DecryptoR

Programme multi-langues de décryptage de messages codés par une permutation de l'alphabet, à l'aide d'un ou plusieurs dictionnaires

Panorama du programme

Il permet de :

- Entrer un message crypté
- Entrer un dictionnaire (langue unique) / Entrer plusieurs dictionnaires (la fonction multi-langues s'active alors automatiquement)
- Attendre très peu de temps :-)
- Recevoir en réponse le message décrypté / la langue d'origine du message et le message décrypté

Détaillons l'utilisation :

- Rien à régler
- Une unique ligne de commande pour le lancer :

  java crypto dico.data message.txt {Programme numéro 0 ou 1}
- Ou si l'on veut plusieurs langues (= plusieurs dictionnaires) :

  java crypto -list N dico1.data ... dicoN.data message.txt [Programme numéro 0 ou 1]

Architecture du programme

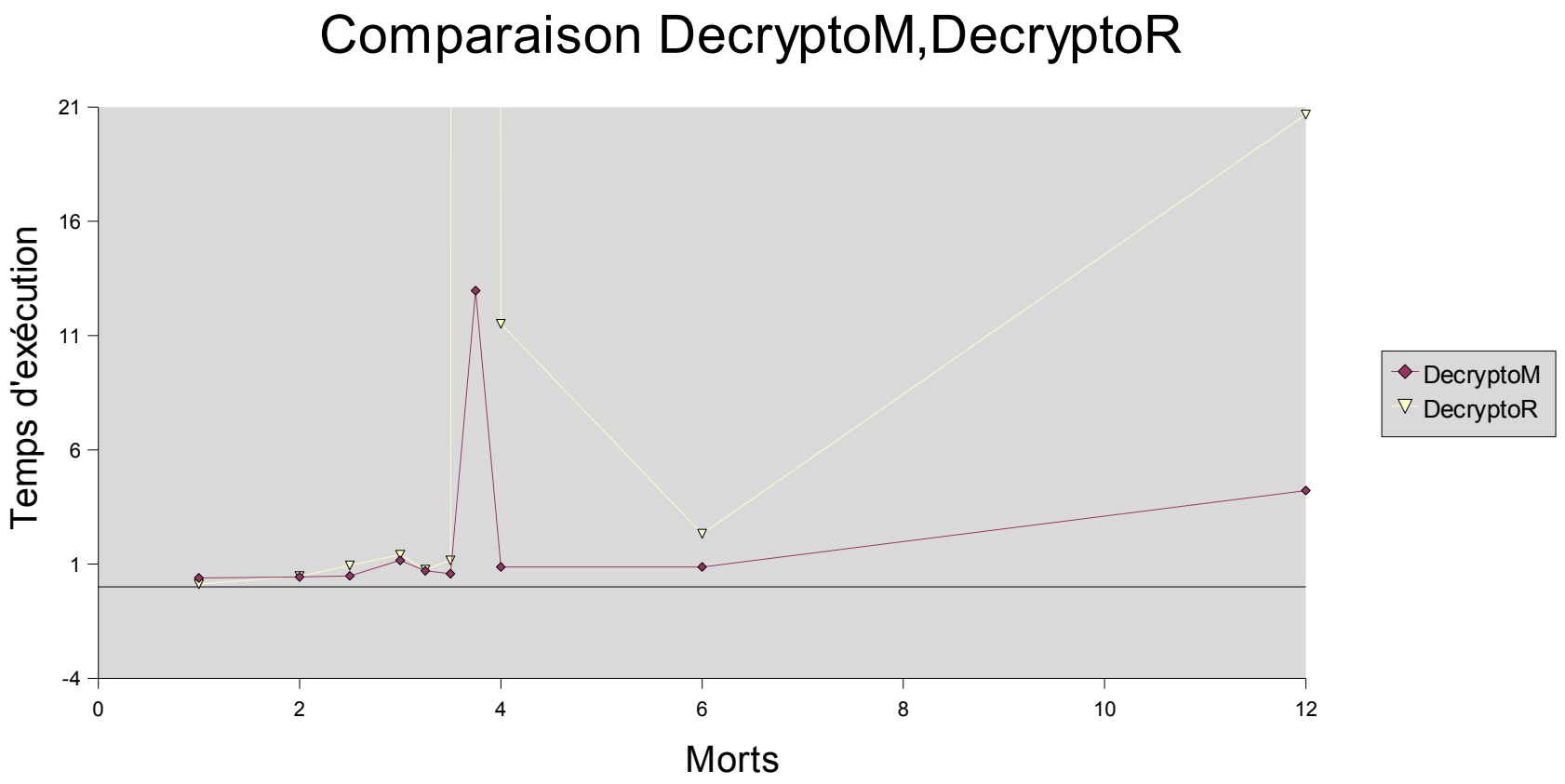
- Structures de données avec interfaces complètes
- Fonction principale (algorithme de résolution du problème) sous forme de classes
- Fonctions d'optimisation annexes activable ou non à volonté
- Interface utilisateur / fonctions de mise en forme

