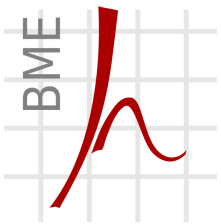
Fordítás – Kódoptimalizálás

Kód visszafejtés.



Híradástechnikai Tanszék

Izsó Tamás

2016. október 27.

Quicksort program

```
void quicksort(int *a, int m, int n )
{
    int i, j;
    int v, x;
    if( n <= m ) return;
    i=m-1; j=n; v = a[n];
    for(;;) {
        do i = i+1; while( a[i] < v );
        do j = j-1; while( a[j] > v );
        if( i >= j ) break;
        x = a[i]; a[i]= a[j]; a[j]=x;
    }
    x=a[i]; a[i]=a[n]; a[n]=x;
    quicksort(a,m,j);
    quicksort(a,i+1,n);
}
```

Lefordított kód + közbenső ábrázolás

```
_quicksort:
1    push        ebp
2    mov         ebp,esp
3    sub         esp,28h
-------------------------------------------------------------
4    mov         eax,dword ptr [ebp+10h]
5    cmp         eax,dword ptr [ebp+0Ch]
6    jg          B1
7    jmp         B10
  B1:
8    mov         eax,dword ptr [ebp+0Ch]
9    sub         eax,1
10   mov         dword ptr [ebp-0Ch],eax
11   mov         eax,dword ptr [ebp+10h]
12   mov         dword ptr [ebp-10h],eax
13   mov         eax,dword ptr [ebp+10h]
14   lea         edx,[eax*4+00000000h]
15   mov         eax,dword ptr [ebp+8]
16   add         eax,edx
17   mov         eax,dword ptr [eax]
18   mov         dword ptr [ebp-14h],eax
  B2:
19   add         dword ptr [ebp-0Ch],1
20   mov         eax,dword ptr [ebp-0Ch]
21   lea         edx,[eax*4+00000000h]
22   mov         eax,dword ptr [ebp+8]
23   add         eax,edx
24   mov         eax,dword ptr [eax]
25   cmp         eax,dword ptr [ebp-14h]
26   jl          B2
  B3:
27   sub         dword ptr [ebp-10h],1
28   mov         eax,dword ptr [ebp-10h]
29   lea         edx,[eax*4+00000000h]
30   mov         eax,dword ptr [ebp+8]
31   add         eax,edx
32   mov         eax,dword ptr [eax]
33   cmp         eax,dword ptr [ebp-14h]
34   jg          B3
25   mov         eax,dword ptr [ebp-0Ch]
36   cmp         eax,dword ptr [ebp-10h]
37   jl          B7
  B4:
38   nop
39   mov         eax,dword ptr [ebp-0Ch]
40   lea         edx,[eax*4+00000000h]
41   mov         eax,dword ptr [ebp+8]
42   add         eax,edx
43   mov         eax,dword ptr [eax]
44   mov         dword ptr [ebp-18h],eax
45   mov         eax,dword ptr [ebp-0Ch]
46   lea         edx,[eax*4+00000000h]
47   mov         eax,dword ptr [ebp+8]
48   add         edx,eax
49   mov         eax,dword ptr [ebp+10h]
50   lea         ecx,[eax*4+00000000h]
51   mov         eax,dword ptr [ebp+8]
52   add         eax,ecx
53   mov         eax,dword ptr [eax]
54   mov         dword ptr [edx],eax
55   mov         eax,dword ptr [ebp+10h]
56   lea         edx,[eax*4+00000000h]
57   mov         eax,dword ptr [ebp+8]
58   add         edx,eax
59   mov         eax,dword ptr [ebp-18h]
60   mov         dword ptr [edx],eax
```

```
_quicksort:
1    push ebp
2    ebp = esp
3    esp = esp-0x28
-------------------------------------------------------------
4    eax = eax,dword ptr [ebp+0x10]
5    cmp  eax,dword ptr [ebp+0x0C]
6    jg   B1
7    jmp  B10
  B1:
8     eax =  dword ptr [ebp+0x0C]
9     eax = eax - 1
10   dword ptr [ebp-0x0C] = eax
11   eax = dword ptr [ebp+0x10]
12   dword ptr [ebp-0x10] = eax
13   eax = dword ptr [ebp+0x10]
14   edx = eax*4
15   eax = dword ptr [ebp+8]
16   eax = eax + edx
17   eax = dword ptr [eax]
18   dword ptr [ebp-0x14] = eax
  B2:
19   dword ptr [ebp-0x0C] = dword ptr [ebp-0x0C] + 1
20   eax = dword ptr [ebp-0x0C]
21   edx = eax*4
22   eax = dword ptr [ebp+8]
23   eax = eax + edx
24   eax = dword ptr [eax]
25   cmp  eax,dword ptr [ebp-0x14]
26   jl   B2
  B3:
27   dword ptr [ebp-0x10] = dword ptr [ebp-0x10] - 1
28   eax = dword ptr [ebp-0x10]
29   edx = eax*4
30   eax = dword ptr [ebp+8]
31   eax = eax +edx
32   eax = dword ptr [eax]
33   cmp  eax,dword ptr [ebp-0x14]
34   jg   B3
25   eax = dword ptr [ebp-0x0C]
36   cmp  eax,dword ptr [ebp-0x10]
37   jl   B7
  B4:
38   nop
39   eax = dword ptr [ebp-0x0C]
40   edx =eax*4
41   eax = dword ptr [ebp+8]
42   eax = eax + edx
43   eax = dword ptr [eax]
44   dword ptr [ebp-0x18] = eax
45   eax = dword ptr [ebp-0x0C]
46   edx = eax*4
47   eax,dword ptr [ebp+8]
48   edx = edx + eax
49   eax= dword ptr [ebp+0x10]
50   ecx = eax*4
51   eax = dword ptr [ebp+8]
52   eax = eax+ecx
53   eax = dword ptr [eax]
54   dword ptr [edx]= eax
55   eax = dword ptr [ebp+0x10]
56   edx = eax*4
57   eax = dword ptr [ebp+8]
58   edx = edx + eax
59   eax=dword ptr [ebp-0x18]
60   dword ptr [edx] = eax
```

```
-------- rekurziv hivas 1 --------
61   mov         eax,dword ptr [ebp-10h]
62   mov         dword ptr [esp+8],eax
63   mov         eax,dword ptr [ebp+0Ch]
64   mov         dword ptr [esp+4],eax
65   mov         eax,dword ptr [ebp+8]
66   mov         dword ptr [esp],eax
67   call        _quicksort
  B5:
-------- rekurziv hivas 2 --------
68   mov         eax,dword ptr [ebp-0Ch]
69   lea         edx,[eax+1]
70   mov         eax,dword ptr [ebp+10h]
71   mov         dword ptr [esp+8],eax
72   mov         dword ptr [esp+4],edx
73   mov         eax,dword ptr [ebp+8]
74   mov         dword ptr [esp],eax
75   call        _quicksort
  B6:
76   jmp         B10
  B7:
77   mov         eax,dword ptr [ebp-0Ch]
78   lea         edx,[eax*4+00000000h]
79   mov         eax,dword ptr [ebp+8]
80   add         eax,edx
81   mov         eax,dword ptr [eax]
82   mov         dword ptr [ebp-18h],eax
83   mov         eax,dword ptr [ebp-0Ch]
84   lea         edx,[eax*4+00000000h]
85   mov         eax,dword ptr [ebp+8]
86   add         edx,eax
87   mov         eax,dword ptr [ebp-10h]
88   lea         ecx,[eax*4+00000000h]
89   mov         eax,dword ptr [ebp+8]
90   add         eax,ecx
91   mov         eax,dword ptr [eax]
92   mov         dword ptr [edx],eax
93   mov         eax,dword ptr [ebp-10h]
94   lea         edx,[eax*4+00000000h]
95   mov         eax,dword ptr [ebp+8]
96   add         edx,eax
97   mov         eax,dword ptr [ebp-18h]
98   mov         dword ptr [edx],eax
99   jmp         B2
B10
100  leave
101  ret
```

```
-------- rekurziv hivas 1 --------
61   eax = dword ptr [ebp-0x10]
62   dword ptr [esp+8] = eax
63   eax=dword ptr [ebp+0x0C]
64   dword ptr [esp+4] = eax
65   eax = dword ptr [ebp+8]
66   dword ptr [esp] = eax
67   call   _quicksort
  B5:
-------- rekurziv hivas 2 --------
68   eax = dword ptr [ebp-0x0h]
69   edx = eax+1
70   eax = dword ptr [ebp+0x10]
71   dword ptr [esp+8] = eax
72   dword ptr [esp+4] = edx
73   eax = dword ptr [ebp+8]
74   dword ptr [esp] = eax
75   call        _quicksort
  B6:
76   jmp  B10
  B7:
77   eax = dword ptr [ebp-0x0C]
78   edx = eax*4
79   eax = dword ptr [ebp+8]
80   eax = eax + edx
81   eax = dword ptr [eax]
82   dword ptr [ebp-0x18] = eax
83   eax = dword ptr [ebp-0x0C]
84   edx = eax*4
85   eax = dword ptr [ebp+8]
86   edx = edx+eax
87   eax = dword ptr [ebp-0x10]
88   ecx = eax*4
89   eax = dword ptr [ebp+8]
90   eax = eax+ecx
91   eax = dword ptr [eax]
92   dword ptr [edx] = eax
93   eax = dword ptr [ebp-0x10]
94   edx = eax*4
95   eax=dword ptr [ebp+8]
96   edx = edx+eax
97   eax=dword ptr [ebp-0x18]
98   dword ptr [edx]=eax
99    jmp   B2
B10
100  leave
101  ret
```

