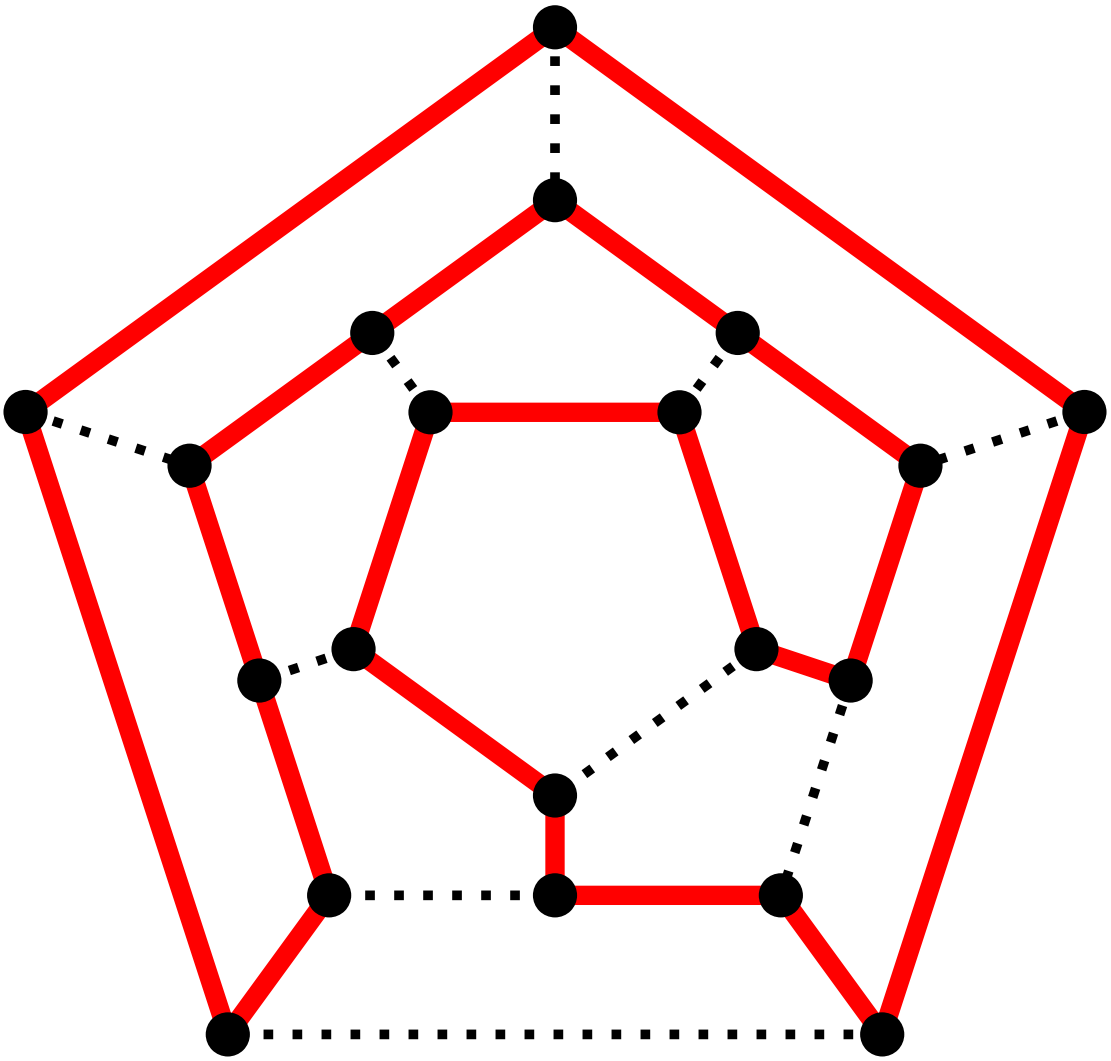
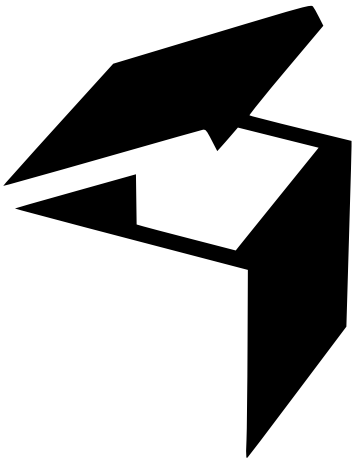
LES ALGORITHMES





SCIENCES MANUELLES DU NUMERIQUE

DÉCONSTRUIRE L'INFORMATIQUE POUR MIEUX LA COMPRENDRE

Page web : `http://www.loria.fr/~quinson/Mediation/CSIRL/`
Sources et ressources : `http://github.com/jcb/CSIRL`
Pour nous joindre : `discussions@listes.nybi.cc`

Crédits image
P1: Chemin hamiltonien par Ch. Sommer (licence GFDL/CC-BY-SA)
`http://en.wikipedia.org/wiki/File:Hamiltonian_path.svg`
P4: Computer Science Major (licence CC-BY-NC)
`http://abstrusegoose.com/206`
P21: exemple de TSP adapté de Wikipedia (licence GFDL/CC-BY-SA)
`http://en.wikipedia.org/wiki/File:Aco_TSP.svg`
P23: Travelling Salesman Problem par XKCD (licence CC BY-NC 2.5)
`http://xkcd.com/399/`

La science informatique, sans ordinateur

Trop souvent, lorsque l'on parle d'informatique, on pense à l'ordinateur utilisé comme outil. L'informatique devient alors l'art d'utiliser l'ordinateur pour une tâche donnée, ou de le réparer. Pourtant, dans les entrailles de cette machine se cache un domaine scientifique très vaste, dont les ramifications dépassent largement l'ordinateur et son fonctionnement.

