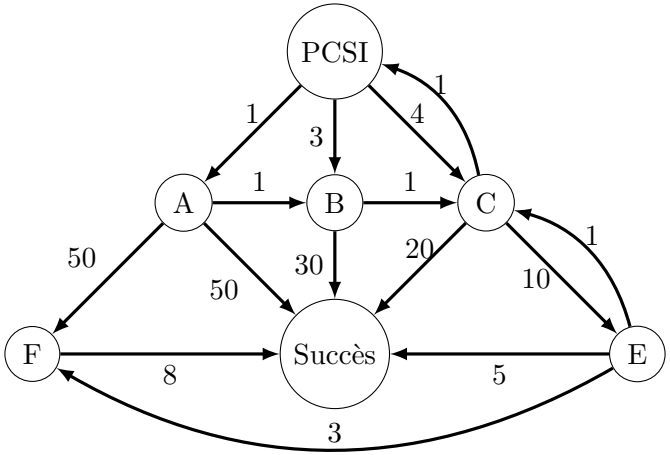
DS5info

6 juin 2024

devilliers.loic@gmail.com

Dijkstra : trouver le plus court chemin vers le succès aux concours (spoiler : c'est par le travail régulier !)

Considérons le graphe suivant :



On souhaite appliquer l'algorithme de Dijkstra à ce graphe partant du sommet `PCSI`. La colonne `T` indique le point que l'on traite actuellement, `D[E]` représentera la distance entre le sommet PCSI et E, `P[E]` représentera le prédécesseur de E.

Étape	T	D[PCSI]	D[A]	D[B]	D[C]	D[E]	D[F]	D[Succès]	P[PCSI]	P[A]	P[B]	P[C]	P[E]	P[F]	P[Succès]
0		0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	PCSI						
1															
2															
3															
4															
5															
6															
7															

1. Complétez le tableau. Vous rendrez donc cette feuille en ayant pris soin d'indiquer votre nom.
2. Quel est le plus court chemin partant du sommet `PCSI` et arrivant au sommet `Succès` ? Justifier brièvement votre réponse.

Graphe : j'espère que vous avez parcouru votre cours en profondeur Sinon, vous vous planterez dans les grandes largeurs

Dans cet exercice, on considère un graphe orienté et non pondéré dont les sommets sont numérotés par des entiers entre 0 et $n-1$. On modélise ce graphe par sa matrice d'adjacence `M`.

1. Rappeler que vaut le coefficient `M[i][j]`.
2. Écrire une fonction `Listevoisins(M:list,som:int)->list` qui, étant donnée une matrice d'adjacence `M` et un sommet noté `som`, renvoie la liste des voisins de `som`.
3. Écrire une fonction `Parcours(M,som)` qui réalise un parcours du graphe dont la matrice d'adjacence est `M` partant du sommet `som`. Vous pourrez faire au choix un parcours en largeur ou en profondeur utilisant ou non la récursivité. Vous justifierez la nature de votre parcours : profondeur ou largeur, pile ou file (si besoin).
4. Quelles sont les éventuels inconvénients (en terme de complexité) de votre programme ? Expliquer en quelques lignes quelles solutions il est possible de mettre en place.
5. Considérons `L=[[1,2,3],[4,5,6]]`, donner les valeurs de `len(L)`, `L[0]`, `len(L[0])` et de `L[1][1]`.
6. Écrire une fonction `CoeffsNonNuls(M:list)->bool` qui, à une matrice carrée `M` renvoie `True` si tous les coefficients non diagonaux de la matrice sont non nuls et `False` sinon.
   Cela permet de savoir si le graphe est complet : c'est-à-dire, qu'étant donné deux sommets différents, il existe toujours un arc reliant l'un de ces sommets à l'autre.
7. Soient A et B deux matrices en python telles que A ait n lignes et p colonnes et B ait p lignes et q colonnes. Alors, on rappelle que si on pose $C = AB$, alors les coefficients de C valent :

$$C[i][j] = \sum_{k=0}^{p-1} A[i][k]B[k][j]$$

   Écrire une fonction `Produit(A:list,B:list)->list` qui a deux matrices renvoie la matrice $C = AB$. On indiquera, à l'aide de la commande `assert`, une ligne qui permet de savoir si le produit est valide.
8. Écrire une fonction `Puissance(A:list,k:int)->list` qui, à une matrice carrée `A` et un entier naturel k, renvoie A^k.
9. Écrire une fonction `Puissance2(A:list,k:int)->list` similaire à la question précédente, mais utilisant l'exponentiation rapide basé sur le principe suivant : $A^{2p} = (A^p)^2$ et $A^{2p+1} = (A^p)^2 A$.
10. On admet que, pour deux sommets i et j, le nombre de chemins de i vers j de longueur k (c'est-à-dire utilisant exactement k arrêtes) vaut le coefficient à la i-ième ligne et j-ième colonne de M^k. Écrire une fonction `NbChemins(i,j,k,M)` renvoyant le nombre de chemin de i vers j de longueur k.