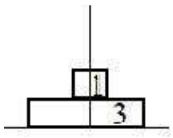
Tours de Hanoi

On a vu en cours (*Algorithmes, chapitre « récursif et itératif »*) une version rudimentaire de ce jeu, où l'on sait seulement de quelle tige vers quelle autre les déplacements sont effectués, sans savoir quel est le disque concerné, sinon qu'il est celui situé le plus haut.

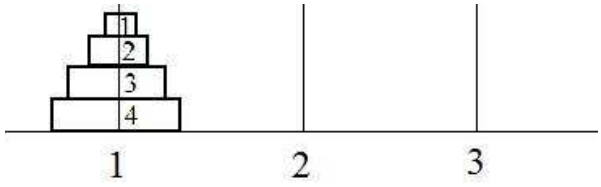
On va maintenant ajouter quels sont les disques déplacés, et pour cela on commence par leur donner un numéro, de 1 à *N*, dans l'ordre de leur rayon croissant. On va aussi gérer la hauteur *H*[*i*] de chaque pile *i*, avec *i* de 1 à 3. Enfin on introduit un tableau à double entrée *p*[*i*][*j*] pour chaque pile *i*, avec *j* variant de 0 à *N* – 1, correspondant à des graduations d'altitude sur la pile concernée. S'il n'y a pas de disque à l'altitude *j*, on met *p*[*i*][*j*] à 0, sinon on met dans *p*[*i*][*j*] le numéro du disque qui se trouve là.



Par exemple, pour *N* = 4, si la pile 1 est de cette forme, on a *H*[1] = 2, et *p*[1][0] = 3, le disque numéro 3 étant à l'altitude 0, *p*[1][1] = 1, *p*[1][2] = 0, *p*[1][3] = 0.

On part de la configuration initiale, et l'on fabrique la fonction *init*() correspondante :

```
void init(void)
  {  int i,j;
     for(j=0;j<N;j++)   p[1][j]=N-j;
     H[1]=N;
    for(i=2;i<=3;i++) for(j=0;j<N;j++)
    p[i][j]=0;
    for(i=2;i<=3;i++)   H[i]=0;
  }
```



Quand un déplacement se produit, à partir d'une certaine pile *i*, c'est le disque le plus haut qui est enlevé, il est à l'altitude *H*[*i*] – 1, et son numéro est enregistré dans *p*[*i*][*H*[*i*]-1]. Il suffit alors de diminuer la hauteur de cette pile d'une unité, puis sachant dans quelle pile *k* le disque va se retrouver, on le place à l'altitude *H*[*k*] en mettant son numéro dans *p*[*k*][*H*[*k*]],et l'on augmente la hauteur *H*[*k*] de 1. D'où le programme complété :

```
#include <conio.h>
#include <stdio.h>
#define N 4
void h(int n,int d,int i,int a);
void init(void);
int compteur=0, p[4][N],H[4], numero;

int main()
{  init();    h(N,1,2,3);
   getch();   return 0;
}

void h(int n,int d,int i,int a)
{  if (n==1) { compteur++; numero=p[d][H[d]-1];  H[d]--;
```

```
            p[a][H[a]]=numero;  H[a]++;
              printf("\n %3.d: le disque %d va de %d a %d",compteur,numero,d,a);
          }
  else  {  h(n-1,d,a,i); h(1,d,i,a); h(n-1,i,d,a); }
}
```

