CHAPITRE 13

EXPLORATION D'ARBORESCENCES ET DE GRAPHES

Une grande quantité de problèmes, notamment en intelligence artificielle, se traduisent par l'exploration d'une arborescence, il est alors nécessaire de réfléchir à l'ordre de parcours de cette arborescence, et aux divers moyens de réduire l'exploration en "coupant des branches" par exemple, si l'on veut éviter ce que l'on appelle l'explosion combinatoire. Ce chapitre est probablement le plus profitable mais aussi le plus difficile.

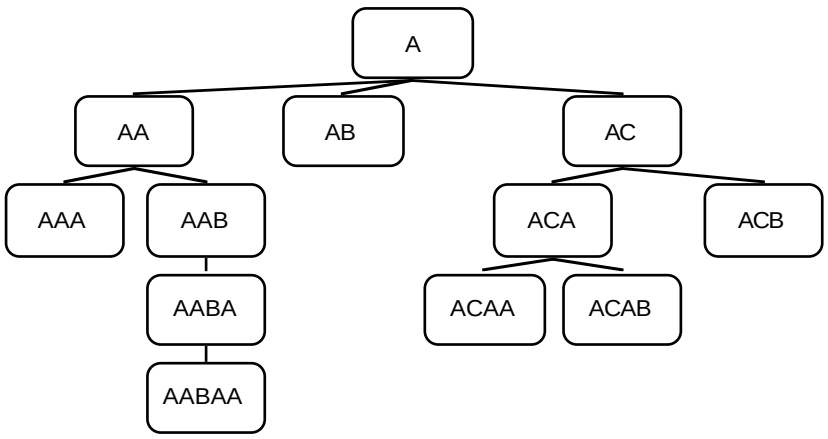
Exemple d'exploration en profondeur : les crocodiles et les moutons

N crocodiles sont sur la rive du Nil (cher aux lispiens) et possèdent une barque contenant au maximum K individus, ils doivent traverser ce fleuve pour atteindre l'autre rive sur laquelle se trouvent N moutons désirant, eux aussi, traverser.
Comment organiser les voyages, sachant que sur les deux rives comme dans la barque, il ne doit jamais y avoir plus (au sens strict) de crocodiles que de moutons ?

Ce problème (dont on ne sait pas s'il a une solution, ou plusieurs solutions dans le cas général) va se résoudre en examinant à chaque étape tous les voyages possibles, qui apporteront donc une modification de la répartition des animaux sur chaque rive, et pour chacun de ces nouveaux "états", on examinera encore tous les voyages possibles. On conçoit donc que, partant d'un état initial, tout un arbre d'états possibles va être développé.

Stratégie adoptée et représentation des données

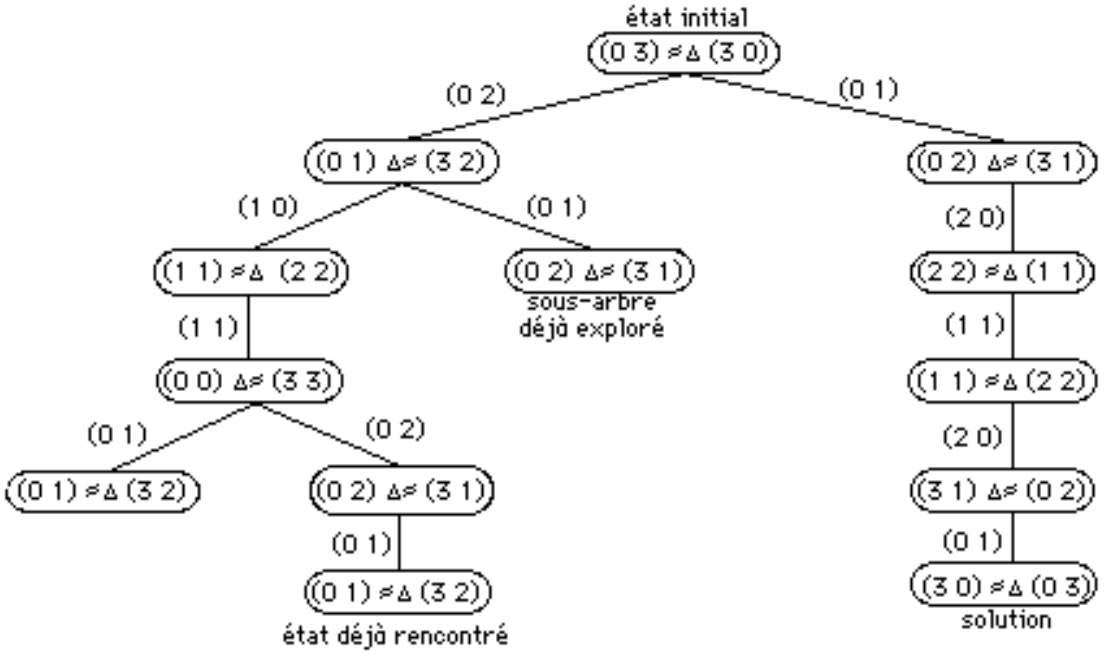
La statégie standard est celle de la lecture suivant l'ordre lexicographique ou racine-gauche-droite, dont il a déjà été question. Suivons ces 12 sommets dans cet ordre :



En suivant l'ordre du dictionnaire : A < AA < AAA < AAB < AABA < AABAA < AB < AC < ACA < ACAA < ACAB < AC, on a bien la même lecture.

Un état au cours de la résolution du problème se représentera naturellement par le couple des nombres de moutons M et crocodiles C sur la première rive, c'est-à-dire (0, N) au départ,

mais également par la position de la barque codée par Q = 1 pour la première rive et Q = -1 pour la seconde.
Les transports possibles forment une liste TR de couples (x, y) des nombres de moutons et crocodiles voyageant dans le bateau, liste que l'on pourra ordonner suivant x-y, ainsi :
TR = ((0 1) (1 0)) pour K= 1 , et :
TR = ((0 2) (0 1) (1 1) (1 0) (2 0)) pour K= 2
Lorsque la barque se trouve sur la rive initiale, il est intéressant d'essayer d'abord les voyages transportant le plus de crocodiles, par contre quand la barque est du côté initial des moutons, on tentera les voyages dans le sens inverse.
Voici l'illustration de la recherche pour N = 3 et K = 2 (le Δ représente la voile du bateau et ≈ est le flot du fleuve, les branches qui amènent à des états ne satisfaisant pas les contraintes ne sont pas indiquées).
Il se trouve que si on continue l'exploration après la découverte de la première solution, il n'y en a pas d'autres. La solution est donc constituée par la suite des voyages :
(0 1) (2 0) (1 1) (2 0) (0 1).



Programmation

L'ensemble TR des voyages est créé grâce à une fonction V comme voyage, de paramètre K telle que : V(1) = {(0, 1), (1, 0)} et V(K) = V(K-1) ∪ {(0 K)} ∪ {(x, y)| x+y = K et x ≥ y}
V(K) est ordonné suivant x - y, il est de cardinal (k+1) (k+2)/2.
On utilise exceptionnellement une boucle "while" avec deux variables locales W et P initialisées par un "let" :

```
(de V(K) (if      (eq K 1) '((0 1) (1 0))                    ; cas où K = 1, sinon :
                        (let      ((W (V(- K 1))) (P 1))  ; déclaration locale
                        (while    (>=  K P)       (if (>=  (* 2 P) K) (set 'W (ins (list P (- K P) ) W )))
                                                  (set 'P (+ P 1)))
                        (cons (list 0 K) W ))))
```

Ins est une fonction d'insertion d'un couple C = (x, y) dans une liste L suivant l'ordre de x-y

```
(de ins (C L) (cond
        ((null L) (list C))
        ((> (- (caar L) (cadar L)) (- (car C) (cadr C))) (cons C L))
        (t  (cons (car L) (ins C (cdr L))))))
```

Les paramètres utilisés dans la procédure générale, outre TR : liste des transports possibles, M et C nombre de missionnaires et cannibales sur la première rive, Q (1 ou -1) position de la barque, sont :
L désignant à chaque instant la liste des transports encore à essayer,
TP est la "pile" des transports précédents, elle permet de vérifier d'abord qu'on ne fait pas deux fois de suite le même transport, et surtout elle permet de revenir en arrière en cas d'échec.
EP est la liste des états antérieurs sur la première rive. Un état est un triplet (M, C, Q) ainsi (0 N 1) est le premier état. EP permet d'éviter de tourner en rond.
CO est un compteur de voyages.
L étant la liste des voyages à essayer, (car L) en sera le premier, (caar L) et (cadar L) seront les nombres x,y de missionnaires et cannibales prenant place dans la barque. Il faut alors respecter les contraintes suivantes :

a) $(x,y) \notin$ (car TP) ne pas refaire juste le voyage précédent.
b) $x = 0$ ou $x \geq y$ cette condition est toujours réalisée par construction de TR
c) si $Q = 1$ il faut $x \leq M$ et $y \leq C$ et si $Q = -1$ il faut $x \leq N-M$ et $y \leq N-C$ pour que le voyage (x,y) soit possible.
d) si $Q = 1$ il faut $x = M$ ou $M-x \geq C-y$ pour que la situation reste calme sur la première rive et $N-M+x = 0$ ou $N-M+x \geq N-C$ sur la deuxième rive. On écrit des conditions analogues si $Q = -1$ et la contrainte peut s'exprimer plus simplement par [$M = Qx$ ou $M-C \geq Q(x-y)$] et [$N = M+Qx$ ou $M-C \leq Q(x-y)$]
e) $(M-Qx, C-Qy, Q) \notin$ EP ne pas faire de boucle, condition est plus générale qu'en a)

```
(de vue () (print CO (list M C) Q)) ; est une fonction pour suivre à la trace les états

(de mouv (M C Q TR L TP EP CO) (cond
        ((and (eq M N) (eq C 0))  (vue) 'fini)     ; on est arrivé au résultat
        ((and (null L) (null TP)) 'impossible)     ; l'exploration de l'arbre est terminée
        ((null L)          ; cas où on doit remonter dans l'arbre (backtracking)
                (mouv (- M (*(caar TP) Q)) (- C (* (cadar TP) Q))
                        (- Q) (reverse TR) (cdr (member (car TP) (reverse TR)))
                ; on essaie le voyage qui suit celui ayant amené à une impasse
                        (cdr TP) (cdr EP) (- CO 1) ))
        ((or    (eq (car L) (car TP))                disjonction de toutes les contraintes
                (and (eq Q 1) (> (caar L) M))
                (and (eq Q 1) (> (cadar L) C))
                (and (eq Q -1) (> (caar L) (- N M)))
                (and (eq Q -1) (> (cadar L) (- N C)))
                (and (<> M (* (caar L) Q))
                (< (- M C) (* (- (caar L) (cadar L)) Q)))
                (and (<> (+ N (* (caar L) Q)) M)
                (< (* (- (caar L) (cadar L)) Q) (- M C)))
                (member (list (- M (* (caar L) Q)) (- C (* (cadar L) Q)) Q) EP))
                ; si l'une des contraintes n'est pas respectée on regarde le voyage suivant :
                (mouv M C Q TR (cdr L) TP EP CO))
        (t      (vue) ; visualisation du trajet et on descend dans l'arbre
                (mouv (- M (* (caar L) Q)) (- C (* (cadar L) Q)) (- Q) (reverse TR)
                        (reverse TR) (cons (car L) TP)
                        (cons (list (- M (* (caar L) Q)) (- C (* (cadar L) Q)) Q)  EP) (+ CO 1) ))))
```

Le programme sera lancé par (miss 3 2) par la fonction principale :

```
(de rives (N K) (mouv 0 N 1 (V K) (V K) nil (list (list 0 N)) 0 ))
```

Exécution pour N = 3 et K = 2 :

```
(rives 3 2)
0(0 3)1         1(0 1)-1        2(1 1)1         3(0 0)-1        3(0 0)-1        1(0 1)-1
2(0 2)1         3(0 0)-1        3(0 0)-1        1(0 1)-1        2(0 3)1         3(0 2)-1
4(2 2)1         5(1 1)-1        6(3 1)1         7(3 0)-1        = fini
```

Autres exemples :

(rives 3 1)	0(0 3)1	= impossible			
(rives 3 4)	0(0 3)1	1(0 0)-1	2(3 0)1	= fini	
(rives 4 2)	0(0 4)1	1(0 2)-1	2(2 2)1	1(0 2)-1	2(0 3)1
3(0 1)-1	4(1 1)1	5(0 0)-1	5(0 0)-1	3(0 1)-1	4(0 2)1
5(0 0)-1	5(0 0)-1	1(0 2)-1	2(0 4)1	0(0 4)1	= impossible
(rves 4 3)	0(0 4)1	1(0 1)-1	2(2 2)1	3(1 1)-1	4(4 1)1
5(4 0)-1	= fini				
(rives 5 2)	0(0 5)1	1(0 3)-1	2(0 4)1	3(0 2)-1	4(2 2)1
3(0 2)-1	4(0 3)1	5(0 1)-1	6(1 1)1	7(0 0)-1	7(0 0)-1
5(0 1)-1	6(0 2)1	7(0 0)-1	7(0 0)-1	1(0 3)-1	2(0 5)1
0(0 5)1	= impossible				
(rives 5 4)	0(0 5)1	1(0 1)-1	2(2 2)1	3(1 1)-1	4(5 1)1
5(5 0)-1	= fini				

Recherche A* d'une solution par tris des chemins dans un graphe

C'est un algorithme consistant à considérer à chaque instant une liste de chemin G.
Au départ G n'est que le chemin de longueur 0 constitué par la racine (l'état initial)
Jusqu'à ce que G soit vide ou que le but soit atteint, on regarde le premier chemin de la liste.
S'il arrive au but c'est fini (succès)
Sinon on le retire en le remplaçant par les chemins allant jusqu'à ses fils,
Toute la liste G est alors retriée par coûts croissants (branch and bound) ou bien par coût du chemin + estimation de la distance restante pour rejoindre le but.
De plus, si plusieurs chemins aboutissent au même noeud, on élimine de G tous ceux de coût supérieur.

