Alguns dos exercícios das Fichas Teórico-práticas resolvidos:

Ficha1

1.

a)

Estrutura estado:

X // Quantidade de água no balde x
Y // Quantidade de água no balde y

b) **Estado inicial: 0, 0**
porque inicialmente os baldes estão vazios
Estado final: 2, Y
porque se pretende obter 2 litros no balde x e a
quantidade em y não interessa, porque não é especificada

c) As acções possíveis são 6, correspondentes a outros tantos operadores:

1) Encher o balde x a partir do tanque
2) Encher o balde y a partir do tanque

3) Esvaziar o balde x no tanque
4) Esvaziar o balde y no tanque

5) Verter o balde x no balde y
6) Verter o balde y no balde x

d) As acções 1 e 2 são sempre possíveis. Contudo, se os baldes já estiverem cheios, não há qualquer progresso do estado actual para o seguinte: Eles são iguais porque a quantidade de água transferida do tanque para os vasos é zero. **Em suma, a aplicação dos operadores 1 e 2 só deve ser considerada se X<4 ou Y<3, respectivamente.**

Com as **acções 3 e 4** passa-se algo de semelhante: Se os baldes estiverem vazios não há qualquer progresso do estado actual para o seguinte: Eles são iguais porque a quantidade de água transferida dos baldes para o tanque é zero. Em suma, **a aplicação dos operadores 3 e 4 só deve ser considerada se X>0 ou Y>0, respectivamente.**
As acções 5 e 6 são sempre possíveis mas se o balde origem estiver vazio ou o balde destino estiver cheio, não há progressão do estado actual para o seguinte. **Portanto, estes operadores só devem ser considerados se o balde origem não estiver vazio e se o balde destino não estiver cheio.**
Além disso, a quantidade transferida só pode ser, no máximo, igual à diferença entre a capacidade máxima do balde destino e o seu conteúdo actual.

e) Vamos considerar que os operadores são aplicados de acordo com a ordem apresentada em c), i.e. o operador 1 primeiro, se possível, o operador 2 em seguida...até ao operador 6.

Inicial: PorExpandir = {(0,0)}
Expandidos = {}

1 NóActual_AExpandir=(0,0):

PorExpandir = {(4,0), (0,3)}
Expandidos = {(0,0)}

2 NóActualA_Expandir=(4,0):
PorExpandir = {(4,3), (0,0), (1,3), (0,3)}
Expandidos = {(4,0), (0,0)}

3 NóActualA_Expandir=(4,3):
PorExpandir = {(0,3), (4,0), (0,0), (1,3), (0,3)}
Expandidos = {(4,3), (4,0), (0,0)}

4 NóActualA_Expandir=(0,3):
PorExpandir = {(4,3), (0,0), (3,0), (4,0), (0,0), (1,3), (0,3)}
Expandidos = {(0,3), (4,3), (4,0), (0,0)}

5 (4,3) e (0,0) já foram expandidos. Portanto, o próximo é:
NóActualA_Expandir=(3,0)
PorExpandir = {(4,0), (3,3), (0,0), (0,3), (4,0), (0,0), (1,3), (0,3)}
Expandidos = {(3,0), (0,3), (4,3), (4,0), (0,0)}

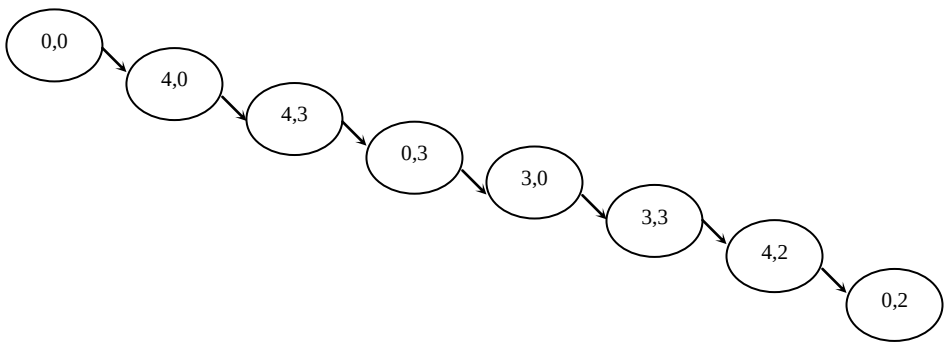
6 (4,0) já foi expandido. Portanto o próximo é:
NóActualA_Expandir=(3,3):
PorExpandir={(4,3), (0,3), (3,0), (4,2), (0,0), (0,3), (4,0), (0,0), (1,3), (0,3)}
Expandidos = {(4,3), (3,3), (3,0), (0,3), (4,3), (4,0), (0,0)}

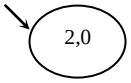
7 (4,3), (0,3) e (3,0) já foram expandidos. Portanto o próximo é:
NóActualA_Expandir=(4,2):
PorExpandir={(4,3), (0,2), (4,0), (3,3), (0,0), (0,3), (4,0), (0,0), (1,3), (0,3)}
Expandidos = {(4,2), (3,3), (3,0), (0,3), (4,3), (4,0), (0,0)}

8 (4,3) já foi expandido. Portanto o próximo é:
NóActualA_Expandir=(0,2):
PorExpandir={(4,2), (0,3), (0,0), (2,0), (4,0), (3,3), (0,0), (0,3), (4,0), (0,0), (1,3), (0,3)}
Expandidos = {(0,2), (4,2), (3,3), (3,0), (0,3), (4,3), (4,0), (0,0)}

9 (4,2), (0,3) e (0,0) já foram expandidos. Portanto o próximo é:
NóActualA_Expandir=(2,0):
Objectivo atingido.

NOTA: A sequência de operações também pode ser visualizada através de uma árvore, mas isto não era pedido no enunciado do problema. Como se trata da pesquisa em profundidade, a árvore reduz-se a um único ramo que atinge o objectivo:





f) Encher os dois baldes a partir do tanque
Despejar o balde de 4 litros no tanque
Verter o balde de 3 litros no de 4 litros
Voltar a encher o balde de 3 litros a partir do tanque
Verter 1 litro do balde de 3 litros no de 4 litros (fica X=4, Y=2)
Despejar o balde de 4 litros no tanque
Verter os 2 litros do balde Y no balde X (o de 4 litros)

Neste caso a solução encontrada não é óptima uma vez que se enchem inicialmente os dois baldes para em seguida se despejar o de 4 litros: É evidente que bastaria encher inicialmente apenas o de 3 litros. Contudo, se os operadores forem aplicados por outra ordem, é possível que a solução encontrada seja óptima. de facto, como se trata de uma pesquisa em profundidade, nada garante que a solução encontrada seja a óptima, mas pode-o ser: Tudo depende da estrutura do problema e da ordem de aplicação dos operadores.

2. a) Uma estrutura possível consiste apenas em 4 inteiros, um para cada entidade do problema – Agricultor, Cão, Coelho, Legume – que assumem o valor 0 quando a entidade está na margem A e o valor 1 quando a entidade está na margem B:

Estrutura Estado

```
Inteiro A          // valor =0 ou 1
Inteiro C          // valor =0 ou 1
Inteiro E          // valor =0 ou 1
Inteiro L          // valor =0 ou 1
```

b) Estado Inicial: (0, 0, 0, 0)
Estado Final: (1, 1, 1, 1)

c) Há 4 operadores possíveis:

1 – Atravessa apenas o agricultor (A)
2 – Atravessa o agricultor e o cão (AC)
3 – Atravessa o agricultor e o coelho (AE)
4 – Atravessa o agricultor e o legume (AL)

De acordo com a estrutura descrita em a) a aplicação de um operador a um dado estado inicial conduz à "complementação" do valor 0 ou 1 respeitante às entidades que atravessam o rio por aplicação desse operador. Por exemplo, a aplicação do operador (AE) ao estado ACEL=0000 conduz ao estado 1010.

d) (A) pode ser aplicado se:
Origem contiver A e não restar na origem (C+E ou E+L)
⇔ A e C≠E e E≠L

(AC) pode ser aplicado se:
Origem contiver A+C e não restar na origem E+L
⇔ A=C e E≠L

(AE) pode ser aplicado se:
Origem contiver A+E
⇔ A=E

(AL) pode ser aplicado se:

Origem contiver A+L e não restar na origem C+E

⇔ A=C e E≠L

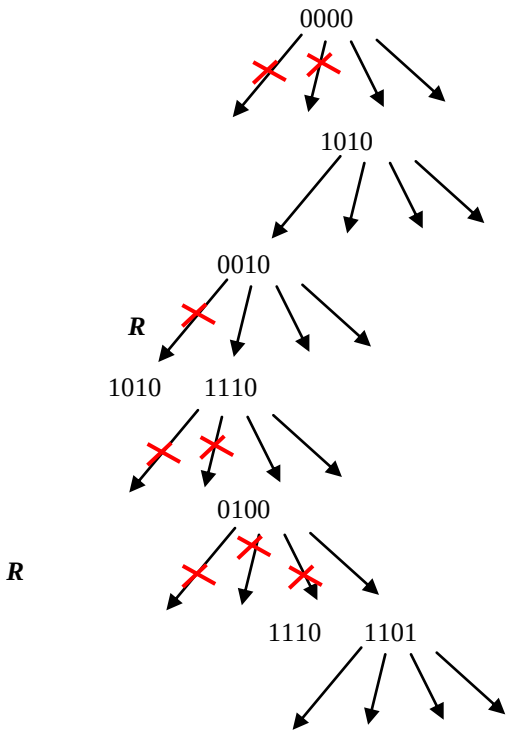
De acordo com a estrutura indicada em a) os estados aos quais cada operador pode ser aplicado são os seguintes, respectivamente:

(A)	(AC)	(AE)		(AL)
ACEL	ACEL	ACEL		ACEL
----	----	----		----
0010	0010	0000	Estado Inicial	0010
0101	0001	0001		0100
1010	1110	0100		1011
1101	1101	0101		1101
		1010		
		1011		
	Objectivo	1110		
		1111		

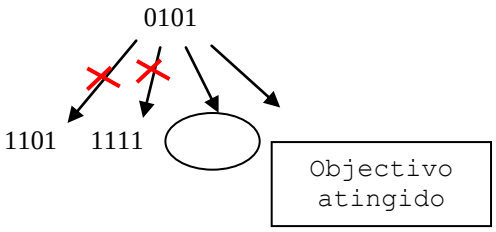
e) Parte-se do estado 0000 (todos) correspondente a todos na margem A . O objectivo é 1111 correspondente a todos na margem B. Como se trata da pesquisa em profundidade, cada novo estado é expandido imediatamente tomando em linha de conta o seguinte:

- Na expansão de cada estado podem apenas aplicar-se os operadores que verifiquem as condições da alínea anterior
- Se a expansão de um estado conduzir a outro já anteriormente expandido, esse novo estado não será considerado. Isto porque, de acordo com o enunciado, se mantém uma lista de "nós já expandidos"

Consideraremos que a ordem de aplicação dos operadores é 1, 2, 3, 4, ou seja, A, AC, AE, AL. Na figura, estes operadores estão ordenados, em cada estado expandido, "da direita para a esquerda



R

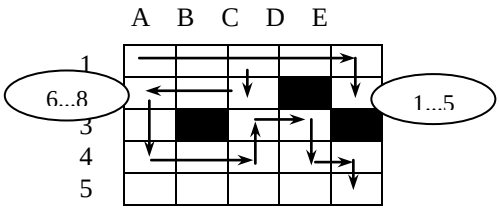


f)