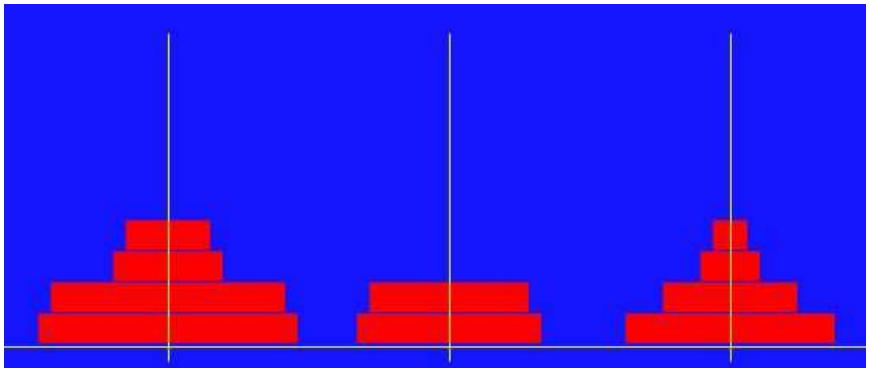
D'où ce résultat pour $N = 4$:

```
  1: le disque 1 va de 1 a 2
  2: le disque 2 va de 1 a 3
  3: le disque 1 va de 2 a 3
  4: le disque 3 va de 1 a 2
  5: le disque 1 va de 3 a 1
  6: le disque 2 va de 3 a 2
  7: le disque 1 va de 1 a 2
  8: le disque 4 va de 1 a 3
  9: le disque 1 va de 2 a 3
 10: le disque 2 va de 2 a 1
 11: le disque 1 va de 3 a 1
 12: le disque 3 va de 2 a 3
 13: le disque 1 va de 1 a 2
 14: le disque 2 va de 1 a 3
 15: le disque 1 va de 2 a 3
```

Il reste enfin à visualiser tout cela. Une fonction *brique*() dessine sous forme de rectangle chaque disque en fonction de sa position et de son numéro. Pour cela on se donne au préalable la hauteur *ha* de chaque disque (*ha* = 20 ici), ainsi que les coordonnées écran du bas de la tige 1 (*x1orig*, *y1orig*), et la distance *L* entre deux tiges (*L* = 180). Puis on réécrit en mode graphique les fonctions précédemment construites, appelées ici *initd*() et *hd*(), ce qui donne au final :

```
int main(int argc, char ** argv)
{  int i;
   SDL_Init(SDL_INIT_VIDEO);
   ecran=SDL_SetVideoMode(800,600,32, SDL_HWSURFACE|SDL_DOUBLEBUF);
   rouge=SDL_MapRGB(ecran->format,255,0,0) ;
   jaune=SDL_MapRGB(ecran->format,255,255,0);
   bleu=SDL_MapRGB(ecran->format,20,20,255);
   SDL_FillRect(ecran,0,bleu);
   for(i=1;i<=N;i++) R[i]=3+8*i;   /* rayon du disque i */
   initd();   hd(N,1,2,3);       pause(); return 0;
}
void brique(int x,int y, int number,Uint32 color)
 {  rectangle= SDL_CreateRGBSurface(SDL_DOUBLEBUF,2*R[number],ha-1,32,0,0,0,0);
    position.x= x; position.y=y;
    SDL_FillRect(rectangle,NULL,color);
    SDL_BlitSurface(rectangle,NULL,ecran,&position);
 }
void initd(void)
{  int i,j;
   for(j=0;j<N;j++)  p[1][j]=N-j;          H[1]=N;
   for(i=2;i<=3;i++)  for(j=0;j<N;j++) p[i][j]=0;       for(i=2;i<=3;i++)  H[i]=0;
   tiges();
   for(i=1;i<=N;i++)   {  brique(x1orig-R[i],y1orig-(N-i+1)*ha,i,rouge); }
   SDL_Flip(ecran);
}
void tiges()
{  ligne(x1orig,y1orig-N*ha,x1orig,y1orig+10,jaune);
```

```
    ligne(x1orig+L,y1orig-N*ha,x1orig+L,y1orig+10,jaune);
    ligne(x1orig+2*L,y1orig-N*ha,x1orig+2*L,y1orig+10,jaune);
    ligne (0,y1orig+1,640,y1orig+1,jaune);
}
void hd(int n,int d,int i,int a)
{  if (n==1)
     { numero=p[d][H[d]-1];
       brique(x1orig+(d-1)*L-R[numero], y1orig-H[d]*ha, numero, bleu);
       H[d]--;    p[a][H[a]]=numero;
       brique(x1orig+(a-1)*L-R[numero], y1orig-(H[a]+1)*ha, numero, rouge);
       H[a]++;
       tiges() ;
       SDL_Flip(ecran);     SDL_Delay(100);  /* pour ralentir le mouvement */
     }
  else { hd(n-1,d,a,i); hd(1,d,i,a); hd(n-1,i,d,a); }
}
```



Pour $N = 10$, les tours en cours de déplacement