Notations
$k^*(u, v)$ est le coût minimal d'un chemin entre les états u et v
$g^*(u)$ est le coût minimal de l'état initial à l'état u
$h^*(u)$ est le coût minimal de u à un état terminal
$f^*(u) = g^*(u) + h^*(u)$ est donc le meilleur coût d'une solution passant par u

Comme en chaque état u, ces fonctions ne sont pas connues, on tâche de les estimer, on estime notamment h^* par une "heuristique" h telle que $h = 0$ sur tous les états terminaux (h doit être "coïncidente"). On dit que :
h minorante ssi $h \leq h^*$ (sinon, il se nomme algorithme A). Si n est le nombre d'états, dans le pire des cas, la complexité est en 2^n, mais si h est minorante, A* a une complexité en n^2.
h est monotone ssi pour tous u et v $h(u) - h(v) \leq k(u, v)$, en ce cas A* a une complexité de l'ordre de n.

L'algorithme AO* (Martelli 1973)

Dans le cas d'un problème se traduisant en sous-problèmes par un graphe et/ou, c'est un algorithme de style A* de complexité coûteuse où chaque fois qu'un successeur d'un état est envisagé, celui-ci perturbe le coût de l'antécédent et donc des ancêtres. A chaque fois les coûts sont révisés et le tri se fait donc sur de nouveaux coûts.

Soit G un graphe et/ou sans circuits, contenant un état initial u_0, des états terminaux T et soit h une heuristique.
On souhaite trouver une solution partielle G' contenant u_0 et telle que pour tout état v de G', c'est une feuille ou bien il y ait un connecteur unique amenant à des fils dans G'. Si les feuilles de G' sont des états de T alors on a une solution complète.

On note par ailleurs I(u) l'ensemble des opérateurs qui peuvent s'appliquer en u et k (u, i) le coût de l'opérateur i.
Soit P l'ensemble des états engendrés mais dont on n'a pas encore développé les fils, P est $\{u_0\}$ initialement.
Q est l'ensemble de tous les états développés (initialement vide)
f(u) est le coût minimal provisoire de u (initialement G' = $\{u_0\}$ et $f(u_0) = h(u)$)
Tant qu'il y a dans G' des états pendants (non développés ou non terminaux) :

Soit u un sommet pendant de G' faire P ← P-{u}; Q ← Q + {u}
Si on ne peut pas décomposer u alors f(u) est infini
sinon Pour tout i ∈ I(u) et tout état v obtenu par i(u) (i est un connecteur et/ou)
P ← P+{v}; f(v) = h(v)
Calculer $f_i(u) = k(u, i) + \sum\{f(v) / v \in i(u)\}$
f(u) est le min des $f_i(u)$ pour i ∈ I(u)
Soit A l'ensemble des pères de u
Pour tout v dans A sans successeur dans A
A ← A-{v} on recalcule $f(v) = \min \{k(v, i) + \sum\{f(w) / w \in i(v)\} / i \in I(v)\}$
A ← A+ pères(v) (et on modifie donc f pour les ascendants de v)

L'algorithme minimax pour un jeu à deux joueurs

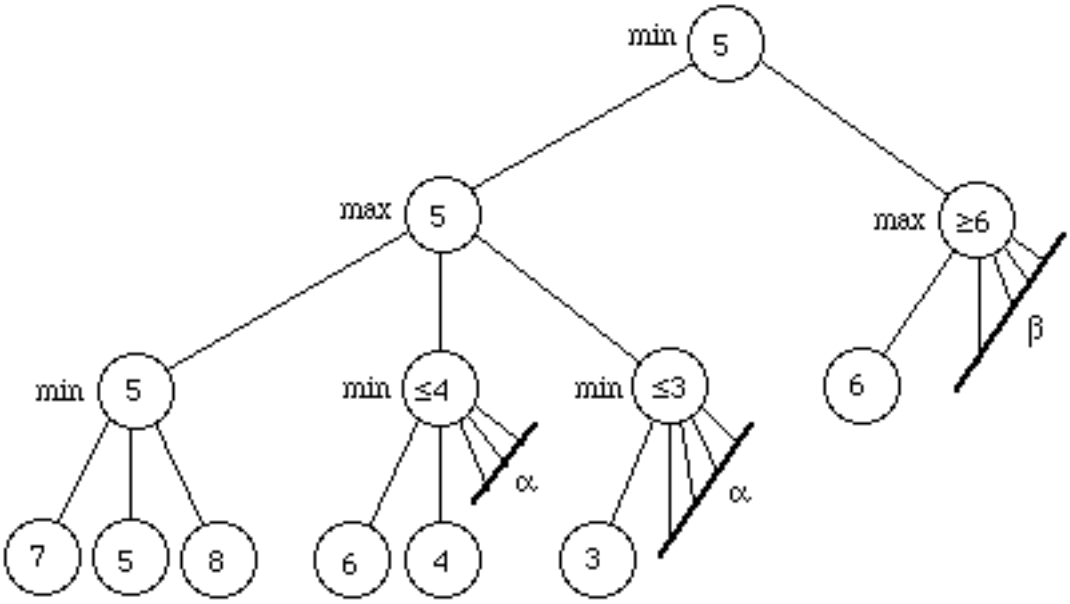
Cet algorithme s'applique pour tout jeu où deux joueurs choisissent un coup parmi plusieurs coups possibles, et ceci à tour de rôle. Si les chances d'un "état" du jeu sont évalués par une certaine fonction heuristique mesurée positivement lorsqu'il favorise un des joueurs (nommé max) et négativement pour son adversaire (nommé min) alors l'algorithme consiste à faire un choix sur la plus grande valeur d'une fonction "minimax" lorsque c'est à max de jouer, (car c'est ce qu'il jouerait pour gagner ou perdre le moins) et sur la plus petite (algébriquement, car c'est ce que min jouerait pour perdre le moins) lorsque c'est à min de jouer.
Cette fonction est définie récursivement par ce qui vient d'être dit, et par l'heuristique lorsqu'elle arrive sur un état terminal ou au niveau de profondeur fixé à l'avance.

Plus précisément :

fonction minimax (e : etat, j : joueur, n : niveau de profondeur souhaité)
retourne : une valeur permettant de classer les coups à jouer
si e est un succes pour max, retourne +∞
si e est un succes pour min, retourne -∞
si e est un match nul (le jeu est bloqué), retourne 0
si n = 0, minimax = heuristique(e)
si joueur = max alors minimax = max{minimax(e', min, n-1) / e' coup pour max à partir de e}
sinon (joueur = min) minimax = min{minimax(e', max, n-1) / e' coup pour min à partir de e}

Les coupes alpha-beta dans le calcul du minimax

Aux étapes min, si un noeud est connu, dès que l'on rencontre un neveu inférieur, il est inutile d'explorer les autres neveux car leur père aura donc un min plus petit que celui de son frère déjà connu et donc il ne sera pas choisi par max à l'étape supérieure. On peut donc élaguer l'arbre, c'est à dire ne pas explorer certaines branches ("coupe alpha"). De même on fera une "coupe beta" à l'étage de max si min, encore au dessus, doit choisir le plus petit.
Plus précisément la fonction minimax à calculer est alors entâchée par deux paramètres supplémentaires α et ß représentant les max et min provisoires (ce qu'ils joueraient). En un état donné e_0, pour un joueur donné j et un niveau de profondeur donné n, le joueur va choisir le coup en fonction de la valeur de f (e, n, adv (j), -∞, +∞) sur chacun de ses fils.



α représente ce que le joueur max peut au moins espérer (max provisoire), et β (min provisoire), ce que min peut obtenir au plus, ils sont donc des bornes inférieure pour un noeud max et borne sup pour min. Le résultat théorique est que si on a c fils à chaque noeud, le nombre de noeuds explorés est c^n pour le minimax et en moyenne$2c^{n/2}$ par αβ.

fonction f (e, n, j, α, ß) =
si max gagne en e alors +∞
si min gagne en e alors - ∞
si match nul en e alors 0
si n=0, f est la valeur de l'heuristique h choisie sur e

si j = max alors	soit m ← α, et pour chaque fils e' de e jusqu'à m ≥ ß
	m ← max (m, f(e', n-1, min, m, ß)
	retourner m
sinon (j = min)	soit m ← ß, pour chaque fils e' de e jusqu'à α ≥ m
	m ← min (m, f(e', n-1, max, α, m)
	retourner m

13-1° Exprimer V(3) et V(4).

13-2° Trouver au moins deux améliorations au programme des missionnaires, en évitant par exemple d'évaluer deux fois (V K), puis surtout de n'appeler "reverse" qu'une seule fois grâce à un paramètre supplémentaire, élaguer les branches déjà explorées etc...

13-3° **Les dames de Gauss :** sur un échiquier N*N, placer N dames qui ne soient pas en position de prise (une par ligne, une par colonne, au plus une pour chaque transversale). Il y a 92 solutions pour N = 8.

13-4° **Problème des camps de base :** Pour aller de A_0 (départ) à A_4 (le sommet), il est nécessaire de passer en A_1, A_2 et A_3 espacés d'un jour de marche. Il faut par contre deux jours de marche de A_3 à A_4. L'équipe ne peut porter que des vivres pour 3 jours. Quel est le nombre minimal de jours pour atteindre le sommet et quelle est la solution détaillée sachant que l'on va déposer des vivres aux camps intermédiaires ? On pourra représenter un voyage par une liste de couples (A_i, q_i) où q_i est la quantité (en nombre de jours) de vivres prise au lieu A_i, donc négative s'il s'agit d'un dépôt.

13-5° Une mini-grammaire en chaîne: On suppose acquise l'analyse des mots délivrant plusieurs catégories syntaxiques pour chacun, par exemple la phrase "je le porte" produit la liste (PRO (DET PRO) (NOM V)). On note V les formes verbales, INF les "inflexions" (qui, que, quoi, dont, où), qui ouvrent une subordonnée, PRO les pronoms, DET les déterminants et PREP les prépositions (de, du ...) Il faut maintenant obtenir une liste plate (PRO PRO V), c'est-à-dire lever les ambiguïtés d'interprétations syntaxiques. On dispose pour cela d'une liste de juxtapositions interdites qui permet de résoudre le problème. Les interdictions sont un DET suivi d'un DET, PREP, INF, CONJ, ADV, PRO, V, puis une PREP suivi de V, (INF PREP), (INF INF), (INF CONJ), (INF NOM), (INF ADJ), (ADV NOM), (PRO DET), (PRO NOM), (PRO ADJ), (NOM DET), (NOM NOM), (ADJ DET), (V V). Il faut trouver un chemin du début à la fin de la phrase en évitant ces 21 couples impossibles (en fait procéder à des éliminations dans un produit cartésien d'ensembles). Ceux-ci ne résolvent pas toutes les ambiguïtés, mais font déjà plus de la moitié du travail. On pourra faire intervenir par la suite les PPS et PPT.

Solution : F explore l'arborescence des possibilités en coupant au fur et à mesure des branches qui donnent une interdiction. L est la liste entrée telle que L = ((DET) (DET PRO) (NOM V) (DET) (ADV V PREP)), Q est le chemin parcouru, X et Y les deux éléments qui suivent, LR tout ce qui les suit. XR et YR sont les possibilités non encore examinées dans X et Y. LP est la liste parallèle à Q des possibilités restantes. BI est la base des couples interdits.

```
(de F (L LP Q X XR Y YR LR) (cond
        ((null XR) (if (null Q) nil ; échec tout était impossible
                       (F (cdr L) (cdr LP) (cdr Q) (car L) (cdr (member (car Q) (car L))) X X (cons Y LR)) ))
        ((member (list (car Q) (car XR)) BI) (F L LP Q X (cdr XR) Y Y LR)) ; frère suivant
        ((null YR) (F L LP X (cdr XR) Y Y LR))
        ((member (list (car XR) (car YR)) BI) (FL LP Q X XR Y (cdr YR) LR))
        ((null LR) (cons (reverse (mcons (car YR) (car XR) Q)) ; on a trouvé une solution
                       (F L LP Q X XR Y (cdr YR) LR)))
        (t (F (cons X L) (cons (cdr XR) LP) ; on descend dans l'arbre
              (cons (car XR) Q) Y YR (car LR) (car LR) (cdr LR))) ))
(de des (L) (F nil nil nil (car L) (car L) (cadr L) (cadr L) (cddr L))) ; desambiguisation
```

Exemples :

```
(des '((det pro) (nom) (pro) (v nom) (prep) (det )(v nom)))      ; "Le soir il lit dans son lit"
= ((det nom pro v prep det nom))                                 ; succès : une seule interprétation
(des '((pro) (det pro) (nom v) (adv nom)))                       ; "Je le porte bien"
= ((pro pro v adv) (pro pro v nom))                              ; 2 interprétations sont possibles
(des '((det pro) (adj nom) (v nom) (det pro) (v nom)))           ; "La belle porte le voile"
= ((det adj v det  nom) (det adj v pro v) (det adj nom pro v) (det nom v det nom) (det nom v pro v))
```

5 interprétations, dont deux en fait sont attestables seulement. On voit qu'il faudrait regarder les triplets interdits, la présence d'un verbe etc... Essayer aussi "Bonne nouvelle ! On a une nouvelle bonne !" Par contre il n'y a qu'une possibilité pour : "La commission conduit ces négociations en consultation avec un comité spécial désigné par le conseil pour l'assister dans cette tâche, et dans le cadre des directives que le conseil peut lui adresser".

