

Documentazione del progetto di ICon di:

**Manzari Antonio,
Melhab Emanuele,
Menga Domenico Francesco**

Indice

- Introduzione
- Requisiti specifici
 - Requisiti funzionali
 - Requisiti non funzionali
- System Design
- OO Design
- Struttura dei moduli principali
 - Belief Network
 - Ricerca su grafo
 - KB
- File e funzioni ausiliarie
- Lista delle librerie esterne usate
- Struttura della directory IconPJ

Introduzione

Con la presente relazione si intende evidenziare la realizzazione di un software il quale cerca di **consigliare** ad utenti posti nei quali svolgere le proprie vacanze, e successivamente **aiutarli** nella ricerca dei tragitti più brevi tra due punti sulla mappa , e di **supportali** attraverso una KB.

Il dominio scelto è quello relativo ai viaggi.

Abbiamo scelto dieci capitali, di queste dieci Capitali abbiamo considerato cinque-sette luoghi chiave per ogni città da visitare assolutamente.

Ogni luogo è stato etichettato , per esempio, gli **stadi** sono stati etichettati come **Sport**, oppure i **musei** come **Musei**.

Sulla base di queste categorie e sulla base dei gusti dell'utente che usa il sistema, una **Belief Network** dopo che l'utente ha risposto ad un questionario fornisce una probabilità di affinità per ogni città .

Di questi posti abbiamo poi visto **Costi, Orari, Giorni di apertura** ecc...

Successivamente abbiamo modellato le varie stagioni per ogni emisfero e per ogni continente.

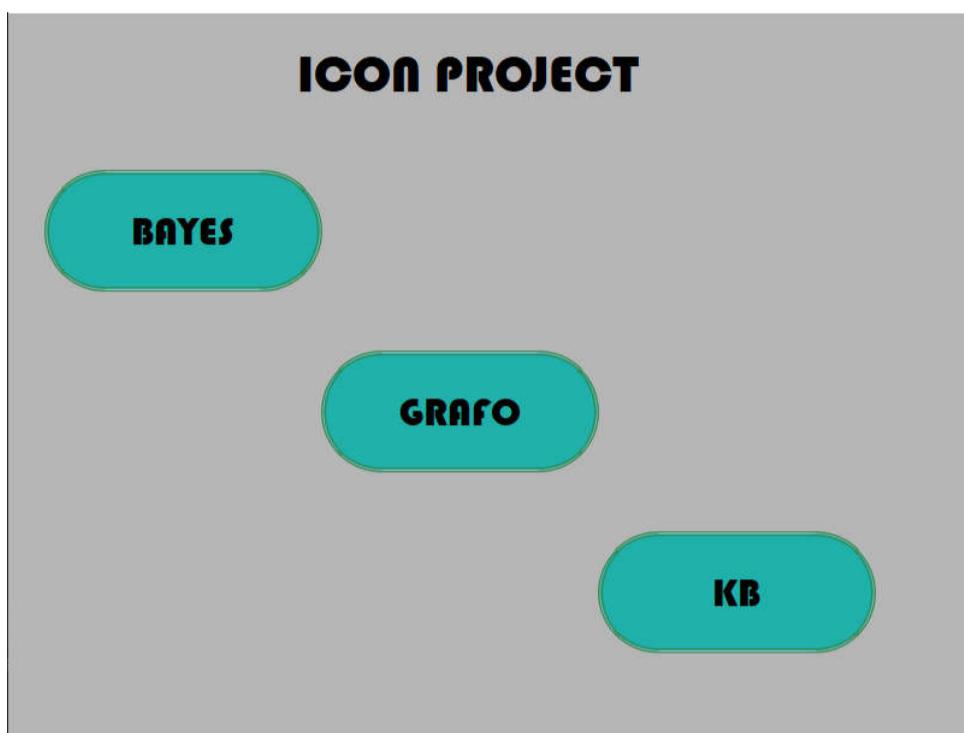
Tutta questa conoscenza è stata modellata nella KB, affinché l'utente possa avere un supporto durante il viaggio.

Oltre alla KB come supporto subentra anche la ricerca su grafo, per ricercare il percorso più breve tra due posti sulla mappa.