Avec le projet **Sciences Manuelles du Numérique**, nous vous proposons d'explorer la science informatique... en retirant l'ordinateur ! Pour cela, nous avons conçu une série d'activités ludiques introduisant des notions fondamentales de l'informatique par le biais d'un support matériel, pour *apprendre avec les mains*. Ces activités sont regroupées en séances thématiques, permettant ainsi d'aborder les notions fondamentales de manière cohérente et progressive.

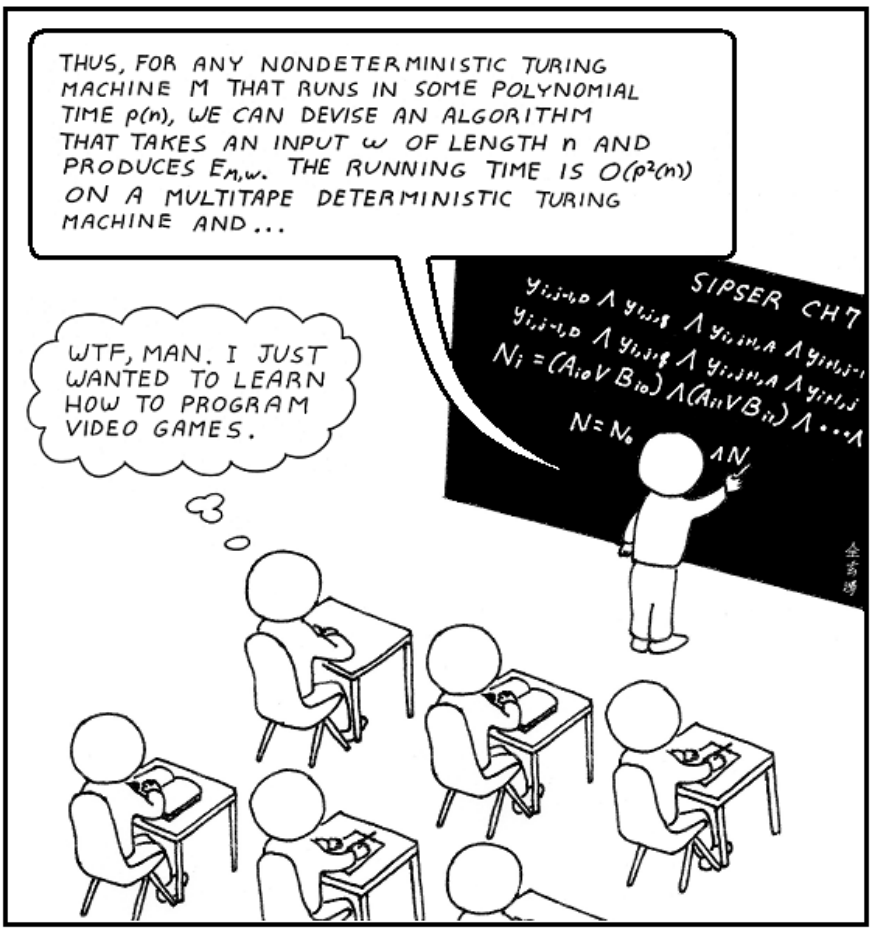
La séance algorithmique

L'objectif de cette séance est d'introduire la notion fondamentale d'**algorithme**. Qu'est-ce-que c'est ? A quoi ça sert ? Comment sont-ils inventés ? Pour cette séance, comptez **une heure et demi à deux heures**. Bien entendu, il ne s'agit pas d'un cours complet sur le sujet ! Si vous souhaitez aller plus loin, voici un cours en 48h : `http://www.loria.fr/~quinson/Teaching/TOP/`.

Les animateurs d'ateliers trouverons des conseils et astuces à la fin de chaque activité dans « le coin de l'animateur » ; mais les conseils les plus importants sont certainement les suivants :

- appropriez vous les activités. Pratiquez-les à l'avance et n'hésitez pas à ne pas suivre les consignes à la lettrer ;

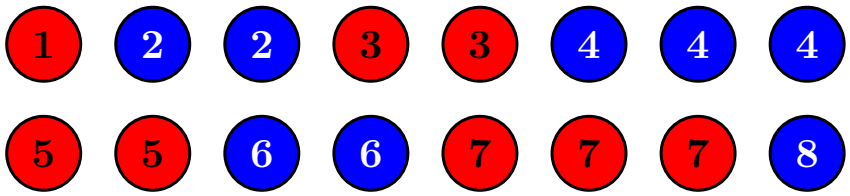
- ces activités sont des bases de discussion avec les participants, il n'y a pas d'évaluation à la finr ;
- évitez les introductions théoriques ; commencez par les activités, elles serviront de support pour discuter de la théorie.



Le jeu de Nim

Voici un premier petit jeu simple, pour rentrer dans le sujet. On dispose sur une table *16 objets*. Chacun leur tour, les deux joueurs ramassent *un, deux ou trois objets* sur la table. Le joueur qui **ramasse le dernier objet** remporte la partie.

Matériel
• 16 petits objets (clous, allumettes... peu importe !)