Utasítások szemantikája

Példa: sub esp,28h

```
iclass SUB       category BINARY ISA-extension BASE       ISA-set I86
instruction-length 3
operand-width 32
effective-operand-width 32
effective-address-width 32
stack-address-width 32
iform-enum-name SUB_GPRv_IMMb
iform-enum-name-dispatch (zero based) 8
iclass-max-iform-dispatch 18
Operands
#    TYPE       DETAILS          VIS       RW    OC2  BITS  BYTES NELEM ELEMSZ   ELEMTYPE
#    ====       =======          ===       ==    ===  ====  ===== ===== ======   ========
0    REG0       REG0=ESP     EXPLICIT      RW     V    32      4     1      0      INT
1    IMM0       0x28(8b)     EXPLICIT      R      B     8      1     1      8      INT
2    REG1       REG1=EFLAGS SUPPRESSED W       Y    32      4     1      0      INT
Memory Operands
  MemopBytes = 0
FLAGS:
  must-write-rflags of-mod sf-mod zf-mod af-mod pf-mod cf-mod
  read:              mask=0x0
  written:           of sf zf af pf cf  mask=0x8d5
  undefined:         mask=0x0
```

Intel Program Instrumentation Tool által előállított információ.

`https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool`

Lefordított kód + közbenső ábrázolás részlet

```
_quicksort:
1     push        ebp
2     mov         ebp,esp
3     sub         esp,28h
-------------------------------------------------
4     mov         eax,dword ptr [ebp+10h]
5     cmp         eax,dword ptr [ebp+0Ch]
6     jg          B1
7     jmp         B10
  B1:
8     mov         eax,dword ptr [ebp+0Ch]
9     sub         eax,1
10    mov         dword ptr [ebp-0Ch],eax
11    mov         eax,dword ptr [ebp+10h]
12    mov         dword ptr [ebp-10h],eax
13    mov         eax,dword ptr [ebp+10h]
14    lea         edx,[eax*4+00000000h]
15    mov         eax,dword ptr [ebp+8]
16    add         eax,edx
17    mov         eax,dword ptr [eax]
18    mov         dword ptr [ebp-14h],eax
  B2:
```

```
_quicksort:
1     esp = esp - 4
      [esp] = ebp
2     ebp = esp
3     esp = esp-0x28
-------------------------------------------------
4     eax = eax,dword ptr [ebp+0x10]
5     cmp   eax,dword ptr [ebp+0x0C]
6     jg    B1
7     jmp   B10
  B1:
8     eax =  dword ptr [ebp+0x0C]
9     eax = eax - 1
10    dword ptr [ebp-0x0C] = eax
11    eax = dword ptr [ebp+0x10]
12    dword ptr [ebp-0x10] = eax
13    eax = dword ptr [ebp+0x10]
14    edx = eax*4
15    eax = dword ptr [ebp+8]
16    eax = eax + edx
17    eax = dword ptr [eax]
18    dword ptr [ebp-0x14] = eax
  B2:
```

Regiszter másolatok terjedésének a megszüntetése

```
1   esp = esp-4                          du(esp,{1,2})   ud(ebp,?) ud(esp,? )
    [esp]=ebp
2   ebp = esp                            du(ebp,{4,5,...})  ud(esp,?)
3   esp = esp-0x28                       du(esp, ...)    ud(esp,?)
4   eax = dword ptr [ebp+0x10]           du(eax,5)       ud(ebp,2)
5   cmp   eax,dword ptr [ebp+0x0C]       du(SF,6) du(OF,{}) ... ud(eax,4)  ud(ebp,2)
6   jg    B1                                            uf(SF,5)
7   jmp   B10
B1
```

```
1   esp = esp-4                          du(esp,{1,2})   ud(ebp,?) ud(esp,? )
    [esp]=ebp
2   ebp = esp                            du(ebp,{4,5,...})  ud(esp,?)
3   esp = esp-0x28                       du(esp, ...)    ud(esp,?)
4   eax = dword ptr [ebp+0x10]                           ud(ebp,2)
5   cmp   dword ptr [ebp+0x10] ,dword ptr [ebp+0x0C]     du(SF,6)  ...  ud(ebp,2)
6   jg    B1                                             uf(SF,5)
7   jmp   B10
B1
```

Feltételkód továbbterjedésének a megszüntetése

```
1   esp = esp-4                                  du(esp,{1,2})   ud(ebp,?) ud(esp,?)
    [esp]=ebp
2   ebp = esp                                    du(ebp,{4,5,...})  ud(esp,?)
3   esp = esp-0x28                               du(esp, ...)    ud(esp,?)
4   eax = dword ptr [ebp+0x10]                                   ud(ebp,2)
5   cmp   dword ptr [ebp+0x10],dword ptr [ebp+0x0C]    du(SF,6)   ...   ud(ebp,2)
6   jg    B1                                                     uf(SF,5)
7   jmp   B10
B1
```

```
1   esp = esp-4                                  du(esp,{1,2})   ud(ebp,?) ud(esp,?)
    [esp]=ebp
2   ebp = esp                                    du(ebp,{4,5,...})  ud(esp,?)
3   esp = esp-0x28                               du(esp, ...)    ud(esp,?)
4   eax = dword ptr [ebp+0x10]                                   ud(ebp,2)
5   cmp   dword ptr [ebp+0x10],dword ptr [ebp+0x0C]                 ...   ud(ebp,2)
6   jcond(dword ptr [ebp+0x10] > dword ptr [ebp+0x0C])    B1
7   jmp   B10
B1
```

