EPFL

ICC
Programmation

GC/MX, **Cours 10**

25 novembre 2022

Jean-Philippe Pellet

Opening music: Queen, album *A Night at the Opera* (1975)

Previously, on Programmation...

- Types de base en Python: int, float, str, bool
- Méthodes, fonctions et slicing pour calculer des valeurs dérivées
- Conditions pour exécuter du code selon la valeur d'une expression booléenne: `if <condition>: ... else: ...` et ses variantes
- Boucles pour exécuter du code plusieurs fois:
  - Boucle `while <condition>: ...`
  - Boucle `for i in range(...): ...`
- Déclaration de fonctions avec type de retour et paramètres:
  - `def calculate_area(r: float) -> float: return ...`
- Utilisation de listes, sets et dictionnaires
- Déclaration de classes: `@dataclass class Rectangle: ...`
- Création, chargement, manipulation et sauvegarde d'images

```
try
{
    Assert(Life.Real);
    Assert(Life.Fantasy);
}
catch (LandSlideException ex)
{
    #region Reality
    while (true)
    {
        character.Eyes.ForEach(eye =>
            eye.Open().Orient(Direction.Sky).See()
        );
        self.Wealth = null;
        self.Sex = Sex.Male;

        self.Sympathies.Clear();

        if (self.ComeDifficulty == Difficulty.Easy &&
            self.GoDifficulty == Difficulty.Easy &&
            self.High < 0.1 &&
            self.Low < 0.1)
        {
            switch (wind.Direction)
            {
                case Direction.North:
                case Direction.East:
                case Direction.South:
                case Direction.West:
                default:
                    self.Matter = false;
                    piano.Play();
                    break;
            }
        }
    }
    #endregion
}
```

Miniprojet et manipulation de pixels — *rappel*

- La lecture d'un pixel via **`img[x, y]`** donne:
  - Un nombre entier si l'image est en niveaux de gris
  - Une liste de trois nombres si l'image est en RBG
- Dans l'écriture d'un pixel via **`img[x, y] = p`**, **`p`** doit être:
  - Un nombre entier si l'image est en niveaux de gris
  - Une liste de trois nombres si l'image est en RBG
- On écrit **`img[x, y]`** plutôt que **`img[x][y]`**
  - Plus efficace, fonctionne ainsi grâce à la bibliothèque numpy
  - Seulement pour les images du miniprojet, pas pour les listes de listes standard Python

Images RBG, images en niveaux de gris

- Attention, on ne peut pas traiter une image RGB comme une image en niveaux de gris
  - Avec numpy, on ne peut pas stocker un int représentant un niveau de gris dans un emplacement prévu pour une liste de trois valeurs
  - Vos fonctions **`apply_kernel`** et **`to_grayscale`** doivent créer de nouvelles images, pas modifier les données des images existantes passées en paramètres
- Si vous avez des soucis:
  - Mettez des breakpoints, décomposez vos calculs sur plusieurs lignes
  - Exécutez ligne par ligne; vérifiez à chaque ligne que ce que vous avez dans vos variables est ce que vous pensez
  - Testez avec des images générées de petites tailles (3×3, 5×5), pour être sûr que ça marche

➡ Démo plus tard

Cours de cette semaine

Algorithme pour trouver le seam de moindre énergie

Processus

niveaux de gris, lissage, Sobel

recherche d'un seam

Miniprojet — structure de base, rappel

```python
def seam_carving(img_path: str, num_cols: int) -> None:
    name, ext = split_name_ext(img_path)
    folder = name + os.path.sep
    os.makedirs(folder, exist_ok=True)

    img = load_image(img_path)

    img_grey = to_grayscale(img)
    save_image(img_grey, folder + "grey" + ext)

    img_grey = smoothen(img_grey)
    save_image(img_grey, folder + "smooth" + ext)

    img_grey = sobel(img_grey)
    save_image(img_grey, folder + "sobel" + ext)
    img_grey = np.uint8(img_grey)

    for i in range(num_cols):
        seam = find_seam(img_grey)

        img_highlight = highlight_seam(img, seam)
        save_image(img_highlight, folder + f"highlight_{i}" + ext)
        img_grey_highlight = highlight_seam(img_grey, seam)
        save_image(img_grey_highlight, folder + f"highlight_{i}_grey" + ext)

        img = remove_seam(img, seam)
        save_image(img, folder + f”step_{i}" + ext)
        img_grey = remove_seam(img_grey, seam)
```

Semaine passée

Boucle qui se répète autant de fois qu’on veut enlever de colonnes

On cherche un *seam* — notre préoccupation principale maintenant!