Requisiti specifici

- Mostrare il menù iniziale

Non appena il software viene avviato deve subito mostrare all' utente un menù iniziale composto da tre **Push Button**, ovvero:



- Pressione del pulsante Bayes

Alla pressione del pulsante Bayes il sistema dovrà mostrare la finestra contenente il questionario che l'utente dovrà compilare se vorrà un feedback sulle città più affini ai suoi gusti.

[INDIETRO](#)

SELEZIONA LE TUE PREFERENZE PER SCOPRIRE LA TUA CITTA' IDEALE

TI PIACE LA SCIENZA?	<input type="radio"/> SI	<input type="radio"/> NO	<input type="radio"/> NE' SI NE' NO
TI PIACE RILASSARTI?	<input type="radio"/> SI	<input type="radio"/> NO	<input type="radio"/> NE' SI NE' NO
TI PIACE ANDARE ALL'AVVENTURA?	<input type="radio"/> SI	<input type="radio"/> NO	<input type="radio"/> NE' SI NE' NO
TI PIACE LA CULTURA?	<input type="radio"/> SI	<input type="radio"/> NO	<input type="radio"/> NE' SI NE' NO
TI PIACE LA RELIGIONE?	<input type="radio"/> SI	<input type="radio"/> NO	<input type="radio"/> NE' SI NE' NO
TI PIACCIONO I LUOGHI SACRI?			

[OK](#)

- Pressione del pulsante OK

Quando l'utente avrà risposto a tutte le domande potrà premere il pulsante OK affinché il sistema possa effettuare l'inferenza probabilistica necessaria, per ottenere tutte le probabilità.

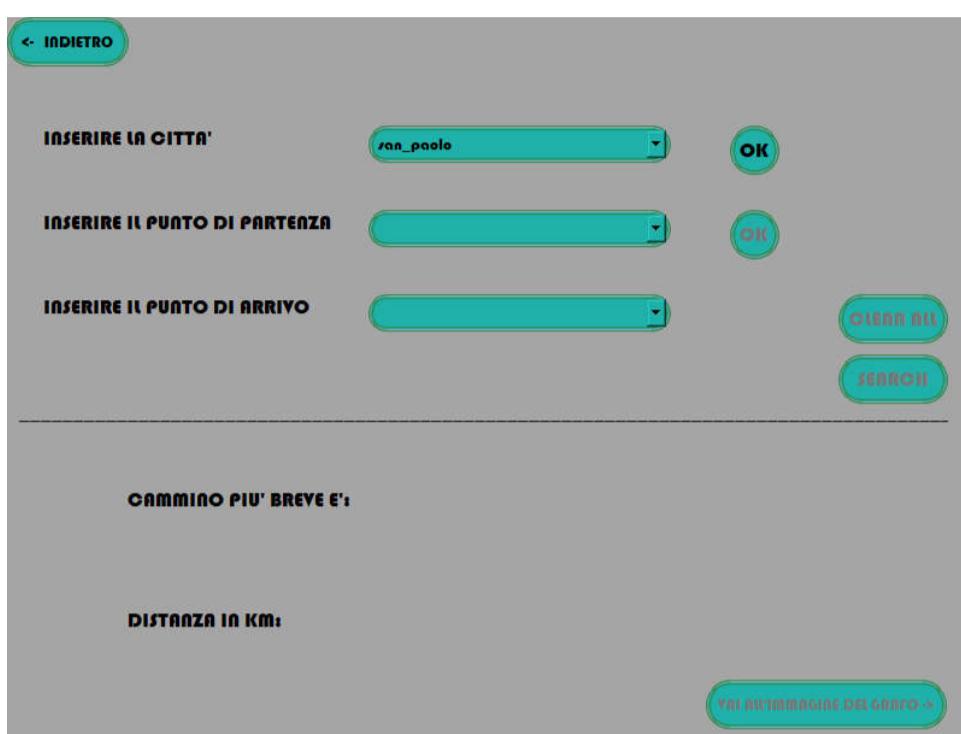
OK

- Parigi > 0.69
- Istanbul > 0.7
- New York > 0.7
- Mosca > 0.7
- Madrid > 0.75

[INDIETRO](#)

- **Pressione del pulsante GRAFO da menù iniziale**

Alla pressione di tale pulsante il sistema mostra all' utente la finestra dalla quale si potrà selezionare **città, posto di partenza e posto di arrivo.**



