodi sono i luoghi dell'edificio, mentre gli archi sono i collegamenti *disponibili* tra due nodi (la disponibilità dipende anche qui dalle situazioni normali/anormali dei nodi, dai permessi e altre situazioni codificate nella Knowledge Base).

La ricerca dei percorsi è stata implementata mediante le classi messe a disposizione dal libro di testo [1] (AIPython), in particolare si è utilizzata la classe '*Search_problem_from_explicit_graph*' del modulo '*SearchProblem.py*' per costruire il grafo di ricerca, in cui i nodi sono banalmente gli ambienti dell'edificio, e le classi '*AStarSearcher*' del modulo '*searchGeneric.py*', '*SearcherMPP*' del modulo '*searchMPP.py*'per risolvere i problemi di ricerca associati al grafo.

Per sfruttare al meglio le potenzialità degli algoritmi forniti dal libro si è sviluppata una funzione euristica, discussa in seguito.

# Elenco argomenti di interesse

- Risoluzione di Problemi Mediante Ricerca:
  - *LCFS, A*, Multiple Path Pruning*: utilizzo di una funzione euristica per la stima del percorso più breve basata sulla distanza euclidea; gli algoritmi sono impiegati per il calcolo del percorso migliore tra due punti dell'edificio nelle condizioni esplicitate (autenticazione, permessi, eventuali posti inaccessibili).
- Rappresentazione e Ragionamento Relazionale:
  - *KB in Prolog*: codifica le coordinate dei luoghi di interesse nell'edificio, gli individui che hanno accesso alla struttura, i corsi tenuti e tutte le altre informazioni che consentono di determinare

in base ai vincoli del problema iniziale l'eventuale accesso a un ambiente dell'edificio.

- *Vincoli di Intergrità*: sono state impiegate nella KB delle clausole speciali per codificare situazioni anomale, perciò appena la KB viene caricata viene fatto un check relativo alla soddisfacibilità della stessa.

# Risoluzione di Problemi Mediante Ricerca

## Sommario

Il KBS implementa la ricerca del cammino più economico all'interno del grafo che rappresenta l'edificio attraverso l'algoritmo A* in congiunzione con il Multiple Path Pruning. Nella parte relativa alla valutazione saranno esposte eventuali considerazioni in merito al confronto della nostra strategia con diverse alternative quali Lowest Cost First Search e A* senza MPP.

In particolare sarà evidenziato come, in alcuni casi limite, strategie come A* senza MPP presentano un notevole aumento del tempo di computazione al minimo aumento della distanza tra i due punti scelti.

## Strumenti utilizzati

Gli strumenti utilizzati sono stati i seguenti: libreria AIPython con le relative classi '*Search_problem_from_explicit_graph*' del modulo '*SearchProblem.py*' per costruire il grafo di ricerca, in cui i nodi sono banalmente gli ambienti dell'edificio, e le classi '*AStarSearcher*' del modulo '*searchGeneric.py*', '*SearcherMPP*' del modulo '*searchMPP.py*' per risolvere i problemi di ricerca associati al grafo.

## Decisioni di Progetto

La principale scelta progettuale, oltre alla tipologia di algoritmo utilizzato, è stata la funzione euristica da impiegare. A* è un algoritmo di ricerca informata, che quindi è notevolmente influenzato dalla bontà della medesima funzione.

Per progettare la funzione euristica ci si è focalizzati sul soddisfare due caratteristiche principali: la prima è ovviamente la accettabilità, la seconda è la consistenza.

La prima proprietà garantisce che la valutazione della funzione euristica relativa a un nodo restituisca sempre una *sottostima* del costo del miglior percorso fino al gol, la seconda invece ci garantisce che, per ogni arco, l'euristica del nodo di partenza non sia mai maggiore strettamente del costo dell'arco sommato

all'euristica del nodo di arrivo; in particolare quest'ultima è essenziale per garantire l'ottimalità della soluzione trovata da A* con MPP (che altrimenti non sarebbe garantita).

La funzione euristica sviluppata si basa perciò sulla distanza euclidea: nel caso in cui il nodo di partenza si trovi sullo stesso piano del nodo goal, il valore calcolato corrisponderà proprio la distanza euclidea tra i due punti; nel caso in cui invece i due nodi si trovino su piani diversi, si sceglierà come valore il minimo, per ogni metodo per cambiare piano, della somma della distanza euclidea tra il nodo iniziale e il metodo per cambiare piano, il costo dell'utilizzo del suddetto mezzo e l'euristica del nodo di arrivo calcolata in maniera ricorsiva.

Nel caso in cui ci siano più nodi goal, l'euristica equivarrà al minimo delle singole euristiche calcolate.

