Expt. No. 1 -Implement DFID algorithm and compare its performance with DFS and BFS algorithm.

In [1]:
```
import time
from collections import deque

g = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['G'], 'F': [], 'G': []}
s, goal = 'A', 'G'

print("=== DFS ===")
t1 = time.time()
stack, v = [s], set()
while stack:
    n = stack.pop()
    if n in v: continue
    v.add(n)
    if n == goal: break
    stack.extend(reversed(g[n]))
print("Visited:", v, "Time:", time.time() - t1, "\n")

print("=== BFS ===")
t1 = time.time()
q, v2 = deque([s]), set()
while q:
    n = q.popleft()
    if n in v2: continue
    v2.add(n)
    if n == goal: break
    q.extend(g[n])
print("Visited:", v2, "Time:", time.time() - t1, "\n")

print("=== DFID ===")
t1 = time.time()
v3, found = set(), False
for d in range(len(g)):
    stack = [(s, 0)]
    while stack:
        n, l = stack.pop()
        if n in v3: continue
        v3.add(n)
        if n == goal:
            found = True
            break
        if l < d:
            stack.extend([(x, l+1) for x in reversed(g[n])])
    if found: break
print("Visited:", v3, "Time:", time.time() - t1)
```

```
=== DFS ===
Visited: {'E', 'B', 'A', 'D', 'G'} Time: 0.0

=== BFS ===
Visited: {'E', 'C', 'B', 'F', 'A', 'D', 'G'} Time: 0.0009899139404296875

=== DFID ===
Visited: {'A'} Time: 0.0
```

Expt. No. 2 - Implement Best-First Search algorithm.

In [1]:
```
graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['G'], 'F': [], 'G': []}
heuristic = {'A': 5, 'B': 4, 'C': 3, 'D': 6, 'E': 2, 'F': 6, 'G': 0}

visited = []
queue = ['A']
goal = 'G'

while queue:
    queue.sort(key=lambda x: heuristic[x])
    node = queue.pop(0)
    print("Visiting:", node)
    if node == goal:
        print("Goal reached:", node)
        break
    visited.append(node)
    for n in graph[node]:
        if n not in visited and n not in queue:
            queue.append(n)
```

```
Visiting: A
Visiting: C
Visiting: B
Visiting: E
Visiting: G
Goal reached: G
```

Expt. No. 3 - Implementation of AND/OR/NOT Gate using Single Layer Perceptron.

In [1]:
```python
def activation(x):
    return 1 if x >= 0 else 0

def and_gate():
    print("AND Gate")
    w1, w2, bias = 1, 1, -1.5
    for x1, x2 in [(0,0), (0,1), (1,0), (1,1)]:
        output = activation(x1*w1 + x2*w2 + bias)
        print(f"{x1} AND {x2} = {output}")

def or_gate():
    print("\nOR Gate")
    w1, w2, bias = 1, 1, -0.5
    for x1, x2 in [(0,0), (0,1), (1,0), (1,1)]:
        output = activation(x1*w1 + x2*w2 + bias)
        print(f"{x1} OR {x2} = {output}")

def not_gate():
    print("\nNOT Gate")
    w, bias = -1, 0.5
    for x in [0, 1]:
        output = activation(x*w + bias)
        print(f"NOT {x} = {output}")

# Run all gates
and_gate()
or_gate()
not_gate()
```

```
AND Gate
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

OR Gate
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

NOT Gate
NOT 0 = 1
NOT 1 = 0
```

Expt. No. 4 - Implementation of XOR Gate using:

