Sylvain, Raphaël
(111 124 564)

Conception et analyse d'algorithmes
IFT-3001

Travail 2
Question 4

Travail présenté à
Yanick Ouellet

Département d'informatique et de génie logiciel
Univesité Laval
Hiver 2019

Sylvain, Raphaël
111 124 564

Question 4

Le problème est-il dans NP ?

Définition du certificat

Le certificat à la forme $((G_1, G_2), (V_1', V_2'))$ où :

- G_1 est un graphe $\langle V_1, E_1 \rangle$ où V_1 est un vecteur de sommets et où $v_i \in V_1$ ne sont pas numérotés.
- G_2 est un graphe $\langle V_2, E_2 \rangle$ où V_2 est un vecteur de sommets et où $v_i \in V_2$ ne sont pas numérotés.
- V_1' est un vecteur d'entier de longueur $|V_1|$.
- V_2' est un vecteur d'entier de longueur $|V_2|$.
- V_1' contient des valeurs entières différentes.
- V_2' contient des valeurs entières différentes.
- $|V_1'| \leq |V_2'|$
- Toutes les valeurs entières de V_1' sont aussi dans V_2'.
- Les indices d'une valeur i présente à la fois dans V_1' et V_2', représente les indices d'un noeud dans V_1 et un noeud dans V_2 ayant les mêmes arrêtes dans E_1 et E_2 respectivement.

Pseudo-code de vérification

```
Verification(g1, g2: graphe, vp1, vp2: vector<int>)
{
  // bloc A
  if vp1.count != g1.vertices.count
  {
    return false
  }

  if vp2.count != g2.vertices.count
  {
    return false
  }

  if vp1.count > vp2.count
  {
    return false
```

```
}

valeurG1 := new vector<int>
valeurG2 := new vector<int>
for i := 0 to vp1.count -1
{
  for(j = 0 to valeurG1.count - 1)
  {
    if valeurG1[j] == vp1[i]
    {
      return false /* 2 fois meme nombre dans g1*/
    }
  }
  valeurG1.add(vp1[i])
}
// Bloc B
for i := 0 to vp2.count -1
{
  for(j := 0 to valeurG1.count - 1)
  {
    if valeurG2[j] == vp2[i]
    {
      return false /* 2 fois meme nombre dans g2*/
    }
  }
  valeurG2.add(vp2[i])
}

vectorIntersect := new vector<int>
// Bloc C
for(i := 0 to valeurG1.count -1)
{
  for(j := 0 to valeurG2.count -1)
  {
    if (valeurG1[i] == valeurG2[j])
    {
      vectorIntersect.add(valeurG1[i])
    }
  }
}

if vectorIntersect.count != vp1.count)
{
  return false; /* Tous les element du plus petit graphe
                   ne sont pas present dans le plus gros */
```

```
}

// Bloc D
for (i := 0 to vp1.count -1)
{
  // Bloc D.A
  label := vp1[i];
  index2 := vp2.find(label)
  vertex1 := g1.vertices[i]
  vertex2 := g2.vertices[index2]
  // Bloc D.B
  otherVertices1 = new vecor<vertex>
  for (j := 0 to vp1.edges.count -1)
  {
    if vp1.edges[j].first == vertex1
    {
      otherVertices1.add(vp1.edges[j].second)
    }
    else if vp1.edges[j].second == vertex1
    {
      otherVertices1.add(vp1.edges[j].first)
    }
  }

  //Bloc D.C
  otherVertices2 = new vector<vertex>
  for (j := 0 to vp2.edges.count -1)
  {
    if vp2.edges[j].first == vertex1
    {
      otherVertices2.add(vp2.edges[j].second)
    }
    else if vp2.edges[j].second == vertex1
    {
      otherVertices2.add(vp2.edges[j].first)
    }
  }
  if (otherVertices1.count != otherVertices2.count)
  {
    return false;
  }

  intersectCounter := 0
  // Bloc D.D
  for(j := 0 to otherVertices1.count -1)
```

```
      {
        for(k := 0 to otherVertices2.count -1)
        {
          if (otherVertices1[k] == otherVertices2[k])
          {
            intersectCounter := intersectCounter + 1
          }
        }
      }

      if intersectCounter != otherVertices1.count
      {
        return false;
      }
    }

    return true;
  }
```

Analyse du pseudo-code de vérification

La fonction de vérification prend deux graphes (un vecteur de noeud et un vecteur de pair de noeud(arrêt)) et deux vecteurs d'entiers.

