**Synthèse des fonctions clés et de leur rôle**

**1. Gestion des pixels et des images**

* **Graphe::Graphe(const char\* nomFichier)**
  + Lit une image PGM et initialise la structure des pixels.
  + Important pour charger une forme noire sur fond blanc dans un graphe.
* **afficherImage()**
  + Affiche l'image en console avec ses intensités lumineuses, utile pour visualiser rapidement une image.
* **sauvegarderImageForme()**
  + Sauvegarde l'image actuelle au format PGM, incluant les dimensions et l'intensité maximale.

**2. Calcul des distances avec Dijkstra**

* **calculerImageDistance(std::vector<int>& distances, std::vector<int>& predecesseurs)**
  + Calcule la distance minimale entre chaque pixel blanc et le pixel noir le plus proche à l’aide de l'algorithme de Dijkstra.
  + Gestion des coûts :
    - **2** pour les déplacements orthogonaux (haut, bas, gauche, droite).
    - **3** pour les déplacements diagonaux.
  + Met à jour :
    - **distances** : Distance minimale pour chaque pixel.
    - **predecesseurs** : Index du prédécesseur pour chaque pixel, permettant de retracer le chemin.
* **sauvegarderImageDistance()**
  + Sauvegarde les distances calculées dans un fichier PGM avec une intensité lumineuse appropriée.
  + Les distances infinies (pixels inaccessibles) sont représentées par **255** (blanc).
* **sauvegarderImagePredecesseurs()**
  + Sauvegarde les prédécesseurs pour chaque pixel dans un fichier texte, utile pour tracer les chemins.

**3. Opérations sur plusieurs images**

* **unionImages(const Graphe& autre)**
  + Combine deux images en prenant la distance minimale pour chaque pixel.
  + Vérifie que les dimensions des deux images sont compatibles avant d’effectuer la fusion.
  + Résultat : Une nouvelle image avec l’union des deux formes.
* **projeterPointSurUnion()**
  + Projette un point donné sur l’union de deux formes.
  + Compare les distances du point aux deux formes pour trouver la plus proche.
  + Retourne les coordonnées du pixel noir correspondant.

**4. Projection et voisinage**

* **projeterPixel()**
  + Trouve le pixel noir le plus proche d’un point donné en utilisant la distance de Manhattan.
  + Utile pour des opérations ponctuelles sans recalculer les distances pour toute l'image.
* **getVoisin()**
  + Retourne l’index global d’un pixel voisin dans une direction donnée.
  + Vérifie que le voisin est dans les limites de l’image.

**Résumé des étapes clés pour travailler avec les images**

1. **Chargement et visualisation :**
   * Charger une image PGM avec **Graphe::Graphe(const char\* nomFichier)**.
   * Visualiser l'image avec **afficherImage()**.
2. **Calcul des distances :**
   * Calculer les distances avec **calculerImageDistance()**.
   * Sauvegarder ces distances avec **sauvegarderImageDistance()** pour une visualisation ultérieure.
3. **Opérations sur les formes :**
   * Fusionner deux formes avec **unionImages()** pour obtenir l’union des distances.
   * Projeter un point sur une forme ou une union avec **projeterPixel()** ou **projeterPointSurUnion()**.
4. **Gestion des prédécesseurs :**
   * Sauvegarder les chemins optimaux avec **sauvegarderImagePredecesseurs()**.

**Complexité et performance**

1. **Algorithme de Dijkstra :**
   * Complexité : **O((E + V) log V)**, où :
     + **E** est le nombre d’arcs (voisins, maximum 8 par pixel).
     + **V** est le nombre de pixels (**L × C**).
2. **Opérations pixel par pixel :**
   * Fonctions comme **unionImages()** et **projeterPixel()** ont une complexité linéaire : **O(L × C)**.
3. **Sauvegarde des données :**
   * Les fonctions de sauvegarde sont également linéaires par rapport au nombre de pixels.

**Points importants pour l'oral**

1. **Présentation des distances :**
   * Expliquez comment les distances sont calculées à l’aide de Dijkstra et des coûts directionnels (orthogonaux et diagonaux).
2. **Union des images :**
   * Décrivez la fusion des distances pour créer l’union des formes.
3. **Projection :**
   * Montrez comment projeter un point sur une forme ou son union.
4. **Gestion des cas limites :**
   * Vérification des dimensions avant fusion.
   * Gestion des pixels hors limites ou inaccessibles.
5. **Applications :**
   * Analyse et manipulation des images binaires (formes noires sur fond blanc).
   * Calcul de distances pour des opérations géométriques ou d’optimisation.