On enregistre des images intermédiaires où on montre où se trouve le *seam* détecté

On supprime le *seam* de l’image (et aussi de l’image en gris) et on enregistre le résultat, puis on recommence la boucle si nécessaire

Tâche: trouver un *seam*

Un pixel par ligne, on peut se déplacer d'une ligne à l'autre en bougeant d'un pixel à gauche ou à droite

129	137	25	147	44	64	192	247	11	81
143	164	152	220	1	155	242	152	205	116
189	16	53	115	248	102	157	234	76	188
122	79	48	186	96	78	32	230	170	1
106	123	17	78	8	27	123	145	231	70

Ne commence pas forcément par la plus petite valeur, ne finit pas forcément par la plus petite valeur…

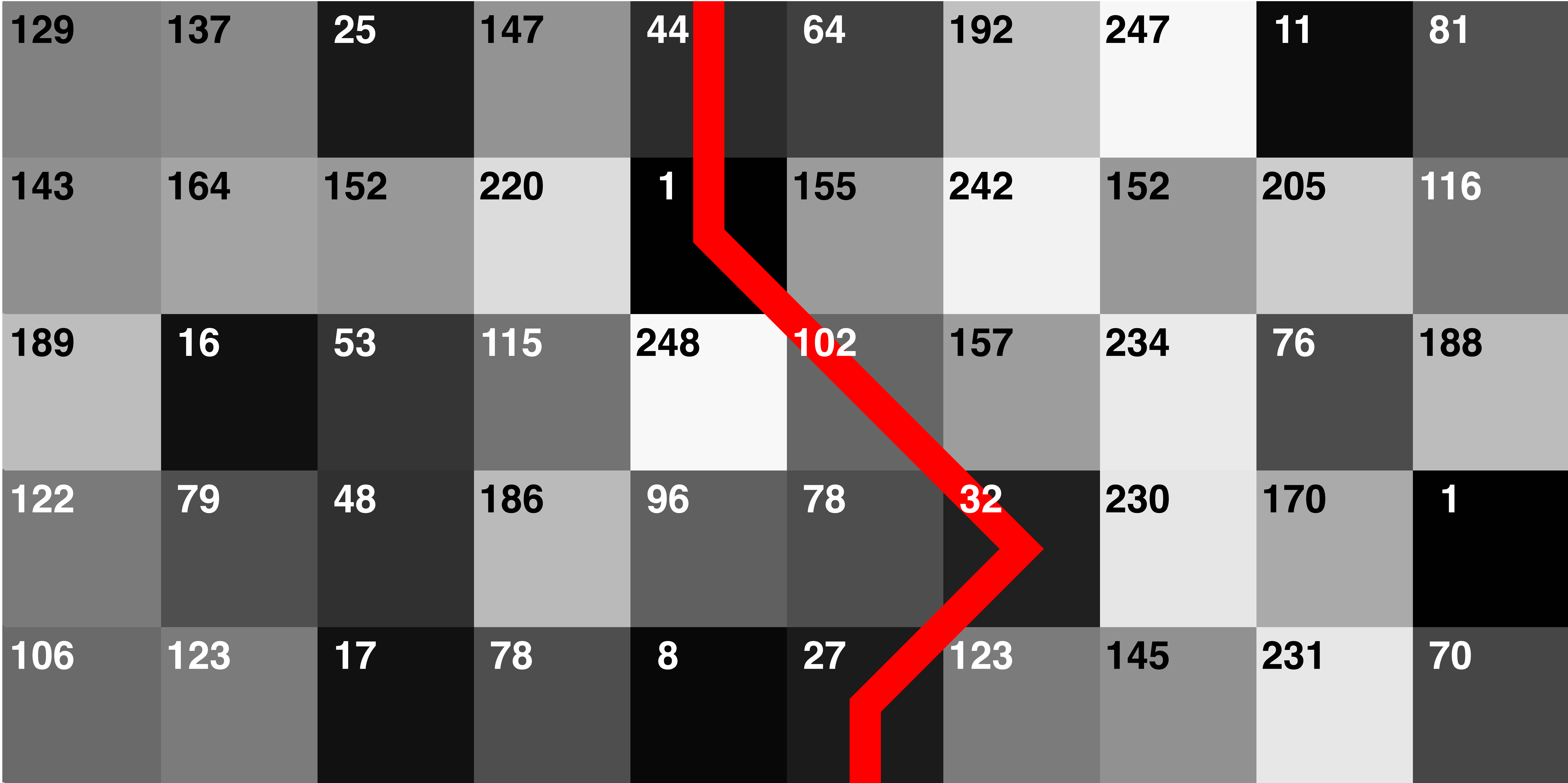
Comment trouver ce chemin?

