Programmation Impérative II

Fonction récursive

minh-anh.tran@u-pec.fr

Fonction récursive

- Récursivité
- Exemple: Factoriel
- Fonction récursive vs. Fonction itérative
- Algorithme de tri par sélection
- Recherche dichotomique

Récursivité

Traitement récursif

Faire un traitement identique sur des données différentes jusqu'à la *condition de terminaison*

Fonction récursive

fonction qui appelle elle-même si la condition de terminaison n'est pas vérifiée

pas d'initialisation des variables locales
initialisation et transmission des valeurs
par l'intermédiaire des paramètres de la fonction

Exemple: Factoriel

Fact(n)=Fact(n-1)*n si n >1
Fact(1)=1

Calculer Fact(n) = 1*2*...*n

Fonction récursive

```
int Fact (int i)
{
   if (i == 1) return (1);
   else return (i * Fact (i-1));
     }
```

Premier appel dans le programme principal

```
printf ("4! = %d\n", Fact (4));
```

Exemple: Factoriel

Fonction récursive

```
int Fact (int i)
{
  if (i == 1) return (1);
  else return (i * Fact (i-1));
}
```

Premier appel

```
printf ("4! = %d\n", Fact (4));
```

Les appels pour Fact(4)

1- i=4 Fact(4) = 4*Fact(3)

4-return(4*6=24)

dernière valeur retournée

2- i=3 Fact(3) = 3*Fact(2)

3-return(3*2=6)

3- i=2 Fact(2) = 2*Fact(1)

2- return(2*1=2)

4- i=1 condition de terminaison

1- return(1)

Fonction récursive vs. fonction itérative

Fonction itérative

```
int Fact_iter (int N)
{
    int j = 1, resultat = 1;
    while (j < N)
    { j++;
      resultat = resultat * j;}
    return (resultat);
}
```

phase d'initialisation: par les paramètres de fonction

while : appels récursifs

Algorithme de tri par sélection

Problème. Trier un tableau T de taille n:

T[0], T[1], ... T[n-1]

Tri par sélection

On trouve le plus petit élément, on le place

On continue avec le second plus petit, etc.

Implémentation

- tri itératif
- tri récursif

Tri itératif par sélection

```
void tri_select_iter (int T[], int n)
{
        int i, j, k, aux;
        for (i=0; i<n-1; i++)
        {
                k=i;
                for (j=i+1; j<=n-1; j++)
                {
                        if (T[j] < T[k])
                                {k = j;}
                }
                aux=T[i]; T[i]=T[k]; T[k]=aux;
        }
}
```

Pour un tableau T de taille n, trier tous ses éléments T[0], T[1], ... T[n-1]

Trouver k: l'indice du plus petit élément d'entre T[i], T[i+1], ... T[n-1]

Echanger T[i] et T[k]

Algorithme de tri par sélection

```
void triselectit(int T[], int n)
{int i,k,j,aux;
  for (i=0; i<n-1;i++)
     {k=i;
      for (j=i+1; j<=n-1;j++)     if (T[j] <T[k]) k=j;
      aux=T[i]; T[i]=T[k]; T[k]=aux;
     }
}
main()
{  int T[6],i;
  for (i=0;i<6;i++)   scanf("%d",&T[i]);
  triselectit( T,6);
  printf("RESULTAT\n");
for (i=0;i<6;i++)   printf("%5d",T[i]);
}
```

Tri récursif par sélection

```
void tri_select_rec (int i, int T[], int n)
{
        int j, k, aux;
        if (i < n-1)
        {
                k=i;
                for (j=i+1; j<=n-1; j++)
                {
                        if (T[j] <T[k]) { k=j; }
                }
                aux=T[i]; T[i]=T[k]; T[k]=aux;

                tri_select_rec (i+1, T, n);
        }
}
```

Pour un tableau T de taille n, trier les éléments suivants T[i], T[i+1], ... T[n-1]

Trouver k: la position du plus petit élément d'entre T[i], T[i+1], ... T[n-1]

Echanger T[i] et T[k]

Appel récursif

Algorithme de tri par sélection

```
void tri_select_rec (int i, int T[], int n)
{
        int j, k, aux;
        if (i == n-1) {}
        else
    {k=i;
     for (j=i+1; j<=n-1;j++)     if (T[j] <T[k]) k=j;
     aux=T[i]; T[i]=T[k]; T[k]=aux;
     tri_select_rec(i+1,T,n);      }
}
main()
{  int T[6],i;
  for (i=0;i<6;i++)   scanf("%d",&T[i]);
  triselrec( 0,T,6);
  printf("RESULTAT\n");
```

Diviser pour Conquérir

1- Diviser le problème en sous-problèmes
de taille plus petite

2- Trouver des solutions des sous-problèmes

3- Trouver la solution globale à partir
des sous-solutions

Il est possible de simplifier la solution de certains problèmes par cette approche

Recherche dichotomique

Paramètres d'entrée :

Tableau T (éléments sont ordonnés)

