Relatório LI2

Grupo 32 (PL3)
Fábio Silva (A75662)
Francisco Lira (A73909)
Pedro Oliveira (A75521)

31 de Maio de 2015

Conteúdo

1 Comando R

A função do comando resolver consiste num ciclo com 3 condições de paragem:

1. O tabuleiro está resolvido. Esta condição é verificada recorrendo à função "resolvido", que percorre o tabuleiro à procura de posições indeterminadas, isto é, posições cujo caractere é 'o' ou '.'.

2. O tabuleiro não é válido. Esta condição é verificada recorrendo à estrutura "data", que funciona como um valor de retorno de certas funcões e foi criada como resposta à necessidade dessas funções retornarem não só se efetuaram ou não alterações ao tabuleiro, bem como se "descobriram" que o tabuleiro é inválido. Na estrutura, o campo "val" corresponde à (in)validade do tabuleiro, tomando o valor 1 se este for inválido, e o campo "alt" toma o valor 1 se a função efetuou alterações ao tabuleiro.

3. A última iteração do ciclo não conseguiu efetuar qualquer alteração no tabuleiro. Esta condição é verificada recorrendo à estrutura "data", referida na alínea anterior, consultando o valor do campo "alt". Para garantir que o ciclo seja sempre executado pelo menos uma vez, esse campo toma o valor 1 no início da função.

Dentro desse ciclo principal existem:

- Um ciclo, com as condições de paragem explicadas nas alíneas 2. e 3. anteriores, que se limita a aplicar as estratégias de resolução (1, 2, 3 e A1), até que as mesmas não efetuem quaisquer alterações no tabuleiro, ou se "descubra" que o tabuleiro não é válido.

- Uma condição que verifica se o valor do campo "val" da estrutura "data" é igual a 0, ou seja, se as estratégias aplicadas no ciclo anterior não alteraram o tabuleiro para um tabuleiro inválido. Caso o tabuleiro seja válido, chama-se a função "descobrir". Caso contrário, é imprimida uma mensagem no ecrã a dizer que o tabuleiro é inválido.

Funções criadas para auxiliar o comando resolver:

- Função "descobrir", que percorre todo o tabuleiro, temporariamente substituindo cada posição indeterminada por um segmento (indeterminados) de barco, ou seja, substituindo o caractere '.' pelo caractere 'o'. Para cada posição substituida, é executado o comando V. Se o tabuleiro não for válido, podemos garantir que nessa posição não estará um segmento de um barco, portanto colocamos água. Caso contrário, nada podemos concluir, portanto a posição volta ao estado anterior ('.'), usando o undo dentro da função do comando V.

- Função "EA1", uma estratégia de resolução que consiste em procurar segmentos de barco e preencher com um 'o' as posições indeterminadas adjacentes aos segmentos que se pode garantir que também têm um segmento de barco. Existe também uma função auxiliar, chamada "EA1centro"que é executada no caso em que o segmento é o centro de um barco (''). Sempre que existe um centro de um barco, o número de segmentos da linha ou da coluna, dependendo da orientação do barco, será sempre pelo menos 3 (o

centro do barco e, pelo menos, as duas posições adjacentes). Essa função começa por verificar essa condição. Se o número de segmentos da linha ou da coluna onde está o barco for menor ou igual a 2, ficaremos a saber a orientação do barco e preenchemos as posições adjacentes com água de acordo com a orientação do barco: se o barco estiver orientado horizontalmente, as posições imediatamente acima e abaixo do barco serão água; se o barco estiver orientado verticalmente, as posições imediatamente à esquerda e à direita do barco serão água. De seguida, a função também se encarga de, nos casos em que sabemos a orientação do barco (verificando a existência de água nas posições acima, abaixo, à esquerda ou à direita do barco), preencher com 'o' as posições adjacentes ao centro do barco que sabemos que terão um segmento de barco, de acordo com a sua orientação.

Funções alteradas para auxiliar o comando resolver:

- As funções dos comandos h, v e p passaram a receber também um apontador para a estrutura "data", de forma a que possam indicar, através do campo "alt" se efetuaram alterações ao tabuleiro, para que se saiba se as estratégias que as usam efetuaram alterações.

