ricorsione 2

fattoriale calcolato al ritorno e all'andata che è più simile al calcolo iterativo

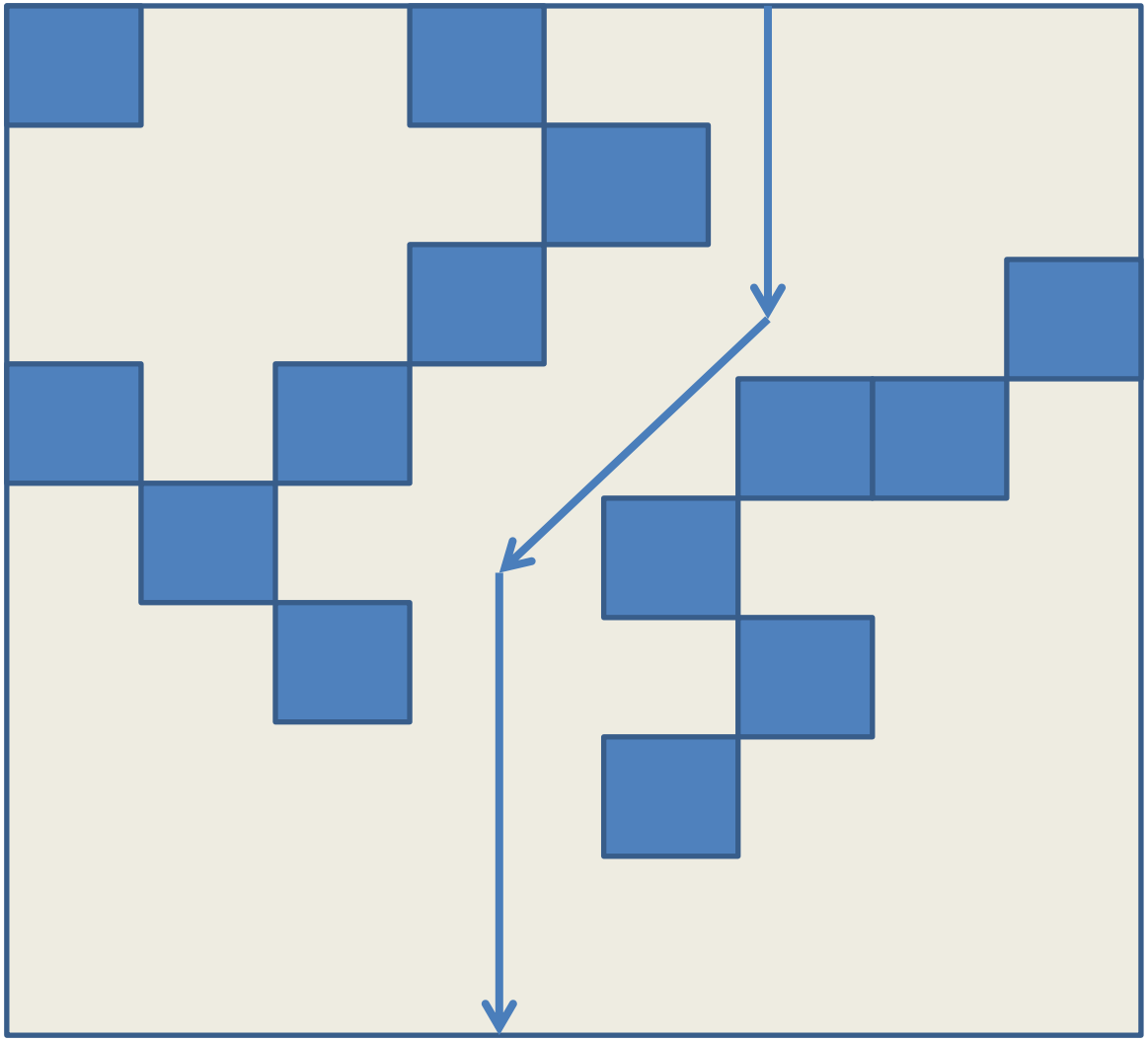
```
int fatt(int n)
{
  if (n==0 || n==1) return 1;
  else
  return n*fatt(n-1);   //   calcola * al ritorno
}

int fatt=1;
while (n>1)
{
   fatt=fatt*n;
   n=n-1;
}

int fatt(int n, int ris)
{
  if (n==0 || n==1) return ris;
  else
  return fatt(n-1, ris*n);  //    calcola * all'andata
}
```

bool S[10][10]

x=n



n
n
n
n-1
n-2
n-2
n-2
..

y

determina se un cammino esiste

```
bool cerca(bool S[][10], int x, int y)
{
 if(x<0 ||  x>9 || !S[y][x])
   return false;
if(y==9) return true;
 return cerca(S,x-1,y+1) || cerca(S,x,y+1) ||
cerca(S,x+1,y+1);}
```

costruiamo il cammino C

PRE=(C ha 10-y elementi)&&(0<=y<=9)&&(-1<=x<=10)

```
bool cercac(bool S[][10], int x, int y, int*C)
{ if(x<0 ||  x>9 || !S[y][x])
   return false;
if(y==9) {*C=x; return true;}
 bool a=cercac(S,x-1,y+1,C+1) || cercac(S,x,y+1,C+1)
|| cercac(S,x+1,y+1,C+1);
if(a) *C=x;
return a;
}
```

POST=(se true=>C[0..10-y-1]=camm. a sinistra che inizia con x, se false=>non esiste cammino da x)

abbiamo bisogno di un programma “di lancio” che invochi cerca a partire da ogni elemento della riga 0:

PRE=(S def e C ha 10 elementi, 0<=i<=10)

```
bool lancia( bool S[][10],int i, int*C)
{
  if(i>=10) return false;
  return cercac(S,i,0,C) || lancia(S,i+1,C);
}
```

POST=(true=>C è cammino più a sinistra
false=>non c’è cammino)

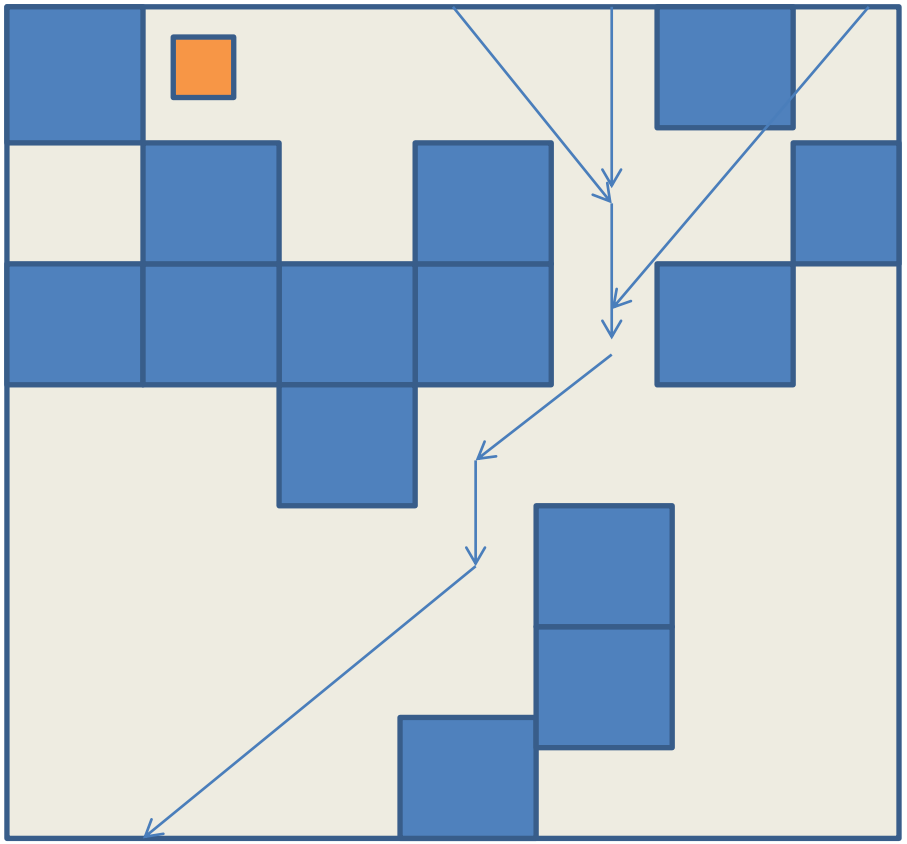
```
main()
{
 bool S[10][10];
int C[10];
//definisci S da “input”
if(lancia(S,0,C))
   //stampa C
else
  //nessun cammino
}
```

possiamo fare un programma simile con l’iterazione ?

Si, ma non è facile e ci serve capire bene che succede con la soluzione ricorsiva

bool S[10][10]

x

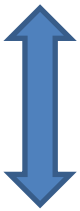


y

da ▫ pila dei dati

x=1,y=0, rit=lancia

x=0,y=1,rit=secondo



memoria degli stati precedenti del calcolo serve a tornare indietro e seguire altre strade

punti del programma PP

```
bool cercac(bool S[][10], int x, int y, int*C)
{ if(x<0 ||  x>9 || !S[x][y])
   return false;
if(y==9) {*C=x; return true;}
 bool a=cercac(S,x-1,y+1,C+1)||
cercac(S,x,y+1,C+1) || cercac(S,x+1,y+1,C+1);
if(a) *C=x;
return a;
}
```

(primo)

(secondo)

(terzo)

(quarto)

```
enum PP{primo, secondo, terzo, quarto};
struct pila{PP dove; int x,y; pila(PP a=0,int b=0, int
c=0){dove=a; x=b; y=c;}};
pila P[10];  //simula la pila della ricorsione
int C[10]; // cammino
P[0]=pila(primo,1,0); int cima=1; bool trovato=false;
while(cima && ! trovato)
{
 switch(P[cima-1].dove)
  {
.....//vedi dopo
  }
}POST=(trovato=> C è ok)&&(!trovato=> no camm)
```

```
case primo:
{if(P[cima-1].x <0 ||P[cima-1].x>9 || !S[y][x])
cima=cima-1;
else
   if(P[cima-1].y==9){C[y]=x; trovato=true;}
else
{  C[y]=x;
   P[cima-1].dove=secondo; P[cima]=pila(primo,x-1,
         y+1); cima++;
}
break;
}
```

```
case secondo:
{ P[cima-1].dove=terzo;
  P[cima]=pila(primo,x,y+1);
  cima++;
  break;
}
case terzo:
{P[cima-1].dove=quarto;
 P[cima]=pila(primo,x+1,y+1);
 cima++;
 break;
}
```

```
case quarto:
{ cima--;
  break;
}
```

il lancio iterativo:

```
bool trovato=false; int cima=0;
for(int i=0; i<10&&!trovato; i++)
{
 P[cima]=pila(primo,i,0); cima=1;
while(cima && ! trovato)
{
 switch(P[cima-1].dove)
  {   }
}
}
```