Optimisations et fonctions annexes

- Retrait des doublons
- Multi-threading dans les cas où on a plusieurs dictionnaires et où l'on cherche la langue
- La fonction de triage
- Ajout automatique des apostrophes
- Complétion d'une permutation incomplète
- Stockage optimisé des apparitions des lettres
- Affichage des solutions au fur et à mesure

Statistiques sur les messages cryptés

Temps de déchiffrage pour différents algorithmes



Statistiques sur les messages cryptés

Temps de déchiffrage pour différents algorithmes

Déchiffrage anglais vs. bilingue

	english	français & english
Chiffreanglais (DecryptoM)	0,03	0,06
Chiffreanglais (DecryptoR)	0,01	0,13

	english	français & english
Chiffretau2 (DecryptoM)	20,45	29,17
Chiffretau2 (DecryptoR)	709	au pire, le double

Preuves de correction et terminaison.

Propriétés générales, définitions:

On veut montrer la propriété suivante à la fin de la boucle:
F="A la sortie de la boucle, meilleurePermutation contient une permutation qui permet de décrypter le texte avec le moins de morts possibles."

- NSP (nombre de superpositions de permutations) ou ordre sur Sp = $n_{fusion()} - n_{retrait()}$

- NSP ou ordre sur MotDuMessage[i] = $n_{appliquerpermutation()} - n_{retirerPermutation()}$

- .decryptable()==true <=> il reste un mot dans la liste du mot qui n'a pas été pris en compte. Juste après retrait() ou appliquerPermutation(), decryptable() <=> nombreMotListe()!=0

-Triage2() ne modifie que l'ordre des mots entre curseur inclus et la fin du message.

