Algorithmique avancée – Examen Réparti 2
UPMC — Master d'Informatique —
Janvier 2013 – durée 2h

Les seuls documents autorisés sont les polys de cours, ainsi que la copie double personnelle.
Le barême donné, sur 22 points, est indicatif.

1 Questions diverses [6 points]

Les questions suivantes sont indépendantes

Question 1. Donner un codage de compression optimal pour l'ensemble des fréquences suivantes, correspondant aux 8 premiers nombres de Fibonacci : `a:1; b:1, c:2; d:3; e:5; f:8; g:13; h:21`
Généraliser la réponse pour proposer un codage optimal lorsque les fréquences sont les n premiers nombres de Fibonacci ($F_1 = 1, F_2 = 1$, et pour $n \geq 2, F_n = F_{n-1} + F_{n-2}$).

Question 2. Soit T un texte contenant 256 caractères, et tel que la fréquence d'apparition est à peu près la même pour chaque caractère : la fréquence maximale vaut moins de deux fois la fréquence minimale. Montrer que dans ce cas, le codage de Huffmann n'est pas plus efficace qu'un codage de longueur fixe sur 8 bits.

Question 3. Compression de répétitions : Soit $T = (ab)^{1000}$ le texte constitué de la répétition de mille fois ab. Comparer la méthode de Huffman et la méthode LZW pour compresser ce texte.

2 Compression par anti-dictionnaire [8 points]

La méthode de compression par anti-dictionnaire utilise le fait que certains facteurs sont absents du texte à compresser. On travaille ici sur un alphabet binaire {0, 1}.
Soit un texte T, on note $F(T) = \{w \in \{0,1\}^*; \exists A, B \in \{0,1\}^* \text{ avec } T = AwB\}$ l'ensemble des facteurs de T. Un anti-dictionnaire de T, $AD(T)$ est un ensemble de facteurs qui ne sont pas dans T, c'est-à-dire un sous-ensemble (fini ou non) du complémentaire de $F(T)$.
Par exemple pour $T = 0101$, on peut avoir $AD(T) = \{00; 11\}$. Et pour $T = 0101010101$, l'ensemble des mots contenant deux lettres 1 à la suite est un anti-dictionnaire.

Question 1. Donner deux anti-dictionnaires différents pour $T = 100101$.
On suppose que l'anti-dictionnaire $AD(T)$ du texte T est fourni ; on compresse T dans un parcours de gauche à droite, en éliminant toutes les lettres pouvant être déduites des lettres précédentes à l'aide de $AD(T)$: en supposant que l'on a déjà compressé le préfixe v de T, s'il existe dans $AD(T)$ un mot $u = u'x$, $x \in \{0,1\}$, tel que u' est un suffixe de v, alors la lettre suivant v est nécessairement $\bar{x}$ (si x vaut 0 alors $\bar{x}$ vaut 1, et inversement). Cette lettre est inutile dans la compression du texte car elle peut être déduite grâce à l'anti-dictionnaire.

Question 2. Appliquer cette méthode à la compression du texte $T = 01101010$ avec l'anti-dictionnaire $AD(T) = \{00; 111; 1011\}$ (expliquer l'utilisation de l'anti-dictionnaire sur la suite des préfixes de T, pour aboutir finalement au texte compressé $TC = 01$).
Même question avec l'anti-dictionnaire $AD(T) = \{00; 0111\}$.

Question 3. On suppose que l'on dispose d'une primitive `estDans(w,E)`, de complexité $O(1)$, permettant de décider si un mot w appartient à l'anti-dictionnaire E. Décrire, en pseudo-code, un algorithme de compression suivant la méthode précedente, et étudier sa complexité.
Inversement, pour décompresser un texte TC à partir de l'anti-dictionnaire qui a été utilisé lors de la compression, on construit la suite des préfixes du texte initial T : à partir de la première lettre de TC, on retrouve, à l'aide de l'anti-dictionnaire, les lettres qui suivent dans T ; quand on ne peut plus deviner la lettre suivante de T, on examine la prochaine lettre dans le texte compressé TC et l'on applique le même processus ; et ainsi de suite jusqu'à épuisement des lettres de TC.

Question 4. Expliquer pourquoi dans l'algorithme de décompression il faut aussi fournir la *taille* du texte initial T. Appliquer l'algorithme pour décompresser $TC = 01$ à l'aide de $AD(T) = \{00; 111; 1011\}$.

Question 5. Décrire, en pseudo-code, un algorithme de décompression suivant cette méthode, et étudier sa complexité.

Question 6. Quelle structure de données proposeriez-vous pour représenter l'anti-dictionnaire ?

Question 7. Discuter du choix et de la construction de l'anti-dictionnaire.

