Projeto Nº 1: Época Normal

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal
2024/2025

Prof. Joaquim Filipe
Eng. Filipe Mariano

1. Descrição do Jogo - Adji-boto*

O Adji-boto* (uma variante do Adji-boto e Oware, especialmente concebida para este projeto) é um jogo de estratégia da família dos jogos de tabuleiro Mancala, que hoje em dia ainda têm uma grande popularidade. Nesta versão do jogo Adji-boto*, existe um tabuleiro com 2 linhas e 6 buracos em cada linha e é iniciado com 8 peças em cada buraco.

Figura 1: Tabuleiro real de Mancala na versão Oware.

1.1. Regras

O jogo desenrola-se da seguinte forma:

- Cada jogador fica com uma das linhas de buracos;
- As jogadas são feitas à vez e, em cada turno, um jogador retira todas as peças de um dos buracos da sua linha e vai depositando cada uma dessas peças retiradas no buraco adjacente e em cada um dos buracos seguintes, no sentido contrário ao dos ponteiros do relógio;

- Quando ao depositar uma peça em cada buraco seguinte, se chegar ao buraco em que inicialmente se retirou as peças (foi efetuada uma volta completa), deve-se saltar essa casa colocando a(s) peça(s) que ainda sobra(m) na(s) casa(s) seguinte(s);
- Se ao terminar a jogada **ficar na casa final 1, 3 ou 5 peças, essas peças devem ser "capturadas", sendo retiradas do tabuleiro**.

1.2. Exemplo de Jogada

Para tornar mais explícito como se desenrola o jogo, irá ser apresentado num esquema de duas imagens um tabuleiro antes de uma jogada e após uma jogada:

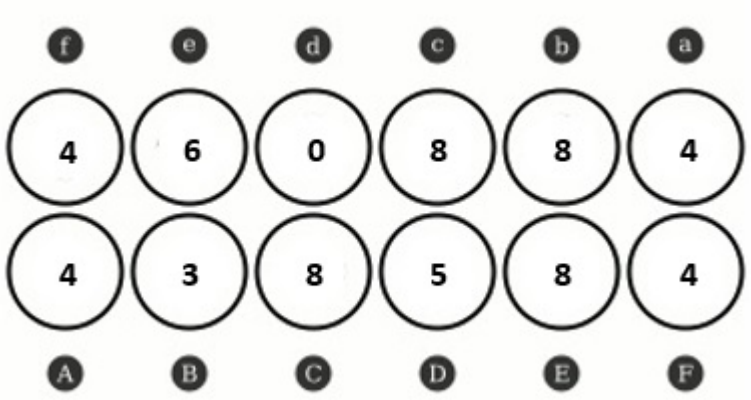

Figura 2: Tabuleiro antes da jogada.

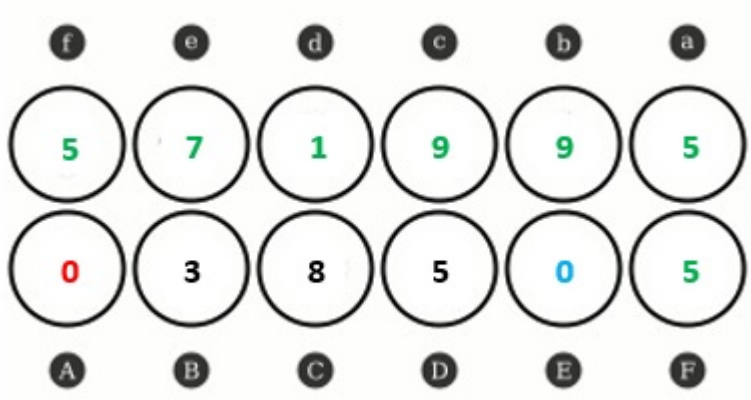

Figura 3: Tabuleiro após a jogada.