Le joueur bleu gagne

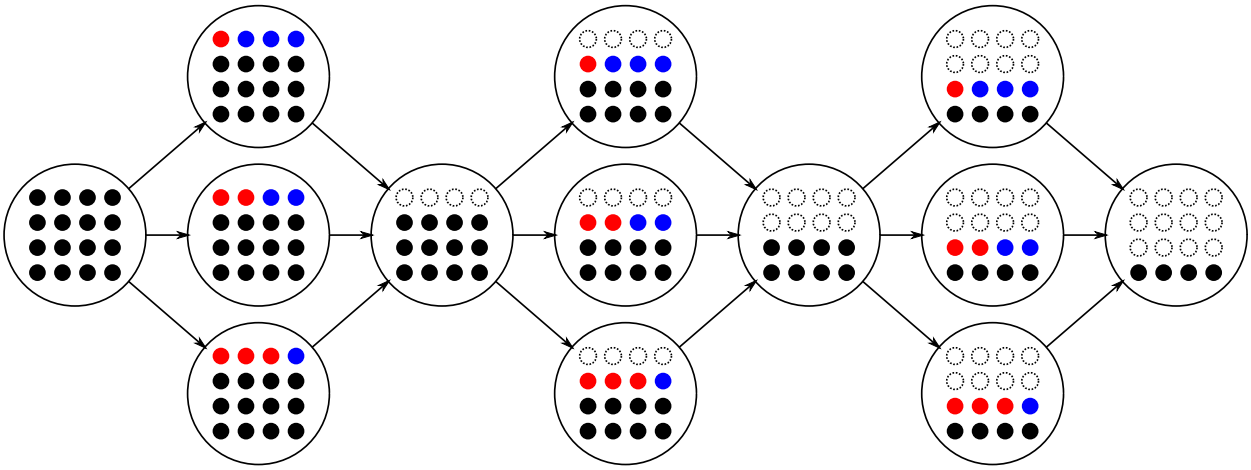
Stratégie gagnante

Le jeu de Nim est sans suspense : le premier à jouer perd, car il existe une astuce pour que le deuxième joueur gagne à tous les coups. La **stratégie gagnante** est de laisser 4, 8, 12 ou 16 objets à l'adversaire (un multiple de 4).

Pour se convaincre de l'efficacité de la stratégie gagnante, prenons le dernier tour comme exemple. Il reste 4 objets, et J1 joue :

- si J1 prend 1 objet, J2 en prend 3 (dont le dernier) ;
- si J1 prend 2 objets, J2 en prend 2 (dont le dernier) ;
- si J1 prend 3 objets, J2 en prend 1 (le dernier).

Dans ce cas, si J2 sait jouer, J1 perd à tous les coups. En appliquant la même méthode, J2 peut guider le jeu de manière à passer de 16 objets à 12, puis 8 et enfin 4. Donc, si J2 sait jouer, J1 a perdu la partie avant même de commencer.



Rapport avec l'informatique

Comme pour le jeu de Nim, **un algorithme est une stratégie gagnante** permettant de trouver la solution à un problème donné. Dans l'exemple précédent, le problème était « comment gagner au jeu de Nim ? »

Pour aller plus loin

On pourrait imaginer un cas plus général du jeu de Nim :

- Il y a N objets sur la table au début du jeu
  (pour notre version, $N = 16$)
- Un joueur peut prendre jusqu'à X objets à la fois
  (pour notre version, $X = 3$)

Quelles modifications doit-on apporter à notre stratégie gagnante pour qu'elle marche dans le cas général ?

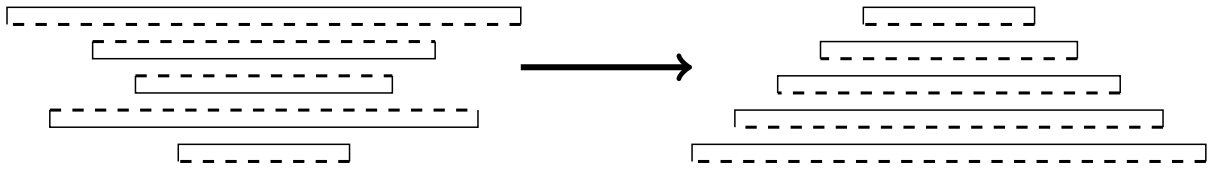
Le coin de l'animateur

L'objectif de cette activité est simplement d'introduire la notion d'algorithme comme stratégie gagnante pour un problème donné.

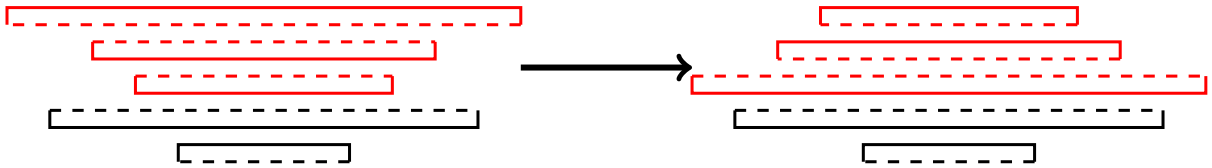
- Commencez par jouer avec les participants, sans dire qu'il y a un truc. Si vous jouez bien, vous gagnerez à tous les coups.
- Bien sûr, pour gagner, vous devez laisser votre adversaire commencer. S'il insiste pour ne pas commencer, vous pouvez toujours essayer de gagner en rattrapant la stratégie gagnante à la première erreur.
- Si un participant connaît déjà la stratégie gagnante du jeu, il pourra vous remplacer pour jouer avec les autres participants.
- Si vous n'êtes pas sûr d'appliquer correctement la stratégie gagnante, proposez un match en 3 — ou en 5, en cas de coup dur ;) — manches gagnantes.
- Pour amener les participants à découvrir la stratégie gagnante, vous pouvez grouper les objets par 4, rendant ainsi l'astuce plus visible.

Le crêpier psycho-rigide

A la fin de sa journée, un crêpier dispose d'une pile de crêpes désordonnée. Le crêpier étant un peu psycho-rigide, il décide de ranger sa pile de crêpes, de la plus grande (en bas) à la plus petite (en haut), avec le coté brûlé caché.



Pour cette tâche, le crêpier peut faire une seule action : glisser sa spatule entre deux crêpes et retourner le haut de la pile. Comment doit-il procéder pour trier toute la pile ?



Matériel

- Des planchettes en bois de tailles et de couleurs différentes (faces reconnaissables)
- Une pelle à tarte pour retourner les planchettes (optionnelle)

Description d'un algorithme

L'algorithme permettant de résoudre le problème du crêpier est le suivant :

1. amener la plus grande crêpe en haut de la pile
2. mettre la face brûlée vers le haut
3. retourner toute la pile - la crêpe est rangée
4. recommencer en ignorant les crêpes rangées

Cet algorithme assez simple nous apprend deux choses. Premièrement, un algorithme n'a d'intérêt que si on peut l'expliquer - pire encore, que si on peut l'*expliquer à un ordinateur*. Il doit donc être **écrit sans ambiguïté**. Deuxièmement, un algorithme décompose le problème en une série de tâches simples. On appelle ce principe « **Diviser pour mieux régner** ».