Formalmente, sia *n* il nodo di cui calcolare l'euristica, *G* l'insieme dei nodi obiettivo, *d(nodo1,nodo2)* la funzione che calcola la distanza euclidea (come radice quadrata della somma dei quadrati delle differenze delle coordinate), *down(nodo)* (risp. *up(nodo))* funzione che restituisce, dato un metodo per scendere (risp. salire), il relativo metodo che si trova al piano sottostante (risp. sovrastante), *cost(nodo1,nodo2)* funzione che calcola il costo di utilizzo del relativo metodo di discesa (risp. salita), si ha che:

$$h(n) = \min_{g \in G} h(n, g)$$

dove:

$$h(n,g) = \begin{cases} d(n,g) & se\ floor(n) = floor(g) \\ \min_{m \in Down}(d(n,m) + cost(m, down(m)) + h(down(m)) & se\ floor(n) > floor(g) \\ \min_{m \in Up}(d(n,m) + cost(m, up(m)) + h(up(m)) & se\ floor(n) < floor(g) \end{cases}$$

Notiamo che *h(n)* è una funzione euristica che è ammissibile, poiché la distanza euclidea, per disuguaglianza triangolare, è sempre sottostima del costo del percorso migliore nel problema 'vincolato', inoltre il calcolo del minimo ci assicura di star andando a prendere sempre la sottostima del percorso più breve possibile. In aggiunta questa funzione è anche consistente per restrizione

di monotonicità, quindi raggiungendo da *n* un qualsiasi nodo *n'*, l'euristica di *n* non sarà mai maggiore rispetto alla somma di *h(n')+cost(n,n')*, questo sempre per le proprietà della distanza euclidea.

## Valutazione

Il nostro sistema lavora in maniera ottimale se ad *A** viene applicato *Multiple Path Pruning*. Senza quest'ultimo, infatti, *A** è soggetta a notevoli problematiche dovute al fatto che, al crescere della dimensione del percorso ottimo, il numero di percorsi nella frontiera cresce in maniera esponenziale. Inoltre l'efficienza di *A** è molto influenzata dalla presenza di una buona euristica, senza la quale un approccio *Lowest Cost First* risulta notevolmente più dispendioso dal punto di vista computazionale.

Esponiamo ora alcuni esperimenti per una valutazione empirica:

- eseguiamo la ricerca del percorso ottimo (identificandoci come *student_1* che nel nostro sistema non ha accesso a corridoi riservati e agli ascensori) tra gli ambienti *hallway_ingresso* e *hallway_2_27_1.* Usando *A** senza *MPP*, il risultato è il seguente:

```
    A* semplice:

542809 paths have been expanded and 752196 paths remain in the frontier

$ hallway_ingresso --> hallway_0_21_1 --> hallway_0_21_5 --> hallway_0_21_11 --> hallway_0_21_17 --> hallway_0_21_21 --> hallway_0_15_21 --> stairs_0_15_25 --> stairs_1_15_25 --
> hallway_1_15_21 --> hallway_1_9_21 --> hallway_1_3_21 --> hallway_1_1_17 --> hallway_1_1_11 --> stairs_1_5_11 --> stairs_2_5_11 --> hallway_2_1_11 --> hallway_2_1_5 --> hallwa
y_2_3_1 --> hallway_2_9_1 --> hallway_2_11_5 --> hallway_2_11_11 --> hallway_2_15_13 --> hallway_2_19_11 --> hallway_2_19_5 --> hallway_2_21_1 --> hallway_2_27_1
$ Cost = 146.83281572999744
$ Time = 2.919647455215454 sec.
```

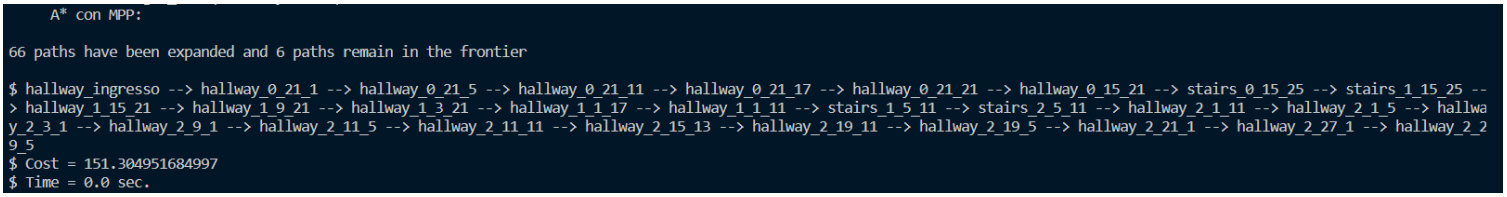
  la ricerca termina in 2.92 secondi, espandendo 542809 percorsi; inoltre al momento del ritrovamento della soluzione, 752196 percorsi si trovano ancora nella frontiera.
  Valutiamo ora la ricerca tra *hallway_ingresso* e *hallway_2_29_5*, che è un ambiente vicino al precedente, perciò comporta uno spostamento minimo:

```
    A* semplice:

1207387 paths have been expanded and 1672021 paths remain in the frontier

$ hallway_ingresso --> hallway_0_21_1 --> hallway_0_21_5 --> hallway_0_21_11 --> hallway_0_21_17 --> hallway_0_21_21 --> hallway_0_15_21 --> stairs_0_15_25 --> stairs_1_15_25 --
> hallway_1_15_21 --> hallway_1_9_21 --> hallway_1_3_21 --> hallway_1_1_17 --> hallway_1_1_11 --> stairs_1_5_11 --> stairs_2_5_11 --> hallway_2_1_11 --> hallway_2_1_5 --> hallwa
y_2_3_1 --> hallway_2_9_1 --> hallway_2_11_5 --> hallway_2_11_11 --> hallway_2_15_13 --> hallway_2_19_11 --> hallway_2_19_5 --> hallway_2_21_1 --> hallway_2_27_1 --> hallway_2_2
9_5
$ Cost = 151.304951684997
$ Time = 7.158999443054199 sec.
```

