Conception d'algorithmes et applications (LI325)

Cours 10: Algorithmes probabilistes

Ana Bušić

ana.busic@inria.fr

Chemin long dans un graphe

Problème : Soit $G = (V, E)$ un graphe non-orienté, k un entier. Trouver un chemin simple (passe au plus une fois par un nœud) de longueur k (si un tel chemin existe).

Note : Pour $k = n - 1$, trouver un cycle hamiltonien pour un graphe donné est un problème NP-complet.

Algorithme simple

L'idée principale : pour un DAG (graphe orienté acyclique), il existe un algoritme polinomial pour trouver le chemin (orienté) le plus long.

Algorithme simple

L'idée principale : pour un DAG (graphe orienté acyclique), il existe un algoritme polinomial pour trouver le chemin (orienté) le plus long.

Exercice : proposer un algorithme de programmation dynamique pour trouver le chemin le plus long dans un DAG qui à la complexité $O(|E|)$.

Algorithme simple

L'idée principale : pour un DAG (graphe orienté acyclique), il existe un algoritme polinomial pour trouver le chemin (orienté) le plus long.

Exercice : proposer un algorithme de programmation dynamique pour trouver le chemin le plus long dans un DAG qui à la complexité $O(|E|)$.

Soit G un graphe. On enumere les nœuds de G par l'ensemble $\{1, \ldots, n\}$ de manière aléatoire. Qu'est ce que "aléatoire" veut dire ici ?
Diriger les arrêtes vers les nœuds avec plus grand label. On obtient un DAG noté G'.

Algorithme simple

L'idée principale : pour un DAG (graphe orienté acyclique), il existe un algoritme polinomial pour trouver le chemin (orienté) le plus long.

Exercice : proposer un algorithme de programmation dynamique pour trouver le chemin le plus long dans un DAG qui à la complexité $O(|E|)$.

Soit G un graphe. On enumere les nœuds de G par l'ensemble $\{1, \ldots, n\}$ de manière aléatoire. Qu'est ce que "aléatoire" veut dire ici ?
Diriger les arrêtes vers les nœuds avec plus grand label. On obtient un DAG noté G'.

Lemme : Si G a un chemin P de longueur k, alors P est un chemin orienté dans G' avec la probabilité $\alpha = \frac{2}{(k+1)!}$.

Algorithme

Repeter $t = \frac{1}{\alpha}$ fois :

1. Choisir un G' aléatoire comme décrit.
2. Trouver le plus long chemin dans G'. On renvoie “faux” si la longueur est plus courte que k.

Algorithme

Repeter $t = \frac{1}{\alpha}$ fois :

1. Choisir un G' aléatoire comme décrit.
2. Trouver le plus long chemin dans G'. On renvoie "faux" si la longueur est plus courte que k.

Prop. La complexité de l'algorithme est $\frac{(k+1)!}{2} O(|E|)$.

Algorithme amélioré

Un graphe colorié est un graphe dans lequel chaque sommet possède une couleur. Un chemin pleinement colorié est un chemin dans lequel tous les sommets portent une couleur différente.

Exercice : proposer un algorithme de programmation dynamique pour trouver un chemin pleinement colorié de longueur k qui à la complexité $O(2^k k|E|)$.

Algorithme amélioré

Si P est un chemin du graphe de logueur k, alors

$$Pr[\text{P pleinement colorié}] \approx \frac{\sqrt{2\pi(k+1)}}{e^{k+1}} \qquad (\beta)$$

Algorithme amélioré

Si P est un chemin du graphe de logueur k, alors

$$Pr[\text{P pleinement colorié}] \approx \frac{\sqrt{2\pi(k+1)}}{e^{k+1}} \qquad (\beta)$$

L'algorithme amélioré :
Répéter t=$\frac{1}{\beta}$ fois :

1. Colorier le graphe G aléatoirement avec $k+1$ couleurs
2. Trouver le plus long chemin dans G (on échoue si aucun de ces chemin existe)

Algorithme amélioré

Si P est un chemin du graphe de logueur k, alors

$$Pr[\text{P pleinement colorié}] \approx \frac{\sqrt{2\pi(k+1)}}{e^{k+1}} \qquad (\beta)$$

L'algorithme amélioré :
Répéter t=$\frac{1}{\beta}$ fois :

1. Colorier le graphe G aléatoirement avec $k+1$ couleurs
2. Trouver le plus long chemin dans G (on échoue si aucun de ces chemin existe)

Chiffrement RSA

RSA (Rivest, Shamir et Adleman, 1977) :
système cryptographique à clé publique dont la sécurité est fondée sur la difficulté de la factorisation des grands entiers.