La taille de l'instance est donc déterminée par :

— n : la cardinalité du vecteur d'arrêt du premier graphe ;

— m : la cardinalité du vecteur d'arrêt du deuxième graphe

— x : la cardinalité du premier vecteur d'entier ;

— y : la cardinalité du deuxième vecteur d'entier.

Nous ne nous intéressons qu'à la borne maximale ($\mathcal{O}$) du comportement asymptotique du pire cas.

Nous devrons faire une analyse en plusieurs blocs.

Analyse du bloc A

Remarquons que toutes les opérations du bloc A ont une complexité constante ou constante amortie.

— L'ajout dans un vecteur se fait en temps constant amorti.

— L'accès à une valeur du vecteur se fait en temps constant.

— L'accès au nombre d'éléments du vecteur se fait en temps constant.

— La création d'un vecteur vide se fait en temps constant.

Remarquons aussi que si le vecteur du noeud du premier graphe ne contient pas le même nombre d'éléments que le nombre d'éléments dans le

premier vecteur d'entier, nous arrêtons le l'exécution de la fonction.

Nous pourrons donc prendre comme opération de base la comparaison `valeurG1[j] == vp1[i]` qui est l'opération exécutée le plus souvent.

La complexité peut être donnée par la sommation suivante :

$$\begin{aligned}
& C^{A}_{worst}(n, m, x, y) \\
= & \sum_{i=0}^{x-1} \sum_{j=0}^{x-1} 1 \\
= & \sum_{i=0}^{x-1} x \\
= & \; x \cdot \sum_{i=0}^{x-1} 1 \\
= & \; x \cdot x \\
= & \; x^2 \\
\in & \; \mathcal{O}(x^2)
\end{aligned}$$

Analyse du bloc B

Remarquons que toutes les opérations du bloc B ont une complexité constante ou constante amortie.

— L'ajout dans un vecteur se fait en temps constant amorti.
— L'accès à une valeur du vecteur se fait en temps constant.
— L'accès au nombre d'éléments du vecteur se fait en temps constant.
— La création d'un vecteur vide se fait en temps constant.

Remarquons aussi que si le vecteur du noeud du deuxième graphe ne contient pas le même nombre d'éléments que le nombre d'éléments dans le deuxième vecteur d'entier, nous arrêtons le l'exécution de la fonction.

Nous pourrons donc prendre comme opération de base la comparaison `valeurG2[j] == vp2[i]` qui est l'opération exécutée le plus souvent..

La complexité peut être donnée par la sommation suivante :

$$\begin{aligned}
& C^{B}_{worst}(n, m, x, y) \\
= & \sum_{i=0}^{y-1} \sum_{j=0}^{y-1} 1 \\
= & \sum_{i=0}^{y-1} y \\
= & \; y \cdot \sum_{i=0}^{y-1} 1
\end{aligned}$$

$$
\begin{aligned}
&= y \cdot y \\
&= y^2 \\
&\in \mathcal{O}(y^2)
\end{aligned}
$$

Analyse du bloc C

Remarquons que toutes les opérations du bloc C ont une complexité constante ou constante amortie.

— L'ajout dans un vecteur se fait en temps constant amorti.
— L'accès à une valeur du vecteur se fait en temps constant.
— L'accès au nombre d'éléments du vecteur se fait en temps constant.
— La création d'un vecteur vide se fait en temps constant.

Nous pourrons donc prendre comme opération de base la comparaison `valeurG1[i] == valeurG2[j]` qui est l'opération exécutée le plus souvent.

