Rapport

Cette fois , je fais les trois projets, pour moi , je trouve que les premier deux sont plus faciles , et le dernier ça prend du temp .

1ER CODE

Le premier code , il faut utiliser le méthode de l'algorithme de parcours en largeur , je commence du 1er nœud , puis ses successeurs , puis les successeurs de ceux successeurs…

Pour ce code , je utilise le ‘struct’ de stack , c’est just pour reviser les cours de TP et c'est plus claire pour conserver le résultât . Pour la fonction , je utilise deux tableaux : find ,et Q . ‘find’ est le tableau des noeuds qui sont déjà visités , ‘Q’ est le tableau de ‘openset’ . je commence de le 1er noeud de tableau Q , et ajouter les successeurs de ce noeud , et après je le sorts de tableau Q , donc je peux utiliser le prochain noeud de Q , ajouter ses successeurs. c'est le 1er successeur de 1er noeud . Et après c'est le 2eme successeur du 1er noeud , jusqu’à je trouve le ‘end’….. Quand le dernier noeud est sort du tableau Q , ça signifie je ne peux pas trouver le ‘end’ . Si je trouve le ‘end’ , je sorts du cycle et montre le résultat .

Ce code n’est pas très difficile . Mais c'est très utile pour reviser l'algorithme de parcours en largeur .

2EME CODE

Le 2eme code , il faut utiliser le méthode de l'algorithme de Dijkstra , l'algorithme de Dijkstra mise à les distances des noeuds . Au début les distances sont infini .

Pour ce code , je utilise le ‘struct’ de stack , c’est just pour reviser les cours de TP et c'est plus claire pour conserver le résultât . Pour la fonction , je utilise trois tableaux : visited , distance ,et parent . Dans ‘visited’ sont les noeuds qui sont déjà visités , ‘distance’ sont les distances de ce noeud à le 1er noeud , ‘parent’ sont le précédent de ce noeud . Pour la commencement , je mise à les distance des successeurs du 1er noeud , et choisis le sommet qui a le distance le plus court , et mise les distance des successeur de ce noeud …. jusqu’à je trouve le ‘end’ . Si après N(size of les noeuds) pas , si je trouve pas le ‘end’ , ‘print’ ‘Not connected’ .

Ce code n’est pas très difficile . Mais c'est très utile pour reviser l'algorithme de Dijkstra.

3EME CODE

Le 3eme code , il faut utiliser le méthode de l'algorithme de A* , c'est un peu difficile , parce que je le connais pas , mais après je l’ai résolu.

Pour ce code , je utilise deux ‘struct’ , le 1er stack , c’est just pour reviser les cours de TP et c'est plus claire pour conserver le résultât . Le 2eme est les noeuds , dans ce struct il ya :

F:= H + G

G:= le distance de ce noeud à 1 er sommet.

H:=le distance de ce noeud à sommet ’S'.

sym := comme le matrix du plan.

pointeur parent : pour conserver le pas précédent .

table is_closed et is open : ajouter les voisins des sommets dans is_open , après les mets dans is_closed.

Et pour ce code , j’ai crée trois fonctions , ‘distance’ est pour calculer H des sommets , la fonction ‘insert_to_opentable’ , c'est pour ajouter les voisins des sommets dans ‘open_table’ , la fonction ‘A’ est la fonction de A* .

Dans la fonction A* , je crée N*N ‘struct node’ , pour conserver les informations du plan. Et un tableau de ‘open_table’, c’est comme l'algorithme de parcours en largeur, au début , dans ce tableau il y a juste le premier noeud, et après ajouter ses voisins dans ‘open_table’, ses parents sont le 1er noeud ses G sont G de parent plus 10 , après je sorts ce noeud du tableau ‘open_table’ ,et utilise le prochain noeud. La différence est je ajoute le noeud qui a plus grand F. Jusqu’à je trouve le ‘end’. Et si G est plus de limit de ‘timer’ *10 , ajoute pas .

Pour la clé et la porte , je change la porte à ‘X’ , s’il y a la porte et je peux pas arriver à le noeud ’S’ , je change la fonction de ’S’ à ‘a’ , après de ‘a’ à ’S’. Pour les noeuds pour ‘TP’ , Apres ajouter les voisins de ce noeud , ajoutent autre noeud de ‘TP’, son parent est ce noeud aussi. quand je trouve le ‘end’, je push ce noeud dans un stack , après push son parent , après … Jusqu’à le 1er noeud.

Pour le résultat , les 18 test j’ai réussi 17 , et le 18eme il y a un ‘bug’ , je peux trouver que la rout de ‘E’ à ‘a’, mais de ‘a’ à ’S’, ça prend trop du temp j’ai pas trouvé la problem.

Pour ce code , ça prend trop du temp , mais après je l'ai résolu , je me sens excité. C'est une très bonne question.

Le cours du semestre est terminé， j'ai appris beaucoup de connaissances, je me sens beaucoup récolté, mais il y a des zones pour faire comprendre que ce n'est pas très approfondi.

à la fin , j'ai crée les codes des modules, parce que chaque fichier ego*.c utilise différent stock.h ,donc je les ai nommé 'stack1.h' 'stack2.h' et ' stack3.h' pour exo1 , exo2 , et exo3.

Merci beaucoup.