  la ricerca qui impiega 7.16 secondi espandendo 1207387 percorsi, con 1672021 percorsi ancora nella frontiera al termine dell'algoritmo. Le misure di complessità che stiamo andando a considerare sono quindi più che raddoppiate.

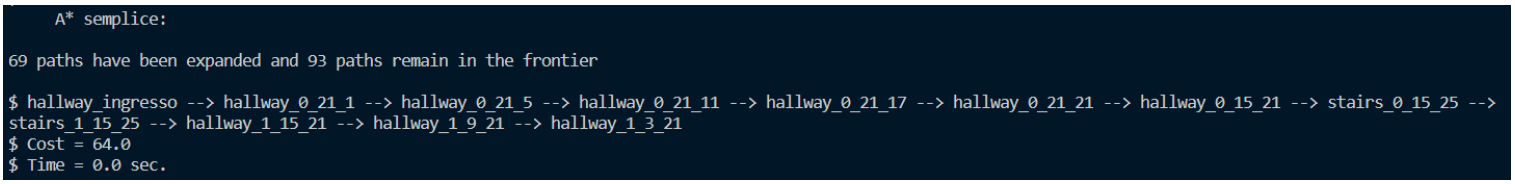
Questo è rappresentativo del fatto che, anche in un problema con un insieme di nodi finito e così piccolo (appena 147 nodi) e un fattore di ramificazione uscente/entrante di 4 vicini per nodo, lo spazio di ricerca cresce in maniera esponenziale nel numero di archi, rendendo infatti poco utile *A* senza MPP* dopo una certa soglia di costo del percorso da rintracciare.

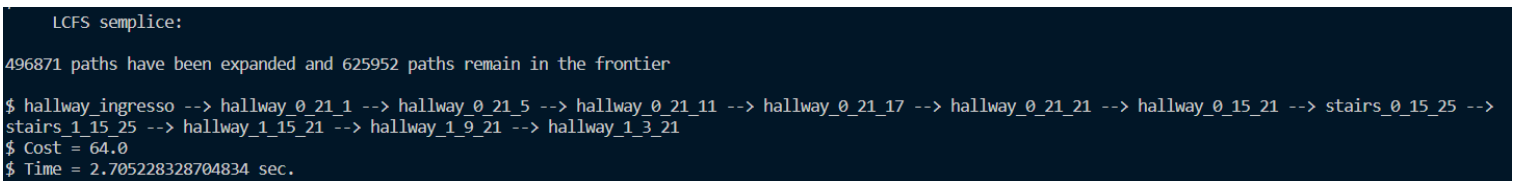
- nel caso dell'ultimo percorso calcolato, confrontiamo inoltre *A** nelle due versioni (con e senza *MPP*):




è evidente come in questo caso i percorsi espansi siano notevolmente minori (66, contro i precedenti 1207387), così come i percorsi rimasti nella frontiera.
Questo avviene perché il *MPP* consente di potare dalla frontiera tutti i percorsi verso nodi già raggiunti (in questo caso la consistenza dell'euristica garantisce che questi non saranno mai meno costosi di quelli già trovati) che quindi non saranno mai ottimali.
Si evidenzia quindi che, come l'euristica, anche l'utilizzo di *MPP* influenza fortemente le prestazioni del sistema.

