Enseignante : Mme Aya Abidi Étudiante : Amira Bouafif Classe : LGLSI 2B	Compte rendu	ISTIC Borj Cédria Matière :TP Intelligence Artificielle Année universitaire : 2024-2025

1- Objectif :

Comparaison de deux algorithmes Dijkstra et Uniform Cost Search

2- Code :

```python
def djikastra(g,s):
    d={} # création de dictionaire vide pour le cout de chaque sommet
    pi={} # création de dictionaire vide pour le pere de chaque sommet
    blanc=[] # création de list vide pour les sommet non visiter
    gris=[] # création de list vide pour les sommet decouvert
    noir=[] # création de list vide pour les sommet qui a été exploité

    for i in g: # boucle pour l'initialisation de d, pi et blanc
        d[i]=float('inf')
        pi[i]=''
        blanc.append(i)
# initialisation de cout de sommet s a zero puisqu'elle est la sommet de depart
    d[s]=0
# supprimer la sommet s de la list blanc et la remetre a la list gris
    gris.append(blanc.pop(blanc.index(s)))
# boucle pour determiner le plus court chemin de sommet s jusqu'il n'y'a encore des sommet a exploiter
    while gris :
        s1=mingris(d,gris) # determine la sommet qui a la distance le plus faibe

        for i in g[s1]:

            if i[0] in blanc or i[0] in gris: # verifier si la sommet est dans la list blanc/gris
                relacher((s1,i),pi,d) # comparaison entre le cout ancien de la sommet

                if i[0] in blanc : # verifier si la sommet est dans blanc
# suprimer la sommet de blanc et la remetre a la list gris pour indiquer que nous avons decouvert cette sommet
                    gris.append(blanc.pop(blanc.index(i[0])))
# supprimer la sommet de gris et la remettre dans noir pour indiquer que cette sommet a été exploiter
        noir.append(gris.pop(gris.index(s1)))

    return pi,d
```

```python
def UCS(g,s,n):
    '''initialisation de dictionneur de priorité qui va contenir les sommet et leur
    cout ce dictionnaire trier de moin prioritaire au plus prioritaire selon leur cout
    le sommet ayant le moin cout est le plus prioritaire'''
    priority={s:0}
    #initialisation de liste des sommets exploré
    explored=[]
    #inisialization de la liste qui va contenir le chemin
    chemin=[]
    #tanque il ya encore du sommets à exploré
    while priority!={}:
        #supprimer la sommet ayant le moin cout
        K,V=priority.popitem()
        #si cette sommet et notre sommet d'arriver
        if K==n:
            #ajouter la sommet au chemin
            chemin.append(K)
            #retourne le cheminet le cout
            return chemin,V
        #si non si elle n'est pas encore exploré
        elif K not in explored:
            #ajouter la sommet dans la liste du sommets exploré
            explored.append(K)
            # pour chaque successeur de  de cette sommet
            for i in g[K]:
                '''changer le cout en ajoutant le cout de S->K (de sommets de depart à la
                sommet selectionée) avec le cout de successeur'''
                c=V+i[1]
                #ajouter le successeur au dict de priorité
                priority[i[0]]=c
                #trier le dictionnaie selon la priorité
            priority=dict(sorted(priority.items(),key=lambda item:item[1],reverse=True))
    # si touts les sommets sont exploré et on ne touve pas la sommet d'arrivé
    print("chemin non trouvé")
```

3- Comparaison :

Dijkstra	UCS
• Il sert à trouver tous les chemins minimaux, depuis un sommet de départ vers tous les autres sommets de graphe • Il commence par initialiser la distance de tous les sommets à l'infini avec la distance de sommet de départ on zéro puis il visite tous les sommets qui a découverts on prend à chaque fois l'un ayant le plus moins cout avec mise en jour de distance et parent si un chemin plus court est trouvé et si tous les sommets sont exploités il s'arrête • Moins efficace et moins rapide puis qui il ne s'arrête qu'après avoir exploiter tous les sommets de graph	• Il sert à trouver un chemin optimal, celui qui permet d'atteindre le sommet d'arrivé depuis un sommet de départ avec le moins cout • Il commence par initialiser un dictionnaire de priorité qui est trier selon la priorité des sommets avec celui ayant le moins cout est le plus prioritaires c'est celui qu'on va explorés le premier si on trouve le sommet d'arrivé on s'arrête si non on ajoute le sommet au dictionnaire de priorité sans oublier de le trier selon la priorité des sommets ajouter • Plus rapide puis qu'il s'arrête si on trouve le sommet d'arriver, notre goal

• Retourne les chemins optimaux pour tous les sommets de graph à partir de sommet de départ donc en dois extraire le chemin optimal qui nous aide à arriver à notre goal	• Retourne le chemin optimale

→Dijkstra est UCS sans un goal et UCS est Dijkstra avec un goal

4- Conclusion :

- On Utilise Dijkstra si on veut trouver tous les chemins optimaux possible d'un sommet spécifique sans un goal spécifique.
- On Utilise Uniform Cost Search si on veut trouver le chemin optimal d'un sommet spécifique à un goal spécifique.