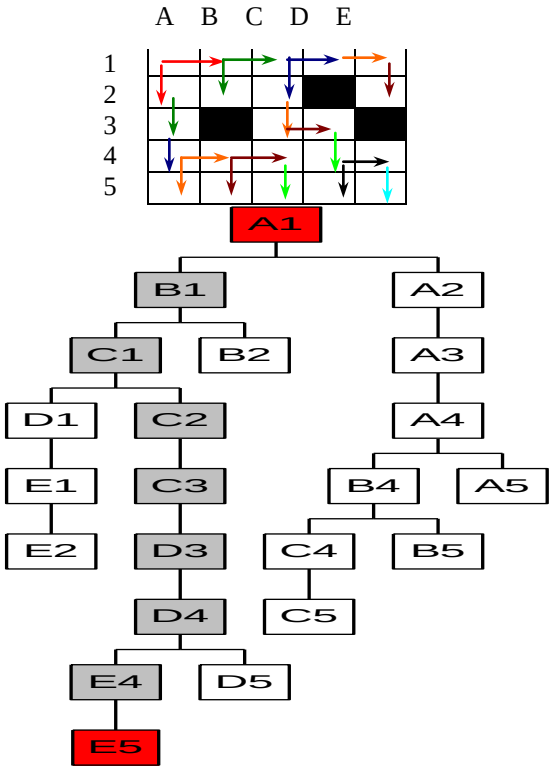
1) Leva coelho de A para B
2) Regressa a A
3) Leva cão de A para B
4) Traz o coelho de B para A
5) Leva legume de A para B
6) Regressa a A
7) Leva coelho de A para B

Ficha2

2. a)


A B C D E
1
2
3
4
5
6...8
1...5

b)


A B C D E
1
2
3
4
5
A1
B1
A2
C1
B2
A3
D1
C2
A4
E1
C3
B4
A5
E2
D3
C4
B5
D4
C5
E4
D5
E5

Ficha3

1.

a) Como a função custo representa o tempo do percurso, pode tomar-se como versão relaxada do problema (i.e. problema sujeito a menos restrições) uma em que o grupo de elevadores A pára em qualquer andar. Como estes elevadores se deslocam à velocidade *v*, suponhamos que gastam *t* segundos para percorrer a distância entre 2 pisos. A função heurística $h=t.q$, em que *q* representa o número de pisos a percorrer até ao alvo, é admissível porque nunca sobre-estima o custo do percurso, uma vez que representa o tempo gasto da origem ao destino, directamente, usando apenas elevadores da classe A.

b) Estrutura estado

```
Piso            // Um número pertencente a [0 a 100]
Tempo_Gasto     // Custo: O acumulado do tempo gasto desde o
                // piso origem (estado inicial) até Piso
Tempo_Estimado  // O tempo estimado até o objectivo dado pela
                // função heurística t.q = t.(Alvo.Piso - Piso)
```

c) Há 2 operadores a considerar: "subir" e "descer". Valerá a pena considerar o operador "descer" ? De facto, como o seguinte exemplo demonstra, o operador "descer" deve ser considerado:

Seja t o tempo de viagem entre dois pisos consecutivos para elevadores da classe A. Como os da classe B viajam à velocidade *v/2*, então *2t* será o tempo de viagem entre dois pisos consecutivos para elevadores da classe B, e *4t* para os da classe C. Assim, se se pretender subir até ao 19º andar, o percurso

Subir em elevador A até ao 20º demorará 20t
Descer em elevador C do 20º até ao 19º demorará 1*4t=4t

Portanto, o tempo total é de 24t.

Pelo contrário, o percurso:

Subir em elevador B até ao 10º demorará 10*2t=20t
Subir em elevador C do 10º até ao 19º demorará 9*4t=36t

Portanto, usando apenas o operador "subir", este percurso demora 56t.

Portanto, um sucessor de um estado deve poder ser gerado por operadores "subir" ou "descer", uma vez que há soluções melhores que são dadas por uma subida até ao piso N seguida de uma descida até ao estado alvo.

Assim, se o estado actual representar um piso múltiplo de 20, os estados seguintes poderão ser obtidos por subida ou descida para pisos:

- Separados do actual por 20, viajando em classe A
- Separados do actual por 10, viajando em classe B
- Separados do actual por 1, viajando em classe C

Se o estado actual representar um piso múltiplo de 10, os estados seguintes poderão ser obtidos por subida ou descida para pisos:

- Separados do actual por 10, viajando em classe B

· Separados do actual por 1, viajando em classe C

Se o estado actual representar um piso qualquer, não múltiplo de 20 nem de 10, os estados seguintes poderão ser obtidos por subida ou descida para pisos:

· Separados do actual por 1, viajando em classe C

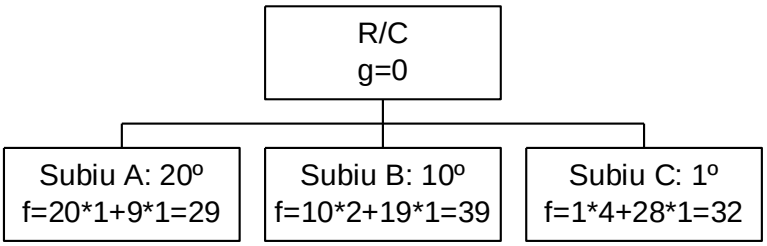
Função Aplica_Operadores (estado, estado_alvo)

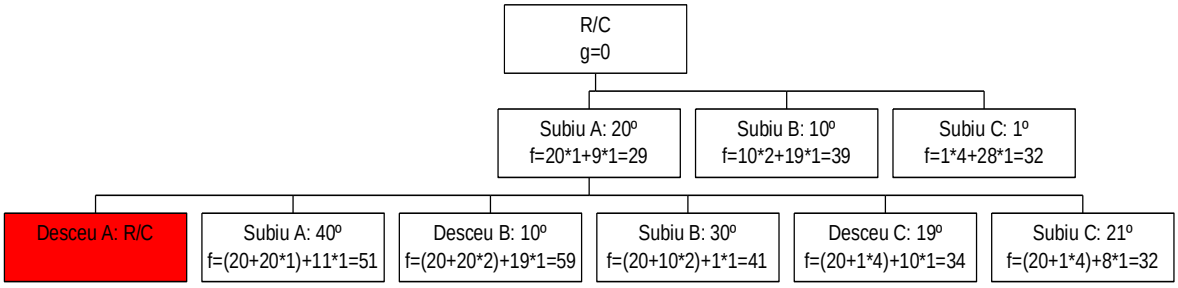
```
        Se estado.Piso % 20=0
                {prox_pisos}=proximos(estado, 20, estado_alvo)
                {prox_pisos}={prox_pisos}∪ proximos(estado, 10, estado_alvo)
                {prox_pisos}={prox_pisos}∪ proximos(estado, 1, estado_alvo)
        Senão
                Se estado_actual%10=0
                        {prox_pisos}=proximos(estado, 10, estado_alvo)
                        {prox_pisos}={prox_pisos}∪ proximos(estado, 1, estado_alvo)
                Senão
                        {prox_pisos}=proximos(estado, 1, estado_alvo)
                FimSe
        FimSe
Return {prox_pisos}
```

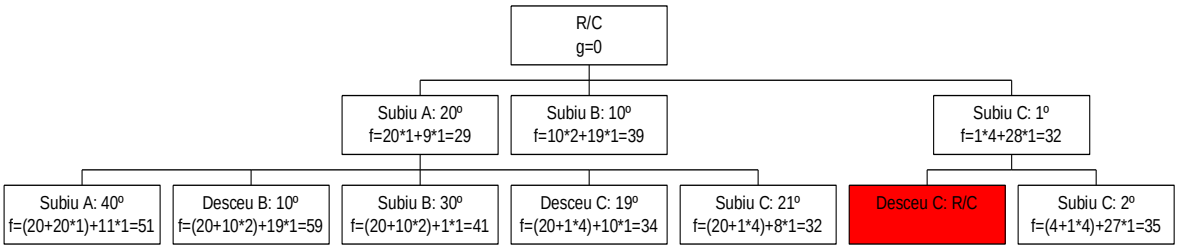
Função proximos (estado, Num_Pisos, estado_alvo)

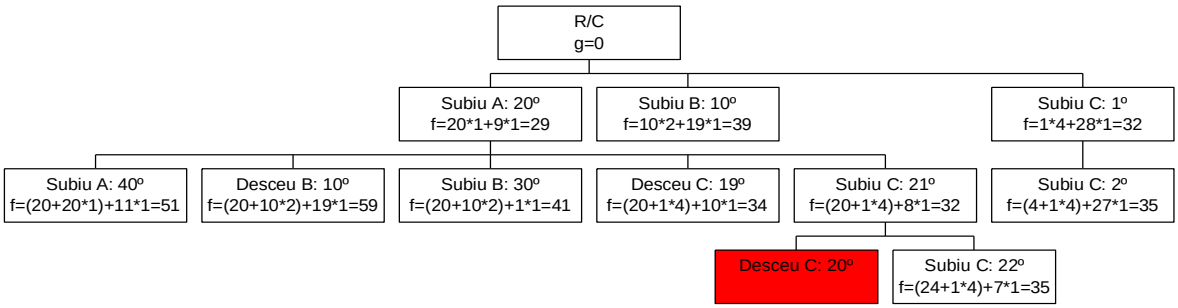
```
        aux ← estado
// Subir
        Num_Pisos
                20: Se estado.Piso < 81
                        aux.Piso ← estado.Piso + Num_Pisos
                        aux.Tempo_Gasto  ← aux.Tempo_Gasto+Num_Pisos*1
                10: Se estado.Piso < 91
                        aux.Piso ← estado.Piso + Num_Pisos
                        aux.Tempo_Gasto ← aux.Tempo_Gasto+Num_Pisos*2
                1: Se estado.Piso < 100
                        aux.Piso ← estado.Piso + Num_Pisos
                        aux.Tempo_Gasto ← aux.Tempo_Gasto+Num_Pisos*4
        aux.Tempo_Estimado ← (estado_alvo.Piso - aux.Piso)
        {estados_seguintes} ← aux
//Descer
        aux.Piso ← estado.Piso - Num_Pisos
        Caso Num_Pisos
                20: Se estado.Piso >19
                        aux.Piso ← estado.Piso - Num_Pisos
                        aux.Tempo_Gasto ← aux.Tempo_Gasto+Num_Pisos*1
                10: Se estado.Piso > 9
                        aux.Piso ← estado.Piso - Num_Pisos
                        aux.Tempo_Gasto ← aux.Tempo_Gasto+Num_Pisos*2
                1: Se estado.Piso ≥ 0
                        aux.Piso ← estado.Piso - Num_Pisos
                        aux.Tempo_Gasto ← aux.Tempo_Gasto+Num_Pisos*4
        aux.Tempo_Estimado ← (estado_alvo.Piso - aux.Piso)
        {estados_seguintes} ← {Novos_estados} ∪ aux
Return estados_seguintes
```

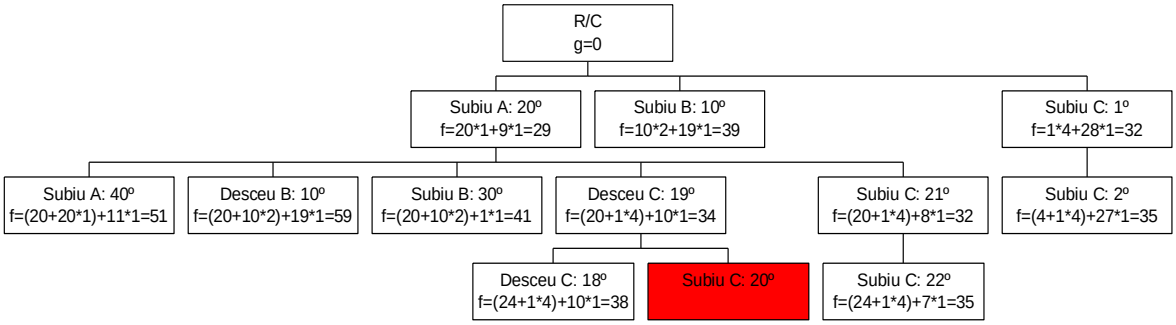
f =g+h =tempo__real_decorrido + tempo_ao_objectivo_em_classe_A