13-6° Minimax en profondeur non limitée Réaliser l'évaluation minimax d'un arbre dont les noeuds sont exactement des valeurs numériques : si le noeud est une feuille, on renvoie sa valeur, s'il a une profondeur paire, on renvoie le max des valeurs de ses fils, et s'il est à une profondeur impaire, on renvoie le min de ses fils (la racine est au niveau 0). Programmer la valeur d'un tel arbre. Dessiner et calculer la valeur de ((((4 5 (1 6 2)) (7 3)) ((0 9 8) 2) (((1 4 0) 5 6) 4))).

```
(de valmax (A) (if (atom A) A (apply 'max (mapcar 'valmin A)))) ; à utiliser à la racine
(de valmin (A) (if (atom A) A (apply 'min (mapcar 'valmax A))))
        ; ou alors à condition que le joueur j soit appelé max ou min :
(de minimax (a j) (if (atom a) a (apply j (mapcar (lambda (x) (minimax a (if (eq j 'max) 'min 'max))) a)))
```

13-7° Problème du professeur Moiron : recouvrement d'un graphe par des cycles. Si G est un graphe non orienté possédant S sommets, A arètes, le nombre de cycles d'un recouvrement du graphe est majoré par le nombre "cyclotomique" A-S+1. On cherche un algorithme donnant exactement A-S+1 cycles trouvés dans ce graphe et dont l'union est G.

Solution :
On numérote les sommets de 1 à S et à chaque sommet est associé la liste des sommets qui lui sont reliés mais uniquement ceux de numéros supérieurs. Par exemple G = ((1 2 3 6 10) (2 3 5) (3 5) (4 7 9) (5 7) (6 7 10) (7) (8 9 10) (9) (10)) sera tel que 1 est relié à 2, 3, 6, 10 et 3 à 1, 2, 5.
On isole le premier sommet et la première arète qui en part. On tente de poursuivre le chemin en revenant à 1. Quand un cycle est trouvé on poursuit avec l'arète suivante. S'il ne reste qu'une arète, (sur k partant de 1) on réunit les k-1 cycles trouvés aux (A-k) - (S-1) +1 cycles du graphe G-{1}.
On note Q un chemin, RX les candidats suivants, et LR la listes des éventualités suivantes.
On signale que "assq" X L donne la première sous-liste débutant par X dans la liste d'associations L, et "cassq" son cdr.

```
(de clef (R G) (cond                    ; liste des clefs des sous-listes de G contenant R
        ((null G) nil)                  ; (les voisins d'indices inférieurs de R)
        ((member R (cdar G)) (cons (caar G) (clef R (cdr G))))
        (t (clef R (cdr G))) ))
(de suiv (R G) (append (cassq R G) (clef R G)))    ; tous les voisins de R dans G
(de F1 (G) (cond
        ((null (cddr G)) nil) ; moins de 3 sommets
        ((null (cddar G)) (F1 (cdr G)))             ; 1 seule arète part du premier sommet
        (t (F2 G nil)))) ; départ avec encore aucun cycle
(de F2 (G C) (F3 G C (list (cadar G) (caar G)) (suiv (cadar G) G) (list (cddar G))))
(de F3 (G C Q RX LR) (cond
        ((null RX) (cond          ; pas de sommet suivant
                ((null LR) (append C (F1 (cdr G))))              ; tous les chemins de 1 ont été explorés
                ((null (car LR)) (F3 G C (cdr Q) nil  (cdr LR)))          ; backtrack de 2 niveaux
                ((member (caar  LR) Q) (F3 G C Q nil (cons (cdar LR) (cdr LR)))) ; point double
                (t (F3 G C (cons (caar LR) (cdr Q)) (suiv (caar LR) G)
                        (cons (vdar LR) (cdr LR))))))            ; backtrack d'un niveau
        ((member (car RX) Q) (F3 G C Q (cdr RX) LR))        ; point double
        ((member (car RX) (cddar G)) (cond                  ; un cycle peut être formé
                ((member (append Q (list (car RX))) C) (F3 G C Q (cdr RX) LR)) ; cycle déjà vu
                (t (F2 (mcons (caar G) (cddar G) (cdr G)) (cons (append Q (list (car RX))) C)))))
                ; on repart avec une arète en moins du sommet 1 et un cycle en plus
        (t (F3 G C (cons (car RX) Q) (suiv (car RX) G) (cons (cdr RX) LR)))))
        ; pas de cycle achevable, on continue de descendre
```

Exemple (faire un dessin du graphe) :
(F1 '((1 2 4 11) (2 3 4 5) (3) (4 5 6 11) (5 11) (6 7 12) (7 8) (8 9) (9 12) (10 11 13) (11 14) (13 14) (14)))
= ((1 4 5 11) (1 2 4 5 11) (2 4 5) (4 5 11) (6 7 8 912) (10 11 114 13))

Soit 19-14+1 = 6 cycles.

13-8° Règles de suffixation en français :
Etant donné un dictionnaire de mots primitifs tel que DICO = (an ane brouette capital colonie culte declamer electrique forme gris labeur monstre note provoquer remarque tracter) et une liste de règles de suffixations à définir de façon informelle puis suivant une représentation rationnelle, concevoir un programme donnant pour un mot dérivé tel que "formativité", une chaîne de mots le reliant si cela est possible, à un mot primitif, par exemple forme → former → format → formatif → formativité. Le but est de pouvoir appréhender des néologismes inconnus, lors de la lecture d'un texte.

Solution :
Après recherche, on va définir les règles suivantes où une terminaison (exemple "ation") peut être obtenue à partir d'une autre par retrait de "er" puis rajout de 5 lettres "ation" ou bien par retrait de "quer" puis rajout de "cation". On donne quelques exemples.

```
(set 'BS '(
(eur (2 r) (3 ion) (4 er))                              ; rédaction -> rédacteur, distribuer -> distributeur
(ique (3 e) (3 sme) (4 ation) (3 on) (4) (5))           ; hystérie -> hystérique, numération -> numérique
(ation (5 er) (cation quer) (6 er))                     ; capituler -> capitulation, abdiquer -> abdication
(ier (3 e) (3) (4) (3 te) (3 erie) (ifier ique) (ifier e)) ;
(ité (3 e) (3) (ilité le) (vité f) (arité ier) (osité eux) (orité eur)      ; agile -> agilité, actif -> activité
        (alité el) (cité que) (nité in) (rité ire))     ; formel -> formalité
(ter (2) (1) (3))                       ; paquet -> paqueter, épouvante -> épouvanter, abri -> abriter
(age (3 er) (cage quer))                ; plaquer -> placage
(able (4 er) (cable quer))              ; avouer -> avouable, manger -> mangeable
(iste (4 er) (4) (aliste el) (oriste eur) (5) (3 e) (riste ire))    ; garage -> garagiste, dent -> dentiste
(eux (2) (ueux e) (3 on) (2 er) (4 ie))         ; haine -> haineux, ambition -> ambitieux, soie -> soyeux
(aire (4 é) (4) (5))                            ; utilité -> utilitaire, fonction -> fonctionnaire,
(rie (2) (erie) (3) (terie))                    ; trésor -> trésorerie, flatter -> flatterie, billet -> billeterie
(isme (4 e) (alisme el) (visme f) (4) (5) (4 er) (5 in) (6 en) (2 te) (2 er))    ; formel -> formalisme
(if (2 e) (2) (1 on) (2 eur))                   ; action, acteur, acte -> actif
(ion (3 er) (ition er) (ction ire) (tion r))    ; inverse -> inversion, poser -> position
(al (2 e) (2) (tal) (cal que))          ; ogive -> ogival, instrument -> instrumental, horizon -> horizontal
(ien (3 e) (1) (3) (cien que))          ; galère -> galèrien, lilliput -> lilliputien, zambie -> zambien
(iser (4 e) (3 e) (3 que) (oriser eur) (iliser le) (aliser el) (riser ire) (niser in) (4 ais) (3 te))
(el (iel e) (3) (2) (1) (tiel ce) (4))                  ; an -> annuel, accident -> accidentel
(at (ariat aire) (orat eur) (cat quer) (2 er) (2))      ; docteur -> doctorat, mander -> mandat
(oire (4 er) (4 ion) (4 e) (4) (5))                     ; vibrer -> vibratoire, noter -> notoire
(er (1) (2) (3) (iver e))                               ; hiver -> hiverner, motif -> motiver
(aille (5 er) (5)) ))                                   ; trouver -> trouvaille, valet -> valetaille
```

Bien sûr ces règles peuvent être redondantes, circulaires et de toutes façons incomplètes.

```
(de deb (M D) (cond       ; teste si D est un début du mot M et renvoie la suite dans M
        ((null D) M)
        ((eq (car L) (car M)) (deb (cdr M) (cdr D)))
        (t nil)))
(de fin (M F) (cond       ; teste si F est une terminaison de M et renvoie le mot tronqué
        (reverse (deb (reverse M) (reverse F))))
(de ret (N M) (reverse (nthcdr N (reverse M))))     ; retire les N dernières lettres de M
(de regle (X B) (cond     ; donne la règle faisant partie de la base B pouvant s'appliquer à la liste X
        ((null B) nil)
        ((fin X (explode (caar B))) (cdar B))     ; on regarde la terminaison de X
        (t (regle X (cdr B)))))

(de uf (X P) (cond        ; renvoie pour un mot X et une prémisse P le mot transformé
        ((null (caddr P)) (cond
                ((numberp (car P))  ( if        (null (cadr P)) (ret (car P) X))
                                                (append (ret (car P) X) (explode (cadr P)))))
                ((null (cadr P)) (fin X (explode (car P))))
                ((fin X (explode (car P))) (append (fin X (explode (car P))) (explode (cadr P))))
                (t nil)))
        (t (eval (list (caddr P) (append (ret (car P) X) (explode (cadr P)))))) ))

(de F (Q X RR R RX) (cond       ; pour un chemin Q suivi de X, RX la liste des possibilités s'offrant à X,
        ; RR est les morceaux de règles restant à examiner
        ((member (if (null X) X (implode X)) DICO) (mapcar 'implode (cons X Q)) ; succès, c'est fini
        ((or (null RX) (member X Q) (null X)) ; plus de choix pour X
                (if     (null R) (if    (null (cdr RR)) nil     ; échec, fini, sinon double backtrack
                                (G (cdr Q) (uf (cadr Q) (caadr RR)) (cons (cdadr RR) (cddr RR)))
                        (G Q (uf (car Q) (car R)) (cons (cdr R) (cdr RR))))     ; simple backtrack
        (t (G (cons X Q) (uf X (car RX)) (cons (cdr RX) RR)) )))                ; on descend
```

```
(de G (Q X R) (F Q X R (car R) (regle X BS))) ; pour lancer F
(de suf (X) (F nil (explode X) nil nil (regle (explode X) BS))) ; fonction principale
```

Exemples :

```
(suf 'remarquable)
= (remarque remarquer remarquable)
(suf 'provocation)
= (provoquer provocation)
(suf 'tractorisme)
= (tracter tracteur tractoriste tractorisme)
(suf 'colonisateur)
= (colonie coloniser colonisation colonisateur)
(suf 'notariat)
= (note notaire notariat)
(suf 'laboratoire)
= (labeur laborat laborater laboratoire) ; on voit là des intermédiaires non attestés
```

Avec les préfixes :

```
(set 'BP '(in de dé re exe con en pro im des pre com inter dis di sur trans contre anti mal tri auto epi para per abs ab ad a télé par))

(de H  (Q X Y PR PQ) (if ; cherche la dérivation par préfixation et suffixation
        (null Y) (cond
                ((null PR) (if      (null Q) nil
                                    (H (cdr Q) (car Q) nil (car PQ) (cdr PQ))))
                ((deb X (explode (car PR))) (H          (cons X Q) (deb X (explode (car PR)))
                                                        (suf (implode (deb X (explode (car PR)))))
                                                        BP (cons (cdr PR) PQ)))
                (t (H Q X Y (cdr PR) PQ)))
        (append Y (mapcar 'implode Q)) ))

(de der (X) (H nil (explode X) (suf X) BP)) ; fonction principale

(der 'réformateur) = (forme former format formater formateur réformateur)
(der 'désinformation) = (forme former formation information désinformation)
(der 'incultivable) = (culte cultiver cultivable incultivable)
```

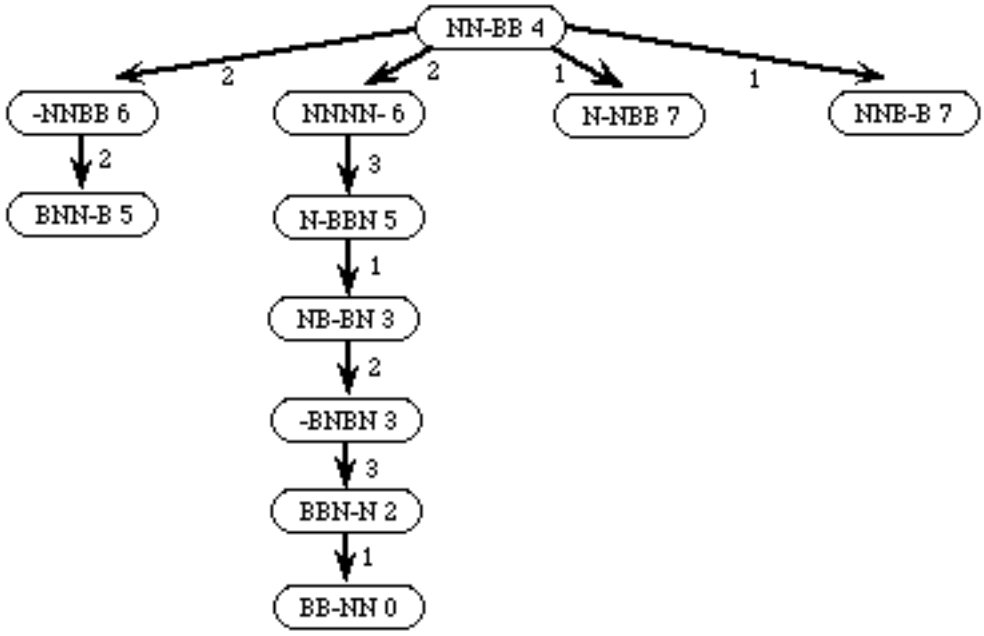
13-9° Le mini-taquin (Sam Lloyd). L'algorithme A* est adapté à tous les problèmes du genre de celui du voyageur de commerce, on prend par exemple le mini-taquin où n pions noirs et n pions blancs sont alignés séparés par un emplacement vide. Le but du problème est d'arriver à la situation symétrique en faisant, soit glisser un pion dans la case voisine si elle est vide (coût 1), soit en le faisant sauter 1 ou 2 pions (coût 2 ou 3) pour arriver dans la case vide.