- A função pComand também indica, através do campo "val" da estrutura "data", que o tabuleiro é inválido, nos casos em que se tenta colocar um segmento de barco numa posição que tem água, ou água numa posição que tem um segmento de barco.

2 Análise do Código Gerado

```
0x08048400 contar_segs+0:     push   %ebp
0x08048401 contar_segs+1:     mov    %esp, %ebp
0x08048403 contar_segs+3:     push   %edi
0x08048404 contar_segs+4:     push   %esi
0x08048405 contar_segs+5:     push   %ebx
0x08048406 contar_segs+6:     sub    $0xc, %esp
0x08048409 contar_segs+9:     xor    %edi, %edi
0x0804840b contar_segs+11:    xor    %ecx, %ecx
0x0804840d contar_segs+13:    xor    %edx, %edx
0x0804840f contar_segs+15:    cmpb   $0x0, 0x2720(%ebp)
0x08048416 contar_segs+22:    mov    0x2724(%ebp), %eax
0x0804841c contar_segs+28:    movl   $0x0, -0x10(%ebp)
0x08048423 contar_segs+35:    movl   $0x0, -0x14(%ebp)
0x0804842a contar_segs+42:    je     0x8048486 "contar_segs+134"
0x0804842c contar_segs+44:    lea    -0x1(%eax), %ecx
0x0804842f contar_segs+47:    movl   $0x1, -0x10(%ebp)
0x08048436 contar_segs+54:    mov    0x2718(%ebp), %eax
0x0804843c contar_segs+60:    test   %eax, %eax
0x0804843e contar_segs+62:    jle    0x804847b "contar_segs+123"
0x08048440 contar_segs+64:    mov    %eax, %esi
0x08048442 contar_segs+66:    lea    0x0(, %edx, 4), %eax
0x08048449 contar_segs+73:    add    %edx, %eax
0x0804844b contar_segs+75:    lea    (%ecx, %ecx, 4), %ebx
0x0804844e contar_segs+78:    mov    %eax, -0x18(%ebp)
0x08048451 contar_segs+81:    lea    0x0(%esi), %esi
0x08048454 contar_segs+84:    lea    (%ebx, %ebx, 4), %eax
0x08048457 contar_segs+87:    lea    0x8(%ebp, %eax, 4), %eax
0x0804845b contar_segs+91:    sub    $0xc, %esp
0x0804845e contar_segs+94:    movsbl (%edi, %eax, 1), %eax
0x08048462 contar_segs+98:    push   %eax
0x08048463 contar_segs+99:    call   0x80483e4 "e_seg"
0x08048468 contar_segs+104:   add    $0x10, %esp
0x0804846b contar_segs+107:   test   %al, %al
0x0804846d contar_segs+109:   je     0x8048472 "contar_segs+114"
0x0804846f contar_segs+111:   incl   -0x14(%ebp)
0x08048472 contar_segs+114:   add    -0x10(%ebp), %edi
0x08048475 contar_segs+117:   add    -0x18(%ebp), %ebx
0x08048478 contar_segs+120:   dec    %esi
0x08048479 contar_segs+121:   jne    0x8048454 "contar_segs+84"
0x0804847b contar_segs+123:   mov    -0x14(%ebp), %eax
0x0804847e contar_segs+126:   lea    -0xc(%ebp), %esp
0x08048481 contar_segs+129:   pop    %ebx
0x08048482 contar_segs+130:   pop    %esi
0x08048483 contar_segs+131:   pop    %edi
0x08048484 contar_segs+132:   leave
0x08048485 contar_segs+133:   ret
0x08048486 contar_segs+134:   lea    -0x1(%eax), %edi
0x08048489 contar_segs+137:   mov    $0x1, %edx
0x0804848e contar_segs+142:   mov    0x271c(%ebp), %eax
0x08048494 contar_segs+148:   jmp    0x804843c "contar_segs+60"
```

2.1 Tabela de alocação de registos

i/tam	%esi (no assembly, em vez de ser criada uma nova variável, para ser incrementada, é usada a variável tam, sendo decrementada até ser 0)
x	%eax/%edi (começa por ser atribuida ao registo %eax, no entanto é movida para %esi)
y	%ecx
dx	−0x10(%ebp)
dy	%edx/−0x18(%ebp) (começa por ser atribuida ao registo %edx, no entanto é movida para −0x18%(ebp) - o quintúplo de dy)
count	−0x14(%ebp)

2.2 Variável tab

tab é uma matriz 100x100 de char's.