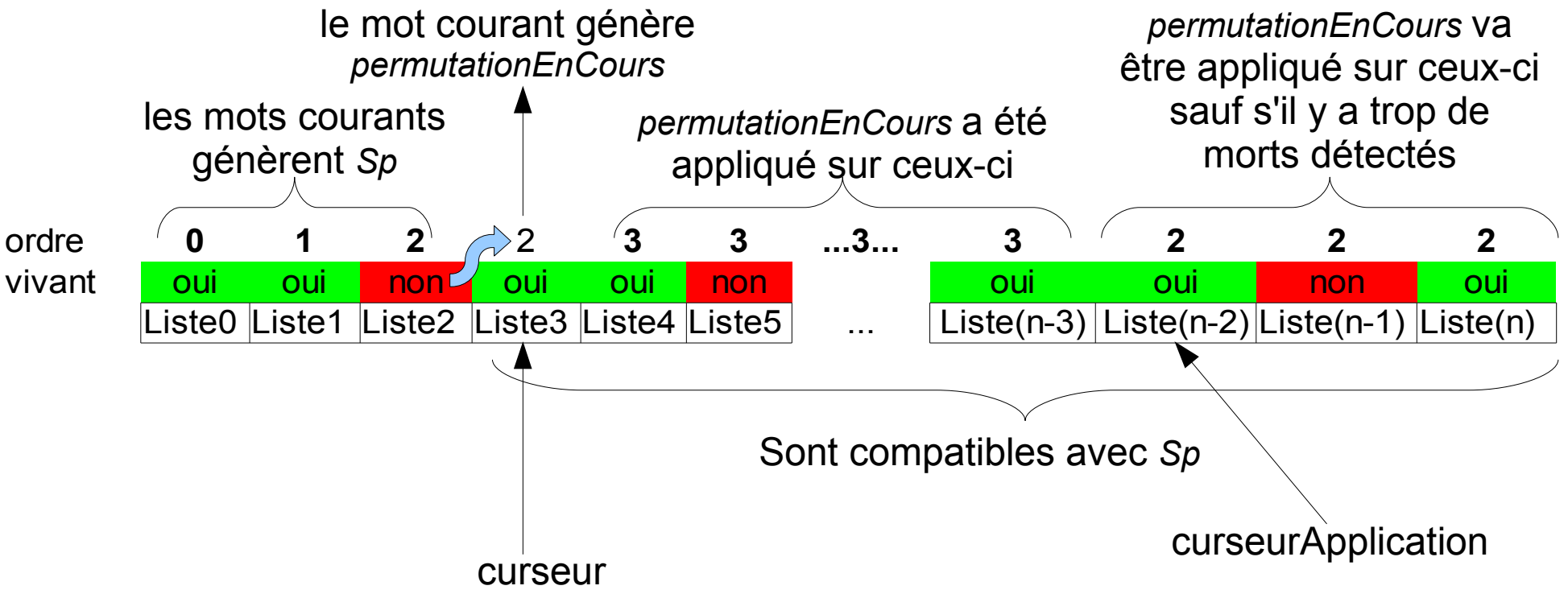
Preuve de la correction

Premiers invariants de boucle

- curseurApplication ≥ curseur
- Phrase[i].decryptable() && Nmorts≤NmortsMax => **!**Phrase[i].estMort()
- ordre(Phrase[i]) (tel que curseurApplication ≤ i) = curseur-MGS(curseur) (morts sur la gauche stricte du curseur)
- ordre(Phrase[i]) (tel que curseur<i<curseurApplication) = curseur-MGS(curseur)+1
- ordre(Phrase[i]) (tel que i≤Curseur) = i-MGS(i)
- (action==RecuperationPermutation || action==RetourArriere) => curseurApplication==curseur
- (action==RecupererPermutation) => quelque soit i>=curseur, m=MotDuMessage[i], m ne peut être à la fois vivant, non décryptable, avec une liste non vide de mots.
- (action==ApplicationPermutation || action==RetraitPermutationGauche) => MotDuMessage[curseur].estMort()==false
- (action==ChercheMeilleurGauche) => curseurApplication == Phrase.length && MotDuMessage[curseur] n'est pas mort.
- ordre(Sp) = (curseur-MGS(curseur))

Preuve de la correction

Image à avoir en tête, à tout moment:



Preuve de la correction

Propriétés générales, définitions secondaires:

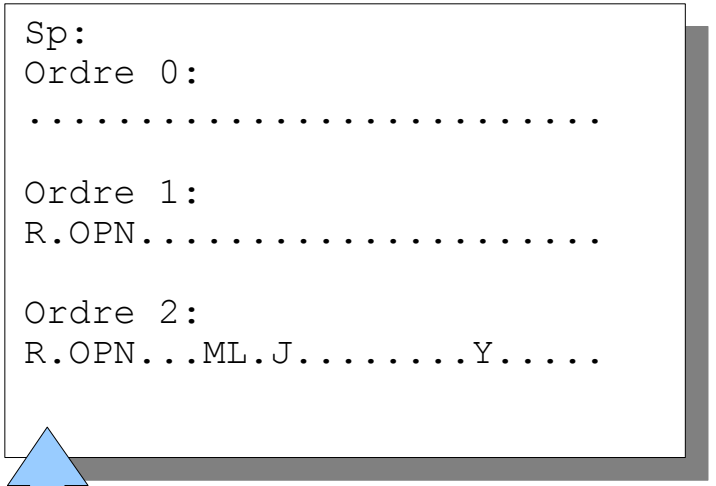
- Toutes les permutations totales compatibles avec une permutation partielle donnée font plus de morts que la permutation partielle seule (logique)