- **Pressione del tasto SEARCH dalla finestra precedente**

Non appena l'utente avrà selezionato, città, punto di partenza , punto di arrivo, il sistema potrà calcolare il percorso più breve tra questi due punti , sulla mappa della città selezionata. Una volta che l'algoritmo A* avrà terminato verrà mostrato il percorso e la relativa distanza. Sarà anche possibile visualizzare una cartina della mappa pigiando il pulsante **vai all'immagine del grafo.**

<- INDIETRO

INSERIRE LA CITTA'	<input type="text" value="san_paolo"/>	OK
INSERIRE IL PUNTO DI PARTENZA	<input type="text" value="monumento_independenza"/>	OK
INSERIRE IL PUNTO DI ARRIVO	<input type="text" value="jaraqua_state_park"/>	CLEAN ALL
		SEARCH

CAMMINO PIU' BREVE E': [17. 29]

DISTANZA IN KM: 3.18 KM

VARI ALL'IMMAGINE DEL GRAFO >

- **Pressione del pulsante KB da menù iniziale**

Quando l'utente avrà pigiato il pulsante **kb** da menù iniziale il sistema dovrà mostrargli la finestra dalla quale successivamente si potrà selezionare la query da porre al sistema.

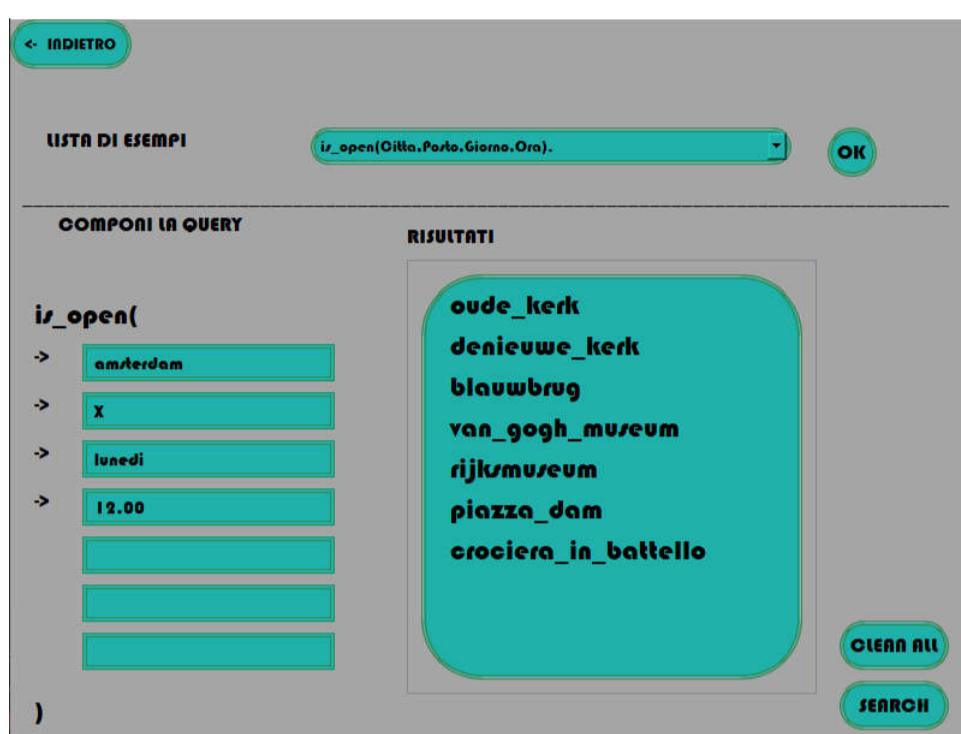
<- INDIETRO

LISTA DI ESEMPI	<input type="text" value="is_open(Citta,Porto,Giorno,Ora)."/>	OK
------------------------	---	-----------

COMPONI LA QUERY	RISULTATI
<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	<div style="border: 1px solid black; width: 150px; height: 150px; margin: 10px;"></div>
CLEAN ALL SERARCH	

- Pressione del pulsante SEARCH dalla finestra kb

Quando l'utente avrà scelto la **query** da porre al sistema, e avrà inserito i termini e/o variabili potrà piggare il tasto **search**, il quale mostrerà nel riquadro “**risultati**” , i risultati derivanti dalla kb rispetto alla query posta.



