Risoluzione di Problemi di Viabilità Tramite Algoritmo A* in Prolog

Definizione del Problema

Dato un grafo pesato con nodi e connessioni, il problema è trovare il cammino più conveniente da un nodo iniziale (`Start`) a un nodo obiettivo (`Goal`). Il grafo è rappresentato tramite i seguenti predicati:

- `edge(Nodo1, Nodo2, Peso)`: definisce un arco tra `Nodo1` e `Nodo2` con un peso specifico.
- `heuristic(Nodo, Valore)`: definisce una stima del costo rimanente (funzione euristica) dal nodo corrente alla destinazione.

L'algoritmo utilizzato per risolvere il problema è una ricerca informata di tipo **A*** (*A-star search*).

Rappresentazione del Problema in Prolog

Ecco come rappresentare il problema in Prolog:

```
% Rappresentazione del grafo
edge(a, b, 1).
edge(a, c, 4).
edge(b, c, 2).
edge(b, d, 6).
edge(c, d, 3).

% Stima euristica (heuristic function)
heuristic(a, 7).
heuristic(b, 6).
heuristic(c, 2).
```

```
heuristic(d, 0).

% A* Search
astar(Start, Goal, Path, Cost) :-
    astar_search([(0, 0, [Start])], Goal, RevPath, Cost),
    reverse(RevPath, Path).

% Caso base: trovato il goal
astar_search([(Cost, _, [Goal|Rest])|_], Goal, [Goal|Rest],
   Cost).

% Espansione dei nodi
astar_search([(_, G, [Current|Rest])|Queue], Goal, Path, Cost
   ) :-
    findall((F, GNew, [Next, Current|Rest]),
        (edge(Current, Next, C),
         \+ member(Next, [Current|Rest]), % Evita cicli
         GNew is G + C,
         heuristic(Next, H),
         F is GNew + H),
        Successors),
    append(Queue, Successors, NewQueue),
    sort(NewQueue, SortedQueue), % Ordina in base a F
    astar_search(SortedQueue, Goal, Path, Cost).
```

Spiegazione del Codice

1. **Rappresentazione del grafo**: Gli archi e i loro pesi sono definiti tramite il predicato `edge/3`, mentre la funzione euristica è definita tramite il predicato `heuristic/2`.

2. **Algoritmo A***:

   - La lista di priorità è rappresentata da tuple della forma (F, G, Path), dove:
     - $F = G + H$ è il costo totale stimato.
     - G è il costo accumulato dal nodo iniziale.
     - `Path` è il cammino attuale.
   - Si espande il nodo con il costo F più basso.

3. **Espansione dei nodi**: Per ogni nodo corrente, si calcolano i nodi successori non ancora visitati e i rispettivi costi G e F.

4. **Soluzione**: Quando il nodo obiettivo (`Goal`) si trova in testa alla lista, viene restituito il cammino e il costo totale.

Esempio di Esecuzione

Supponendo di voler trovare il cammino più conveniente da a a d, si esegue la query:

```
?- astar(a, d, Path, Cost).
Path = [a, b, c, d],
Cost = 6.
```