**학습활동보고서 # 7**

|  |  |  |  |
| --- | --- | --- | --- |
| 대학/학부/학과 | 엘텍공과대학 | 전공 | 컴퓨터공학과 |
| 학번 | 1871001 | 이름 | ZHU ZHAOLING |

|  |  |
| --- | --- |
| 학습활동 주제 및 목표 | 외판원 문제 |

(필요한 항목 및 내용을 자유로운 양식으로 작성하세요)

1. 학습활동 –내용을 상세히 작성합니다.

**7.1 해밀턴 경로와 외판원 문제**

▪ 해밀턴 경로 문제: The Hamiltonian Path Problem

• 주어진 그래프: 연결된 무방향 그래프

• 한 정점에서 출발하여 그래프의 모든 정점을 한 번씩만 방문하고

⁃ 다시 원래 출발한 정점으로 되돌아오는 경로

• cf) 오일러 경로 (Eulerian Path): 모든 간선을 한 번씩만 방문. 한 붓 그리기 문제.

**▪ 해밀턴 경로: 백트래킹으로 풀기**

• 상태공간트리의 구축

⁃ 트리의 레벨 0에서 출발 정점을 선정(경로의 0번째 정점)

⁃ 트리의 레벨 1에서 출발 정점을 제외한 각 정점을 출발 정점 다음에 올 첫째 정점으로 고려한다.

⁃ 트리의 레벨 2에서는 앞에서와 같은 정점들을 각각 둘째 정점으로 고려한다.

⁃ 그 이후 수준에서도 같은 방식으로 계속

⁃ 마지막으로, 수준째 정점으로 고려한다.

**Algorithm7.1:Backtracking for the Hamiltonian Circuits Problem**

1. **def** hamiltonian (i, W, vindex):
2. n = len(W) - 1
3. **if** (promising(i, W, vindex)):
4. **if** (i == (n - 1)):
5. **print**(vindex[0:n])
6. **else**:
7. **for** j **in** range(2, n + 1):
8. vindex[i + 1] = j
9. hamiltonian(i + 1, W, vindex)
11. **def** promising (i, W, vindex):
12. flag = True
13. **if** ((i == (n - 1)) **and** **not** W[vindex[n-1]][vindex[0]]):
14. flag = False
15. **elif** ((i > 0) **and** **not** W[vindex[i-1]][vindex[i]]):
16. flag = False
17. **else**:
18. j = 1
19. **while** (j < i **and** flag):
20. **if** (vindex[i] == vindex[j]):
21. flag = False
22. j += 1
23. **return** flag
25. n = 4
26. edges = [
27. [1, 2],
28. [1, 3],
29. [1, 4],
30. [2, 3],
31. [2, 4],
32. [3, 4]
33. ]
34. W = [[False] \* (n + 1) **for** \_ **in** range(n + 1)]
35. **for** e **in** edges:
36. W[e[0]][e[1]] = W[e[1]][e[0]] = True
37. vindex = [-1] \* (n + 1)
38. vindex[0] = 1
39. hamiltonian(0, W, vindex)

**▪ 외판원 문제: (TSP: The Traveling Salesperson Problem)**

• 외판원 문제의 이해:

⁃ 외판원이 20개 도시로 판매 출장을 계획하고 있다고 가정하자.

⁃ 외판원은 출발 도시에서 모든 도시를 방문하고 다시 되돌아와야 한다.

⁃ 출장시간(또는 비용)을 최소로 줄이기 위한 방문 계획을 구하고 싶다.

• 주어진 그래프는 가중치 있는 방향 그래프라고 가정

• 가중치 있는 그래프에서 해밀턴 경로 문제의 최적화 문제

**7.2 외판원 문제와 동적 계획법**

▪ 외판원 문제: 동적 계획법으로 풀기

•

•

•

• 에 속한 도시를 각각 한 번씩만 거쳐서

**▪**

def isIn (i, A):

if ((A & (1 << (i - 2))) != 0):

return True

else:

return False

**▪**

def diff (A, j):

t = 1 << (j - 2)

return (A & (~t))

def count (A, n):

count = 0

for i in range(n):

if ((A & (1 << i)) != 0):

