E - Tree and Hamilton Path 2

# 문제 요약

N개의 도시가 트리 구조로 연결되어 있으며, 각 간선에는 비용이 있습니다. 한 도시에서 시작하여 모든 도시를 최소 한 번 이상 방문할 때의 최소 이동 거리를 구하는 문제입니다.

# 예제 입력 및 출력

|  |  |
| --- | --- |
| 입력 | 출력 |
| 4 1 2 2 1 3 3 1 4 4 | 11 |

# 해법

트리의 모든 간선을 왕복하면 총 거리의 합은 2 × (모든 간선 가중치 합)입니다. 하지만 시작점과 종착점을 적절히 고르면 되돌아가지 않아도 되는 가장 긴 경로(지름)가 존재합니다. 따라서 최소 이동 거리는 다음과 같습니다:  
  
 최소 거리 = 2 × 모든 간선의 가중치 합 − 트리의 지름  
  
DFS 또는 BFS를 두 번 수행하여 지름을 구하고, 이 공식을 이용하여 답을 계산합니다.  
  
트리의 지름 구하는 방법:  
- DFS 또는 BFS를 이용하여 트리에서 임의의 한 정점에서 가장 먼 정점 u를 찾습니다.  
- 다시 u에서 가장 먼 정점까지의 거리 D를 계산합니다.  
- 이 거리 D가 바로 트리의 지름입니다.  
  
트리는 사이클이 없고 모든 정점 간 경로가 유일하므로, DFS/BFS 2번으로 지름을 정확하게 구할 수 있습니다.

# 복잡도

시간 복잡도: O(N)  
공간 복잡도: O(N)  
N은 노드의 개수이며, 트리는 항상 N-1개의 간선을 가지므로 전체 연산은 선형입니다.

# 요약

1. 모든 간선의 가중치 합을 2배로 계산합니다.  
2. 트리의 지름을 DFS를 통해 계산합니다.  
3. 최단 방문 거리는 2 × 전체 가중치 합 − 트리 지름입니다.

# C++ 코드

|  |
| --- |
| // typedef 정의 typedef long long ll;  typedef std::pair<int, ll> ilp;  typedef std::vector<ilp> int\_pair\_vector;    int main() {   ios\_base::sync\_with\_stdio(false);   cin.tie(NULL);   cout.tie(NULL);     int nrNode;   cin >> nrNode;     // 트리 저장: 인접 리스트  vector<int\_pair\_vector> tree(nrNode + 1);   ll totalWeight = 0;     // 간선 정보 입력  for (int i = 0; i < nrNode - 1; ++i) {   int s, d;   ll w;   cin >> s >> d >> w;   tree[s].emplace\_back(d, w);   tree[d].emplace\_back(s, w);   totalWeight += w \* 2; // 왕복 비용으로 계산   }     // DFS: 가장 먼 정점과 거리 반환  auto dfs = [&](auto self, int node, int parent = -1) -> ilp {   ilp res = {node, 0};   for (auto &[next, w] : tree[node]) {   if (next == parent) continue;   auto sub = self(self, next, node);   if (sub.second + w > res.second) {   res = {sub.first, sub.second + w};   }   }   return res;   };     // 첫 DFS: 임의의 노드(1)에서 가장 먼 노드 탐색  auto farthest = dfs(dfs, 1);   // 두 번째 DFS: 지름 계산  ll diameter = dfs(dfs, farthest.first).second;     // 최소 거리 = 2 \* 전체 가중치 합 - 지름  cout << totalWeight - diameter << "\n";   return 0;  } |

출처 : https://atcoder.jp/contests/abc361/tasks/abc361\_e