Correction du DM1 : Plus proches voisins dans un nuage de points

1. (a) Pour obtenir la première composante d'un couple, on utilise la commande `fst`. Pour la deuxième composante d'un couple, il faut utiliser la commande `snd`.

   (b) Voici la fonction `distance` :

```
let distance m1 m2 = ((fst m2) -. (fst m1))**(2.0)
  +. ((snd m2) -. (snd m1))**(2.0) ;;
```

2. Pour avoir l'abscisse du deuxième point, on utilise la commande `fst (nuage.(1))`.
3. Voici la fonction `distance_i` :

```
let distance_i nuage i =
  let n = Array.length nuage and d = ref 0.
        and m = ref distance nuage.(i) nuage.(i+1) in
    for k = i+2 to (n-1) do
      d := distance nuage.(i) nuage.(k);
      if !d < !m then
        m := !d
    end;
    !m
;;
```

4. Voici la fonction `distance_mini` :

```
let distance_mini nuage =
  let n = Array.length nuage and d = ref 0.
        and m = ref distance_i nuage 0 in
    for k = 1 to (n-2) do
      d := distance_i nuage k;
      if !d < !m then
        m := !d
    end;
    sqrt !m
;;
```

5. Pour la fonction `distance_i`, on effectue $(n-1)-(i+1)+1 = n-i-1$ calculs de distance entre deux points.

   Pour la fonction `distance_mini`, on a $\sum_{i=0}^{n-2}(n-i-1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = O(n^2)$ calculs de distance entre deux points.

   La complexité de la fonction `distance_mini` est quadratique.
6. Voici la fonction `distance_mini_1D` :

```
let distance_mini_1D nuage =
  let n = Array.length nuage and d = ref 0.
        and m = ref distance nuage.(0) nuage.(1) in
    for k = 1 to (n-2) do
      d := distance nuage.(k) nuage.(k+1);
      if !d < !m then
        m := !d
    end;
  !m
;;
```

7. Dans la fonction `distance_mini_1D`, on effectue $(n-1)$ appels à la fonction `distance`. Donc `distance_mini_1D` est bien linéaire.

8. On commence par donner une fonction `fusion_array_abs` :

```
let fusion_array_h a l m u =
  let aux = Array.make (u-l+1) (0.0,0.0)
        and i = ref l and j = ref (m+1) and k = ref 0 in
    while (!i <= m) && (!j <= u) do
      if (a.(!i) < a.(!j)) then
        begin
          aux.(!k) <- a.(!i);
          i := !i + 1
        end
      else
        begin
          aux.(!k) <- a.(!j);
          j := !j + 1
        end;
      k := !k + 1
    done;
    if (!i <= m) then
      for r = (!i) to m do
        aux.(!k) <- a.(r);
        k := !k + 1
      done
    else
      for r = (!j) to u do
        aux.(!k) <- a.(r);
        k := !k + 1
      done;
    for r = 0 to u-l do
      a.(r+l) <- aux.(r)
    done;
;;
```

Remarque : la commande `a.(!i) < a.(!j)` compare les deux couples `a.(!i)` et `a.(!j)` selon l'ordre lexicographique.

On peut maintenant donner la fonction de tri fusion :

```
let tri_fusion_array_h a =
  let n = Array.length a in
    let rec tri_fusion_aux a l u =
      if l<u then
        let m = (u=l)/2 in
          tri_fusion_aux a l m;
          tri_fusion_aux a (m+1) u;
          fusion_array_h a l m u
    in tri_fusion_aux a 0 (n-1);
;;
```

9. Le tri fusion est de complexité quasi-linéaire $O(n \log_2(n))$.
10. On commence par définir une fonction qui échange les coordonnées d'un couple :

```
let swap (x,y) = (y,x) ;;
```

On adapte alors les fonctions pour de la question 8 :

```
let fusion_array_v a l m u =
  let aux = Array.make (u-l+1) (0.0,0.0) and i = ref l
        and j = ref (m+1) and k = ref 0 in
    while (!i <= m) && (!j <= u) do
      if ((swap a.(!i)) < (swap a.(!j))) then
        begin
          aux.(!k) <- a.(!i);
          i := !i + 1
        end
      else
        begin
          aux.(!k) <- a.(!j);
          j := !j + 1
        end;
      k := !k + 1
    done;
    if (!i <= m) then
      for r = !i to m do
        aux.(!k) <- a.(r);
        k := !k + 1
      done
    else
      for r = (!j) to u do
        aux.(!k) <- a.(r);
        k := !k + 1
      done;
    for r = 0 to u-l do
      a.(r+l) <- aux.(r)
    done;
;;
```

```
let tri_fusion_array_v a =
  let n = Array.length a in
    let rec tri_fusion_aux a l u =
      if l<u then
        let m = (u+l)/2 in
          tri_fusion_aux a l m;
          tri_fusion_aux a (m+1) u;
          fusion_array_v a l m u
    in tri_fusion_aux a 0 (n-1);
;;
```

