Tree

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

Prefix Sum

아름다운 수열

## 문제요약

길이 N짜리 수열 A에서 길이 K 이상의 연속 부분 수열을 하나 골라 아래의 '아름다운 정도'를 최대화하라.

아름다운 정도는 다음과 같이 정의된다:

아름다운 정도 = (A₁² + A₂² + ... + Aₙ²) / (A₁ + A₂ + ... + Aₙ)

답은 실수로 출력하며, 절대 또는 상대 오차가 1e-6 이하인 경우 정답으로 인정된다.

## 예제 입력 및 출력

|  |
| --- |
| 예제 입력 1: 3 1 10 8 14  출력: 14.0000000  예제 입력 2: 3 2 2 1 2  출력: 1.8000000 |

## 해법

1. 이분 탐색을 통해 아름다운 정도 x를 고정하고, 이를 만족하는 부분 수열이 존재하는지 확인한다.

2. A[i]^2 - x\*A[i]를 B[i]로 정의하고, 이 B[i]에 대해 누적합을 계산하여 길이 K 이상의 구간에서 합이 0 이상인 경우를 찾는다.

3. prefix[j] - min(prefix[0..j-K]) ≥ 0인 경우가 존재하면 해당 x는 가능한 값이다.

4. 이를 바탕으로 가능한 아름다운 정도의 최댓값을 탐색한다.

## 복잡도

- 이분 탐색: O(log(maxA / EPS)) ≒ 60

- 각 check(x) 함수는 O(N)

- 전체 시간복잡도: O(N log(maxA))

## 요약

- 목표: 연속된 부분 수열 중에서 아름다운 정도를 최대로 하는 것

- 방법: 이분 탐색 + 누적합을 이용한 조건 확인

- 테크닉: prefix sum, 최소 prefix 값 저장, 실수 오차 제어

## C++ 코드

|  |
| --- |
| double EPS = 1e-7; // 허용 오차 (이분 탐색 종료 조건)    int main() {  ios\_base::sync\_with\_stdio(false); // C++ 표준 입출력 비동기화 해제  cin.tie(NULL); // cin과 cout의 연결 해제 (속도 향상)  cout.tie(NULL);  cout << fixed; // 고정 소수점 표기  cout.precision(7); // 소수점 7자리까지 출력  int N, K; // N: 수열 길이, K: 최소 부분 수열 길이  cin >> N >> K;  vector<int> arr(N + 1); // 1-based 인덱싱 사용  int maxVal = 0; // 수열 내 최댓값 저장  for (int i = 1; i <= N; i++) {  cin >> arr[i]; // 수열 입력  maxVal = max(maxVal, arr[i]); // 최대값 갱신  }  auto check = [&](double x) -> bool { // 아름다운 정도가 x 이상인 부분 수열이 있는지 판단  vector<double> prefix(1); // 누적합 벡터 초기화  double acc = 0;  for (int i = 1; i <= N; i++) {  acc += (double)arr[i] \* arr[i] - x \* arr[i]; // B[i] = A[i]^2 - x \* A[i]  prefix.push\_back(acc); // 누적합 계산  }  double minPreFix = 0; // prefix[j-K] 중 최소값  for (int j = K; j <= N; j++) {  minPreFix = min(minPreFix, prefix[j - K]);  if (prefix[j] - minPreFix >= 0) // 길이 K 이상 부분합이 0 이상이면  return true; // 해당 x로 가능한 부분 수열 존재  }  return false; // 해당 x로는 불가능  };  double low = 0, high = maxVal, ans = 0; // 이분 탐색 범위 설정  while (low <= high) {  double mid = low + (high - low) / 2; // 중간값 계산  if (check(mid)) {  low = mid; // 가능한 경우 오른쪽으로 이동  ans = max(ans, mid);  } else {  high = mid; // 불가능하면 왼쪽으로 이동  }  if (abs(low - high) < EPS) break; // 충분히 좁혀졌으면 종료  }  cout << ans << endl; // 최종 결과 출력  return 0;  } |

출처 : https://codejam.lge.com/contest/problem/1520/3