Felesleges utasítások törlése

```
1   esp = esp-4                                  du(esp,{1,2})   ud(ebp,?) ud(esp,? )
    [esp]=ebp
2   ebp = esp                                    du(ebp,{4,5,...})  ud(esp,?)
3   esp = esp-0x28                               du(esp, ...)   ud(esp,?)
4   eax = dword ptr [ebp+0x10]                                  ud(ebp,2)
5   cmp   dword ptr [ebp+0x10],dword ptr [ebp+0x0C]                              ...   ud(ebp,2)
6   jcond(dword ptr [ebp+0x10] > dword ptr [ebp+0x0C])    B1
7   jmp   B10
B1
```

```
1   esp = esp-4                                  du(esp,{1,2})   ud(ebp,?) ud(esp,? )
    [esp]=ebp
2   ebp = esp                                    du(ebp,{4,5,...})  ud(esp,?)
3   esp = esp-0x28                               du(esp, ...)   ud(esp,?)
4
5
6   jcond(dword ptr [ebp+0x10] > dword ptr [ebp+0x0C])    B1
7   jmp   B10
B1
```

Regiszterek által kijelölt területek átnevezése

```
1    esp = esp-4                                  du(esp,{1,2})    ud(ebp,?) ud(esp,? )
     [esp]=ebp
2    ebp = esp                                    du(ebp,{4,5,...})  ud(esp,?)
3    esp = esp-0x28                               du(esp, ...)    ud(esp,?)
4
5
6    jcond(dword ptr [ebp+0x10] > dword ptr [ebp+0x0C])     B1
7    jmp    B10
B1
```

```
1    esp = esp-4                                  du(esp,{1,2})    ud(ebp,?) ud(esp,? )
     [esp]=ebp
2    ebp = esp                                    du(ebp,{4,5,...})  ud(esp,?)
3    esp = esp-0x28                               du(esp, ...)    ud(esp,?)
4
5
6    jcond(par2 > par1)     B1
7    jmp    B10
B1
```

Komplex példa 1

```
39  eax = dword ptr [ebp-0x0C]     du(eax,40)
40  edx =eax*4                     du(edx,42) ud(eax,39)
41  eax = dword ptr [ebp+8]        du(eax,41)
42  eax = eax + edx                du(eax,43), ud(eax,41), ud(edx,40)
43  eax = dword ptr [eax]          du(eax,44)  ud(eax,42)
44  dword ptr [ebp-0x18] = eax                 ud(eax,43)
45  eax = dword ptr [ebp-0x0C]     du(eax,46)
46  edx = eax*4                    du(edx,48)  ud(eax,45)
47  eax = dword ptr [ebp+8]        du(eax,48)
48  edx = edx + eax                du(edx,48)  ud(eax,47) ud(edx,46)
49  eax= dword ptr [ebp+0x10]      du(eax,50)
50  ecx = eax*4                    du(ecx,52)  ud(eax,49)
51  eax = dword ptr [ebp+8]        du(eax, 52)
52  eax = eax+ecx                  du(eax,53)  ud(eax,51) ud(ecx,50)
53  eax = dword ptr [eax]          du(eax,54)  ud(eax,52)
54  dword ptr [edx]= eax                       ud(eax,53) ud(edx,48)
55  eax = dword ptr [ebp+0x10]     du(eax,56)
56  edx = eax*4                    du(edx,58)  ud(eax,55)
57  eax = dword ptr [ebp+8]        du(eax,58)
58  edx = edx + eax                du(edx,60)  ud(edx,56) ud(eax(57)
59  eax=dword ptr [ebp-0x18]       du(eax,60)
60  dword ptr [edx] = eax                      ud(eax,59)
```

```
39  eax = dword ptr [ebp-0x0C]
40  edx =dword ptr [ebp-0x0C]*4
41  eax = dword ptr [ebp+8]
42  eax = dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4
43  eax = dword ptr [dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4]
44  dword ptr [ebp-0x18] = dword ptr [dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4]
45  eax = dword ptr [ebp-0x0C]
46  edx = dword ptr [ebp-0x0C]*4
47  eax = dword ptr [ebp+8]
48  edx = dword ptr [ebp-0x0C]*4 + dword ptr [ebp+8]
49  eax= dword ptr [ebp+0x10]
50  ecx = dword ptr [ebp+0x10]*4
51  eax = dword ptr [ebp+8]
52  eax = dword ptr [ebp+8] + dword ptr [ebp+0x10]*4
53  eax = dword ptr [dword ptr [ebp+8] + dword ptr [ebp+0x10]*4]
54  dword ptr [dword ptr [ebp-0x0C]*4 + dword ptr [ebp+8] ]= dword ptr [dword ptr [ebp+8] + dword ptr [ebp+0x10]*4]
55  eax = dword ptr [ebp+0x10]
56  edx = dword ptr [ebp+0x10]*4
57  eax = dword ptr [ebp+8]
58  edx = dword ptr [ebp+0x10]*4 + dword ptr [ebp+8]
59  eax=dword ptr [ebp-0x18]
60  dword ptr [dword ptr [ebp+0x10]*4 + dword ptr [ebp+8]] = dword ptr [ebp-0x18]
```

Komplex példa 2

```
39  eax = dword ptr [ebp-0x0C]
40  edx =dword ptr [ebp-0x0C]*4
41  eax = dword ptr [ebp+8]
42  eax = dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4
43  eax = dword ptr [dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4]
44  dword ptr [ebp-0x18] = dword ptr [dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4]
45  eax = dword ptr [ebp-0x0C]
46  edx = dword ptr [ebp-0x0C]*4
47  eax = dword ptr [ebp+8]
48  edx = dword ptr [ebp-0x0C]*4 + dword ptr [ebp+8]
49  eax= dword ptr [ebp+0x10]
50  ecx = dword ptr [ebp+0x10]*4
51  eax = dword ptr [ebp+8]
52  eax = dword ptr [ebp+8] + dword ptr [ebp+0x10]*4
53  eax = dword ptr [dword ptr [ebp+8] + dword ptr [ebp+0x10]*4]
54  dword ptr [dword ptr [ebp-0x0C]*4 + dword ptr [ebp+8] ]= dword ptr [dword ptr [ebp+8] + dword ptr [ebp+0x10]*4]
55  eax = dword ptr [ebp+0x10]
56  edx = dword ptr [ebp+0x10]*4
57  eax = dword ptr [ebp+8]
58  edx = dword ptr [ebp+0x10]*4 + dword ptr [ebp+8]
59  eax=dword ptr [ebp-0x18]
60  dword ptr [dword ptr [ebp+0x10]*4 + dword ptr [ebp+8]] = dword ptr [ebp-0x18]

44  dword ptr [ebp-0x18] = dword ptr [dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4]
54  dword ptr [dword ptr [ebp-0x0C]*4 + dword ptr [ebp+8] ]= dword ptr [dword ptr [ebp+8] + dword ptr [ebp+0x10]*4]
60  dword ptr [dword ptr [ebp+0x10]*4 + dword ptr [ebp+8]] = dword ptr [ebp-0x18]
```

Komplex példa 3

```
44  dword ptr [ebp-0x18] = dword ptr [dword ptr [ebp+8] + dword ptr [ebp-0x0C]*4]
54  dword ptr [dword ptr [ebp-0x0C]*4 + dword ptr [ebp+8] ]= dword ptr [dword ptr [ebp+8] + dword ptr [ebp+0x10]*4]
60  dword ptr [dword ptr [ebp+0x10]*4 + dword ptr [ebp+8]] = dword ptr [ebp-0x18]
```