La complexité peut être donnée par la sommation suivante :

$$
\begin{aligned}
& C^{B}_{worst}(n, m, x, y) \\
&= \sum_{i=0}^{x-1} \sum_{j=0}^{y-1} 1 \\
&= \sum_{i=0}^{y-1} x \\
&= x \cdot \sum_{i=0}^{y-1} 1 \\
&= x \cdot y \\
&\in \mathcal{O}(x \cdot y)
\end{aligned}
$$

Analyse du sous-bloc D.A

Remarquons que toutes les opérations du sous-bloc D.A ont une complexité constante, sauf l'appel à `find`.

`find` à une complexité linéaire en longueur du vecteur, en pire cas.

La complexité peut donc être donnée comme suit :

$C^{D.A}_{worst}(n, m, x, y) = y \in \mathcal{O}(y)$

Analyse du sous-bloc D.B

Remarquons que toutes les opérations du sous-bloc D.B ont une complexité constante ou constante amortie.

— L'ajout dans un vecteur se fait en temps constant amorti.
— L'accès à une valeur du vecteur se fait en temps constant.
— L'accès au nombre d'éléments du vecteur se fait en temps constant.
— La création d'un vecteur vide se fait en temps constant.

Nous pourrons donc prendre comme opération de base la comparaison `vp1.edges[j].first == vertex1` qui est l'opération exécutée le plus souvent.

La complexité peut être donnée par la sommation suivante :

$$\begin{aligned} & C_{worst}^{D.B}(n, m, x, y) \\ = & \sum_{j=0}^{n-1} 1 \\ = & \ n \\ \in & \ \mathcal{O}(n) \end{aligned}$$

Analyse du sous-bloc D.C

Remarquons que toutes les opérations du sous-bloc D.C ont une complexité constante ou constante amortie.

— L'ajout dans un vecteur se fait en temps constant amorti.
— L'accès à une valeur du vecteur se fait en temps constant.
— L'accès au nombre d'éléments du vecteur se fait en temps constant.
— La création d'un vecteur vide se fait en temps constant.

Nous pourrons donc prendre comme opération de base la comparaison `vp2.edges[j].first == vertex1` qui est l'opération exécutée le plus souvent.

La complexité peut être donnée par la sommation suivante :

$$\begin{aligned} & C_{worst}^{D.C}(n, m, x, y) \\ = & \sum_{j=0}^{m-1} 1 \\ = & \ m \\ \in & \ \mathcal{O}(m) \end{aligned}$$

Analyse du sous-bloc D.D

Remarquons que toutes les opérations du sous-bloc D.D ont une complexité constante.

— L'accès à une valeur du vecteur se fait en temps constant.
— L'accès au nombre d'éléments du vecteur se fait en temps constant.

Nous pourrons donc prendre comme opération de base la comparaison `otherVertices1[i] == otherVertices2[j]` qui est l'opération exécutée le plus souvent.

La complexité peut être donnée par la sommation suivante :

$$\begin{aligned} & C^{D.D}_{worst}(n, m, x, y) \\ = & \sum_{j=0}^{n-1} \sum_{k=0}^{m-1} 1 \\ = & \sum_{j=0}^{n-1} m \\ = & \; m \cdot \sum_{j=0}^{n-1} 1 \\ = & \; m \cdot n \\ \in & \; \mathcal{O}(m \cdot n) \end{aligned}$$

Analyse du bloc D

Le bloc D fait appel a chacun des sous-Bloc D. ? de façon séquentielle.

En pire cas, il fera appel a chacun d'entre eux au complet.

Mon l'analyse asymptotique, nous pouvons donc prendre le sous-bloc qui a la complexité la plus élevée.

Dans notre cas, ce serait donc le sous-bloc D.D, car $\max y, n, m, m \cdot n = m \cdot m \cdot n$ si $m > 2^n > 2$.

On peut conclure que $m \cdot n >= y$ si $m > 2^n > 2$, car il y a au moins $nbNoeud - 1$ arrêt dans un graphe et y représente le nombre de noeuds d'un graphe.

