Code Jam 2010 - Round 2

Analysis: World Cup 2010

Reformulation

In the analysis, we will turn the picture upside-down. This conforms to the usual convention that the root of a binary tree is drawn on top. This is actually easy to do implementation-wise. We just read the input numbers in reverse order; the element at position 0 is the root, i.e., the final match, and for any node `i`, its two children are indexed at `2i+1` and `2i+2`. In our particular solution, we include nodes for the teams as well as for the matches, so our graph is a rooted complete binary tree, where all the internal nodes are matches, and all the leaves are the teams.

Perhaps it is clear that the solution will be dynamic programming on the tree. There are many ways to do it, but some can be much more complicated than others. Let us introduce one solution that is very simple. It is based on the following reformulation of our problem.

Let us color a node yellow if we are going to buy the ticket for the corresponding match. For each leaf (a team), the path to it from the root has `P` internal nodes. We call a plan good if no matter what the outcomes of the matches are, we will never miss more than `M[i]` matches for team `i`. We have

> (*) A plan is good if and only if for every team `i`, the path from the root to the leaf corresponding to team i has at least `P-M[i]` yellow nodes.

This is indeed a conceptual leap in solving the problem. For in the new form, we no longer need to worry about the outcome of any particular match. And once it is stated, it is very easy to justify. We leave the justification to the readers.

Simple solutions for small inputs and large inputs both follow easily from this reformulation.

Small dataset

For the small dataset the best plan must be *upwards closed*, i.e., if a node is yellow, all the nodes on the path from the root to it must all be yellow. One can prove this by using (*) and noting that all the prices are the same, and you can always get a cheaper plan if you push some yellow node upwards. So, one solution for the small dataset looks like this: For each team `i`, we find the path from root to it, and color the topmost `P-M[i]` nodes yellow. It can be done in linear time by a simple modification of depth first search.

The justification and the implementation details are a nice piece of dessert. We leave them to the readers.

Large dataset

We use dynamic programming on the tree. Let us just define the subproblems clearly. For any node `a` and any number `b` between 0 and `P`, let `P(a, b)` be the problem

> If there are `b` yellow nodes on the path from the root to `a` (not including `a`), what is the cheapest way to color the nodes in the sub-tree rooted at a, such that all the leaves in this sub-tree satisfy the condition in (*)?

And we can use a impossibly big answer to indicate that the condition can not be satisfied.

Once the DP is set up, the code follows easily. For each internal node there are just two choices, color it yellow or not. We provide an excerpt from one of the judges' solutions:

```
i64 A[1<<11][11];  // answers for the dynamic programming.
i64 inp[1<<11];    // input, in reversed order.
int P;

// rooted at a, already buy b tickets from above.
i64 dp(int a, int b) {
  i64& r=A[a][b];
  if(r>=0) return r;
  if(a>=((1<<P)-1)) {r=(b>=(P-inp[a]))?0:1LL<<40; return r;}
  r=std::min(
      inp[a]+dp(2*a+1,b+1)+dp(2*a+2,b+1),
      dp(2*a+1,b)+dp(2*a+2,b));
  return r;
}

int main() {
  int T;
  cin>>T;
  for (int cs=1; cs<=T; ++cs) {
    cin>>P; int M=(2<<P)-1;
    for (int i=0; i<M; i++) cin>>inp[M-1-i];
    memset(A, -1, sizeof(A));
    cout << "Case #" << cs << ": " << dp(0, 0) <<endl;
  }
  return 0;
}
```