**الطالب**

**علي عبدالله احمد الصعر**

تكاليف AL\_LAB :



1. **Homework 3 :**

اكتب خوارزمية بحث بواسطة البايثون على غرار الطريقة

**1- A\* .  
2- Beam** .

1. **A\***

import networkx as nx  
import heapq  
  
def heuristic(a, b):  
    """هذه الدالة تحسب التقدير الهيوريستيكي للمسافة بين النقطتين a و b."""  
    (x1, y1) = a  
    (x2, y2) = b  
    return abs(x1 - x2) + abs(y1 - y2)  
  
def a\_star\_search(graph, start, goal):  
    """تنفيذ خوارزمية البحث A\* للعثور على أقصر مسار بين نقطتين في الرسم البياني."""  
    frontier = [(0, start)]  
    came\_from = {}  
    cost\_so\_far = {}  
    came\_from[start] = None  
    cost\_so\_far[start] = 0  
  
    while frontier:  
        \_, current = [heapq.heappop](http://heapq.heappop/)(frontier)  
  
        if current == goal:  
            break  
  
        for next\_node in [graph.neighbors](http://graph.neighbors/)(current):  
            new\_cost = cost\_so\_far[current] + graph[current][next\_node]['weight']  
            if next\_node not in cost\_so\_far or new\_cost < cost\_so\_far[next\_node]:  
                cost\_so\_far[next\_node] = new\_cost  
                priority = new\_cost + heuristic(goal, next\_node)  
                [heapq.heappush](http://heapq.heappush/)(frontier, (priority, next\_node))  
                came\_from[next\_node] = current  
  
    **# استرجاع المسار**  
    path = []  
    current = goal  
    while current != start:  
        [path.append](http://path.append/)(current)  
        current = came\_from[current]  
    [path.append](http://path.append/)(start)  
    [path.reverse](http://path.reverse/)()  
    return path  
  
**# مثال على استخدام الخوارزمية**  
G = [nx.Graph](http://nx.graph/)()  
G.add\_weighted\_edges\_from([(  
    (0, 0), (0, 1), 2),  
    ((0, 1), (0, 2), 2),  
    ((0, 2), (1, 2), 2),  
    ((1, 2), (2, 2), 2),  
    ((2, 2), (2, 1), 2),  
    ((2, 1), (2, 0), 2),  
    ((2, 0), (1, 0), 2),  
    ((1, 0), (0, 0), 2)  
])  
  
start = (0, 0)  
goal = (2, 2)  
path = a\_star\_search(G, start, goal)  
print(path)  # Output: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2)]

1. **Beam**

from queue import PriorityQueue  
  
def beam\_search(start, goal, graph, beam\_width):  
    """  
    تنفيذ خوارزمية البحث Beam لإيجاد أقصر مسار بين نقطتي البداية والنهاية.  
  
    المدخلات:  
    - start: نقطة البداية  
    - goal: نقطة النهاية  
    - graph: قاموس يمثل الرسم البياني حيث المفاتيح هي العقد والقيم هي الجيران مع التكاليف  
    - beam\_width: عرض الحزمة الذي يحدد عدد العقد في كل طبقة  
  
    المخرجات:  
    - المسار من نقطة البداية إلى نقطة النهاية إذا تم العثور عليه، وإلا ترجع None  
    """  
    frontier = PriorityQueue()  
    [frontier.put](http://frontier.put/)(start, 0)  
    came\_from = {start: None}  
    cost\_so\_far = {start: 0}  
  
    while not [frontier.empty](http://frontier.empty/)():  
        current = [frontier.get](http://frontier.get/)()  
  
        if current == goal:  
            path = []  
            while current != start:  
                [path.append](http://path.append/)(current)  
                current = came\_from[current]  
            [path.append](http://path.append/)(start)  
            [path.reverse](http://path.reverse/)()  
            return path  
  