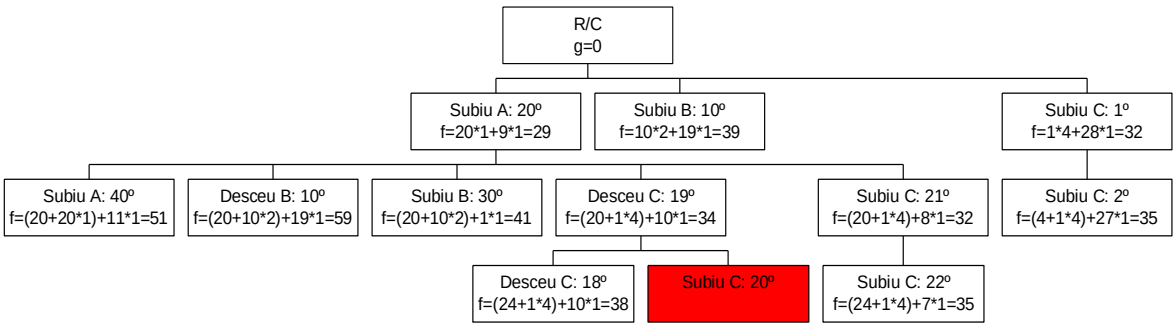


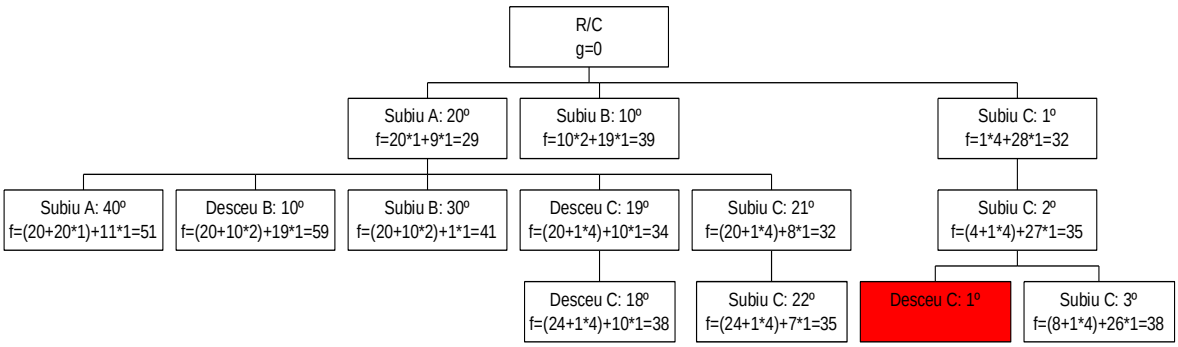


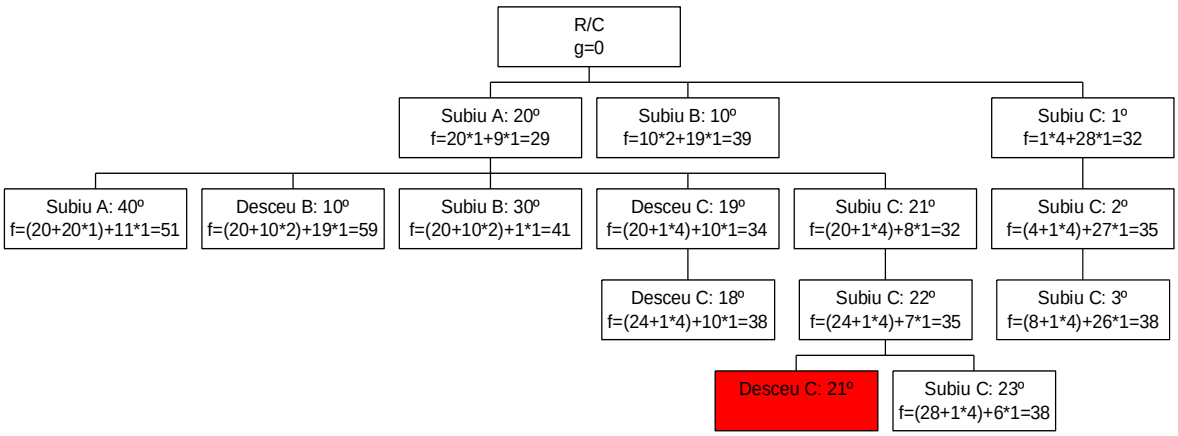


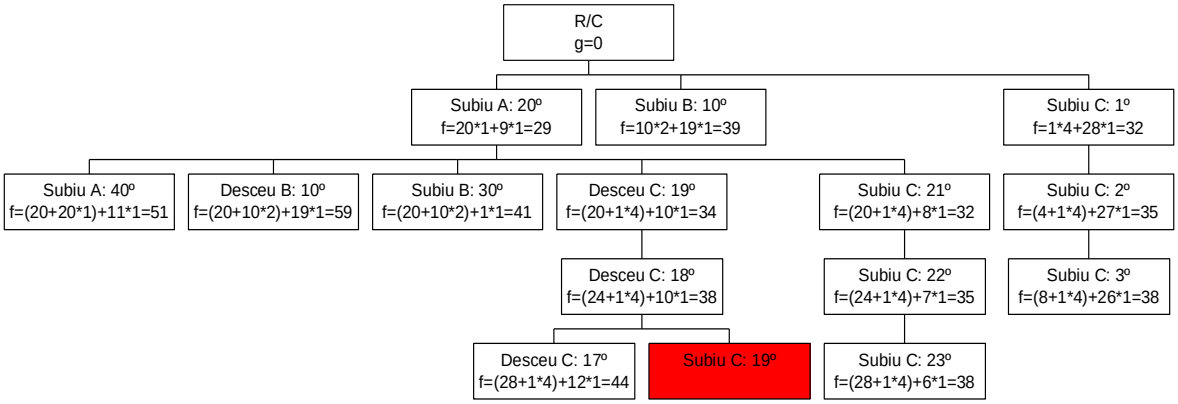


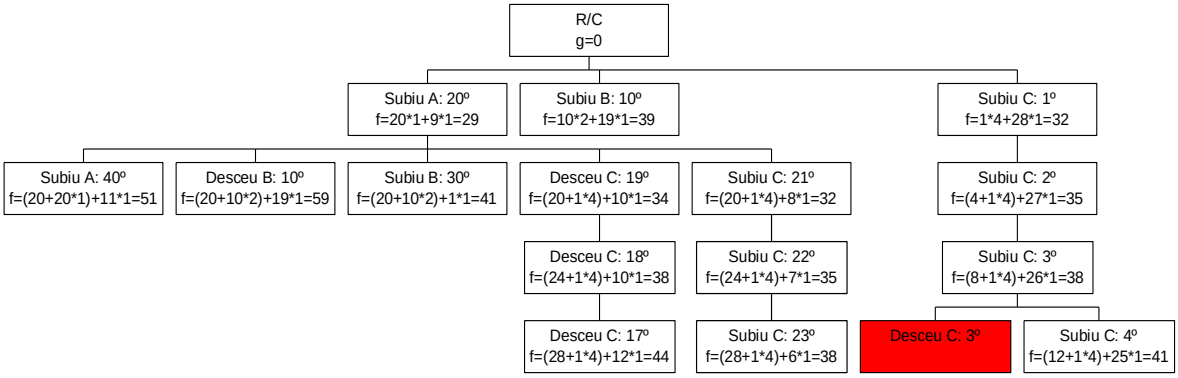

R/C
g=0
Subiu A: 20º
f=20*1+9*1=29
Subiu B: 10º
f=10*2+19*1=39
Subiu C: 1º
f=1*4+28*1=32
Subiu A: 40º
f=(20+20*1)+11*1=51
Desceu B: 10º
f=(20+10*2)+19*1=59
Subiu B: 30º
f=(20+10*2)+1*1=41
Desceu C: 19º
f=(20+1*4)+10*1=34
Subiu C: 21º
f=(20+1*4)+8*1=32
Subiu C: 2º
f=(4+1*4)+27*1=35
Desceu C: 18º
f=(24+1*4)+10*1=38
Subiu C: 20º
Subiu C: 22º
f=(24+1*4)+7*1=35

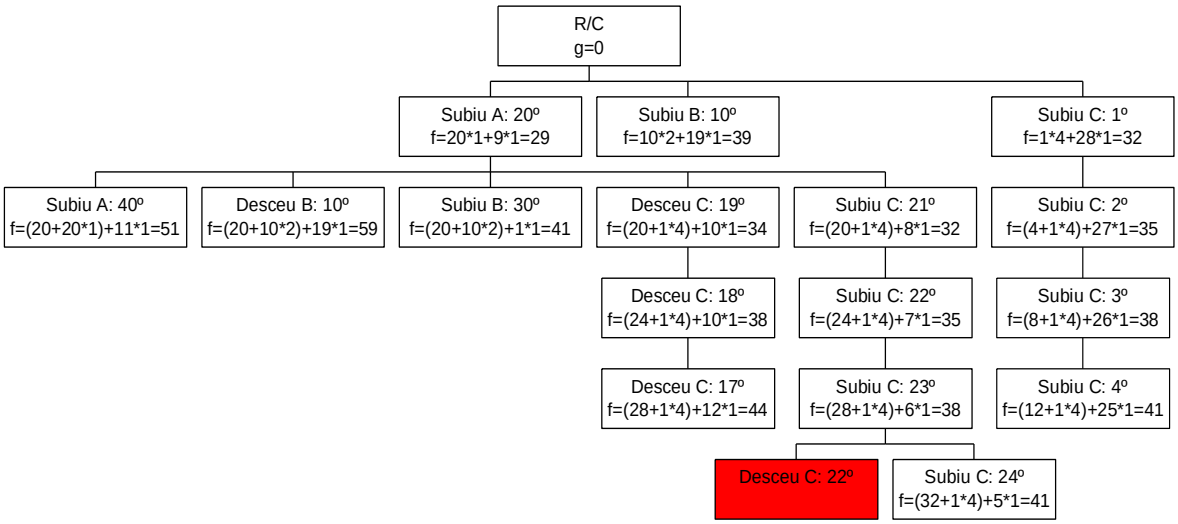

R/C
g=0
Subiu A: 20º
f=20*1+9*1=29
Subiu B: 10º
f=10*2+19*1=39
Subiu C: 1º
f=1*4+28*1=32
Subiu A: 40º
f=(20+20*1)+11*1=51
Desceu B: 10º
f=(20+10*2)+19*1=59
Subiu B: 30º
f=(20+10*2)+1*1=41
Desceu C: 19º
f=(20+1*4)+10*1=34
Subiu C: 21º
f=(20+1*4)+8*1=32
Subiu C: 2º
f=(4+1*4)+27*1=35
Desceu C: 18º
f=(24+1*4)+10*1=38
Subiu C: 20º
Subiu C: 22º
f=(24+1*4)+7*1=35