3 Triangulations d'un polygone simple [8 points]

L'objectif de cet exercice est de découper un polygone simple en triangles.
Un polygone *simple* est un polygone tel que, pour tous côtés e et e' : si e et e' ne sont pas consécutifs alors e et e' n'ont aucun point commun. Une *triangulation* d'un polygone simple est une liste de triangles tels que : *(i)* les sommets des triangles sont des sommets du polygone, *(ii)* l'union de tous les triangles est égale au polygone, *(iii)* deux triangles distincts ont au plus un côté en commun.

Dans tout l'exercice, on suppose que l'on connaît un contour positif des polygones simples considérés. Dans les calculs de complexité, vous préciserez quelle(s) opération(s) significative(s) vous comptez.

Question 1. Donner un algorithme de triangulation d'un polygone **convexe**. Quelle complexité ?

Une *oreille* d'un polygone simple est un triangle pqr tel que p, q, r sont trois sommets consécutifs du contour et le triangle pqr est contenu dans le polygone.
Admis : tout polygone simple a au moins une oreille.
Admis : étant donnés trois sommets consécutifs p, q, r, le triangle pqr est une oreille ssi (p, q, r) est un tour gauche et le segment $[pr]$ n'intersecte que les côtés dont une extrémité est p ou r.

Question 2. Montrer qu'un polygone simple auquel on a coupé une oreille est encore un polygone simple. En déduire que tout polygone simple admet au moins une triangulation. Combien y-a-t-il de triangles dans une triangulation d'un polygone simple à n sommets ? (Preuves par récurrence.)

Question 3. On suppose qu'on dispose d'une fonction `intersecte?(a,b,c,d)` qui teste si les segments $[ab]$ et $[cd]$ ont au moins un point en commun et que cette fonction est de complexité constante. Donner un algorithme de calcul d'une oreille d'un polygone simple. Quelle complexité ?

Question 4. En déduire un algorithme de triangulation d'un polygone simple. Quelle complexité ?

Un polygone simple est une *montagne* s'il admet un contour positif $(p_0, \ldots, p_{n-1})$ tel que :
$abscisse(p_0) < abscisse(p_1)$ et les abscisses de p_1, p_2, ... , p_{n-1}, p_0 décroissent. Un sommet p_i est une *pointe* si $\boldsymbol{i > 1}$ et (p_{i-1}, p_i, p_{i+1}) est un tour gauche (les opérations se font modulo n).
Admis : toute montagne admet au moins une pointe.
Admis : si p_i est une pointe d'une montagne alors le triangle $p_{i-1}p_ip_{i+1}$ est contenu dans cette montagne.
Remarque : si p_i est une pointe alors $p_{i-1}p_ip_{i+1}$ est une oreille.

Question 5. Montrer qu'une montagne à laquelle on a enlevé une pointe est encore une montagne.

Question 6. Donner un algorithme de complexité linéaire de triangulation d'une montagne.

On appelle *montagne miroir* un polygone simple qui admet un contour positif $(p_0, \ldots, p_{n-1})$ tel que : $abscisse(p_0) < abscisse(p_{n-1})$ et les abscisses de p_0, p_1, ... , p_{n-1} croissent. Les montagnes miroirs peuvent elles aussi être triangulées en temps linéaire.

Un polygone simple est *monotone* s'il admet un contour positif $(p_0, \ldots, p_{n-1})$ et un indice k tels que les abscisses de p_0, p_1, ... , p_k croissent et les abscisses de p_k, p_{k+1}, ... , p_{n-1}, p_0 décroissent. Dans l'exemple donné en annexe $k = 4$.

Question 7. On suppose que $abscisse(p_{n-1}) < abscisse(p_1)$. Soit m le plus petit indice tel que $abscisse(p_m) < abscisse(p_1)$. Dans l'exemple donné en annexe $m = 7$.

1. Que peut-on dire du polygone $(p_0, p_1, p_m, p_{m+1}, \ldots, p_{n-1})$?
2. On admet que le polygone $(p_1, p_2, \ldots, p_{m-1}, p_m)$ est encore simple, montrer qu'il est monotone.

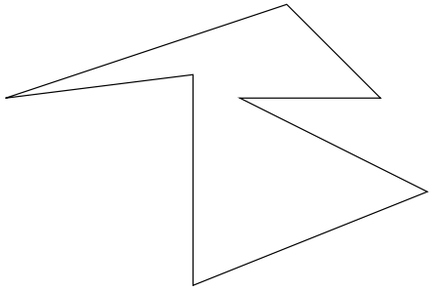
Le cas où $abscisse(p_{n-1}) > abscisse(p_1)$ est analogue.

Question 8. Que se passe-t-il lorsque $abscisse(p_{n-1}) = abscisse(p_1)$?

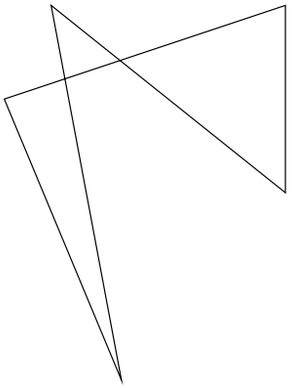
Question 9 En déduire un algorithme de triangulation d'un polygone monotone. Quelle complexité ?