System Design

Lo stile architetturale scelto per questo progetto è il Model – View – Controller, in quanto permette di separare la logica di presentazione dei dati, dalla logica di base dei tre moduli, ovvero, KB, Grafo e Belief network.

OO Design

Per quanto concerne questo progetto, abbiamo la seguente lista di classi:

- MainWindow
- GraphFW
- BayesFW
- BatesSW
- KBFW

MainWindow:

Questa classe gestisce il menù principale.

Il costruttore della classe per prima cosa carica il file **mainPage.ui**, file in XML che descrive l'interfaccia della schermata iniziale.

Successivamente il costruttore chiama i file CSS contenenti la grafica da applicare al file XML.

La classe MainWindow possiede anche tre metodi, i quali se chiamati vanno a richiamare altre classi:

I metodi sono:

- goGraphInterface()
- GoBayesInterface()
- GoKBInterface()

GraphFW:

Questa classe gestisce l'interfaccia relativa alla sezione “Ricerca su grafo”, e la relativa logica.

Il costruttore di tale classe, apre il file **GraphInterface.ui**, file scritto in XML che descrive l'interfaccia. Successivamente vengono aperti nove file CSS contenenti lo style dell'interfaccia.

La classe GraphFW possiede sei metodi, di seguito descritti:

- Back() : metodo che consente di tornare alla pagina iniziale del software
- Ok() : inibisce la prima ComboBox e abilita la seconda , caricando dalla KB tutti i posti presenti nella città scelta
- Ok_2() : inibisce la seconda combo box , attivando la terza e caricando dalla KB tutti i posti presenti nella città scelta.
- Search() : questo metodo richiama la funzione SearchPath(partenza,destinazione,città) presente nel file **Graph.py**, ritornando il percorso più breve tra i due punti nella città scelta.
- CleanAll() : questo metodo riporta l'interfaccia della sezione “ricerca su grafo” alle condizioni iniziali, quindi eliminando tutti i valori precedentemente selezionati nelle comboBox.
- GoGraphImage() : questo metodo consente di visualizzare un immagine png della città selezionata, tale mappa possiede la numerazione dei nodi e i pesi su ogni arco.

BayesFW:

La classe BayesFW consente di visualizzare il questionario al quale l'utente dovrà rispondere se vuole sapere quale città è più affine ai suoi gusti.

Il costruttore di tale classe, apre il file **Bayes1.ui**, file scritto in XML che descrive l'interfaccia. Successivamente vengono aperti due file CSS contenenti lo style dell'interfaccia.

I metodi che questa classe possiede sono:

- Clickok() : tale metodo una volta cliccato il pulsante **OK** acquisisce tutte le risposte date dall'utente e richiama i metodi presenti sul file **Bayes.py** (un metodo per ogni città). Questi metodi ritornano le probabilità di affinità con le città. In fine il metodo stampa in una seconda istanziando la classe **BayesSW** finestra le varie probabilità.
- Clickclose() : questo metodo consente di tornare alla schermata del menù iniziale.

BayesSW:

Questa classe carica una finestra nella quale vengono mostrati i risultati provenienti dal metodo **infer.query(evidenze)** richiamato sulla rete bayesiana.

Il costruttore di tale classe, apre il file **Bayes2.ui**, file scritto in XML che descrive l'interfaccia. Successivamente vengono aperti due file CSS contenenti lo style dell'interfaccia.

I metodi della classe sono:

- Clickclose() : chiude la finestra
- Display() : stampa i risultati nella textArea

KBFW:

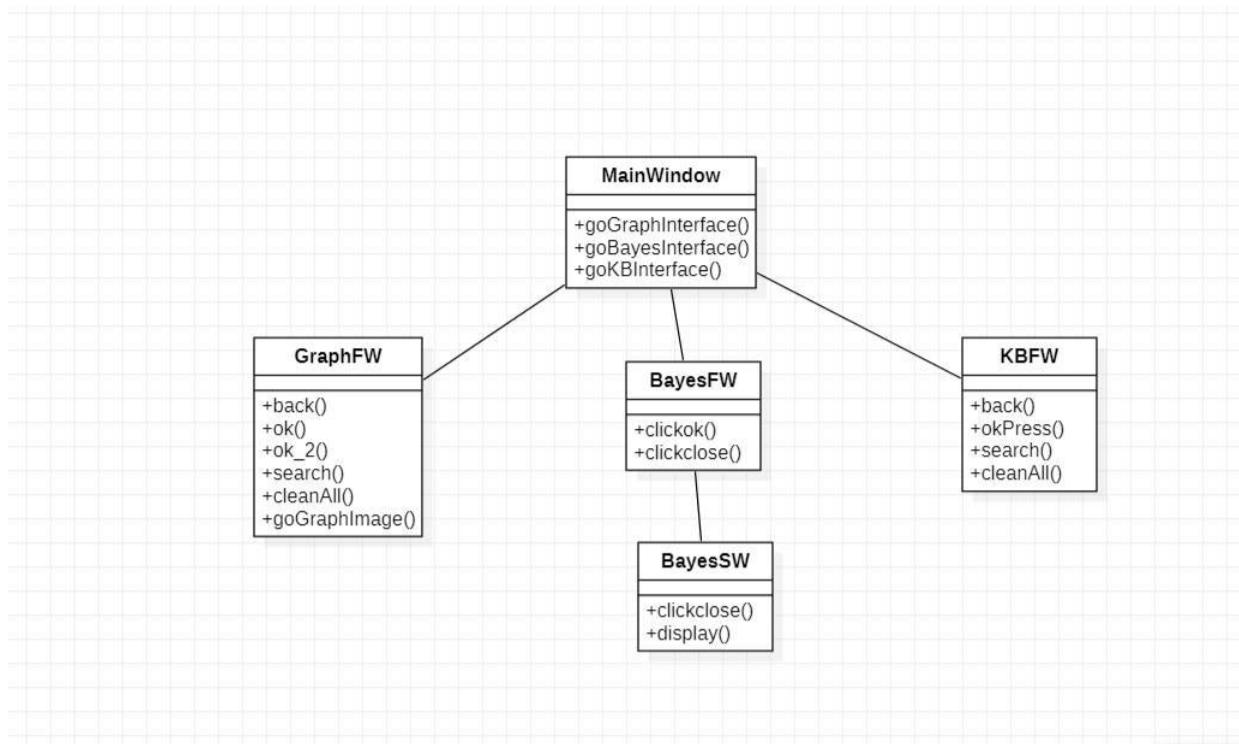
La classe KbFW consente l'interazione tra utente e KB , permettendogli la selezione di query da poter porre al sistema e permettendogli di inserire termini e variabili.

Il costruttore di tale classe, apre il file **KBInterface.ui**, file scritto in XML che descrive l'interfaccia. Successivamente vengono aperti dodici file CSS contenenti lo style dell'interfaccia.

I metodi contenuti in questa classe sono:

- Back() : consente di tornare al menù principale.
- OkPress() : sulla base della query dall' utente selezionata, rende editabili le textArea nelle quali andranno inseriti termini e/o variabili.
- Search() : acquisisce la query scritta dall'utente e interroga la KB
- CleanAll() : questo metodo riporta l'interfaccia della sezione "ricerca su grafo" alle condizioni iniziali, quindi eliminando tutti i valori precedentemente selezionati nelle comboBox e nelle textArea.

Diagramma delle classi:



Struttura dei moduli principali

Belief network

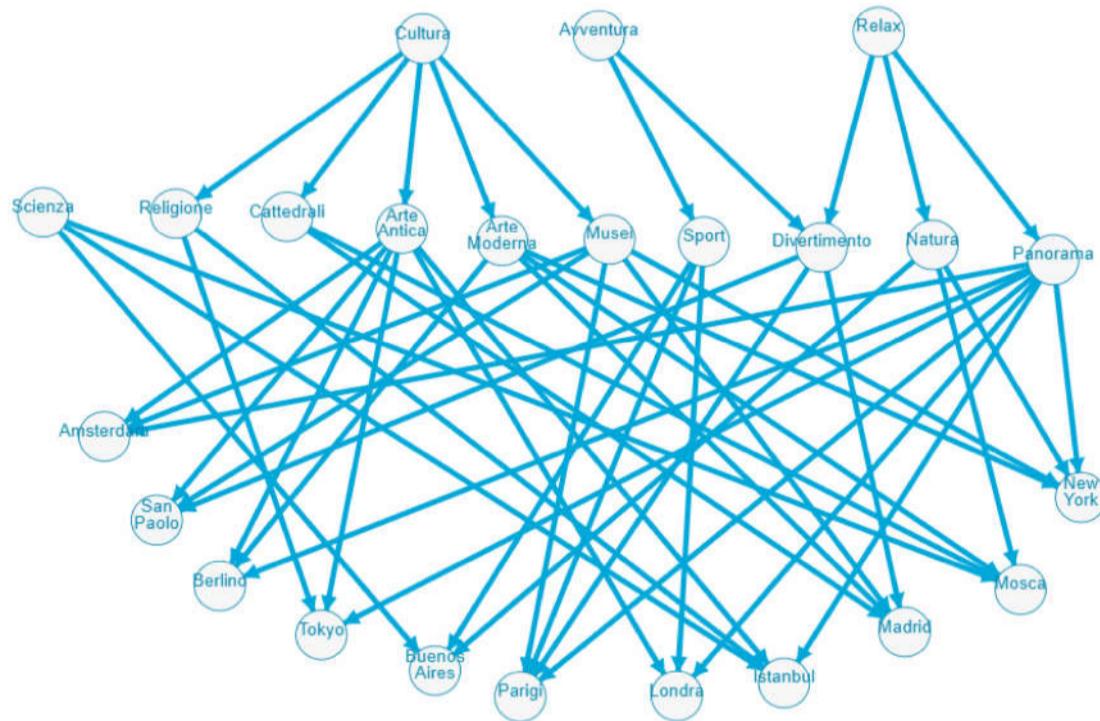
La rete bayesiana da noi realizzata è stata usata per poter dare una stima dell'affinità che un utente possiede con le città da noi modellate nel software, tale affinità si basa sulle categorie di posti che ogni città possiede.

Ad esempio Amsterdam al momento possiede nel nostro progetto otto posti visitabili, dei quali:

- 4 etichettati come **Arte antica**
- 2 etichettati come **Panorama**
- 2 etichettati come **Musei**

Sulla base delle frequenze di ogni categoria per ogni città abbiamo costruito le CPT della rete. Ad esempio per Amsterdam abbiamo considerato la città influenzata da tre variabili, ovvero Arte antica, Musei, Panorama ogni variabile.

Di seguito viene riportato il grafo rappresentante la rete pocanzi descritta.



Come si evince dal grafo la rete è composta di tre livelli , il primo di tre variabili, il secondo di dieci e l'ultimo, il terzo livello con undici variabili, ovvero tutte le città presenti nel progetto.

Le variabili sul secondo livello sono influenzate da quelle al livello precedente, ovvero il primo, così come le variabili sul terzo livello vengono influenzate direttamente solo da quelle al secondo livello.

N.B. è disponibile anche il file **.JSON** della rete.

Ricerca su grafo

Nel nostro progetto abbiamo impiegato la ricerca su grafo per ricercare su di una mappa fittizia(non reale della città) la distanza più breve tra due punti.

L'utente selezionando punto di partenza, punto di arrivo e città , otterrà un cammino composto da una lista di nodi, che sono i vari incroci e posti da visitare nella mappa relativa alla città.

Nel progetto in tutto ci sono undici file testuali contenenti i grafi, da noi creati con il supporto di una libreria esterna.

