Algorithmes gloutons

Lundi 09 Avril 2018

Michael FRANÇOIS

francois@esiea.fr

Introduction

• Un **algorithme glouton** (ou vorace) fait toujours le choix qui lui semble le meilleur sur le moment :

- Il fait un choix localement optimal dans l'espoir que ce choix mènerait à une solution globalement optimale.
- Autrement dit, l'algorithme glouton n'aboutit pas toujours à la solution optimale même s'il y arrive dans des nombreux cas.

• Les algorithmes gloutons sont très utilisés dans les problèmes d'optimisation discrète (*i.e.* optimisation combinatoire).

• Ils sont en général plus faciles à programmer que les algorithmes exacts et trouvent même quelques fois la solution exacte.

• Un algorithme glouton se contente donc de faire à chaque étape le meilleur choix local qui n'est pas forcément le meilleur choix global.

• D'une manière plus générale on peut dire qu'un algorithme glouton se contente d'une (très) bonne ≪tactique locale≫ sans chercher à avoir la meilleure stratégie globale.

• On peut même dire qu'un algorithme glouton n'a pas de stratégie, il n'a qu'une tactique.

Quelques problèmes classiques

Problème du rendu de monnaie

• C'est un problème courant de l'algorithmique.

• Lorsque vous passez à la caisse dans un magasin pour payer vos produits, il est fréquent que le caissier doive vous rendre la monnaie car, le montant que vous lui avez donné est supérieur à celui que vous devez payer.

• Supposons qu'on doit vous rendre la somme de 1.66 €. Il y a évidemment énormément de possibilités pour vous rendre la monnaie :

- 166 pièces de 1 centime d'euro (pas sûr que ça vous convienne) ;
- 83 pièces de 2 centimes d'euro ;
- 3 pièces de 50 cents, 1 pièce de 10 cents, 1 pièce de 5 cents et 1 pièce de 1 centime ;
- ...

• Le problème qui se pose ici est : comment minimiser le nombre de pièces à rendre pour un montant donné ?

Solution naïve :

• La solution la plus simple consiste à énumérer toutes les combinaisons possibles, et de sélectionner la meilleure à savoir celle qui utilise le minimum de pièces.

• Cette solution exhaustive, prenant en compte tous les cas de figure fonctionnera toujours. Elle peut être efficace dans certains cas, mais dès que l'ensemble de combinaisons est très élevé, elle est **à éviter**.

• Notre tâche est donc de formuler une solution plus efficace pour ce type de problème.

Méthode gloutonne :

• La méthode gloutonne va converger ici vers la solution optimale en effectuant localement le meilleur choix, et cela étape par étape.

• L'astuce consiste à répéter le choix de la pièce de plus grande valeur jusqu'à presque épuisement de la somme, puis de continuer avec la pièce de plus petite valeur, et ainsi de suite.

• Appliquons cette méthode sur notre exemple précédent de 1.66 €:

- **étape 1** (1.66 € à rendre) : solution locale ⇒ 1 pièce de 1 euro et il reste 66 centimes ;
- **étape 2** (0.66 € à rendre) : solution locale ⇒ 1 pièce de 50 centimes et il reste 16 centimes ;
- **étape 3** (0.16 € à rendre) : solution locale ⇒ 1 pièce de 10 centimes et il reste 6 centimes ;
- **étape 4** (0.06 € à rendre) : solution locale ⇒ 1 pièce de 5 centimes et il reste 1 centime ;
- **étape 5** (0.01 € à rendre) : solution locale ⇒ 1 pièce de 1 centime.

• L'algorithme glouton est très efficace avec le système monétaire européen, il fournit une solution qui est optimale.

• Imaginons à présent un système monétaire dans lequel il y aurait seulement des pièces de **1**, **4** et **5** unités et que la monnaie à rendre est de 8 unités :

- quelle est la monnaie à rendre en utilisant l'algorithme glouton ?
  $\Longrightarrow$??????????
- quel est le minimum de pièces à rendre dans ce cas ?
  $\Longrightarrow$??????????

• L'algorithme glouton est très efficace avec le système monétaire européen, il fournit une solution qui est optimale.

• Imaginons à présent un système monétaire dans lequel il y aurait seulement des pièces de **1**, **4** et **5** unités et que la monnaie à rendre est de 8 unités :

- quelle est la monnaie à rendre en utilisant l'algorithme glouton ?
  $\Longrightarrow$ la méthode gloutonne rendra 1 pièce de 5 unités et 3 pièces de 1 unité, ce qui fait 4 pièces au lieu de 2. La solution n'est donc pas optimale pour ce système monétaire
- quel est le minimum de pièces à rendre dans ce cas ?
  $\Longrightarrow$ 2 pièces de 4 unités

Exercice :
quelle est la solution optimale, dans le cas où on doit vous rendre la monnaie de 39 €, dans le système monétaire classique ?