Si n = 2 on a Départ : NNvBB But : BBvNN

Tout d'abord, la fonction de tri, on reprend un algorithme de tri par fusion déjà vu afin de trier une liste de listes suivant une clé se trouvant en tête de chacune de ces listes.

```
(de fusion (L1R L2R LF) (cond    ; chaque élément de LR est de la forme (clé ...code de l'état...)
        ; version récursive terminale
        ((null L1R) (if (null L2R) (reverse LF) (fusion nil (cdr L2R) (cons (car L2R) LF))))
        ((null L2R) (fusion nil (cdr L1R) (cons (car L1R) LF)))
        ((< (caar L1R) (caar L2R)) (fusion (cdr L1R) L2R (cons (car L1R) LF)))
        (t (fusion L1R (cdr L2R) (cons (car L2R) LF))) ))
(de sep (pair LP LI LR)   ; liste des éléments de rangs pairs, impairs et restant à séparer
        ; l'argument "pair" ou "impair" est inutile en inversant les arguments
        (if LR (sep (cons (car LR) LI) LP (cdr LR))
        (cons LP LI)) )          ; renvoie, lorsque LR est vide, ((a0 a2 a4 ...) a1 a3 a5 ....)
(de tri (L) (if (null (cdr L)) L (tribis (sep t nil nil L))))
(de tribis (LC) (fusion (tri (car LC)) (tri (cdr LC)) nil))
```

On représente ci-dessous les états explorés (ceux déjà vus ne sont pas réexaminés), le coût est porté sur chaque arc.
On va représenter chaque état du problème par une liste telle que (3 N B B v N) où le premier élément une définition d'heuristique : la distance au but, au sens du nombre d'éléments non égaux, dans l'ordre à ceux du but. On pourrait prendre le nb de noirs à gauche d'un blanc + le nb de noirs à gauche du second blanc, ce qui ferait h(nnbvn) = 4 et h(nbbvn) = 2.




```
(de distance (L n)  (dist L 1 n 0))
(de dist (L k n d) (cond ; compte la somme des "éloignements" des b et n mal placés par rapport au milieu
        ((null L) d)
        ((< k (1+ n)) (cond     ((eq 'b (car L)) (dist (cdr L) (1+ k) n d))
                                ((eq (car L) 'n) (dist (cdr L) (1+ k) n (- (+ d n 1) k)))
                                (t (dist (cdr L) (1+ k) n (1+ d)))))
        ((eq k (1+ n)) (if (eq (car L) 'v) (dist (cdr L) (1+ k) n d) (dist (cdr L) (1+ k) n (1+ d))))
        ((eq (car L) 'b) (dist (cdr L) (1+ k) n (1- (- (+ d k) n))))
        ((eq (car L) 'n) (dist (cdr L) (1+ k) n d))
        (t (dist (cdr  L) (+ 1 k) n (1+ d))) ))
```

On construit à présent les fils d'un noeud (p a1 a2 a3 ...) sous forme d'au plus 6 états ordonnés

```
(de transpo (L i j) (cond ; renvoie L où les éléments i<j ont été transposés
        ((< j i) (transpo L j i))
        ((< 1 i) (cons (car L) (transpo (cdr L) (1- i) (1- j))))
        ((eq i 1) (cons  (nth (1- j) L) (append (firstn (- j 2) (cdr L)) (cons (car L) (nthcdr j L)))))))
```

Calcul des successeurs d'un noeud

```
(de fils1 (E) (fils2 E E 0 nil 0)); donne la liste des fils de l'état E, chacun étant précédé de son coût

(de fils2 (E ER r  vide l) (cond                ; r sera le rang de "vide" (à partir de 1) et l la longueur de E
        ((null ER) (fils3 E r l -3 nil))      ; on débute les transpositions 3 cases avant la vide
        ((eq (car ER) 'v) (fils2 E (cdr ER) (1+ r) t (1+ l)))
        (vide (fils2 E (cdr ER) r t (1+ l)))
        (t (fils2 E (cdr ER) (+ 1 r) nil (1+ l)))))
(de fils3 (E r l k F) (cond ; pour l'état E renvoie la liste F des fils obtenus en transposant à ±k la case vide
        ((or (eq k 4) (< l (+ r k))) F)       ; on va voir 3 places max après la case vide
        ((eq k 0) (fils3 E r l 1 F))
        ((< (+ r k) 1) (fils3 E r l (1+ k) F))
        (t (fils3 E r l (+ 1 k) (cons (cons (abs k) (transpo E (+ r k) r)) F))) ))
```

```
(de present (E CH) (cond            ; vérifie la présence d'un état au sein d'un chemin
        ((null CH) nil)
        ((equal (cdar CH) (cdr E)) t)
        (t (present E (cdr CH)))))
```

Recherche d'un chemin arrivant au but par l'algorithme A*

Un chemin est une suite d'états précédés de leur coût total (somme des arêtes) et un état est précédé de sa distance.

```
(de chemin (n) (chemin0 (cons (* 2 n) (append (compte n 'N) (cons 'v (compte n 'B))))
                (append (compte n 'B) (cons 'v (compte n 'N))) n)) ; construit les états de départ et d'arrivée
(de compte (n x) (if (eq n 0) nil (cons x (compte (- n 1) x)))) ; produit une liste formée de n fois la valeur de x

(de chemin0 (dep but n) (chemin1 (list (list 0 dep)) (list 0 dep) but n))
(de chemin1 (G CH but n) ; (print ch )              ; G est une liste de chemins CH en est le premier
        (if (equal (cdadr CH) but) (car G)          ; succès
                (chemin2 (fils1 (cdadr CH)) CH (cdr G) but n)))
(de chemin2 (LF CH G but n) (chemin3 (bontri (distrib LF CH G n)) but n))
        ; rajoute les fils LF de CH à G en éliminant les superflus et en triant
(de chemin3 (G but n) (chemin1 G (car G) but n))

(de distrib (LF CH G n) (cond       ; rajoute à G tous les chemins issus de CH par la liste LF des successeurs
        ((null LF) G)               ; renvoie une liste de chemins
        ((present (car LF) (cdr CH)) (distrib (cdr LF) CH G n))
        (t (distrib (cdr LF) CH (distrib2 (car LF) (car CH) (cdr CH) nil G n) n)) ))
; chaque noeud du chemin est précédé de sa distance au but, le chemin est précédé de son coût
(de distrib2 (F c CH LV LR n) (cond ; renvoie un nouveau G où CH+F est rajouté et excluant l'éventuel chemin
                ; plus coûteux arrivant à F, CH est une suite d'états dont c est le coût
        ((null LR) (cons (mcons (+ c (car F)) (cons (distance (cdr F) n) (cdr F)) CH) LV))
        ((equal (cdr F) (cdadar LR)) ; (print 'déjàvenu) ; cas où on est déjà venu en F
                (if  (< (+ c (car F)) (caar LR))
                        (append LV (cons (mcons (+ c (car F))  ; on ajoute le coût du fils
                                (cons (distance (cdr F) n) (cdr F)) CH) (cdr LR))  )
                        (append LV LR)))        ; cas où le chemin arrivant à F était meilleur
        ((present F (cdr CH)) (append LV LR))           ; cas où F déjà présent au sein d'un autre chemin
        (t (distrib2 F c CH (cons (car LR) LV) (cdr LR) n) )))

(de bontri (G) ; trie suivant une heuristique
        (mapcar 'cdr (tri (mapcar (lambda (x) (cons (heurist (car x) (caadr x)) x)) G))))
(de heurist (cout dist) (+ (* 1 cout) (* 3  dist)))     ; est une proposition certes discutable
```

Execution

(chemin 1) = (4 (0 b v n) (2 v b n) (3 n b v) (2 n v b))
(chemin 2) = (12 (0 b b v n n) (2 b v b n n) (3 b n b n v) (3 b n v n b) (5 b n n v b) (6 v n n b b) (4 n n v b b))

Remarque : 11 chemins ont été examinés, en prenant une heuristique c+2d au lieu de c+5d il en a fallu 16, et si on donne un plus grand poids à c avec par exemple 2c+d, on arrive à la même solution après l'examen de 20 chemins.

(chemin 3) = (24 (0 b b b v n n n) (2 b b b n v n n) (3 b v b n b n n) (3 v b b n b n n) (4 n b b v b n n) (6 n b v b b n n) (7 n b n b b v n) (8 n b n b v b n) (8 n b n b n b v) (9 n b n v n b b) (11 n b n n v b b) (12 n v n n b b b) (6 n n n v b b b))
(chemin 4) = (46 (0 b b b b v n n n n) (2 b b b v b n n n n) (3 b b b n b n v n n) (3 b b b n v n b n n) (5 b b b n n v b n n) (6 b b v n n b b n n) (6 b v b n n b b n n) (7 b n b n v b b n n) (8 b n b n b v b n n) (8 b n b n b n b v n) (9 b n b n v n b b n) (11 b n v n b n b b n) (13 b n n n b v b b n) (13 b n n n b n b b v) (14 b n n n b n b v b) (15 b n n n v n b b b) (15 b n v n n n b b b) (15 v n b n n n b b b) (16 n v b n n n b b b) (17 n n b n v n b b b) (19 n n b n n v b b b) (20 n n v n n b b b b) (8 n n n n v b b b b))

Pour 5 blancs et 5 noirs on obtient une solution en 30 mouvements.

13-10° Reprendre le mini-taquin en cherchant cette fois en profondeur dans l'arbre, mais en profondeur limitée.

Le chemin en cours est CH, LF est la liste parallèle des frères non encore explorés, F les fils de CH. On conserve les fonctions de tri, fils, heuristique et on trie cette fois les fils suivant une heuristique telle que "coût de la dernière étape + 3*distance au but"

```
(de distance (L1 L2 d) (cond ; départ avec L1 L2 0
        ((null L1) d)
        ((eq (car L1) (car L2)) (distance (cdr L1) (cdr L2) d))
        ((eq (car L1) 'v) (distance (cdr L1) L2 d))
        ((eq (car L2) 'v) (distance L1 (cdr L2) d))
        (t (distance (cdr  L1 ) (cdr L2) (+ 1 d)))))
```

Tri des fils

```
(de heurist1 (F CH n) (cond ; sur chacun des fils  "distance au but+cout" n'étant pas ancêtres
        ((null F) nil)
        ((present (car F) CH) (heurist1 (cdr F) CH n))
        (t (cons (cons (heurist (caar F) (distance (cdar F) n)) (cdar F)) (heurist1 (cdr F) CH n)))))

(de fils (CH n); calcule et ordonne la liste des fils de la tete du chemin CH
        (tri (heurist1 (fils1 (cdar CH)) CH n)))
```

Exploration en profondeur limitée

Le chemin en cours est CH, LF est la liste parallèle des frères non encore explorés, F les fils de CH. Le dernier état trouvé est la tete de CH se trouvant à la profondeur niv.

```
(de explo (CH CF F niv lim but nd n) (cond
        ((equal (cdar CH) but) CH); cas d'un succès
        ((or (null F) (< lim niv)) (bak (cdr CH) (cdr CF) (car CF) (1- niv) lim but nd n))
        (t (explo (cons (car F) CH) (cons (cdr F) CF) (fils (cons (car F) CH) n) (1+ niv) lim but (1+ nd) n))))

(de bak (CH CF FR niv lim but nd n) (cond ; FR fils restants au niveau niv+1
        ((null CH) nil); échec
        ((null FR) (bak (cdr CH) (cdr CF) (car CF) (1- niv) lim but nd n))
        (t (explo (cons (car FR) CH) (cons (cdr FR) CF) (cdr FR) (1+ niv) lim but (1+ nd) n)) ))
```

Initialisation

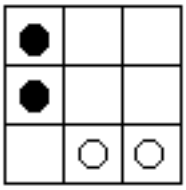
```
(de machin (n lim) (machin1 lim (cons (* 2 n)(append (compte n 'N) (cons 'v (compte n 'B))))
                        (append (compte n 'B) (cons 'v (compte n 'N))) n))
(de machin1 (lim dep but n)
        (explo (list dep) nil (fils (list dep) n) 0 lim but 0 n))
```

