Loria
Laboratoire lorrain de recherche en informatique et ses applications

Modélisation sensorielle
- Audition –
TD – HMM discrets

Denis Jouvet
LORIA – INRIA - Nancy

Inria informatics mathematics
cnrs
UNIVERSITÉ DE LORRAINE

Indexation phonétique

- Indexation permet de retrouver un document contenant un mot ou une expression, et sa position temporelle
- Décodage grand vocabulaire
  - Permet un accès par les « mots »
  - Mais ne permet pas de retrouver des mots non présents dans le lexique
- Décodage phonétique
  - Permet de retrouver n'importe quel mot ou expression
  - Mais pénalisé par erreurs de décodage phonétique
- Rmq: d'autres approches existent

- Exemple d'extrait de décodage phonétique
  - … d @ g o S d R w a t @ w a m a r s E j p a r i d i k y l @ a v w a r …

  - Marseille

➔ Comparaison d'une suite de phonèmes (provenant du décodage) à une suite de phonèmes de référence (prononciation d'un mot)

Comparaison de suites de phonèmes distance de Levenshtein (ou distance d'édition)

⇔ Nombre minimal de symboles (ici phonèmes) qu'il faut supprimer, insérer ou remplacer pour passer d'une suite à l'autre

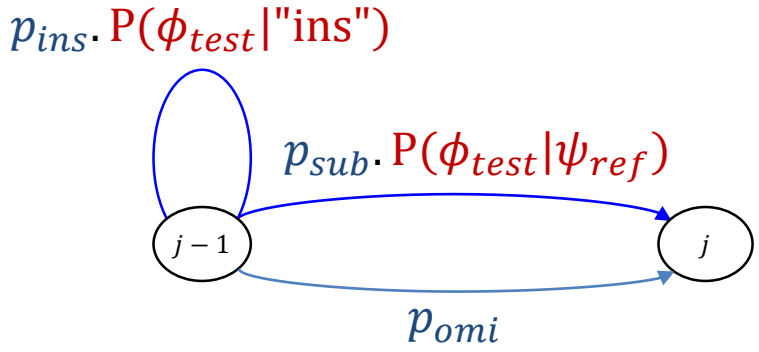
- Coûts arbitraires pour substitutions, insertions et omissions de symboles
  - $C_{sub} = \begin{cases} 1 & \text{si } \phi_{test} \neq \psi_{ref} \quad (\text{i.e., erreur}) \\ 0 & \text{si } \phi_{test} = \psi_{ref} \quad (\text{i.e., correct}) \end{cases}$
  - $C_{ins} = 1$ si insertion d'un symbole (phonème) ϕ_{test}
  - $C_{omi} = 1$ si omission d'un symbole (phonème) ψ_{ref}

- Exemples:

Ref:	m a R s E j	m a R s E j
Test:	m a s e j	m a R @ s E
Correspondances:	c c o c e c	c c c i c c o
Coûts:	0 0 1 0 1 0	0 0 0 1 0 0 1
➔ Coût total:	**2**	**2**

Comparaison de suites de phonèmes
Modèle de Markov discret

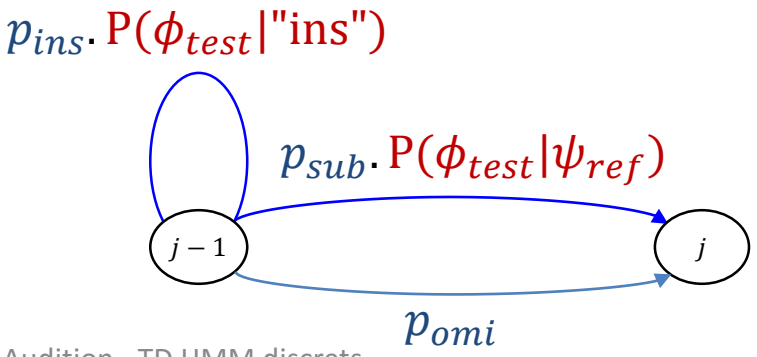
- Permet de tenir compte des confusions et erreurs entre phonèmes
  - Chaîne de Markov ➔ probabilités
    - p_{ins} d'avoir une insertion de symbole
    - p_{omi} d'avoir une omission de symbole
    - p_{sub} d'avoir une substitution de symbole (correcte ou pas)
  - Densités probabilité ➔ probabilité
    - $\mathrm{P}(\phi_{test}|\psi_{ref})$ d'observer phonème ϕ_{test} pour le phonème de référence ψ_{ref}
    - $\mathrm{P}(\phi_{test}|\text{"ins"})$ d'observer une insertion du phonème ϕ_{test}