Exercice :
quelle est la solution optimale, dans le cas où on doit vous rendre la monnaie de 39 €, dans le système monétaire classique ?

Solution optimale :

- 1 billet de 20 euros ;
- 1 billet de 10 euros
- 1 billet de 5 euros ;
- 2 pièces de 2 euros.

Problème du sac à dos

• On dispose ici d'un panier contenant plusieurs objets. Chaque objet i possède une valeur v_i et un poids p_i associé.

• On souhaite pour notre voyage emporter une partie S de ces objets dans notre sac à dos, malheureusement ce dernier dispose d'une capacité limitée $\mathcal{P}$.

• On va cependant chercher à maximiser la somme des valeurs des objets qu'on va emporter sous la contrainte : $\sum_{i \in S} p_i \leq \mathcal{P}$

• On dispose d'un sac de capacité $\mathcal{P} = 22$ et d'un panier contenant les objets suivant :

Objets	O_1	O_2	O_3	O_4	O_5	O_6
Valeurs	10	8	7	13	6	6
Poids	7	5	4	10	3	3

• Objectif : maximiser la somme des valeurs des objets à emporter.

Solution naïve :

• On peut énumérer toutes les combinaisons d'objets possibles qui satisfont à la capacité maximale du sac (le sac ne doit pas forcément être rempli complètement).

• Beaucoup de combinaisons à calculer :
$C_6^1 = 6; C_6^2 = 15; C_6^3 = 20; C_6^4 = 15; C_6^5 = 6; C_6^6 = 1$
(Tirage d'une quantité d'objets sans ordre !)

• Solution coûteuse et inefficace.

Méthode gloutonne :

- Idée : sélectionner les objets de valeurs élevées en premier, jusqu'à saturation du sac.
- Le sac sera ainsi rempli de la façon suivante :
  - $O_4(13, 10)$, $O_1(10, 7)$ et $O_2(8, 5)$.
  - valeur totale $\Longrightarrow$ 31
  - poids total $\Longrightarrow$ 22
- À priori rien ne garantit l'optimalité de cette solution.

- Prenons un autre exemple avec cette fois-ci des poids déséquilibrés.
- On dispose d'un sac de capacité $\mathcal{P} = 41$.

Objets	O_1	O_2	O_3	O_4	O_5	O_6
Valeurs	30	13	13	13	13	5
Poids	38	10	10	10	10	1

- Ici, l'algorithme glouton choisira l'objet O_1 et l'objet $O_6 \Longrightarrow$ valeur 35.
- On remarque en choisissant les objets O_2, O_3, O_4, O_5, O_6 on aurait pu atteindre la valeur de 57 qui est largement meilleure.
- Comment faire alors pour obtenir une solution optimale ?
  - De fois, les algorithmes gloutons fournissent d'excellents résultats et sont appropriés au problème, dans d'autres cas non. Ce sont des algos qui ont une mauvaise vision globale du problème.
  - Le mieux est de réfléchir à la méthode à adopter en fonction du problème, afin de converger vers la solution la plus optimale.

• Dans le cas où les objets ont une valeur sentimentale, alors comment choisir ?

• Supposons que l'on doit choisir seulement deux objets pour notre voyage entre :

- un bijoux (alliance, bague de fiançailles, bracelet, ...) ;
- un porte-bonheur (gri-gri) ;
- une photo ;
- un vêtement ;
- une peluche ;
- un chapeau ;
- un livre ;
- etc.

Lesquels choisir ? c'est compliqué !

Le problème d'ordonnancement

- Soit un gymnase et des associations qui souhaitent l'utiliser pour leurs activités.
- Il y a n demandes :
  - chaque association i souhaite débuter à `deb[i]` ;
  - chaque association i souhaite finir à `fin[i]`.
- Problème : planifier le nombre maximum d'associations (sans chercher à optimiser le temps d'occupation, qui constitue un autre problème).

Voilà un exemple de données :

Association	Données du problème									
A1	deb1=0		fin1=2							
A2		deb2=1		fin2=3						
A3					deb3=4			fin3=7		
A4									deb4=fin4=8	
A5									deb5=8	fin5=9
TEMPS	0	1	2	3	4	5	6	7	8	9

Modélisation :

- `taille` $\longleftarrow$ 2000
- `deb[taille]`
- `fin[taille]`
- `L_Asso[taille]` "ordre des associations acceptées"

NB : pour simplifier on suppose que les `deb[i]` sont triés : `deb[0]` $\leq$ `deb[1]` $\leq \cdots \leq$ `deb[taille-1]`. Dans le cas où deux asso. commencent à la même heure, celle qui finit le plus tôt est d'abord classée.