Execution

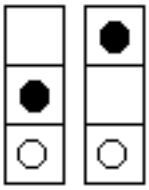
```
(machin 2 9)
= ((1 b b v n n) (9 b v b n n) (11 b n b n v) (10 b n v n b) (18 b n n v b) (20 v n n b b) (4 n n v b b))

(machin 3 25)
= ((1 b b b v n n n) (9 b b v b n n n) (11 b b n b n v n) (11 b b n v n b n) (18 b v n b n b n) (23 b n n b v b n) (24 b
n n b n b v) (26 b n n v n b b) (25 b v n n n b b) (27 v b n n n b b) (28 n b n v n b b) (36 n b n n v b b) (37 n v n n
b b b) (36 v n n n b b b) (6 n n n v b b b))

(machin 4  50)
= ((1 b b b b v n n n n) (8 b b b b n v n n n) (10 b b b v n b n n n) (9 b b b n v b n n n) (13 b v b n b b n n n) (15 v
b b n b b n n n) (24 n b b v b b n n n) (26 n b b n b b v n n) (27 n b b n v b b n n) (31 n b b n n b b v n) (33 n b b
n n b b n v) (42 n b b n n v b n b) (44 n b v n n b b n b) (37 v b n n n b b n b) (37 b v n n n b b n b) (41 b n v n n
b b n b) (40 v n b n n b b n b) (45 n v b n n b b n b) (48 n n b n v b b n b) (52 n n b n n b b v b) (51 n n b n n b b
b v) (60 n n b n n v b b b) (62 n n v n n b b b b) (55 v n n n n b b b b) (60 n v n n n b b b b) (8 n n n n v b b b b))
```

13-11° Jeu du taquin à l'aide de A*. On considère le carré 3*3 empli en ligne par 2 8 3, 1 6 4, 7 _ 4 avec pour but de parvenir à 1 2 3, 8 _ 4, 7 6 5. On prendra comme évaluation d'un sommet quelconque de l'arbre de recherche le nombre de coups joués pour y parvenir augmenté du nombre de pièces mal placées.
On peut essayer également la profondeur du noeud + la somme des distances de Hamming de chaque pièce entre sa position courante et sa position finale + prime de 6 pour une case périphérique si elle n'est pas suivie par son successeur + prime de 3, si la case centrale n'est pas vide. L'état initial sera représenté par (1 2 3 8 0 4 7 6 5) puis il faudra définir les opérations haut, bas, gauche, droite.

13-12° Minimax pour le dodgem ou jeu d'Hex (Piet Hein) Ce jeu consiste à sortir ses deux pions (à droite du damier pour les noirs et en haut pour les blancs). Les joueurs déplacent à tour de rôle un de leur pions dans une case libre adjacente en respectant les actions suivantes : droite, haut et bas pour les noirs, droite, gauche et haut pour les blancs. Ainsi un pion ne peut reculer relativement à son éventuelle sortie, mais peut se déplacer latéralement.


Jeu initial.

Valeurs des cases pour les noirs.


Primes 8 et 6 pour les noirs.

Valeurs des cases pour les blancs.

On imagine une heuristique tenant compte des positions des pions en regard de leurs sorties respectives (max = noir, min = blanc), mais en donnant une prime si un joueur bloque partiellement son adversaire (8 et 6 également pour une situation horizontale analogue où un blanc barre la route à un noir). On donne enfin 9 points de plus si un noir est déjà sorti (-9 pour un blanc).
Un état est la liste constituée par son évaluation suivie des trois lignes successives du damier, emplies de N, B, ou - (pour une case vide).

```
(de rg (n l) (cond ; donne l'élément de rang n dans la liste l
        ((< n 0) nil)
        ((null l) nil)
        ((eq n 0) (car l))
        (t (rg (1- n) (cdr l)) )))
(de di (q n) (eq (modulo n q) 0)) ; est le prédicat de divisibilité

(de prime (e er r p nb nn) (cond
        ((null er) (+ p (* 9 (- nb nn)))) ; prime 9 si un seul noir restant, -9 si un seul blanc
        ((eq (car er) 'n) (cond    ((neq (rg (+ r 3) e) 'b) (if (eq (rg (+ r 6) e) 'b)
                                                (prime e (cdr er) (1+ r) (+ p 6) nb (1+ nn))
                                                (prime e (cdr er) (1+ r) p nb (1+ nn))))
                                   ((eq (rg (+ r 6) e) 'b) ; neq signifie "non égal
                                                (prime e (cdr er) (1+ r) (+ p 14) nb (1+ nn)))
                                   (t  (prime e (cdr er) (1+ r) (+ p 8) nb (1+ nn)))))
        ((eq (car er) 'b) (cond    ((neq (rg (1- r) e) 'n) (cond
                                        ((neq (rg (- r 2) e) 'n) (prime e (cdr er) (1+ r) p (1+ nb) nn))
                                        ((di 3 r) (prime e (cdr er) (1+ r) (- p 6) (1+ nb) nn))
                                        (t (prime e (cdr er) (1+ r) p (1+ nb) nn))))
                                   ((neq (rg (- r 2) e) 'n) (if (di 3 (1- r))
                                                (prime e (cdr er) (1+ r) p (1+ nb) nn)
                                                (prime e (cdr er) (1+ r) (- p 8) (1+ nb) nn)))
                                   ((di 3 r) (prime e (cdr er) (1+ r) (- p 14)(1+ nb) nn))
                                   (t (prime e (cdr er) (1+ r) (- p 8) (1+ nb) nn))))
        (t (prime e (cdr er) (1+ r) p nb nn))))
```

```
(de heurist1 (e refn refb h) (cond ; donne la valuation h d'un état e sans les primes
        ((null e) h)
        ((eq (car e) 'n) (heurist1 (cdr e) (cdr refn) (cdr refb) (+ h (car refn))))
        ((eq (car e) 'b) (heurist1 (cdr e) (cdr refn) (cdr refb) (+ h (car refb))))
        (t (heurist1 (cdr e) (cdr refn) (cdr refb) h))))
(de heurist (e) (+ (heurist1 (cdr e) '(2 5 8 1 4 7 0 3 6) '(-6 -7 -8 -3 -4 -5 0 -1 -2) 0) (prime e (cdr e) 1 0 0 0)))

(de libre (r e) (eq (rg r e) '-)); teste si l'élément de rang r dans l'état e est une case libre

(de fils (e joueur) (fils1 e (cdr e) 1 joueur nil)) ; renvoie la liste des coups possibles à partir de e pour le loueur

(de fils1 (e er r j F) (cond
;F est l'ensemble des fils de e, r le rang de l'élément (car er) dans e, j = joueur (noir=programme, b = personne)
        ((null er) (mapcar (lambda (x) (cons (heurist x) (cdr x))) F))
        ((and (eq (car er) 'n) (eq j 'n)) (let ((m f))
                (if (and (libre (1+ r) e) (not (di 3 r))) (set 'm (cons (transpo e r (1+ r)) m)))
                (if (and (libre (+ r 3) e) (< r 7)) (set 'm (cons (transpo e r (+ r 3)) m)))
                (if (and (libre (- r 3) e) (< 3 r)) (set 'm (cons (transpo e r (- r 3)) m)))
                (if (di 3 r) (set 'm (cons (append (tete r e) (cons '- (cdr er))) m)))
                (fils1 e (cdr er) (1+ r) j m)))
        ((and (eq (car er) 'b) (eq j 'b)) (let ((m f))
                (if (and (libre (1+ r) e) (< r 9)) (set 'm (cons (transpo e r (1+ r)) m)))
                (if (and (libre (1- r) e) (< 1 r)) (set 'm (cons (transpo e r (1- r)) m)))
                (if (and (libre (- r 3) e) (< 3 r)) (set 'm (cons (transpo e r (- r 3)) m)))
                (if (< r 4) (set 'm (cons (append (tete r e) (cons '- (cdr er))) m))) (fils1 e (cdr er) (1+ r) j m)))
        (t (fils1 e (cdr e) (1+ r) j F)) )) ; dans chacun des deux derniers cas on ajoute une prime de sortie

(de transpo (L i j) (cond ; renvoie L où les éléments de rangs i<j ont été transposés (à partir de 0)
        ((< j i) (transpo L j i))
        ((< 0 i) (cons (car L) (transpo (cdr L) (1- i) (1- j))))
        ((eq i 0) (cons  (nth j L) (append (firstn (- j 1) (cdr L)) (cons (car L) (nthcdr (1+ j) L)))))))
```

Fonction minimax permettant pour un joueur donné de classer les coups à jouer en regardant à un niveau de profondeur donné.

```
(de minimax (e n j) ; e est l'état du jeu, j le joueur (n ou b) et n le niveau de profondeur
(cond   ((succes e 'n) 90)
        ((succes e 'b) -90)
        ((eq n 0) (car e))
        (t   (let       ((score (if (eq j 'n) -90 90)) (fi (fils e j)))
                        (while fi (let  ((m (minimax (car fi) (- n 1) (advers j))))
                                        (set 'score (if (eq j 'n) (max score m) (min score m)))
                                        (set 'fi (cdr fi))))
                        score)) ))

(de succes (e j) (not (member j e)))
(de advers (joueur) (if (eq joueur 'n) 'b 'n))
```

Choix du programme pour un état e donné, -100 < -90 est pris afin de choisir quand même un coup s'ils sont tous à -90

```
(de choix (e n) (choix1 n (fils e 'n) nil -100)); renvoie l'état choisi par le programme (les noirs)
(de choix1 (n f ch nu) (if (null f) ch (choix2 n (cdr f) (car f) (minimax (car f) (- n 1) 'b) ch nu)))
(de choix2 (n f ch2 nu2 ch1 nu1) (if (< nu1 nu2) (choix1 n f ch2 nu2) (choix1 n f ch1 nu1)))

(de jeu  (n) ; n est la profondeur du regard en avant
(print '" Vous avez les blancs, voulez-vous commencer (o / n) ? ")
(let ((e '(0 n - - n - - - b b)) (nc 0))
        (if (eq (read) 'o)(vue e) (print '" Je joue ") (set 'e (choix e n)) (vue e) (set 'nc (1+ nc)))
        (while (not  (or (succes e 'n) (succes e 'b)))
                (set 'e (modif e (prin '"Coup numéro " (1+ nc) '" pion " (read)) (print '" mouvement " (read))))
        (ifn (succes e 'b) (set 'e (choix e n))); le programme choisit son coup
                (set 'nc (+ nc 2)) (vue e)) ; le damier n'est visualisé qu'après que le programme ait joué
        (print  (if (succes e 'n) '" Vous avez perdu " '" Vous avez gagné ") nil )))
```

```
(de vue (e) (print (firstn 3 (cdr e)) '"         Coup numéro  "  nc '"   Valuation  " (car e))
        (print (firstn 3 (nthcdr 4 e))  '"      Donnez votre coup par le numéro du pion blanc")
        (print (nthcdr 7 e) '"       dans l'ordre de gauche à droite et de haut en bas")
                (print '"                 suivi de h (haut) g (gauche) d (droite) "))
(de modif (e num mouv) ; renvoie un autre état où un blanc a été déplacé (éventuellement sorti)
        (modif1 e (cdr e) 1 num mouv))
(de modif1 (e er r num mouv) (cond ; aucune validité des coups n'est vérifiée ici
        ((and (eq (car er) 'b) (eq num 2)) (modif1 e (cdr er) (1+ r) 1 mouv))
        ((eq (car er) 'b) (let      ((j (cond       ((eq mouv 'h) (- r 3))
                                                    ((eq mouv 'd) (+ r 1))
                                                    ((eq mouv 'g) (- r 1)))))
                                    (if (< j 1)  (append (tete r e) (cons '- (cdr er))); cas d'une sortie
                                                 (transpo e r j))))
        (t (modif1 e (cdr er) (1+ r) num mouv)) ))
```

Exemple d'éxecution (jeu 2)	Coup numéro 7 pion 1 mouvement h
Vous avez les blancs, voulez-vous commencer (o / n)?	(- n n) Coup numéro 8 Valuation 7
(n - -) Coup numéro 0 Valuation 0	(- - b) Donnez votre coup par le numéro du pion blanc
(n - -) Donnez votre coup par le numéro du pion blanc	(- - -) dans l'ordre de gauche à droite et de haut en bas
(- b b) dans l'ordre de gauche à droite et de haut en bas	suivi de h (haut) g (gauche) d (droite)
suivi de h (haut) g (gauche) d (droite)	Coup numéro 9 pion 1 mouvement g
Coup numéro 1 pion 1 mouvement h	(- n -) Coup numéro 10 Valuation 9
(- n -) Coup numéro 2 Valuation 0	(- b -) Donnez votre coup par le numéro du pion blanc
(n b -)Donnez votre coup par le numéro du pion blanc	(- - -) dans l'ordre de gauche à droite et de haut en bas
(- - b) dans l'ordre de gauche à droite et de haut en bas	suivi de h (haut) g (gauche) d (droite)
suivi de h (haut) g (gauche) d (droite)	Coup numéro 11 pion 1 mouvement g
Coup numéro 3 pion 2 mouvement h	(- - n) Coup numéro 12 Valuation 5
(- - n) Coup numéro 4 Valuation -6	(b - -) Donnez votre coup par le numéro du pion blanc
(n b b)Donnez votre coup par le numéro du pion blanc	(- - -) dans l'ordre de gauche à droite et de haut en bas
(- - -) dans l'ordre de gauche à droite et de haut en bas	suivi de h (haut) g (gauche) d (droite)
suivi de h (haut) g (gauche) d (droite)	Coup numéro 13 pion 1 mouvement h
Coup numéro 5 pion 1 mouvement h	(b - -) Coup numéro 14 Valuation 3
(- b n) Coup numéro 6 Valuation 0	(- - -) Donnez votre coup par le numéro du pion blanc
(- n b)Donnez votre coup par le numéro du pion blanc	(- - -) dans l'ordre de gauche à droite et de haut en bas
(- - -) dans l'ordre de gauche à droite et de haut en bas	suivi de h (haut) g (gauche) d (droite)
suivi de h (haut) g (gauche) d (droite)	Vous avez perdu ()