Passos:

- O buraco selecionado pelo jogador para iniciar a jogada foi o `E` na 2ª linha, por isso retirou todas as peças desse buraco - número a azul;
- As peças existentes no buraco anteriormente mencionado foram depositadas nos buracos seguintes adjacentes, uma peça por buraco (`F`, `a`, `b`, `c`, `d`, `e`, `f`) - número a verde - tendo sido colocada a última peça no buraco `A` - número a vermelho;
- Como o buraco `A` ficou com `5` peças, pelo facto de ter sido depositada uma peça proveniente do buraco `E`, pode ser aplicada a regra de **se na casa final ficarem 1, 3 ou 5 peças, essas peças podem ser capturadas**, sendo retiradas do tabuleiro - número a vermelho.

NOTA: Apenas por essa razão é que o buraco `A` ficou com `0` peças, caso contrário tinha ficado com o número de peças igual ao antes da jogada, mais a peça proveniente do buraco `E`.

1.3. Determinar o Vencedor

O jogo termina quando não existirem peças no tabuleiro, sendo o vencedor o jogador que capturar **mais** peças.

2. Objetivo do Projeto

No âmbito deste projeto vamos considerar o Adji-boto* como uma versão simplificada dos jogos mencionados anteriormente, em que o principal objetivo consiste em explorar o espaço de possibilidades tentando vários caminhos possíveis até encontrar a solução, recorrendo aos algoritmos de procura lecionados nas aulas. Neste sentido, o intuito do projeto passa por testar diversos tabuleiros com diversos algoritmos e analisar as soluções encontradas.

Sendo o Adji-boto* uma variante mais simplificada, as diferenças ao nível das regras são as seguintes:

- Existe apenas um jogador;
- O jogador pode jogar em qualquer uma das linhas;
- Cada jogada pode ser feita em qualquer uma das linhas e em qualquer um dos buracos, sem qualquer restrição na repetição da jogada.

Pretende-se que os alunos desenvolvam um programa, em `Lisp`, aplicando diversos métodos de procura em espaço de estados conforme se indica detalhadamente mais adiante, para apresentar a sequência de estados ou de jogadas que conduzem de uma posição inicial do puzzle até uma posição final. A solução óptima consiste no caminho com menor número de jogadas entre o estado inicial e o estado final.

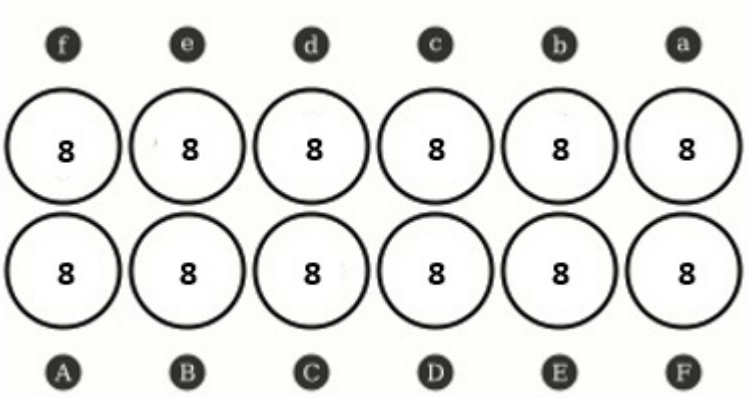

Figura 4: Estado inicial.

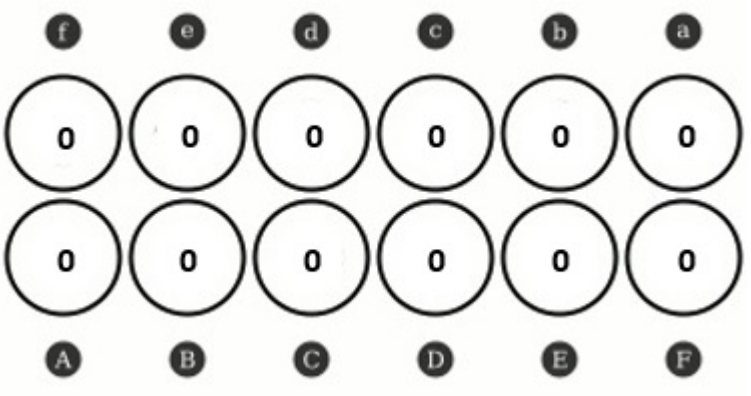

Figura 5: Estado final.