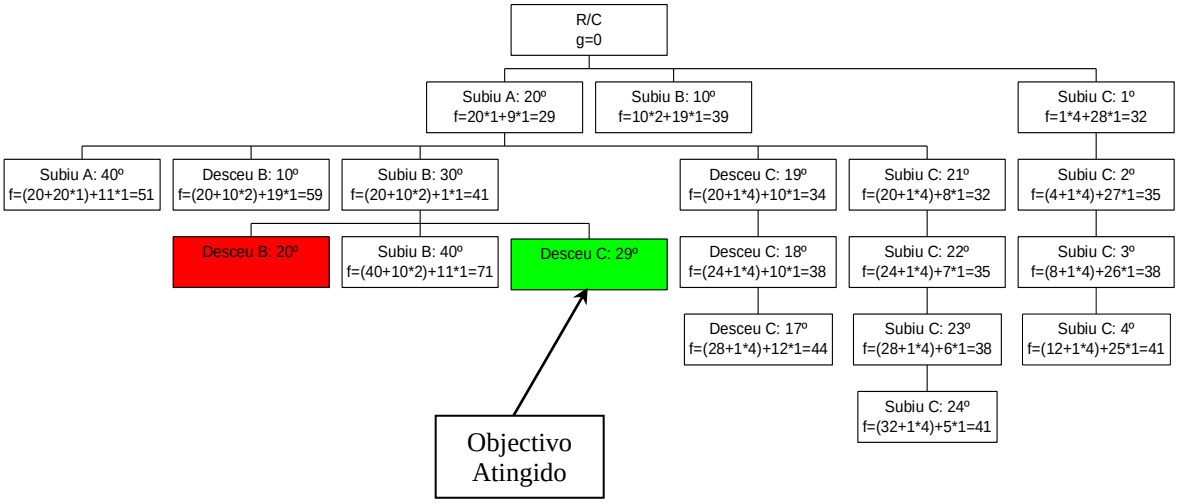

R/C
g=0
Subiu A: 20º
f=20*1+9*1=29
Subiu B: 10º
f=10*2+19*1=39
Subiu C: 1º
f=1*4+28*1=32
Subiu A: 40º
f=(20+20*1)+11*1=51
Desceu B: 10º
f=(20+10*2)+19*1=59
Subiu B: 30º
f=(20+10*2)+1*1=41
Desceu C: 19º
f=(20+1*4)+10*1=34
Subiu C: 21º
f=(20+1*4)+8*1=32
Subiu C: 2º
f=(4+1*4)+27*1=35
Desceu C: 18º
f=(24+1*4)+10*1=38
Subiu C: 22º
f=(24+1*4)+7*1=35
Desceu C: 1º
Subiu C: 3º
f=(8+1*4)+26*1=38


R/C
g=0
Subiu A: 20º
f=20*1+9*1=29
Subiu B: 10º
f=10*2+19*1=39
Subiu C: 1º
f=1*4+28*1=32
Subiu A: 40º
f=(20+20*1)+11*1=51
Desceu B: 10º
f=(20+10*2)+19*1=59
Subiu B: 30º
f=(20+10*2)+1*1=41
Desceu C: 19º
f=(20+1*4)+10*1=34
Subiu C: 21º
f=(20+1*4)+8*1=32
Subiu C: 2º
f=(4+1*4)+27*1=35
Desceu C: 18º
f=(24+1*4)+10*1=38
Subiu C: 22º
f=(24+1*4)+7*1=35
Subiu C: 3º
f=(8+1*4)+26*1=38
Desceu C: 21º
Subiu C: 23º
f=(28+1*4)+6*1=38


R/C
g=0
Subiu A: 20º
f=20*1+9*1=29
Subiu B: 10º
f=10*2+19*1=39
Subiu C: 1º
f=1*4+28*1=32
Subiu A: 40º
f=(20+20*1)+11*1=51
Desceu B: 10º
f=(20+10*2)+19*1=59
Subiu B: 30º
f=(20+10*2)+1*1=41
Desceu C: 19º
f=(20+1*4)+10*1=34
Subiu C: 21º
f=(20+1*4)+8*1=32
Subiu C: 2º
f=(4+1*4)+27*1=35
Desceu C: 18º
f=(24+1*4)+10*1=38
Subiu C: 22º
f=(24+1*4)+7*1=35
Subiu C: 3º
f=(8+1*4)+26*1=38
Desceu C: 17º
f=(28+1*4)+12*1=44
Subiu C: 19º
Subiu C: 23º
f=(28+1*4)+6*1=38


R/C
g=0
Subiu A: 20º
f=20*1+9*1=29
Subiu B: 10º
f=10*2+19*1=39
Subiu C: 1º
f=1*4+28*1=32
Subiu A: 40º
f=(20+20*1)+11*1=51
Desceu B: 10º
f=(20+10*2)+19*1=59
Subiu B: 30º
f=(20+10*2)+1*1=41
Desceu C: 19º
f=(20+1*4)+10*1=34
Subiu C: 21º
f=(20+1*4)+8*1=32
Subiu C: 2º
f=(4+1*4)+27*1=35
Desceu C: 18º
f=(24+1*4)+10*1=38
Subiu C: 22º
f=(24+1*4)+7*1=35
Subiu C: 3º
f=(8+1*4)+26*1=38
Desceu C: 17º
f=(28+1*4)+12*1=44
Subiu C: 23º
f=(28+1*4)+6*1=38
Desceu C: 3º
Subiu C: 4º
f=(12+1*4)+25*1=41


R/C
g=0
Subiu A: 20º
f=20*1+9*1=29
Subiu B: 10º
f=10*2+19*1=39
Subiu C: 1º
f=1*4+28*1=32
Subiu A: 40º
f=(20+20*1)+11*1=51
Desceu B: 10º
f=(20+10*2)+19*1=59
Subiu B: 30º
f=(20+10*2)+1*1=41
Desceu C: 19º
f=(20+1*4)+10*1=34
Subiu C: 21º
f=(20+1*4)+8*1=32
Subiu C: 2º
f=(4+1*4)+27*1=35
Desceu C: 18º
f=(24+1*4)+10*1=38
Subiu C: 22º
f=(24+1*4)+7*1=35
Subiu C: 3º
f=(8+1*4)+26*1=38
Desceu C: 17º
f=(28+1*4)+12*1=44
Subiu C: 23º
f=(28+1*4)+6*1=38
Subiu C: 4º
f=(12+1*4)+25*1=41
Desceu C: 22º
Subiu C: 24º
f=(32+1*4)+5*1=41

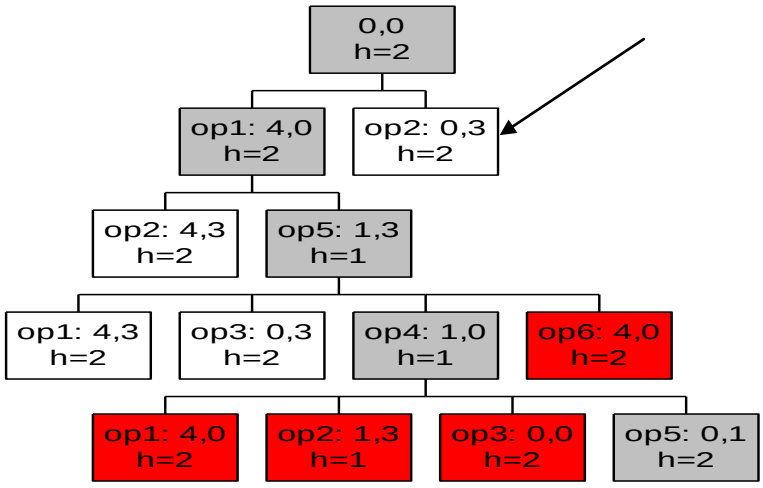


Solução: Subir em classe A até ao 20º
Subir em classe B até ao 30º
Descer em classe C até ao 29º

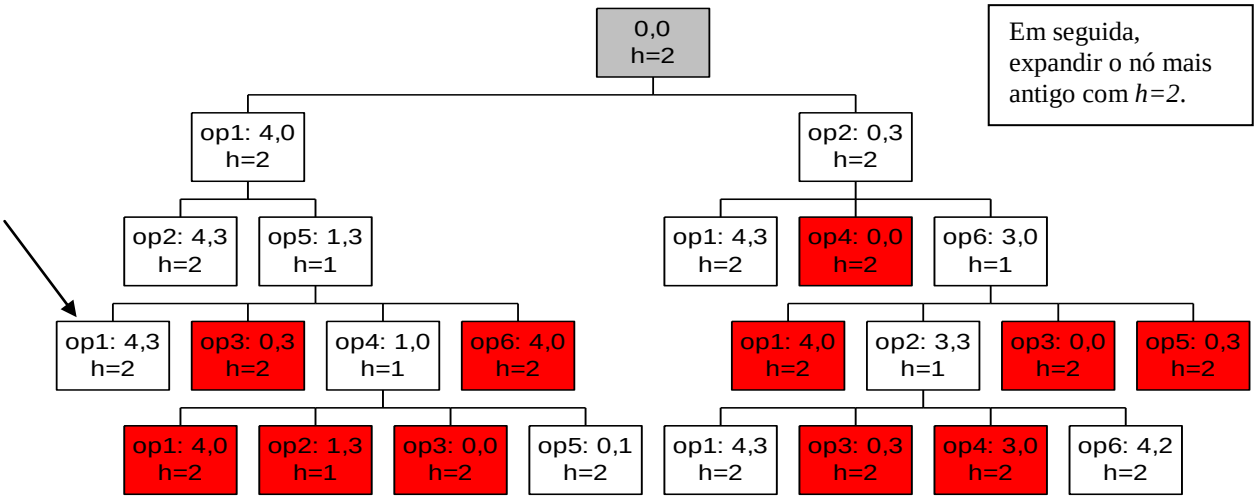
2.

O estado final é (2,Y), isto é, pretende-se 2l em X. Em Y pode ter-se uma quantidade qualquer. Ou seja, o objectivo é X=2 (e apenas isto). Além disso considere-se que o custo do caminho é representado pelo número de litros de água que se movimentam ao todo entre tanque e baldes e entre baldes até se atingir o objectivo.

Nestas condições uma heurística admissível - uma função do tipo distância ao objectivo "em linha recta" - pode ser a diferença entre o que X contém em cada estado e o objectivo X=2litros, visto que esta diferença traduz o caminho mínimo que falta percorrer até ao objectivo, e portanto nunca sobrestima este custo. Então, para a pesquisa sôfrega tem-se:

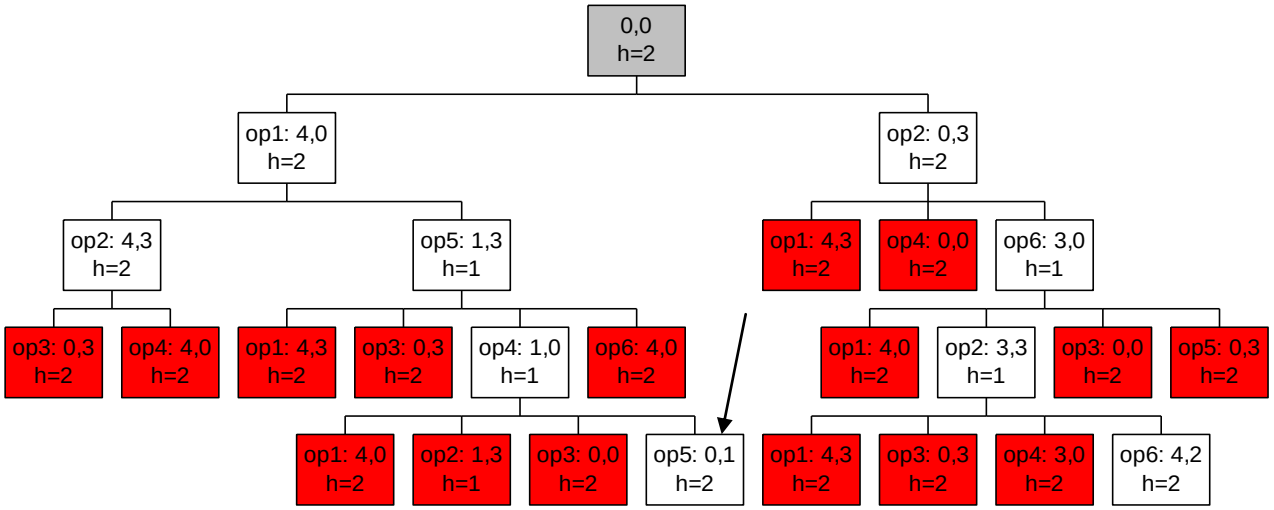


Como há vários nós com valor heurístico h=2, vamos expandir o que foi gerado em primeiro lugar, ou seja, o estado (0,3).

