Projet PERT

Monnier Benjamin
Mourret Basile
Mouze Guillermo

0)Notre classe graphe

```python
class Graphe :
    def __init__(self, noeuds : set, arcs : list[tuple[int,int]],poids :dict[str : int],description:dict[str:str]) -> None:
        """rajouter les descriptions

        Args:
            noeuds (set): _description_
            arcs (_type_): _description_
        """
        self.noeuds = noeuds
        self.adj = {n : [a[1] for a in arcs if a[0] == n] for n in noeuds}
        self.poids = poids
        self.description=description
```

1)De tableau python à csv

```python
def to_csv(name_csv_file: str, fieldnames:list[str], rows:list[list[str]]) -> None:
    """
    Args:
        namecsvfile (str) : nom du fichier csv (sanssuffixe)
        fieldnames(list[str]): liste des noms de champs
        rows(list[list[str]]): liste des lignes (liste de colonnes)
    """
    with open(name_csv_file + ".csv" ,'w', encoding="utf-8") as csvfile:
        csvwriter=csv.writer(csvfile,delimiter=',')
        csvwriter.writerow(fieldnames)
        for r in rows:
            csvwriter.writerow(r)
```

Fonction créée pour convertir le graphe du cours en fichier csv, ce code à été donné dans le cours.

Fichier csv généré

```
PC,Permis de construire ,60,,0,,
F,Fondations,7,PC,1,,
GE1,Passage des gaines et évacuations ,3,F,2,,
DRC,Dalle rez de chaussée,7,GE1,1,,
MP,Murs porteurs ,14,DRC,2,,
DP,Dalles plafond ,7,MP,,,
GE2,Passage gaines et évacuation,3,DP,,,
T,Toiture,14,GE2,,,
FE,Fenêtres,7,T,,,
PE,Portes extèrieures,3,T C,,,
IE,Installation électrique et évacuation,3,GE2,,,
C,Chape,7,DP,,,
C1,Carrelage du sol,7,C0,,,
P,Parquets,7,C,,,
CM,Cloisons et menuiserie intérieure,10,FE C1 P FC,,,
FC,Finition des cloisons,7,CM,,,
IC,Implantation de la cuisine ,7,FC,,,
IW,Implantation des wc,3,FC,,,
IS,Implantation des salles de bain,7,FC,,,
PP,Peinture des plafonds ,7,IC IW IS,,,
PM,Peinture des murs,7,PP,,,
S,Serrurerie,3,P CM,,,
R1,Revêtement des sols (moquettes) ,2,C S,,,
R,Réception de la maison ,0.5,MP S,,,
```

2)De csv à un graphe

Une fois le fichier csv créé comportant les données de notre projet, il est nécessaire de le convertir en 4 graphes pour pouvoir exploiter les données.

La fonction `csv_to_Graphe` renvoit donc un tuple contenant:

- le graphe du projet initial
- les 3 graphes des suivis du projet

```python
def csv_to_Graphe(nom_fichier:str,k:int=0) -> Graphe:
    """
    Creer et renvois le graphe associé à un fichier csv.
    Args:
        nom_fichier (str): nom fichier à convertir
        k(int):nombre de suivi réalisé
    Returns:
        Graphe: Graphe issu du fichier.
    """
    Dict_poids={}
    Dict_description={}
    with open(nom_fichier+'.csv','r',encoding='utf8') as file:
        noeud=set()
        arcs=[]

        if k==0:
            for ligne in file:
                s=ligne.split(',')
                noeud = noeud  | {s[0]}
                Dict_poids[s[0]] = float(s[2])
                Dict_description[s[0]]=s[1]
                if len(s)>3 and not s[3]=='':
                        arcs += couple(s)
            return(Graphe(noeud,arcs,Dict_poids,Dict_description),csv_to_Graphe(nom_fichier,k+1),csv_to_Graphe(nom_fichier,k+2),csv_to_Graphe
        else:
            for ligne in file:
                s=ligne.split(',')
                if s[3+k]=='' or s[3+k]=='\n':
                    return None
                else:
                    noeud = noeud  | {s[0]}
                    Dict_poids[s[0]] = float(s[3+k])
                    Dict_description[s[0]]=s[1]
                    if len(s)>3 and not s[3]=='':
                            arcs += couple(s)
                    else:
                        pass
            return Graphe(noeud,arcs,Dict_poids,Dict_description)
```

