알고리즘분석

**HW#3 Report**

2011147068 김정환

2011147115 허재화

2012147562 최인호

1. **목차**
   1. **문제 분석**
   2. **알고리즘**
   3. **구현**
      1. **Class 설명**
      2. **알고리즘에 대한 세부구현**
   4. **토의 과정**
      1. **Big-O notation을 통한 분석**
      2. **구현 과정에서 겪은 어려움**

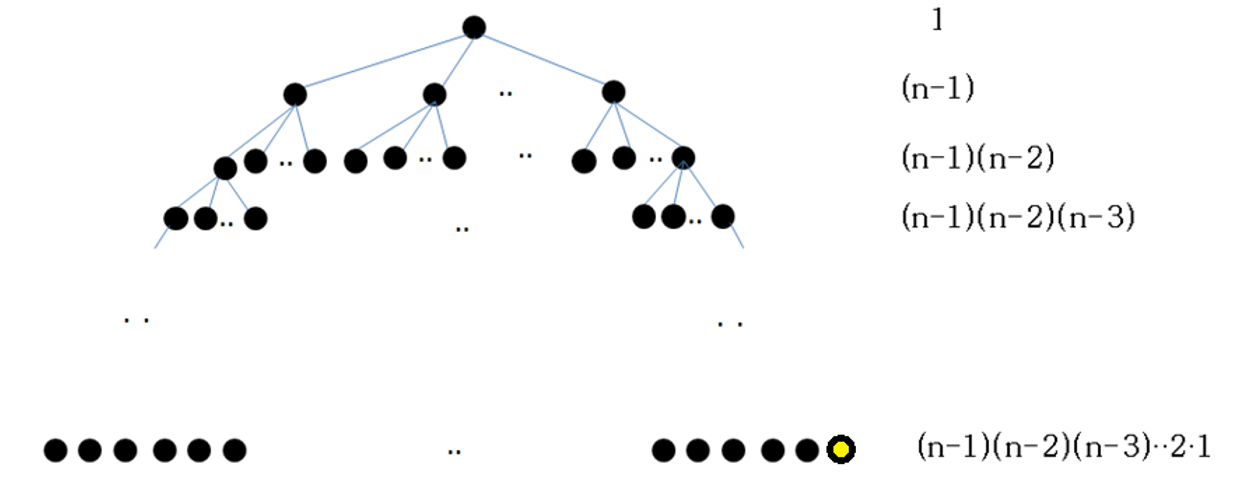
**문제 분석**

* 여기를
  + 채워 주세용

1. **알고리즘**

**알고리즘 : Backtracking**

Backtracking은 기본적으로 State Space Tree를 이용해 가능한 모든 경우를 검사하여 최적해를 찾는 알고리즘이다. TSP의 경우 시작 vertex를 root로하여 가능한 모든 경로들을 트리로 찾고, 각각의 경우에 대해 경로의 값을 구하여 비교한다.



TSP를 수행했을 때, State Space Tree는 위의 그림과 같이 형성된다. 그래서 실행시간은 vertex수 n에대하여 exponential하게 걸린다.

이 알고리즘을 이용하여 TSP문제는 푼 과정은 다음과 같다.

State Space Tree의 각 노드는, 각각의 tour와 curDist, 그리고 vlist를 갖는다. tour는 그 시점에서 TSP가 지나온 vertex들을 담고있다. 그리고 curDist는 그 tour경로의 모든 edge값의 합이다. vlist는 그래프의 vertex들을 저장하고 있는 list로서 각 vertex의 flag데이터를 이용하여 특정 vertex가 이미 도달한 vertex인지 검사하는데 사용한다. 우리는 Backtracking기법은 재귀적으로 구현하였다. 그래서 매번 BacktrackTSP가 실행될 때마다 현재 tour가 그래프의 총 vertex개수만큼의 크기를 갖는지 검사한다. 그렇다면 그 경로는 complete한 하나의 경로이므로 그 경로의 길이를 구해서, 이전까지 나온 경로의 최소값과 비교하여 값과 최종경로를 갱신한다. 현재 tour가 그래프의 총 vertex개수만큼의 크기를 갖지 못한다면, 아직 그 tour가 도달하지 못한 vertex들을 하나씩 전부 추가하여 새로운 subproblem을 만들고, 그 subproblem들은 State Space Tree상에서 그 노드의 자식노드들이 된다. 그 자식노드들의 curDist값은 부모노드의 curDist값에다가 새로운 vertex를 추가하기 위해 이어지는 edge의 값을 더한값이 된다. 그리고 Backtracking의 구현 과정에서 우리는 재귀함수를 사용하며, DFS(Depth-First-Search)방식을 사용한다. 이렇게 거의 모든 State Space Tree를 탐색하지만, 만약에 Tree를 형성하는 중간과정에서의 curDist값이 지금까지 구해진 TSP경로의 최소값보다 크다면, 그 경로로부터는 subproblem을 더 만들지 않고, Tree상에서 더 이상 자식노드를 형성하지 않는다. 그 이유는 문제에서 주어지는 그래프는 각 edge의 값이 0보다 크며, 따라서 남은 vertex를 추가하는 과정에서 dist값이 더 작아지는 경우는 기대할 수 없기 때문이다. 이렇게 모든 State Space Tree를 탐색하고 나면 최적의 TSP경로와 그 값이 구해진다.

**알고리즘 : Branch And Bound**

BranchAndBound는 BackTracking 알고리즘과 같이 State Space Tree를 구현하여 문제를 해결한다. 따라서 최악의 경우에 Backtracking과 마찬가지로 vertex수 n에 대하여 exponential한 시간이 걸린다. 하지만 실제로는 좋은 설계에 의해 전형적인 경우에 대하여 매우 효율적인 시간에 최적해를 구할 수 있다. 이 알고리즘은 보통 최적해 구하기(Optimization problem)에 적용될 수 있다. Backtracking기법과의 차이점은, 각 subproblem을 구할 때마다, 그 subproblem이 유망한지 여부를 결정하기 위해 Bound를 계산하고, 그 Bound값이 지금까지 찾은 최적해보다 좋지 않으면 State Space Tree의 그쪽 가지를 잘라내는 방식을 사용한다는 점이다. 여기서 Bound란, 그 subproblem으로부터 가지를 뻗어나갔을 때(branch) 얻을 수 있는 최적해의 한계를 의미한다. 그리고 Backtracking에서 BFS나 DFS를 사용했던 것과는 달리 Branch And Bound에서는 lowerBound에 대한 Best-First-Search를 사용한다. Best-First-Search란 State Space Tree에서 다음 진행할 subproblem을 선택하는데 있어서 lowerBound값이 가장 유리한(최소값 찾는 문제에선 가장 작은)것을 선택해나가는 것이다. 그리고 complete path가 구해져서 어떤 TSP경로값이 구해졌을 때, 그 값보다 큰 lowerBound를 갖는 subproblem들은 Tree에서 더 이상 subproblem을 만들지 않는다. 이게 가지치기(branch) 개념이다.

1. **구현**

**2.1 Class 설명**

**2.2 알고리즘에 대한** 세**부 구현**

* **여기를**
  + **채워주세요**

1. **토의**

* **여기를**
  + **채워주세요**

1. **구현과정에서 겪은 어려운 점**

* 여기를
  + 채워주세요