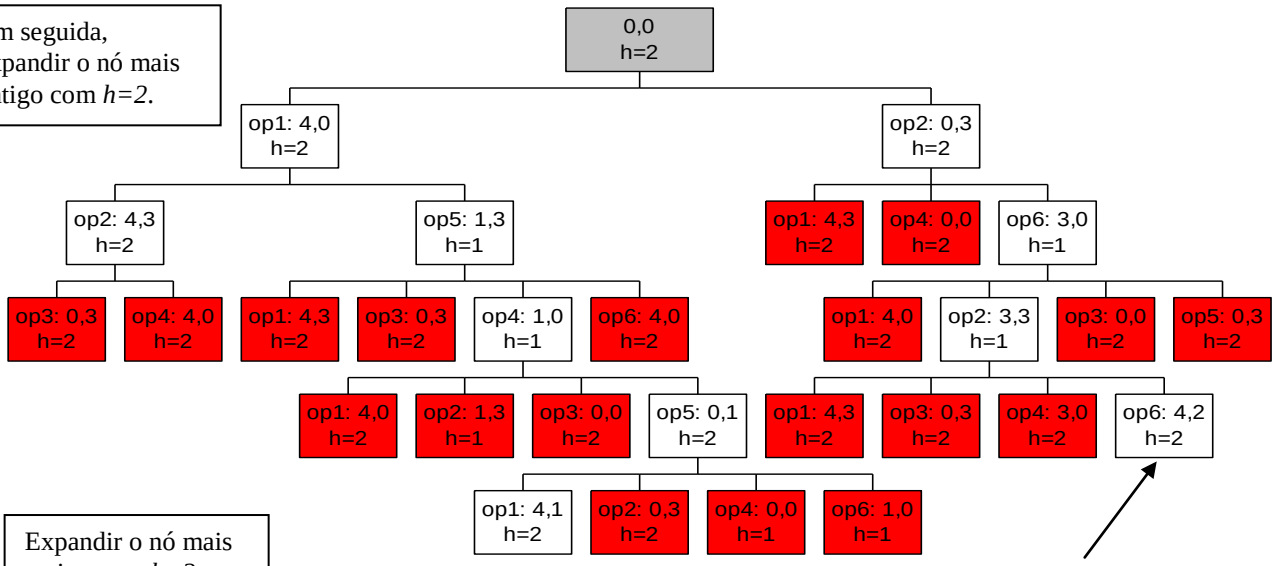


Em seguida, expandir o nó mais antigo com *h=2*.

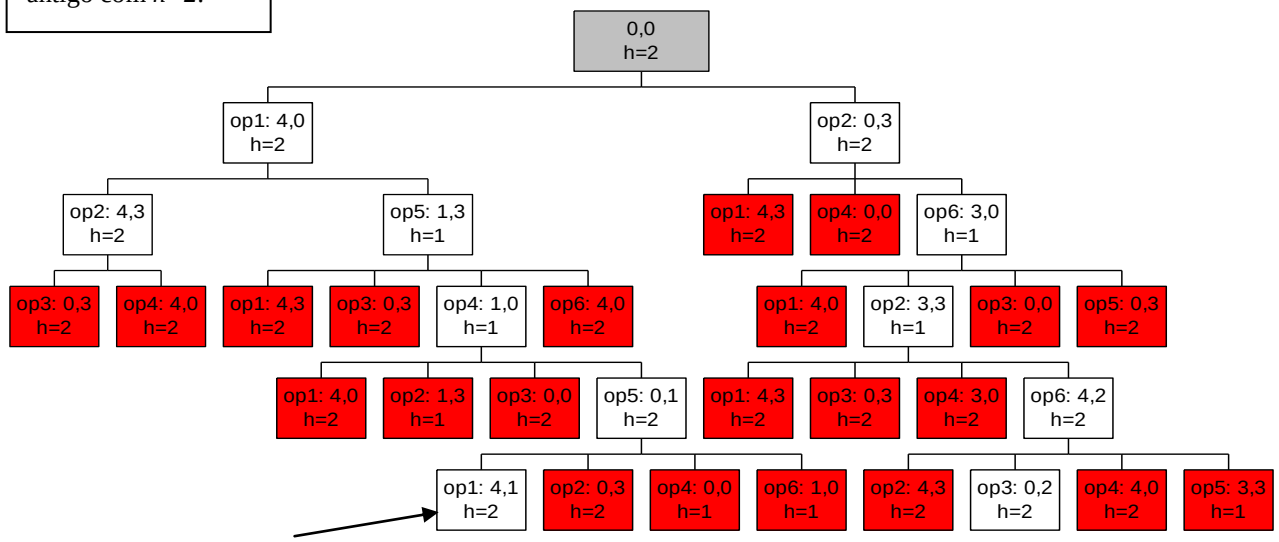
Gerou 2 já expandidos. Agora, Expandir o actualmente nó mais antigo com *h=2*.

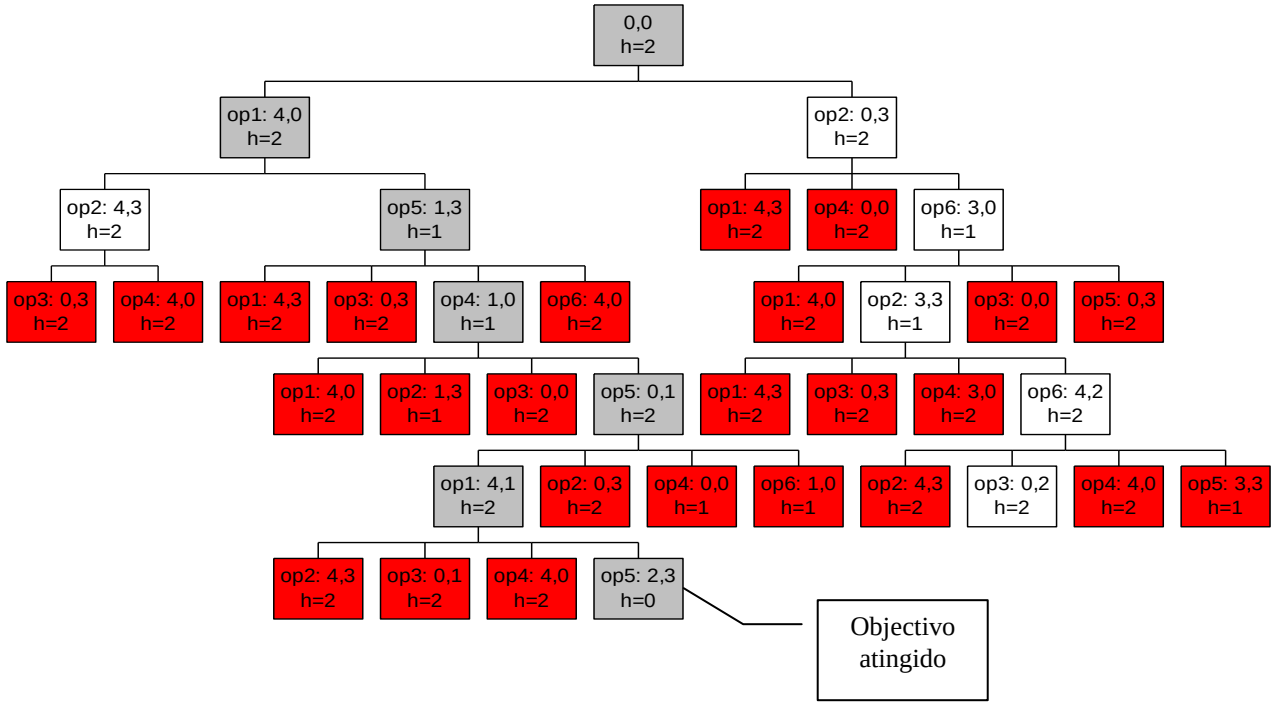


Em seguida, expandir o nó mais antigo com *h=2*.



Expandir o nó mais antigo com *h=2*.