- nelle medesime condizioni, effettuiamo un confronto tra gli algoritmi *A* senza Multiple Path Pruning* e *Lowest Cost First* (rimuovendo quindi nel secondo caso l'euristica dei nodi), calcolando il percorso tra *hallway_ingresso* e *hallway_1_3_21*:




Si evidenzia che nel primo algoritmo vengono espansi 69 percorsi e al termine dell'algoritmo 93 percorsi rimangono nella frontiera, mentre nel secondo sono 496871 percorsi espansi e 625952 quelli nella frontiera al termine.
Questo denota ancora una volta quanto le strategie di ricerca informate

possano essere considerevolmente più efficienti di quelle non informate e, di conseguenza, quanto sia importante l'implementazione di una buona euristica, la cui stima sia un compromesso tra *efficacia* e *costo della sua computazione*.

In conclusione, gli esperimenti effettuati evidenziano come, delle strategie utilizzate, solo *A* con MPP* sia in linea con le nostre esigenze in quanto gli altri algoritmi risultano inutilizzabili superata una certa soglia di distanza dai nodi goal.

# Rappresentazione e Ragionamento Relazionale

## Sommario

Il KBS effettua ragionamento relazionale allo scopo di recuperare dalla Knowledge Base informazioni essenziali per effettuare le query sul percorso tra due ambienti.

In particolare sono di interesse le informazioni sui permessi dell'utente, sullo stato degli ambienti e le loro posizioni, sugli orari in cui è possibile entrare negli ambienti (confrontati poi con l'orario reale a runtime).

## Strumenti utilizzati

Gli strumenti utilizzati sono stati i seguenti: libreria **pyswip** per l'utilizzo dell'interprete Prolog tramite Python, allo scopo di utilizzare la Knowledge Base scritta in maniera semiautomatica nel medesimo linguaggio logico.

## Decisioni di Progetto

Per manipolare la conoscenza riguardante il dominio abbiamo sviluppato una Knowledge Base seguendo la metodologia standard.

In una fase preliminare, dopo lo sviluppo della planimetria dell'edificio, abbiamo identificato gli individui del mondo reale sui quali ragionare, cioè gli ambienti dell'edificio e gli utenti, e le relazioni utili al ragionamento.

Successivamente abbiamo associato ad ogni individuo una costante che corrisponde nel caso degli ambienti al nome riportato nella planimetria (si veda ***Struttura dell'Edificio***) e nel caso degli utenti a costanti generiche (in un'ottica di reale utilizzo, questi potrebbero essere le matricole di studenti e professori) quali *student_x* e *teacher_x* (dove $x \in \{1,2,3,4\}$).

In seguito abbiamo scelto, per ogni relazione, un simbolo di predicato da utilizzare nella Knowledge Base, di cui si riportano i relativi significati:

- is_student(S) : indica che l'individuo rappresentato da S è uno studente.
- is_teacher(T) : indica che l'individuo rappresentato da S è un insegnante.
- is_person(P) : indica che l'individuo rappresentato da P è una persona.

- teaches_class(T,C) : indica che T insegna la materia C.
- follows_class(S,C) : indica che S segue le lezioni della materia C.
- takes_part_in_class(P,C) : indica che l'individuo P partecipa alle lezioni della materia C.
- is_taking_place(C,R,T) . indica che una lezione della materia C si sta svolgendo della aula di lezioni R al tempo T
- office_owner(T,O) : indica che l'individuo T è proprietario dell'ufficio O.
- has_access(P,R,T) : indica che l'individuo P può accedere alla stanza R al tempo T.
- can_enter_smoke_area(P) : indica che l'individuo P può entrare nella smoke area.
- can_enter_room(P,R,T) : indica che l'individuo P al tempo T può entrare nella stanza R.
- can_go_up_with(P,M) : indica che l'individuo P può utilizzare il metodo M per salire di piano (scale,ascensori).
- can_go_down_with(P,M) : indica che l'individuo P può utilizzare il metodo M per scendere di piano (scale,ascensori).
- can_go_up_with_from(P,M,F) : indica che l'individuo P può utilizzare il metodo M per salire di piano (scale,ascensori), se attualmente si trova nell'ambiente F.
- can_go_down_with(P,M,F) : indica che l'individuo P può utilizzare il metodo M per scendere di piano (scale,ascensori), se attualmente si trova nell'ambiente F.
- get_destination_up(M,D) : indica che il metodo per salire M ha come corrispettivo al piano superiore l'ambiente D.
- get_destination_down(M,D) : indica che il metodo per scendere M ha come corrispettivo al piano inferiore l'ambiente D.
- can_use_elevator(P) : indica che l'individuo P può usare l'ascensore.
- has_elevator_permission(P) : indica che l'individuo P ha il permesso per utilizzare un qualsiasi ascensore.
- has_res_hallway_permission(P) : indica che l'individuo P ha il permesso per attraversare un qualsiasi corridoio riservato.
- can_pass_hallway(P, H) : indica che l'individuo P può attualmente passare attraverso il corridoio H.
- has_permission_to_pass(P,H) : indica che l'individuo P ha il permesso per passare attraverso il corridoio H.
- position(P,X,Y) : indica che il luogo P ha le coordinate (X,Y).