3. Formulação do Problema

3.1. Tabuleiro

O tabuleiro é representado sob a forma de uma lista composta por `2` outras listas, cada uma delas com `6` átomos. Cada uma das listas representa uma linha do tabuleiro, enquanto que cada um dos átomos representa um buraco dessa linha. Por outras palavras, o tabuleiro é representado por uma lista de listas em que `0` representa um buraco vazio e `#n` representa um buraco com `n` peças.

Abaixo mostram-se respectivamente a representação do tabuleiro apresentado anteriormente como estado inicial (Figura 4) e do tabuleiro finalizado (Figura 5).

```
(
 (8 8 8 8 8 8)
 (8 8 8 8 8 8)
)
```

Figura 6: Representação do tabuleiro inicial.

```
(
 (0 0 0 0 0 0)
 (0 0 0 0 0 0)
)
```

Figura 7: Representação do tabuleiro finalizado.

3.2. Operadores

Existe um operador para este problema que consiste no retirar das peças de um buraco, colocando uma peça retirada sequencialmente em cada buraco seguinte até esgotar as peças retiradas do buraco inicial. Este operador poderá ser aplicado em qualquer buraco que contenha peças.

3.3. Solução Encontrada

A solução pode representar-se por uma sequência de estados, desde o estado inicial até ao estado final. **A solução para o problema corresponde a ter todos os buracos do tabuleiro sem peças (como na figura 5 e 7).**

3.4. Estrutura do Programa

O programa deverá estar dividido em três partes, cada uma num ficheiro diferente:

1. Uma parte com a implementação dos métodos de procura, de forma independente do domínio de aplicação;

2. Outra para implementar a resolução do problema, incluindo a definição dos operadores e heurísticas, específicos do domínio de aplicação;
3. E a terceira parte para fazer a interação com o utilizador e para proceder à escrita e leitura de ficheiros.

Enquanto a primeira parte do programa deverá ser genérica para qualquer problema que possa ser resolvido com base no método de procura selecionado, a segunda parte é específico do domínio de aplicação, nomeadamente o problema que este projeto se propõe resolver.

O projeto deverá apresentar um estudo comparativo do comportamento dos três métodos seguintes, conforme explicado na secção 4: procura em largura (BFS), procura em profundidade (DFS) e A*.

Para além destas três formas de procura, cada grupo pode programar, aplicar e estudar os algoritmos Simplified Memory A* (SMA*), Interactive Deepening A* (IDA*) e Recursive BestFirst Search (RBFS), que representarão um bónus para quem os implementar (ver Secção 7).

No caso dos métodos informados, o programa deverá utilizar funções heurísticas modulares, ou seja, que possam ser colocadas ou retiradas do programa de procura como módulos.

As heurísticas não devem estar embutidas de forma rígida no programa de procura. Exige–se a utilização de duas heurísticas, uma fornecida no fim do presente documento e outra desenvolvida pelos alunos.

O projeto deverá incluir a implementação de cada um dos métodos, de forma modular, permitindo que o utilizador escolha qualquer um deles, conjuntamente com os seus parâmetros (heurística, profundidade, etc.) para a resolução de um dado problema.

4. Experiências e Estudos

Pretende-se que o projeto estude, para cada problema fornecido em anexo, o desempenho de cada algoritmo e, no caso dos algoritmos de procura informados, de cada uma das heurísticas propostas, apresentando, em relação a cada problema, a solução encontrada e dados estatísticos sobre a sua eficiência, nomeadamente o fator de ramificação média, o número de nós gerados, número de nós expandidos, a penetrância, o tempo de execução e o caminho até à solução.