Nós já expandidos anteriormente

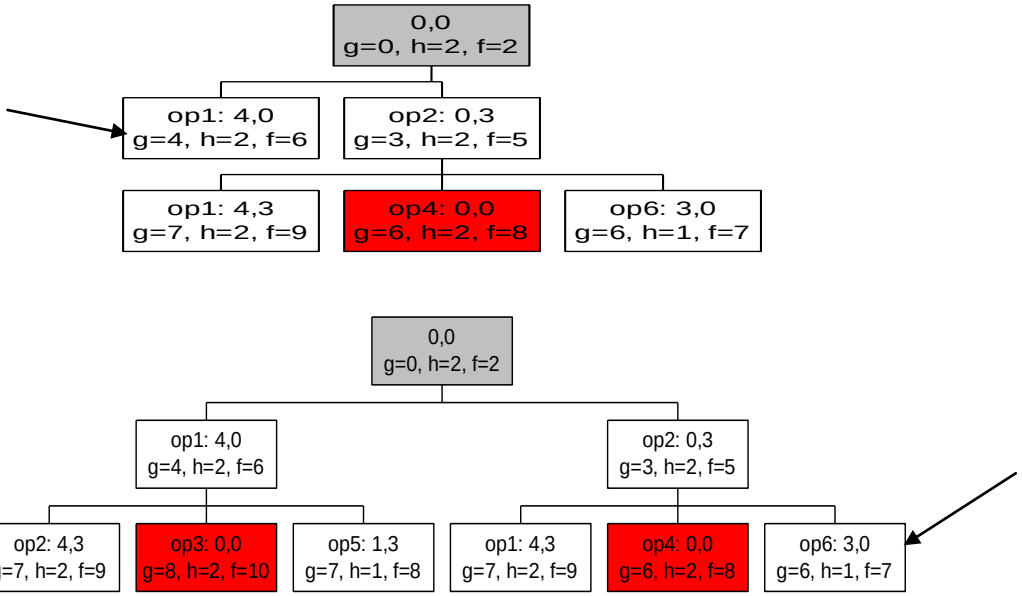
Caminho da solução

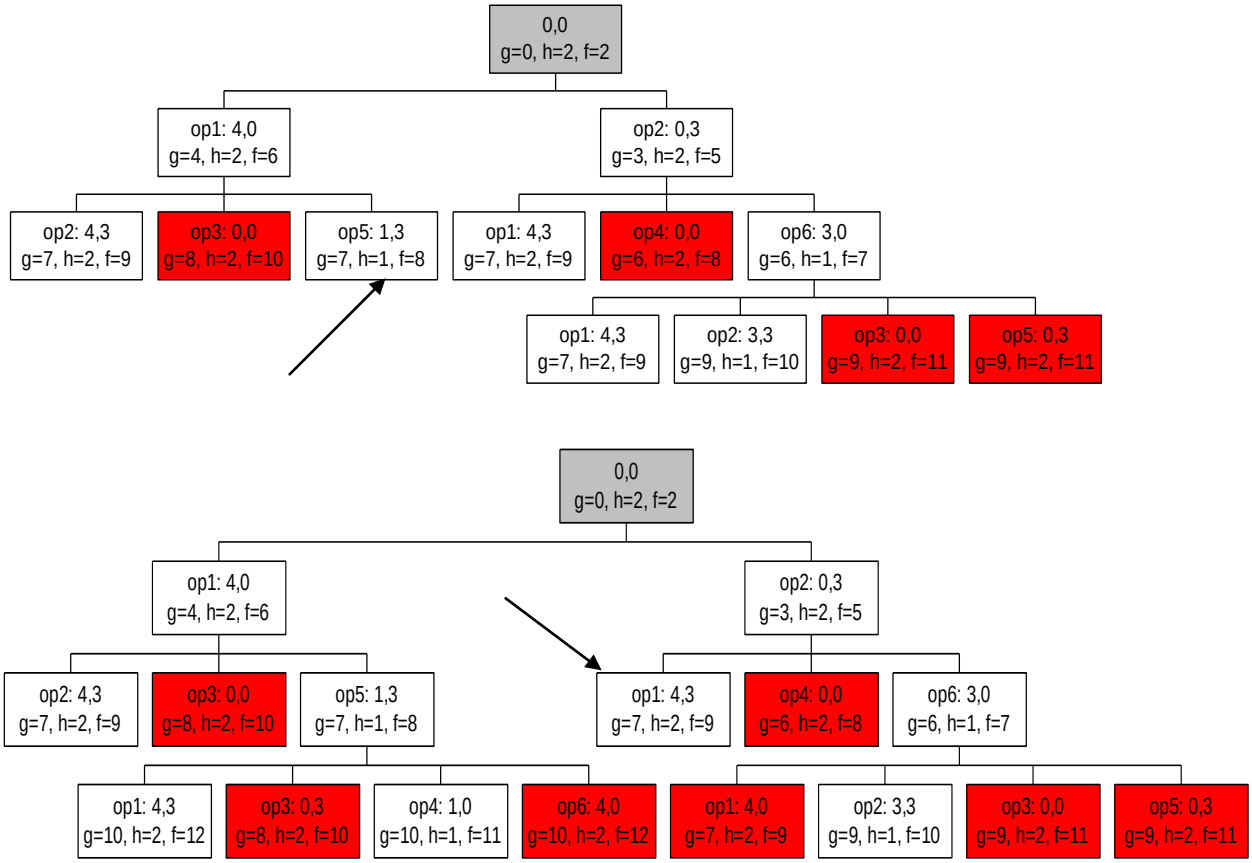
Operação		**Resultado após operação**	**Custo**
Estado Inicial		0,0	
Encher o balde X	4,0		4
Transferir X para Y		1,3	3
Despejar Y		1,0	3
Transferir X para Y		0,1	1
Encher X		4,1	4
Transferir X para Y		2,3	2

Em X ficam 2 litros, objectivo alcançado.

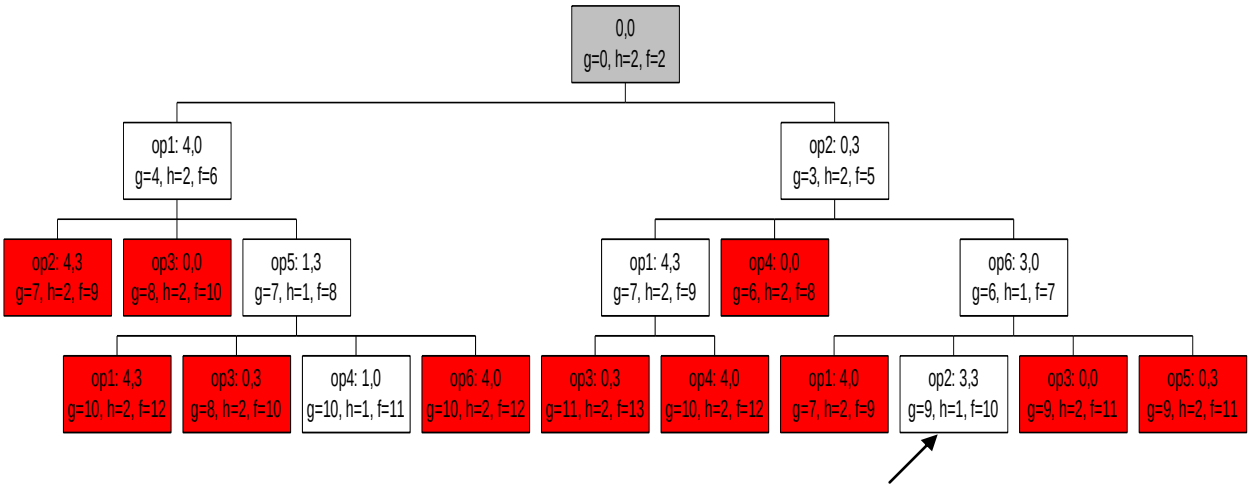
Custo: 17

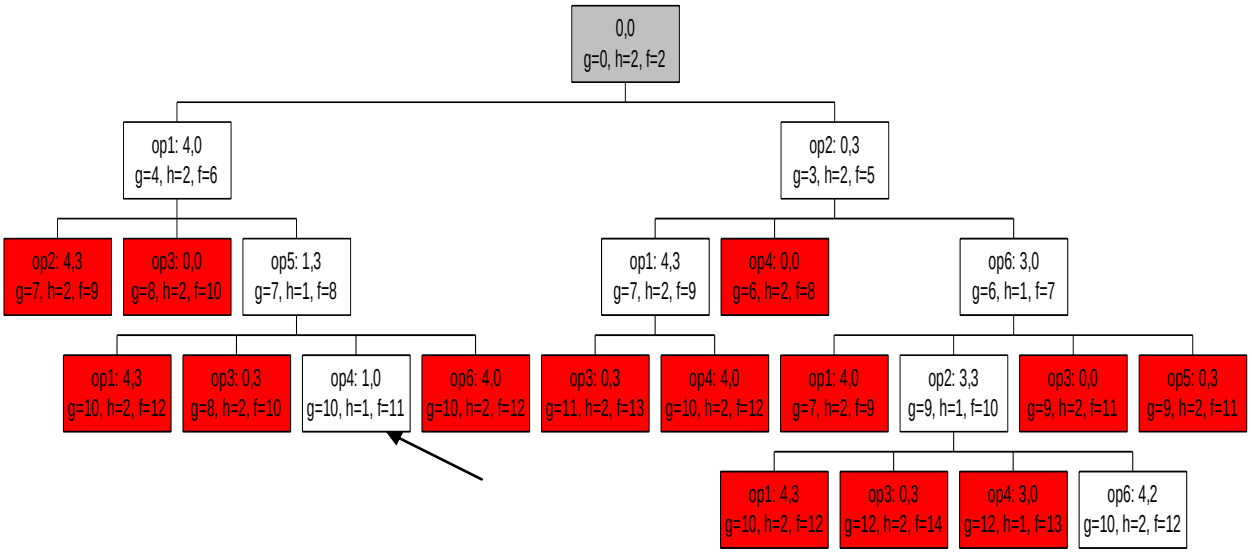
b)

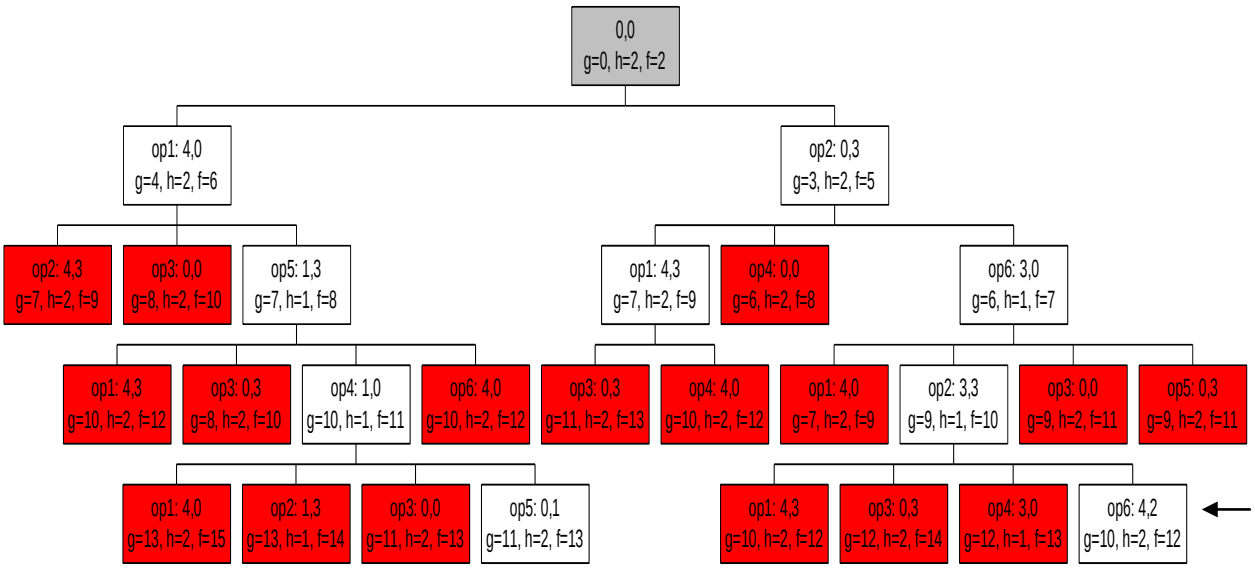


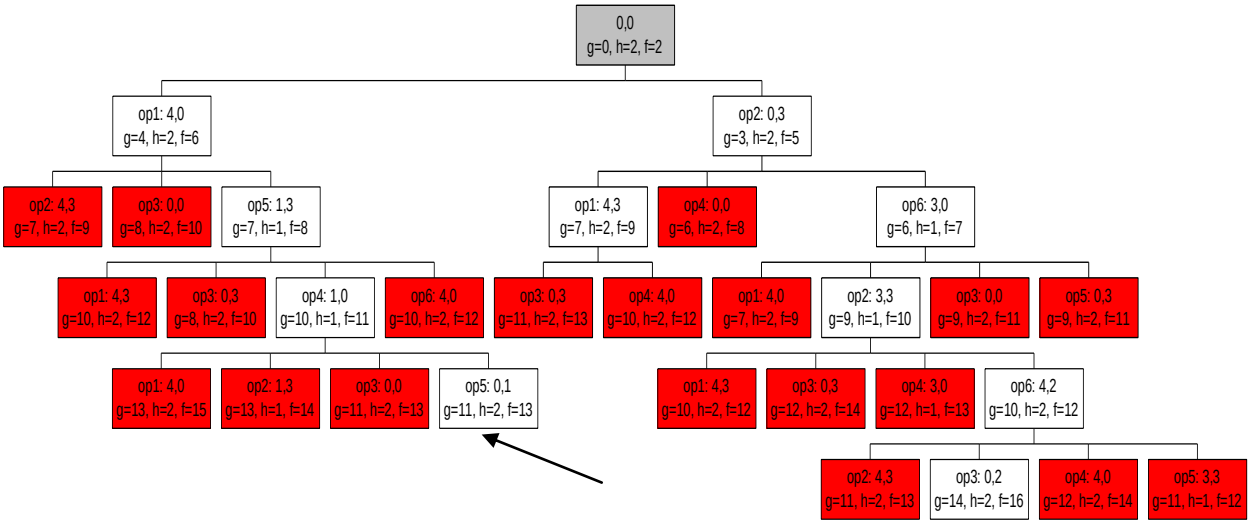


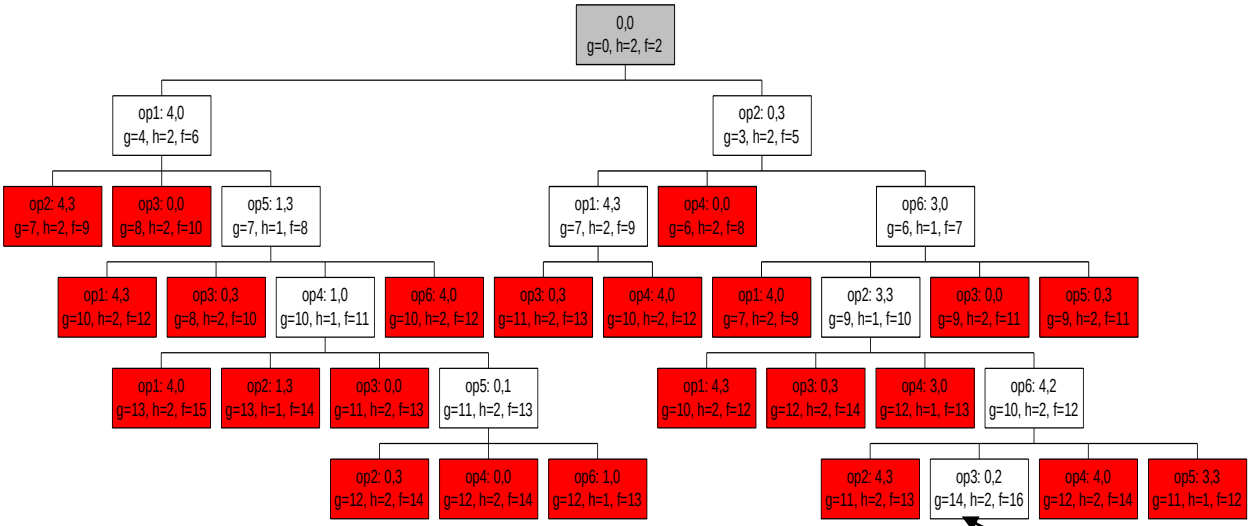
O menor *f* é 9 mas ocorre em dois nós correspondentes ao mesmo estado (4,3). Expande-se o mais antigo.