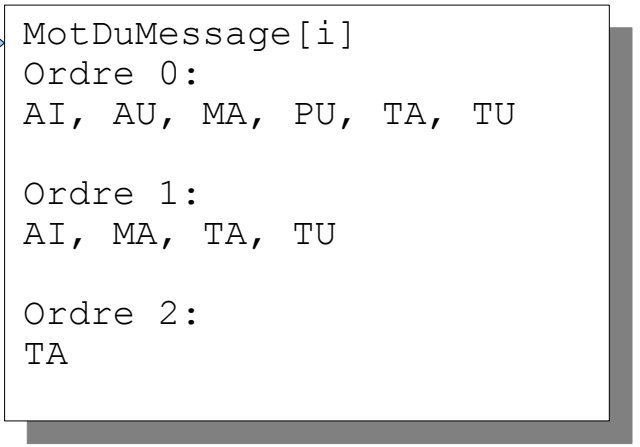
- On définit le mot courant d'un MotDuMessage m comme m.motCourant. Si le MotDuMessage est mort, le mot courant est dit vide.

- On peut définir la permutation d'ordre i à partir de Sp comme étant la i-ième permutation, puisque **Sp est similaire à une pile.**

- On peut définir la liste d'ordre j d'un MotDuMessage puisque les retraits et applications de permutations fonctionnent comme une **pile de listes.**

Au début, la liste est d'ordre 0 et la permutation de Sp est d'ordre 0 car vide.
Si 0≤i≤j≤ordre(Sp), alors la permutation de Sp d'ordre i se déduit de celle d'ordre j par retrait de lettres.





Preuve de la correction

Deuxièmes invariants de boucle

- la liste d'un MotDuMessage m à l'ordre j (j ≤ ordre(m)) est compatible avec toutes les permutations de Sp à l'ordre i≤j où i est inférieur ou égal à ordre(Sp).
Si i=j, alors la **liste contient TOUS les mots compatibles avec elle et Sp.**

- Les listes de mots curseur<i<curseurApplication sont compatibles avec la permutationEncours et Sp et contiennent TOUS les mots qui sont compatibles avec ces deux dernières permutations.

- (action==ApplicationPermutation||chercheMeilleurGauche et permutationCourante==null) => le motCourant de MotduMessage[curseur] est entièrement défini par Sp.

- (permutationEnCours !=null)=> le mot courant de MotDuMessage[curseur] est entièrement défini par permutationEnCours.

- Pour tout i<curseur, le mot courant du MotDuMessage[i] est compatible avec Sp à l'ordre j de manière totalement définie pour j≥(ordre(MotDuMessage[i])) ou est vide

- permutationsEnCours est compatible avec la Sp ou est null

Preuve de la correction

Soit P la meilleure permutation qu'on peut obtenir théoriquement.

H_j: « Il arrive un moment où les j premiers MotDuMessage 0, ... j-1 sont décodés comme si c'était avec P ou sont morts, et Sp est alors un préfixe de P, et curseur est à la position j ».

■ H_0: Vrai car Sp est nul au début, donc Sp est un préfixe de P.