count += 1

return count

**Algorithm 7.2: Dynamic Programming for the TSP**

1. **def** travel (W):
2. n = len(W) - 1
3. size = 2 \*\* (n - 1)
4. D = [[0] \* size **for** \_ **in** range(n + 1)]
5. P = [[0] \* size **for** \_ **in** range(n + 1)]
6. **for** i **in** range(2, n + 1):
7. D[i][0] = W[i][1]
8. **for** k **in** range(1, n - 1):
9. **for** A **in** range(1, size):
10. **if** (count(A, n) == k):
11. **for** i **in** range(2, n + 1):
12. **if** (**not** isIn(i, A)):
13. D[i][A], P[i][A] = minimum(W, D, i, A)
14. A = size - 1
15. D[1][A], P[1][A] = minimum(W, D, 1, A)
16. **return** D, P
18. **def** minimum (W, D, i, A):
19. minValue = INF
20. minJ = 1
21. n = len(W) - 1
22. **for** j **in** range(2, n + 1):
23. **if** (isIn(j, A)):
24. m = W[i][j] + D[j][diff(A, j)]
25. **if** (minValue > m):
26. minValue = m
27. minJ = j
28. **return** minValue, minJ
30. INF = 999
31. W = [
32. [-1, -1, -1, -1, -1],
33. [-1, 0, 2, 9, INF],
34. [-1, 1, 0, 6, 4],
35. [-1, INF, 7, 0, 8],
36. [-1, 6, 3, INF, 0]
37. ]
39. D, P = travel(W)
40. **print**('D =')
41. **for** i **in** range(1, len(D)):
42. **print**(D[i])
43. **print**('P =')
44. **for** i **in** range(1, len(P)):
45. **print**(P[i])

**7.3 외판원 문제와 분기 한정법**

**Algorithm 6.3: Best-First-Search with Branch-and-Bound for the TSP**

1. **class** SSTNode:
2. **def** \_\_init\_\_ (self, level):
3. self.level = level
4. self.path = []
5. self.bound = 0
6. **def** contains(self, x):
7. **for** i **in** range(len(self.path)):
8. **if** (self.path[i] == x):
9. **return** True
10. **return** False
11. **def** travel2 (W):
12. **global** minlength, opttour
13. n = len(W) - 1
14. PQ = PriorityQueue()
15. v = SSTNode(0)
16. v.path = [1]
17. v.bound = bound(v, W)
18. minlength = INF
19. PQ.put((v.bound, v))
21. **while** (**not** PQ.empty()):
22. v = PQ.get()[1]
23. **if** (v.bound < minlength):
24. **for** i **in** range(2, n + 1):
25. **if** (v.contains(i)):
26. **continue**
27. u = SSTNode(v.level + 1)
28. u.path = v.path[:]
29. u.path.append(i)
31. **if** (u.level == n - 2):
32. **for** k **in** range(2, n + 1):
33. **if** (**not** u.contains(k)):
34. u.path.append(k)
35. u.path.append(1)
36. **if** (length(u.path, W) < minlength):
37. minlength = length(u.path, W)
38. opttour = u.path[:]
39. **else**:
40. u.bound = bound(u, W)
41. **if** (u.bound < minlength):
42. PQ.put((u.bound, u))
44. **def** bound(v, W):
45. n = len(W) - 1
46. total = length(v.path, W)
47. **for** i **in** range(1, n + 1):
48. **if** (hasOutgoing(i, v.path)):
49. **continue**
50. min = INF
51. **for** j **in** range(1, n + 1):
52. **if** (i == j): **continue**
53. **if** (hasIncoming(j, v.path)): **continue**
54. **if** (j == 1 **and** i == v.path[len(v.path) - 1]): **continue**
55. **if** (min > W[i][j]): min = W[i][j]
56. total += min
57. **return** total
59. **def** length(path, W):
60. total = 0
61. prev = 1
62. **for** i **in** range(len(path)):
63. **if** (i != 0):
64. prev = path[i - 1]
65. total += W[prev][path[i]]
66. prev = path[i]
67. **return** total
68. **def** hasOutgoing(v, path):
69. **for** i **in** range(0, len(path) - 1):
70. **if** (path[i] == v):
71. **return** True
72. **return** False
73. **def** hasIncoming(v, path):
74. **for** i **in** range(1, len(path)):
75. **if** (path[i] == v):
76. **return** True
77. **return** False
79. INF = 999
80. W = [
81. [-1, -1, -1, -1, -1, -1],
82. [-1, 0, 14, 4, 10, 20],
83. [-1, 14, 0, 7, 8, 7],
84. [-1, 4, 5, 0, 7, 16],
85. [-1, 11, 7, 9, 0, 2],
86. [-1, 18, 7, 17, 4, 0],
87. ]
88. minlength = INF
89. opttour = None
90. travel2(W)
92. **print**("minlength =", minlength)
93. **print**("optimal tour =", opttour)