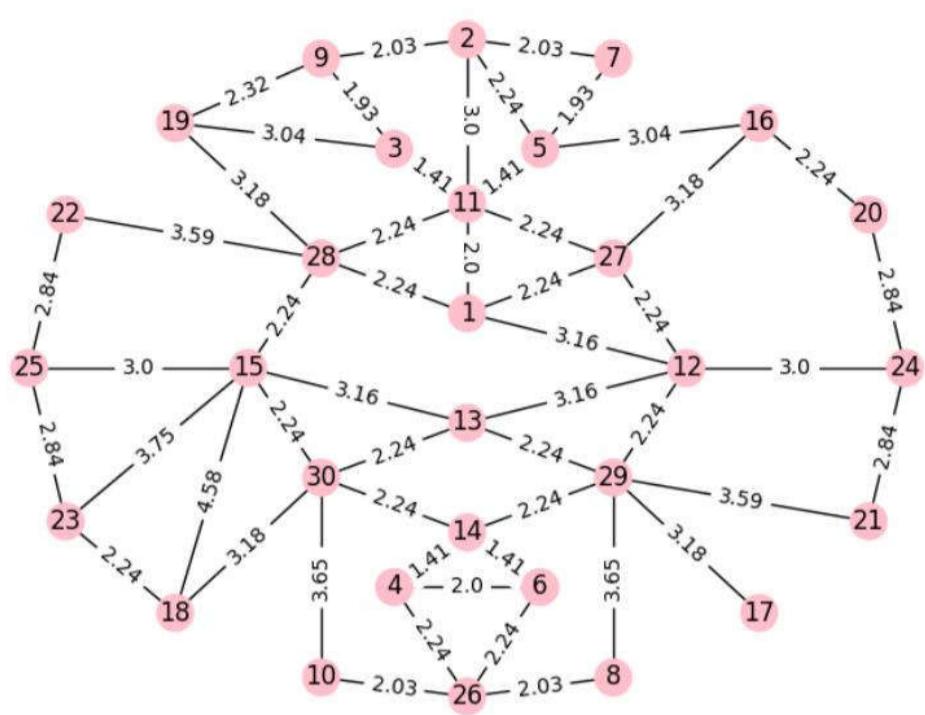
Per la ricerca su grafo ci siamo avvalsi dell' algoritmo di ricerca informata **A***, già implementato nella libreria, inoltre abbiamo usato un euristica da noi definita.

L'euristica stima la distanza euclidea tra il nodo attualmente espanso e il nodo goal in linea d'aria.

Di seguito viene riportato il codice della funzione euristica:

```
def euclideanDistance(source, goal): #funzione che calcola la distanza euclidea tra due punti
    x1, y1 = pos.get(source)
    x2, y2 = pos.get(goal)
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
```

Di seguito viene riportata la mappa in formato png di Amsterdam



KB

La KB da noi costruita cerca di modellare conoscenza da noi acquisita sul web e conoscenza pregressa relativa al dominio dei viaggi e della geografia.

In particolare abbiamo inserito una serie di fatti riguardanti:

- Città
- Stagioni
- Valute
- Porti visitabili
- Stazioni

Ora viene di seguito mostrato in dettaglio le regole da noi create.

- E_nel_continente(X,Y) : modella il fatto che una città **X** sia nel continente **Y**

esempio da KB: **e_nel_continente(san_paolo,america_sud).**

- Boreale(A) : modella il fatto che un continente **A** sia boreale

esempio da KB: **boreale(europa).**

- Austral(A) : modella il fatto che un continente **A** sia australe

Esempio da KB: **australe(africa_meridionale).**

- Mese_estivo_boreale(M) : modella il fatto che un mese **M** sia estivo nell' emisfero boreale

Esempio da KB: **mese_estivo_boreale(june).**

- Mese_autunnale_boreale(M) : modella il fatto che un mese **M** sia autunnale nell' emisfero boreale

Esempio da KB: **mese_primaverile_boreale(march).**

- Mese_primaverile_boreale(M) : modella il fatto che un mese **M** sia primaverile nell' emisfero boreale

Esempio da KB: **mese_autunnale_boreale(september).**

- Mese_invernale_boreale(M) : modella il fatto che un mese **M** sia invernale nell' emisfero boreale

Esempio da KB: **mese_invernale_boreale(december).**

- Mese_estivo_australe(M) : modella il fatto che un mese **M** sia estivo nell' emisfero austral

Esempio da KB: **mese_estivo_australe(december).**

- Mese_autunnale_australe(M) : modella il fatto che un mese **M** sia autunnale nell' emisfero australe

Esempio da KB: **mese_primaverile_australe(september)**

- Mese_primaverile_australe(M) : modella il fatto che un mese **M** sia primaverile nell' emisfero australe

Esempio da KB: **mese_autunnale_australe(march).**

- Mese_invernale_australe(M) : modella il fatto che un mese **M** sia invernale nell' emisfero australe

Esempio da KB: **mese_invernale_australe(june).**

- Station(C,N,No) : modella il fatto che in una città **C** c'è una stazione **N** ed è posizionata sul grafo nel nodo **No**

Esempio da KB: `station(san_paolo, estacao_agua_branca,3).`

- City(C,Co,N,F): modella il fatto che una città **C** sia in un continente **Co**, in una nazione **N** e ha fuso orario **F**.