Os projetos deverão apresentar os dados acima referidos num ficheiro produzido automaticamente pelo programa, sendo descontado `0,5` valor por cada problema não resolvido. No caso de ser apresentada a solução, mas não o estudo de desempenho das heurísticas o desconto é de apenas `0,2` valor por cada caso.

Estes problemas deverão estar num ficheiro `problemas.dat`, utilizando a notação atrás indicada. O último problema (`G`) será apresentado durante a avaliação oral e inserido no ficheiro `problemas.dat` para verificar o funcionamento do projeto.

5. Heurísticas

Sugere-se usar como heurística de base, uma heurística que privilegia os tabuleiros com o maior número de peças capturadas. Para um determinado tabuleiro x:

$h(x) = o(x) - c(x)$

em que:

- o(x) é o número de peças a capturar no tabuleiro x;

- c(x) é o número de peças já capturadas no tabuleiro x.

Esta heurística pode ser melhorada para refletir de forma mais adequada o conhecimento acerca do puzzle e assim contribuir para uma maior eficiência dos algoritmos de procura informados.

Além da heurística acima sugerida, deve ser definida pelo menos uma segunda heurística que deverá melhorar o desempenho dos algoritmos de procura informados em relação a primeira fornecida.

6. Grupos

Os projetos deverão ser realizados em grupos de, no máximo, duas pessoas sendo contudo sempre sujeitos a avaliação oral individual para confirmação da capacidade de compreensão dos algoritmos e de desenvolvimento de código em LISP.

O grupo poderá ser constituido por alunos que frequentaram turmas diferentes.

7. Datas

Entrega do projeto: 20 de Dezembro de 2024, até as 23:59.

8. Documentação a Entregar

A entrega do projeto e da respetiva documentação deverá ser feita através do Moodle, na zona do evento "Entrega do Projeto em Época Especial". Todos os ficheiros a entregar deverão ser devidamente arquivados num ficheiro comprimido (`ZIP` com um tamanho máximo de 5Mb), até à data acima indicada. O nome do arquivo deve seguir a estrutura `<iniciaisAluno1>_<numeroAluno1>_<iniciaisAluno2>_<numeroAluno2>_P1`.

8.1. Código fonte

Os ficheiros de código devem ser devidamente comentados e organizados da seguinte forma:

projeto.lisp Carrega os outros ficheiros de código, escreve e lê ficheiros, e trata da interação com o utilizador.

puzzle.lisp Código relacionado com o problema.

procura.lisp Deve conter a implementação de:

1. Algoritmo de Procura de Largura Primeiro (BFS)
2. Algoritmo de Procura do Profundidade Primeiro (DFS)
3. Algoritmo de Procura do Melhor Primeiro (A*)
4. Os algoritmos SMA*, IDA* e/ou RBFS (caso optem por implementar o bónus)

8.2. Exercícios

Deverá haver um ficheiro de exercícios, com a designação `problemas.dat`, contendo todos os exemplos de tabuleiro que se quiser fornecer ao utilizador, organizados de forma sequencial, e que este deverá escolher mediante um número inserido no interface com o utilizador.

Esse número representa o número de ordem na sequência de exemplos. O ficheiro deverá ter várias listas, separadas umas das outras por um separador legal, e não uma lista de listas. Essas listas serão tantas quantos

os problemas fornecidos.

Na oral, os docentes irão solicitar que se adicione mais um exemplo, numa dada posição do ficheiro, que deverá imediatamente passar a ser selecionável através do interface com o utilizador e ser resolvido normalmente.