Fonctionnement 2.0:

Création d'un graphe

```python
for ligne in file:
    s=ligne.split(',')
    noeud = noeud  | {s[0]}
    Dict_poids[s[0]] = float(s[2])
    Dict_description[s[0]]=s[1]
    if len(s)>3 and not s[3]=='':
            arcs += couple(s)
return(Graphe(noeud,arcs,Dict_poids,Dict_description),csv_to_Graphe(nom_fichier,k+1),
      csv_to_Graphe(nom_fichier,k+2),csv_to_Graphe(nom_fichier,k+3))
```

La fonction va récupérer grâce à une boucle itérative pour chaque ligne i de notre fichier csv:

-le noeud i en l'ajoutant à l'ensemble des noeuds

-la durée nécessaire pour effectuer la tâche du noeud i. On créé un dictionnaire où chaque tâche est reliée à sa durée(ici poids).

-la description nécessaire pour afficher les différentes tâche dans le rendu final.

On créé un dictionnaire où chaque diminutif est reliée à sa tâche.

-les arcs reliant le noeud i à d'autre noeud
Pour ce faire nous avons créer une fonction `couple(s:list)` renvoyant tout les arcs d'un certain noeud.

Fonctionnement 2.0.1:

```
def couple(s:list)->tuple:
    """retourne tout les arcs possibles relatif à une certaine tâche sous la forme de tuple(precedents,arrivée,durée)
    Args:
        s (list): éléments de la ligne sous fortme de liste

    Returns:
        tuple: retourne les arcs du noeuds de la i-ème ligne
    """
    arcs=[]
    if s[3]=='Précédente(s)' :      Basile Mouret, il y a 3 jours • fin chemin critique
        #return [[],'Tâche','Poids']
        return []
    else:
        precedents=s[3].split()
        for i in range(len(precedents)):
            arcs.append((precedents[i],s[0]))
        #return [arcs,s[0],s[2]]
        return arcs
```

Les arcs sont définis comme une liste de tuple(int,int) dans notre programme. Cette fonction prend donc en entrée la liste des éléments de la ligne i et va créer k-tuples:(noeud,k précédents).

FONCTIONNEMENT 2.1

Afin de créer le graphe du kème- suivi il suffit de changer les poids de notre graphe obtenu en les remplaçant par ceux renseignés par l'utilisateur dans notre fichier csv .Pour cela on utilise 3 appelle récursif de notre fonction `csv_to_Graphe.` Afin que ces appels ne soit pas infinis, on définit un paramètre k correspondant au numéro de suivi que l'on souhaite obtenir. On réalise donc une disjonction de cas pour que si l'on souhaite tracer le graphe d'un suivi alors la fonction ne nous retourne que le graphe et non un appelle récursif d'elle même.

```
if k==0:
    for ligne in file:
        s=ligne.split(',')
        noeud = noeud  | {s[0]}
        Dict_poids[s[0]] = float(s[2])
        Dict_description[s[0]]=s[1]
        if len(s)>3 and not s[3]=='':
                arcs += couple(s)
    return(Graphe(noeud,arcs,Dict_poids,Dict_description),csv_to_Graphe(nom_fichier,k+1),csv_to_Graphe(nom_fichier,k+2),csv_to_Graphe
else:
    for ligne in file:
        s=ligne.split(',')
        if s[3+k]=='' or s[3+k]=='\n':
            return None
        else:
            noeud = noeud  | {s[0]}
            Dict_poids[s[0]] = float(s[3+k])
            Dict_description[s[0]]=s[1]
            if len(s)>3 and not s[3]=='':
                    arcs += couple(s)
            else:
                pass
    return Graphe(noeud,arcs,Dict_poids,Dict_description)
```

