|  |
| --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA  **KHOA Công Nghệ Thông Tin** |

**ĐỀ THI VÀ BÀI LÀM**

Tên học phần: **Trí tuệ nhân tạo**

Mã học phần: Hình thức thi: *Tự luận có giám sát*

Đề số: **00002** Thời gian làm bài: 70 phút *(không kể thời gian chép/phát đề)*

Được sử dụng tài liệu khi làm bài.

**Họ tên:** Phạm Trung Hiếu **Lớp**:21TCLC\_NHAT1 **MSSV**: 102210034

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV\_HọTên.pdf và nộp bài thông qua MSTeam:

***Câu 1*** (*3 điểm*): Cho bài toán múc nước như sau:

* Cho *n* cái gáo nước, mỗi gáo *i* có thể chứa tối đa *ai* lít nước. Bạn cần múc đúng *M* lít nước từ bờ sông qua bể nước lớn với số thao tác ít nhất, không được múc quá cũng như múc thiếu. Biết, bạn không có bất kỳ dụng cụ nào khác để đo số lượng nước. Bạn cũng có thể vứt bỏ số nước đã múc nếu cần và việc vứt bỏ này **được tính là 1 thao tác**.

Hãy viết chương trình sử dụng thuật toán A\* nhập vào các số nguyên *n*, *M* và *a*1, *a*2,…,*an* và in ra cách thức múc nước. Nếu không có đáp án thì in “**Không có đáp án**”.

Ví dụ: - Nhập: 2 4 9 7

- Xuất:

* + Chuyển/Múc 9 lít nước từ **bờ sông** qua **gáo 1** (Gáo 1: 9 lít, Gáo 2: 0 lít, Bể: 0 lít)
  + Chuyển/Múc 7 lít nước từ **gáo 1** qua **gáo 2** (Gáo 1: 2 lít, Gáo 2: 7 lít, Bể: 0 lít)
  + Chuyển/Múc 2 lít nước từ **gáo 1** qua **bể** (Gáo 1: 0 lít, Gáo 2: 7 lít, Bể: 2 lít)
  + Vứt bỏ toàn bộ lít nước của **gáo 2**.(Gáo 1: 0 lít, Gáo 2: 0 lít, Bể: 2 lít)
  + Chuyển/Múc 9 lít nước từ **bờ sông** qua **gáo 1** (Gáo 1: 9 lít, Gáo 2: 0 lít, Bể: 2 lít)
  + Chuyển/Múc 7 lít nước từ **gáo 1** qua **gáo 2** (Gáo 1: 2 lít, Gáo 2: 7 lít, Bể: 2 lít)
  + Chuyển/Múc 2 lít nước từ **gáo 1** qua **bể** (Gáo 1: 0 lít, Gáo 2: 7 lít, **Bể: 4 lít**)

