نام مقاله :
تعریف الگوریتم برج های هانوی و الکوریتم تابع اصلی جهت جابجایی بین برجها
define hanoy towers algorithm & movement between towers - main function algorithm
کپی رایت :

redterm@yahoo.com
www.redterm.com
Copyright 2007

الگوریتم برج های هانوی

برای شروع ابتدا باید روش پیاده سازی را تعین کنیم . البته در نظر اول و در نهایت ، روش تابع بازگشتی بهترین روش است . چرا که به عنوان مثال فرض کنید که در یک مرحله از مسئله برجهای هانوی دو حرکت امکان پذیر باشد . پیاده سازی روشی که ابتدا به یک حرکت به طور کامل بپردازد و سپس در صورت عدم موفقیت به سراغ روش دوم برود که اکنون دیگر از حالت اولیه در آمده و تغییر کرده به غیر از روش تابع بازگشتی کاری مشکل است .

مشکل دوم چگونگی پیاده سازی خود برجها (ساختمان داده آنها) است . برنامه باید طوری باشد که کاربر بسته به نظر خود در هر مرتبه اجرای برنامه عدد نتفاوتی را برای آن وارد میکند . پس اولین راهی که به نظر میرسد استفاده از حافظه پویاست . یعنی پس از اعلان کاربر مبنی بر تعداد اعضای برج هانوی در حالت اولیه ، با حافظه پویا میتوان حافظه مورد نظر را برای سه برج در نظر گرفت . البته در هر برج ما حافظه ای برار مقدار درخواستی کاربر + 1 را در نظر میگیریم . این حافظه اضافی در ذخیره شماره عنصر سرآمد (یا به عبارتی همان TOP) به ما کمک میکند .

مشکل سوم اینکه به دلیل پیچیدگی الگوریتمی که با روش بازگشتی در مراحل انجام این مسئله پیش می آید حافظه زیادی از Stack اشغال میشود . زیرا در زمان افزایش اعضاء مسئله به همان نسبت تعداد فراخوانی ها افزایش می یابد و در هر مرحله که دو حرکت داشته باشیم نیز فراخوانی ها به دو شاخه تقسیم شده و هر کدام راه خود را می رود . بنابراین مشکل سوم کمبود حافظه است . با شیوه بالا جهت تخصیص حافظه این مشکل نیز حل میشود و تا مراحل بالایی از ورودی کاربر این الگوریتم جواب میدهد .

طبق تجربه ثابت شده که برج هانوی در 2^n-1 مرحله بیهنه ترین جواب را میدهد . پس اگر شرط جهت پایان یکی از راه های رفته در بازخوانی های پیاپی بگذاریم باید روی این مورد بحث کند .

