SACO 2010 Day 2 Solutions

SACO Scientific Committee

1 Coaster

This problem can be solved by simulation: create an array with a boolean for each car, and if somebody can sit in car C, simply loop through all cars from C to K until an empty car is found.

We can analyze the worst-case performance of this algorithm. If F cars are already full, we will examine at most $F + 1$ cars before finding an empty one (or finding that there is none).

If $N \leq K$, the total number of cars examined is at most

$$1 + 2 + \cdots + N = \frac{N(N+1)}{2}.$$

IF $N > K$, the total is at most

$$1 + 2 + \cdots + (K-1)) + (N-K)K = \frac{K(K-1)}{2} + (N-K)K$$

.

In either case, the answer is less than N^2, so this algorithm is fast enough. There are, however, faster algorithms for this problem.

2 Track

2.1 Partial solutions

One solution is to consider every possible pair of a and b, and find the sum of the track lengths between them, checking whether any of the sums if equal to the desired length. This takes time proportional to the cube of N, and scores around 20%.

The problem with this solution is that, for $a = 4$, we add up the tracks from 1 to 1, then 1 to 2, then 1 to 3, etc. We can do much better by simply adding another track until the total is equal to the desired length; if it gets larger, we stop and consider the next a. This avoids recalculating the full total every time b changes, and takes quadratic time, scoring 40%.

2.2 Full solution

To get a full score, we apply a similar change when we move a: instead of recalculating the totals every time the total gets too large and a is incrememnted, simply remove track number a from the total and increment it.

In other words, every time the total is too large we increment a and whenever it is too small we increment b.

Since a takes on each value from 1 to N at most once, and similarly for b, this algorithm takes linear time, and therefore scores 100%.

3 Haunted

Finding the longest path in a graph is a well known NP-Complete problem. This means there is no known "quick" solution to it. If you could find a "quick" solution to it, you would win a million dollars from the Clay Mathematics Institute for solving the P=NP problem.

Luckily the problem statement only allows for graphs called trees, allowing us to solve the problem "quickly". Let's first discuss a sub-optimal solution. Given a starting room, we can recursively visit every other room using a technique called Depth-First Search (DFS) `http://en.wikipedia.org/wiki/Depth-first_search`. We can modify the DFS to record the length of the path so far when we visit a room. Now for each room we generate every path starting with it and record the maximum length seen. This should be the answer. The DFS takes time $O(N)$ and we do it for each room, so we get a total running time of $O(N^2)$.

One of the full solution requires noticing a property of the longest path. Pick an arbitrary room as the "root" of your tree. Now find the furthest room from the root by modifying the DFS from the previous paragraph to return the a furthest found room. Due to the properties of trees, this room must be the start of a longest path. So use the DFS from the previous paragraph to find the length of the path starting at this room. The length calculated is the answer. The runtime complexity of this algorithm is $O(N + N) = O(N)$.

4 PPP

This problem was essentially a generalisation of the binary and linear search algorithms. When $S = 0$, the a linear search is an optimal solution and when $T = 0$, a binary search is optimal.

4.1 40% Solution

The 40% uses a technique called dynamic programming. Let $\tau(a, b)$ be the maximum time taken to determine which factory is polluting the river if the factory is in the range a to b. You can determine the following recurence by considering moving i factories along the river. To move a distance you could either need to move to the left or the right so we take the maximum of these cases.

$\tau(a,b) = \min_{i=0}^{a-b+1}((i+1)T + max(\tau(a+i+1,b), \tau(a,b-i-1)))$
$\tau(a,a) = 0$
$\tau(a,a+1) = S+T$

The solution can be found to be $\tau(1,N)$. This runs in $\mathcal{O}(N^3)$ time.

In all cases, once the maximum time is obtained then a simple linear pass can be made in order to determine which factory to probe first. At every factory ask what time is needed to travel there and then add the maximum of the time needed to find the factory each of the subsections to the left and to the right of this point. A probing position which takes a time equal to the calculated maximum time is valid.

4.2 60% Solution

The 40% can be easily modified to run in $\mathcal{O}(N^2)$ time by noticing that $\tau(a,b)$ is the same for cases with the same distance between a and b. This is because no matter how far down the river you are, only the distance you need to travel affects the maximum time needed.

$\tau'(n) = \min_{i=0}^{n-1}((i+1)T + max(\tau'(i+1), \tau'(n-i-1)))$
$\tau'(0) = 0$
$\tau'(1) = S+T$

The solution can is then $\tau(N)$.

4.3 Linear-time Solution

The above situation can be simplified to yeild a faster solution. Firstly, notice that i only needs to go up to $\lceil \frac{n}{2} \rceil - 1$ and the maximum can be eliminated.

$\tau'(n) = \min_{i=0}^{\lceil \frac{n}{2} \rceil - 1}((i+1)T + \tau'(n-i-1))$
$\tau'(n) = \min_{i=\lfloor \frac{n}{2} \rfloor}^{n-1}((n-i)T + \tau'(i))$
$\tau'(n) = nT + \min_{i=\lfloor \frac{n}{2} \rfloor}^{n-1}(\tau'(i) - iT)$

A sliding window can now be used to calculate the minimum efficiently. The values of $\tau'(i) - iT$ are added to an appropriate data structure once $\tau'(i)$ is calculated the mimimum is queried as needed. Once the values in the data structure get too old they need to be removed. If a priority queue is used to query the minimum an $\mathcal{O}(N \log N)$ algorithm is obtained. Some clever use of a deque can reduce this to $\mathcal{O}(N)$ time. Both of these score 100%.

4.4 Logarithmic-time Solution

There is an even faster solution which is not required to get 100% on this problem. By examining the linear time solution it can be seen that $T(N-1)$ time is taken travelling in all cases. In can further be shown that exactly $\lceil log_2 N \rceil$ probes are needed in the worst case. This means that the total time is simply $T(N-1) + S\lceil log_2 N \rceil$.

The first factory to probe can be found as above or by noticing that choosing the middle factory always yields optimal results.