(a) Multi-layer Perceptron/Error Back Propagation

In [1]:
```
import math

def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def xor(x1, x2):
    # Hidden layer
    h1 = sigmoid(x1 * 20 + x2 * 20 - 30)   # neuron 1
    h2 = sigmoid(x1 * 20 + x2 * 20 - 10)   # neuron 2

    # Output layer
    o = sigmoid(h1 * -60 + h2 * 60 - 30)

    return round(o)

# Test XOR gate
print("XOR Gate using simple MLP:")
for x1, x2 in [(0,0), (0,1), (1,0), (1,1)]:
    print(f"{x1} XOR {x2} = {xor(x1, x2)}")
```

```
XOR Gate using simple MLP:
0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0
```

b) Radial Basis Function Network

In [9]:
```
import numpy as np

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([0,1,1,0])

centers = np.array([[0,1],[1,0]])
spread = 1.0

def rbf(x, c, s): return np.exp(-np.linalg.norm(x-c)**2 / (2*s**2))

G = np.array([[rbf(x, c, spread) for c in centers] for x in X])
weights = np.linalg.pinv(G) @ y
output = np.where(G @ weights >= 0.5, 1, 0)

print("XOR Gate using Radial Basis Function Network:")
for i in range(4):
    print(f"{X[i][0]} XOR {X[i][1]} = {output[i]}")
```

```
XOR Gate using Radial Basis Function Network:
0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0
```

Expt. No. 5 - Implement Hebbian learning rule and Correlation learning rule.

In [3]:
```python
import numpy as np

# Input and output vectors
X = np.array([[1, -1], [-1, 1]])   # 2 samples, 2 features each
Y = np.array([[1], [-1]])          # Corresponding targets

# Hebbian Learning Rule: W = Σ(x.T * y)
W_hebb = np.zeros((2, 1))
for i in range(len(X)):
    W_hebb += X[i].reshape(2,1) * Y[i]
print("Hebbian Weights:\n", W_hebb)

# Correlation Learning Rule: W = Σ(x.T * y), assuming input is binary/unipolar
W_corr = np.zeros((2, 1))
for i in range(len(X)):
    W_corr += np.outer(X[i], Y[i])
print("Correlation Weights:\n", W_corr)
```

```
Hebbian Weights:
 [[ 2.]
 [-2.]]
Correlation Weights:
 [[ 2.]
 [-2.]]
```

Expt. 6 - Implement Find-S and candidate elimination algorithms.