Les poids des graphes de suivi ont changé et ne se trouve plus à la position 3 du tableau csv mais à celle 4+k.
Il nous suffit donc pour les créer de reprendre le meme code juste en changeant l'indice des poids.

```
for ligne in file:
    s=ligne.split(',')
    if s[3+k]=='' or s[3+k]=='\n':
        return None
    else:
        noeud = noeud  | {s[0]}
        Dict_poids[s[0]] = float(s[3+k])
        Dict_description[s[0]]=s[1]
        if len(s)>3 and not s[3]=='':
                arcs += couple(s)
        else:
```

Cas où un suivi mal renseigné où non renseigné
Le programme vérifie que chaque poids de chaque noeuds i est renseigné dans le cas inverse Le suivi étant incomplet , il renvoie à la place du graphe du suivi :None.

```
for ligne in file:
    s=ligne.split(',')
    if s[3+k]=='' or s[3+k]=='\n':
        return None
    else:
        noeud = noeud  | {s[0]}
        Dict_poids[s[0]] = float(s[3+k])
        Dict_description[s[0]]=s[1]
        if len(s)>3 and not s[3]=='':
                arcs += couple(s)
        else:
            pass
return Graphe(noeud,arcs,Dict_poids,Dict_description)
```

Fonction Parents: Cette fonction renvoie un dictionnaire avec les parents d'un noeuds, elle est utile durant l'analyse du chemin critique.On retrouve les fils d'un noeuds dans graphe.adj.

```
def parents(graphe):
    """retrouve les noeuds parents de chaque noeud
    Returns:
        dict[noeud:set] : dictionnaire contenant le noeud en indice et
son/ces parents
    """
    dico_parents = {n : set() for n in graphe.noeuds}
    for n_d in graphe.noeuds:
        for n_f in graphe.adj[n_d]:
            dico_parents[n_f].add(n_d)
    return dico_parents
```

Fonction Forward_Pass : Cette fonction permet de déterminer les dates de débuts de chaque Tache. Elle s'appuie sur un tri topologiques pour ordonner les taches.

```
def forward_pass(graphe) :
    """Détermine les dates de départ au plus tot et de fin au plus tot
(départ au plus tot + )
    Args:
        graphe (_type_): _description_
    Returns:
        _type_: _description_
    """
    #forward pass
    debut_tot = {n : -1 for n in graphe.noeuds}#date de départ minimum
    fin_tot = {n : -1 for n in graphe.noeuds}#date de fin minimum
    par = parents(graphe)
    noeuds_départ  = {n for n in graphe.noeuds if par[n] == set()}#
donne les noeuds de départ du graphe (qui n'ont pas de parents)
    file = []
    for n in noeuds_départ:
        debut_tot[n] = 0
        fin_tot[n] = graphe.poids[n]#date de départ minimale
        file += graphe.adj[n]

    while len(file)>0:
        n = file.pop(0)
        #on vérifie que tout les parents du noeuds ont déjà étés
parcourus
        parents_parcourus = True
```

```
        for n_parent in par[n]:
            if fin_tot[n_parent]<0: parents_parcourus = False

        #si tout les parents sont parcourus, on détermine le début au
plus tot
        if parents_parcourus:
            file += graphe.adj[n]
            for n_parent in par[n]:
                if fin_tot[n_parent]>debut_tot[n]:
                    debut_tot[n] = fin_tot[n_parent]
                    fin_tot[n] = fin_tot[n_parent]+graphe.poids[n]
    return debut_tot,fin_tot
```