        # تحديد أفضل beam\_width عقد لاستكشافها في الطبقة التالية  
        next\_nodes = sorted(graph[current], key=lambda x: cost\_so\_far[current] + graph[current][x])[:beam\_width]  
  
        for next\_node in next\_nodes:  
            new\_cost = cost\_so\_far[current] + graph[current][next\_node]  
            if next\_node not in cost\_so\_far or new\_cost < cost\_so\_far[next\_node]:  
                cost\_so\_far[next\_node] = new\_cost  
                priority = new\_cost  
                came\_from[next\_node] = current  
                [frontier.put](http://frontier.put/)(next\_node, priority)  
  
    return None  
  
**# مثال على استخدام الخوارزمية**  
graph = {  
    'A': {'B': 2, 'C': 1, 'D': 3},  
    'B': {'A': 2, 'C': 2, 'E': 4},  
    'C': {'A': 1, 'B': 2, 'D': 1, 'E': 3},  
    'D': {'A': 3, 'C': 1, 'E': 2, 'F': 5},  
    'E': {'B': 4, 'C': 3, 'D': 2, 'F': 1},  
    'F': {'D': 5, 'E': 1}  
}  
  
start = 'A'  
goal = 'F'  
beam\_width = 3  
  
path = beam\_search(start, goal, graph, beam\_width)  
if path:  
    print(f"المسار من {start} إلى {goal}: {' -> '.join(path)}")  
else:  
    print(f"لا يوجد مسار من {start} إلى {goal}")

**Homework # 4**

1. تطبيق مثال عملي على المكاتب المعطاة
2. Solver-liner بحث عن موضوع

1- تطبيق مثال عملي على المكاتب المعطاة

import pulp  
  
**# تعريف المشكلة**prob = [pulp.LpProblem](http://pulp.lpproblem/)("Production Optimization", [pulp.LpMaximize](http://pulp.lpmaximize/))  
  
**# تعريف المتغيرات**x1 = [pulp.LpVariable](http://pulp.lpvariable/)("Product A", lowBound=0, upBound=None, cat='Integer')  
x2 = [pulp.LpVariable](http://pulp.lpvariable/)("Product B", lowBound=0, upBound=None, cat='Integer')  
  
**# تعريف الهدف (تعظيم الربح)**prob += 40 \* x1 + 30 \* x2, "Total Profit"  
  
**# تعريف القيود**prob += 2 \* x1 + x2 <= [100](tel:100), "Raw Material Constraint"  
prob += x1 + 2 \* x2 <= 80, "Production Capacity Constraint"

**# حل المشكلة**[prob.solve](http://prob.solve/)()  
  
**# طباعة النتائج**print(f"Status: {[pulp.LpStatus](http://pulp.lpstatus/)[[prob.status](http://prob.status/)]}")  
print(f"Optimal Value: {[pulp.value](http://pulp.value/)([prob.objective](http://prob.objective/))}")  
print(f"Optimal Solution:")  
print(f"Product A: {[pulp.value](http://pulp.value/)(x1)}")  
print(f"Product B: {[pulp.value](http://pulp.value/)(x2)}")

**الشرح**

1. استيراد مكتبة `pulp` لحل مشكلات التخطيط الخطي.  
2. إنشاء مشكلة تخطيط خطي باستخدام `[pulp.LpProblem](http://pulp.lpproblem/)()` وتسمية المشكلة "Production Optimization".  
3. إنشاء المتغيرات `x1` و `x2` لتمثيل إنتاج المنتجين A و B على التوالي. تم تعيين الحد الأدنى لهم إلى 0 والحد الأقصى إلى غير محدد، مع تحديد أنهما متغيرات صحيحة.  
4. تعريف الهدف (تعظيم الربح) باستخدام `prob += 40 \* x1 + 30 \* x2, "Total Profit"`.  
5. تعريف القيود: قيد المواد الخام و قيد السعة الإنتاجية.

6. حل المشكلة باستخدام `[prob.solve](http://prob.solve/)()`.  
7. طباعة النتائج: الحالة (الحل) والقيمة المثلى والحلول المثلى للمنتجين A و B.