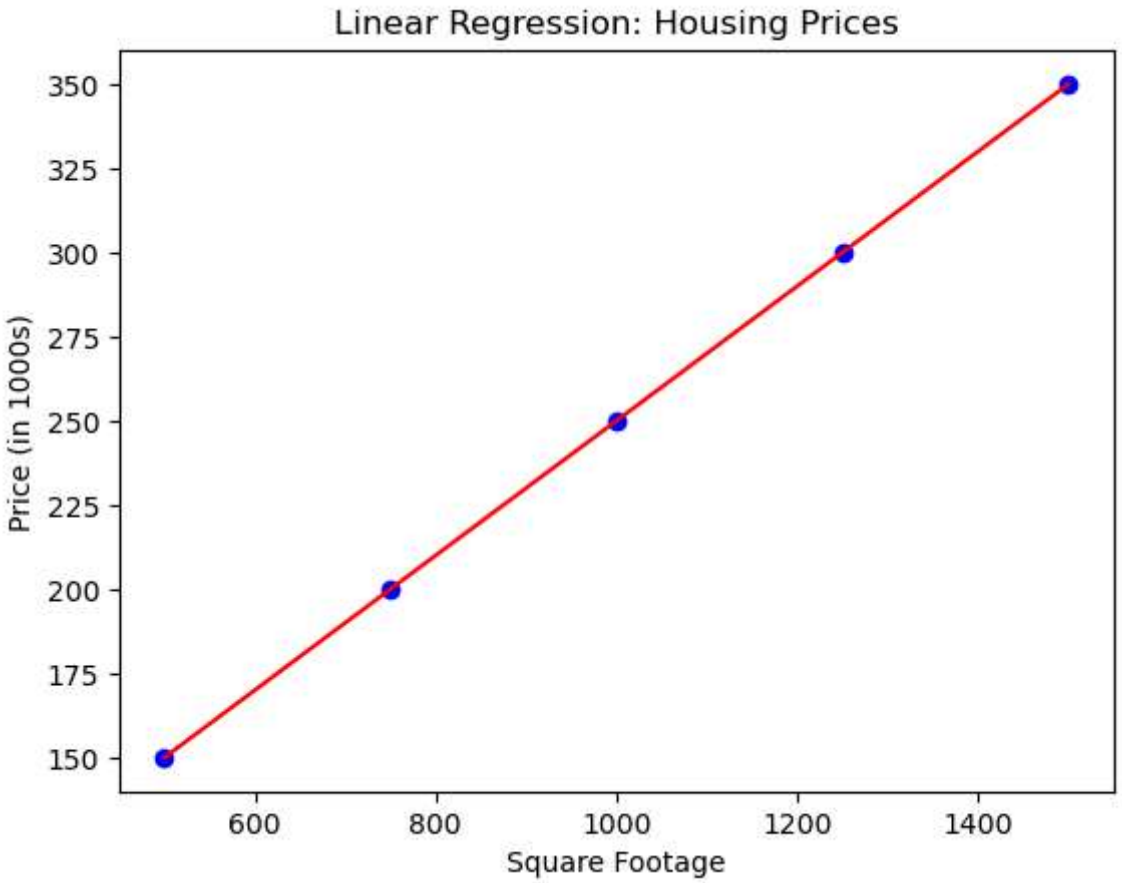
In [3]:
```
data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
]

n=len(data[0])-1
s=['Ø']*n
for x in data:
    if x[-1]=='Yes':
        for i in range(n):
            if s[i]=='Ø':s[i]=x[i]
            elif s[i]!=x[i]:s[i]='?'
print("Find-S:",s)
g=[['?' for _ in range(n)]]
for x in data:
    if x[-1]=='No':
        new_g=[]
        for h in g:
            for i in range(n):
                if h[i]=='?' and s[i]!='?':
                    h1=h[:];h1[i]=s[i]
                    if all(h1[j]==s[j] or h1[j]=='?' for j in range(n)):new_g.append(h1)
        g=new_g
print("Candidate Elimination G:",g)
print("Candidate Elimination S:",s)
```

```
Find-S: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
Candidate Elimination G: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?',
'?'], ['?', '?', '?', 'Strong', '?', '?']]
Candidate Elimination S: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Expt. 7 - Build a linear regression model housing prices.

In [7]:
```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

X = np.array([[500], [750], [1000], [1250], [1500]])
y = np.array([150, 200, 250, 300, 350])

model = LinearRegression()
model.fit(X, y)

print("Slope:", model.coef_[0])
print("Intercept:", model.intercept_)
print("Predicted price for 1100 sqft:", model.predict([[1100]])[0])

plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X), color='red')
plt.xlabel('Square Footage')
plt.ylabel('Price (in 1000s)')
plt.title('Linear Regression: Housing Prices')
plt.show()
```

