```
Interface Etat {
};

Interface Action {
    resultat(e : Etat) : Etat ;    // état obtenu par application d'une action valide sur un état
    cout(e : Etat) : Reel ;        // coût de réalisation de l'action
};

Interface Probleme {
    etatInitial : Etat ;
    actions(e : Etat) : Ensemble d'Action ;     // actions possibles pour un état donné
    but?(e : Etat) : Booleen ;
};

Interface Noeud {
    etat : Etat ;
    parent : Noeud ;
    action : Action ;
    cout : Reel ;    // coût d'enchainement des actions pour atteindre ce noeud depuis la racine
    Noeud(e : Etat, p : Noeud, a : Action, c : Reel) : Noeud ;
};

Interface Liste<Noeud> {
    Liste() : Liste ;           // constructeur de liste vide
    vide?() : Booleen ;
    oterTete() : Noeud ;
    oterNoeud(n : Noeud) : void ;
    insererTete(n : Noeud) : void ;
    insererQueue(n : Noeud) : void ;
    insererCroissant(n: Noeud) : void ;
};
```

Fonction Explorer(p : Probleme) : Noeud (ou null)

```
racine ← new Noeud(p.etatInitial, null, null, 0) ;
frontiere ← new Liste<Noeud>() ;
frontiere.inserer(racine) ;
tant que non frontiere.vide ?() faire
    n ← frontiere.oterTete() ;
    si p.but ?(n.etat) alors retourner n;
    pour toute action a dans p.actions(n.etat) faire
        sn ← new Noeud(a.resultat(n.etat), n, a, n.cout+a.cout(e)) ;
        frontiere.inserer(sn) ;
retourner null ;
```