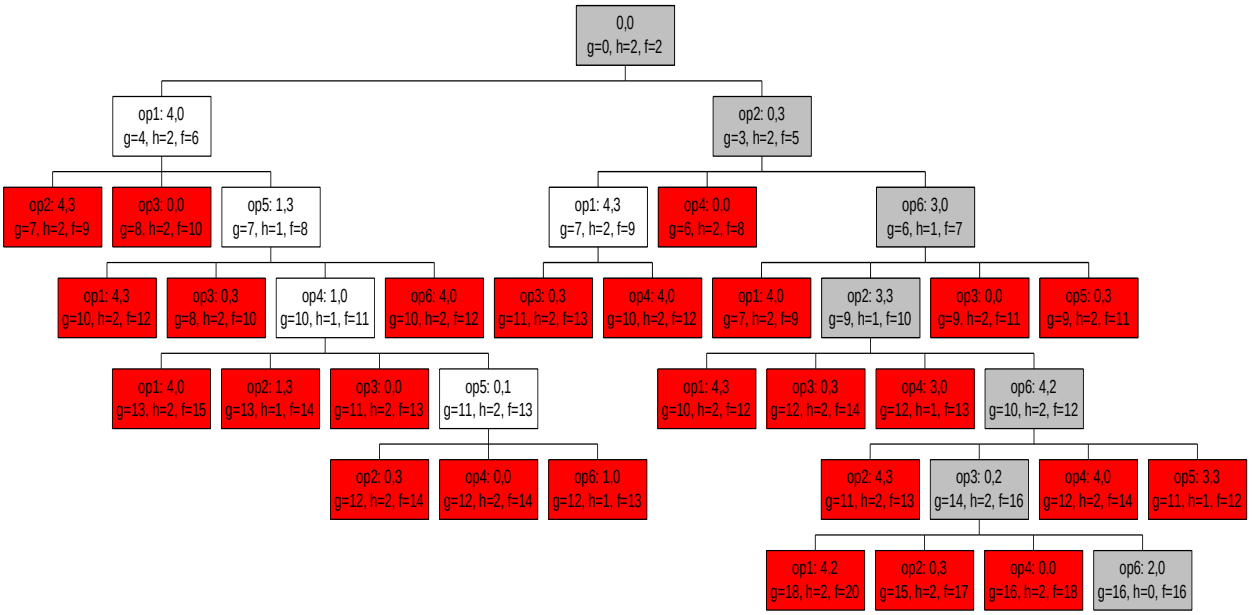



0,0
g=0, h=2, f=2
op1: 4,0
g=4, h=2, f=6
op2: 0,3
g=3, h=2, f=5
op2: 4,3
g=7, h=2, f=9
op3: 0,0
g=8, h=2, f=10
op5: 1,3
g=7, h=1, f=8
op1: 4,3
g=7, h=2, f=9
op4: 0,0
g=6, h=2, f=8
op6: 3,0
g=6, h=1, f=7
op1: 4,3
g=10, h=2, f=12
op3: 0,3
g=8, h=2, f=10
op4: 1,0
g=10, h=1, f=11
op6: 4,0
g=10, h=2, f=12
op3: 0,3
g=11, h=2, f=13
op4: 4,0
g=10, h=2, f=12
op1: 4,0
g=7, h=2, f=9
op2: 3,3
g=9, h=1, f=10
op3: 0,0
g=9, h=2, f=11
op5: 0,3
g=9, h=2, f=11
op1: 4,3
g=10, h=2, f=12
op3: 0,3
g=12, h=2, f=14
op4: 3,0
g=12, h=1, f=13
op6: 4,2
g=10, h=2, f=12


0,0
g=0, h=2, f=2
op1: 4,0
g=4, h=2, f=6
op2: 0,3
g=3, h=2, f=5
op2: 4,3
g=7, h=2, f=9
op3: 0,0
g=8, h=2, f=10
op5: 1,3
g=7, h=1, f=8
op1: 4,3
g=7, h=2, f=9
op4: 0,0
g=6, h=2, f=8
op6: 3,0
g=6, h=1, f=7
op1: 4,3
g=10, h=2, f=12
op3: 0,3
g=8, h=2, f=10
op4: 1,0
g=10, h=1, f=11
op6: 4,0
g=10, h=2, f=12
op3: 0,3
g=11, h=2, f=13
op4: 4,0
g=10, h=2, f=12
op1: 4,0
g=7, h=2, f=9
op2: 3,3
g=9, h=1, f=10
op3: 0,0
g=9, h=2, f=11
op5: 0,3
g=9, h=2, f=11
op1: 4,0
g=13, h=2, f=15
op2: 1,3
g=13, h=1, f=14
op3: 0,0
g=11, h=2, f=13
op5: 0,1
g=11, h=2, f=13
op1: 4,3
g=10, h=2, f=12
op3: 0,3
g=12, h=2, f=14
op4: 3,0
g=12, h=1, f=13
op6: 4,2
g=10, h=2, f=12


0,0
g=0, h=2, f=2
op1: 4,0
g=4, h=2, f=6
op2: 0,3
g=3, h=2, f=5
op2: 4,3
g=7, h=2, f=9
op3: 0,0
g=8, h=2, f=10
op5: 1,3
g=7, h=1, f=8
op1: 4,3
g=7, h=2, f=9
op4: 0,0
g=6, h=2, f=8
op6: 3,0
g=6, h=1, f=7
op1: 4,3
g=10, h=2, f=12
op3: 0,3
g=8, h=2, f=10
op4: 1,0
g=10, h=1, f=11
op6: 4,0
g=10, h=2, f=12
op3: 0,3
g=11, h=2, f=13
op4: 4,0
g=10, h=2, f=12
op1: 4,0
g=7, h=2, f=9
op2: 3,3
g=9, h=1, f=10
op3: 0,0
g=9, h=2, f=11
op5: 0,3
g=9, h=2, f=11
op1: 4,0
g=13, h=2, f=15
op2: 1,3
g=13, h=1, f=14
op3: 0,0
g=11, h=2, f=13
op5: 0,1
g=11, h=2, f=13
op1: 4,3
g=10, h=2, f=12
op3: 0,3
g=12, h=2, f=14
op4: 3,0
g=12, h=1, f=13
op6: 4,2
g=10, h=2, f=12
op2: 4,3
g=11, h=2, f=13
op3: 0,2
g=14, h=2, f=16
op4: 4,0
g=12, h=2, f=14
op5: 3,3
g=11, h=1, f=12





Operação		**Resultado após operação**	**Custo**
Estado Inicial		0,0	
Encher o balde Y	0,3		3
Transferir Y para X		3,0	3
Encher o balde Y	3,3		3
Transferir Y para X		4,2	1
Despejar X		0,2	4
Transferir Y para X		2,0	2

Em X ficam 2 litros, objectivo alcançado.
Custo: 16

Ficha4

3)

O enunciado permite identificar as seguintes variáveis, domínios e restrições:

Variáveis: A, B, C, D
Domínio de A, B C e D: {1,2,3,4}

Valores possíveis das variáveis, consequência das restrições unárias 2 a 8, na forma:
Variável={valores possíveis):

$A \in \{1,2,3,4\}$
$B \in \{3,4\}$
$C \in \{1,2,3,4\}$
$D \in \{3,4\}$

Valores possíveis das variáveis, consequência das restrições binárias 1 e 9, na forma (Var, Var) = {valores possíveis):

$$(A,B) \in \{(3,2)\} \vee (B,C) \in \{(3,2)\} \vee (C,D) \in \{(3,2)\}$$

Hill Climbing usando um operador que gere apenas 1 estado seguinte: Esse operador pode ser, por exemplo: *"procurar, da direita para a esquerda, a 1ª restrição violada; então, colocar nessa posição um valor que permita cumpri-la, por troca com a posição que ele actualmente ocupa* (*swap* de valores)*"*. Além disso é preciso avaliar cada estado para se progredir para "estados melhores". A função de avaliação pode ser, apenas, *h= número de restrições unárias e binárias já cumpridas*.

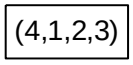
Valores possíveis das variáveis, consequência das restrições unárias 2 a 8:

1. $A \in \{1,2,3,4\}$
2. $B \in \{3,4\}$
3. $C \in \{1,2,3,4\}$
4. $D \in \{3,4\}$

Valores possíveis das variáveis, consequência das restrições binárias 1 e 9:

5. $(A,B) \in \{(3,2)\} \vee (B,C) \in \{(3,2)\}$ ou $(C,D) \in \{(3,2)\}$

Passo 0
Como se trata de um método de melhoramento iterativo, o estado inicial é completamente definido por um processo aleatório. De acordo com o enunciado, seja esse estado:
(4,1,2,3).
Este estado cumpre as restrições 1, 3, 4 e nenhuma das binárias. Usando *h=número de restrições cumpridas*, tem-se h=3.

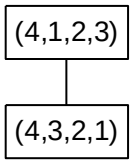


h=3

Note-se que uma qualquer solução terá h=5 dado que basta cumprir todas as unárias e uma das binárias por estarem ligadas pelo operador **ou**.

Passo 1
Estado Actual:
(4,1,2,3)

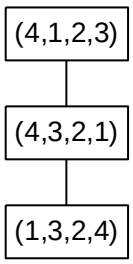
Como a restrição 2 é a única unária não cumprida, vamos passar a cumpri-la colocando o valor 3 (o primeiro possível para B) na 2ª posição; simultaneamente o valor actual de B (valor 1) passará a ocupar o lugar actual do valor 3. Resulta (4,3,2,1).



h=4

Este estado cumpre as restrições 1, 2, 3 e uma das binárias. Portanto, h=4.

Passo 2
Estado Actual: (4,3,2,1)
Como a restrição 4 é a única unária não cumprida, vamos passar a cumpri-la colocando o valor 3 (o primeiro possível para D) na 4ª posição; mas isto conduz ao estado (4,1,2,3) que já foi visitado. Portanto, vamos das a D o 2º valor possível, ou seja, D=4 por troca com a 1ª posição (a de A). Resulta (1,3,2,4)



h=5

Este estado cumpre as restrições 1, 2, 3,4 e uma das binárias. Portanto é uma solução.

Hill-Climbing usando um operador que gere *vários* estados seguintes: Conforme enunciado, vamos partir do estado (1,2,3,4). Além disso, um operador possível para gerar vários estados seguintes consiste em percorrer exaustivamente todas as combinações possíveis, isto é:

- Experimentar todas as combinações em que o 1 ocupa a 1ª posição
- Depois todas aquelas em que o 2 ocupa a 1ª posição
- etc.