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới (1.5 điểm)  n = 3  M = 15  a = [7, 8, 9]  def init\_state():  state = []  for \_ in range(n + 3):  state.append(0)  return state  def check\_state(state):  return state[-3] == M  def check\_duplicate(state\_1, state\_2):  for i in range(n + 1):  if state\_1[i] != state\_2[i]:  return True  return False  def pour\_full\_water\_to\_ladle\_i\_from\_river(state, i):  finish\_state = state.copy()  global count  count += 1  finish\_state[-2] = count  finish\_state[-1] = state[-2]  if state[i] < a[i]:  finish\_state[i] = a[i]  return finish\_state  def make\_empty\_ladle\_i(state, i):  finish\_state = state.copy()  global count  count += 1  finish\_state[-2] = count  finish\_state[-1] = state[-2]  if state[i] > 0:  finish\_state[i] = 0  return finish\_state  def pour\_water\_from\_ladle\_i\_to\_ladle\_j(state, i, j):  finish\_state = state.copy()  global count  count += 1  finish\_state[-2] = count  finish\_state[-1] = state[-2]  if (state[j] < a[j]) and (state[i] > 0):  finish\_state[i] = max(0, state[i] - (a[j] - state[j]))  finish\_state[j] = min(a[j], state[i] + state[j])  return finish\_state  def pour\_water\_from\_ladle\_i\_to\_pool(state, i):  finish\_state = state.copy()  global count  count += 1  finish\_state[-2] = count  finish\_state[-1] = state[-2]  if (state[i] > 0) and (state[i] + state[-3] <= M):  finish\_state[i] = 0  finish\_state[-3] += state[i]  return finish\_state  def generate\_state(state):  childs = []  global count  for i in range(n):  child\_state = pour\_full\_water\_to\_ladle\_i\_from\_river(state, i)  if check\_duplicate(child\_state, state):  childs.append(child\_state)  for i in range(n):  child\_state = pour\_water\_from\_ladle\_i\_to\_pool(state, i)  if check\_duplicate(child\_state, state):  childs.append(child\_state)  for i in range(n):  child\_state = make\_empty\_ladle\_i(state, i)  if check\_duplicate(child\_state, state):  childs.append(child\_state)  for i in range(n):  for j in range(n):  if i != j:  child\_state = pour\_water\_from\_ladle\_i\_to\_ladle\_j(state, i, j)  if check\_duplicate(child\_state, state):  childs.append(child\_state)  return childs  def check\_not\_in(child, arr):  for value in arr:  if not check\_duplicate(child, value):  return False  return True  count = 0  init = init\_state()  open = [init]  close = []  while open:  state = open[0]  close.append(state)  open.pop(0)  if check\_state(state):  break  childs = generate\_state(state)  for child in childs:  if check\_not\_in(child, close):  if check\_not\_in(child, open):  open.append(child)  step = close[-1]  steps = [step]  while True:  for state in close:  if step[-1] == state[-2]:  step = state  steps.insert(0, step)  break  if step[-1] == 0:  break  for step in steps:  stage = ""  for i in range(n):  stage += "Gáo " + str(i + 1) + ": " + str(step[i]) + " lít, "  stage += "Bể: " + str(step[n])  print(stage)  **# Trả lời:** Dán kết quả thực thi với dữ liệu Nhập: “3 15 7 8 9” vào bên dưới (1 điểm)    **# Trả lời:** Hãy giải thích hàm *h’* (hàm khoảng cách trong thuật toán A\* ở chương trình trên. (0.5 điểm)  Trong đoạn mã trên:   * *Open* là danh sách chứa các trạng thái có thể duyệt tiếp. * *Close* là danh sách chứa các trạng thái đã duyệt qua. * *init\_state()* khởi tạo trạng thái ban đầu. * *check\_state(state)* kiểm tra xem trạng thái có phải trạng thái đích hay không. * *pour\_full\_water\_to\_ladle\_i\_from\_river, make\_empty\_ladle\_i, pour\_water\_from\_ladle\_i\_to\_ladle\_j, pour\_water\_from\_ladle\_i\_to\_pool* thực hiện các hành động trên trạng thái như đổ nước từ gáo sang bể, từ bể ra ngoài hay từ gáo này sang gáo khác. * *generate\_state(state*) tạo các trạng thái mới từ trạng thái hiện tại dựa trên các hành động có thể thực hiện. * *check\_not\_in(child, arr*) kiểm tra xem trạng thái con đã tồn tại trong danh sách các trạng thái hay chưa.   Thuật toán sử dụng vòng lặp while để duyệt qua các trạng thái có thể từ open và tạo ra các trạng thái con, kiểm tra xem chúng đã được duyệt qua hay chưa, sau đó thêm vào open để xem xét tiếp theo. Khi đạt được trạng thái đích *(check\_state(state)),* thuật toán dừng lại và trả về đường đi từ trạng thái cuối cùng ngược về trạng thái ban đầu.  Hàm heuristic đánh giá một ước lượng chi phí từ trạng thái hiện tại đến trạng thái mục tiêu. Nó không cần phải chính xác nhưng cần phải là một ước lượng tốt về chi phí còn lại để đi từ trạng thái hiện tại đến trạng thái mục tiêu.  Trong đoạn mã này, hàm heuristic không rõ ràng, chỉ sử dụng biến *count* để ước tính số lượng bước đi phù hợp, đồng thời cũng xét các bước khả thi trong việc múc nước để đạt được kết quả cuối cùng. |

***Câu 2*** (*4 điểm*): Cho tập dữ liệu [input.csv](https://drive.google.com/file/d/1BFZOngKU3CJrFu16MQWwDr1jfkXJMGYh/view?usp=sharing) với 90 mẫu dữ liệu, mỗi mẫu có 4 đặc trưng ( chiều dài đài hoa, chiều rộng đài hoa, chiều dài cánh hoa, chiều rộng cánh hoa) và tên loài hoa tương ứng.

