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

Thierry Géraud
theo@lrde.epita.fr

2015

Rappel

Dans le pseudo-code à droite,
— 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)
```

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.

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.

1 L'algorithme

Votre code doit ressembler *in fine* à ça :

```
using bool_t = int;

image2d<unsigned> compute_dmap__SPECIFIC(const image2d<bool_t>& input)
{
   box2d D = input.domain();

   const unsigned max = std::numeric_limits<unsigned>::max();
   image2d<unsigned> dmap(D);

   box2d_iterator p(D);
   for (p.start(); p.is_valid(); p.next())
      dmap(p) = max;

   std::queue<point2d> q;
   neighb2d_iterator n;

   for (p.start(); p.is_valid(); p.next())
      if (input(p) == true)
      {
         dmap(p) = 0;
         n.center_at(p);
         for (n.start(); n.is_valid(); n.next())
            if (D.has(n) and input(n) == false)
            {
               q.push(p);
               break;
            }
      }

   while (not q.empty())
   {
      point2d p = q.front();
      q.pop();
      n.center_at(p);
      for (n.start(); n.is_valid(); n.next())
         if (D.has(n) and dmap(n) == max)
         {
            dmap(n) = dmap(p) + 1;
            q.push(n);
         }
   }

   return dmap;
}
```

Comprenez et expliquez ce code dans le détail.

2 Les structures de données en code à *trous*

Elles doivent ressembler à ça :

```
class box2d // is a Domain type
{
public:
  using point_type = point2d;
  using p_iterator_type = box2d_iterator;
  // ...
};


class box2d_iterator
// iterator over the set of points
// contained in a 2D box
{
  // ...
};


class neighb2d_iterator
// iterator over the set of neighbors
// of a 2D point (the attribute p_)
{
public:
  neighb2d_iterator() // ctor
  {
    delta_.push_back(point2d(-1, 0));
    // ...
  }

  void center_at(const point2d& p); // change p_

  // as an iterator:
  void start();
  bool is_valid() const;
  void next();

  // to allow automatic coercion of objects from
  // this type to point2d:
  operator point2d() const {
    point2d n;
    n.row = p_.row + delta_[i_].row;
    n.col = p_.col + delta_[i_].col;
    return n;
  }

private:
  std::vector<point2d> delta_;
  unsigned i_; // current index in delta_
  point2d p_;  // center at p_
};
```

```
struct point2d
{
  // ...
};


template <typename T>
class image2d
{
public:
  using value_type = T;
  using domain_type = box2d;
  using point_type = typename domain_type::point_type;
  using p_iterator_type = typename domain_type::p_iterator_type;
  using n_iterator_type = typename domain_type::n_iterator_type;

  // 1) a generic type alias written in C++-11
  // that allows to change the value of the formal parameter:
  //
  //   template <typename U>
  //   using with_value_type = image2d<U>;
  //
  // example: image2d<int>::with_value_type<float> is image2d<float>
  //
  // 2) in older C++ standards, rely on:

  template <typename U>
  struct with_value_type {
    using ret = image2d<U>;
  };

  // example: with I being image2d<int>
  // typename I::template with_value_type<float>::ret is
  // image2d<float>

  // ctors:
  image2d(const domain_type& d);
  image2d(unsigned nrows, unsigned ncols);

  // access to pixel values:
  T& operator()(const point_type& p);
  T  operator()(const point_type& p) const;

  const domain_type& domain() const;
  // ...
private:
  domain_type d_;
  std::vector<T> data_;  // T cannot be bool :-(
};
```

3 Ce qu'il y a à faire + des questions

— Avoir un code complet (algorithme & structures de données & tests).
— Pourquoi le code de l'algorithme n'est-il pas générique ?
— Comment le rendre générique ?
— Quels sont les concepts associés aux types ?
— Rendre générique le code de l'algorithme.