8.3. Manuais

No âmbito da Unidade Curricular de Inteligência Artificial pretende–se que os alunos pratiquem a escrita de documentos recorrendo à linguagem de marcação ***Markdown***, que é amplamente utilizada para os ficheiros ***ReadMe*** no ***GitHub***. Na Secção 4 do guia de Laboratório nº 2, encontrará toda a informação relativa à estrutura recomendada e sugestões de ferramentas de edição para ***Markdown***.

Para além de entregar os ficheiros de código e de problemas, é necessário elaborar e entregar 2 manuais (o manual de utilizador e o manual técnico), em formato PDF juntamente com os sources em MD, incluídos no arquivo acima referido.

Manual Técnico:

O Manual Técnico deverá conter o algoritmo geral, por partes e devidamente comentado; descrição dos objetos que compõem o projeto, incluindo dados e procedimentos; identificação das limitações e opções técnicas. Deverá ser apresentada uma análise critica dos resultados das execuções do programa, onde deverá transparecer a compreensão das limitações do projeto. Deverão usar uma análise comparativa do conjunto de execuções do programa para cada algoritmo e cada problema, permitindo verificar o desempenho de cada algoritmo e das heurísticas. Deverá, por fim, apresentar a lista dos requisitos do projeto (listados neste documento) que não foram implementados.

Manual de Utilizador:

O Manual do Utilizador deverá conter a identificação dos objetivos do programa, juntamente com descrição geral do seu funcionamento; explicação da forma como se usa o programa (acompanhada de exemplos); descrição da informação necessária e da informação produzida (ecrã/teclado e ficheiros); limitações do programa (do ponto de vista do utilizador, de natureza não técnica).

9. Avaliação

Tabela 1: Grelha de classificação.

Funcionalidade	**Valores**
Representação do estado e operadores	2.5
Funções referentes ao puzzle	2.5
Procura em profundidade e largura	2.5
Procura com A* e heurística dada	2.5
Implementação de nova heurística	1.5
Resolução dos problemas a) a g)	3
Resolução do problema h) (dado na avalição oral)	0.5

Funcionalidade	Valores
Qualidade do código	2
Interação com o utilizador	1
Manuais (utilizador e técnico)	2
Total	20
Bónus	3

O valor máximo da nota são 20 valores, já com a integração do valor do bónus.

Os problemas a) a g) estão descritos na Secção Experiências, enquanto que o problema h) será facultado durante a avaliação oral. A avaliação do projeto levará em linha de conta os seguintes aspectos:

- Data de entrega final – Existe uma tolerância de 4 dias em relação ao prazo de entrega, com a penalização de 1 valor por cada dia de atraso. Findo este período a nota do projeto será 0.
- Correção processual da entrega do projeto – (Moodle; manuais no formato correto). Anomalias processuais darão origem a uma penalização que pode ir até 3 valores.
- Qualidade técnica – Objetivos atingidos; Código correto; Facilidade de leitura e manutenção do programa; Opções técnicas corretas.
- Qualidade da documentação – Estrutura e conteúdo dos manuais que acompanham o projeto.
- Avaliação oral – Eficácia e eficiência da exposição; Compreensão das limitações e possibilidades de desenvolvimento do programa. Nesta fase poderá haver lugar a uma revisão total da nota de projeto.

10. Recomendações Finais

Com este projeto pretende–se motivar o paradigma de programação funcional. A utilização de variáveis globais, de instruções de atribuição do tipo set, setq, setf, de ciclos, de funções destrutivas ou de quaisquer funções com efeitos laterais é fortemente desincentivada dado que denota normalmente uma baixa qualidade técnica. A sequenciação só será permitida no contexto das funções de leitura/escrita.

As únicas excepções permitidas a estas regras poderão ser a utilização da instrução loop para implementar os ciclos principais dos algoritmos implementados (como alternativa a uma solução puramente recursiva), em conjugação com as variáveis globais Abertos e Fechados para manutenção das listas de nós.

ATENÇÃO: Suspeitas confirmadas de plágio serão penalizadas com a anulação de ambos os projetos envolvidos (fonte e destino), e os responsáveis ficam sujeitos à instauração de processo disciplinar.