11. Si n est pair, `nuage_g` et `nuage_d` contiennent exactement $n/2$ points.
    Si n est impair, `nuage_g` contient $(n-1)/2$ points et `nuage_d` contient $(n+1)/2$ points.
    Dans tous les cas, ce partitionnement découpe le nuage `nuage_h` en deux sous-nuages de taille approximativement `n/2`.
12. Voici la fonction `separation` :

```
let separation nuage_h nuage_v =
  let n = Array.length nuage_h in
    let m = n/2 in
      let med = nuage_h.(m) in
        let nuage_h_g = Array.sub nuage_h 0 m
              and nuage_h_d = Array.sub nuage_h m (n-m) in
          let nuage_v_g = Array.make m (0.0,0.0)
                and nuage_v_d = Array.make (n-m) (0.0,0.0)
                and i = ref 0 and j = ref 0 in
            for k = 0 to n-1 do
              if (nuage_v.(k) < med) then
                begin
                  nuage_v_g.(!i) <- nuage_v.(k);
                  i := !i + 1
                end
              else
                begin
                  nuage_v_d.(!j) <- nuage_v.(k);
                  j := !j + 1
                end;
            done;
            nuage_h_g , nuage_h_d , nuage_v_g , nuage_v_d
;;
```

    La partition de `nuage_h` est immédiate puisqu'il est trié. On obtient deux sous-nuages `nuage_h_g` et `nuage_h_d` qui sont triés.
    Le découpage de `nuage_v` demande plus de travail. Nous lisons tout le tableau `nuage_v` et nous répartissons les points en les comparant avec le point médian `med`. `i` est l'indice courant pour la partie gauche et `j` est l'indice courant pour la partie droite. Si le point courant `nuage_v.(k)` est plus petit que `med`, on le met dans le

sous-nuage de gauche et on augmente `i`. Sinon, on le met dans le sous-nuage de droite et on augmente `j`.

13. La valeur `delta` n'est pas forcément la valeur minimale attendue, il suffit de voir le dessein de l'énoncé pour avoir un contre-exemple.

14. Notons (x_1, y_1) les coordonnées de M_1 et (x_2, y_2) les coordonnées de M_2. Comme M_1 est situé dans `nuage_g`, $x_1 \leq x_{med}$. Comme M_2 est situé dans `nuage_d`, $x_2 \geq x_{med}$.
Si $x_1 < x_{med} - \delta$, $(x_2 - x_1) \geq (x_{med} - x_1) > \delta$.
Si $x_2 > x_{med} + \delta$, $(x_2 - x_1) \geq (x_2 - x_{med}) > \delta$.
Dans tous les cas, on obtient :

$$d(M_1, M_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \geq |x_2 - x_1| > \delta.$$

15. Voici la fonction `bande_du_plan` :

```
let bande_du_plan nuage_h nuage_v delta =
  let n = Array.length nuage_h and c = ref 0 and i = ref 0 in
    let xmed = fst (nuage_h.(n/2)) in
      while (fst (nuage_h.(!i)) < xmed -. delta) do
        i := !i + 1
      done;
      while (!i < n) && (fst (nuage_h.(!i)) <= xmed +. delta) do
        i := !i + 1
        c := !c + 1
      done;
      let b = Array.make !c (0.0,0.0) and j = ref 0 in
        for k = 0 to n-1 do
          if ((fst (nuage_v.(k)) <= xmed +. delta)
                && (fst (nuage_v.(k) >= xmed -. delta) then
            begin
              b.(!j) <- nuage_v.(k);
              j := !j + 1
            end
        done;
        b
;;
```

On commence par calculer la taille `c` du tableau `b` à partir du tableau `nuage_h` : on parcourt le tableau `nuage_h` et on teste si les abscisses des points sont entre $x_{med} - \delta$ et $x_{med} + \delta$.
On créé ensuite le tableau `b` qu'il reste à remplir à partir de `nuage_v` afin de garder les points rangés par ordonnée croissante. Pour cela, on parcourt `nuage_v` et si l'abscisse de `nuage_v.(k)` est entre $x_{med} - \delta$ et $x_{med} + \delta$, alors on le place dans `b.(j)`, `j` étant l'indice courant du tableau `b`.