Stratégie gloutonne :

1. On considère `L_Asso[taille]`, la liste des associations déjà planifiées pour occuper le gymnase.
2. Supposons `j` la dernière association planifiée.
3. Le gymnase est donc libre à la fin de l'activité de l'asso. `j`, à savoir au temps `fin[j]`.
4. On ajoute une association *i* si et seulement si `deb[i]` $>$ `fin[j]`.
5. Et on itère jusqu'à épuisement.

Fonction *planification* : "Stratégie gauche-droite"

```
-------------------------------------------------------------------
fonction planification(taille, deb[taille], fin[taille],L_Asso[taille])
Debut :
  dernier <-- 0     /*Contient le num de la dernière asso planifiée*/
  L_Asso[0] <-- 0   /*Ajouter asso. 0 à la liste des asso.*/
  nbre_asso <-- 1   /*Contient le nombre total d'asso planifiées et
                      représente aussi la place que va occuper la
                      prochaine asso dans le tableau L_Asso*/

  pour i allant de 1 à taille-1 faire
    si (deb[?????] > fin[?????]) alors
       ????? /*on ajoute l'asso. i à la liste*/
       ????? /*incrémentation du nombre d'asso.*/
       ????? /*sauvegarde du num de la dernière asso ajoutée*/
    fin si
  fin pour

  retourner nbre_asso /*retour du nombre d'asso. planifiés*/
Fin :
-------------------------------------------------------------------
```

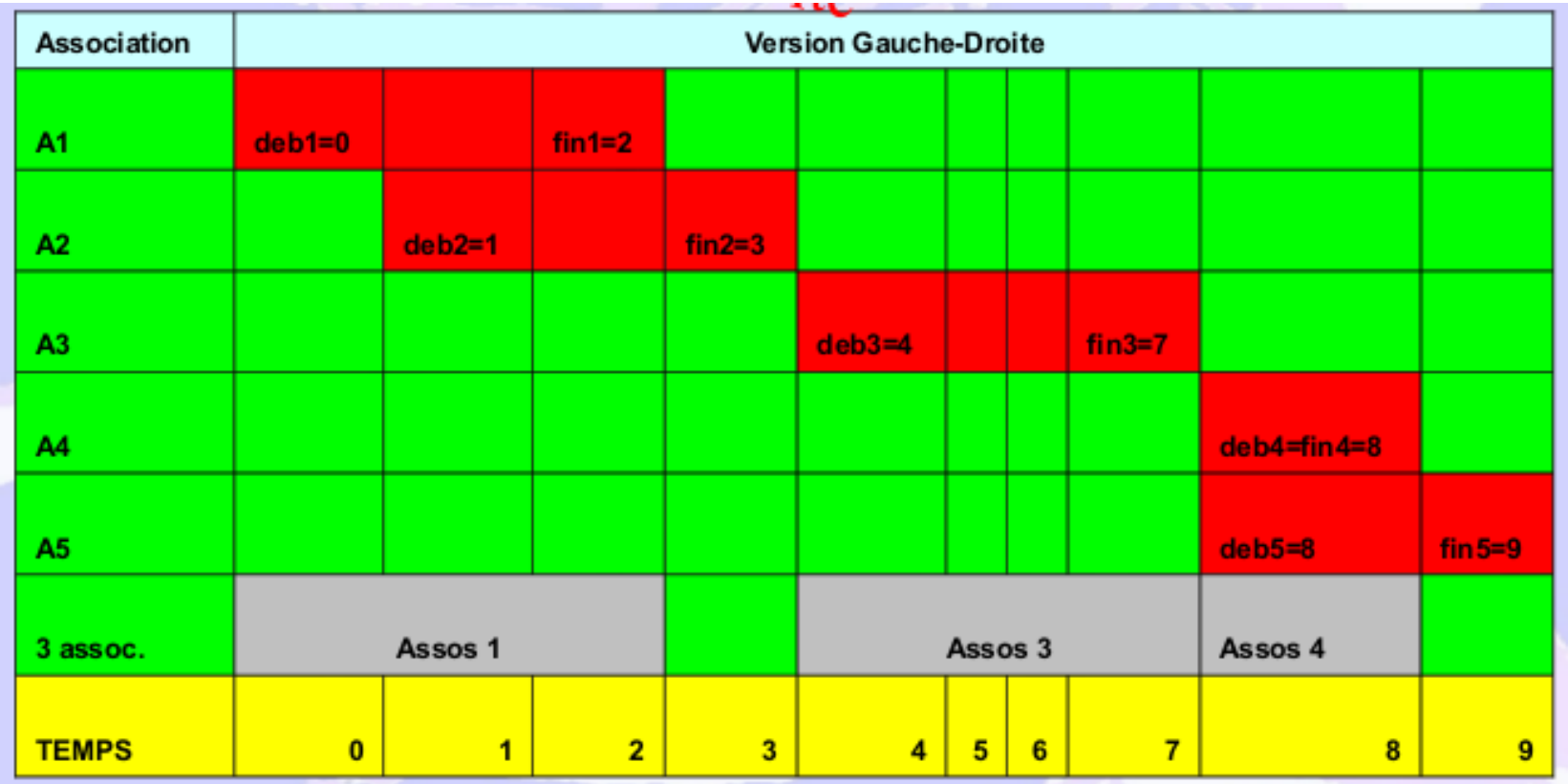
Fonction *planification* : "Stratégie gauche-droite"

```
-------------------------------------------------------------------
fonction planification(taille, deb[taille], fin[taille],L_Asso[taille])
Debut :
  dernier <-- 0     /*Contient le num de la dernière asso planifiée*/
  L_Asso[0] <-- 0   /*Ajouter asso. 0 à la liste des asso.*/
  nbre_asso <-- 1   /*Contient le nombre total d'asso planifiées et
                      représente aussi la place que va occuper la
                      prochaine asso dans le tableau L_Asso*/

  pour i allant de 1 à taille-1 faire
    si (deb[i]>fin[dernier]) alors
       L_Asso[nbre_asso] <- i /*on ajoute l'asso. i à la liste*/
       nbre_asso <- nbre_asso + 1 /*incrémentation du nombre d'asso.*/
       dernier <-- i /*sauvegarde du num de la dernière asso ajoutée*/
    fin si
  fin pour

  retourner nbre_asso /*retour du nombre d'asso. planifiés*/
Fin :
-------------------------------------------------------------------
```

