```
public void dyProSolve() {
    // there's a maximum amount of fuel which we would ever want to store in the tank
    int maxStorage;
    if (tankCapacity*storageCost <= deliveryCost) maxStorage = tankCapacity;
    else maxStorage = (deliveryCost - (deliveryCost % storageCost)) / storageCost;

    // this is our lookup table
    TableEntry[][] bestCosts = new TableEntry[maxStorage + 1][numDays];

    // for the first day, we have to pay a flat delivery rate regardless of the amount
    of gas purchased.
    // we also set all the parent pointers to null to mark these days don't have parent
    cells
    for(int gasLeftOver = 0; gasLeftOver < maxStorage + 1; gasLeftOver++) {
        TableEntry newCost = new TableEntry(0, gasSoldPerDay[0] + gasLeftOver,
        deliveryCost, null);
        bestCosts[gasLeftOver][0] = newCost;
    }

    // for every day..
    for(int today = 1; today < numDays; today++) {
        //and for every amount of gas which we want to have left over on that day...
        for(int gasLeftOver = 0; gasLeftOver <= maxStorage; gasLeftOver++) {
            // we need to know the cheapest way to reach that state.
            int[] costs = new int[gasSoldPerDay[today] + gasLeftOver + 1];
            // we can only store so much in the tank, so in certain cases we must
            purchase at least a minimum amount.
            int smallestPurchase = 0;
            if((gasSoldPerDay[today] + gasLeftOver) > maxStorage) smallestPurchase =
            gasSoldPerDay[today] + gasLeftOver - maxStorage;
            // for every amount of gas we could choose to order today...
            for(int gasBoughtToday = smallestPurchase; gasBoughtToday <=
            gasSoldPerDay[today] + gasLeftOver; gasBoughtToday++) {
                // we need to know how much it would cost given the amount of gas we
                have left over from yesterday
                costs[gasBoughtToday] = bestCosts[gasSoldPerDay[today] + gasLeftOver -
                gasBoughtToday][today - 1].getCost();
                costs[gasBoughtToday] += deliveryCost;
                costs[gasBoughtToday] += storageCost * (gasSoldPerDay[today] +
                gasLeftOver - gasBoughtToday);
            }
            // if we don't purchase gas, we don't pay the delivery cost.
            costs[0] -= deliveryCost;

            // now we find the minimum cost that gives us the desired amount of gas
            left over.
            int minCost = costs[smallestPurchase];
            int amountPurchased = smallestPurchase;
            int amountFromYesterday = gasSoldPerDay[today] + gasLeftOver -
            smallestPurchase;
            for(int gasBoughtToday = smallestPurchase; gasBoughtToday <=
            gasSoldPerDay[today] + gasLeftOver; gasBoughtToday++) {
                if(costs[gasBoughtToday] < minCost) {
                    minCost = costs[gasBoughtToday];
                    amountPurchased = gasBoughtToday;
                    amountFromYesterday = gasSoldPerDay[today] + gasLeftOver -
                    gasBoughtToday;
                }
            }
            // now we store the best option in the table (the day, the amount we bought
            today, the cost so far, and the state we were in yesterday).
            TableEntry newCost = new TableEntry(today, amountPurchased, minCost,
            bestCosts[amountFromYesterday][today-1]);
            bestCosts[gasLeftOver][today] = newCost;
        }
    }

    // this is the cost of the solution.
    totalCost = bestCosts[0][numDays - 1].getCost();
```

```
55      // now we need to look back through the table and see how much we bought on each day.
56      TableEntry startCell = bestCosts[0][numDays - 1];
57      while(startCell != null){
58          if( startCell.getGasPurchased() != 0) {
59              numDaysPurchased++;
60              deliveryDays[startCell.getDay()] = true;
61          }
62          gasOrderedPerDay[startCell.getDay()] = startCell.getGasPurchased();
63          startCell = startCell.getParent();
64      }
65  }
66
67  public class TableEntry {
68      private int day;
69      private int gasPurchased;
70      private int cost;
71      private TableEntry parent;
72
73      public TableEntry(int day, int gasPurchased, int cost, TableEntry parent) {
74          this.day = day;
75          this.gasPurchased = gasPurchased;
76          this.cost = cost;
77          this.parent = parent;
78      }
79
80      public int getDay() {
81          return day;
82      }
83
84      public int getGasPurchased() {
85          return gasPurchased;
86      }
87
88      public int getCost() {
89          return cost;
90      }
91
92      public TableEntry getParent() {
93          return parent;
94      }
95  }
96
97
```