USACO OPEN09 Problem 'job' Analysis

by Richard Peng

The key observation is to wait as much as possible before doing a job. In other words, we process the jobs in reverse order of their due times. Then once a job becomes 'doable', it will remain doable thereafter.

The second observation is that if we have a series of jobs that are doable, it's always best to do the one that gives the most profit. This is because we can always interchange between jobs, so we can gradually turn any 'optimal' solution to one where the max profit job is always chosen. Of course, if no such jobs remain, we can afford to be idle for that turn.

So after sorting the jobs by their deadlines, the order we iterate through them requires a data structure that supports the insertion of a profit value and the extraction of the maximum profit value. This can be done using a max-heap, which is essentially a complete binary tree where the value on each node is less than that of its parent. STL heaps can also be used.

C++ code by John Pardon:

```
#include <cstdio>
#include <algorithm>
#include <queue>

using namespace std;

struct JOB
{
	long long dead;
	long long prof;
	bool operator <(const JOB &o) const
	{
		return dead < o.dead;
	}
};

long long N;
JOB job[100001];

int main(void)
{
	FILE *inFile = fopen("job.in", "rt");
	fscanf(inFile, "%lld", &N);
	for(long long i = 0 ; i < N ; i++)
		fscanf(inFile, "%lld%lld", &job[i].dead, &job[i].prof);
	fclose(inFile);
	job[N].dead = 0;
	job[N].prof = 0;
	N++;

	sort(job, job + N);
```

```
	priority_queue<long long> pq;
	long long curTime = 2000000000LL;
	long long profit = 0;
	for(long long i = N - 1 ; i >= 0 ; i--)
	{
		while(curTime > job[i].dead && pq.size())
		{
			curTime--;
			profit += pq.top();
			pq.pop();
		}

		curTime = job[i].dead;
		pq.push(job[i].prof);
	}

	FILE *outFile = fopen("job.out", "wt");
	fprintf(outFile, "%lld\n", profit);
	fclose(outFile);

	return 0;
}
```