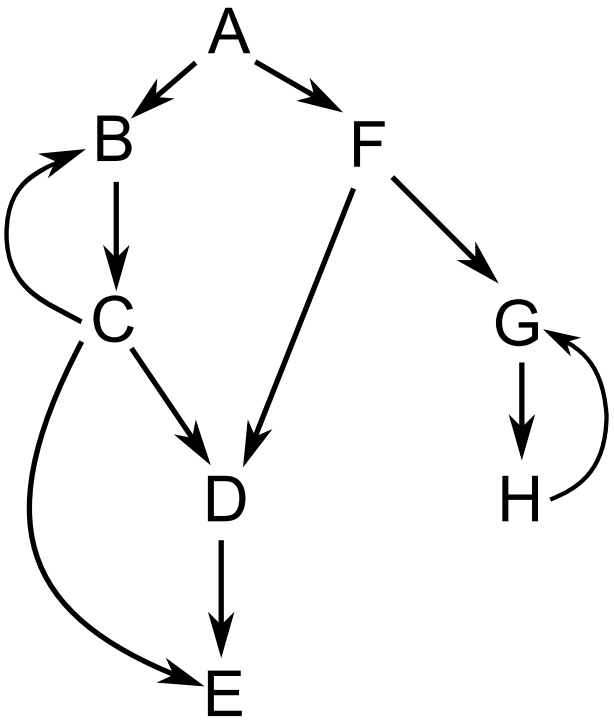
```
44  loc4 = dword ptr [loc1 + loc2*4]
54  dword ptr [loc2*4 + loc1 ]= dword ptr [loc1 + loc3*4]
60  dword ptr [loc3*4 + loc1] = loc4
```

```
44  loc4 = a[loc2*4]
54  a[loc2*4]= a[loc3*4]
60  a[loc3*4] = loc4
```

Section 1

Vezérlésfolyam analízis

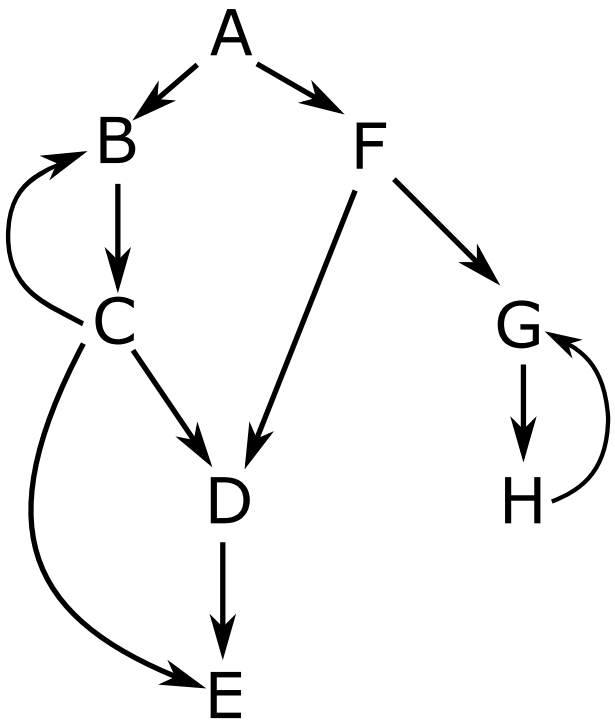
Gráf alapfogalmak

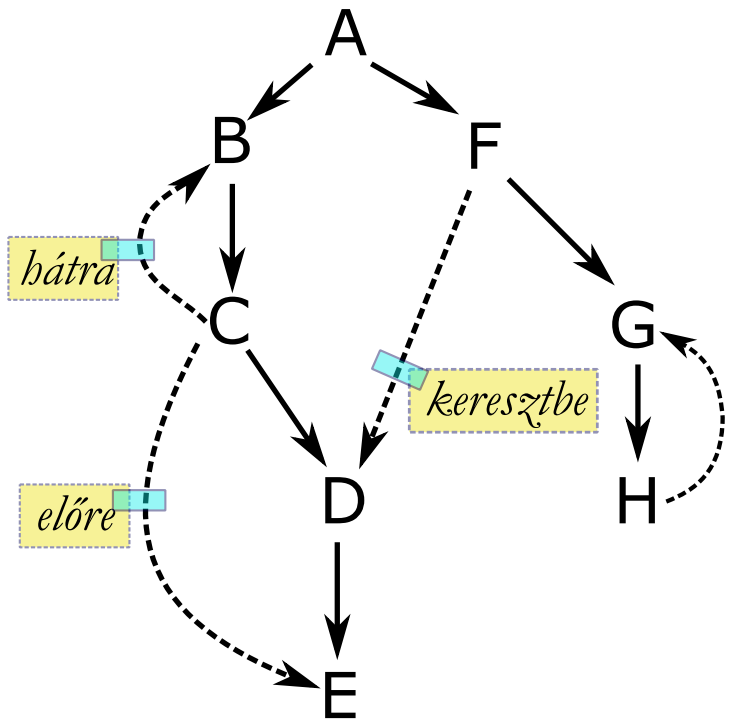


- G – irányított gráf, N – csomópontok halmaza, E – élek halmaza;
- $G = \langle N, E \rangle$;
- $N = \{A, B, C, D, E, F, G, H\}$
- $E \subseteq N \times N = \{A \to B, A \to F, B \to C, C \to B, C \to D, C \to E, D \to E, F \to D, F \to G, G \to H, G \to H\}$
- $Succ(b) = \{n \in N | \exists e \in E$ hogy $e = b \to n\}$
- $Pred(b) = \{n \in N | \exists e \in E$ hogy $e = n \to b\}$

Depth-First Search

Mélységi keresés





Jelölés:

PRE: A
POST: A

A B C D E E D C B F G H H G F A

A F D E E D G H H G F B C C B A

DFS algoritmus

```
N : in set of Node;
r, x : Node
i:=1, j:=1 : integer;
Pre, Post : Node → integer;
Visit: Node → boolean;
Etype: (Node x Node )
    → enum{ tree, forward, back, cross };

procedure Deap_First_Search( N, Succ, x )
    N : in set of Node;
    Succ : in Node → set of Node;
    x : in Node;
begin
    y : Node;
    Visit(x) := true;
    Pre(x) := j;
    j = j + 1;
    for each y ∈ Succ(x) do
        if !Visit(y) then
          Deep_First_Search(N,Succ,y);
          EType(x → y) := tree;
        elif Pre(x) < Pre(y) then
          EType(x → y) := forward;
        elif Post(y) = 0 then
          EType(x → y) := back;
        else
          EType(x → y) := cross;
        fi;
    od;
    Post(x) := i; i :+= 1;
end

begin
    for each x ∈ N do;
        Visit(x) := false;
        Pre(x) := Post(x) := 0;
    od
    Deep_First_Search(N,Succ,r);
end
```

