Code Jam 2011 - Round 2

Analysis: Airport Walkways

Some people will tell you that programming contest problems, while interesting to think about, have no practical use in the real world. After solving this problem, you can laugh at these people as you catch your flight and they are left waiting in line at the airport with all the other chumps.

An important thing to notice about this problem is that the location of the walkways does not matter, since you can instantly transition between different speeds. This means that two walkways with speed **v** of length $\mathbf{L_1}$ and $\mathbf{L_2}$ can be combined into a single walkway of length $\mathbf{L_1 + L_2}$ with speed **v**. By combining this with the observation that any section of corridor with no walkway is equivalent to a walkway with $\mathbf{v = 0}$, you reduce the problem to having 101 different speed walkways of variable length (possibly 0) and deciding for each whether to run or walk (or do some of each).

The small dataset can be solved by brute force. Let **W** be the set of all positive length walkways (each with a unique speed). Since there are only $\mathbf{|W| \le 21}$ different speed walkways to consider, you can simply try choosing each subset $\mathbf{S \subseteq W}$ of them to run (and just walk the rest, **W - S**). This solution is slightly complicated by the fact that once you choose **S**, you still have to decide which one to only run partially, in case you donâ€™t have time to run them all fully. However, you can again just brute force this decision by iterating over each walkway $\mathbf{x \in S}$ and only running on walkway **x** with whatever time is left after completely running all walkways in **S - {x}**. You simply take the minimum over all choices, discarding any choice which requires more than **t** seconds of running time. This can easily be implemented in $\mathbf{O(N^2 * 2^N)}$ and will run in time for **N** at most 20.
Bonus: Implement this algorithm in $\mathbf{O(N * 2^N)}$ time.

However, this approach will time out for the large data set, where **N** is at most 1000 and you can have walkways of all 101 different speeds. For this, we need a better way to decide when to run and when to walk. We just need to decide if itâ€™s better to run on slower or faster walkways. It turns out we can solve this with a simple greedy algorithm, and we can prove it is optimal by using a simple exchange argument.

Letâ€™s say we have two arbitrary walkways of speed $\mathbf{w_1}$ and $\mathbf{w_2}$, with $\mathbf{w_1 < w_2}$. Now letâ€™s say that in some algorithm weâ€™ve decided to run for $\mathbf{r_1}$ and $\mathbf{r_2}$ seconds on each of the walkways, respectively. If $\mathbf{s_1}$ and $\mathbf{s_2}$ are the amount of time we spent walking on each of the two walkways, then our total time spent would be $\mathbf{T = r_1 + s_1 + r_2 + s_2}$ seconds. What if instead we decided to run for $\mathbf{r_1 + \varepsilon}$ seconds on the first walkway and $\mathbf{r_2 - \varepsilon}$ seconds on the second walkway, with $\mathbf{\varepsilon > 0}$? Then our total time would be **Tâ€™ =** $\mathbf{(r_1 + \varepsilon) + s_1}$**â€™** $\mathbf{+ (r_2 - \varepsilon) + s_2}$**â€™** seconds. Solving for **T - Tâ€™**, you will get $\mathbf{\varepsilon * (w_2 - w_1) * (R - S) / ((w_1 + S) * (w_2 + S)) > 0}$, which says that **T > Tâ€™**, and so the change will always be beneficial. Simply put, itâ€™s always better to run on slower walkways as much as possible. *Note that some of the details of these equations have been excluded from this analysis and are left as an exercise to the reader.*

Here is some simple Java code which solves the problem:

```
double res=0;
for (int i=0; i<=100; ++i) {
  double runTime=Math.min(t,W[i]/(i+R));
  double walkDist=W[i]-runTime*(i+R);
  double walkTime=walkDist/(i+S);
  res+=runTime+walkTime;
  t-=runTime;
}
System.out.printf("Case #%d: %.12f%n",TC,res);
```

Bonus: Solve the problem if the limits changed to $1 \le \mathbf{w_i} \le 10^{9.}$