- floor(P,F) : indica che il luogo P è al piano F.
- is_same_floor(P1,P2) : indica che P1 e P2 si trovano sul medesimo piano.
- Is_lower_floor(P1,P2) : indica che P1 si trova sul piano inferiore a P2.
- is_place(P) : indica che l'individuo P è un ambiente.
- is_room(P) : indica che l'individuo P è una stanza.
- is_study_room(R) : indica che il luogo R è una aula studio.
- is_office_room(R) : indica che il luogo R è un ufficio.
- is_bath_room(R) : indica che il luogo R è un bagno.
- is_lesson_room(R) : indica che il luogo R è una aula per lezioni.
- is_smoke_area(R) : indica che il luogo R è una area fumatori.
- is_hallway(R) : indica che il luogo R è un corridoio.
- is_only_with_permission(R) : indica che il corridoio R è accessibile solo con permesso speciale.
- is_elevator(E) : indica che E è un ascensore.
- is_stairs(S) : indica che S è una scala.
- is_elevator_up(E) : indica che E è un ascensore che va a piani superiori.
- is_stairs_up(S) : indica che S è una scala che porta a piani superiori.
- is_elevator_down(E) : indica che E è un ascensore che va a piani inferiori.
- is_stairs_down(S) : indica che S è una scala che porta a piani inferiori.
- is_available_room(R) : indica che la stanza R è attualmente disponibile/accessibile.
- is_available_hallway(H) : indica che il corridoio H è attualmente disponibile/accessibile.
- is_available_smoke_area(A) : indica che la smoke area A è attualmente disponibile/accessibile.
- is_available_elevator(E) : indica che l'ascensore E è attualmente disponibile/accessibile.
- is_available_stairs(S) : indica che le scale S sono attualmente disponibili/accessibili.
- is_unavailable(P) : indica che l'ambiente P non è attualmente accessibile/disponibile.
- is_scheduled(C,R,TS,TE) : indica che la materia C ha una lezione nell'aula R schedulata con inizio TS e fine TE.
- direct_arc(PS,PE) : indica che i due luoghi PS e PE sono collegati tra di loro.

- distance(P1,P2,D) : indica che D è la distanza tra i due ambienti (usata solamente se sono attaccati).
- is_time_included_in(T,TS,TE) : indica che il tempo T è incluso nell'intervallo [TS,TE].
- is_legal_time(T) : indica che il tempo T è scritto in un formato corretto.
- is_legal_day(D) : indica che il giorno D è uno dei giorni della settimana.
- Is_before_time(T1,T2) : indica che T1 precede T2 nello stesso giorno.

Dopodiché è stata fatta l'assiomatizzazione delle clausole della Knowledge Base; queste sono state divise nei file '*fatti.pl*' e '*regole.pl*' (nella directory KB), dove il primo file è stato prodotto mediante script python e assiomatizza tutti i fatti relativi alla planimetria, ai permessi e alle tipologie degli utenti, allo scheduling delle lezioni, mentre il secondo file è stato scritto a mano e fornisce delle regole per dedurre, a partire dai fatti, delle conseguenze logiche della Knowledge Base utili alla ricerca sul grafo.

Si evidenzia che è stato utilizzato un solo simbolo di funzione, per la gestione del concetto di 'tempo', ossia **get_time(D, H, M)** che rappresenta l'individuo "istante di tempo" di giorno D, ora H e minuto M.

Sono state assiomatizzate le seguenti clausole:

*Fatti:*

- direct_arc(P1,P2) ∀ arco nel grafo di ricerca <P1,P2>
- position(P,X,Y) ∀ ambiente P, dove X e Y sono le coordinate.
- floor(P,F) ∀ ambiente P, dove F è il piano dell'ambiente
- is_bath_room(P) ∀ ambiente P che è un bagno
- is_lesson_room(P) ∀ ambiente P che è un'aula lezioni
- is_office_room(P) ∀ ambiente P che è un ufficio
- is_study_room(P) ∀ ambiente P che è un'aula studio
- is_elevator_down(E) ∀ ambiente E che è un ascensore che porta al piano inferiore.
- is_elevator_up(E) ∀ ambiente E che è un ascensore che porta al piano superiore.
- is_stairs_down(S) ∀ ambiente S che è una scala che porta al piano inferiore.
- is_stairs_up(S) ∀ ambiente S che è una scala che porta al piano superiore.
-

- is_smoke_area(S) ∀ ambiente S che è una area fumatori.
- is_hallway(H) ∀ ambiente H che è un corridoio.
- Is_only_with_permission(H) ∀ corridoio H che è riservato, solo con permesso.
- is_student(S) ∀ studente S
- is_teacher(T) ∀ docente T
- office_owner(T,R) ∀ docente T proprietario dell'ufficio R
- follows_class(S,C) ∀ studente S ∀ corso C da lui frequentato
- teaches_class(T,C) ∀ docente T ∀ Corso C da lui insegnato.
- has_elevator_permission(S) ∀ studente S che può usare l'ascensore
- has_res_hallway_permission(S) ∀ studente S che può usare i corridoi riservati.
- is_scheduled(C,R,TS,TE) ∀ lezione programmata nell'aula R per il corso C con orario iniziale TS e orario finale TE.