■ Supposons qu'on ait H_j

Si MotDuMessage[j] est mort dans la condition optimale, on attend que la liste se termine pour passer au cran suivant, ce qui est possible, puisque P étant la meilleure **on ne plus revenir en arrière**.
Sinon a CEP de MotDuMessage[j] contient le mot décrypté par P pour la même raison. **On ne plus pas retourner en arrière** avant d'avoir fini le parcours de la liste, sinon ça impliquerait qu'une permutation meilleure que P a été trouvée.
Donc on arrive au mot décrypté par P, on applique sa permutation, **cela ne nous fait pas repartir en arrière** par le même argument, on augmente curseur, on fusionne la permutation obtenue à Sp, qui reste alors un préfixe de P.

Donc on a H_{j+1}

Preuves de la terminaison

La phrase est de taille n, le dico de taille m, un mot courant vide est par convention le (m+1)-ième de la liste. On définit un (n+1)-uplet numérotés de -1 à n-1 où chaque coordonnée $0 \leq i \leq n-1$ vaut un couple:

Le motcourant du MotDuMessage[i] étant le k-ème mot de la liste,
(k,
SI (curseur==i)&&(action==ApplicationPermutation||ChercheMeilleurGauche) ALORS curseurApplication-curseur+(chercheMeilleurGauche?1:0)
SINON SI (curseur==i)&&-action==RetraitPermutationGauche) ALORS 4*n-(curseurApplication-curseur)
SINON SI curseur==i ET action==RetourArriere ALORS 5*n
SINON SI curseur==i ET action==RecuperationPermutation && Phrase[curseur].estDecryptable() && Nmorts<=NmotsNonDecryptesMax ALORS 6*n
SINON SI i<curseur ALORS 2*n (*on a par défaut appliqué les permutations*)
SINON SI curseur<=i ALORS 0
)

et la coordonnée -1 vaut n-NmotsNonDecryptesMax

Toutes les coordonnées sont bornées de manière absolue et les uplets augmentent de manière lexicographique à chaque tour.

Exemple d'application

Soit la phrase:
```
servez ce whisky aux petits juges blonds qui fument
abcdbe fb ghiajk lmn obpipa qmrba stuvwa xmi ymzbvp
```
on la crypte.

L'algorithme la trie une première fois, les mots les plus importants d'abord.

```
ce petits fument whisky juges blonds servez aux qui
fb obpipa ymzbvp ghiajk qmrba stuvwa abcdbe lmn xmi
```

Et c'est parti!

Exemple d'application

curseur=0, pas encore rentré dans la boucle

`Sp = ..........................`

ce	*petits*	*fument*	*whisky*	*juges*	*blonds*	*servez*	*aux*	*qui*
fb	**obpipa**	**ymzbvp**	**ghiajk**	**qmrba**	**stuvwa**	**abcdbe**	**lmn**	**xmi**
ah	aceres	abetir	abetir	abces	abetir	aerien	ace	ace
ai	aleser	abimer	abimer	abime	abimer	aeriez	age	age
an	alibis	abimes	abimes	aboie	abimes	aigrie	agi	agi
as	amener	abimez	abimez	abois	abimez	aigrin	aie	aie
au	amenes	abject	abject	aboli	abject	aigris	aie	aie
ay	amenez	abjure	abjure	abord	abjure	aigrit	ail	ail
bd	ameres	ablier	ablier	about	ablier	andine	air	air
bu	amitie	abolie	abolie	aboye	abolie	andins	ais	ais
ca	aqueux	abolir	abolir	abris	abolir	angine	ait	ait
ce	arenes	abolis	abolis	abuse	abolis	angons	ale	ale
...	...	...	...	...	...	...	...	...

