Patrol: Solution

The road network forms a tree T. A tree with N nodes has $N-1$ edges. In T, the length of a tour that visits all edges is $2(N-1)$, because each edge is visited twice. Recall that adding edges into a tree creates cycles.

Simpler case

We consider a simpler case when $K = 1$. Suppose that we add edge e to T. The resulting graph T' contain exactly one cycle C. The cheapest tour visiting all edges uses each edge in C once and all other edges twice. Denote $C - e$ as path P. The new length of the required tour is

$$2(N-1) - L + 1$$

where L is the length of P. Thus, for $K = 1$, we need to find the maximum path length for paths in T. This value is called the *diameter* of T.

There are many ways to find the diameter. We shall use dynamic programming, which can be turn to be the solution for the general case.

First we root the tree at some node r; the parent-child relation between adjacent nodes can be defined naturally. For each node u, let $H[u]$ denote the length of the longest path from u to some of its descendants. We can compute $H[u]$ for each u, in $O(N)$ time, using a simple dynamic programming.

Consider the longest path P, let node u be the node on P closest to the root r. By definition, u is unique. Given u, the length of P must be either $H[u]$, if u has one children, or

$$\max_{v,w \text{ : different children of } u} (2 + H[v] + H[w])$$

when u has more than one children. The value above is important to the case where $K = 2$ as well, so let's define it as $L[u]$. Formally, $L[u]$ is the maximum length of paths containing u such that u is the closest node to root r.

Thus by enumerating all nodes, one can find the length of the longest path; thus, one can compute the answer to the case where $K = 1$.

When $K = 2$

Let's call both edges e_1 and e_2. Let path P_i be a unique path that join two endpoints of e_i, also let's call a unique cycle induced by adding each edge e_i (separately) as C_i. Note that C_i is a union of P_i and e_i.

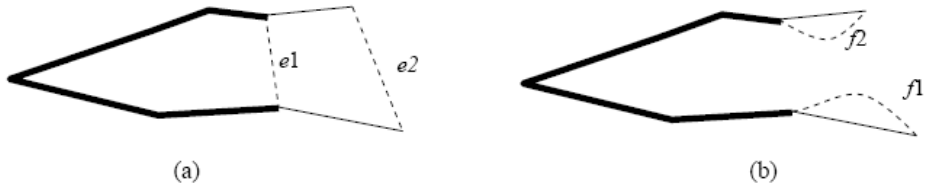

Figure 1: (a) Edges e_1 and e_2 are shown as dashed lines. Paths P_1 and P_2 intersect. The intersection is shown as thick line. In the tour, these edges must still be traversed over twice. (b) The new edges f_1 and f_2 are shown as dashed lines. Note that the number of times each edge on the tree is traversed on is the same as before.

When P_1 and P_2 are disjoint, the length of the desired tour that traverses all edges is

$$2(N-1) - |L_1| - |L_2| + 2$$

where L_i is the length of P_i.

It gets more complicated when P_i's intersect. However, since one must traverse on each e_i exactly once, it is not hard to prove the following claim.

Claim: If P_1 and P_2 intersects, there is another pair of edges f_1 and f_2 such that the paths joining each edge's endpoints are disjoint, and the length of the tour traverses all edges in $T + f_1 + f_2$ is the same as in $T + e_1 + e_2$.

The proof is left out, but Figure 1 illustrates the idea of the proof.

From the claim, to find how to add two edges to minimize the tour, we need to only consider finding a pair of disjoint paths whose sum of lengths is maximum. This, again, can be solved using dynamic programming in $O(N)$ time.

Beside $H[u]$, we need other variables. Let T_u be the subtree rooted at u. We define:

- $A[u]$ is the maximum length of paths inside T_u.
- $B[u]$ is the maximum sum of lengths of any pairs of edge-disjoint paths P and Q in T_u such that one endpoint of P is u.

Figure 2 shows examples of paths considered in $A[u]$ and $B[u]$.

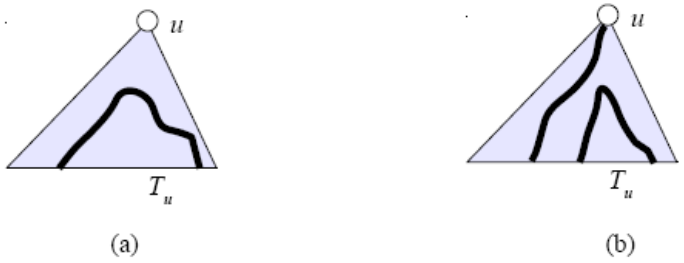

Figure 2: (a) Paths considered in $A[u]$. (b) A pair of paths considered in $B[u]$.

Let $ch(u)$ denote the number of children of u on the rooted tree T. It takes $O(ch(u))$ time to compute $A[u]$ from information from its children by taking the maximum of $A[v]$ for all children v of u and $L[u]$.

To compute $B[u]$, a straight-forward implementation takes $O(ch(u)2)$ time. A careful implementation only takes $O(ch(u))$ time. (See discussion in the next section.)

With A's and B's of all child nodes of u at hand, one can find $D[u]$ the maximum sum of lengths of pairs of paths P_1 and P_2 such that

- P_1 and P_2 are disjoint,
- P_1 contains u, and
- Among all nodes in P_1 and P_2, u is the closest to root r of T.

Again, a careful implementation runs in $O(ch(u))$ time. Easier implementations that run in $O(ch(u)^2)$ time and $O(ch(u)^3)$ time exist. We discuss the implementations later.

After computing all $O[u]$'s, the minimum length of the desired tour is

$$2(N-1) - \max_u D[u] + 2$$

Computing $B[u]$ and $D[u]$

We first discuss how to compute $B[u]$. Let $CH(u)$ denote u's children. Recall that $B[u]$ is the maximum sum of the length of a pair of edge-disjoint paths P and Q such that u is one end of P.

There are many cases to for P and Q:

- Case 1: Both P and Q contains u. In this case, we can compute $B[u]$ by finding 3 children with largest hight.
- Case 2a: P contains edge (u, v) for some child v in $CH(u)$, and Q also lies entirely in T_v. In this case, we have that $B[u] = 1 + B[v]$.
- Case 2b: P contains edge (u, v) for some child v in $CH(u)$, but Q lies entirely in T_w for some child w not equal v. In this case, $B[u] = 1 + H[v] + A[w]$.

Case 1 and Case 2a can be considered in $O(ch(u))$ time. By checking all pairs of children in $CH(u)$, we can consider Case 2b in $O(ch(u)^2)$ time. The time can be reduced to linear by noticing that we can preprocess by finding a child x with maximum $A[x]$. With that, we can consider the value of $1 + H[v] + A[x]$ when v is not equal to x, and $1 + H[x] + \max_{w \neq x} A[w]$ when $v = x$. The total running time is $O(ch(u))$.

The same idea can be applied to computing $D[u]$. In this case we want to find two edge-disjoint paths P and Q in T_u. There are 3 cases to consider:

- Both P and Q contain u.
- Neither P nor Q contain u.
- One contains u.

The first two cases are easy to implement to run in time $O(ch(u))$. The last one can be implemented to run in $O(ch(u)^3)$. The idea from the computation of $B[u]$ can be applied here to reduce the running time to $O(ch(u)^2)$ and $O(ch(u))$.

Scoring

Since optimizing the computation of B's and D's are not the essential part of the task, solutions that uses both $O(ch(u)^3)$ and $O(ch(u)^2)$ per node u should score the majority of the test cases.