Code Jam 2011 - Round 1A

Analysis: Pseudominion

Let's denote the bonus numbers of a card c by c.card_draw, c.score and c.turns. Let c.index be the (zero based) index of the card in the ordering from the input file.

After looking at the problem constraints, we can classify a card c as:

- a T card, if c.turns > 0.
- a C_0 card, if c.turns = 0 and c.card_draw = 0
- a C_1 card, if c.turns = 0 and c.card_draw = 1
- a C_2 card, if c.turns = 0 and c.card_draw = 2

Let T[i] denote the i-th T card in the sequence of all T cards sorted by the index. We define C_0[i], C_1[i] and C_2[i] analogously.

The first key observation is that it never hurts to play a T card whenever we have one in our hand. We never lose turns or score and we may draw new cards by doing so.

Let's think about some arbitrary sequence of played cards. There are two observations we need to make before we can proceed.

1. Because C_0 cards do not add turns or cards to our hand, we can postpone playing them until we have no other cards we want to play. So we can transform any valid sequence of played cards into another sequence that has all C_0 cards at the very end. The total score, being the sum of individual scores of all played cards, will obviously remain the same.
2. Let's say there are two C_1 cards a and b such that we played a before b, but b.index < a.index. Because b.index < a.index, we already had both cards in our hand when we played card a. So we can instead play card b first and card a second. The resulting sequence with a and b swapped will remain valid because a and b both have the same card bonus (1 card) and the same turn bonus (0 turns). Furthermore, the score will remain the same. We can do the same for two C_2 cards.

The first observation shows us that we don't care about C_0 cards until the very end when we can use our remaining turns on whatever C_0 cards we have already drawn. Obviously, we should sort the C_0 cards in our hand by decreasing score bonus and then play as many as we can.

The second observation shows us that we can transform any optimal sequence of played cards into another one that has cards of the same type ordered by index. This means that we can play C_1 (and C_2) cards in increasing order by index.

So, whenever we have more than one C_1 (or C_2) cards in our hand we can always look at the one with the smallest index and choose whether we will play it right now or never at all.

These observations lead us to construct a directed acyclic weighted graph. A node represents a state of the game and is defined by these properties:

- hand - the number of drawn cards.
- turns - the number of turns left.
- t - T[t] is the first T card that we haven't played yet.
- c1 - C_1[c1] is the first C_1 card which we are not yet sure if we are going to play.

- c2 - C_2[c2] is the first C_2 card which we are not yet sure if we are going to play.

An edge represents a valid transition from one game state to another and is usually associated with playing a card or deciding not to play a certain card at all. The weight of an edge indicates how much your score goes up when you switch between the two game states. For each node (hand, turns, t, c1, c2), we add edges according to these rules:

1. If we have a T card in our hand, we can play it.
    - Condition: T[t].index < hand
    - Weight: T[t].score
    - Target node: (min(N, hand + T[t].card_draw), min(N, turns + T[t].turns - 1), t + 1, c1, c2)
2. If we have a C_1 card in our hand, we can play the first one.
    - Condition: C_1[c1].index < hand
    - Weight: C_1[c1].score
    - Target node: (min(N, hand + 1), turns - 1, t, c1 + 1, c2)
3. If we have a C_1 card in our hand, we can throw away the first one.
    - Condition: C_1[c1].index < hand
    - Weight: 0
    - Target node: (hand, turns, t, c1 + 1, c2)
4. If we have a C_2 card in our hand, we can play the first one.
    - Condition: C_2[c2].index < hand
    - Weight: C_2[c2].score
    - Target node: (min(N, hand + 2), turns - 1, t, c1, c2 + 1)
5. If we have a C_2 card in our hand, we can throw away the first one.
    - Condition: C_2[c2].index < hand
    - Weight: 0
    - Target node: (hand, turns, t, c1, c2 + 1)
6. We can always decide to finish the game by spending the remaining turns on C_0 cards. This edge leads to the special final node and the weight of the edge is defined by the greedy algorithm that spends all remaining turns picking the best C_0 cards.

The answer to our problem is the length of the longest path in the graph described above. The graph is acyclic, so we can use dynamic programming or recursion with memoization to find the length of the longest path.

The number of nodes in the graph is $O(n^5)$, so the asymptotic time complexity of the algorithm is $O(n^5)$ too. In practice, the runtime of the program is really small because the vast majority of these states are unreachable. You can speed things up even more if you use the preceding observations to their full power. For example, if there is an edge corresponding to a T card, you should always follow it first.

Remark: This problem is inspired by the game Dominion, published by Rio Grande Games.