Coût: 206

Algorithme: idée générale

- Variante de l'algorithme du plus court chemin de Dijkstra (1956)
- On commence par la ligne du haut et on considère tous les chemins de longueur 1 qui commence à chaque pixel. Leur coût est égal à la valeur du pixel en question.
- Pour chaque ligne suivante:
  - — Pour chaque pixel de la ligne, on cherche, parmi les 3 prédécesseurs possibles, quel serait le chemin de moindre coût pour arriver à ce pixel-ci, en additionnant la valeur du pixel aux coûts des chemins menant aux 3 prédécesseurs. On garde ensuite le meilleur.
    - ➡ On se rappelle non seulement le coût du meilleur chemin, mais également quel pixel du rang précédent nous a donné ce meilleur coût — sinon on ne saura pas revenir en arrière
- À la fin, on sélectionne le pixel d'arrivée qui a le meilleur coût total en regardant la dernière ligne, et on revient en arrière pour trouver le chemin complet

Algorithme: exemple



Implémentation: représentation de la grille de recherche

- Les valeurs de gris sont directement tirées des données de l'image
- Les coûts cumulés et les prédécesseurs pour chaque cellule doivent être gérés par vous séparément
- Base: une `List[List[PixelData]]` à créer

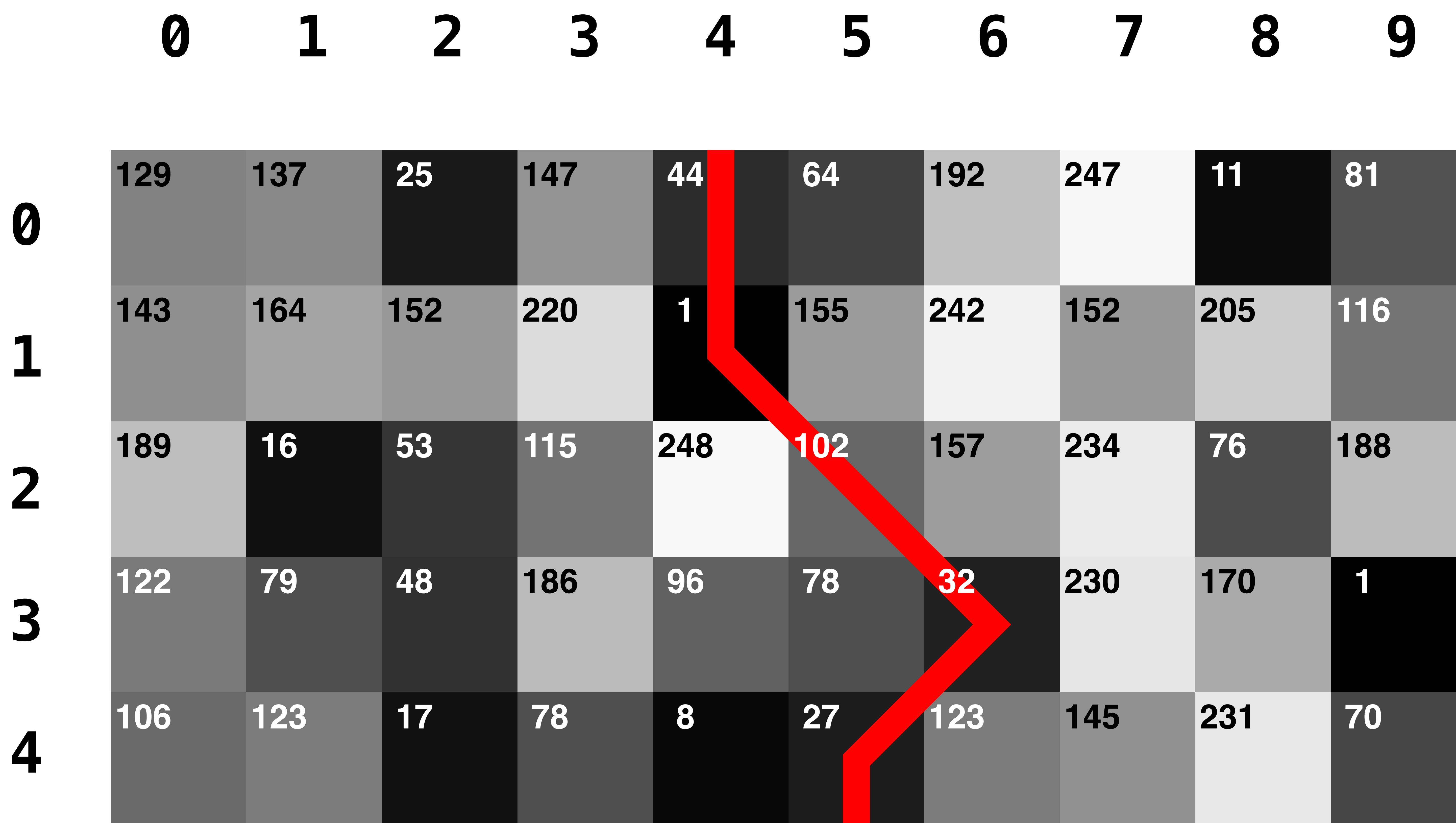
```
@dataclass
class PixelData:
    min_energy: int  = Infinity
    parent: Position  = (-1, -1)
```