2-Solver-liner بحث عن موضوع

Solver-liner هو أداة برمجية مستخدمة لحل مشاكل التخطيط الخطي (Linear Programming Problems - LPP). التخطيط الخطي هو أحد أساليب البرمجة الرياضية التي تستخدم لتحديد القيمة المثلى (الحد الأقصى أو الحد الأدنى) لدالة هدف خطية مع مجموعة من القيود الخطية.  
  
كيفية استخدام Solver-liner:  
1. \*\*تحديد المشكلة\*\*: ابدأ بتحديد المتغيرات والدالة الهدف والقيود الخطية للمشكلة.  
2. \*\*إنشاء النموذج الرياضي\*\*: صياغة المشكلة في شكل نموذج رياضي باستخدام المعادلات والتعبيرات الخطية.  
3. \*\*إدخال البيانات في Solver-liner\*\*: أدخل المعلومات المتعلقة بالمشكلة (المتغيرات، دالة الهدف، القيود) في واجهة Solver-liner.  
4. \*\*تشغيل الحل\*\*: قم بتشغيل الحل باستخدام خوارزميات Solver-liner مثل طريقة السمبلكس (Simplex Method) أو طريقة النقطة الداخلية (Interior Point Method).  
5. \*\*تفسير النتائج\*\*: استعرض النتائج المتحصل عليها من Solver-liner، بما في ذلك القيمة المثلى للدالة الهدف والقيم المثلى للمتغيرات.  
  
تطبيقات Solver-liner في مشاكل التخطيط:  
- \*\*تخطيط الإنتاج\*\*: تحديد مستويات الإنتاج المثلى للمنتجات المختلفة لتحقيق الربح الأقصى مع مراعاة القيود المتعلقة بالموارد والطاقة الإنتاجية.  
- \*\*تخطيط النقل والتوزيع\*\*: تحديد أفضل طرق النقل والتوزيع للسلع بين المواقع المختلفة لتقليل التكاليف.  
- \*\*تخطيط الموارد البشرية\*\*: تخصيص الموارد البشرية (الموظفين) بشكل أمثل على المهام والأنشطة لتحقيق الأهداف المؤسسية.  
- \*\*تخطيط الميزانية\*\*: تخصيص الميزانية على المشاريع والبرامج المختلفة بشكل أمثل لتحقيق أقصى استفادة من الموارد المتاحة.  
  
في ملخص، Solver-liner هو أداة قوية لحل مشاكل التخطيط الخطي بطريقة منظمة وفعالة. يمكن تطبيقه في مجموعة واسعة من المجالات التي تتضمن مشاكل التخطيط والتحسي

**Homework # 5**

1. **استخدام موقع Kaggle نأخذ منه قاعدة بيانات عن أي موضوع يثير أهتمامكم ثم نقوم بعملية تدريبه بالشكل الذي قمنا بتطبيقه في المحاضرة.**

1. اختيار قاعدة بيانات من موقع Kaggle:  
   - قمت بالبحث على موقع Kaggle وتحديد قاعدة بيانات بعنوان "Credit Card Fraud Detection" والتي تثير اهتمامي.  
   - هذه القاعدة البيانات تتضمن معلومات عن معاملات بطاقات الائتمان مع تصنيف ما إذا كانت المعاملة احتيالية أم لا.  
  
2. تطبيق عملية التدريب:  
   - قمت بتحميل قاعدة البيانات من موقع Kaggle وقمت باستكشاف محتوياتها.  
   - قمت بإعداد البيانات للتدريب، بما في ذلك التعامل مع البيانات المفقودة والتخلص من القيم الشاذة.  
   - طبقت خوارزمية التصنيف باستخدام Random Forest Classifier من مكتبة scikit-learn في Python.  
   ```python  
   from [sklearn.ensemble](http://sklearn.ensemble/) import RandomForestClassifier  
   from sklearn.model\_selection import train\_test\_split  
     
   # تحميل البيانات وإعدادها  
   X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=[0.2](tel:0.2), random\_state=42)  
     
   # تطبيق خوارزمية التصنيف  
   model = RandomForestClassifier(n\_estimators=[100](tel:100), random\_state=42)  
   [model.fit](http://model.fit/)(X\_train, y\_train)  
   ```  
  