1. *(3 điểm)* Hãy viết chương trình phân loại hoa sử dụng Logistic Regression kết hợp với lớp softmax. Nêu rõ mô hình thức phân loại trong chương trình như thế nào (Ví dụ: có bao nhiêu tế bào nơron, mỗi nơron phụ trách công việc gì, làm sao để phân loại,…)?

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới.  import csv  import json  import numpy as np  # Activation functions and their gradients  def linear(x):  return x  def linear\_grad(x):  return np.ones(x.shape)  def sigmoid(x):  return 1.0 / (1.0 + np.exp(-x))  def sigmoid\_grad(x):  return sigmoid(x) \* (1.0 - sigmoid(x))  def relu(x):  return np.maximum(0, x)  def relu\_grad(x):  return np.where(x <= 0, 0, 1)  # Loss functions  def mse(y\_hat, y):  return 1.0 / y.shape[0] \* 0.5 \* np.linalg.norm(y\_hat - y, 2) \*\* 2, y\_hat - y  def binary\_cross\_entropy(y\_hat, y):  return -np.sum(  y \* np.log(y\_hat) + (np.ones(y.shape) - y) \* np.log(np.ones(y.shape) - y\_hat)  ), -y / y\_hat + (1 - y) / (1 - y\_hat)  def categorical\_cross\_entropy(Y\_hat, Y):  losses = [-np.sum(y \* np.log(y\_hat)) for y\_hat, y in zip(Y\_hat, Y)]  return np.sum(losses), Y\_hat - Y  grad\_f = {  linear: linear\_grad,  sigmoid: sigmoid\_grad,  relu: relu\_grad,  }  # Optimizer class  class Optimizer:  container = {}  def \_\_init\_\_(self, learning\_rate):  self.learning\_rate = learning\_rate  def optimize(self, module, grad, hash\_id="random"):  pass  def container\_dict(self):  container = self.container.copy()  for key, value in container.items():  if isinstance(value, np.ndarray):  container[key] = value.tolist()  return container  def load\_container(self, container\_dict):  self.container = container\_dict  for key, value in self.container.items():  try:  self.container[key] = np.array(value)  except:  pass  # Module class  class Module:  optimizer = Optimizer  def forward(self, x):  pass  def backward(self, loss\_grad):  pass  def weights(self):  pass  def save(self, checkpoint):  data = {"weights": self.weights(), "optimizer": self.optimizer.container\_dict()}  with open(checkpoint, "w") as f:  f.write(json.dumps(data))  def load\_weights(self, weights\_dict):  pass  def load\_optimizer(self, optimizer\_dict):  self.optimizer.load\_container(optimizer\_dict)  def load(self, checkpoint):  self.load\_weights(checkpoint["weights"])  self.load\_optimizer(checkpoint["optimizer"])  # GD Optimizer  class GD(Optimizer):  def \_\_init\_\_(self, learning\_rate=0.001, momentum=0.5):  super().\_\_init\_\_(learning\_rate)  self.momentum = momentum  self.container = {}  def optimize(self, module, grad, hash\_id="random"):  if hash\_id not in self.container:  v\_last = np.zeros(module.shape)  else:  v\_last = self.container.get(hash\_id)  v\_now = self.momentum \* v\_last + self.learning\_rate \* grad  module = module - v\_now  self.container[hash\_id] = v\_now  return module  # Dense layer  class Dense(Module):  def \_\_init\_\_(self, in\_dim, out\_dim, bias=False, activation=linear):  self.in\_dim = in\_dim  self.out\_dim = out\_dim  self.bias = bias  self.activation = activation  self.w = np.random.rand(out\_dim, in\_dim)  self.b = None  if bias:  self.b = np.random.rand(out\_dim, 1)  self.optimizer = None  def forward(self, x):  self.x = x  if self.bias:  x = np.c\_[x, np.ones(x.shape[0])]  W = self.w  if self.bias:  W = np.c\_[W, self.b]  self.z = W.dot(x.T)  self.a = self.activation(self.z)  self.grad\_a\_z = grad\_f[self.activation](self.z)  return self.a.T  def backward(self, loss\_grad):  e = np.array(  [  grad \* a\_z\_grad if grad.shape == a\_z\_grad.shape else grad.dot(a\_z\_grad)  for grad, a\_z\_grad in zip(loss\_grad, self.grad\_a\_z.T)  ]  )  e = e.reshape(loss\_grad.shape[0], loss\_grad.shape[1]).T  w\_grad = e.dot(self.x)  if self.bias:  b\_grad = np.sum(e, axis=1, keepdims=True)  self.b = self.optimizer.optimize(self.b, b\_grad, str(hash(self)) + "\_b")  self.w = self.optimizer.optimize(self.w, w\_grad, str(hash(self)) + "\_w")  return e.T.dot(self.w)  def weights(self):  return self.w.tolist()  def load\_weights(self, weights\_dict):  self.w = np.array(weights\_dict)  # Softmax layer  class Softmax(Module):  def \_\_init\_\_(self):  self.eps = 1e-8  def forward(self, x):  e\_x = np.exp(x - np.max(x, axis=1, keepdims=True))  A = e\_x / e\_x.sum(axis=1, keepdims=True)  self.A = A  return A - self.eps  def backward(self, loss\_grad):  Ds = []  for id, grad in enumerate(loss\_grad):  Sz = self.A[id]  D = -np.outer(Sz[:], Sz[:]) + np.diag(Sz.flatten())  Ds.append(grad \* D)  return np.array(Ds)  # Model class  class Model(Module):  def \_\_init\_\_(self, layers: list[Module], optimizer):  self.layers = layers  self.optimizer = optimizer  for layer in self.layers:  layer.optimizer = self.optimizer  def forward(self, x):  for layer in self.layers:  x = layer.forward(x)  return x  def backward(self, loss\_grad):  for layer in reversed(self.layers):  loss\_grad = layer.backward(loss\_grad)  return loss\_grad  def weights(self):  return [layer.weights() for layer in self.layers]  def load\_weights(self, weights\_dict):  for id, weight in enumerate(weights\_dict):  self.layers[id].load\_weights(weight)  **# Trả lời:** Mô tả mô hình phân loại bằng hình ảnh hoặc bằng lời. |