13-13° Tri topoplogique dans un graphe orienté sans circuit, il faut ranger les sommets dans un ordre tel que tous les prédécesseurs d'un sommet soient situés avant lui. En disposant d'un graphe G où chaque sommet a pour attributs possibles les listes de prédécesseurs et de suivants (utiliser les p-listes), il faut construire une liste L en suivant l'algorithme :

R ← Ø L ← Ø (R servira pour les sommets restant à placer)
Du premier au dernier sommet s de G si s sans pred, placer s dans R
Tant que R ≠ Ø soit s premier sommet de R (ou un quelconque)
R ← R - {s}
pred (s) ← Ø
L ← Insérer s en queue de L
Pour tous les successeurs s' de s pred(s') ← pred(s') - {s}
si pred (s') = Ø alors R ← R + {s'}

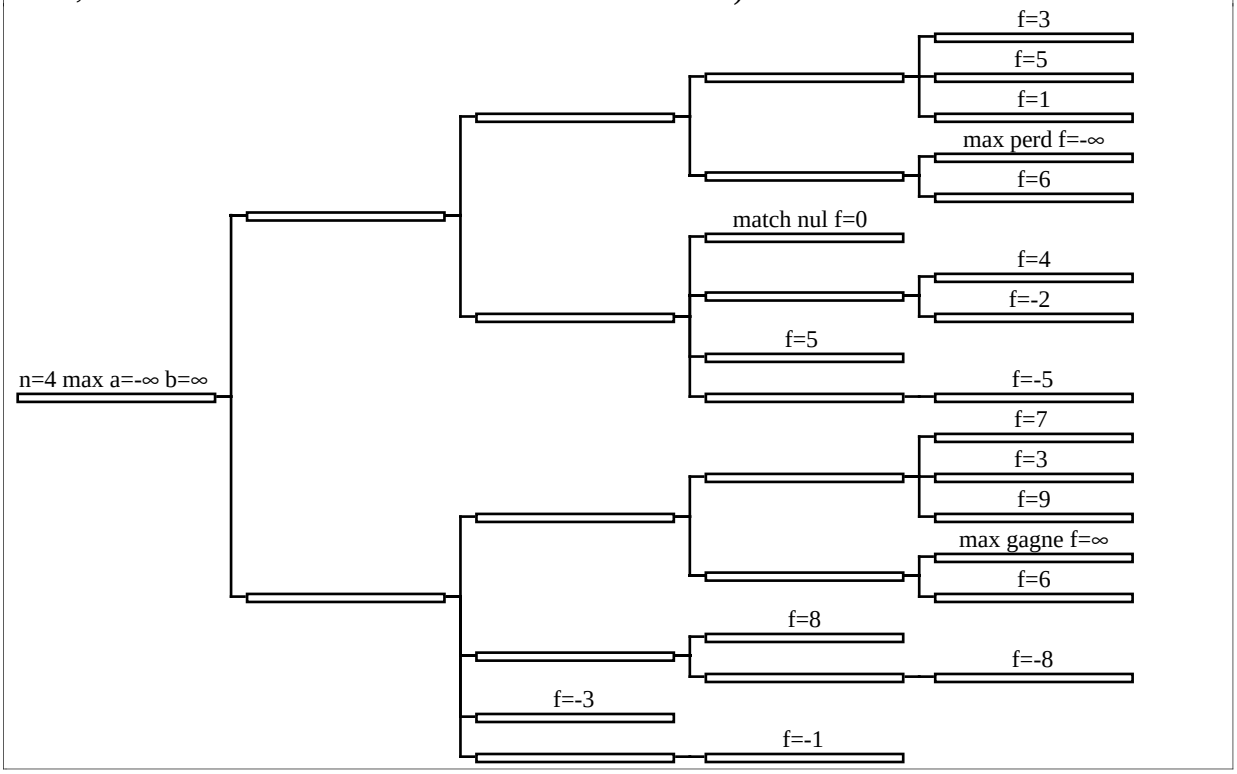
Exécuter à la main pour un graphe à 5 sommets tous reliés mais sans circuit.

13-14° Plus court chemin dans un graphe, (algorithme de Floyd) on suppose donné G et la matrice A où a_{ij} est le coût du sommet a_i vers a_j s'il est défini, (l'infini sinon et $a_{ii} = 0$). On construit la matrice P où p_{ij} est le sommet père de ai dans le plus court chemin de a_i à a_j :

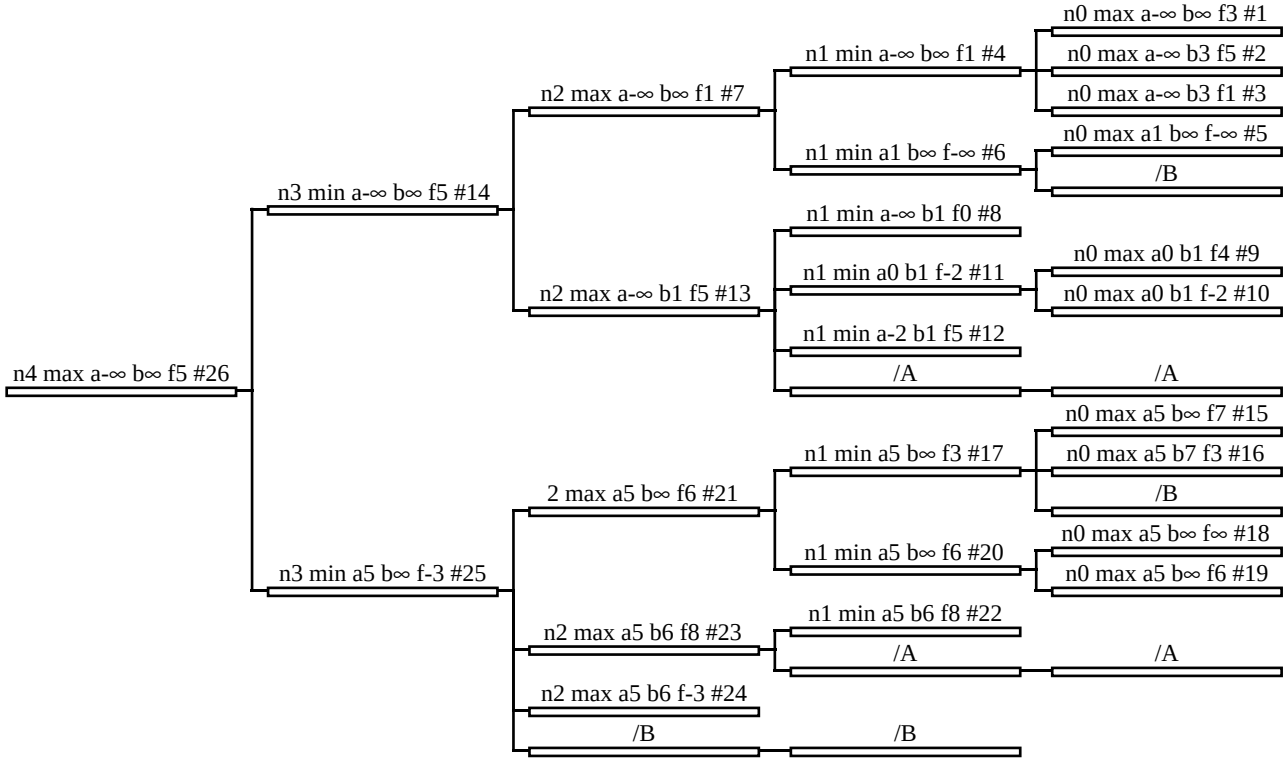
Pour i = 1 à n faire pour j = 1 à n P(i, j) ← i {initialisation}
Pour k = 1 à n faire pour i = 1 à n faire pour j = 1 à n faire
si A(i, k)+A(k, j) < A(i, j) alors A(i, j) ← A(i, k) + A(k, j) puis P(i, j) ← P(k, j)

A chaque k on calcule le plus court chemin de a_i à a_j passant par des sommets de rangs < k.

13-15° Appliquer l'algorithme alpha-beta à la main sur l'arbre suivant (en écrivant pour chaque noeud le niveau n, le joueur, les valeurs d'appels de alpha et beta, puis la valeur de f lorsqu'elle est connue avec son numéro dans l'ordre des calculs, le départ est fixé avec n = 4, a = -∞, b = ∞ et certaines valeurs terminales sont données).



Solution (on indique successivement à chaque état la profondeur n (4 à 0), le joueur (max ou min), les valeurs de α, β, f, et le numéro d'ordre # des calculs de la fonction f.



13-16° Concevoir le jeu du Tic tac toe à l'aide du minimax

13-17° Trouver une condition d'existence d'une solution au problème définit par deux colonnes emplis d'entiers à chercher, tels que la somme des colonnes soit donnée c1 et c2, ainsi également que la somme de chaque ligne. Par exemple si c1=7 c2=3 et les sommes de lignes sont imposées SL = (4 1 2 3) alors le couple (3 1 2 1) (1 0 0 2) est solution.
Programmer une fonction "ok" vérifiant une solution, puis une fonction "solution" donnant toutes les solutions.

Solution :
La condition ∑sommes des lignes = $c_1 + c_2$ est suffisante : car si $l_1,...l_n$ sont les sommes de lignes et en supposant $c_1 \geq c_2$, soit k le plus grand entier tel que $\sum l_i \ _{1 \leq i \leq k} \leq c1$, on a donc $l_1+l_2+...l_k > c_1$ au cas où $k \neq n$.
Si k = n la solution $(l_1...l_n)$ (0...0) est évidente, sinon on considère les lignes $(l_1\ 0)$ $(l_2\ 0)$ ... $(l_k\ 0)$ puis $(c1 - \sum_{1 \leq i \leq k} l_i, x)$ avec $x = l_{k+1} - c_1 + \sum_{1 \leq i \leq k} l_i$ puis enfin les lignes $(0\ l_{k+2})$ ... $(0\ l_n)$. Toutes les conditions sont satisfaites, mais il faut vérifier que x > 0, ce qui est assuré par l'inégalité plus haut.

```
(de ok (DV SC SL) (cond ; ex: SC = ( 7 3);  SL = (4 1 2 3);  DV = ((1 1  2 3) (3 0 0 0))
        ((null (car DV)) (if (eq SC '(0 0)) t))
        (eq (+ (caar DV) (caadr DV)) (car SL))
           (ok (list (cdar DV) (cdadr DV)) (list (- (car SC) (caar DV)) (- (cadr SC) (caadr DV))) (cdr SL))) ))
```

On simplifie légèrement par rapport à l'énoncé, en désignant C1 la somme de la première colonne et C2 celle de la seconde.

```
(de solution (C1 C2 SL) (sol1 0 (car SL) C1 (- C2 (car SL)) (cdr SL))) ; SL est la liste des sommes des lignes
(de sol1 (A B C1 C2 SLR)
    (if          (null SLR)  (if (and (eq C1 0) (eq C2 0)) (list (list A) (list B)) nil)
                 (sol2 A B C1 C2 SLR (solution C1 C2 SLR))))
```

Sol1 cherche une solution débutant par A et B en haut des colonnes et se poursuivant avec une liste SLR des sommes de lignes restantes, sol2 repasse la main à sol1 si cela est nécessaire.

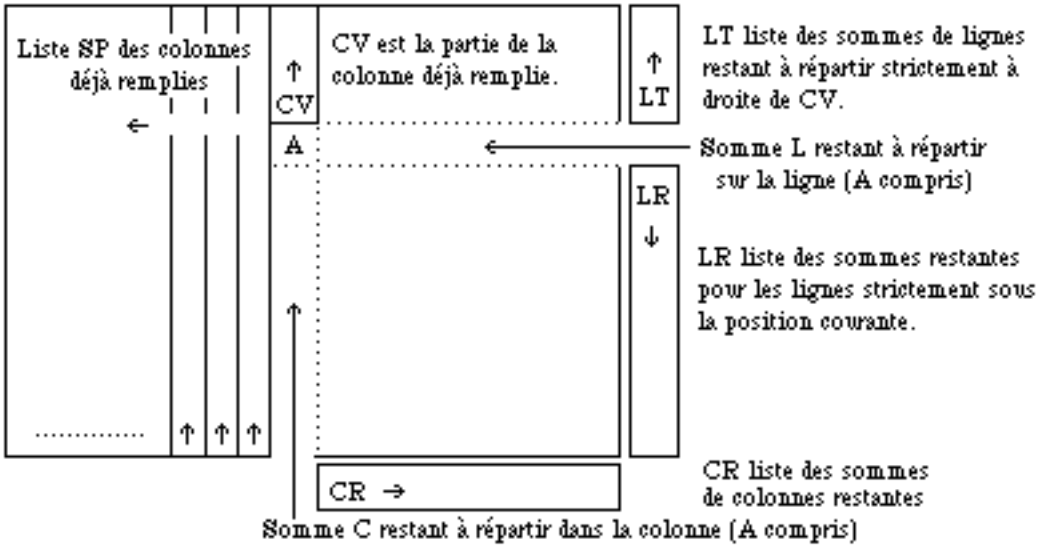
```
(de sol2  (A B C1 C2 SLR SOL)
    (if  (null SOL) (if     (eq B 0) nil (sol1 (+ 1 A) (- B 1) (- C1 1) (+ C2 1) SLR))
        (list (cons A (car SOL)) (cons B (cadr SOL)))))
```

Exemple :
```
(solution 14 18 '(3 5 7 8 9))
= ((0 0 6 8 0) (3 5 1 0 9))
```

13-18° Rectangles semi-magiques, généraliser l'exercice précédent à un tableau rectangulaire.