(n+1)-uplet de terminaison:
(0, (0,54),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0))

Exemple d'application

curseur=0, on est arrivé au traitement de « ce »

`Sp = .E...C.....................`

ce	*petits*	*fument*	*whisky*	*juges*	*blonds*	*servez*	*aux*	*qui*
fb	**obpipa**	**ymzbvp**	**ghiajk**	**qmrba**	**stuvwa**	**abcdbe**	**lmn**	**xmi**
ce	b**e**nins	abs**e**nt	abolir	adieu	abolir	aerien	agi	agi
	g**e**nant	adi**e**ux	abolis	admet	abolis	aeriez	ail	ail
	g**e**nons	adm**e**ts	abolit	adret	abolit	beames	air	air
	m**e**nant	adv**e**nu	abords	agres	abords	beates	ais	ais
	m**e**nons	agu**e**ts	abouti	ailes	abouti	begues	ait	ait
	p**e**rdra	aim**e**nt	abrupt	aimer	abrupt	beiges	ami	ami
	p**e**rira	air**e**nt	abruti	aimes	abruti	belges	and	and
	p**e**tits	alb**e**do	absolu	aimez	absolu	belier	ans	ans
	t**e**nons	amb**e**rt	abusif	aines	abusif	benies	api	api
	v**e**nant	ard**e**nt	adonis	aires	adonis	bequet	ars	ars
	...	...	...	...	...	...	...	...

(n+1)-uplet de terminaison:
(0, (9,10),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0))

Exemple d'application

curseur=1, on est arrivé au traitement de « petits »

`Sp = SE...C..I.....PT.........`

ce	*petits*	*fument*	*whisky*	*juges*	*blonds*	*servez*	*aux*	*qui*
fb	**obpipa**	**ymzbvp**	**gfajk**	**qmrba**	**stuvv**	**abcdbe**	**lmn**	**xmi**
ce	petits	amb**e**r**t**	briska	agres	abords	servez	and	agi
		ard**e**n**t**	frison	aloes	albums	sevrez	aux	ami
		arg**e**n**t**	grison	anges	amours	sexuel	bah	bai
		arm**e**n**t**	grisou	armes	argons		bal	foi
		auv**e**n**t**	maison	auges	armons		ban	fui
		bav**e**n**t**	raison	aunes	aurons		bar	gai
		bay**e**n**t**	whisky	baves	azygos		bau	goi
		box**e**n**t**		bayes	banjos		boa	gui
		bur**e**n**t**		blues	barons		bof	hai
		dam**e**n**t**		bores	bavons		bol	hui
		...		...	...		...	...

(n+1)-uplet de terminaison:
(0, (9,18),(7,9),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0))

Exemple d'application

curseur=2, « servez » remplace « fument »

`Sp = SERVZC..I.....PT.........`

ce	*petits*	*servez*	*whisky*	*juges*	*blonds*	*fument*	*aux*	*qui*
fb	**obpipa**	**abcdbe**	**ghiajk**	**qmrba**	**stuvwa**	**ymzbvp**	**lmn**	**xmi**
ce	petits	servez	maison	aloes	albums	bayent	and	agi
			whisky	anges	banjos	boxent	aux	ami
				auges	blonds	dament	bah	bai
				aunes	boudas	dolent	bal	foi
				bayes	boulas	dument	ban	fui
				blues	damons	fluent	bau	gai
				boues	flouas	fument	boa	goi
				boxes	fluons	gobent	bof	gui
				dames	fondas	halent	bol	hai
				domes	fondus	houent	bon	hui
				...	...	...	...	...

(n+1)-uplet de terminaison:
(0, (9,18),(7,18),(0,8),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0))

Exemple d'application

curseur=3, On arrive à « whisky »

`Sp = SERVZCWHIKY...PT.........`

ce	*petits*	*servez*	*whisky*	*juges*	*blonds*	*fument*	*aux*	*qui*
fb	**obpipa**	**abcdbe**	**ghiajk**	**qmrba**	**tuvw**	**ymzbvp**	**lmn**	**xmi**
ce	petits	servez	whisky	aloes	albums	boxent	and	agi
				anges	banjos	dament	aux	ami
				auges	blonds	dolent	bal	bai
				aunes	boudas	dument	ban	foi
				blues	boulas	fluent	bau	fui
				boues	damons	fument	boa	gai
				boxes	flouas	gobent	bof	goi
				dames	fluons	jouent	bol	gui
				domes	fondas	jugeat	bon	lai
				doues	fondus	jugent	box	loi
				...	...	...	...	...