La complexité du bloc D peut donc être donnée par la sommation suivante :

$$\begin{aligned} & C^{D}_{worst}(n, m, x, y) \\ = & \sum_{i=0}^{x-1} (m \cdot n) \\ = & \; (m \cdot n) \cdot \sum_{i=0}^{x-1} 1 \\ = & \; (m \cdot n) \cdot x \\ = & \; m \cdot n \cdot x \\ \in & \; \mathcal{O}(m \cdot n \cdot x) \end{aligned}$$

Conclusion de l'analyse

La fonction fait appel a chacun des blocs de façon séquentielle.
En pire cas, elle fera appel à chacun.
Nous aurons donc :

$$\begin{aligned}
& C_{worst}(n, m, x, y) \\
= \ & \max(x^2, y^2, x \cdot y, m \cdot n \cdot x) \\
\leq \ & \langle\ \forall n, m, x, y | x > 2 \wedge y > 2 \wedge n \geq x - 1 \wedge n \geq y - 1\ \rangle \\
& m \cdot n \cdot x \\
\in \ & \mathcal{O}(m \cdot n \cdot x)
\end{aligned}$$

Puisque $\mathcal{O}(m \cdot n \cdot x)$ est polynomial, alors la fonction de vérification appartient à P.

Analyse en pire cas de l'algorithme de réduction

La fonction de réduction prend un graphe et modifie deux autres graphes pour la sortie.
La taille de l'instance est par :
— n : le nombre de sommets du graphe entrant ;
— m : la cardinalité du vecteur d'arrêt du graphe entrant.
Nous ne nous intéressons qu'à la borne maximale ($\mathcal{O}$) du comportement asymptotique du pire cas.
Nous devrons faire une analyse en plusieurs blocs.

Bloc A

Le bloc A est composé de la boucle `for` et de la ligne qui suit (grapheG1).
Comptons le nombre de fois que `std::vector::emplace_back` est exécuté.
Nous pouvons prendre cette opération, car c'est elle qui a la plus grande complexité (constante amortie), en plus d'être appelé le plus souvent.
La complexité peut donc être donnée par la sommation suivante :

$$\begin{aligned}
& C^D_{worst}(n, m) \\
= \ & \sum_{i=0}^{n} 1 \\
= \ & n \\
\in \ & \Theta(n)
\end{aligned}$$

Bloc B

Le bloc A est composé de la copie dans le grapheG2.

La copie de la structure de graphe inclut une copie d'un entier et une copie d'un vecteur.

La copie de l'entier se fait en $\Theta(1)$.

Une copie d'un vecteur est linéaire sur le nombre d'éléments. Puisqu'il y a m éléments dans le vecteur d'arrêtes, la copie du vecteur aura une complexité de $\Theta(m)$.

La copie du graphe est donc en $C^B_{worst}(n,m)\,\Theta(m+1) = \Theta(m)$.

Conclusion de l'analyse de la réduction

La réduction d'exécute donc en :

$$\begin{aligned} & C^A_{worst}(n,m) \\ = \; & C^A_{worst}(n,m) + C^B_{worst}(n,m) \\ \in \; & \Theta(m+n) \end{aligned}$$

CycleHamiltonien = Oui ⇒ Sous-Graphe = Oui

Pour qu'un graphe contienne un cycle hamiltonien, il faut qu'il y ait un cycle reliant tous les noeuds du graphe en ne les visitant qu'une fois.

La représentation de ce chemin représente un graphe cyclique.

Donc la comparaison avec un graphe cyclique de la réduction retournera vraie, si ce graphe contient ce cycle hamiltonien.

CycleHamiltonien = Non ⇒ Sous-Graphe = Non

Pour qu'un graphe ne contienne pas un cycle hamiltonien, il faut qu'il soit impossible de relier tous les noeuds de ce graphe en ne les visitant qu'une fois.

La représentation d'un tel chemin serait un graphe cyclique.

Puisque ce cycle n'existe pas, la comparaison avec un graphe cyclique de la réduction retournera faux, si ce graphe ne contient pas de cycle hamiltonien.