On revient à l'exemple précédent : "Stratégie gauche-droite"

Association	Données du problème									
A1	deb1=0		fin1=2							
A2		deb2=1		fin2=3						
A3					deb3=4			fin3=7		
A4									deb4=fin4=8	
A5									deb5=8	fin5=9
TEMPS	0	1	2	3	4	5	6	7	8	9

Stratégie gauche-droite



Que se passe-t-il si `deb1=0` et que `fin1=9` ?

Stratégie gauche-droite

Association	Version Gauche-Droite									
A1	deb1=0		fin1=2							
A2		deb2=1		fin2=3						
A3					deb3=4			fin3=7		
A4									deb4=fin4=8	
A5									deb5=8	fin5=9
3 assoc.	Assos 1				Assos 3				Assos 4	
TEMPS	0	1	2	3	4	5	6	7	8	9

L'asso A1 sera la seule à être planifiée.

On revient à l'exemple précédent : Que donne la "Stratégie droite-gauche" ?

Association	Données du problème									
A1	deb1=0		fin1=2							
A2		deb2=1		fin2=3						
A3					deb3=4			fin3=7		
A4									deb4=fin4=8	
A5									deb5=8	fin5=9
TEMPS	0	1	2	3	4	5	6	7	8	9

Stratégie droite-gauche

Association	Version Droite-Gauche									
A1	deb1=0		fin1=2							
A2		deb2=1		fin2=3						
A3					deb3=4			fin3=7		
A4									deb4=fin4=8	
A5									deb5=8	fin5=9
3 assoc.		Assos 2			Assos 3				Assos 5	
TEMPS	0	1	2	3	4	5	6	7	8	9

Que se passe-t-il si `deb1=0` et que `fin1=9` ?

Stratégie droite-gauche

Association	Version Droite-Gauche									
A1	deb1=0		fin1=2							
A2		deb2=1		fin2=3						
A3					deb3=4			fin3=7		
A4									deb4=fin4=8	
A5									deb5=8	fin5=9
3 assoc.		Assos 2			Assos 3				Assos 5	
TEMPS	0	1	2	3	4	5	6	7	8	9

On aura la même planification.

Comment améliorer légèrement l'algorithme glouton, afin de maximiser le nombre d'associations ???

Comment améliorer légèrement l'algorithme glouton, afin de maximiser le nombre d'associations ???

$\Longrightarrow$Il suffit de comparer $fin[i]$ et $fin[i+1]$, c'est-à-dire :

- Si $deb[i] = deb[i+1]$ alors on choisira l'asso. i si $fin[i] < fin[i+1]$ sinon on choisira l'asso. $i+1$
  (comportement normal de l'algo. précédent)
- (mais) Si $deb[i] < deb[i+1]$, idem: on choisira l'asso. i si $fin[i] < fin[i+1]$ sinon on choisira l'asso. $i+1$

Association										
A1	deb1=0									fin1=9
A2		deb2=fin2=1								
A3			deb3=fin3=2							
A4				deb4=fin4=3						
A5					deb5=fin5=4					
4 assos		Assos 2	Assos 3	Assos 4	Assos 5					
TEMPS	0	1	2	3	4	5	6	7	8	9

En voici l'exemple!

