Candy Machine

Spoiler

A first, simple approach is to use dynamic programming over the wagon positions. This would lead to a run time of $O(n^w)$; 20% of the score should be awarded to such a solution.

When looking for a better solution, we observe that the paths of two wagons never need to cross. Two crossing paths can be replaced by two paths with the parts after the crossing exchanged. Then, we have a “smallest” wagon, i.e. a wagon such that there is no candy with a smaller position than any of the candies caught by that wagon. Now assume we can compute an *optimal smallest wagon path* (OSWP) such that this wagon catches as many candies at possible. Afterwards, we disregard all candies caught by the smallest wagon, compute the OSWP for the remaining candies, and so on. If the computation of the OSWP can be done in $O(n)$, the whole algorithm runs in $O(n^2)$.

Indeed, it is possible to compute an OSWP in $O(n)$. We pass through all candies by increasing output time and maintain a stack of candies; candies on the stack are caught by the wagon. The first candy is pushed on top of the stack. Furtheron, we consider the current candy c and the candy on top of the stack (the last candy to be caught by the wagon) c_s. If the wagon can run from the position of c_s to the position of c such that it catches c on time, then c is put on top of the stack. Else, and if the position of c is smaller than that of c_s, the stack is popped until c is reachable from c_s again. Afterwards, c is put on the stack. 60% of the score should be awarded to a correct $O(n^2)$ algorithm.

A more efficient algorithm builds upon the idea non-crossing paths, too, but passes candies by position. Each candy is to be assigned to the leftmost possible wagon. If binary search is used to determine the wagon and a balanced tree is used to store each wagon’s candies, a run time of $O(n \log^2 n)$ is possible. Full score should be awarded to a correct algorithm with such runtime.

Further improvements may even yield a runtime of $O(n \log n)$:

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

typedef pair<long long, long long> pii;

int N;
vector<pii> candies;
vector<long long> wagons;
vector<pair<pii, int> > output;

int main() {
        scanf("%d", &N);
        for (int i = 0; i < N; i++) {
                long long p, t;
                scanf("%lld%lld", &p, &t);
                candies.push_back(pii(p + t, t - p));
        }
        sort(candies.begin(), candies.end());
        for (int i = 0; i < N; i++) {
                long long a = candies[i].first, b = -candies[i].second;
                int pos = lower_bound(wagons.begin(), wagons.end(), b) - wagons.begin();
                if (pos == (int) wagons.size()) {
                        wagons.push_back(b);
                } else {
                        wagons[pos] = b;
                }
                output.push_back(make_pair(pii((a + b) / 2, (a - b) / 2), pos + 1));
        }
        printf("%d\n", wagons.size());
        for (int i = 0; i < N; i++) {
                printf("%lld %lld %d\n", output[i].first.first, output[i].first.second, output[i].second);
        }
        return 0;
}
```