Ce que nous venons de décrire est le cœur de métier des informaticiens : analyser un problème, le subdiviser en problèmes plus simples, formaliser le tout sous la forme d'un algorithme, et traduire l'algorithme dans un langage compréhensible par l'ordinateur.

Performance d'un algorithme

Le premier objectif de l'écriture d'un algorithme est qu'il résolve le problème. Le second objectif est qu'il le résolve le plus vite possible.

Évaluer la **performance d'un algorithme** nous permet de nous faire une idée du temps nécessaire à la résolution d'un problème de grande taille. Par exemple, combien de temps faudra-t-il à notre algorithme pour trier une pile d'un million de crêpes ?

De plus, s'il existe plusieurs algorithmes résolvant le même problème, l'évaluation de la performance nous donne un critère objectif pour savoir lequel est le plus efficace.

Évaluer la performance d'un algorithme

Pour évaluer la performance, on compte le nombre de « coups » nécessaires pour résoudre le problème dans le cas général. Pour le problème du crêpier :

- pour ranger une crêpe, il faut entre 0 coup (la crêpe est déjà rangée) et 3 coups (amener en haut, retourner, amener à sa place) ;
- pour n crêpes (cas général), il faut entre 0 coup (meilleure situation) et $3 \times n$ coups (pire situation). La performance de l'algorithme dépend donc beaucoup de l'état initial, mais on s'intéresse surtout aux cas intermédiaires, qui sont les plus probables.

Ici, n est une variable qui exprime la taille du problème. La performance d'un algorithme est notée comme une fonction de la taille du problème nommée O. Pour le crêpier, on peut donc écrire $O(3 \times n)$.

Le temps d'exécution variant selon l'état initial, la performance exprime l'ordre de grandeur du temps d'exécution. Pour des grandes valeurs de n, il n'est pas très utile de faire la distinction entre $O(n)$, $O(n+4)$ ou encore $O(3 \times n)$ — en particulier quand on compare avec un autre algorithme dont la performance est $O(n^2)$.

On simplifie donc en retirant les constantes pour ne garder que les termes importants. Pour le crêpier, on notera donc la performance de notre algorithme $O(n)$; on dit alors que la performance de l'algorithme est *linéaire*, car le temps de calcul croît linéairement avec n. Un algorithme avec une performance $O(n^2)$ est dit *quadratique*, tandis qu'un algorithme $O(2^n)$ est dit *exponentiel*.

À la recherche du meilleur algorithme possible

On arrive parfois à montrer qu'on a le meilleur algorithme possible. Par exemple, on ne peut pas trier les éléments en moins de n étapes, car on doit forcément tous les considérer.

On peut aussi prouver qu'un tri comparatif ne peut pas se faire en moins de $n \times log(n)$ étapes, car il n'accumule pas assez d'informations pour choisir la bonne permutation en moins d'étapes.

Mais la plupart du temps, on ne sait pas prouver que l'algorithme connu est le meilleur possible. C'est alors le meilleur *connu*, sans être forcément le meilleur *possible*.

Le coin de l'animateur

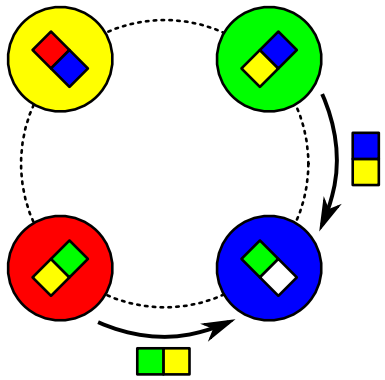
L'objectif de cette activité est de trouver un algorithme, de le faire verbaliser par les participants et d'en mesurer la performance.

- Expliquez les règles et demandez aux participants de tenter de résoudre le problème ;
- s'ils bloquent, conseillez-les. Par exemple : « essaye d'abord de mettre la grande crêpe en bas », ou encore « où doit se trouver la grande crêpe pour pouvoir l'amener en bas ? »
- Quand les participants ont trouvé l'algorithme, demandez-leur de l'expliquer.
- Demandez ensuite de calculer le nombre de coups nécessaires pour ranger la pile de crêpes. Le nombre de coups dépendant de l'état initial, faites-les généraliser en trouvant le nombre de coups maximal pour ranger une crêpe, puis n crêpes.
- Le discours sur le $O(n)$ est volontairement approximatif. On veut faire sentir les choses ; faire un vrai cours prend une douzaine d'heures (cf. `http://www.loria.fr/~quinson/Teaching/TOP/`).

Le Base-ball multicolore

On dispose de quatre bases de couleurs différentes, et deux joueurs associés à chaque base. Le but du jeu est de déplacer les joueurs afin d'amener chaque joueur sur la base correspondant à sa couleur. Il y a cependant trois contraintes :

- les bases sont disposées en cercle, et un joueur ne peut se déplacer que vers les deux bases voisines (il ne peut pas traverser le terrain) ;
- on ne peut déplacer qu'un joueur à la fois ;
- chaque base a deux places, et un joueur ne peut se déplacer vers une base que si elle possède une place libre.