Note-se que o problema admite 24 disposições possíveis, dado que:

- O primeiro dígito pode ser 1,2,3,4
- Para cada valor atribuído ao primeiro dígito, sobram três valores distintos para o 2º dígito (1,2,3 ou 1,2,4, ou 1,3,4 etc.)
- Para cada dois valores atribuídos aos primeiros dois dígitos, sobram dois valores para o 3º dígito (1,2 ou 2,3 ou 3,4, etc.)
- Para cada três valores atribuídos aos primeiros três dígitos, sobra um só valor para o 4º dígito (1, 2 3 ou 4)

Ou seja, o número de estados diferentes é 4!=4.3.2.1=24.

Embora este número não levante quaisquer problemas em termos de complexidade temporal ou espacial, vamos admitir que numa situação real isto poderia acontecer. Assim, em vez de se gerarem de

imediato os 24 estados seguintes, iremos gerar apenas, por exemplo, 6, o que significa fixar o primeiro dígito e gerar as seis combinações possíveis dos três últimos. Quanto às restrições, as unárias e binárias são as mesmas das alíneas anteriores. Assim:

Valores possíveis das variáveis, consequência das restrições unárias 2 a 8:

1. $A \in \{1,2,3,4\}$
2. $B \in \{3,4\}$
3. $C \in \{1,2,3,4\}$
4. $D \in \{3,4\}$

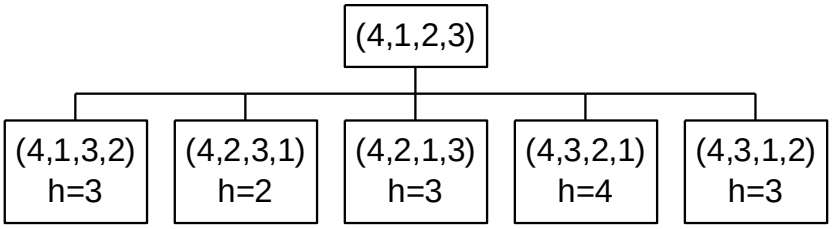
Valores possíveis das variáveis, consequência das restrições binárias 1 e 9:

5. $(A,B) \in \{(3,2)\} \vee (B,C) \in \{(3,2)\} \vee (C,D) \in \{(3,2)\}$

Partindo do estado inicial (4,1,2,3) conforme enunciado, tem-se:

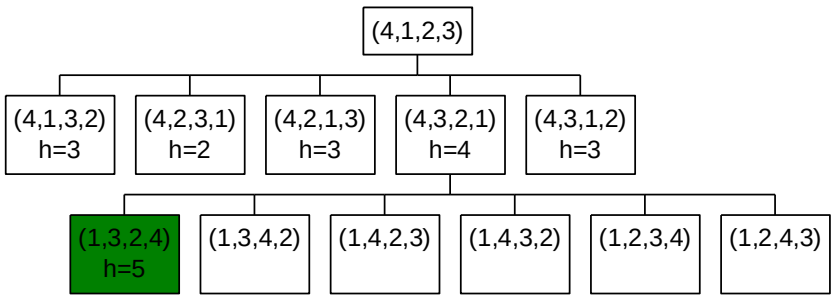
Passo 0
(4,1,2,3)
Este estado cumpre as restrições 1, 3, 4 e nenhuma das binárias. Usando *h=número de restrições cumpridas*, tem-se h=3.

Passo 1
Estado Actual: (4,1,2,3)
Estados seguintes:



Portanto, escolhe-se para expandir em seguida o estado (4,3,2,1)

Passo 2
Estado Actual: (4,3,2,1)
Estados seguintes: Como tínhamos o "4" em primeiro lugar, agora é a vez do "1" em primeiro lugar. Obtém-se:



Como para o primeiro estado se tem h=5, o problema está resolvido.

e) *Reparação Heurística* pelo *min-conflicts method*: O método consiste em percorrer cada uma das variáveis e atribuir-lhes os diversos valores possíveis. Escolher o estado que provoca menor número de conflitos. Se todos provocam mais, descartar e manter o actual. Se existirem vários com o mesmo

número, escolher um deles aleatoriamente. Se esse número for igual ao número de conflitos do estado actual, incluir o estado actual neste sorteio.

Depois de cada modificação, testar se ainda há restrições violadas. Quando todas forem cumpridas, o problema está resolvido. As restrições são:

1. $A \in \{1,2,3,4\}$
2. $B \in \{3,4\}$
3. $C \in \{1,2,3,4\}$
4. $D \in \{3,4\}$

Nesta abordagem, a 5ª restrição $(A,B) \in \{(3,2)\} \vee (B,C) \in \{(3,2)\} \vee (C,D) \in \{(3,2)\}$ é mais convenientemente expressa na forma:

5. $x_3 < x_2$

significando que o objecto 3 tem de preceder o objecto 2.

Além disso, dado que neste problema cada variável tem de assumir um valor diferente, consideraremos explicitamente ainda uma 6ª restrição, implicitamente considerada nas resoluções anteriores, mas de especial conveniência para a presente abordagem:

6. $\forall\, x \in \{A,B,C,D\},\ x_i \neq x_j \mid i,j=1..4 \wedge i \neq j$

Sobre este algoritmo notar que:

- Cada variável é tratada separadamente (uma de cada vez).
- Ao estado actual seguem-se *n* estados gerados por modificação do valor apenas da variável em consideração. Para escolher um estado de entre vários com o mesmo número de conflitos, o actual também conta, i.e., acaba por nunca se progredir para um estado que origine maior número de conflitos na variável em consideração. Quando houver um número de conflitos igual, escolhe-se um estado aleatoriamente.
- Como consequência disto, se um estado tem 0 conflitos numa dada variável, importa na mesma gerar os seus sucessores, porque se houver outro com número de conflitos 0 na variável considerada, ele poderá ser escolhido aleatoriamente.
- A avaliação do número de conflitos faz-se, para cada variável, apenas para essa variável (os conflitos nas outras não interessam nesse momento) incluindo os gerados por restrições binárias.
- Para se testar solução atingida, verificar se nenhuma restrição é ainda violada depois de se escolher um novo estado, ou se se consegue completar um loop de iterações por todas as variáveis com 0 conflitos em todas elas.

Nesta resolução vamos assumir que:

- Em caso de empate no número de conflitos, a escolha aleatória do estado seguinte devolve sempre o primeiro estado. Se o empate for com o estado actual, opta-se pelo estado actual.
- Como consequência, se um estado apresenta 0 conflitos numa dada variável, não interessa gerar os estados seguintes.

Assim, partindo de (4,1,2,3) conforme indicado no enunciado (num caso real o estado inicial seria aleatório), tem-se:

Ficha5

1. a) O operador de recombinação opera sobre dois indivíduos criando dois novos indivíduos com a mistura do material genético dos progenitores. O operador de mutação opera apenas sobre os indivíduos separadamente.
De uma forma geral, o operador de recombinação é aplicado com uma probabilidade mais alta que o operador de mutação (ex: 70% e 0.1%)
O operador de mutação permite explorar zonas próximas do espaço de procura, enquanto que o operador de mutação permite explorar zonas mais distantes do espaço de procura.

Ambos os operadores são necessários, essencialmente pela última diferença apresentada. O operador de recombinação permite ao AG ir melhorando a solução através de "pequenos saltos" no espaço de procura. No entanto, se a população do AG convergir para um óptimo local, o operador de recombinação não permite a procura de outras zonas que levam o AG ao encontro da solução óptima. O operador de mutação, permite manter na população indivíduos mais dispersos no espaço de procura permitindo a fuga dos óptimos locais.

b) Pode dizer-se que os AG não são algoritmos de optimização, visto que não são métodos exactos que dêem a garantia de encontrar a solução óptima de um determinado problema. O funcionamento dos AG permite sim, ir melhorando a solução encontrada através de um processo iterativo ao longo de um conjunto de gerações.

De facto, partindo de um conjunto de soluções candidatas, que são avaliadas de acordo com uma função de avaliação, e aplicando ciclicamente os operadores de selecção, recombinação e mutação, o AG permite ir melhorando a melhor solução que possui numa determinada geração. Neste sentido podem ser chamados de algoritmos de melhoramento.

c) Seja $f(x_i)$ o fitness dos indivíduos i, i=1,... 6.

1º Calcular o somatório dos fitnesses da população:

$$FitTotal = \sum_{i=1}^{6} f(x_i) = 205$$

2º Calcular as probabilidades individuais de um indivíduo ser seleccionado através da fórmula:

$$p(x_i) = \frac{f(x_i)}{FitTotal}$$

Indivíduo	**Probabilidade de ser seleccionado**
1	5/205 = 0.024
2	10/205 = 0.049
3	15/205 = 0.073
4	25/205 = 0.122
5	50/205 = 0.244
6	100/205 = 0.488

2. a) Função de avaliação

Uma solução possível é minimizar o nº de caracteres que diferem relativamente à solução final. No entanto, a solução ideal é a seguinte: para diferenciar mais as soluções relativamente aos caracteres que a compõem é mais conveniente utilizar a Distância de Hamming entre a solução óptima e a solução candidata:

$$Diff(s^{opt}, s^{cand}) = \sum_{i=1}^{N} \left[Ascii(s_i^{opt}) - Ascii(s_i^{cand}) \right]^2$$

onde **N** é o tamanho da palavra e **Ascii(x)** é o valor do código ascii do caracter **x**.

b) Função de avaliação - exemplos

Utilizando a segunda alternativa indicada na alínea anterior a função de avaliação devolveria os seguintes valores:

Fitness(MALINHA) =
= [Ascii(G)-Ascii(M)]^2 + [Ascii(A)-Ascii(A)]^2 + [Ascii(L)-Ascii(L)]^2+ [Ascii(I)-Ascii(I)]^2+ [Ascii(N)-Ascii(N)]^2+ [Ascii(H)-Ascii(H)]^2+ [Ascii(A)-Ascii(A)]^2
= (71-77)^2 + 0 + 0 + 0 + 0 + 0 + 0 =36

Fitness(ALINHAG) =
[Ascii(G)-Ascii(A)]^2 + [Ascii(A)-Ascii(A)]^2 + [Ascii(L)-Ascii(L)]^2+ [Ascii(I)-Ascii(I)]^2+ [Ascii(N)-Ascii(N)]^2+ [Ascii(H)-Ascii(H)]^2+ [Ascii(A)-Ascii(G)]^2
= (71-65)^2 + 0 + 0 + 0 + 0 + 0 + (65-71)^2 =36 + 36 =72

Fitness(SALINHA) =
= [Ascii(G)-Ascii(S)]^2 + [Ascii(A)-Ascii(A)]^2 + [Ascii(L)-Ascii(L)]^2+ [Ascii(I)-Ascii(I)]^2+ [Ascii(N)-Ascii(N)]^2+ [Ascii(H)-Ascii(H)]^2+ [Ascii(A)-Ascii(A)]^2
= (71-83)^2 + 0 + 0 + 0 + 0 + 0 + 0 =144

A palavra MALINHA seria aquela que teria mais hipóteses de se reproduzir, visto que o que se pretende é minimizar o valor de *Fitness*.

c) Operadores de recombinação
O operador de recombinação de 1 ponto de corte é apropriado para este problema, apesar de poder destruir uma solução *quase* perfeita.
Um operador de recombinação mais apropriado seria: Preservar a parte comum dos progenitores e trocar as restantes. Por exemplo:

Pai 1: MALINHA
Pai 2: ALINHAG

Preservando a parte comum (ALINHA) trocar-se-ia apenas o M com o G. Este operador de crossver, como se pode observar operador de forma *desalinhada.* O resultado seria:

Pai 1: M**ALINHA**
Pai 2: **ALINHA**G

Filho1: G**ALINHA**
Filho2: **ALINHA**M

d) Operadores de mutação

Duas possibilidades:

Operador de mutação por troca aleatória de um caracter:

AGLINHG → **G**GLINHG (o **A** foi trocado por um **G**)

Operador de mutação por troca de posição entre 2 caracteres:

AGLINH**G** → **G**GLINH**A** (o **A** e o G trocaram de posição na palavra)