**Problem 2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

|  |
| --- |
| **import** **random**  **import** **csv** |

|  |
| --- |
| **def** g\_0(n):  **return** ("?",)\*n  **def** s\_0(n):  **return** ('0',)\*n |

|  |
| --- |
| **def** more\_general(h1, h2):  more\_general\_parts = []  **for** x, y **in** zip(h1, h2):  mg = x == "?" **or** (x != "0" **and** (x == y **or** y == "0"))  more\_general\_parts.append(mg)  **return** all(more\_general\_parts)  l1 = [1, 2, 3]  l2 = [3, 4, 5]  list(zip(l1, l2)) |

|  |
| --- |
| **def** fulfills(example, hypothesis):  *### the implementation is the same as for hypotheses:*  **return** more\_general(hypothesis, example)  **def** min\_generalizations(h, x):  h\_new = list(h)  **for** i **in** range(len(h)):  **if** **not** fulfills(x[i:i+1], h[i:i+1]):  h\_new[i] = '?' **if** h[i] != '0' **else** x[i]  **return** [tuple(h\_new)] |

|  |
| --- |
| min\_generalizations(h=('0', '0' , 'sunny'),  x=('rainy', 'windy', 'cloudy')) |

|  |
| --- |
| **def** min\_specializations(h, domains, x):  results = []  **for** i **in** range(len(h)):  **if** h[i] == "?":  **for** val **in** domains[i]:  **if** x[i] != val:  h\_new = h[:i] + (val,) + h[i+1:]  results.append(h\_new)  **elif** h[i] != "0":  h\_new = h[:i] + ('0',) + h[i+1:]  results.append(h\_new)  **return** results |

|  |
| --- |
| min\_specializations(h=('?', 'x',),  domains=[['a', 'b', 'c'], ['x', 'y']],  x=('b', 'x')) |

|  |
| --- |
| **with** open('C:**\\**Users**\\**Desktop**\\**c1.csv') **as** csvFile:  examples = [tuple(line) **for** line **in** csv.reader(csvFile)]  examples |

|  |
| --- |
| **def** get\_domains(examples):  d = [set() **for** i **in** examples[0]]  **for** x **in** examples:  **for** i, xi **in** enumerate(x):  d[i].add(xi)  **return** [list(sorted(x)) **for** x **in** d]  get\_domains(examples) |

|  |
| --- |
| **def** candidate\_elimination(examples):  domains = get\_domains(examples)[:-1]    G = set([g\_0(len(domains))])  S = set([s\_0(len(domains))])  i=0  print("**\n** G[**{0}**]:".format(i),G)  print("**\n** S[**{0}**]:".format(i),S)  **for** xcx **in** examples:  i=i+1  x, cx = xcx[:-1], xcx[-1]  **if** cx=='Y':  G = {g **for** g **in** G **if** fulfills(x, g)}  S = generalize\_S(x, G, S)  **else**: *# x is negative example*  S = {s **for** s **in** S **if** **not** fulfills(x, s)}  G = specialize\_G(x, domains, G, S)  print("**\n** G[**{0}**]:".format(i),G)  print("**\n** S[**{0}**]:".format(i),S)  **return** |

|  |
| --- |
| **def** generalize\_S(x, G, S):  S\_prev = list(S)  **for** s **in** S\_prev:  **if** s **not** **in** S:  **continue**  **if** **not** fulfills(x, s):  S.remove(s)  Splus = min\_generalizations(s, x)  S.update([h **for** h **in** Splus **if** any([more\_general(g,h)  **for** g **in** G])])  S.difference\_update([h **for** h **in** S **if**  any([more\_general(h, h1)  **for** h1 **in** S **if** h != h1])])  **return** S |

|  |
| --- |
| **def** specialize\_G(x, domains, G, S):  G\_prev = list(G)  **for** g **in** G\_prev:  **if** g **not** **in** G:  **continue**  **if** fulfills(x, g):  G.remove(g)  Gminus = min\_specializations(g, domains, x)  G.update([h **for** h **in** Gminus **if** any([more\_general(h, s)  **for** s **in** S])])  G.difference\_update([h **for** h **in** G **if**  any([more\_general(g1, h)  **for** g1 **in** G **if** h != g1])])  **return** G |

|  |
| --- |
| candidate\_elimination(examples) |

**OUTPUT**

|  |
| --- |
| G[0]: {('?', '?', '?', '?', '?', '?')}  S[0]: {('0', '0', '0', '0', '0', '0')}  G[1]: {('?', '?', '?', '?', '?', '?')}  S[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}  G[2]: {('?', '?', '?', '?', '?', '?')}  S[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}  G[3]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same')}  S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}  G[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}  S[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}  In [ ]: |

**Note: save the file c1.csv on desktop in your folder and change the path of file name in open() function in the program code**