Section 2

Kód struktúra visszafejtése

CFG struktúrájának az analízis

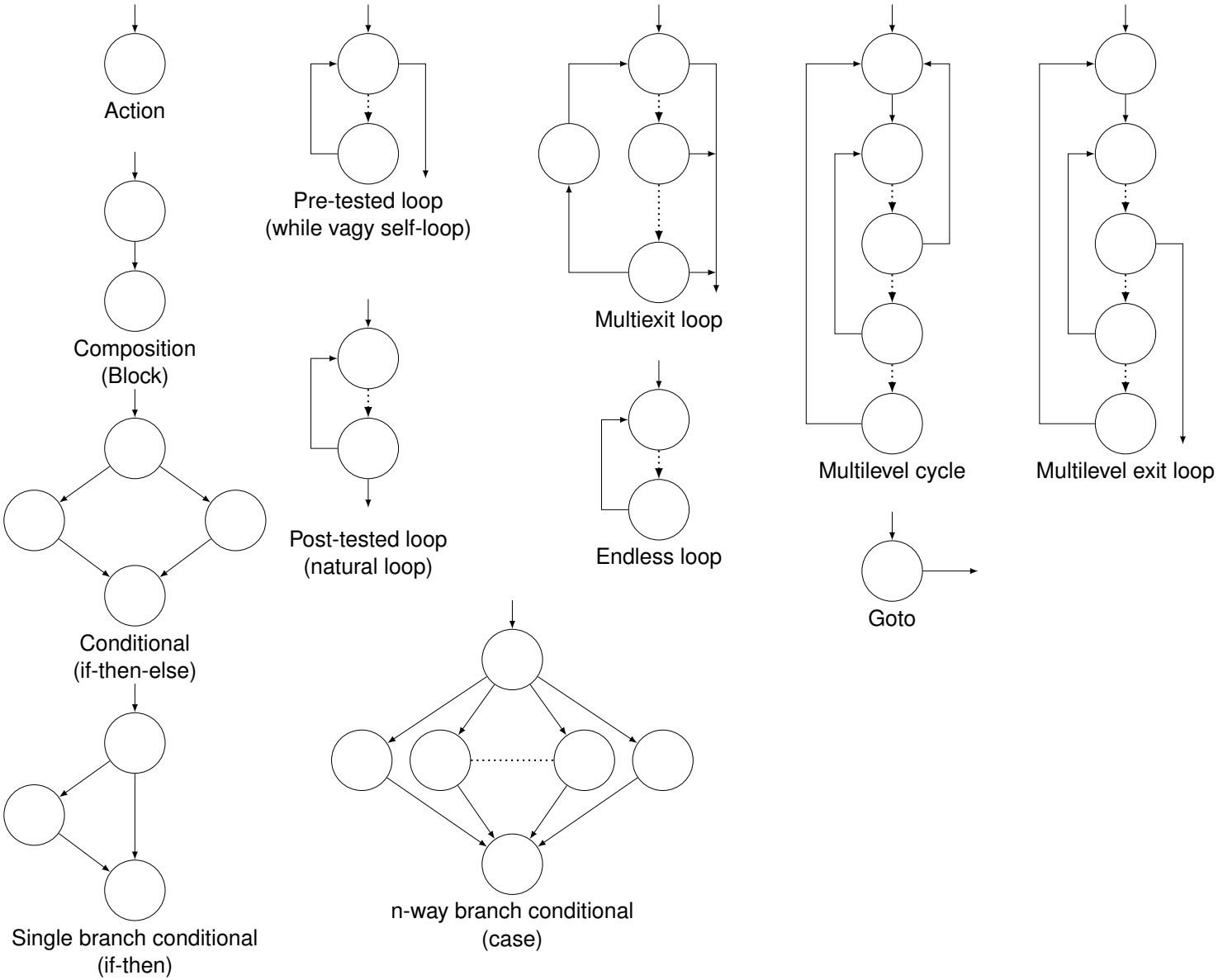
- célja felismerni a magasszintű vezérlési szerkezeteket
  - if ... then ..., if ... then ... else ..., switch ... case ...
  - for, while, repeat;
- nem struktúrált részben lévő blokkok számának a minimalizálása.

Irodalom:

Cristina Cifuentes. Reverse compilation techniques. 1994.

Steven Muchnick. Advanced Compiler Design and Implementation. 1997.

Magas-szintű vezérlési szerkezetek



Magas-szintű vezérlési szerkezetek I

1. **Action:** egyszerű blokk;
2. **Composite:** alapblokkok sorozata;
3. **Conditional:** if p then B1; else B2; feltételes elágazás;
4. **Pre-tested loop:** while p do B1; elöltesztelős ciklus;
5. **Single branch conditional:** if p then B1; feltételes elágazás else ág nélkül;
6. **n-way branch conditional:**
   switch(e) { case 1: B1; break; case 2: B2; break; ...}
   többfelé ágazás. Itt az e kifejezés értéke sorszámozott típus vagy egész szám, ami egy címeket tartalmazó vektort indexel.
7. **Post-tested loop:** hátultesztelős ciklus do ... while p;

Magas-szintű vezérlési szerkezetek II

8. **Multiexit loop:** több kilépési ponttal rendelkező ciklus; Ha a ciklusból történő kilépés során a break utasítást használjuk, akkor a futás közvetlenül a ciklus után következő utasításon folytatódik.
9. **Goto:** feltétel nélküli vezérlésátadás (goto);
10. **Multilevel cycle:** Egymásba ágyazott ciklusok, ahol a belső ciklusból a continue(i) utasítás segítségével vezérelhetjük az i-edik külső ciklusszervező utasítást, hogy a következő iterációra térjen;
11. **Multilevel exit cycle:** Egymásba ágyazott ciklusok, ahol a belső ciklusból a break(i) utasítás segítségével több ciklusból is kiléphetünk.
12. **Infinite loop:** Végtelen ciklus.

C-ben lévő magas-szintű vezérlési szerkezetek

- Action
- Composite
- Conditional
- Pre-tested loop
- Single branch conditional
- n-way branch conditional
- Post-tested loop
- Multiexit loop
- Infinite loop pl: `for (;;) { ... }`
- Goto

Strukturált programozás

Strukturált ciklusszervező utasítások

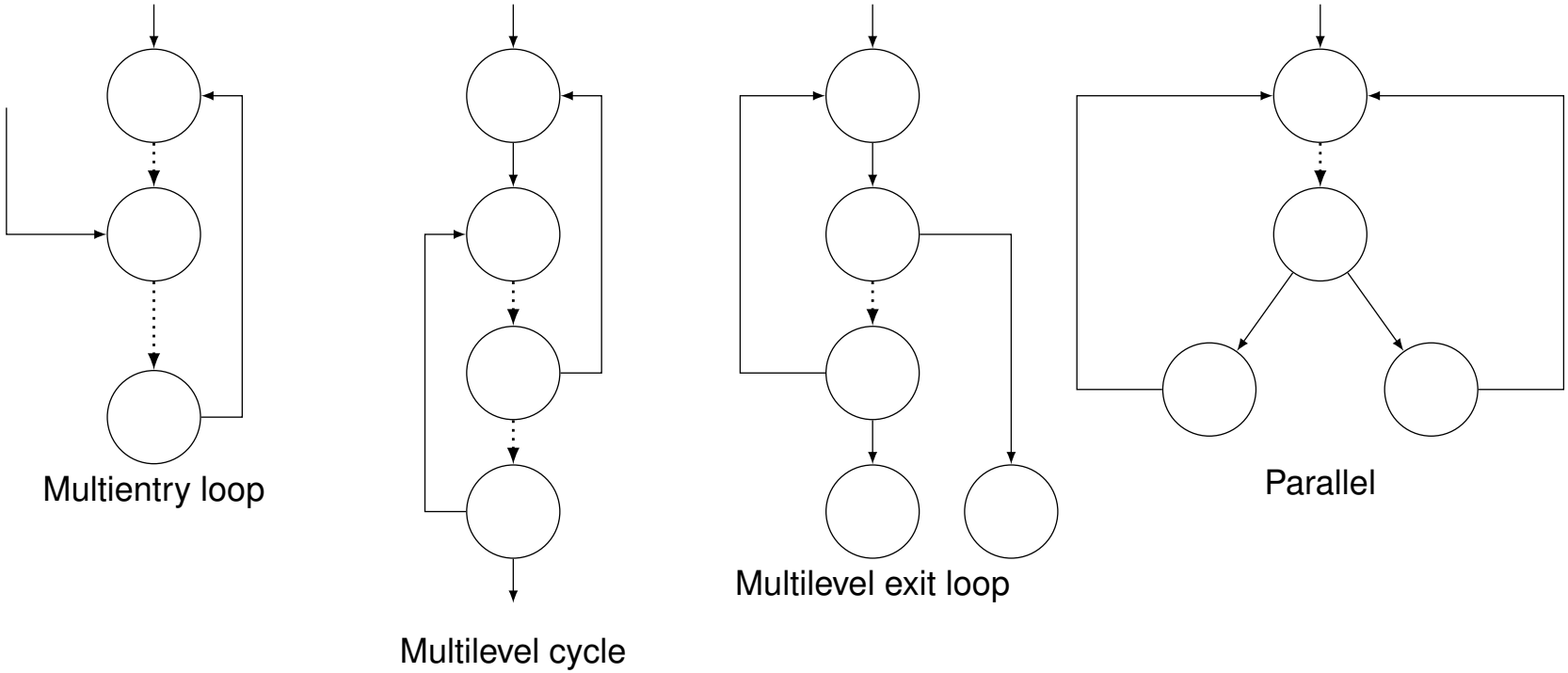
- Pre-tested loop
- Post-tested loop
- infinite loop
- Multiexit loop. A ciklus befejezése után mindig a ciklust követő első utasításra jutunk!