Anexos - Problemas

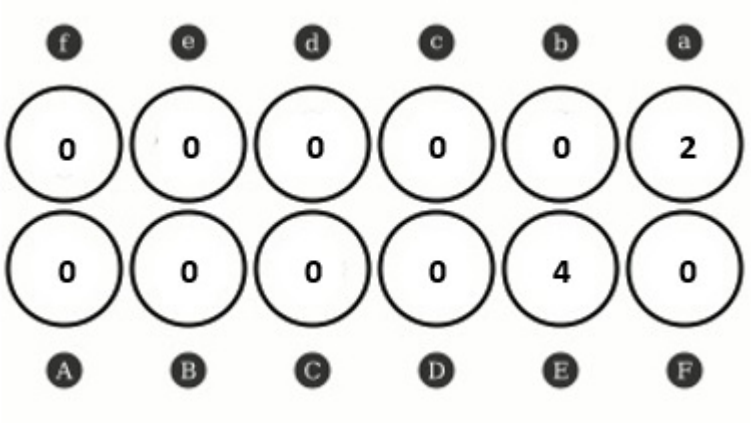

Figura 8: Problema a).

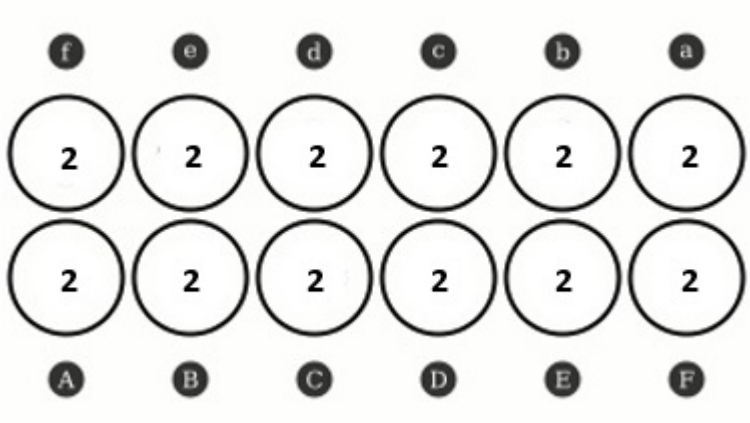

Figura 9: Problema b).

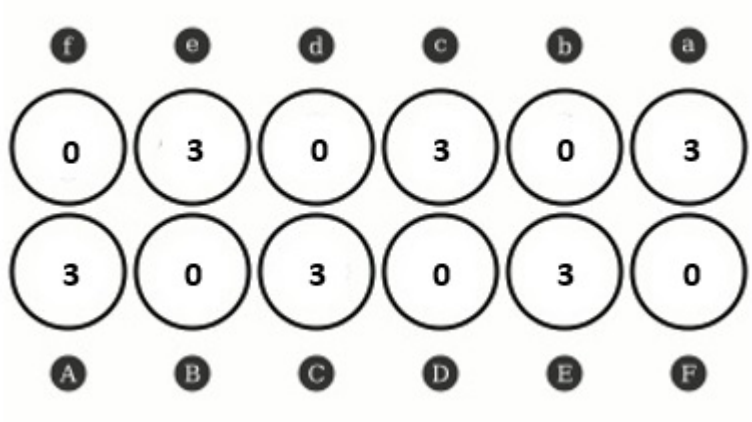

Figura 10: Problema c).