Matériel

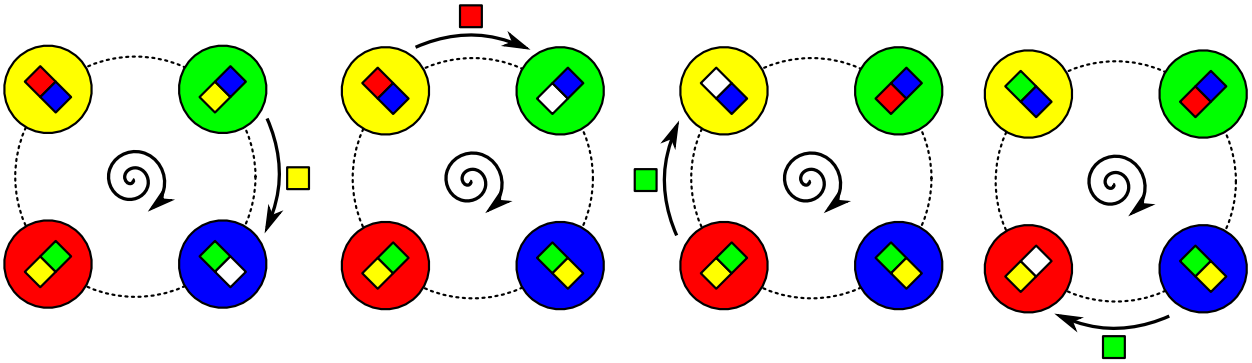
- Plusieurs équipes bien différenciables, chacune composée d'une base et de deux joueurs (des LEGO, des bouts de bois, des cailloux, du fil électrique de différentes couleurs, etc.)
- 4 équipes au minimum. On peut mettre des équipes ou des joueurs supplémentaires pour augmenter la difficulté.

L'objectif de cette activité est d'**expliquer clairement** la méthode de résolution du problème (algorithme), ainsi que le raisonnement qui a permis de trouver cette méthode.

Premier algorithme

En suivant les règles du jeu, on observe que quelle que soit la disposition des joueurs, 4 joueurs peuvent être déplacés vers la place vide : les deux de la base de gauche, les deux de la base de droite. Notre algorithme sera donc une méthode permettant de choisir à chaque étape quel coup jouer parmi ces 4 possibles.

- On ne s'autorise à tourner que dans un seul sens. Ainsi, le nombre de coups possibles descend de 4 à 2 (car 2 joueurs tourneraient à l'envers).
- Parmi les 2 coups restants, on déplace le joueur qui a la plus grande distance à parcourir avant d'arriver à sa base (Si la distance est la même, c'est que les deux joueurs ont la même couleur - les deux coups sont alors équivalents).
- Tant que tous les joueurs ne sont pas rentrés à leur base, on continue les déplacements.

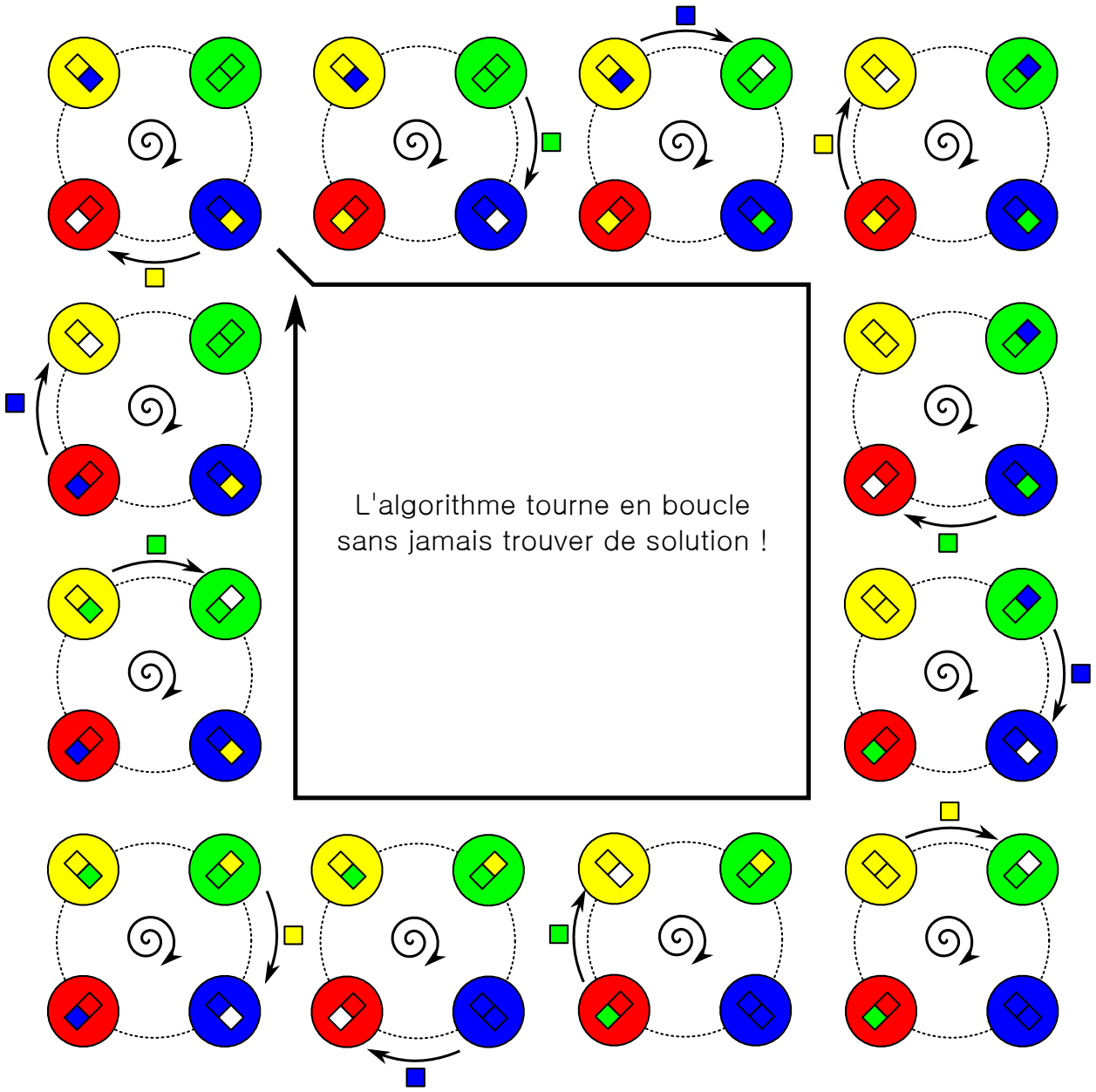


Ici, nous n'avons représenté que les 4 premières étapes, mais l'algorithme arrive à la solution en 15 étapes.