- Optimisation des paramètres sur un corpus d'apprentissage

HMM discret vs. distance de Levenshtein

	Coûts distance de Levenshtein	HMM discret Coûts ⇔ -log(proba)
Substitution $\phi_{test} \neq \psi_{ref}$	$C_{sub} = 1$	$C_{sub} = -log(p_{sub}) - log(\mathrm{P}(\phi_{test}\|\psi_{ref}))$
Substitution $\phi_{test} = \psi_{ref}$	$C_{sub} = 0$	
Insertion ϕ_{test}	$C_{ins} = 1$	$C_{ins} = -log(p_{ins}) - log(\mathrm{P}(\phi_{test}\|\text{"ins"}))$
Omission ψ_{ref}	$C_{omi} = 1$	$C_{omi} = -log(p_{omi})$



Attention au log, si valeur égale à 0.

Estimation des paramètres du HMM discret

- Compteurs:
  - $N_{sub}, N_{ins}, N_{omi}$ ⇔ les nombres de substitutions, insertions et omissions
  - $N(\psi_{ref}, \phi_{test})$ ⇔ nombre de fois où le phonème ϕ_{test} est aligné avec ψ_{ref}.
  - $Ins(\phi_{test})$ ⇔ nombre de fois où le phonème ϕ_{test} est inséré

- Approximation des probabilités par les fréquences observées (ajout de « 1 » pour éviter valeurs nulles), soit par exemple:

$$p_{sub} = \frac{N_{sub} + 1}{N_{sub} + N_{ins} + N_{omi} + 3}$$

$$\mathrm{P}(\phi_{test}|\psi_{ref}) = \frac{N(\phi_{test}|\psi_{ref}) + 1}{\sum_i (N(\phi_i|\psi_{ref}) + 1)}$$

$$\mathrm{P}(\phi_{test}|\text{"ins"}) = \frac{Ins(\phi_{test}) + 1}{\sum_i (Ins(\phi_i) + 1)}$$

Objectifs du TD

- On se limite à l'aspect reconnaissance : i.e. trouver la séquence de référence du lexique qui ressemble le plus à la séquence de test (plus simple à mettre en œuvre que la détection de mots clés)
- Évaluer l'impact de la longueur des mots du lexique : 1, 2, 3 ou 4+ syllabes
- Évaluer l'impact de la taille du lexique : 100, 500 ou 1000 mots
- Découvrir de manière pratique l'apprentissage d'un HMM (et avec des corpus d'apprentissage de différentes tailles)
- Évaluer l'impact l'apport du HMM discret, en comparaison de la distance de Levenshtein

Données : lexiques de référence

- Lexiques de référence ⇔ fichiers lexicon-***<nbSyll>***-***<nbWords>***.lex où:
  - ***<nbSyll>*** ⇔ longueurs des mots : « 1syll », « 2syll », « 3 syll » et « 4+syll »
  - ***<nbWords>*** ⇔ taille du lexique : « 0100words », « 0500words », « 1000words »
  - Ex: lexicon-2syll-0100words.lex ⇔ lexique de 100 mots de 2 syllabes

- Format:

```
avec        a v E k
alors       a l o R
été         e t e
aussi       O s i
était       e t e t
était       e t E
était       e t e
était       e t E t
```

Mot — Suite de phonèmes (de référence)

Données : données de test

- Données de test ⇔ fichiers test-***<nbSyll>***-***<nbWords>***.test
  où:
  - ***<nbSyll>*** ⇔ longueurs des mots : « 1syll », « 2syll », « 3 syll » et « 4+syll »
  - ***<nbWords>*** ⇔ taille du lexique : « 0100words », « 0500words », « 1000words »
  - Ex: test-2syll-0100words.lex ⇔ données de test pour reconnaissance avec le lexique de 100 mots de 2 syllabes

- Format:

Mot (pour comptage des erreurs de reco)	Suite de phonèmes de test (décodage phonétique)
ici	j z i l
aussi	O s i
alors	a l o R
maintenant	m e~ t @ n a~
avez	E v i
compris	k o~ p R i
vraiment	v R e m a~

Données : données d'apprentissage

- Données d'apprentissage ⇔ fichiers train-***<nbItems>***.train
  où:
  - ***< nbItems >*** ⇔ nombre d'exemples : « 01000items », « 05000items », « 25000items »
  - Ex: train-01000items.train ⇔ données d'apprentissage comprenant 1000 exemples

- Format:

Mot (pour info)	Suite de phonèmes (de référence)	Suite de phonèmes de test (décodage phonétique)
allez	[a l e]	[l e]
cinq	[s e~]	[s e~]
quatre	[k a t R]	[k a t R]
trois	[t R w a]	[t R w]
un	[e~]	[e~]
et	[e]	[e]
zéro	[z e R o]	[z e R o]

Données : liste des phonèmes

- Le fichier « liste_symboles.dat » contient la liste des phonèmes (notation SAMPA) apparaissant dans les données de référence et de test
- Contient un phonème par ligne :

```
2
9
@
E
E
O
O
A
I
U
.....
```

Données : modèle HMM discret initialisé

- Fichier « modele_discret_initialise.dat », format csv

```
Psub;Pins;Pomi
0.800;0.100;0.100
#Une ligne par symbole de reference; une colonne par symbole de test
 ;2;9;@;e;E;o;O;a;i;u;y;a~;o~;e~;H;w;j;R;l;p;t;k;b;d;g;f;s;S;v;z;Z;m;n;J
2;0.967;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001
9;0.001;0.967;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001
@;0.001;0.001;0.967;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001
e;0.001;0.001;0.001;0.967;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.00
1;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001;0.001
……
Proba insertions...
<ins>;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;
0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;
0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029;0.029
```

Etapes du TD
1 – Reconnaissance avec distance de Levenshtein

Ecrire programme (ex. reco_dist_levenshtein <lex> <test>)

- Pour chaque suite de phonèmes du fichier test
  - Comparaison à toutes les suites de phonèmes du fichier lexique avec la distance de Levenshtein
  - La distance la plus faible identifie le mot reconnu
  - Comparer le mot du test avec le mot reconnu pour compter les erreurs
- Remarque
  - Fonction de comparaison retourne la **distance** et **l'alignement**
- Ecrire fichier log détaillé

  <mot test> [phones test] <mot reconnu> [phones ref] <OK/Err> <coût> <align>

  Par exemple:

```
ici   [j z i l] => aussi [O s i]   Erreur   3 <=> (O=>j) (s=>z) i (=>l)
aussi [O s i] => aussi [O s i]   Correct   0 <=> (O=>O) (s=>s) (i=>i)
```

Evaluer

- Pour un nombre de syllabes donné (2, 3 ou 4+) l'impact de la taille du lexique
- Pour une taille de lexique donnée (100, 500 ou 1000 mots) l'impact de la longueur des mots

Etapes du TD
2 – Reconnaissance avec HMM discret (initialisé)

Ecrire programme (ex. reco_HMM_discret <lex> <modeleHMM> <test>)

- Faire une copie du programme précédent (reco_dist_levenshtein)
- Ajouter du code pour lire les paramètres du modèle HMM discret (ici le HMM discret initialisé)
- Convertir les paramètres du modèle HMM discret en coûts (attention au log, si valeur nulle!)
- Modifier le code de programmation dynamique pour tenir compte des coûts associés au HMM discret (au lieu des coûts arbitraires précédents)

Evaluer

- Pour un nombre de syllabes donné (2, 3 ou 4+) l'impact de la taille du lexique
- Pour une taille de lexique donnée (100, 500 ou 1000 mots) l'impact de la longueur des mots

Etapes du TD
3 – Apprentissage du HMM discret

Ecrire un programme

(ex. apprentissage_HMM_discret <modl_init> <donnees_app> <modl_appris>)

- Pour chaque exemple des données d'apprentissage
  - Aligner la suite de phonèmes de test (décodage) avec la suite de référence
  - Incrémenter les compteurs correspondant aux paramètres du modèle
  - Mettre à jour le coût global (somme des coûts des alignements)
- Après traitement de l'ensemble des données
  - Estimer les probabilités du modèle à partir des compteurs
  - Ecrire les paramètres du modèle dans un fichier (utiliser le même format que celui du modèle initialisé, et même ordre des phonèmes)