(n+1)-uplet de terminaison:
(0, (9,18),(7,18),(0,18),(1,7),(0,0),(0,0),(0,0),(0,0),(0,0))

Exemple d'application

curseur=4, On arrive à «fument» mieux que « juges »

`Sp = SERVZCWHIKY.U.PT.....N..FM`

ce	*petits*	*servez*	*whisky*	*fument*	*blonds*	*juges*	*aux*	*qui*
fb	**obpipa**	**abcdbe**	**ghiajk**	**ymzbvp**	**stuvwa**	**qmrba**	**lmn**	**xmi**
ce	petits	servez	whisky	fument	blonds	auges	aux	gui
					glands	jubes	duo	lui
						juges	lux	oui
						jules	quo	qui
						luges		
						luxes		

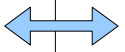
(n+1)-uplet de terminaison:
(0, (9,18),(7,18),(0,18),(1,18),(5,6),(0,0),(0,0),(0,0),(0,0))

Exemple d'application

curseur=5, On arrive à « blonds »

`Sp = SERVZCWHIKY.U.PT..BLOND.FM`

ce	*petits*	*servez*	*whisky*	*fument*	*blonds*	*juges*	*aux*	*qui*
fb	**obpipa**	**abcdbe**	**ghiajk**	**ymzbvp**	**stuvwa**	**qmrba**	**lmn**	**xmi**
ce	petits	servez	whisky	fument	blonds	auges	aux	gui
						juges		qui



(n+1)-uplet de terminaison:
(0, (9,18),(7,18),(0,18),(1,18),(5,18),(0,5),(0,0),(0,0),(0,0))

Exemple d'application

curseur=6, On arrive à « aux » mieux que « juges »

`Sp = SERVZCWHIKYAUXPT..BLOND.FM`

ce	*petits*	*servez*	*whisky*	*fument*	*blonds*	*aux*	*juges*	*qui*
fb	**obpipa**	**abcdbe**	**ghiajk**	**ymzbvp**	**stuvwa**	**lmn**	**qmrba**	**xmi**
ce	petits	servez	whisky	fument	blonds	aux	juges	gui
								qui

(n+1)-uplet de terminaison:
(0, (9,18),(7,18),(0,18),(1,18),(5,18),(0,18),(0,4),(0,0),(0,0))

Exemple d'application

curseur=7, On arrive à « juges »

`Sp = SERVZCWHIKYAUXPTJGBLOND.FM`

ce	*petits*	*servez*	*whisky*	*fument*	*blonds*	*aux*	*juges*	*qui*
fb	**obpipa**	**abcdbe**	**ghiajk**	**ymzbvp**	**stuvwa**	**lmn**	**qmrba**	**xmi**
ce	petits	servez	whisky	fument	blonds	aux	juges	qui

(n+1)-uplet de terminaison:
(0, (9,18),(7,18),(0,18),(1,18),(5,18),(0,18),(0,18),(0,3),(0,0))

Exemple d'application

curseur=8, On arrive à « qui »

`Sp = SERVZCWHIKYAUXPTJGBLONDQFM`

ce	*petits*	*servez*	*whisky*	*fument*	*blonds*	*aux*	*juges*	*qui*
fb	**obpipa**	**abcdbe**	**ghiajk**	**ymzbvp**	**stuvwa**	**lmn**	**qmrba**	**xmi**
ce	petits	servez	whisky	fument	blonds	aux	juges	qui

(n+1)-uplet de terminaison:
(0, (9,18),(7,18),(0,18),(1,18),(5,18),(0,18),(0,18),(0,18),(0,2))

Et pour finir, en avant première et en direct live...

Une démonstration de notre programme ; quelle chance !