Exercice :

- programmer l'algorithme d'ordonnancement
- faire la version gauche-droite
- faire la version droite-gauche
- faire la version améliorée

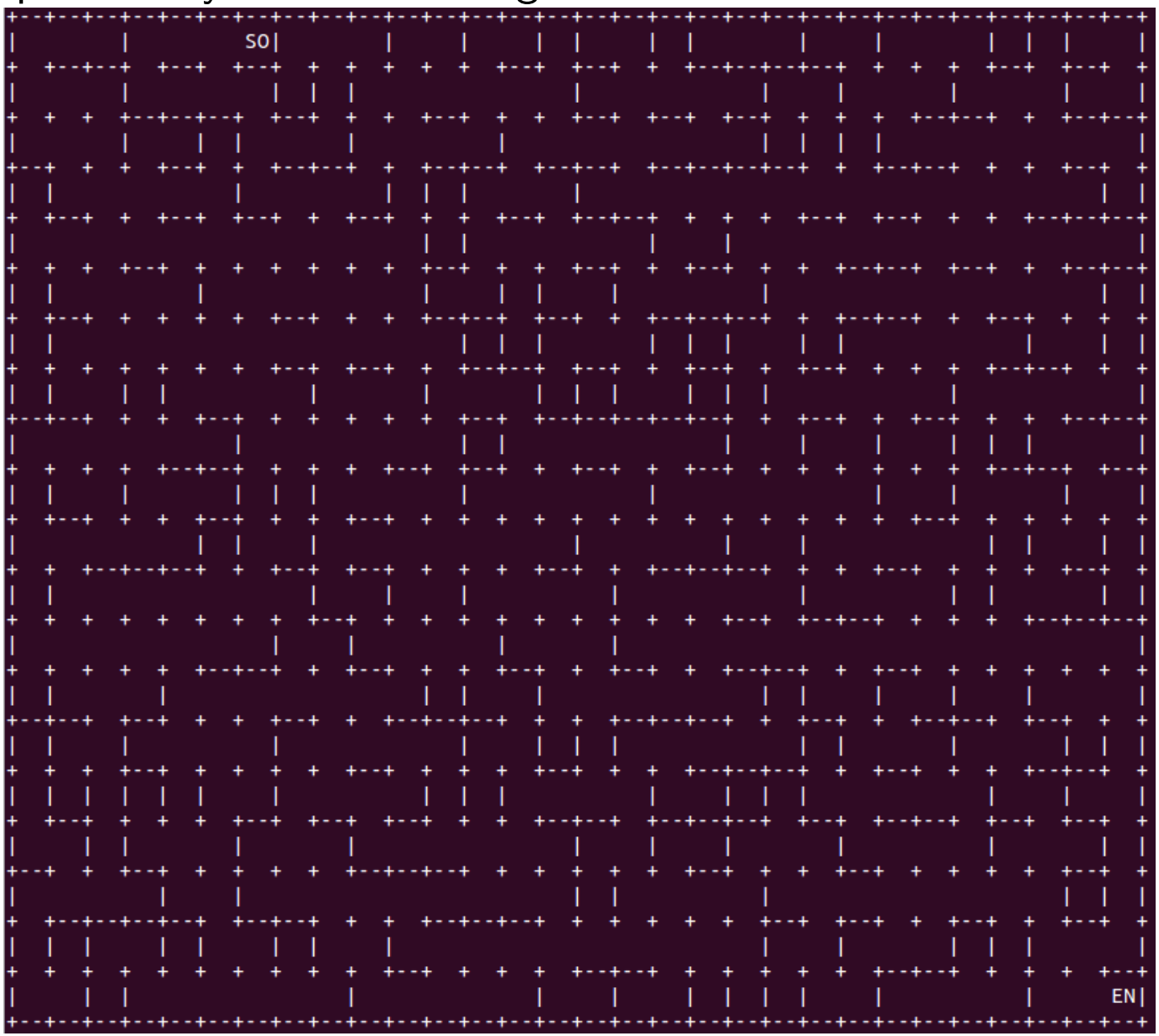
Entrée : un fichier de $N + 1$ entiers : la première ligne contient N qui est le nombre d'associations à planifier et ensuite le début et fin de chaque association.

Sortie : la liste des associations retenues dans l'ordre.

Labyrinthe 2D

- On dispose d'un labyrinthe 2D possédant :
  - une entrée (EN) ;
  - une sortie (SO).
- Objectifs :
  - faire une recherche d'un chemin quelconque entre l'entrée et la sortie ;
  - puis le plus court chemin si possible dans ce labyrinthe.