Sujet de réflexion Si cela vous intéresse, vous pourrez travailler chez vous à la façon d'utiliser l'algorithme de triangulation d'un polygone monotone pour trianguler un polygone simple.

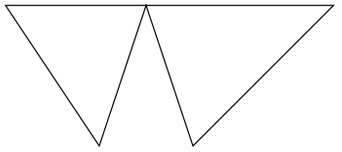
un polygone simple



un polygone non simple

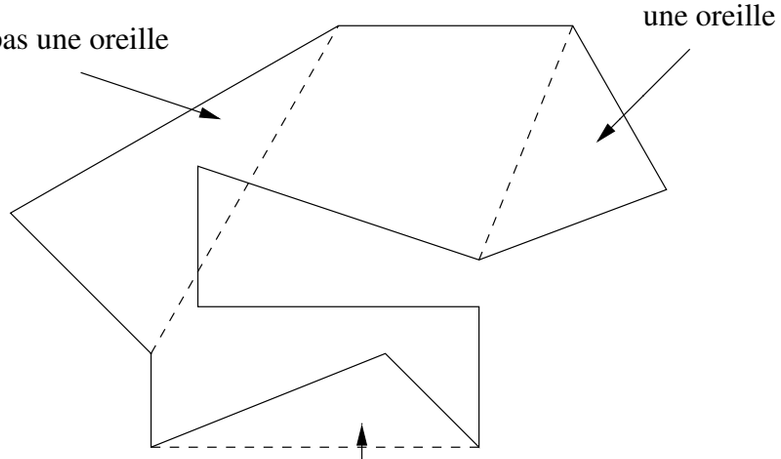


un polygone non simple



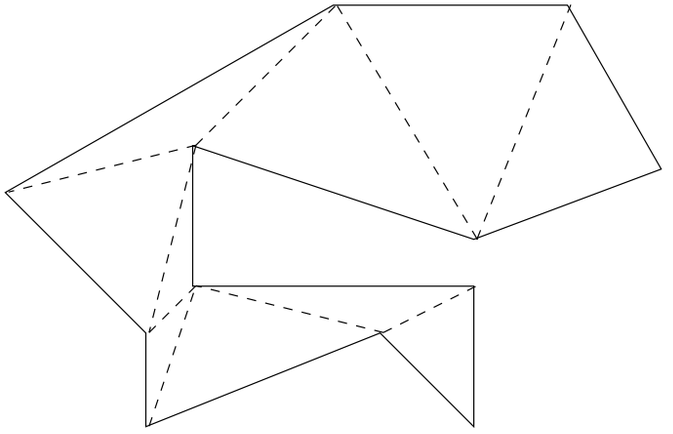
pas une oreille

une oreille

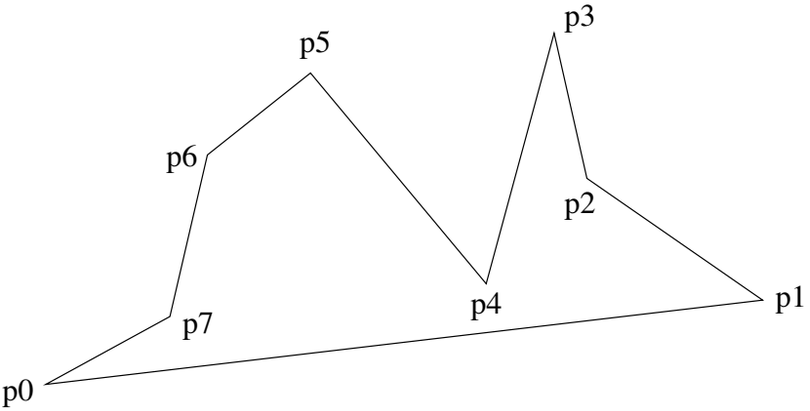


pas une oreille

une triangulation

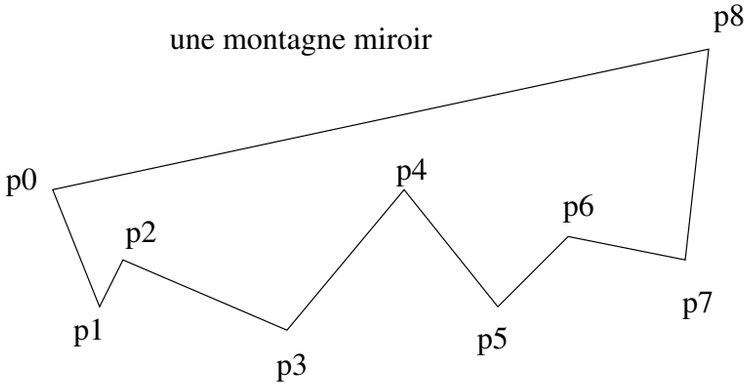


une montagne



p3, p5, p6 sont des pointes

une montagne miroir



un polygone monotone