Strukturált feltételes vezérlésátadó utasítások

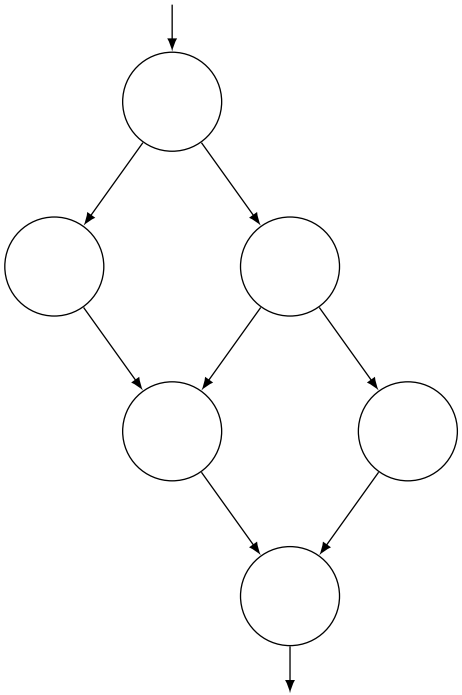
- Conditional
- Single branch conditional
- n-way branch conditional

A nem strukturált utasítás szerkezetek több belépési vagy több kilépési ponttal rendelkeznek.

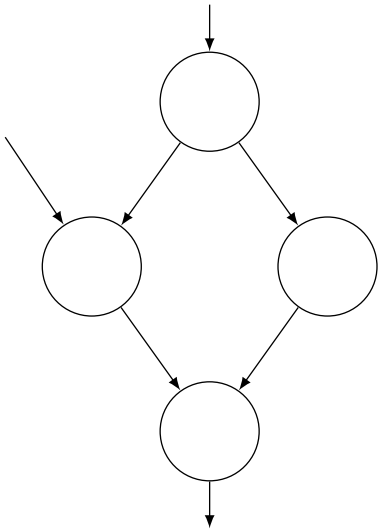
Nem strukturált ciklusszervező utasítások



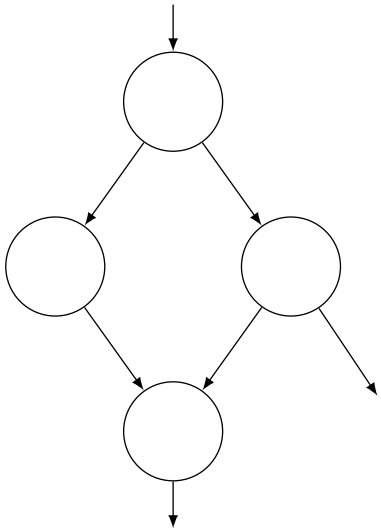
Nem strukturált vezérlésátadó utasítások



(a)



(b)



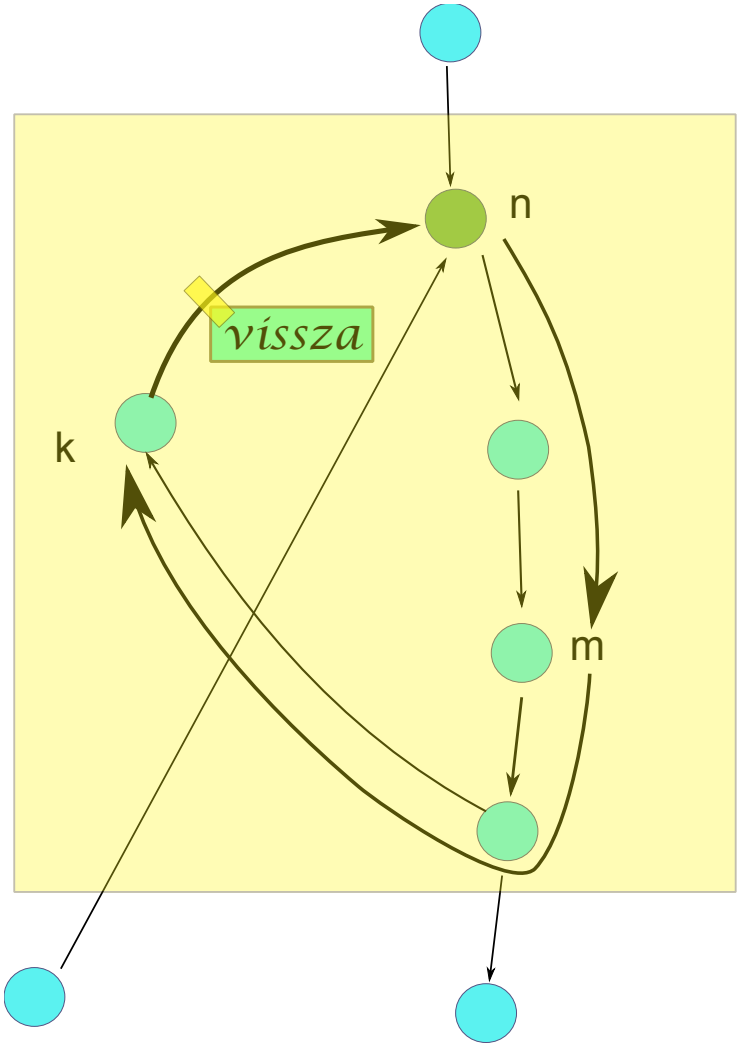
(c)

Struktúra meghatározása

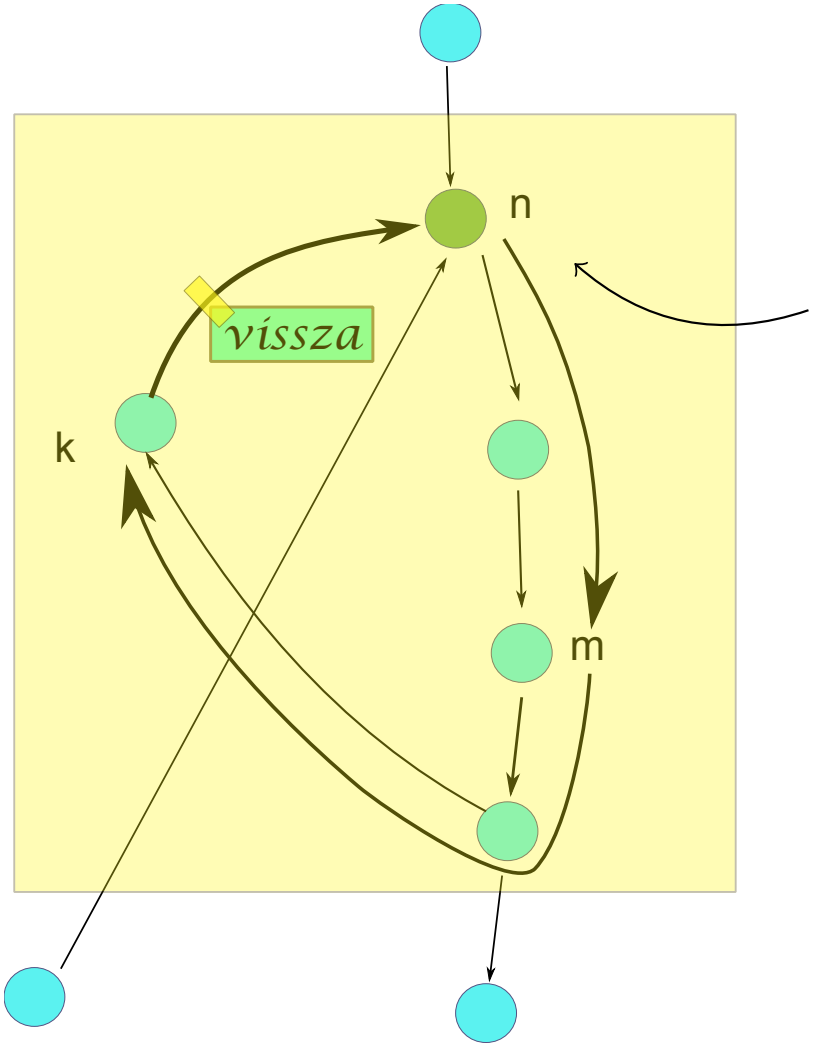
Globális adatok:

Succ, Pred : Node $\rightarrow$ set of Node;
RegionType = enum { Block, IfThen, IfThenElse, Case, Proper, SelfLoop,
WhileLoop, NaturalLoop, Improper } ;
// Blokk hozzárendelése az őt tartalmazó összevont struktúrához
StructOf: Node $\rightarrow$ Node;
// Összevont struktúra típusa (pl. if-then)
StructType: Node $\rightarrow$ RegionType;
// Összevont struktúrák halmaza
Structures: set of Node;
// Összevont struktúra alá tartozó blokkok
StructNodes : Node $\rightarrow$ set of Node;
// Összevont struktúra hierarchiája
CTNodes: set of Node;
CTEdges : set of Node x Node;
PostCtr, PostMax:integer;
Post: integer $\rightarrow$ Node;
Visit: Node $\rightarrow$ boolean;

Ciklus keresés

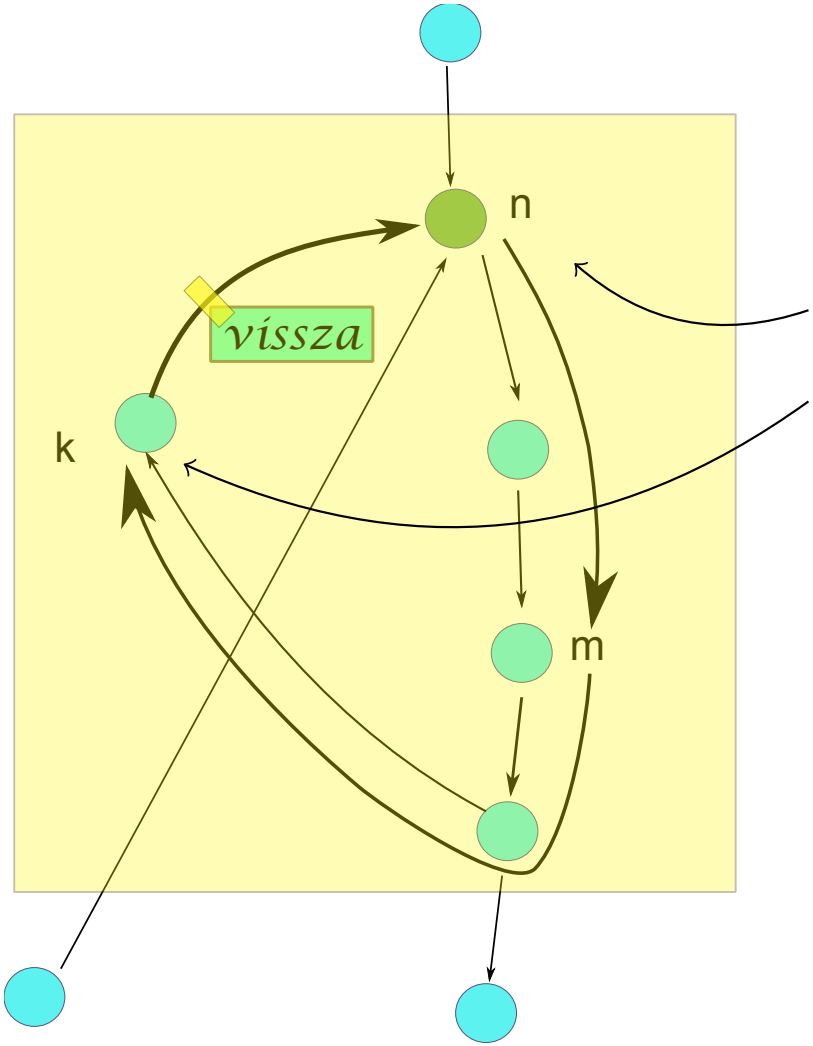


Ciklus keresés



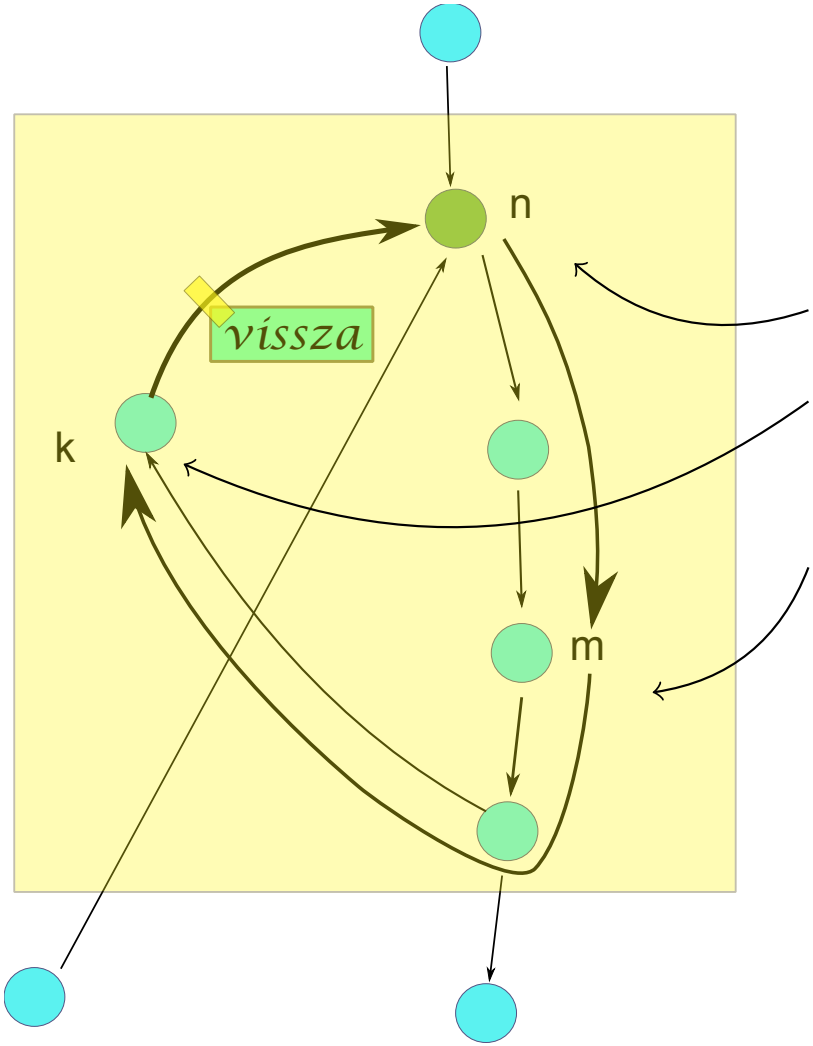
- *n* pont a ciklus kezdete;

Ciklus keresés



- n pont a ciklus kezdete;
- $\exists k \in N$ pont, ahol $Pred(n) = k$;

Ciklus keresés



- n pont a ciklus kezdete;
- $\exists k \in N$ pont, ahol $Pred(n) = k$;
- $\exists$ olyan m pont, hogy n-ből m-be és m-ből k-ba van út.