Regardons plus en détail. . .

À première vue, cet algorithme est attirant : il est assez simple et semble relativement rapide — 15 coups pour 4 bases et 7 joueurs — ce n'est pas si mal. Pourtant, il y a un problème : *cet algorithme est faux.*

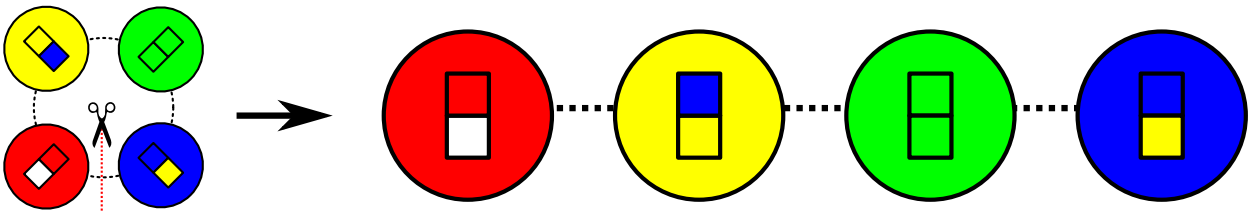
Pour s'en convaincre, il suffit de prendre le jeu dans son état résolu, et d'intervertir deux joueurs. On observe alors que notre algorithme ramène le jeu à son état initial sans atteindre la solution - notre algorithme boucle donc à l'infini.



Deuxième algorithme

Notre premier algorithme étant faux, réfléchissons à un autre algorithme.

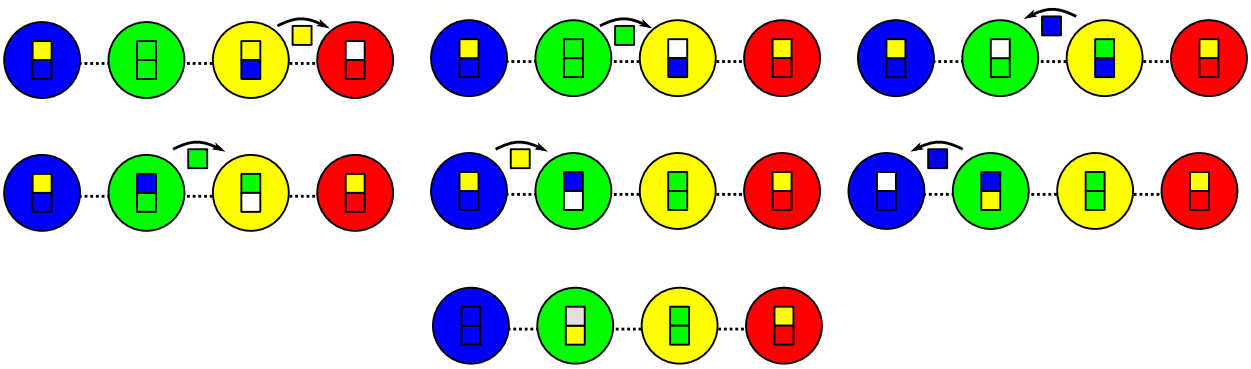
Commençons par *apprendre de nos échecs*. Notre premier algorithme boucle parfois à l'infini. Pour réparer cela, on pourrait s'interdire de faire un tour complet en coupant le cercle. Pour ne pas se tromper, cela revient à disposer les bases sur une ligne.



Puis, *apprenons de nos réussites*. Dans le problème du crêpier, on a résolu le problème en réduisant progressivement sa taille : on place la première crêpe, puis la deuxième etc. On s'est fixé des objectifs intermédiaires qui décomposent le problème en étapes plus simples.

À partir de ces enseignements, on peut construire l'algorithme suivant :

- On coupe le cercle de sorte que la base avec la place vide soit à l'extrémité droite.
- On s'occupe des bases les unes après les autres, de gauche à droite.
- Pour rapprocher un joueur de sa base, on déplace les joueurs des autres couleurs pour amener le trou à gauche du joueur à déplacer.
- On répète l'opération jusqu'à ce que les deux joueurs soient revenus à leur base, et on n'y touche plus.



On peut maintenant ignorer la première base, qui est déjà rentrée.

Correction d'algorithmes

Notre deuxième algorithme est un peu plus complexe que le précédent, mais il est correct — une propriété très importante quand on écrit un algorithme. Mais comment être sûr de la **correction** d'un algorithme ? Il existe plusieurs approches :

Tester tous les cas. On vérifie que l'algorithme trouve la solution pour tous les cas possibles. Mais ça ne marche que pour un problème de taille finie, et montrer de cette manière que notre algorithme est correct pour 4 bases n'implique pas sa correction pour 5 bases.

Écrire une preuve mathématique. On peut prouver mathématiquement que cet algorithme fonctionne pour le cas général m bases. Ce n'est pas trivial, mais les chercheurs en informatique en ont écrit des plus difficiles.

Ça ressemble à un algorithme classique. On peut observer une certaine ressemblance avec un algorithme déjà connu. Mais ça ne prouve rien, au fond...

Les algorithmes classiques

Les informaticiens apprennent par cœur des algorithmes (abstraits) à l'école. Face à un problème nouveau, un informaticien cherche généralement à le ramener à un problème connu. Ce rapprochement se fait en trouvant des analogies ou en décomposant le problème en plusieurs sous-problèmes connus.

Par exemple, quand des collègues informaticiens jouent au crêpier, ils demandent avant tout si c'est « une tour de Hanoï », jeu bien connu des informaticiens. Pour le base-ball multicolore, notre algorithme ressemble à un « tri à bulle », autre algorithme bien connu. Mais cette ressemblance ne suffit pas à prouver la correction de notre algorithme.

Pour la prouver, il faudrait démontrer que notre algorithme est un cas particulier du tri à bulle.