1. *(1 điểm)* Hãy thực thi chương trình và cho biết nhãn của 60 mẫu dữ liệu trong [output.csv](https://drive.google.com/file/d/1rloJhWAzl8r3weUKmUKqgBMFtkHYW3WR/view?usp=sharing)

|  |
| --- |
| **# Trả lời:** Dán code thực thi thành công.  LR = 6e-4  CLASS\_MAPPING = {  "Iris-setosa": 0,  "Iris-versicolor": 1,  "Iris-virginica": 2,  }  def dataset():  X = []  Y = []  with open("input.csv") as f:  reader = csv.reader(f)  for row in reader:  x = list(map(float, row[:4]))  y = np.zeros(3)  y[CLASS\_MAPPING[row[-1]]] = 1  X.append(x)  Y.append(y)  return np.array(X), np.array(Y)  def test():  X = []  with open("output.csv") as f:  reader = csv.reader(f)  for row in reader:  x = list(map(float, row))  X.append(x)  return np.array(X)  X, Y = dataset()  model = Model(  [Dense(4, 3, bias=True, activation=sigmoid)], GD(learning\_rate=LR, momentum=0.1)  )  for i in range(1000):  Y\_hat = model.forward(X)  loss, grad = binary\_cross\_entropy(Y\_hat, Y)  Y\_hat = model.forward(X)  Y\_hat\_label = np.argmax(Y\_hat, axis=1)  Y\_label = np.argmax(Y, axis=1)  acc = np.sum(Y\_hat\_label == Y\_label)  print(f"Epoch {i + 1}: loss: {loss}. acc: {acc / X.shape[0]}")  model.backward(grad)  model.save("weights.txt")  test\_X = test()  predicted = model.forward(test\_X)  predicted = np.argmax(predicted, axis=1)  for idx in predicted:  key = list(filter(lambda x: CLASS\_MAPPING[x] == idx, CLASS\_MAPPING))[0]  print(key)  **# Trả lời:** Dán kết quả nhãn ứng với 60 mẫu dữ liệu.  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-setosa  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-virginica  Iris-versicolor  Iris-versicolor  Iris-versicolor  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica  Iris-virginica |

