PR-4207 Atelier génie logiciel : C++ et généricité

Thierry Géraud
theo@lrde.epita.fr

2015

1 Le projet en quelques mots

Le contexte du projet est le traitement d'images.

On dispose d'un algorithme qui prend en entrée une image binaire (en noir et blanc) et qui calcule une image dite "carte de distances". En chaque point de cette image, on lit la distance au plus proche point d'objet de l'image d'entrée (point à "vrai" ou "1"). Ci-dessous l'entrée à gauche et la sortie à droite.

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  0  0  0
0  0  0  0  0  0  0  0  1  0  1  0  1  1  1  1  0  0  0
0  0  0  0  0  0  0  1  0  1  1  1  1  1  1  0  0  0  0
0  0  0  0  0  0  0  1  0  0  1  1  1  1  1  0  0  0  0
0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  0  0  0  0
0  0  0  1  0  0  0  0  1  1  1  1  1  1  1  1  1  1  0
0  0  0  1  0  0  0  0  1  1  1  1  1  1  1  1  1  1  0
0  0  0  1  1  0  0  0  1  1  1  1  1  1  1  1  1  1  0
0  0  1  1  1  0  1  0  1  1  1  1  1  0  1  0  0  0  0
0  1  1  1  1  1  1  0  0  1  1  1  0  0  1  1  0  0  0
0  1  1  1  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0
0  0  1  1  1  0  0  0  0  1  0  0  0  0  0  1  0  0  0
0  0  0  1  0  1  0  0  0  1  1  1  0  0  1  0  0  0  0
0  0  0  0  0  0  1  1  0  1  0  1  1  1  1  1  1  1  0
0  0  1  1  1  0  1  1  1  1  0  1  1  1  1  1  1  1  0
0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1  1
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1
```

```
13 12 11 10 9  8  7  6  5  4  3  3  4  4  4  5  6  7
12 11 10 9  8  7  6  5  4  3  2  2  3  3  3  4  5  6
11 10 9  8  7  6  5  4  3  2  1  1  2  2  2  3  4  5
10 9  8  7  6  5  4  3  2  1  0  0  1  1  1  2  3  4
9  8  7  6  5  4  3  2  3  2  1  0  0  0  0  1  2  3
8  7  6  5  4  3  2  1  2  1  1  0  0  0  0  1  2  3
7  6  5  4  3  2  1  0  1  0  1  0  0  0  0  1  2  3
6  5  4  3  3  2  1  0  1  0  0  0  0  0  1  2  3  4
5  4  3  2  3  2  1  0  1  1  0  0  0  0  1  2  2  3
4  3  2  1  2  2  1  0  0  0  0  0  0  0  0  1  1  2
3  2  1  0  1  2  2  1  0  0  0  0  0  0  0  0  0  1
3  2  1  0  1  2  2  1  0  0  0  0  0  0  0  0  0  1
3  2  1  0  0  1  1  1  0  0  0  0  0  0  0  0  0  1
2  1  0  0  0  1  0  1  0  0  0  0  0  1  0  1  1  2
1  0  0  0  0  0  0  0  1  1  0  0  0  1  1  0  0  1  2
1  0  0  0  0  0  1  1  2  1  0  1  1  2  2  1  1  2  3
2  1  0  0  0  1  2  2  1  0  1  1  2  2  1  0  1  2
3  2  1  0  1  0  1  1  1  0  0  0  1  1  0  1  1  2
3  2  1  1  1  1  0  0  1  0  1  0  0  0  0  0  0  1
2  1  0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  1
1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0
```

Cet algorithme est très simple et, en fait, il est intrinsèquement générique. Le but du projet sera de voir qu'on peut modifier le comportement de cet algorithme sans toucher à son code. En particulier, cet algorithme nous servira à résoudre le problème de labyrinthe suivant : "étant donnés une image de labyrinthe, un point de départ et un point d'arrivée, trouver le chemin reliant ces deux points".

2 L'algorithme

Dans le pseudo-code ci-dessous,

— l'image d'entrée est `input`, une image binaire ;

— `D` est le domain de définition de l'image d'entrée (pour une image 2D "classique", c'est un rectangle, une boîte) ;

— la sortie est `dmap`, une image contenant des `unsigned` ;

— `max` est la valeur maximale des `unsigned` ;

— `p` et `n` sont des itérateurs sur des points (un point est un couple de coordonnées, 2 entiers donc) ;
— `N` est un voisinage ; par exemple, si $p = (2, 3)$, ses voisins forment l'ensemble $\mathcal{N}(p) = \{ (1, 3), (2, 2), (2, 4), (3, 3)\}$;
— `q` est une queue (conteneur *first in, first out*) de points.

```
D <- input.domain

for all p in D
,  dmap <- max

for all p in D
,  if input(p) = true
,  ,  dmap(p) <- 0
,  ,  for all n in N(p) and in D
,  ,  ,  if input(n) = false
,  ,  ,  ,  q.push(p)
,  ,  ,  ,  break

while q is not empty
,  p <- q.pop()
,  for all n in N(p)
,  ,  if (dmap(n) = max)
,  ,  ,  dmap(n) <- dmap(p) + 1
,  ,  ,  q.push(n)
```

Comprendre cet algorithme, le faire tourner à la main sur un exemple simple.

3 Les outils

On veut définir les *concepts* suivants :
— un point ; un domaine / ensemble de points ; un itérateur sur un domaine ;
— une image ;
— un itérateur sur un voisinage de point.

On veut les classes concrètes correspondantes pour le cas classique :
— point2d ; box2d ; box2d_iterator ;
— image2d< T > ;
— neighb2d_iterator.

4 Se rapprocher de l'objectif

Sans modifier l'algorithme, on veut :
— pouvoir calculer une carte de distance à une image d'étiquettes ;
— pouvoir calculer en même temps (que la carte de distances) une carte de la plus proche étiquette.