1 char – 1 byte

100 * 100 = 10000 bytes = 0x2710 bytes

Localiza-se em [0x8(%ebp);0x2717(%ebp)]

0x8(%ebp)	t.tab[0][0]
0x9(%ebp)	t.tab[0][1]
0xa(%ebp)	t.tab[0][2]
0xb(%ebp)	t.tab[0][3]
0x2714(%ebp)	t.tab[99][96]
0x2715(%ebp)	t.tab[99][97]
0x2716(%ebp)	t.tab[99][98]
0x2717(%ebp)	t.tab[99][99]

2.3 Indexação da matriz

Preparação:

contar_segs+66 : lea	0x0(, %edx, 4), %eax	%eax = 4 * dy (%edx = dy)
contar_segs+73 : add	%edx, %eax	%eax = 5 * dy (%eax = 4 * dy; %edx = dy)
contar_segs+75 : lea	(%ecx, %ecx, 4), %ebx	%ebx = 5 * y (%ecx = y)
contar_segs+78 : mov	%eax, −0x18(%ebp)	−0x18(%epb) = 5 * dy (%eax = 5 * dy)

Acesso (ciclo):

contar_segs+84 : lea	(%ebx, %ebx, 4), %eax	%eax = 25 * y (%ebx = 5 * y)
contar_segs+87 : lea	0x8(%ebp, %eax, 4), %eax	%eax = 0x8(%ebp) + 100 * y (%eax = 25 * y)
contar_segs+94 : movsbl	(%edi, %eax, 1), %eax	%eax = 0x8(%ebp) + 100 * y + x (%eax = 0x8(%ebp) + 100 * y; %edi = x)

Logo, a indexação é feita do seguinte modo: 0x8(%ebp) + 100 * y + x

2.4 Código correspondente

contar_segs+9 : xor	%edi,%edi	x = 0
contar_segs+11 : xor	%ecx,%ecx	y = 0
contar_segs+13 : xor	%edx,%edx	dy = 0
contar_segs+28 : movl	$0x0,−0x10(%ebp)	dx = 0
contar_segs+35 : movl	$0x0,−0x14(%ebp)	count = 0
contar_segs+15 : cmpb	$0x0,0x2720(%ebp)	
contar_segs+42 : je	0x8048486 “contar_segs+134”	if (lin) – neste caso, se lin for 0, salta, caso contrário, continua
contar_segs+44 : lea	−0x1(%eax),%ecx	y = num − 1
contar_segs+47 : movl	$0x1,−0x10(%ebp)	dx = 1
contar_segs+54 : mov	0x2718(%ebp),%eax	tam = t.lins
contar_segs+134 : lea	−0x1(%eax),%edi	x = num − 1
contar_segs+137 : mov	$0x1,%edx	dy = 1
contar_segs+142 : mov	0x271c(%ebp),%eax	tam = t.cols
contar_segs+60 : test	%eax,%eax	
contar_segs+62 : jle	0x804847b “contar_segs+123”	if (tam = 0) – se tam for menor ou igual a 0 o ciclo não é iniciado
contar_segs+99 : call	0x80483e4 e_seg	
contar_segs+107 : test	%al,%al	
contar_segs+109 : je	0x8048472 “contar_segs+114”	if ($e_seg(t.tab[y][x]) - -se\%al$ (valor retornado pela função) for 0, salta
contar_segs+111 : incl	−0x14(%ebp)	count++
contar_segs+114 : add	−0x10(%ebp),%edi	x += dx
contar_segs+117 : add	−0x18(%ebp),%ebx	y += dy – na verdade, o que acontece é 5 * y += 5 * dy
contar_segs+120 : dec	%esi	tam−− – em vez de ser incrementado o i até ser igual a tam, é decrementado o tam até ser igual a 0
contar_segs+121 : jne	0x8048454 “contar_segs+84”	se tam for diferente de 0, o ciclo é repetido
contar_segs+123 : mov	−0x14(%ebp),%eax	return count