***Câu 3*** (3 *điểm*): Cho tập dữ liệu [Contries.csv](https://drive.google.com/file/d/1LdYTJ2aSl6y3zveaUArq7bCfEtwKtokt/view?usp=sharing), Hãy viết chương trình phân cụm bằng thuật toán *k*-means

1. *(1 điểm)* Viết hàm thực thi thuật toán *k*-means

|  |
| --- |
| **# Trả lời:** Dán code vào bên dưới:  import math  import random  def euclidean\_distance(point1, point2):  return math.sqrt(sum((p1 - p2) \*\* 2 for p1, p2 in zip(point1, point2)))  def assign\_points\_to\_clusters(centroids, data):  clusters = [[] for \_ in centroids]  for point in data:  shortest\_distance = float("inf")  for i, centroid in enumerate(centroids):  distance = euclidean\_distance(point, centroid)  if distance < shortest\_distance:  shortest\_distance = distance  closest\_centroid\_index = i  clusters[closest\_centroid\_index].append(point)  return clusters  def calculate\_centroids(clusters):  return [  tuple(  map(  lambda x: sum(x) / len(cluster\_points)  if len(cluster\_points) > 0  else 0,  zip(\*cluster\_points),  )  )  for cluster\_points in clusters  ]  def centroids\_have\_stabilized(old\_centroids, centroids):  return all(  euclidean\_distance(old, new) < 1e-6  for old, new in zip(old\_centroids, centroids)  )  def k\_means(data, k, max\_iterations=100):  centroids = random.sample(data, k)  for \_ in range(max\_iterations):  old\_centroids = centroids  clusters = assign\_points\_to\_clusters(centroids, data)  centroids = calculate\_centroids(clusters)  if centroids\_have\_stabilized(old\_centroids, centroids):  break  return clusters, centroids |

1. *(2 điểm)* Nếu sử dụng thuật toán *k*-means với k = 5 thì kết quả phân nhóm sẽ như thế nào? (Trọng tâm của các cụm, tỷ lệ phân cụm đúng, tiêu chí đánh giá việc phân cụm đúng là gì?).

|  |
| --- |
| **# Trả lời**: viết câu trả lời vào bên dưới  1. Trọng tâm của các cụm (in ra trọng tâm của 5 cụm):  2. Tỷ lệ phân cụm đúng (kết quả %):  3. Tiêu chí đánh giá việc phân cụm (viết bằng lời):  K-means là một thuật toán phổ biến trong việc phân cụm dữ liệu. Tiêu chí đánh giá phân cụm của K-means thường bao gồm:   * Độ đồng nhất trong cụm (Intra-cluster similarity): Khi dữ liệu trong một cụm gần nhau và cách xa các cụm khác, độ đồng nhất trong cụm là cao. Các điểm dữ liệu càng gần nhau, càng tốt cho độ đồng nhất này. * Độ phân tách giữa các cụm (Inter-cluster dissimilarity): Đây là khoảng cách giữa các cụm. Khoảng cách lớn giữa các cụm cho thấy sự phân tách tốt giữa chúng. * Số lượng cụm tối ưu (Optimal number of clusters): Việc chọn số lượng cụm phù hợp cũng là một tiêu chí quan trọng. Phân cụm tốt là khi số lượng cụm được chọn phản ánh đúng sự phân bố của dữ liệu. * Độ hội tụ của thuật toán (Convergence): K-means thực hiện các vòng lặp để cập nhật vị trí của các cụm. Việc thuật toán hội tụ nhanh chóng và không cần nhiều vòng lặp là điều mong muốn. * Khả năng xử lý dữ liệu nhiễu (Noise Robustness): Mức độ mà K-means có thể xử lý được dữ liệu nhiễu mà không ảnh hưởng đến kết quả phân cụm. * Độ phức tạp của thuật toán (Algorithmic Complexity): Thời gian chạy và độ phức tạp của thuật toán K-means cũng là một tiêu chí quan trọng trong việc đánh giá hiệu suất của nó.   Kết quả của K-means được đánh giá dựa trên sự kết hợp của các tiêu chí này để xác định xem liệu phân cụm có phản ánh đúng cấu trúc trong dữ liệu hay không. |

Đà Nẵng, ngày 14 tháng 12 năm 2023

|  |  |
| --- | --- |
| **GIẢNG VIÊN BIÊN SOẠN ĐỀ THI** | **TRƯỞNG BỘ MÔN** |
|  | (đã duyệt) |