Esempio da KB: `city(san_paolo , america_sud, brasile, gmt-3).`

- Currency(C,V): modella il fatto che la città **C** ha valuta **V**

Esempio da KB: `currency(amsterdam , eur).`

- Showplace(C,P,Cat,G,O,P,No) : modella i posti da visitare, ovvero in una città **C** ho un posto **P** etichettato come **Cat** che apre nei giorni **G** ad ora **O** con prezzo per la visita **P**, inoltre il posto è posizionato sul grafo nel nodo **No**.

Esempio da KB: `showplace(amsterdam , palazzo_reale , arte_antica , ven-dom , 10:00-17:00 , 7.50 ,1).`

Di seguito viene riportata una lista riguardante le regole inserite nella KB:

- La regola che segue viene usata per capire se nella Città City nel Mese indicato e nel giorno indicato è estate oppure no, nella KB abbiamo due regole come questa la differenza sta che in una usiamo l'atomo estateBoreale() e nella seconda estateAustrale()

N.B. ci sono regole simili per tutte le stagioni non riportate nella documentazione, poiché molto simili alla seguente regola

```
estate(City , Mese , Giorno) :- eNelContinente(City,X) , estateBoreale(X , Mese ,
Giorno) .
```

- La seguente regola indica se in un dato continente in un mese indicato e in un giorno indicato è estate boreale oppure no

N.B. Ci sono regole simili non riportate nella documentazione per ogni stagione , per ognuno dei due emisferi poiché molto simili alla seguente regola

```
estate_boreale(Continente ,Mese, Giorno) :- Giorno > 0 , Giorno < 31 ,
meseEstivoBoreale(Mese) ,
boreale(Continente).
```

- La regola che segue risponde **true** se il posto specificato è aperto nell' orario indicato.

```
lim_ora(amsterdam , palazzo_reale , Ora) :- Ora > 9.59 , Ora < 17.00 .
```

- La regola di seguito riportata risponde **true** se il posto indicato è aperto nel giorno specificato.

N.B. i giorni della settimana sono riportati come numeri nella KB ovvero ogni giorno ha il valore numerico di posizione nella settimana, per esempio Lunedì vale 1,
Martedì 2 ecc...

```
lim_giorno(Amsterdam , crociera_in_battello , Giorno) :- Giorno > 0 , Giorno < 7 .
```

- La seguente regola indica se in una data città, partendo da un certo continente serve il passaporto oppure no

```
non_serveIl_passaporto(City,ContPart) :- eNelContinente(City,ContPart) .
```

- La seguente regola risponde **true** se il luogo indicato, nella città indicata è aperto nel giorno e nell' ora specificata.

File e funzioni ausiliarie

Di seguito andremo ad esplicitare i file presenti nel progetto e funzioni.

Nella directory IconPj sono presenti, oltre al file **main.py** i seguenti file:

- Bayes.py
- Graph.py
- GeneralFunction.py

Bayes.py

In questo file è stata definita la rete bayesiana con le relative CPT, il file inoltre possiede una serie di funzioni che vengono chiamate dal main per poter sottoporre le query alla rete.

Graph.py

In questo file ci sono principalmente due funzioni, la funzione che calcola la distanza euclidea, e una funzione chiamata searchPath(Start,Arrival,City) la quale ritorna il percorso più breve tra i due punti specificati, sulla mappa "city" indicata.

GeneralFunction.py

In questo file abbiamo raccolto una serie di funzioni ausiliarie , come funzioni di check di stringhe o numeri, oppure funzioni di sistema.

Lista delle librerie esterne usate

- **Re** -> usata per le espressioni regolari
- **Numpy** -> usata per la manipolazione di matrici
- **Pyswip** -> usata per creare un bridge tra SWIProlog e Python
- **Os** -> usata per muoversi nel sistema
- **NetworkX** -> usata per implementare i grafi e ricerca
- **Matplotlib** -> usata per il plot dei grafi
- **Pgmpy** -> usata per l'implementazione della rete bayesiana
- **PIL** -> usata per aprire l'immagine png del grafo
- **Sys** -> usata per funzioni di sistema
- **PyQT5** -> usata per realizzare la grafica

N.B per la realizzazione dei file XML delle varie interfacce grafiche ci siamo avvalsi dello strumento QTDesign, il quale ci ha consentito di creare semplicemente e rapidamente interfacce grafiche con metodologia drag and drops.



Struttura della directory IConPj