$T[0] < T[1] < \ldots < T[n]$

X: élément à chercher dans le tableau

Sous-problème : chercher X dans le sous-tableau

$T[g] < T[g+1] < \ldots < T[d]$

g: indice du premier élément

d: indice du dernier élément

Recherche dichotomique

Dicho(T,g,d,X)

si (g>d) **alors** X n'est pas dans T

sinon Diviser le tableau T(g,d) en 2
T(g,m-1) et T(m+1,d) où m=(g+d)/2

si (X=T[m]) **alors** trouvé
sinon si (X < T[m]) **alors** chercher dans le 1er
sinon chercher dans le 2ème

```
int dicho (int T[], int g, int d, int X)
{int m;
  if (g <= d)
    {m = (g+d)/2;
      if (X==T[m]) return (m);
      else
        if (X<T[m]) return (dicho(T,g,m-1,X));
        else return (dicho(T,m+1,d,X));
    }
  else return (-1);
}

int main(void)
{int T[6], i;
  for (i=0; i<=5; i++) scanf("%d",&T[i]);
  printf("%d\n", dicho(T,0,5,12));}
```

```
int dicho (int T[], int g, int d, int X)
{
        int m;
        if (g <= d)
        {
                m = (g + d) / 2;
                if (X == T[m])
                {
                        return (m);
                }
                else
                {
                        if (X < T[m])
                        {
                                return (dicho(T, g, m-1, X));
                        }
                        else
                        {
                                return (dicho(T, m+1, d, X));
                        }
                }
        }
        else return (-1);
}
```

Une bonne habitude:

Mettre des parenthèses **{}** après chaque

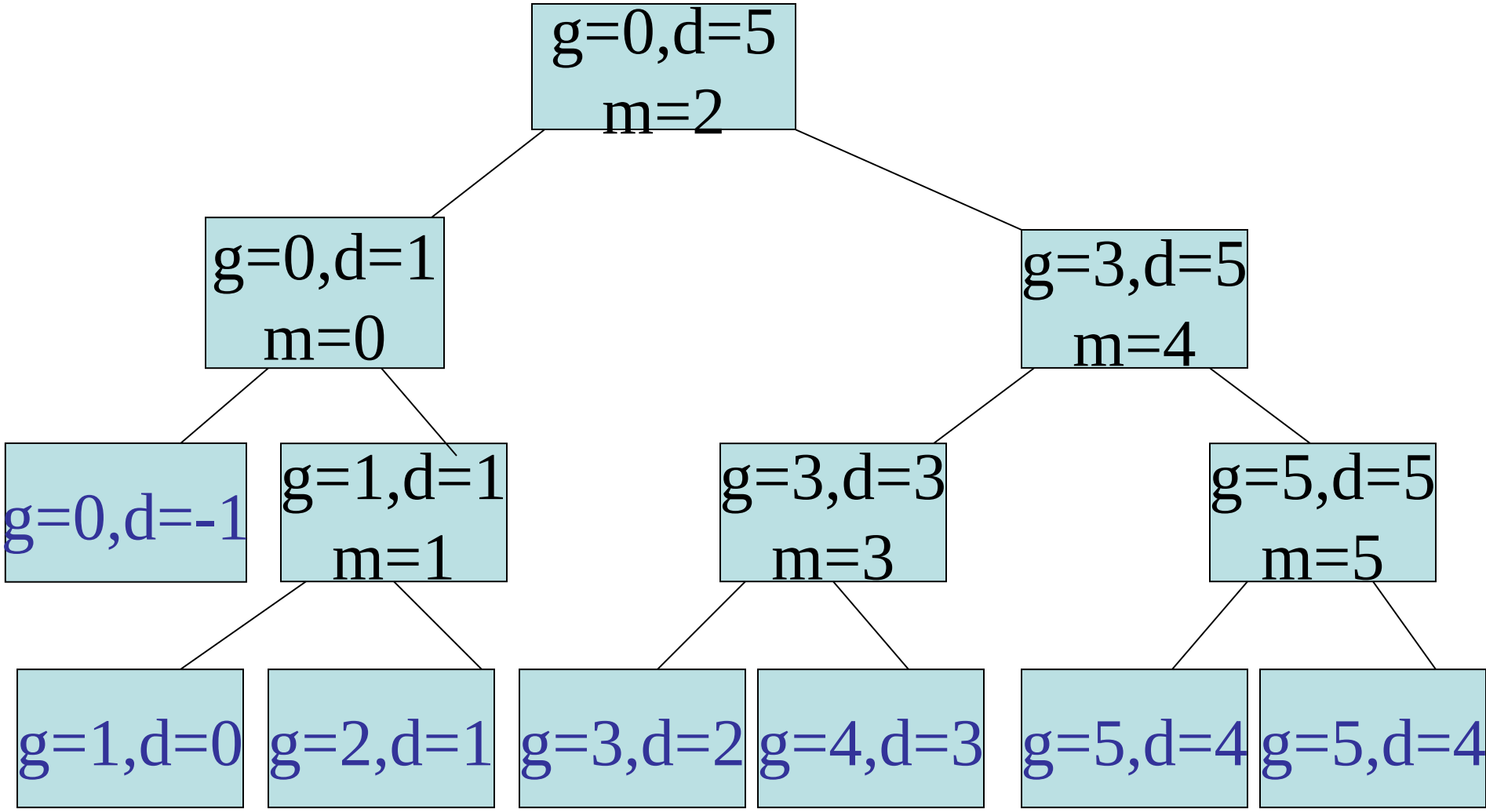
if else for while

même s'il y a une seule instruction qui suit

Bien !

Pas bien !

Les appels pour un tableau de taille 6



Recherche dichotomique

Nombre de comparaisons (T[m]==X)
dans le pire des cas
si X n'est pas dans T

Recherche dichotomique = 3
Recherche séquentielle = 5

Recherche dichotomique

Nombre de comparaisons (T[m]==X)
dans le pire des cas
si X n'est pas dans T

Ce nombre est de l'ordre $\log_2$(taille de X)