Il y a bien sûr une foule de solutions de programmation différentes pour trouver les figures magiques, cependant on insiste ci-dessous sur le point de vue fonctionnel récursif terminal qui donne des écritures très élégantes. On propose ici une solution reprenant tout à zéro sans chercher à s'appuyer sur des solutions au même problème de taille réduite. On prend tout le backtrack à bras le corps en désignant à chaque instant, les paramètres utilisés sur le schéma. A est la valeur proposée et les flèches indiquent le sens des listes.

On crée deux fonctions, dont la première "exam" examine l'entier A proposé pour une place donnée, et l'autre, "bak", provoque une remontée dans les essais successifs sachant que A n'est pas une valeur acceptable. Les deux fonctions se donnent donc la main tout au long de l'exploration des possibilités.



L'examen se poursuit de colonne en colonne, chacune étant remplie de haut en bas

```
(de exam (SP CV A C L CR LR LT) (cond
        ((or (> A C)(> A L)) (bak SP CV  C L CR LR LT))
        ((null LR)                      ; on est au bas d'une colonne
        (cond    ((null CR)             ; on est à la toute dernière position
                        (if (and (eq A C) (eq A L)) (print (reverse (mapcar 'reverse (cons (cons A CV)SP))))
                        (exam SP CV (+ A 1) C L CR LR LT))
                ((eq A C) (exam         (cons (cons A CV) SP) nil 0 (car CR) (car (last LT)) (cdr CR)
                                (cdr (reverse (cons(- L A) LT))) nil))
                (t (exam SP CV (+ 1 A) C L CR LR LT))))
        ((null CR)                      ; on est dans la dernière colonne, mais pas en bas
        (cond    ((eq A L) (exam SP (cons A CV) 0 (- C A) (car LR) CR (cdr LR) (cons 0 LT)))
                (t  (exam SP CV (+ 1 A) C L CR LR LT))); A n'est pas suffisant
        (t ; examen de la position suivante dans la même colonne
                (exam SP (cons A CV) 0 (- C A) (car LR) CR (cdr LR) (cons (- L A) LT)))))

(de bak  (SP CV C L CR LR LT) (cond
        ((null CV)                      ; on est en haut d'une ligne
        (cond    ((null SP) nil)        ; fini, on renvoie nil ou bien l'ensemble des solutions
                (t ; il faut aller au bas de la colonne précédente
                        (exam (cdr SP) (cdar SP) (+ 1 (caar SP)) (caar SP) ; provoque un second bak
                                (+ (caar SP)(car (last LR))) (cons C CR) nil (cdr (reverse (cons L LR)))))))
        (t ; on va simplement dans la case au dessus
          (exam SP (cdr CV) (+ 1 (car CV)) (+ C (car CV)) (+ (car CV) (car LT)) CR (cons L LR) (cdr LT)))))

(de reso (SC SL) (exam nil nil 0 (car SC) (car SL) (cdr SC) (cdr SL) nil))
```

L'intérêt de cette solution est sa "terminalité" c'est-à-dire qu'un interpréteur lisp vérifiant que les appels récursifs sont terminaux n'encombre pas la pile d'exécution et peut travailler avec élégance.

Exemple (les impressions ont été remises après coup sur trois colonnes) :

```
(reso'(6 7 5)'(8 4 6))
((0 0 6) (3 4 0) (5 0 0))
((0 0 6) (4 3 0) (4 1 0))
((0 0 6) (5 2 0) (3 2 0))
((0 0 6) (6 1 0) (2 3 0))
((0 0 6) (7 0 0) (1 4 0))
((0 1 5) (3 3 1) (5 0 0))
((0 1 5) (4 2 1) (4 1 0))
((0 1 5) (4 3 0) (4 0 1))
((0 1 5) (5 1 1) (3 2 0))
((0 1 5) (5 2 0) (3 1 1))
((0 1 5) (6 0 1) (2 3 0))
((0 1 5) (6 1 0) (2 2 1))
((0 1 5) (7 0 0) (1 3 1))
((0 2 4) (3 2 2) (5 0 0))
((0 2 4) (4 1 2) (4 1 0))
((0 2 4) (4 2 1) (4 0 1))
((0 2 4) (5 0 2) (3 2 0))
((0 2 4) (5 1 1) (3 1 1))
((0 2 4) (5 2 0) (3 0 2))
((0 2 4) (6 0 1) (2 2 1))
((0 2 4) (6 1 0) (2 1 2))
((0 2 4) (7 0 0) (1 2 2))
((0 3 3) (3 1 3) (5 0 0))
((0 3 3) (4 0 3) (4 1 0))
((0 3 3) (4 1 2) (4 0 1))
((0 3 3) (5 0 2) (3 1 1))
((0 3 3) (5 1 1) (3 0 2))
((0 3 3) (6 0 1) (2 1 2))
((0 3 3) (6 1 0) (2 0 3))
((0 3 3) (7 0 0) (1 1 3))
((0 4 2) (3 0 4) (5 0 0))
((0 4 2) (4 0 3) (4 0 1))
((0 4 2) (5 0 2) (3 0 2))
((0 4 2) (6 0 1) (2 0 3))
((0 4 2) (7 0 0) (1 0 4))
((1 0 5) (2 4 1) (5 0 0))
((1 0 5) (3 3 1) (4 1 0))
((1 0 5) (3 4 0) (4 0 1))
((1 0 5) (4 2 1) (3 2 0))
((1 0 5) (4 3 0) (3 1 1))
((1 0 5) (5 1 1) (2 3 0))
((1 0 5) (5 2 0) (2 2 1))
((1 0 5) (6 0 1) (1 4 0))
((1 0 5) (6 1 0) (1 3 1))
((1 0 5) (7 0 0) (0 4 1))
= ( )
```

13-19° Carrés magiques
La somme de chaque ligne, chaque colonne et des deux diagonales est la même.

Donnons une première solution complétant le programme précédent, mais dont l'inconvénient sera d'obtenir toutes les solutions avant d'en vérifier les diagonales, voire les isométries. Ce n'est évidemment pas ainsi qu'il faut procéder pour n > 3.
On introduit une petite fonction calculant la somme de ces éléments diagonaux d'une liste de listes L, R=0 pour la diagonale, R=1 pour la diagonale supérieure, R=-1 pour l'inférieure etc...

```
(de diag (L R) (cond
        ((null L) 0)
        ((null (car L)) 0)
        (( < R 0) (diag (mapcar 'cdr L) (+ 1 R)))
        (( < 0 R) (diag (cdr L) (- R 1)))
        (t (+ (caar L) (diag (cdr L) (- R 1))) ) ))       ; dernier cas si R = 0
```

Exemple

```
(set 'e '((1 2 3 4 5 6) (2 4 6 8 10 12) (3 6 9 12 15 18) (4 8 12 16 20 24)))
(diag e -2) = 50
(diag e 3) = 4
(diag e 0) = 30
```

Par ailleurs :

```
(de entiers(n) (reverse (entierbis n))) ; (entiers 5) = (1 2 3 4 5)
(de entierbis (n) (if (eq n 0) nil (cons n (entierbis (- n 1)))))

(de som (n) (quotient (* n (+ 1 (* n n)))2))
(de repete (k x) (if (eq k 0) nil (cons x (repete (- k 1) x)))) ; (repete 4 'a) = (a a a a)
(de ret (X L) (cond
        ((null L) nil)
        ((eq X (car L)) (cdr L))
        (t (cons (car L) (ret X (cdr L))))))
(de mapcons (X L) (ifn (null L) (cons (cons X (car L)) (mapcons X (cdr L)))))
(de inclu (L M) (cond       ((null L) t)
                            ((member (car L) M) (inclu (cdr L) M))
                            (t nil)))
(de diff (L) (cond          ((null L) t) ; vérifie qu'une liste a tous ses éléments distinct
                            ((member (car L) (cdr L)) nil)
                            (t (diff (cdr L)))))
(de elim (L) (cond
        ((null L) nil) ; produit une liste sans répétition
        ((member (car L) (cdr L)) (elim (cdr L)))
        ( t (cons (car L) (elim (cdr L))))))
```

```
(de verifdiag (ES X) (cond
        ((null ES) nil)
        ((not (eq (diag (car ES) 0) X)) (verifdiag (cdr ES) X))
        ((not (eq (diag (reverse (car ES)) 0) X)) (verifdiag (cdr ES) X))
        (t (cons (car ES) (verifdiag (cdr ES) X)))))
```

Maintenant l'essentiel est de calculer une colonne et de trouver des tableaux semi-magiques (on donne une solution sans variables locales, mais en décomposant en plusieurs fonctions) :

```
(de reso1 (SC SL R) ; donne la liste des rectangles semi-magiques avec des valeurs dans R
        (if  (null (cdr SC)) (if       (and (inclu SL R)  (diff SL) (eq (car SC) (apply '+ SL)))  (list (list SL)) )
                                       (reso2 (colonne (car SC) SL R) (cdr SC))))
(de reso2 (ES SCR) (ifn (null ES) (reso3 (caar ES) (cadar ES) (cddar ES) (cdr ES) SCR)))
(de reso3 (RR SLR COL ESR SCR) (append (reso4 (reso1 SCR SLR RR) COL) (reso2 ESR SCR)))
(de reso4 (SOLR COL) (ifn (null SOLR)  (mapcons COL SOLR)))
```

Une colonne sera une liste du type ((liste des valeurs encore disponibles) (liste des sommes restant à répartir) liste des valeurs de la colonne) "colonne" donne la liste des colonnes de nombres strict. majorés par ceux de SL, de somme S et pris distinctement dans R

```
(de colonne (S SL R) (cond
        ((null R) nil)
        ((null (cdr SL))   (cond     ((<= (car SL) S) nil)         ; cas de colonnes à une ligne
                                     ((null (member S R)) nil)
                                     (t (list (list (ret S R)(list (- (car SL) S)) S)))))          ; seule solution
        (t (col1 R (car SL) S (cdr SL) R))))
(de col1 ( R L S SL DEB) (ifn (null DEB)
        (append (col2 (car DEB) L S SL (ret (car DEB) (ret (car DEB) R))) (col1 R L S SL (cdr DEB)))))
```

COL1 donne la liste des colonnes , confrontées à L en leur tête, R valeurs possibles de somme S vérifiant Sl pour leur queue, DEB l'ensemble des débuts possibles (pris dans R).

```
(de col2 (A L S SL R) ;liste des colonnes de début A< L, S et suivies par une suite vérifiant SL, R
  (ifn (or (<=  S A) (<= L A)) (col3  A  L (colonne (- S A) SL R)))) ; ces tests devant être stricts si R contient 0

(de col3 (A L ES); donne la liste des colonnes de ES où A est rajouté en tête
        (ifn (null ES)  (cons  (mcons (caar ES) (cons (- L A)  (cadar ES)) A (cddar ES)) (col3 A L (cdr ES)))))

(de carre (n) (verifdiag (reso1 (repete n (som n)) (repete n (som n)) (entiers (* n n))) (som n)))
```

Exemple :

```
(carre 3)
= (((2 7 6) (9 5 1) (4 3 8))  ((2 9 4) (7 5 3) (6 1 8))  ((4 3 8) (9 5 1) (2 7 6))  ((4 9 2) (3 5 7) (8 1 6))
 ((6 1 8) (7 5 3) (2 9 4))  ((6 7 2) (1 5 9) (8 3 4))  ((8 1 6) (3 5 7) (4 9 2))  ((8 3 4) (1 5 9) (6 7 2)))
```

On vérifie qu'elles sont isométriques, le carré chinois d'ordre 3 est bien unique. Les isométries du carrés sont au nombre de 8, engendées par reverse (symétrie verticale) et une transposition (symétrie de la matrice par rapport à sa diagonale principale à définir). On pourra aussi se servir de la symétrie horizontale définie par mapcar 'reverse.

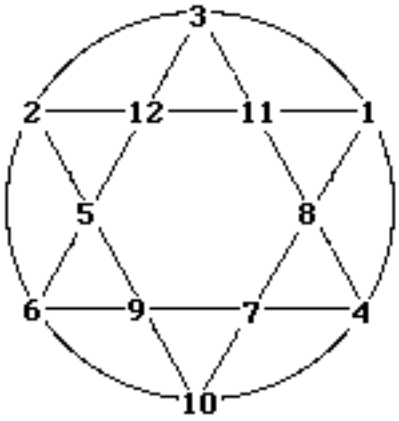
Pour n = 4, la première solution donnée (on place "verifdiag" en cours de route) est ci dessous dans le tableau central ((1 12 13 8) (2 14 7 11) (15 3 10 6) (16 5 4 9)) mais après un temps très long. Il faut dire que la complexité est en $(n2)^{n2}$ ce qui n'est pas mal. A gauche le fameux carré chinois.

2	9	4
7	5	3
6	1	8

1	2	15	16
12	14	3	5
13	7	10	4
8	11	6	9

1	3	16	14
13	15	2	4
8	6	11	9
12	10	5	7

13-20° Etoile magique à 6 sommets construite avec les entiers de 1 à 12. Pour l'étoile ainsi que pour les carrés et hexagones magiques on peut, soit reconsidérer chaque problème séparément, ce qui n'est pas très satisfaisant pour l'esprit, soit se raccrocher au cas général d'une matrice.