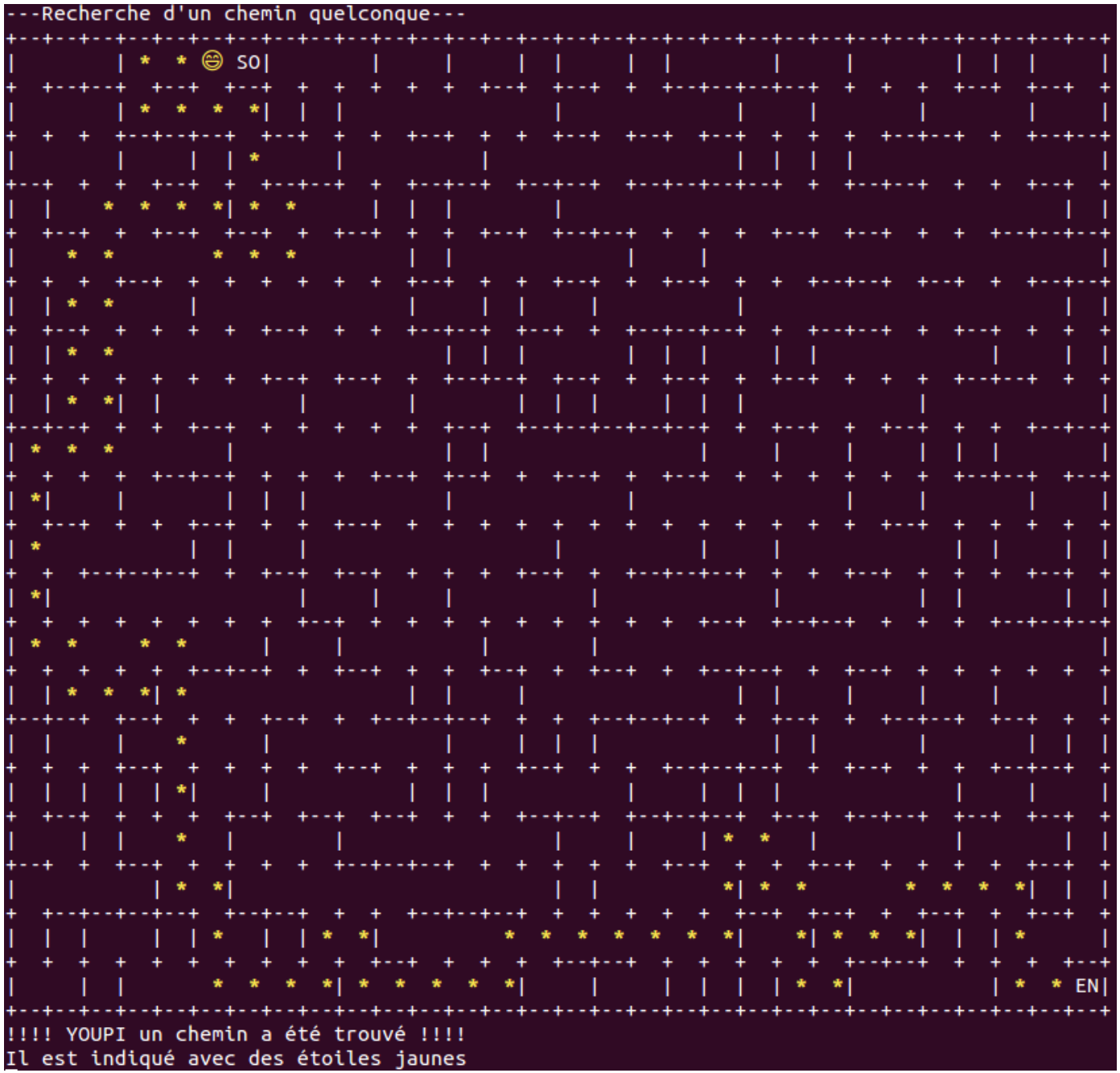
Exemple de labyrinthe 20 × 30 généré aléatoirement