*Regole:*

*Utilizzo degli ascensori:*

- can_use_elevator(Person) :- is_teacher(Person).
- can_use_elevator(Person) :- is_student(Person), has_elevator_permission(Person).

*Salita/Discesa e collegamenti tra scale/ascensori*

- can_go_up_with(Person, Method) :- is_elevator_up(Method), can_use_elevator(Person), is_available_elevator(Method).
- can_go_up_with(Person, Method) :- is_stairs_up(Method), is_person(Person), is_available_stairs(Method).
- can_go_down_with(Person, Method) :- is_elevator_down(Method), can_use_elevator(Person), is_available_elevator(Method).
- can_go_down_with(Person, Method) :- is_stairs_down(Method), is_person(Person), is_available_stairs(Method).
- can_go_up_with_from(Person, Method,From) :- can_go_up_with(Person,Method), is_same_floor(Method,From).
- can_go_down_with_from(Person,Method,From) :- can_go_down_with(Person,Method), is_same_floor(Method,From).

- get_destination_up(Method, Destination) :- position(Method, X, Y), position(Destination, X, Y), floor(Method, Floor1), floor(Destination, Floor2), Floor1 is Floor2 - 1.
- get_destination_down(Method,Destination):- position(Method, X, Y), position(Destination, X, Y), floor(Method, Floor1), floor(Destination, Floor2), Floor1 is Floor2 + 1.

*Accesso a tutti i luoghi:*

- has_access(Person, Hallway, _) :- can_pass_hallway(Person, Hallway).
- has_access(Person, Room, Time) :- can_enter_room(Person, Room, Time).
- has_access(Person, Method, _) :- can_go_up_with(Person, Method).
- has_access(Person, Method, _) :- can_go_down_with(Person, Method).
- has_access(Person, Smoke_Area, _) :- can_enter_smoke_area(Person, Smoke_Area).

*Utilizzo dei corridoi:*

- can_pass_hallway(Person,Hallway) :- is_available_hallway(Hallway), has_permission_to_pass(Person,Hallway).
- has_permission_to_pass(Person, Hallway) :- is_hallway(Hallway), \+is_only_with_permission(Hallway), is_person(Person).
- has_permission_to_pass(Person, Hallway) :- is_hallway(Hallway), is_only_with_permission(Hallway), is_teacher(Person).
- has_permission_to_pass(Person, Hallway) :- is_hallway(Hallway), is_only_with_permission(Hallway), is_student(Person), has_res_hallway_permission(Person).

*Ingresso nelle stanze:*

- can_enter_room(Person, Room, _) :- is_available_room(Room), is_office_room(Room), office_owner(Person, Room).
- can_enter_room(Person, Room, _) :- is_available_room(Room), is_office_room(Room), follows_class(Person, Class), teaches_class(Teacher, Class), office_owner(Teacher, Room).
- can_enter_room(Person, Room, Time) :- is_available_room(Room), is_study_room(Room), is_student(Person) is_legal_time(Time).

- can_enter_room(Person, Room, Time) :- is_available_room(Room), is_bath_room(Room), is_legal_time(Time), is_person(Person).
- can_enter_room(Person, Class_Room, Time) :- is_available_room(Class_Room), is_lesson_room(Class_Room), takes_part_in_class(Person, Class), is_taking_place(Class, Class_Room, Time).

*Ingresso nelle aree fumatori:*

- can_enter_smoke_area(Person, Smoke_Area) :- is_available_smoke_area(Smoke_Area), is_person(Person).

*Disponibilità dei Luoghi:*

- is_available_room(Room) :- is_room(Room), \+is_unavailable(Room).
- is_available_hallway(Hallway) :- is_hallway(Hallway), \+is_unavailable(Hallway).
- is_available_smoke_area(Smoke_Area) :- is_smoke_area(Smoke_Area), \+is_unavailable(Smoke_Area).
- is_available_elevator(Elev) :- is_elevator(Elev), \+is_unavailable(Elev).
- is_available_stairs(Stairs) :- is_stairs(Stairs), \+is_unavailable(Stairs).
- is_unavailable(Place) :- there_is_a_problem_in(Place).
- is_unavailable(Place) :- has_wet_floor(Place).