- Effectuer 5 itérations d'apprentissage
  - model_initial ➔ model_iter1 ➔ … ➔ model_iter5

- Noter l'évolution des scores (coût global) au fil des itérations, et pour les différents ensembles d'apprentissage (1000, 5000 et 25000 items)

Etapes du TD
4 – Reconnaissance avec HMM discret (appris)

Utiliser le programme (ex. reco_HMM_discret <lex> <modeleHMM> <test>) de l'étape 2

- Si le modèle résultant de l'apprentissage a été écrit au même format que le modèle initialisé, il n'y a pas de modifications à faire dans le code

Evaluer, pour l'un des modèles appris

- Pour un nombre de syllabes donné (2, 3 ou 4+) l'impact de la taille du lexique
- Pour une taille de lexique donnée (100, 500 ou 1000 mots) l'impact de la longueur des mots

Evaluer pour un lexique donné (par exemple 500 mots de 1 ou 2 syllabes)

- L'évolution des performances en fonction du corpus d'apprentissage utilisé (1000, 5000 ou 25000 items).

Etapes du TD
5 - analyse des résultats

- Etudier les paramètres du modèle appris

  (rmq: format CSV, donc peut être ouvert sous excel pour faciliter la lecture)

  – Est-ce que les valeurs probabilités sont similaires à celles du modèle initialisé ?
  – Est-ce qu'il y a des confusions entre phonèmes plus fréquentes que d'autres ? Si oui, donner quelques exemples et les commenter.

Résultats à retourner

(éventuellement par mail, dans des fichiers zip)

- Par groupe
  - Les programmes écrits
  - Et les logs correspondants

- Individuellement
  - Une synthèse commentée des résultats (évaluations) obtenus (2 à 3 pages maximum)

Annexe

Comparaison de chaînes de caractères

- *Application à la reconnaissance d'épellations :*
  Recherche du mot du lexique le plus "similaire" à une séquence de lettres donnée (par exemple la suite des lettres reconnues)

- Exemple de "distances" entre chaînes de caractères :

- *Réf. :* P A R I S
  Test : B A R I F
  => distance = 2

- *Réf. :* P A R I S
  Test : P A I F
  => distance = 2

- Distorsions : *Omissions* ***(Une lettre manquante)***
  Insertions ***(Une lettre de plus)***
  Substitutions ***(Correct ou Erreur)***

Programmation dynamique

$$D(I,J) = D(Test, Ref)$$
$$= D(Test[1..I], Ref[1..J])$$

D(I,J) est la « distance » entre les I lettres du test et les J lettres de la référence

$$D(i,j) = D(Test[1..i], Ref[1..j])$$
$$= \text{Min}\begin{cases} D(Test[1..i-1], Ref[1..j-1]) + d(Test[i], Ref[j]) \\ D(Test[1..i-1], Ref[1..j]) + \textit{CoutInsertion}(Test[i]) \\ D(Test[1..i], Ref[1..j-1]) + \textit{CoutOmission}(Ref[j]) \end{cases}$$

$$d(Test[i], Ref[j]) \Leftrightarrow \textit{CoutSubstitution}.$$

Par exemple :
1 si lettres différentes
0 si lettres identiques

$$D("PAIF","PARIS") = \text{Min}\begin{cases} D("PAI","PARI") + d("F","S") \\ D("PAI","PARIS") + \textit{CoutInsertion}("F") \\ D("PAIF","PARI") + \textit{CoutOmission}("S") \end{cases}$$

Programmation dynamique

- Ne pas oublier de mémoriser le chemin optimal ➔ Path(i,j)

$$D(I,J) = D(Test,Ref)$$
$$= D(Test[1..I],Ref[1..J])$$

$$D(i,j) = D(Test[1..i],Ref[1..j])$$
$$= \text{Min}\begin{cases} D(Test[1..i-1],Ref[1..j-1]) + d(Test[i],Ref[j]) \\ D(Test[1..i-1],Ref[1..j]) + \mathit{CoutInsertion}(Test[i]) \\ D(Test[1..i],Ref[1..j-1]) + \mathit{CoutOmission}(Ref[j]) \end{cases}$$

