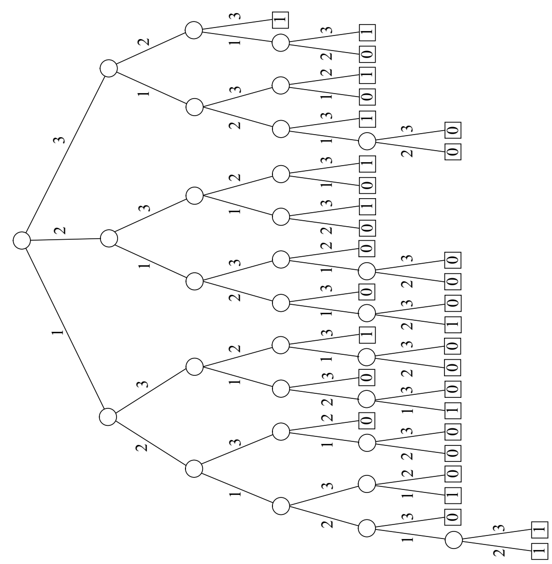
Arbres des jeux

Jeu de huit:

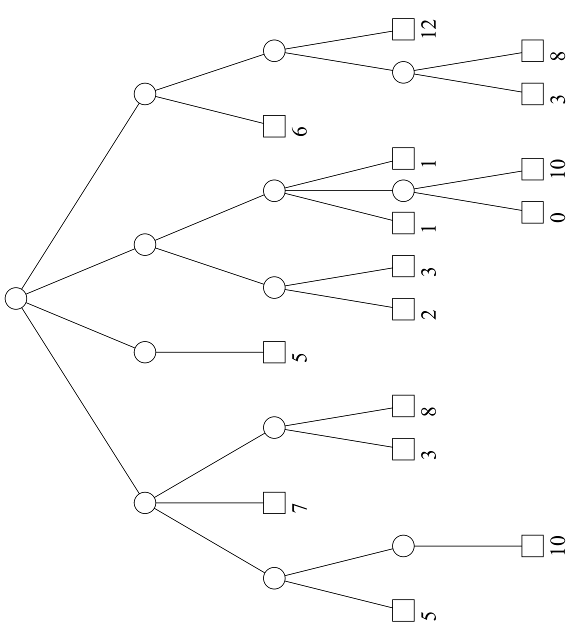
Chaque joueur choisi, à son tour, un nombre parmi 1,2,3. Cependant, il ne peut pas choisir un nombre qui a été choisi par son adversaire dans le tour précedent. Un joueur gagne s'il choisit un nombre qui fait que la somme de tous les nombres (choisis par les deux joueurs) est égale à huit, ou bien si son adversaire est forcé de choisir un nombre qui fait que la somme de tous les nombres dépasse huit. Partie nulle n'est pas possible.

Arbre de jeu pour le jeu de huit

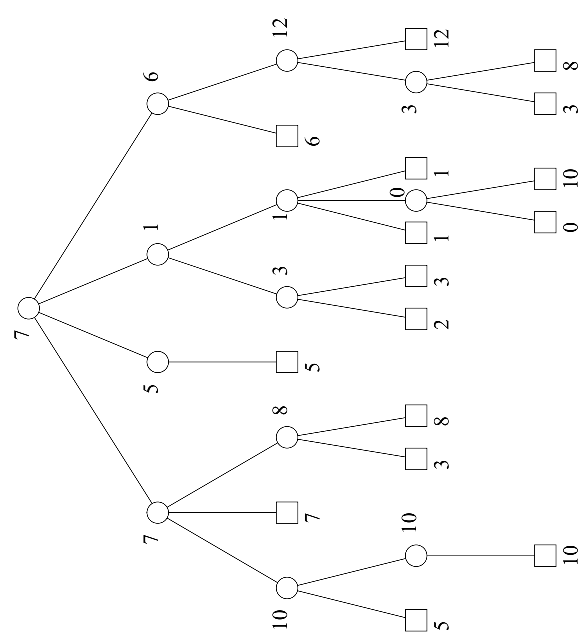


Méthode de minimax

Un arbre de jeu avec de valeurs associées aux feuilles.



L'évaluation d'un arbre de jeu en utilisant la méthode de minimax



Codage basé sur les tries. Compression de données.