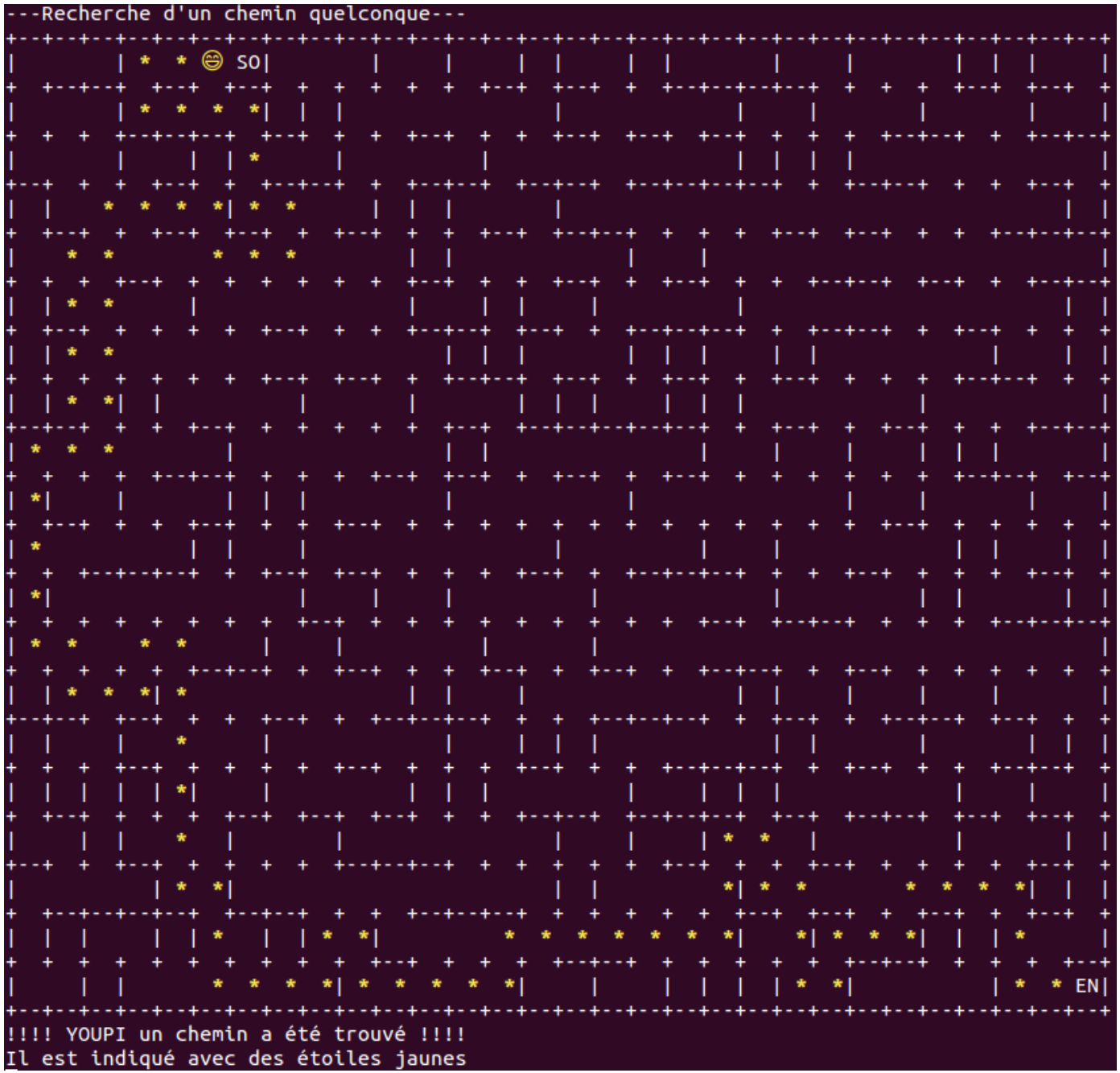
Recherche d'un chemin quelconque :

Quelle stratégie gloutonne adopter ? plusieurs stratégies possibles !!!!!

- **STRATÉGIE 1 :** à partir d'une cellule, on peut sortir aléatoirement vers une des cellules voisines quand cela est possible, sinon retour en arrière.
- **STRATÉGIE 2 :** à partir d'une cellule, on peut sortir vers une des cellules voisines, par exemple dans le sens des aiguilles d'une montre en fonction de l'entrée de la cellule actuelle. Si aucune sortie possible, revenir en arrière.
- **STRATÉGIE 3 :** à partir d'une cellule, on peut sortir vers une des cellules voisines, par exemple dans le sens **contraire** des aiguilles d'une montre en fonction de l'entrée de la cellule actuelle. Si aucune sortie possible, revenir en arrière.
- ...



Quelle **STRATEGIE** a été adoptée ??? **STRATEGIE 1 ? STRATEGIE 2 ?** ou **STRATEGIE 3 ?**


---Recherche d'un chemin quelconque---
!!!! YOUPI un chemin a été trouvé !!!!
Il est indiqué avec des étoiles jaunes

STRATEGIE 2

Recherche de chemin le plus court :