```
Slope: 0.20000000000000007
Intercept: 49.99999999999994
Predicted price for 1100 sqft: 270.0
```



Expt. 8 - Implement spam detection using Naïve Bayes Algorithm.

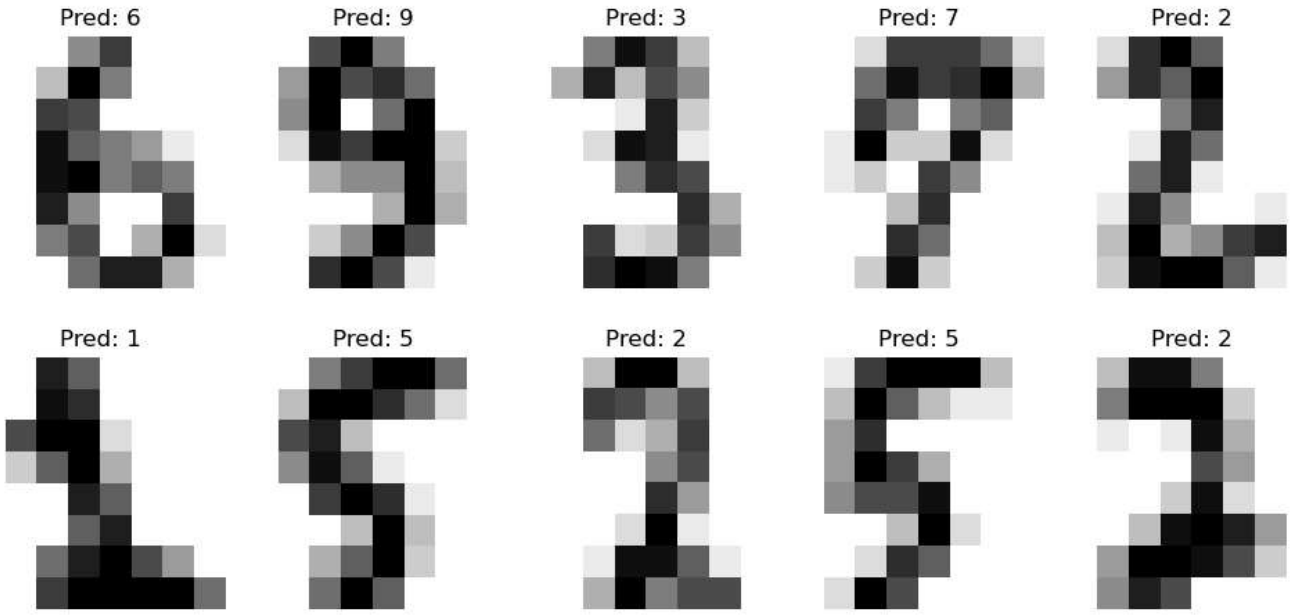
In [3]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

data = ['Free money now!!!',
        'Hi, how are you?',
        'Win cash prizes',
        'Hello friend',
        'Cheap meds available',
        'Will meet tomorrow']
labels = [1, 0, 1, 0, 1, 0]  # 1 = spam, 0 = not spam

cv = CountVectorizer()
X = cv.fit_transform(data)
model = MultinomialNB()
model.fit(X, labels)

test = cv.transform(['Win a free ticket', 'Are you coming to the party?'])
print(model.predict(test))
```

```
[1 0]
```

Expt. 9 - Implement hand writing classification using Support Vector Machines.

In [39]:
```
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()
X_train, X_test, y_train, y_test = train_test_split(
    digits.images.reshape(len(digits.images), -1),
    digits.target, test_size=0.3, random_state=42)

clf = svm.SVC(gamma=0.001)
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)

print("Accuracy:", metrics.accuracy_score(y_test, predicted))

fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for ax, image, prediction in zip(axes.ravel(), X_test.reshape(-1, 8, 8), predicted):
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='none')  # sharper pixels
    ax.set_title(f'Pred: {prediction}')
    ax.axis('off')
plt.tight_layout()
plt.show()
```

```
Accuracy: 0.9907407407407407
```



Expt. 10 - Implement FP-tree for finding co-occurring words in a twitter feed.

In [1]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter
from itertools import combinations
import string

nltk.download('punkt')
nltk.download('stopwords')

# Sample tweets
tweets = [
    "I love AI and machine learning!",
    "Machine learning is fascinating.",
    "AI will change the world.",
    "Deep learning is a subset of machine learning.",
    "I love exploring data science and AI.",
]

# Preprocessing function
def preprocess(tweet):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(tweet.lower())
    words = [word for word in words if word.isalpha() and word not in stop_words]
    return words

# Preprocess all tweets
transactions = [preprocess(tweet) for tweet in tweets]

# Count word pairs (co-occurring in same tweet)
pair_counter = Counter()
min_support = 2

for words in transactions:
    unique_words = set(words)
    pairs = combinations(sorted(unique_words), 2)
    pair_counter.update(pairs)

# Filter pairs by min support
frequent_pairs = {pair: count for pair, count in pair_counter.items() if count >= min_support}

# Output
print("Frequent co-occurring word pairs:")
for pair, count in frequent_pairs.items():
    print(f"{pair}: {count}")
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\vtu\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\vtu\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
Frequent co-occurring word pairs:
('ai', 'love'): 2
('learning', 'machine'): 3
```