- Soit $n = pq$, où p et q sont deux nombres premiers impairs distincts. Soient a, b tels que $ab \equiv 1 \pmod{\phi(n)}$, où $\phi(n) = (p-1)(q-1)$. Notons $K = (n, p, q, a, b)$.
- Chiffrement :

$$e_K(x) = x^b \bmod n,\ x \in \mathbb{Z}_n.$$

- Dechiffrement :

$$d_K(y) = y^a \bmod n,\ y \in \mathbb{Z}_n.$$

- Les valeurs n et b forment la clé publique et les valeurs p, q et a la clé privée.

Exemple

- Supposons que Bob choisisse $p = 101$ et $q = 113$. On a $n = 11413$ et $\phi(n) = 100 \times 112 = 11200$.
- Soient $a = 6597$ et $b = 3533$. Alors :

$$ab = 23307201 = 2081 \times 11200 + 1,$$

  donc on a bien $ab \equiv 1 \pmod{\phi(n)}$.
- $K = (n, p, q, a, b) = (11413, 101, 113, 6597, 3533)$.

Exemple

- Supposons que Bob choisisse $p = 101$ et $q = 113$. On a $n = 11413$ et $\phi(n) = 100 \times 112 = 11200$.
- Soient $a = 6597$ et $b = 3533$. Alors :

$$ab = 23307201 = 2081 \times 11200 + 1,$$

  donc on a bien $ab \equiv 1 \pmod{\phi(n)}$.

- $K = (n, p, q, a, b) = (11413, 101, 113, 6597, 3533)$.
- Bob publie $n = 11413$ et $b = 3533$.

Exemple

- Supposons que Bob choisisse $p = 101$ et $q = 113$. On a $n = 11413$ et $\phi(n) = 100 \times 112 = 11200$.
- Soient $a = 6597$ et $b = 3533$. Alors :

$$ab = 23307201 = 2081 \times 11200 + 1,$$

  donc on a bien $ab \equiv 1 \pmod{\phi(n)}$.
- $K = (n, p, q, a, b) = (11413, 101, 113, 6597, 3533)$.
- Bob publie $n = 11413$ et $b = 3533$.
- Si Alice souhaite chiffrer le message 9726 pour l 'envoyer à Bob, elle calcule :

$$9726^{3533} \bmod 11413 = 5761$$

  et envoie le message chiffré 5761.

Exemple

- Supposons que Bob choisisse $p = 101$ et $q = 113$. On a $n = 11413$ et $\phi(n) = 100 \times 112 = 11200$.
- Soient $a = 6597$ et $b = 3533$. Alors :

$$ab = 23307201 = 2081 \times 11200 + 1,$$

  donc on a bien $ab \equiv 1 \pmod{\phi(n)}$.
- $K = (n, p, q, a, b) = (11413, 101, 113, 6597, 3533)$.
- Bob publie $n = 11413$ et $b = 3533$.
- Si Alice souhaite chiffrer le message 9726 pour l 'envoyer à Bob, elle calcule :

$$9726^{3533} \bmod 11413 = 5761$$

  et envoie le message chiffré 5761.
- Lorsque Bob reçoit 5761, il calcule : $5761^{6597} \bmod 11413 = 9726$.

Sécurité de RSA

Repose sur la clé privée a. Supposons que Martin intercepte le message chiffré 5761 :

Sécurité de RSA

Repose sur la clé privée a. Supposons que Martin intercepte le message chiffré 5761 :

- S'il connait a, il peut obtenir le message clair comme Bob :

$$5761^{6597} \mod 11413 = 9726.$$

Sécurité de RSA

Repose sur la clé privée a. Supposons que Martin intercepte le message chiffré 5761 :

- S'il connait a, il peut obtenir le message clair comme Bob :

$$5761^{6597} \mod 11413 = 9726.$$

- S'il connait $\phi(n)$, il peut calculer $a = b^{-1} \mod \phi(n)$.

  Calcul d'inverse par l'algorithme d'Euclide étendu en $O(k^2)$ où $k = \log(\max(a, b))$.