Path(i,j) =

"Sub"

ou "Ins"

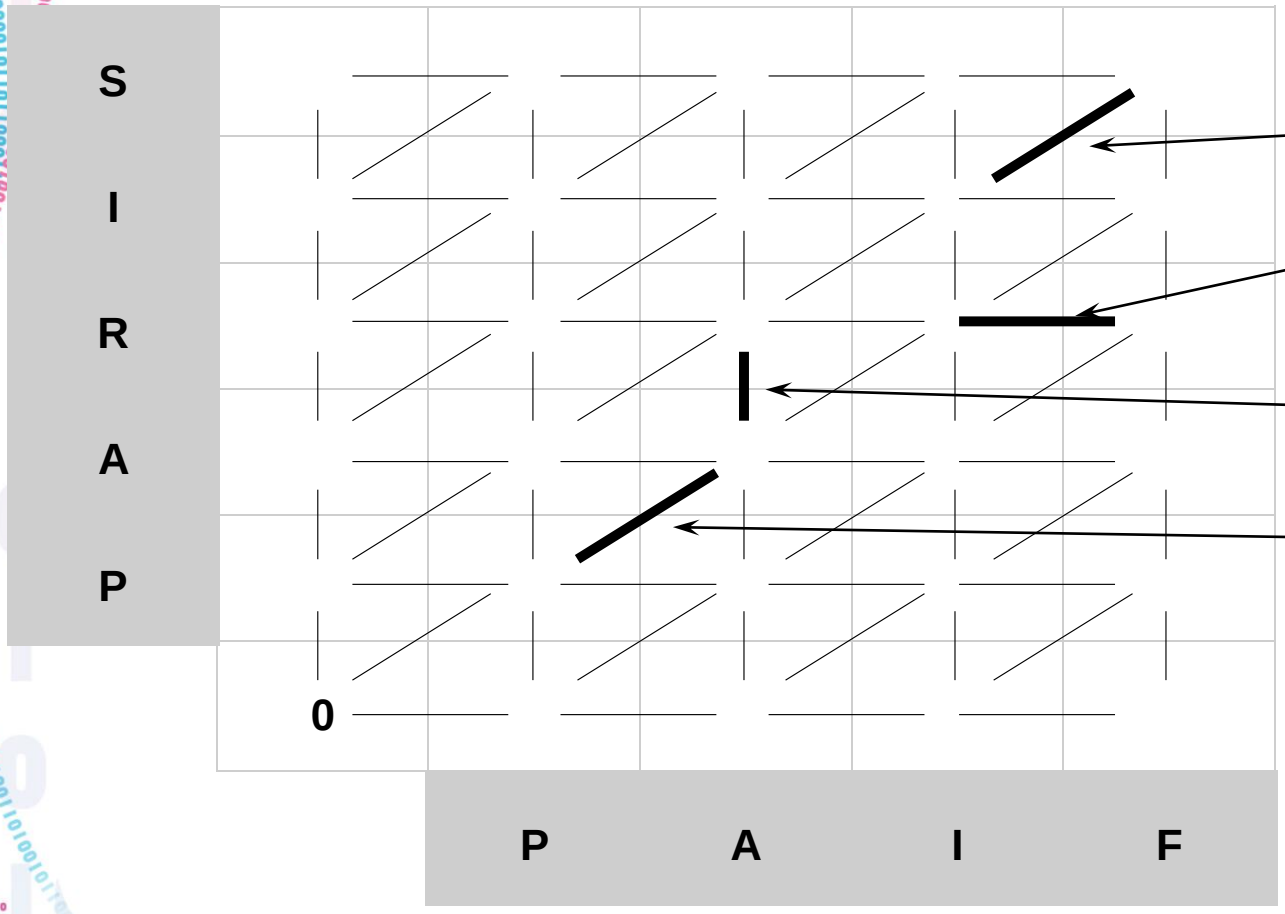
ou "Omi"

Programmation dynamique

Parcourir le chemin optimal

- Initialisation: i=I, j=J
- Boucle
  - Si Path(i,j) == "Sub"
    - Test[i] ⇔ Ref[j]
    - i=i-1; j=j-1
  - Si Path(i,j) == "Ins"
    - Test[i] inséré
    - i=i-1; j=j
  - Si Path(i,j) == "Omi"
    - Ref[j] omis
    - i=i; j=j-1
- Remettre dans l'ordre!