*Partecipazione alle lezioni:*

- takes_part_in_class(Person, Class) :- follows_class(Person, Class).
- takes_part_in_class(Person, Class) :- teaches_class(Person, Class).

*Attuale Svolgimento delle Lezioni:*

- is_taking_place(Class, Class_Room, Time) :- is_scheduled(Class, Class_Room, Start_Time, End_Time), is_time_included_in(Time, Start_Time, End_Time).

*Gestione del concetto di Tempo:*

- is_time_included_in(Time, Start_Time, End_Time) :- is_before_time(Time, End_Time), is_before_time(Start_Time, Time), is_legal_time(Time).
- is_legal_time(get_time(Day, Hour, Minute)) :- Minute =< 59, Minute >= 0, Hour =< 23, Hour >= 0,  is_legal_day(Day).
- is_legal_day(monday).
- is_legal_day(tuesday).
- is_legal_day(wednesday).
- is_legal_day(thursday).
- is_legal_day(friday).
- is_legal_day(saturday).
- is_legal_day(sunday).
- is_before_time(get_time(Day, Hour1, _), get_time(Day, Hour2, _)) :- Hour1 < Hour2.
- is_before_time(get_time(Day, Hour, Minute1), get_time(Day, Hour, Minute2)) :- Minute1 =< Minute2.

*Discriminazione dei luoghi:*

- is_place(Place) :- is_room(Place).
- is_place(Place) :- is_elevator(Place).
- is_place(Place) :- is_stairs(Place).
- is_place(Place) :- is_hallway(Place).
- is_place(SmokeArea) :- is_smoke_area(SmokeArea).
- is_room(Room) :- is_bath_room(Room).
- is_room(Room) :- is_lesson_room(Room).
- is_room(Room) :- is_study_room(Room).
- is_room(Room) :- is_office_room(Room).
- is_elevator(Method) :- is_elevator_down(Method).
- is_elevator(Method) :- is_elevator_up(Method).
- is_stairs(Method) :- is_stairs_down(Method).
- is_stairs(Method) :- is_stairs_up(Method).

*Discriminazione delle persone:*

- is_person(Person) :- is_student(Person).
- is_person(Person) :- is_teacher(Person).

*Confronto sui Piani:*

- is_same_floor(Place1, Place2) :- floor(Place1, Floor), floor(Place2, Floor).
- is_lower_floor(Place1, Place2) :- floor(Place1,Floor1), floor(Place2,Floor2), Floor1 < Floor2.

*Calcolo delle distanze per casi:*

- distance(Place1, Place2, Distance) :- position(Place1, X1, Y1), position(Place2, X2, Y2), floor(Place1, Floor),floor(Place2, Floor), X is X2 - X1, Y is Y2 - Y1, Distance is sqrt(X * X + Y * Y).
- distance(Place1, Place2, Distance) :- position(Place1, X, Y), position(Place2, X, Y), floor(Place1, Floor1), floor(Place2, Floor2), Floor1 \= Floor2, is_elevator(Place1), is_elevator(Place2), Distance is 6
- distance(Place1, Place2, Distance) :- position(Place1, X, Y), position(Place2, X, Y), floor(Place1, Floor1), floor(Place2, Floor2), Floor1 \= Floor2, is_stairs(Place1), is_stairs(Place2), Distance is 12.

Sono stati inoltre utilizzati dei Vincoli di Integrità, che sono implementati mediante il simbolo speciale di predicato 0-ario false. All'avvio del programma viene caricata la Knowledge Base tramite la classe *KnowledgeBase* implementata nel file '*KnowledgeBase.py*', che effettua successivamente il controllo sulla soddisfacibilità della stessa base di conoscenza mediante la query 'ask false'.

Non riportiamo nella documentazione la lista completa dei vincoli di integrità, che è comunque presente in coda al file '*regole.py*', ma si discutono brevemente le casistiche ricoperte:

- Aule occupate contemporaneamente da più di una lezione.
- Schedulazioni con orari non validi o inconsistenti tra di loro (es. ora di inizio successiva all'ora di fine, oppure ora di inizio e di fine contraddistinte da giorni differenti)
- Persone che sono sia studenti che docenti.

- Ambienti che sono di due tipi diversi (poiché i tipi sono mutui-esclusivi).
- Lezioni schedulate in luoghi che non sono aule da lezione.
- Persone che seguono lezioni senza essere studenti.
- Persone che insegnano lezioni ma senza essere docenti.
- Ambienti che possiedono più coordinate diverse.
- Ambienti diversi che hanno le stesse coordinate.
- Altre situazioni varie ed eventuali.

## Utilizzo della Knowledge Base mediante Query

Rappresentare la conoscenza usando Prolog ci ha permesso di poter effettuare semplici query a runtime per strutturare lo specifico grafo di ricerca in base alla situazione reale dell'edificio e ai permessi utente.