Fonction Backward_Pass : Cette fonction détermine les dates de début au plus tard et de fin au plus tard. Il s'agit du même algorithme que le Forward_Pass, mais dans le sens inverse, elle part des noeuds de fin.

```
def backward_pass(graphe,debut_tot,fin_tot):
    #backward pass
    #même principe que à l'aller mais on parcours le graphe dans le
sens inverse
    debut_tard = {n : math.inf for n in graphe.noeuds}#date de départ
maximum
    fin_tard = {n : math.inf for n in graphe.noeuds}#date de fin
minimum
    par = parents(graphe)
    noeuds_fin = {n for n in graphe.noeuds if
len(graphe.adj[n])==0}#noeud de fin
    file = []
    duree_chemin_critique = max(fin_tot.values())

    for n in noeuds_fin:
        fin_tard[n] = duree_chemin_critique
        debut_tard[n] = duree_chemin_critique - graphe.poids[n]#date de
départ minimale
        file += par[n]



    while len(file)>0:
        n = file.pop(0)
        #on détermine si tout les fils ont étés parcourus
        fils_parcourus = True
```

```
        for n_fils in graphe.adj[n]:
            if fin_tard[n_fils] is None: fils_parcourus = False
        #si tous les fils sont parcourus, on détermine la date de fin
au plus tard (cad la plus petite des dates de début au plus tot de ses
fils)
        if fils_parcourus:
            file += par[n]
            for n_fils in graphe.adj[n]:
                if debut_tard[n_fils]<fin_tard[n]:
                    fin_tard[n] = debut_tard[n_fils]
                    debut_tard[n] = debut_tard[n_fils]-graphe.poids[n]

    return debut_tard,fin_tard
```

Fonction Noeuds_critiques : Cette fonction détermine si les noeuds peuvent prendre du retard. Elle retourne les noeuds qui ne le peuvent pas, ils sont dis critiques.

```
def noeuds_critiques(graphe,debut_tot,fin_tard):
    """Détermine les noeuds qui ne peuvent pas avoir de retard

    Args:
        graphe (Graphe): graphe a analyser
        debut_tot (dict): dates de début au plus tot de chaque tache
        fin_tard (dict): dates de fin au plus tard de chaque tache

    Returns:
        set: noeuds dis critique
    """
    noeud_critiques = set()
    for noeud in graphe.noeuds:
        if debut_tot[noeud]+graphe.poids[noeud] == fin_tard[noeud]:
            noeud_critiques.add(noeud)
    return noeud_critiques
```

Fonction Analyse_Pert : Cette fonction permet de rassembler l'analyse d'un projet PERT
Elle retourne les noeuds triés par ordre topologique, les noeuds qui sont dit critiques,
les dates de départ au plus tôt de chaque tache et les dates de fin au plus tard.

```
def analyse_PERT(graphe):
    """fonction principale d'analyse PERT

    Args:
```

```
        graphe (Graphe): graphe à analyser
    """
    #déterminer les dates importantes pour l'analyser
    debut_tot,fin_tot= forward_pass(graphe)
    debut_tard,fin_tard = backward_pass(graphe,debut_tot,fin_tot)
    #déterminer les noeuds qui sont critiques
    n_crit = noeuds_critiques(graphe,debut_tot,fin_tard)
    #on trie les noeuds dans l'ordre topologique (chaque noeuds peut
etre réalisé si tout les noueds qui le précèdent sont réalisés)
    n_triés = sorted(debut_tot, key = debut_tot.get)
    return(n_triés,n_crit,debut_tot,fin_tard)
```

Le Fichier analyse.py

Le Fichier Analyse Permet de créer un fichier latex qui une fois compiler donne le compte rendu de l'analyse du projet. Ce fichier à été compiler pour créer un application (analyse.exe).
L'application demande à l'utilisateur de rentrer le nom du fichier (format csv, décrit dans le manuel utilisateur). Il crée alors un fichier latex dans le dossier Résultat_analyse_PERT.

Vous pouvez trouver les exemples de lancement du logiciel pour nos exemples de projets dans le dossier du compte rendu.

Tests

Le fichier test permet de vérifier le bon fonctionnement de nos fonctions. Pour les fonctions les plus compliquées (analyse) nous avons vérifié à la main grâce à nos nombreux exemples.
Résultat du fichier test :
collected 5 items

tests.py .....
[100%]

==========
5 passed in 0.43s
==========

Bilan

Nous avons trouvé ce Projet très intérressant, le programme PERT étant très utilisé en Entreprises, il etait judicieux de l'étudier en Prépa. On a aussi apris beaucoup sur le travail de groupe et le partage de code grâce à Git et Github. On a également découvert des bibliothèques Latex, qui nous sont utile pour rédiger des comptes rendus.