Les algorithmes de tri

Les algorithmes de tri sont très classiques en informatique. D'une part, ils permettent d'expliquer les grands principes aux élèves (« diviser pour régner », récursivité, algorithmes gloutons ...). D'autre part, les ordinateurs trient très souvent des données, car beaucoup de problèmes sont plus simples après (trouver un livre est plus simple dans une bibliothèque rangée, par exemple). *Les musiciens font leurs gammes, les informaticiens débutants apprennent leurs algorithmes.*

Le coin de l'animateur

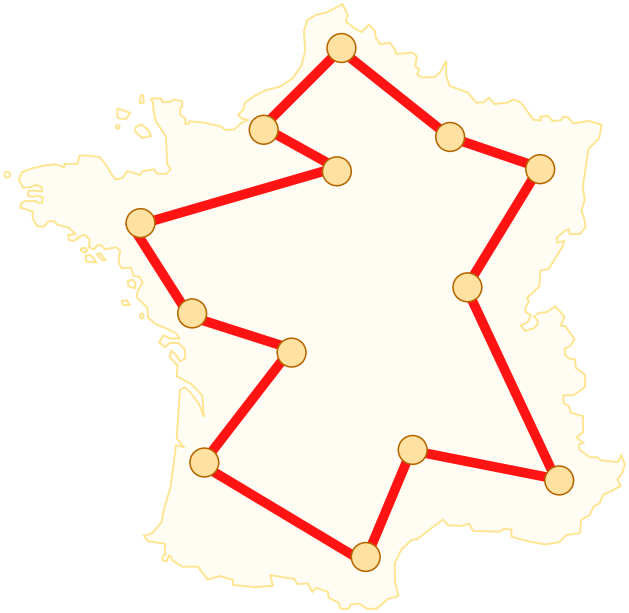
L'objectif de cette activité est d'introduire la notion de correction d'algorithme.

- Laissez les participants chercher un peu en les faisant verbaliser.
- S'ils sont sur le point de trouver l'algo juste, introduisez très vite l'algo faux pour préserver un enchaînement logique : « oui, ok, mais je vais vous montrer une façon de faire rigolote. »
- Quand l'algo juste est établi, et avant de parler de performance, on peut partir sur une variante :
  - Chaque participant prend une couleur (une base placée au sol entre ses pieds)
  - Chaque participant (sauf 1) prend un bonhomme dans chaque main
  - À chaque étape, celui qui a une main libre prend un bonhomme dans la main d'un voisin
  - (Attention, c'est fastidieux à 8 ou 9 couleurs, il vaut mieux faire deux rondes car l'algorithme semble $O(n^2)$)
- Expérimentalement, l'algorithme qui tourne converge très souvent vers la solution à 5 bases, mais converge souvent vers la boucle infinie quand il y a plus de couleurs. Ne tentez pas le diable ;)

- Dans la disposition linéaire, il est plus simple de mettre la couleur avec un seul bonhomme à une extrémité, et commencer par remplir la maison de l'autre extrémité. Sinon, on se retrouve avec une maison remplie de un seul au milieu, et il faut comprendre que la solution passe par le stockage temporaire d'un pion de la maison d'à coté sur le trou.
- Il serait intéressant de prouver effectivement la correction de l'algorithme linéaire, ainsi que de quantifier la probabilité de fonctionnement de l'algo qui tourne en fonction du nombre de maisons.

Le plus court chemin

Soit un ensemble de villes représenté par des clous plantés dans une planche de bois. Le but du jeu est de trouver le plus court chemin passant une fois et une seule par chaque ville et terminant sur la ville de départ. Pour représenter le chemin, on attache une ficelle à un clou et on la fait passer de clou en clou.



Matériel

- Une planche avec des trous au hasard,
- autant de longs clous que de trous,
- une ficelle suffisamment longue et **qui ne soit pas élastique**,
- un marqueur.

Il existe un grand nombre de chemins possibles. Arriverez-vous à trouver le plus court ?

Recherche de la solution optimale

Ce problème a de nombreuses applications dans la vie de tous les jours : minimiser la tournée du facteur, la longueur des pistes d'un circuit imprimé, les déplacements d'un bras robotiquex... C'est un problème très étudié, plus connu sous le nom de « **problème du voyageur de commerce** ».

Le problème du voyageur de commerce appartient à une catégorie de problèmes très difficiles à résoudre. Pour ces problèmes, on ne dispose pas d'algorithmes assez performants pour le résoudre quand n est élevé. Le problème du voyageur de commerce ayant fait l'objet de nombreuses études, beaucoup d'algorithmes ont été proposés pour le résoudre le plus efficacement possible.