129	137	25	147	44	64	192	247	11	81
143	164	152	220	1	155	242	152	205	116
189	16	53	115	248	102	157	234	76	188
122	79	48	186	96	78	32	230	170	1
106	123	17	78	8	27	123	145	231	70

129 129	137 137	25 25	147 147	44 44	64 64	192 192	247 247	11 11	81 81
[illegible] 272	164 189	152 177	220 245	1 45	155 199	242 306	152 163	205 216	116 127
189 378	16 193	53 230	115 160	248 293	102 147	157 320	234 397	76 203	188 315
122 315	79 272	48 208	186 346	96 243	78 225	32 179	230 433	170 373	1 204
106 378	123 331	17 225	78 286	8 233	27 206	123 302	145 324	231 435	70 274

Implémentation: représentation d'un *seam*

- Un *seam* est simplement une liste de ints: l'index de la colonne de chaque pixel, de haut en bas




```
seam = [4, 4, 5, 6, 5]
```

Démo

miniprojectutils.py: nouvelles fonctions

- `def save_image_greyscale_details(...)`
  - — Sauvegarde une image en niveaux de gris avec les valeurs écrites dessus (comme dans mes slides)
  - — On peut également passer un seam comme argument supplémentaire pour qu'il soit dessiné
- `def new_image_grey_with_data(data: List[List[int]]) -> Image`
  - — Permet de créer une petite image en niveaux de gris avec directement une matrice de valeurs de 0 à 255
- `def new_random_grey_image(height: int, width: int) -> Image`
  - — Crée une nouvelle image avec des niveaux de gris aléatoires et les dimensions données
    - ➡ Utile pour tester votre fonction sur de petites instances inspectables facilement

find_seam — structure de base

```
def find_seam(img_grey: Image) -> Seam:
    """Find the seam with the lowest energy."""
    print("   Finding seam...")

    # (a) création de la liste de liste de PixelData, un par pixel,
    #     qui contient le coût cumulé du chemin jusqu'à ce pixel
    #     dans le champ min_energy et les coordonnées du meilleur pixel
    #     précédent sur la ligne précédente dans le champ parent

    # (b) parcours de la première ligne pour définir le coût initial
    #     de chaque pixel comme simplement sa valeur de gris (ici, le champ
    #     parent ne voudra rien dire comme on est sur la première ligne)

    # (c) parcours de chacune des lignes suivantes et de chacun des pixels
    #     avec la recherche du meilleur parent sur la ligne d'avant et
    #     du coût cumulé

    # (d) recherche du pixel de la dernière ligne avec le coût cumulé
    #     le plus bas

    # (e) reconstruction du chemin complet à partir du pixel trouvé en (d)
    #     en remontant la liste de liste de PixelData chaque fois via les
    #     coordonnées du parent

    return ...
```

Résumé Cours 10

- Un *seam* (pour le miniprojet) est un chemin du haut de l'image vers le bas avec un pixel par ligne, qui peut se décaler d'un pixel à gauche ou à droite à chaque ligne
- Pour réduire l'image en largeur d'une unité, on va chercher le *seam* dont la somme des valeurs des pixels est la plus petite (qui représente donc le moins d'information)
- Pour le chercher, on applique un algorithme qui est un cas particulier de l'algorithme du plus court chemin de Dijkstra
- La recherche se fait en deux grandes phases: recherche progressive puis *backtracking*

- *Si vous avez encore des soucis avec l'installation de modules supplémentaires:*
  - *— Passez directement en CM 1 103 (presque en face d'ici) pour qu'on regarde ensemble*