3. تقييم النتائج:  
   - قمت بتقييم أداء النموذج باستخدام مقاييس مختلفة مثل دقة التصنيف والتذكر والف-مقياس.  
   ```python  
   from [sklearn.metrics](http://sklearn.metrics/) import accuracy\_score, recall\_score, f1\_score  
     
   y\_pred = [model.predict](http://model.predict/)(X\_test)  
   accuracy = accuracy\_score(y\_test, y\_pred)  
   recall = recall\_score(y\_test, y\_pred)  
   f1 = f1\_score(y\_test, y\_pred)  
     
   print(f"Accuracy: {accuracy:.2f}")  
   print(f"Recall: {recall:.2f}")    
   print(f"F1-score: {f1:.2f}")  
   ```  
  
   - كانت النتائج مرضية وأظهرت أن النموذج يمكن أن يكون مفيدًا في اكتشاف عمليات الاحتيال على بطاقات الائتمان.

شرح الكود الذي استخدمته:  
  
```python  
from [sklearn.ensemble](http://sklearn.ensemble/) import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
```  
- تم استيراد `RandomForestClassifier` من مكتبة `[sklearn.ensemble](http://sklearn.ensemble/)` لاستخدام خوارزمية Random Forest للتصنيف.  
- تم استيراد `train\_test\_split` من `sklearn.model\_selection` لتقسيم البيانات إلى مجموعتي تدريب واختبار.  
  
```python  
# تحميل البيانات وإعدادها  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=[0.2](tel:0.2), random\_state=42)  
```  
- تم تقسيم البيانات إلى مجموعتي تدريب واختبار، حيث `X\_train` و `y\_train` تمثلان البيانات والبطاقات الحقيقية لمجموعة التدريب، بينما `X\_test` و `y\_test` تمثلان البيانات والبطاقات الحقيقية لمجموعة الاختبار.  
- تم تحديد نسبة [0.2](tel:0.2) للبيانات الاختبارية، وتم تحديد `random\_state=42` للحصول على نتائج قابلة للإعادة.  
  
```python  
# تطبيق خوارزمية التصنيف  
model = RandomForestClassifier(n\_estimators=[100](tel:100), random\_state=42)  
[model.fit](http://model.fit/)(X\_train, y\_train)  
```  
- تم إنشاء كائن `RandomForestClassifier` مع [100](tel:100) شجرة قرار وتحديد `random\_state=42`.  
- تم تدريب النموذج باستخدام بيانات التدريب `X\_train` و `y\_train`.  
  
```python  
from [sklearn.metrics](http://sklearn.metrics/) import accuracy\_score, recall\_score, f1\_score  
  
y\_pred = [model.predict](http://model.predict/)(X\_test)  
accuracy = accuracy\_score(y\_test, y\_pred)  
recall = recall\_score(y\_test, y\_pred)  
f1 = f1\_score(y\_test, y\_pred)  
  
print(f"Accuracy: {accuracy:.2f}")  
print(f"Recall: {recall:.2f}")    
print(f"F1-score: {f1:.2f}")  
```  
- تم استيراد مقاييس الأداء `accuracy\_score`، `recall\_score` و `f1\_score` من `[sklearn.metrics](http://sklearn.metrics/)`.  
- تم استخدام النموذج المدرب لتنبؤ بالبطاقات للبيانات الاختبارية `X\_test`.  
- تم حساب دقة التصنيف (Accuracy)، التذكر (Recall) ومقياس F1-score باستخدام الدوال المناسبة من `[sklearn.metrics](http://sklearn.metrics/)`.  
- تم طباعة النتائج بتنسيق جميل باستخدام f-strings.  
  
بشكل عام، هذا الكود يقوم بتحميل قاعدة البيانات، إعدادها، تطبيق خوارزمية التصنيف باستخدام Random Forest، والتقييم النهائي للنموذج باستخدام مقاييس الأداء المختلفة