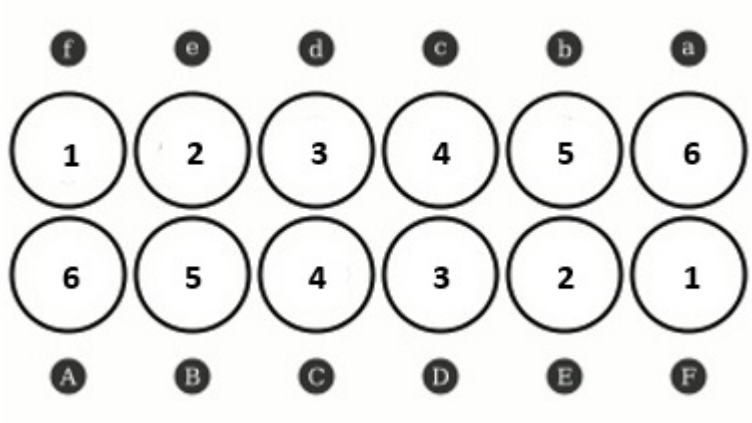

Figura 11: Problema d).

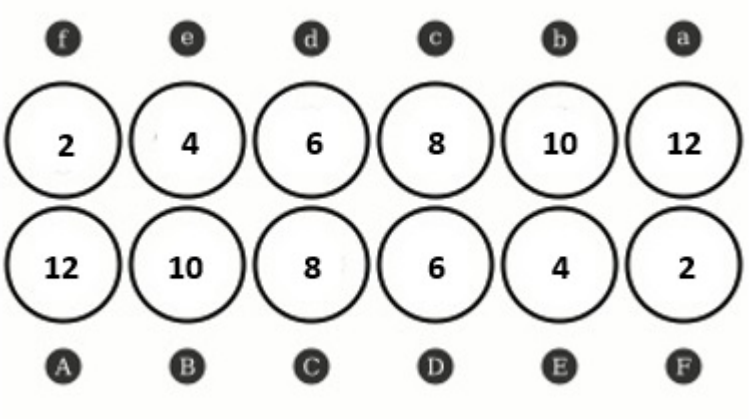

Figura 12: Problema e).

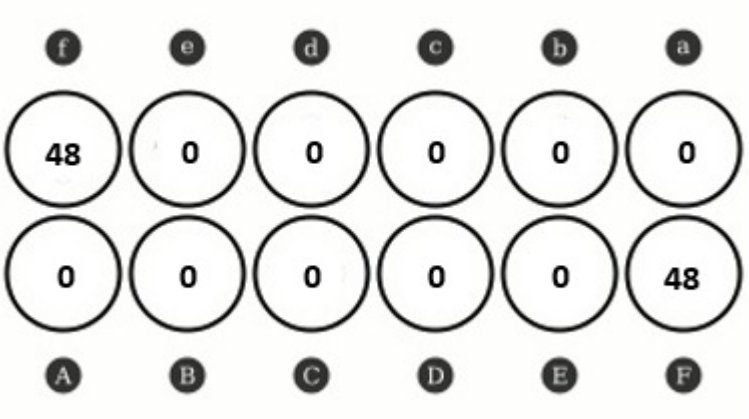

Figura 13: Problema f).

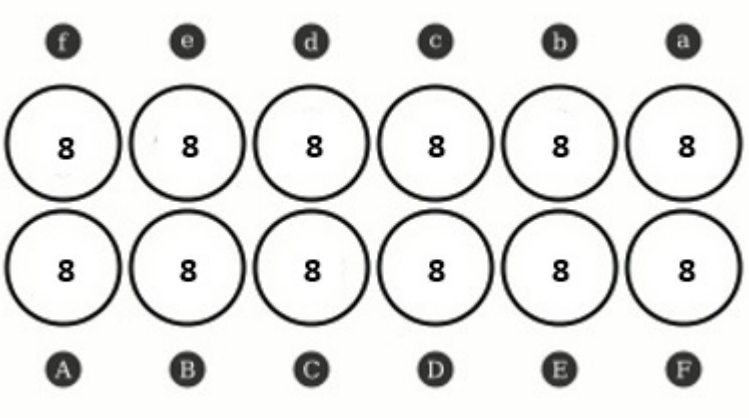

Figura 14: Problema g).