All'avvio del programma viene chiesto all'utente di inserire un *id_utente* per identificarsi. La prima query che viene fatta alla Knowledge Base è quindi **is_person(id_utente)**, per capire se questo è realmente un nome utente valido.

Successivamente viene richiesto un luogo di partenza e uno di arrivo per effettuare la ricerca, perciò alla Knowledge Base vengono sottoposte le query **is_place(place_requested)** dove *place_requested* è appunto sostituito con i luoghi immessi dall'utente, per verificare che siano luoghi validi codificati.

Nel caso in cui l'utente scelga una delle query di default, '*/bath*' oppure '*/study*', il sistema effettua la query **is_bath_room(Bath)** oppure **is_study_room(Study)** ottenendo tutti i luoghi desiderati nelle due casistiche, da passare poi al costruttore del grafo di ricerca.

Ottenute tutte le info necessarie, si procede a costruire il grafo mediante la classe '*My_Problem*': viene effettuata una chiamata all'OS per ottenere l'orario attuale di richiesta '*current_time'*, da utilizzare successivamente per controllare l'accesso ai luoghi.

Con la query **has_access(id_utente, place, current_time)** viene verificato che l'utente abbia accesso alla stanza di partenza e alle stanze di arrivo della richiesta, per stampare eventualmente dei messaggi di warning.

In seguito, la query **is_place(X)** recupera tutti gli ambienti codificati, che verranno manipolati tramite la classe '*My_Node*'; questa classe effettua le query **floor(place, Floor)** e **position(place, X, Y)** per ricavare le coordinate del

nostro nodo. Inoltre viene richiesto direct_arc(place, Neighbor) per memorizzare una lista dei vicini del nodo nel grafo di ricerca. Nella stessa classe sono implementati anche i metodi necessari al calcolo dell'euristica (si veda la sezione precedente dedicata all'analisi dell'euristica) che utilizzano le query is_same_floor(start, goal), is_lower_floor(start,goal), distance(start,goal,Distance), can_go_up_with_from(id_utente, Method, start), can_go_down_with_from(id_utente, Method, start), get_destination_up(method, D) e get_destination_down(method, D).

Nel grafo di ricerca viene inserito per ogni nodo un arco verso i suoi vicini (calcolati precedentemente), ma solo se la query has_access(is_utente, place, current_time) ha esito positivo per entrambi i nodi (start e end), altrimenti lo specifico arco non viene aggiunto perché lo specifico utente non ha la possibilità di percorrerlo. Il costo del singolo arco viene dato dalla query distance(start, end, Cost) che restituisce la distanza euclidea per posti sullo stesso piano, 6 per ascensori collegati, 12 per scale collegate (nulla in altri casi, situazione di errore).

Il grafo è quindi pronto per essere utilizzato dalla relativa classe solver, che ricercherà il percorso migliore con A*+MPP.

# Conclusioni

I requisiti iniziali preposti sono stati raggiunti con successo e il programma rispetta quanto ci si era prefissati di ottenere.

Tuttavia, a posteriori, ci siamo resi conto che sarebbe stato utile strutturare la Base di Conoscenza utilizzando lo standard suggerito dal Capitolo 14 di [1], mediante una rappresentazione caratterizzata da triple "*Individuo Proprietà Valore*". Durante la fase di progettazione ci si è resi conto di non aver strettamente bisogno di implementare tale rappresentazione; tuttavia, potrebbe essere necessaria per agevolare l'utilizzo di query riguardanti la classe di appartenenza di uno specifico individuo, che potrebbero risultare utili per eventuali sviluppi futuri.

## Possibili estensioni

Una estensione possibile, attualmente non implementata per motivi di tempo, è un'euristica perfetta cost_to_goal(n) mediante l'utilizzo di Dynamic Programming, per le query di default '/bath' e '/study' (che ricercano il bagno più vicino e l'aula studio più vicina). L'euristica menzionata è infatti molto utile nei casi in cui i nodi goal sono fissi, per cui si calcola per ogni nodo del grafo la lunghezza minima di una soluzione che parte dal nodo stesso. In particolare, nel nostro caso, entrambi i requisiti per la costruzione di cost_to_goal(n) sono soddisfatti, poiché per effettuare una ricerca in direzione opposta (a partire dai nodi goal) i nodi goal devono essere di numero finito e si deve avere a disposizione il grafo inverso (che nel nostro caso è proprio quello di partenza, avendo per ogni nodo <n1,n2> anche il nodo <n2,n1>).

Un'altra idea potrebbe essere quella di rendere indisponibili determinate stanze ad una certa ora se occorre un problema di 'illuminazione guasta' oppure consentire l'autenticazione come *GUEST* avendo accesso solo ad alcune aule (es. bagni e smoke area).

# Riferimenti Bibliografici

[1] D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press