Présentation de l'Algorithme

1. Objectif Principal

L'algorithme a pour but d'assigner chaque événement/réunion à un créneau horaire où toutes les ressources nécessaires (salles, collaborateurs, équipements) sont disponibles simultanément.

2. Structure de l'Algorithme

1. **Initialisation** :
   * resultats : Liste pour stocker les affectations réussies
   * alertes : Liste pour les événements non planifiables
   * creneaux\_utilises : Ensemble pour éviter les doubles réservations
2. **Parcours des événements** :
   * Pour chaque événement à planifier, l'algorithme cherche un créneau approprié
3. **Vérification des disponibilités** :
   * Pour chaque créneau, vérifie si toutes les ressources requises sont disponibles
   * Utilise any() pour vérifier si au moins une disponibilité de la ressource correspond au créneau
4. **Affectation** :
   * Si toutes les ressources sont disponibles et le créneau n'est pas déjà utilisé :
     + Ajoute l'affectation aux résultats
     + Marque le créneau comme utilisé
     + Passe à l'événement suivant
5. **Gestion des échecs** :
   * Si aucun créneau valide n'est trouvé, ajoute un message d'alerte

3. Complexité

* Complexité temporelle : O(n\*m\*k) où :
  + n = nombre d'événements
  + m = nombre de créneaux
  + k = nombre de ressources par événement
* Complexité spatiale : O(n) pour stocker les résultats

4. Avantages

* Simple à comprendre et implémenter
* Garantit que toutes les ressources nécessaires sont disponibles
* Évite les conflits de réservation

5. Limites

* Algorithme glouton qui ne cherche pas la solution optimale
* Ne gère pas les priorités entre événements
* Ne tient pas compte de la durée des événements (suppose que la durée correspond exactement aux créneaux)
* Ne gère pas les contraintes avancées (comme le nombre max d'heures par jour)

6. Améliorations Possibles

1. **Optimisation** :
   * Utiliser un algorithme de programmation linéaire (comme suggéré par l'import pulp)
   * Implémenter un système de priorités pour les événements
2. **Gestion de la durée** :
   * Permettre des événements de durée variable
   * Gérer les créneaux qui se chevauchent partiellement
3. **Contraintes avancées** :
   * Prendre en compte les préférences horaires
   * Limiter le nombre d'heures consécutives
   * Éviter les trous dans l'emploi du temps

7. Exemple d'Exécution

Prenons un exemple simple :

**Créneaux disponibles** :

* Lundi 9h-10h
* Mardi 14h-15h

**Ressources** :

* Salle A (dispo Lundi 9h-10h)
* Collaborateur B (dispo Lundi 9h-10h et Mardi 14h-15h)

**Événements** :

* Réunion X (nécessite Salle A et Collaborateur B)

**Exécution** :

1. Pour Réunion X, teste Lundi 9h-10h :
   * Salle A dispo ? Oui
   * Collaborateur B dispo ? Oui
   * Créneau non utilisé ? Oui  
     → Affecte la réunion à ce créneau

Résultat : Réunion X planifiée le Lundi de 9h à 10h

Cet algorithme constitue une base solide pour un générateur d'emploi du temps, qui pourrait être enrichi avec des fonctionnalités plus avancées pour répondre à des besoins spécifiques.