Programmation dynamique (exemple : coûts)



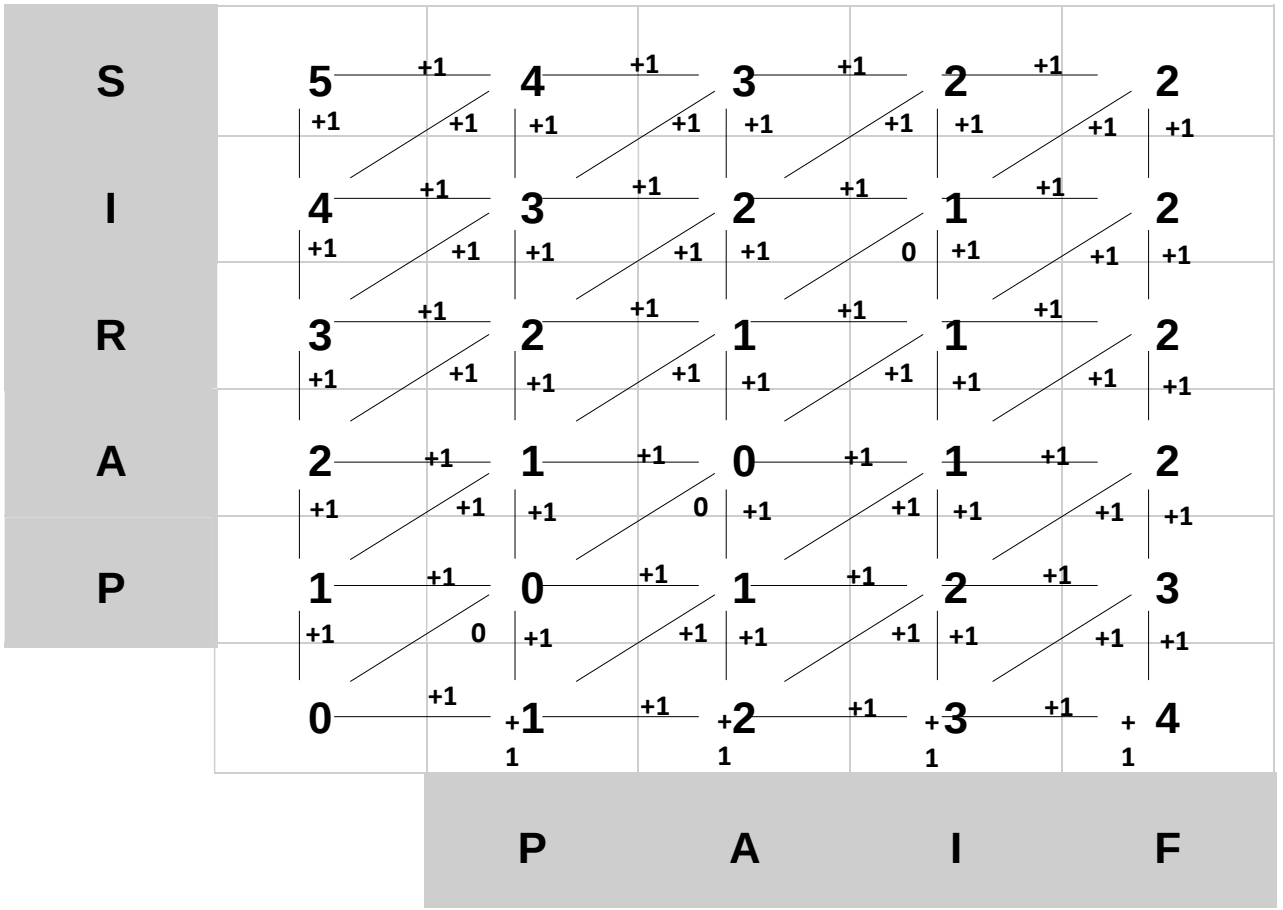
Substitution de "S" par "F"
donc coût de 1

Insertion lettre "F"
donc coût de 1

Omission de la lettre "R"
donc coût de 1

Substitution de "A" par "A"
donc coût de 0 (i.e. correct)

Programmation dynamique (exemple : coûts)



Programmation dynamique (exemple alignement)