1	3	2	6	10	4
11	12	5	9	7	8
12	5	9	7	8	11
2	6	10	4	1	3

1	11
3	12
2	5
6	9
10	7
4	8

C1 C2

Si nous choisissons la seconde option, nous pouvons considérer le premier rectangle dont les colonnes représentent les branches, en ce cas, les deux lignes extrêmes sont le "tour extérieur" de l'étoile et les deux lignes médianes "le tour intérieur". On peut alors montrer que si les entiers de 1 à 12 sont présents, en comptant deux fois chaque sommet, et si S est la somme commune des branches et du tour extérieur, on a 6S = n(n+1) soit S = 26 pour n = 12 et le tour intérieur admet la somme 3S-S = 2S = 52. On peut donc appliquer la fonction précédente avec :

```
(de etoile () (reso1 (repete 6 26) '(26 52 52 26) '(1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12 12)))
```

Mais nous tombons alors dans un maelström combinatoire, il est donc plus judicieux de regarder les deux colonnes formées par les tours extérieurs et intérieurs, (figure à droite) la somme de chaque branche de l'étoile se trouvant être la somme des deux élément d'un étage, avec celui de dessous à droite et de celui deux crans plus bas à gauche (circulairement parlant) de ces deux listes. On peut donc revenir à une programmation de backtrack sur chacun des choix de la première liste. Par ailleurs d'autres étoiles magiques pourraient se voir étudier de manière analogue.

A chaque instant, les colonnes déjà construites se nomment L1 et L2, A (éventuellement nil) est la valeur proposée en tête de L1, B celle pour L2, R est la liste des valeurs restantes (contenant A et B), S la somme 26, C1, C2 les sommes restant à répartir (en comprenant A, B). P, Q sont les premières valeurs à avoir été mises, on veillera alors à conserver P le minimum de L1, ainsi on évitera dès le départ les solutions obtenues par rotation ce qui diminuera la recherche. On pourra éliminer les solutions symétriques (axiales) par la suite.

```
(de suiv (X L) (cond ; donne le suivant de X dans L, liste ordonnée croissante d'entiers, nil s'il n'y en a plus
        ((null L) nil)
        ((<=  (car L) X ) (suiv X (cdr L)))
        (t (car L))))

(de raj ( X L) (cond  ; rajoute en l'insérant à sa place, l'entier X dans la liste ordonnée L
        ((null L) (list X))
        ((< (car L) X) (cons (car L) (raj X (cdr L))))
        (t (cons X L)) ))

(de ex1 (A L1 L2 C1 C2 S P Q R) ; solution entièrement terminale entre les 3 fonctions ex1, ex2 et bak
(cond   ((null A) (bak A L1 L2 C1 C2 S P Q R))
        ((null L1) (ex2 A (car R) () () C1 C2 S A (car R) R))
        ((or (<  C1 A) (<  A P)) (bak A L1 L2 C1 C2 S P Q R))
        (t (ex2 A (- S A (car L1) (car L2)) L1 L2 C1 C2 S P Q R))))
```

```
(de ex2 (A B L1 L2 C1 C2 S P Q R) (cond
        ((not (member B R)) (bak A L1 L2 C1 C2 S P Q R))
        ((eq (length L1) 5) (cond  ((not (and (eq A C1) (eq  B C2) (eq S (+ A B P Q))))
                                        (bak A L1 L2 C1 C2 S P Q R))
                                (t (print (cons A L1) (cons B L2))  (ex1 (suiv A R) L1 L2 C1 C2 P Q R))))
        (t (ex1 (car (ret A (ret B R))) (cons A L1) (cons B L2) (- C1 A) (- C2 B)S P Q (ret A (ret B R))))))
```

A l'appel de bak, A n'a pas été retenu, on fait le choix suivant sauf au premier étage où on cherche le B (c'est à dire Q) suivant.

```
(de bak (A L1 L2 C1 C2 S P Q R) (cond
        ((null L1) ;on est au premier étage des choix
        (cond    ((null A) nil) ; fini
                 ((null Q) (ex1 (suiv A R) () () C1 C2 S A () R))
                 (t (ex2 A (suiv Q R) () () C1 C2 S A (suiv Q R) (raj Q R)))))
        ((null A)
           (bak (car L1) (cdr L1) (cdr L2) (+ C1 (car L1)) (+ C2 (car L2)) S P Q (raj (car L1) (raj (car L2) R))))
        (t (ex1 (suiv A R) L1 L2 C1 C2 S P Q R))))

(de etoile () (ex2 1 2 () () 26 52 26 1 2 (entiers 12))) ; on démarre toujours avec 1 et 2 comme premiers choix
```

(etoile) donne ((3 2 6 10 4 1) (11 12 5 9 7 8)) = ()

Solution itérative

```
(de rempli (L1 L2 C1 C2 S R n)
(if (eq (length L1) n)  (print L1 L2)
        (let ((A (car R)))
        (until (null A)
                (let ((B (car (ret A R))))
                (until (null B)
                        (if (possible A B L1 L2 C1 C2 S R n)
                                (rempli (cons A L1) (cons B L2) (- C1 A) (- C2 B) S (ret A (ret B R)) n))
                        (set 'B (suiv B (ret A R)))))
                (set 'A (suiv A R)))) ))

(de possible (A B L1 L2 C1 C2 S R n) (cond ; ici on a toujours A ≠ B
        ((null L1) t)
        ((null (cdr L1)) (and (< (+ A B (car L2)) S) (< (car L1) A)))
        ((eq (length L1) (- n 1)) (and (eq A C1) (eq B C2) (eq S (+ A B (cadr L1) (car L2)))(< (car (last L1))A)
                (eq S (+ (car L1) B (car (last L1)) (car (last L2))))))
        (t (and (eq S (+ A B (cadr L1) (car L2))) (< A C1) (< B C2) (< (car (last L1)) A)))))
```

Le tout dernier test signifiant que le premier choix doit être le minimum du tour extérieur

Extension Pour une étoile à n sommets, le tour extérieur étant défini par (An ... A2 A1) de somme C1, et le tour intérieur obtenu en joignant seulement les sommets en en sautant un, ce qui fait toujours des "branches" de 4 éléments, c'est (an ... a2 a1) de somme C2. Si toutes les branches ont la même somme S le même algorithme s'applique et en faisant la somme de toutes les branches on a nS = 2(C1 + C2) = 2(1+2+3+.....+2n) d'où S = 2(2n+1). Si de plus on veut C1 = S comme pour l'étoile à six branches, alors on en déduit C2 = (n-2)(2n+1). On lancera donc :

```
(de etoile (n) (rempli nil nil (* 2 (1+ (* 2 n))) (*(- n 2) (1+ (* 2 n))) (* 2 (1+ (* 2 n))) (entiers (* 2 n)) n ))
```

(etoile 6) renvoie (7 9 2 3 4 1) (11 6 8 12 10 5) (8 7 2 5 3 1) (12 4 10 11 9 6) (3 10 6 2 4 1) (8 9 5 11 12 7) (4 10 6 2 3 1) (7 9 5 12 11 8) (5 4 6 8 2 1) (12 3 11 7 10 9) (3 5 8 7 2 1) (11 4 10 6 12 9) = t

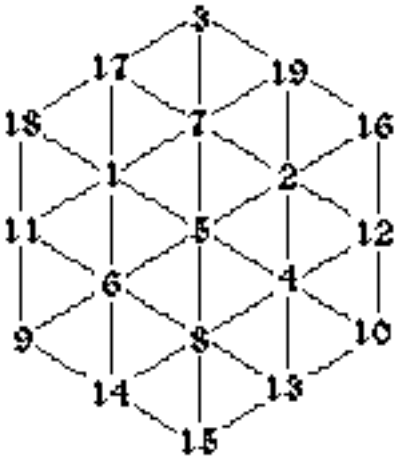
Etoile pour 5, 7 et 8 ne donne aucune solution.

On peut aussi remplacer le "print" par (ifn (member (sym L1 L2 n) SOL) (set 'SOL (cons (print (list L1 L2)) SOL)))) avec un "sol" global préalable pour faire plus "pascal".

```
(de sym (L1 L2 n) ; Rappelons que "firstn (n L)" fournit les n premiers éléments de la liste L.

        (list    (append (reverse (firstn (- n 1) L1)) (last L1))
                 (append (reverse (firstn (- n 2) L2)) (reverse (nthcdr (- n 2) L2)))))
```

13-21° Solution d'un hexagone magique construits avec les entiers de 1 à 19

On pense d'abord reprendre la solution consistant à résoudre les colonnes récursivement et les rectangles comme étant une colonne plus un rectangle plus petit, l'hexagone magique pouvant se représenter par un carré 5*5 ayant 3 zéros dans le coin bas à gauche ainsi qu'en haut à droite. Mais (de hexa () (reso1 (repete 5 38) (repete 5 38) (append (repete 6 0) (entiers 19)))) nous fait plonger au coeur d'un abîme combinatoire, c'est pourquoi on préfère une solution bestialement pascalienne reposant sur la représentation de l'hexagone suivant une seule liste consistant à mettre bout à bout les colonnes du carré 5*5 :
(15 13 10 0 0 14 8 4 12 0 9 6 5 2 16 0 11 1 7 19 0 0 18 17 3)
On examine à chaque étage k de sa construction (depuis la droite) les contraintes :
a) Impérativement des zéros aux rangs 4, 5, 10, 16, 21, 22
b) Les sommets en k = 3, 11, 15, 23, 25 doivent avoir une valeur A > (car (last L)) pour éviter les rotations.
c) Pour k = 5, 10, 15, 20, 25, il faut que la somme des 5 précédents soit 38.
d) Les sommes de 5 en 5 pour k = 11, 17, 23, 24, 25 doivent être 38.
e) Les sommes de 6 en 6 pour k = 15, 20, 23, 24, 25 doivent être 38.




```
(de rempli (L k R) (cond
        ((eq (modulo k 5) 0) (let ((A (- 38 (car L) (cadr L) (caddr L) (cadddr L))))
                (cond   ((member k '(5 10)) (if (eq A 0) (rempli (cons 0 L) (1+ k)R)))
                        ((not (member A R)) nil)
                        ((and (eq k 25) (eq (saut A L 5) 38) (eq (saut A L 6) 38) (<(car (last L))A))
                                (print (cons A L)) )
                        ((and (member k '(15 20)) (eq (saut A L 6) 38))
                                (ifn (and (eq k 15) (< A (car (last L)))) (rempli (cons A L) (1+ k) (ret A R))))
                        (t (rempli (cons A L) (1+ k)(ret A R))))))
        ((member k '(4 16 21 22)) (rempli (cons 0 L) (1+ k) R))
        ((member k '(11 17 23 24)) (let ((A (- 38 (saut 0 L 5))))
                (cond   ((not (member A R)) nil)
                        ((eq k 24) (if (eq (saut A L 6) 38) (rempli (cons A L) (1+ k) (ret A R))))
                        ((eq k 23) (if (and (eq (saut A L 6) 38) (< (car (last L))A))
                                        (rempli (cons A L) (1+ k) (ret A R))))
                        ((eq k 11) (ifn (< A  (car (last L))) (rempli (cons A L) (1+ k) (ret A R))))
                        (t (rempli (cons A L) (1+ k) (ret A R))))))
        (t (let ((A (car R)))
                (until (null A) (ifn (and (eq k 3) (< A (cadr L))) (rempli (cons A L) (1+ k) (ret A R)))
                (set 'A (suiv A R)))))))
(de saut (A L k) (sautbis A (nthcdr (1- k) L) k)) ; donne la somme de A et des éléments de L de k en k.
(de sautbis (S L k) (if (null L) S (saut (+ S (car L)) (cdr L) k)))
(de hexa () (rempli nil 1  (entiers 19))) ; est la fonction principale

(hexa)
(15 13 10 0 0 14 8 4 12 0 9 6 5 2 16 0 11 1 7 19 0 0 18 17 3)
= t ; après une quarantaine d'heures de petit macintosh mais avec cependant des impressions intermédiaires de L
```

13-22° (D'après le concours Kangourou) En s'inspirant de toutes les méthodes précédentes, chercher les carrés magiques d'ordre n, constitués par des nombres à k chiffres tels qu'en retournant le carré, il soit encore magique. Le renversement de 6 est 9 et 0, 1, 8 sont fixes. On donne ci-dessous un exemple pour n = 4 et k = 2.

96	**61**	**89**	**68**
88	**69**	**91**	**66**
61	**86**	**68**	**99**
69	**98**	**66**	**81**

Commentaires

Il y a beaucoup de choses à dire et une abondante littérature sur les carrés magiques, sachez seulement qu'ils ont commençé à être étudiés en Chine il y a 4200 ans, que tous les mathématiciens se sont exité là dessus notamment Chou Kin, Ghazali, Moschopoulos, Fermat, Euler, Cardan, Galois, Desargues et Poncelet. Ils interviennent en statistique et en théorie des graphes. Mais c'est l'aspect ludique qui motive le plus les savants fous, et les lispiens fanatiques, ainsi pour l'hexagone de Clifford Adams, celui-ci aurait mis 47 ans pour le découvrir en 1957, mais, embrouillé dans ses papiers, il a encore mis 5 ans avant de remettre au propre la solution. Ce motif est unique aux isométries près.

La construction de carrés magiques d'ordre impair avec des entiers consécutifs a été faite par Claude-Gaspar Bachet sieur de Meziriac en parcourant sphériquement le quadrillage dans le sens "vers le bas à droite". Le point de départ étant la case située immédiatement sous celle du milieu, et la règle étant que si une case est déjà occupée, on descend de deux cases (circulairement) dans la même colonne (chapitre 5).

M.Domergue (qui n'a jamais été président de la république) a trouvé une étoile magique à 18 branches où sont répartis les entiers de 1 à 144. Un comité d'admirateurs agit sans relâche pour lui obtenir une place au panthéon.

Dans le même ordre d'idée : chercher les cubes magiques 3*3*3, (il y en a 4 aux 48 isométries près).