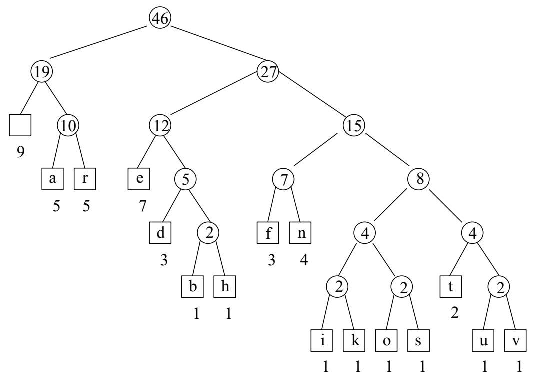
Codage de Huffman

Exemple:

a fast runner need never be afraid of the dark

Caractère		a	b	d	e	f	h	i	k	n	o	r	s	t	u	v
Fréquence	9	5	1	3	7	3	1	1	1	3	1	5	1	2	1	1

Fréquences d'apparition des lettres en français



A	8.40 %	N	7.13 %
B	1.06 %	O	5.26 %
C	3.03 %	P	3.01 %
D	4.18 %	Q	0.99 %
E	17.26 %	R	6.55 %
F	1.12 %	S	8.08 %
G	1.27 %	T	7.07 %
H	0.92 %	U	5.74 %
I	7.34 %	V	1.32 %
J	0.31 %	W	0.04 %
K	0.05 %	X	0.45 %
L	6.01 %	Y	0.30 %
M	2.96 %	Z	0.12 %

Algorithmes à essais successifs (algorithmes à retour, ang. backtrack)

L'approche suggeré pour trouver la solution par une recherche exhaustive dans un espace contenant toutes les solutions logiquement possibles.

Chaque étape correspond à:

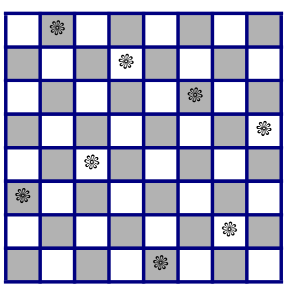
- on a trouvé une solution,

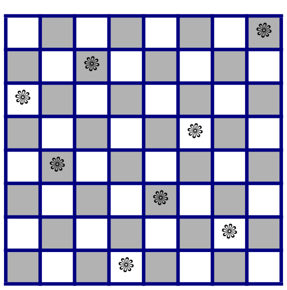
- il y a un certain nombre de continuations possibles qui peuvent nous approcher à la solution,

- aucune continuation n'est possible

Problème de huit reines.

Placer sur l'échiquier huit reines afin qu'elles ne s'attaquent pas. Les deux reines s'attaquent si elles se trouvent sur la même ligne, la même colonne ou sur la même diagonale.





AjouterReine ()
{
    Pour (chaque position libre *p* sur l'échiquier) {
        Placer une reine en position *p*;
        n++;
        si (*n*=8)
            afficher la configuration;
        sinon
            AjouterReine();
        Supprimer la reine de la position *p*;
        n--;
    }
}

```
Programme
{
   n ← 0;
   AjouterReine ();
}
int Col[8];                /* colonne contenant la reine              */
bool ColLibre[8]           /* est-ce la colonne libre?                */
bool DiagMontLibre[15]     /* est-ce la diagonale montante libre?     */
bool DiagDescLibre[15]     /* est-ce la diag. descendante libre?      */

int ligne = -1;            /* dans cette ligne on place la reine      */

void main (void);
{
   int i;

   for (i=0; i<8; i++)
      ColLibre[i] = VRAI;
   for (i=0; i<15; i++) {
      DiagMontLibre[i] = VRAI;
      DiagDescLibre[i] = VRAI;
   }
   AjouterReine();
}
```

Pour chaque diagonale montante la somme des indices i,j est constante (varie de 0 à 14)

Pour chaque diagonale descendante la différence des indices i,j est constante (varie de -7 à 7)


```
/* AjouterReine: essaye de placer les reines, en retournant si nécessaire */
void AjouterReine(void);
{
   int c;                              /* on va essayer de placer        */
                                       /* une reine dans cette colonne */
   ligne++;
   for (c=0; c<8; c++)
      if (ColLibre[c] && DiagMontLibre[ligne+c] &&
            DiagDescLibre[ligne-c+7]) {
         Col[ligne] = c;
         ColLibre[c]= FAUX;
         DiagMontLibre[ligne+c] = FAUX;
         DiagDescLibre[ligne-c+7] = FAUX;
         if (ligne==7)                 /* condition de terminaison       */
            AfficherEchiquier();
         else
            AjouterReine();            /* continuer récursivement        */
         ColLibre[c]= VRAI;
         DiagMontLibre[ligne+c] = VRAI;
         DiagDescLibre[ligne-c+7] = VRAI;
      }
   ligne--;
}
```

Algorithme sans logique: $\binom{64}{8} = 4, 426, 165, 368$ configurations.

Observation qu'il y a seulement une reine par ligne: $8^8 = 16, 777, 216$ configurations.

Observation qu'il y a une reine par colonne: $8! = 40(3230)$ configurations. Dans notre cas - beaucoup moins.