Code Jam 2013 - Round 1A

Analysis: Manage your Energy

The small dataset

The limits in the small dataset are very small for this problem, and they allow brute-force approaches. With a fast enough language, even an algorithm that for each activity in the row tries each possible amount of energy usage (from 0 to current energy) will run in time.

The large dataset obviously does not allow such a solution — both the number of activities and the number of possible amounts of energy to consider are significantly too large.

The highest-valued activity

We will begin solving this problem by looking at the highest-value activity (if there is more than one, pick any of them), let this be activity number **a**. We will prove that there exists an optimal solution that uses full **E** joules for this activity.

Consider any optimal solution. First assume that we had less than **E** joules at the start of activity **a**. This means we spent energy on some other activity before (since we started with **E** energy), let **b** be the number of the last activity we spent non-zero energy on before **a**. Now consider a solution in which we spend one joule less on **b**. Since our energy wasn't **E** in the original solution at the beginning of **a**, if we spend one less joule on **b**, we will still have at most **E** energy at the beginning of **a**, and so it won't go to waste. Thus, we can spend this extra joule on **a**, and then proceed as in the original solution, since after **a** we have exactly the same amount of energy as in the original. The difference in values of these two solutions is $-\mathbf{v_b} + \mathbf{v_a}$, which is non-negative (since **a** was one of the highest-value solutions). We can repeat this procedure until we obtain a solution which is no worse than the original (and thus still optimal), and has **E** joules at the beginning of **a**.

Now assume that we have an optimal solution that enters **a** with **E** joules, but does not use all of them on **a**. We can change it in a similar fashion. Find the first activity after **a** on which we spend non-zero energy, call it **c**. If it doesn't exist, or if any energy is wasted between **a** and **c**, we can simply expend one more joule on **a** and obtain a strictly better solution. Otherwise, we can spend one joule less on **c** and one more on **a**, to obtain a solution that's no worse.

After repeating these procedures as long as we can, we obtain an optimal solution in which, indeed, we spend full **E** joules on **a**.

Other activities

Notice that the fact we are spending **E** energy on **a** has some implications on the nearby activites. For instance, we will have at most **R** energy available at **a**+1, at most 2**R** at **a**+2, etc. At the same time, we need to leave **a**-1 with at least **E**-**R** energy, leave **a**-2 with at least **E**-2**R** energy, and so on.

Consider the activity **d** with the next-highest value after **a**. We have a limit on how much energy we have at most when entering **d** (it's the lower of two numbers — **E** and whatever limit the spending on **a** imposed); and we know how much energy we have left unspent (the higher of 0 and the limit imposed by **a**). We will spend anything between these two limits on activity **d** — a reasoning identical to the one for **a** proves that this doesn't prevent us from getting an optimal solution.

We will continue in this fashion. At each step, we will have for each activity an amount of energy that we have to leave after ending it for activities we already considered, and the maximum amount of energy we can have beginning this activity. At each step, we will take the highest-valued activity we haven't considered yet, and assign to it all the energy we can, updating the limits afterwards.

The time complexity of this solution, as formulated, is O($\mathbf{N}^2$) — in each of the **N** steps we find the highest-valued activity not considered yet, assign energy to it, and update limits for all other activities. Due to the constraints on input size that we have this will be fast enough (at least if written reasonably efficiently). It is possible to do better, though.

An O(N logN) solution

First note that finding the highest-valued activity not considered yet can be done faster — it's enough to just sort the activities by value up front, and then consider them in descending order.

The more difficult part is updating the limits. To do this, we look at how the limits are set again. Each activity we assign imposes a limit on how much energy do we have in later activities, and how much do we have to leave behind in the previous activities. We would like to prove that for each activity **a** the limit we have to consider comes from the latest activity in the day we considered before **a** for how much energy we have, and the first activity we considered later in the day than **a** for how much we need to leave unspent.

This is not surprising. Consider, for instance, the activity **b**, which is the latest activity considered before **a**, and some activity **c** which comes even earlier in the day. If we considered **c** before **b**, the loss of energy on activity **c** is already taken into account when we consider **b**, so the limits imposed by **b** will be no less stringent. On the other hand, if we considered **c** after **b**, it means that the energy spent on **c** was already limited so that it would all be recovered by the time we reach **b** — and so it does not impact the amount of energy we have available at **a**. A similar reasoning works for activities later in the day than **a**.

A solution in which each step takes logarithmic time will take each considered activity and insert it (along with the limits it imposes) into a binary search tree allowing logarithmic-time insertions and lower/upper-bound operations (like the "set" structure of many languages). Then, at each subsequent activity, we find the nearest activity already considered before and after it, compute the limits they impose on the current activity, spend all the energy we can (updating the total value) and insert this activity (along with new limits) into the tree.

Note that if **R** ≥ **E**, you can just assume **R** = **E**, since any energy you regain above **E** will necessarily go to waste. We observed this and changed the problem statement to avoid this unnecessary case; but we missed some of the inputs in which this should have been corrected, due to which we had to change the statement back and issue a clarification during the contest. We apologize for the confusion this caused.

An O(N) solution

One nice thing about running a contest for a group of very smart people is that they come up with solutions for problems that the authors didn't even think of! This was the case with this problem — after the contest we learned some contestants came up with a O(**N**) solution.

The key to this solution is observing that we can actually know up front how much energy we will want to spend at a given day. As seen above, if there is no more valuable activity ahead of us, we should just spend all the energy we have at the moment. On the other hand, if there is something more valuable, it's always a good idea to save up energy if we will be able to use it

for the more valuable activity. To make use of this we need to know, for each activity, the nearest activity in the future that's more valuable.

Calculating such an array is a classic problem, it is described, e.g., as the "Stock Span Problem" in the Wikipedia article on stacks. Once we have this array, we can, for each activity in chronological order, do the following:

- If there is no more valuable activity coming up, spend all the energy we have.
- Consider the nearest more valuable activity **X**. If we can spend any energy, and still have **E** energy when **X** comes (assuming we don't spend any between now and **X**), spend as much as we can while still having **E** when **X** comes. Notice the alternative to spending it now would be to spend it between now and **X** — but there's no activity more valuable than the current one in the period to make it worthwhile.
- If we can't spend any energy and still have **E** when **X** comes, we shouldn't spend any — all our energy will be better spent at **X**.

Thanks to for pointing this out to us! The credit for this solution goes to pedrosorio; other O(**N**) solutions are possible as well (see, e.g., misof's submission).