16. Notons (x_1, y_1) les coordonnées de M_1 et (x_2, y_2) les coordonnées de M_2, avec $y_2 > y_1 + \delta$. Alors :

$$d(M_1, M_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \geq |y_2 - y_1| > \delta.$$

17. Supposons qu'il existe deux points M_1 et M_2 du nuage dans un carré de coté de longueur $\delta/2$ situé dans le demi-plan $x < x_{med}$. Alors $d(M_1, M_2) \leq \delta/\sqrt{2} < \delta$ (longueur de la diagonale du carré). Mais M_1 et M_2 sont deux points de `nuage_g` donc $d(M_1, M_2) \geq dg \geq \delta$. On aboutit à une contradiction.
    On peut faire le même raisonnement dans le demi-plan $x \geq x_{med}$.
18. La partie de la bande B au-dessus de M_1 avec une ordonnée inférieure ou égale à $y_1 + \delta$ peut être découpée en 8 carrés de coté de longueur $\delta/2$ (voir le dessein de l'énoncé). Un des carrés contient M_1 et chaque carré contient au plus un point du nuage. Il y a donc au plus 7 points du nuage dans cette partie autre que M_1.
19. Voici la fonction `minimum_bande` :

```
let minimum_bande b delta =
  let d_min = ref (delta**2.0) and p = Array.length b in
    for k = 0 to p-1 do
      let l = ref (k+1) in
        while (!l <= k+7) && (!l < p) do
          let d_temp = distance b.(k) b.(!l) in
            if d_temp < !d_min then d_min := d_temp;
            l := !l + 1
        done;
    done;
  sqrt (!d_min)
;;
```

    Nous prenons les points `b.(k)` de la bande `b` un par un (pour `k` allant de `0` à `p-1`). Nous calculons la distance entre `b.(k)` et chacun des 7 points de `b` au-dessus s'il y en a. Nous actualisons `delta` (représenté par la référence `d_min`) si nous trouvons une distance plus petite.
20. On commence par définir la fonction auxiliaire suivante :

```
let rec distance_mini_dpr_aux nuage_h nuage_v =
  let n = Array.length nuage_h in
    if (n < 4) then
      distance_mini_naif nuage_h
    else
      begin
        let nuage_h_g , nuage_h_d , nuage_v_g , nuage_v_d =
              separation nuage_h nuage_v in
          let dg = distance_mini_dpr_aux nuage_h_g nuage_v_g and
                dd = distance_mini_dpr_aux nuage_h_d nuage_v_d in
            let delta = if (dg <= dd) then dd else dg in
              let b = bande_du_plan nuage_h nuage_v delta in
                minimum_bande b delta
      end
;;
```

    Le cas de base, s'il y a 2 ou 3 points dans le nuage, se traite par l'algorithme naïf.
    Sinon, on sépare les nuages triés `nuage_h` et `nuage_v`. On calcule ensuite la distance minimale dans le nuage de gauche et dans le nuage de droite puis on définit `delta`.

On en déduit la distance minimale du nuage avec la fonction `minimum_bande b delta`.

Voici maintenant la fonction demandée :

```
let distance_mini_dpr nuage =
  let nuage_h = Array.copy nuage and nuage_v = Array.copy nuage in
    tri_fusion_array_h nuage_h;
    tri_fusion_array_v nuage_v;
    distance_mini_dpr_aux nuage_h nuage_v
;;
```

21. Avec les appels `tri_fusion_array_h nuage_h` et `tri_fusion_array_v nuage_v`, nous trions le nuage de points une seule fois dans les deux sens en complexité $O(n\log_2(n))$.
    Pour le reste :
    — L'étape de partitionnement, c'est-à-dire la fonction `separation nuage_h nuage_v`, est linéaire (on trouve la médiane immédiatement dans le premier tableau et on parcourt le deuxième en comparant l'abscisse de chaque point avec la médiane).
    — Une fois δ calculée, l'étape de sélection des points dans la bande B du plan, c'est-à-dire la fonction `bande_du_plan nuage_h nuage_v delta`, est linéaire (on parcourt les tableaux `nuage_h` et `nuage_v` en comparant les abscisses à $x_{med} - \delta$ et $x_{med} + \delta$).
    — L'étape de recherche dans la bande centrale, c'est-à-dire la fonction `minimum_bande b delta`, est linéaire en le nombre de point de la bande (puisqu'il s'agit de parcourir le tableau qui lui est associé et de faire sept calculs de distances).
    Au final, la complexité $C(n)$ dans le pire des cas pour un nuage de n points vérifie une équation de complexité de la forme :

$$C(n) = 2C(n/2) + O(n).$$

    Grâce au théorème de complexité, on obtient : $C(n) = O(n\log_2(n))$.
    Ainsi, la distance minimale entre deux points d'un nuage de n points est obtenue en $O(n\log_2(n))$ opérations.