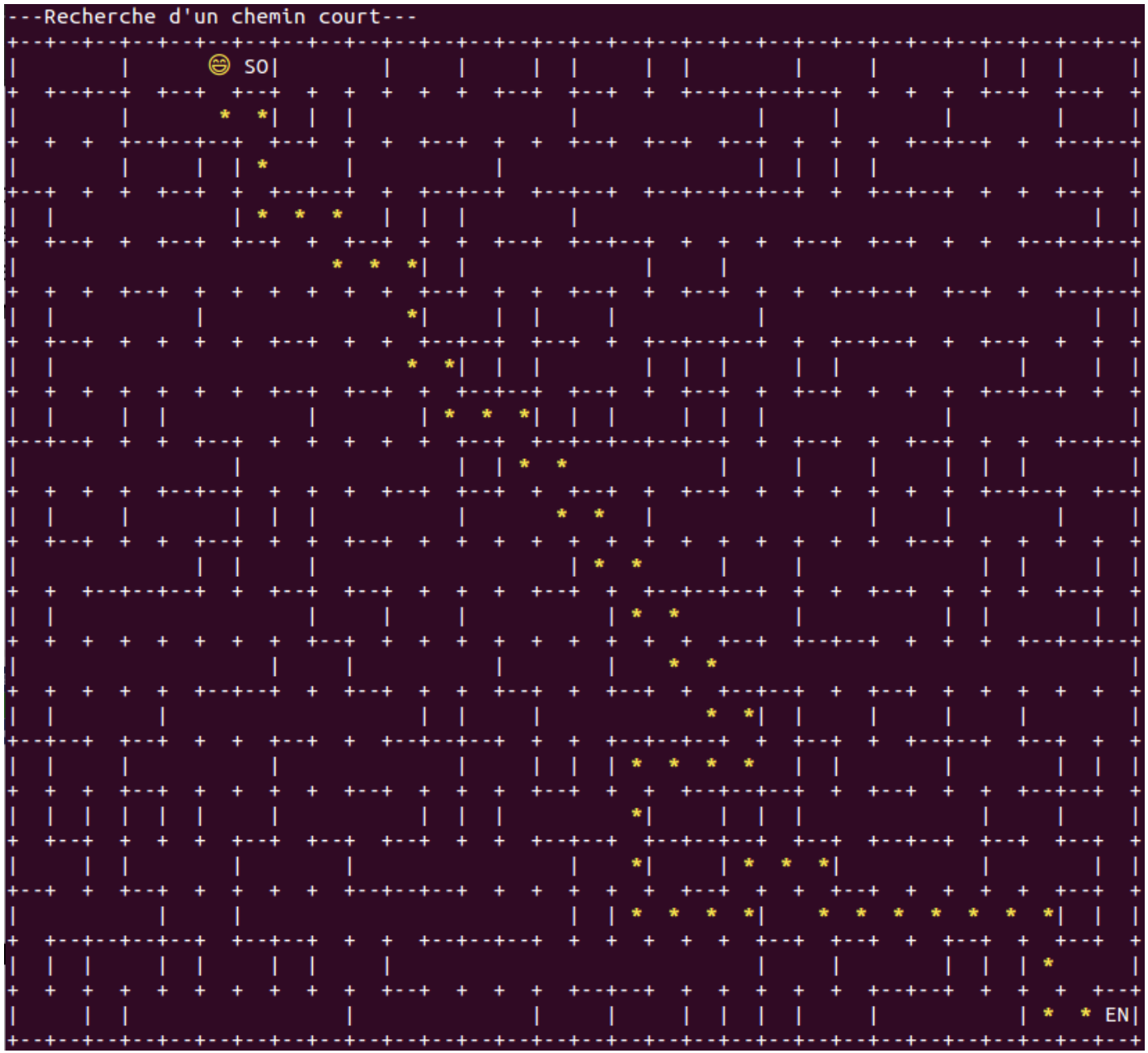
Quelle stratégie gloutonne adopter ?

- À partir d'une cellule, on peut sortir vers la cellule voisine la plus proche de la sortie finale quand cela est possible. Avec cette stratégie on est sur de faire le meilleur choix de sortie localement. Mais est-ce que ce choix conduit tout le temps au chemin le plus court entre l'entrée et la sortie du labyrinthe ???????

Recherche de chemin le plus court :

Quelle stratégie gloutonne adopter ?

- À partir d'une cellule, on peut sortir vers la cellule voisine la plus proche de la sortie quand cela est possible. Avec cette stratégie on est sur de faire le meilleur choix de sortie localement. Mais est-ce que ce choix conduit tout le temps au chemin le plus court entre l'entrée et la sortie du labyrinthe ???????
  NON !!!



Est-ce que ce chemin est réellement le plus court entre l'entrée et la sortie ?

Comment améliorer cette stratégie afin d'atteindre le chemin le plus court dans le labyrinthe ?

Comment améliorer cette stratégie afin d'atteindre le chemin le plus court dans le labyrinthe ?

- une fois le chemin court calculé on le stocke, puis on recalcule un autre chemin court en changeant de cellules de sorties par moment. Si le nouveau chemin est plus court que l'ancien, alors on met à jour le chemin et on continue le processus.
- On peut calculer dès le départ tous les chemins courts possibles, et prendre le plus court de tous.
- On peut aussi travailler sur le problème dual qui consiste à commencer par la sortie pour atteindre l'entrée.
- ...

Bibliographie

- R. ERRA, "Cours d'informatique", 1A-S2 2013-2014 ESIEA.
- Cormen, Leiserson, Rivest, Stein "Algorithmique", 3ème éd. DUNOD.
- `http://openclassrooms.com/courses/les-algorithmes-gloutons`