الکوریتم تابع اصلی جهت جابجایی بین برجها به صورت زیر است :

```
#define A_B 1
#define B_A 2
#define A_C 3
#define C_A 4
#define C_B 5
#define B_C 6

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <conio.h>

int n;
char *fifo;

char migrate( char *A , char *B , char *C , char last , int step );

void showfifo(char *fifo);

void main(void)
{
   char *A,*B,*C,i;

   clrscr();

   printf("_____________________ This Soft Ware if Hannoy Problem Solve
_____________________________\n");
   printf("_________________ Please insert how Number you want in Numeric
_____________________\n");

   scanf("%d",&n);

   A=(char *)malloc((sizeof(char)*n)+1);
   B=(char *)malloc((sizeof(char)*n)+1);
   C=(char *)malloc((sizeof(char)*n)+1);
   fifo=(char *)malloc((sizeof(char)*(pow(2,n)-1))+1);

   for(i=1;i<=n;i++)
   {
   A[i]=n+1;
   B[i]=n+1;
   C[i]=n+1;
   }

   for(i=0;i<=pow(2,n);i++)
   fifo[i]=0;

   A[0]=n;
   B[0]=0;
   C[0]=0;
```

```
    for(i=1;i<=n;i++)
    A[n-i+1]=i;

    if(n%2)
    {
    C[++C[0]]=A[A[0]];
    A[0]--;
    B[++B[0]]=A[A[0]];
    A[0]--;
    fifo[++fifo[0]]=A_C;
    fifo[++fifo[0]]=A_B;
    migrate( A , B , C , A_B , 0 );
    }
    else
    {
    B[++B[0]]=A[A[0]];
    A[0]--;
    C[++C[0]]=A[A[0]];
    A[0]--;
    fifo[++fifo[0]]=A_B;
    fifo[++fifo[0]]=A_C;
    migrate( A , B , C , A_C , 0 );
    }

    showfifo(fifo);

    getch();
}

//_____________________________________________________
```

```
//_____________________________________________
//_____________________________________________

char migrate( char *A , char *B , char *C , char last , int step )
{
   char *AL,*BL,*CL;

   if (( step==((int)pow(2,n)-1)) && (C[0]/n==2) )
   return 1;
   else
   if ( step > ( pow(2,n)-1 ))
      return 0;
//_____________________________________________
   AL=(char *)realloc(A,sizeof(char)*n);
   BL=(char *)realloc(B,sizeof(char)*n);
   CL=(char *)realloc(C,sizeof(char)*n);
//_____________________________________________

   if ( A[A[0]] < B[B[0]] )
   {
   if (( last != B_A ) && ( A[0]!=0 ))
   {
      BL[++BL[0]]=AL[AL[0]];
      AL[AL[0]]=n+1;
      AL[0]--;
      if ( ( migrate(AL,BL,CL,A_B,++step) ) )
      {
            fifo[++fifo[0]]=A_B;
            return 1;
      }
   }
   }
   else
   {
   if (( last != A_B ) && ( B[0]!=0 ))
   {
      AL[++AL[0]]=BL[BL[0]];
      BL[BL[0]]=n+1;
      BL[0]--;
      if ( (migrate(AL,BL,CL,B_A,++step) ) )
      {
            fifo[++fifo[0]]=B_A;
            return 1;
      }
   }
   }
```

```
//_____________________________________________
  if ( A[A[0]] < C[C[0]] )
  {
  if (( last != C_A ) && ( A[0]!=0 ))
  {
     CL[++CL[0]]=AL[AL[0]];
     AL[AL[0]]=n+1;
     AL[0]--;
     if ( ( migrate(AL,BL,CL,A_C,++step) ) )
     {
            fifo[++fifo[0]]=A_C;
            return 1;
     }
  }
  }
  else
  {
  if (( last != A_C ) && ( C[0]!=0 ))
  {
     AL[++AL[0]]=CL[CL[0]];
     CL[CL[0]]=n+1;
     CL[0]--;
     if ( (migrate(AL,BL,CL,C_A,++step) ) )
     {
            fifo[++fifo[0]]=C_A;
            return 1;
     }
  }
  }
```

```
//_____________________________________________________

   if ( C[C[0]] < B[B[0]] )
   {
  if (( last != B_C ) && ( C[0]!=0 ))
  {
     BL[++BL[0]]=CL[CL[0]];
     CL[CL[0]]=n+1;
     CL[0]--;
     if ( ( migrate(AL,BL,CL,C_B,++step) ) )
     {
             fifo[++fifo[0]]=C_B;
             return 1;
     }
  }
  }
  else
  {
  if (( last != C_B ) && ( B[0]!=0 ))
  {
     CL[++CL[0]]=BL[BL[0]];
     BL[BL[0]]=n+1;
     BL[0]--;
     if ( (migrate(AL,BL,CL,B_C,++step) ) )
     {
             fifo[++fifo[0]]=B_A;
             return 1;
     }
  }
  }


  free((void *)AL);
  free((void *)BL);
  free((void *)CL);
}
//_____________________________________________________
```

```
//_____________________________________________________
//_____________________________________________________

void showfifo(char *fifo)
{
   char counter;
   for(counter=fifo[0];counter>=0;counter--,printf("\n"))
   switch (fifo[counter])
   {
      case A_B:
             printf("A->B");
             break;
      case B_A:
             printf("B->A");
             break;
      case A_C:
             printf("A->C");
             break;
      case C_A:
             printf("C->A");
             break;
      case C_B:
             printf("C->B");
             break;
      case B_C:
             printf("B->C");
             break;
   }
                                                    }
```