Sécurité de RSA

Repose sur la clé privée a. Supposons que Martin intercepte le message chiffré 5761 :

- S'il connait a, il peut obtenir le message clair comme Bob :

$$5761^{6597} \mod 11413 = 9726.$$

- S'il connait $\phi(n)$, il peut calculer $a = b^{-1} \mod \phi(n)$.

  Calcul d'inverse par l'algorithme d'Euclide étendu en $O(k^2)$ où $k = \log(\max(a, b))$.

- S'il connait p et q, alors $\phi(n) = (p-1)(q-1)$.

La sécurité de RSA est donc fondée sur la difficulté de la factorisation de n. Les nombres premiers p et q doivent être très grands (512 bits).

Test de primalité de Miller-Rabin

- Test de Miller-Rabin (1976) : un algorithme probabiliste du type Monte Carlo à erreur unilatérale $1/4$.
- Idée : l'algorithme cherche une preuve de non-primalité et s'il n'en trouve pas déclare le nombre premier.

Test de primalité de Miller-Rabin

- Test de Miller-Rabin (1976) : un algorithme probabiliste du type Monte Carlo à erreur unilatérale $1/4$.
- Idée : l'algorithme cherche une preuve de non-primalité et s'il n'en trouve pas déclare le nombre premier.
- Repose sur la propriété suivante :
  Soit p un nombre premier impair et $a \in \{1, ..., p-1\}$, on note r l'unique entier impair tel que $p - 1 = 2^s r$ alors :
  - soit $a^r = 1 \mod p$
  - soit $a^{2^j r} = -1 \mod p$ pour un entier j, $0 \leq j \leq s-1$.

Test de primalité de Miller-Rabin

- Test de Miller-Rabin (1976) : un algorithme probabiliste du type Monte Carlo à erreur unilatérale $1/4$.
- Idée : l'algorithme cherche une preuve de non-primalité et s'il n'en trouve pas déclare le nombre premier.
- Repose sur la propriété suivante :
  Soit p un nombre premier impair et $a \in \{1, ..., p-1\}$, on note r l'unique entier impair tel que $p - 1 = 2^s r$ alors :
  - soit $a^r = 1 \mod p$
  - soit $a^{2^j r} = -1 \mod p$ pour un entier j, $0 \leq j \leq s-1$.

  Preuve. Par le petit théorème de Fermat, $a^{p-1} = a^{2^s r} = 1 \mod p$. Donc, ou bien $\forall j$, $0 \leq j \leq s$, $a^{2^j r} = 1 \mod p$ et en particulier $a^r = 1 \mod p$. Ou bien il existe un j tel que $a^{2^j r} \neq 1 \mod p$, on prend alors le plus grand j possible. Ce choix de j implique que, $(a^{2^j r})^2 = a^{2^{j+1} r} = 1 \mod p$. Donc $a^{2^j r}$ est une racine carrée modulaire de 1, c'est à dire 1 ou -1. Donc, $a^{2^j r} = -1 \mod p$. □

Témoins forts de non-primalité

- Soient n impair et $a \in \{1, ..., n-1\}$. Alors $n-1 = 2^s r$, où r impair. On dit que a est un témoin fort de non-primalité pour n, si $a^r \neq 1 \mod n$ et $\forall j, 0 \leq j \leq s-1$, $a^{2^j r} \neq -1 \mod n$.
- Un nombre premier n'a donc pas de témoin fort de non-primalité.
- Propriété : Tout entier n, impair et non premier, a au moins $3(n-1)/4$ témoins forts de non-primalité.
- Le test de Miller-Rabin cherche des témoins de non-primalité forts : si n est un entier impair et non premier alors un entier $a \in \{1, \ldots, n-1\}$ choisi de manière uniforme a une probabilité d'au moins $3/4$ d'être un témoin fort de non-primalité.

Test de Miller-Rabin

Entrées : Un entier n impair.

Sorties : Composé ou (probablement) Premier

```
1  r := n − 1 ;  s := 0 ;
2  tant que r est pair faire
3  |  r := r/2 ; s := s + 1 ;
4  Tirer un entier a uniformément dans {1, ..., n − 1};
5  x := a^r  mod n;
6  si x = 1 ou x = n − 1 alors
7  |  retourner Premier
8  j := 1;
9  tant que j < s faire
10 |  x := x^2  mod n;
11 |  si x = n − 1 alors retourner Premier;
12 |  si x = 1 alors retourner Composé;
13 |  j := j + 1;
14 retourner Premier
```