Strukturális analízis

```
Procedure StructuralAnalysis(N, E, entry)
    N : in set of Node;
    E: in set of (Node x Node);
    entry: in Node;
begin
    m, n, p : Node;
    rtype : RegionType;
    NodeSet, ReachUnder: set of Node;
    StuctOf := StructType := ∅;
    Stuctures := StructNodes := ∅;
    CTNodes := N; CTEdges := ∅;
    repeat
        Post := ∅; Visit := ∅
        PostMax := 0; PostCtr := 1
        DFS_Postorder(N, E, entry );
        while |N| > 1∧ PostCtr ≤ PostMax do
            n := Post(PostCtr) ;
            rtype :=AcyclicRType(N,E,rtype,NodeSet);
            if rtype ≠ nil then
              p:= Reduce(N,E,rtype,NodeSet) ;
              if entry ∈ NodeSet then entry := p; fi;
            else
                ReachUnder := {n};
                for each n ∈ N do
                    // ∃k amely vissza élen eljut n-be és,
                    // ∃m pont, amihez van n → m,
                    // és m → k-ba él.
                    if PathBack(m,n) then
                      ReachUnder ∪ = {m};
                    fi;
                od;
                rtype :=CyclicRType(N,E,rtype,
                    ReachUnder);
                if rtype ≠ nil then
                  p:= Reduce(N,E,rtype,ReachUnder) ;
                  if entry ∈ ReachUnder then
                    entry := p;
                  fi
                else
                  PostCtr += 1;
                fi
            fi
            od;
        until |N| = 1;
    end
```

Blokkok DFS bejárása

```
Procedure DFS_Postorder(N, E, x)
    N : in set of Node;
    E: in set of (Node x Node);
    x: in Node;
begin
    y : Node;
    Visit(x):=true;
    for each y ∈ Succ(x) do
        if !Visit(y) then
          DFS_Postorder(N,E,y);
        if;
    od;
    PostMax+= 1;
    Post(PostMax) := x;
end
```

Aciklikus struktúrák feltérképezése

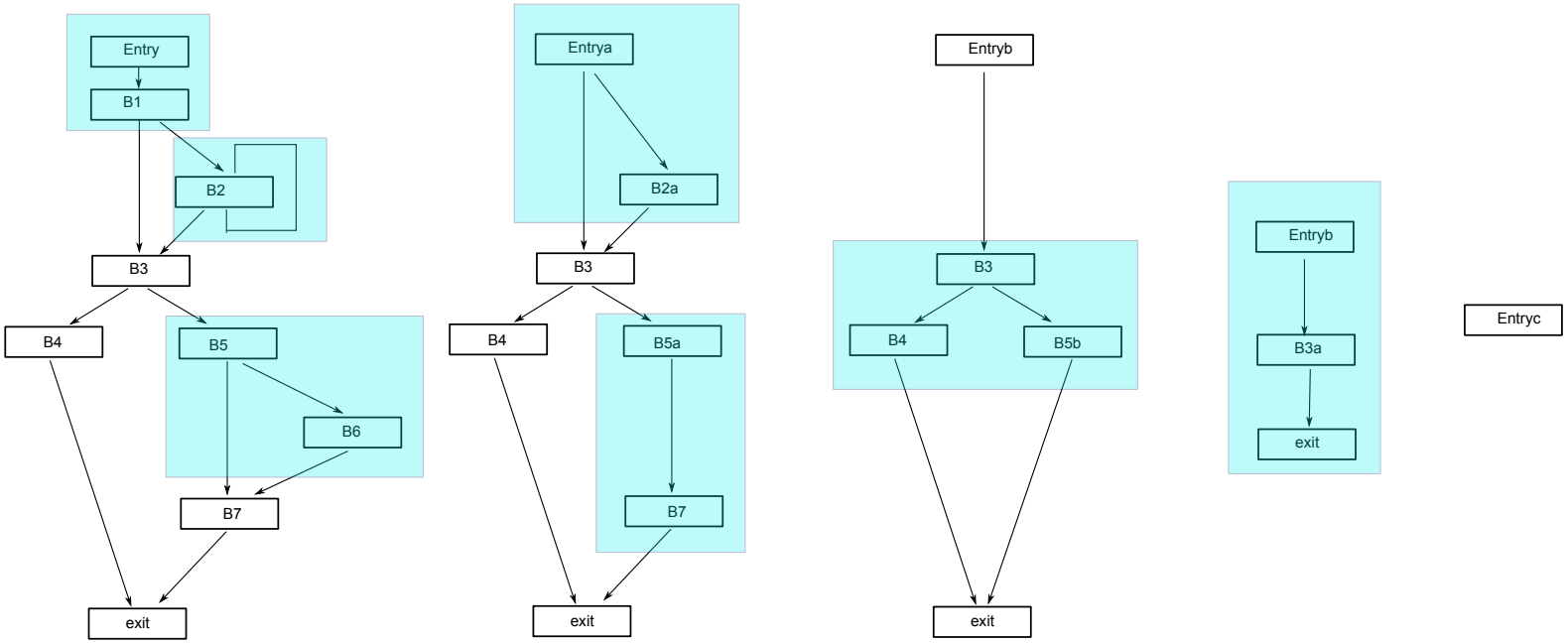
```
Procedure AcyclicRType(N,E,node,nset) : RegionType
    N : in set of Node;
    E: in set of (Node x Node);
    node: inout Node;
    nset: out set of Node;
begin
    m, n : Node;
    p, s: boolean;
    nset := ∅;
    n := node; p := true; s := |Succ(n)| = 1;
    while p ∧ s do
        nset ∪ = {n}; n := ♦Succ(n);
        p := |Pred(n)| = 1; s := |Succ(n)| = 1;
    od;
    if p then
        nset ∪ = {n};
    fi
    n := node; p := |Pred(n)| = 1; s:=true;
    while p ∧ s do
        nset ∪ = {n}; n := ♦Pred(n);
        p := |Pred(n)| = 1; s := |Succ(n)| = 1;
    od
    if s then
      nset ∪ = {n};
    fi
    node := n;
    if |nset| ≥ 2 then
      return Block;
    elif |Succ(node)| = 2 then
      m := Succ(node); n:= (Succ(node)-{m});
      if Succ(m) = Succ(n)
        ∧ |Succ(m)| = 1∧ |Succ(n)| = 1
        ∧ |Pred(m)| = 1∧ |Pred(n) = 1| then
        nset := {node, m, n};
        return IfThenElse;
      elif ...
        ...
      else
        return nil;
      fi;
    fi;
end

/* ♦ halmaz egyik eleme */
```

Ciklikus struktúrák feltérképezése

```
Procedure CyclicRType(N,E,node,nset) : RegionType
    N : in set of Node;
    E: in set of (Node x Node);
    node: in Node;
    nset: inout set of Node;
begin
    m : Node;
    if |nset| = 1 then
      if node → node ∈ E then
        return SelfLoop;
      else
        return nil;
      fi;
    fi;
    if ∃m ∈ nset !Path(node,m,N) then
      nset := MinimizeImproper(N,E,node,nset);
      return Improper;
    fi
    m := ♦(nset - { node });
    if |Succ(node)| = 2∧ |Succ(m)| = 1∧
      |Pred(node)| = 2 ∧ |Pred(m) = 1| then
      return WhileLoop;
    else
      return NaturalLoop;
    fi;
end
```

Redukció


Entry
B1
B2
B3
B4
B5
B6
B7
exit
Entrya
B2a
B3
B4
B5a
B7
exit
Entryb
B3
B4
B5b
exit
Entryb
B3a
exit
Entryc

Összetett feltételes elágazások