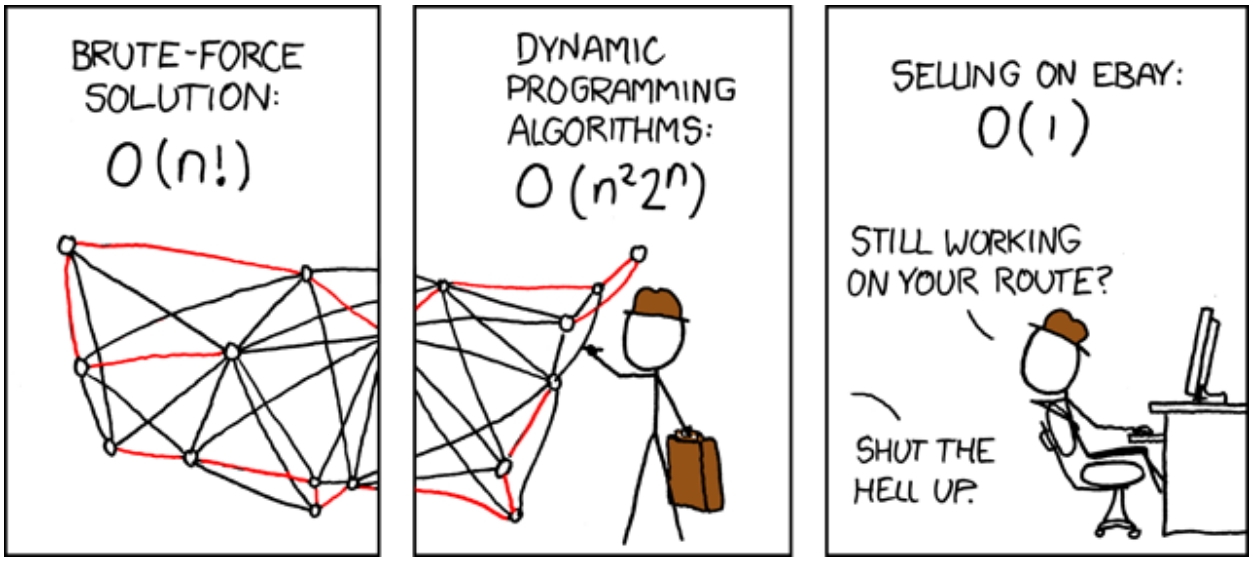
Comparaison de deux algorithmes

L'approche naïve consiste à calculer la longueur de tous les chemins possibles, et comparer les résultats pour ne retenir que le plus court. Pour n villes, le nombre de chemins possible est $n!$. La performance de l'algorithme est donc $O(n!)$.

Il existe cependant des algorithmes plus efficaces : par exemple, la performance de l'algorithme Held-Karp est $O(n^2 2^n)$. Pour illustrer la différence, comparons l'augmentation des calculs nécessaires à mesure que n augmente :

Nombre de sommets	5	10	15	20
Méthode naïve $O(n!)$	120	3628800	1.3×10^{12}	2.4×10^{18}
Held-Karp $O(n^2 2^n)$	800	102400	7.3×10^{6}	4.1×10^{8}

Ce tableau indique qu'en testant un milliard de chemins par seconde, il faudrait à la méthode naïve plus de **77 ans** pour trouver le chemin le plus court entre 20 sommets ! Dans les mêmes conditions, l'algorithme Held-Karp met moins d'**une demi-seconde** pour trouver le même résultat.

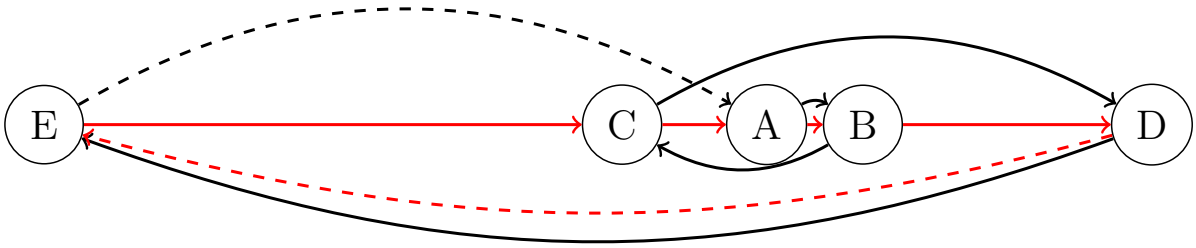


Recherche d'une solution approchée

Pour ces problèmes complexes, on préfère souvent trouver une solution raisonnablement bonne (solution approchée) très rapidement, plutôt que de chercher très longtemps la solution optimale. Une **heuristique** est une méthode pour fouiller intelligemment l'espace des solutions possibles à la recherche des bonnes solutions. La recherche d'heuristiques efficaces fait partie du travail des chercheurs en informatique.

Heuristique : le plus proche voisin

Partez d'une ville, et allez vers la ville la plus proche par laquelle vous n'êtes pas encore passé. Recommencez, jusqu'à passer par toutes les villes, et revenez au point de départ. Cette heuristique aura tendance à prendre de préférence les distances les plus courtes... Mais elle n'aboutira pas forcément à la meilleure solution. Par exemple :



Si on démarre de (A), aller toujours vers le voisin le plus proche nous fait faire un trajet (⟶) beaucoup plus long que le trajet optimal (⟶).

Heuristiques et méta-heuristiques

Une heuristique est spécifique au problème qu'elle traite : elle exploite certaines propriétés du problème pour orienter la recherche vers des « régions »susceptibles de contenir des bonnes solutions. Cependant, certaines heuristiques reposent sur des concepts assez génériques pour s'appliquer à de nombreux problèmes, moyennant quelques adaptations. On parle alors de **méta-heuristiques**. Les méta-heuristiques sont souvent inspirées de la nature. En voici quelques exemples.

Le recuit-simulé s'inspire d'un processus utilisé en métallurgie pour minimiser l'énergie d'un matériau.

Les colonies de fourmis s'inspirent du comportement des insectes sociaux : avez-vous remarqué que les fourmis finissent toujours par trouver le chemin le plus court entre la fourmilière et la source de nourriture ?

Les algorithmes génétiques reproduisent les mécanismes de l'évolution dans le vivant :

- une population de solutions aléatoires est créée ;
- on soumet cette population à une sélection naturelle (les meilleures solutions sont les plus adaptées) ;
- on crée de nouvelles solutions à partir des solutions existantes par croisements et mutations ;

Génération après génération, on observe une amélioration des solutions.

Le coin de l'animateur

L'objectif de cette activité est d'introduire la notion de complexité des problèmes.

- Commencez par donner un exemple de chemin, en faisant volontairement des détours. Laissez ensuite les participants trouver de meilleures solutions.
- À mesure qu'ils avancent, il sera de plus en plus difficile d'améliorer les résultats. Insistez sur le fait que lorsqu'ils trouvent une meilleure solution, ils ne peuvent pas être sûrs qu'il n'en existe pas une meilleure.
- Pour qu'ils se fassent une idée de la complexité du problème, vous pouvez demander aux participants de calculer le nombre de chemins possibles ($n!$), et les amener à refaire l'estimation du temps de calcul nécessaire à raison d'un milliard de chemins testés par seconde.