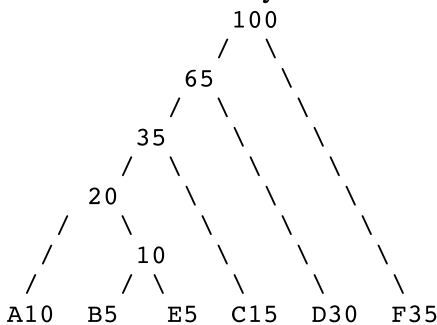
Solution for CISC 365 Test 4, Fall 2010

1. No, greedy is not optimal. Consider this counterexample.
Task 1 starts at 1, ends at 3 (length 2)
Task 2 starts at 4, ends at 6 (length 2)
Task 3 starts at 2, ends at 5 (length 3)
Greedy selects task 3, and thereby eliminates both task 1 and 2; total time is 3 hours.
Optimal selects tasks 1 and 2, for a total time of 4 hours.

2. (a) Yes. The denominations are multiples of one another. If you start with a set of coins chosen according to the greedy method, then the only way to change the set of coins is to replace a 6 coin by smaller coins, or replace a 3 coin by pennies. Either way increases the coin count, so greedy is optimal.
(b) No. As a counterexample, select coins that total to a value of 8. Greedy first selects a 6 coin, and then must fill the remainder with two pennies, for a total of 3 coins. Optimal selects two coins of value 4; this outperforms the greedy algorithm.

3. There are two ways to make the tree, because of a tie at node value 35.

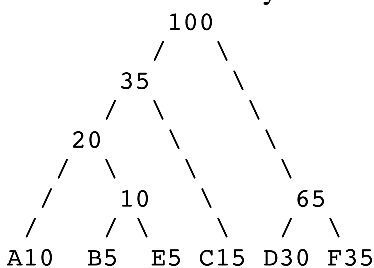


Using 0 on the left branch and 1 on the right, the code is

```
A 0000   B 00010   C 001
D 01     E 00011   F 1
```

The ones and zeros can be switched around, but all codes from this tree have 4 bits for A, 5 bits for B, 3 bits for C, 2 bits for D, 5 bits for E, 1 bit for F.

Here is a second way to make the tree, choosing the other “35” node first.



Using 0 on the left branch and 1 on the right, the code is

```
A 00   B 0010   C 01
D 10   E 0011   F 11
```

The ones and zeros can be switched, but all codes from this tree have 3 bits for A, 4 bits for B, 2 bits for C, 2 bits for D, 4 bits for E, 2 bits for F.

4. Here are some possible greedy strategies. The first three strategies are non-optimal; the four strategy produces an optimal solution.

- Schedule the task with largest penalty first. Sort the tasks by penalty and schedule them in that order. This can fail: a task with large penalty is placed needlessly early in the schedule, and that blocks another task that has an earlier deadline. For example

task	deadline	penalty
1	2	100
2	1	50

  Greedy uses task order 1, 2, for a penalty of 50.
  Optimal uses task order 2, 1, for a penalty of 0.

- Schedule the task with the earliest deadline first. Sort the tasks by deadline and schedule them in that order. This can fail: we may miss the deadline for a task that has a late deadline and very high penalty, because there are a bunch of tasks with earlier deadlines and low penalties. For example

task	deadline	penalty
1	1	5
2	1	5
3	2	100

  Greedy uses task order 1, 2, 3, for a penalty of 100.
  Optimal uses task order 3, 1, 2, for a penalty of 5.

- Sort by the ratio of penalty/deadline and schedule tasks with highest penalty/deadline ratio first. This can fail. For example

task	deadline	penalty
1	2	100
2	2	100
3	1	51

  Greedy uses task order 3, 1, 2, for a penalty of 100.
  Optimal uses task order 1, 2, 3, for a penalty of 51.

- The following greedy strategy works. Proceed by placing large-penalty tasks at or before their deadline, whenever possible. In detail, find schedule-spots for the tasks in order from largest-penalty task to smallest-penalty task. When finding a schedule-spot for a task: put the task at its deadline, or if that spot is full then use the first empty spot before its deadline. If all of the spots at and before the deadline are full, then we have to incur the penalty for this task, and we place the task at the very end of the schedule (into the empty spot that is closest to the end of the schedule). Placing the penalty tasks this way ensures that they do not needlessly block the on-time completion of any other task we schedule later. The code for this algorithm is on the next page.

```
// The schedule is created in array “schedule”.  Initially,
// schedule[j]=-1 for 1<=j<=n.  This indicates that slot j (hour j)
// hasn’t had a task assigned to it yet.
// When task i is scheduled for hour j, we set schedule[j]=i.

int schedule[1..n] = -1;  // initialize all entries to -1
int penalty[1..n];       // the penalities (given as input)
int deadline[1..n];      // the deadlines (given as input
int totalpenalty = 0;;  // the total penalty for this schedule

for (i=1 to n) {
    // Find the next task to schedule.  That’s the unscheduled task
    // with highest penalty.
    int maxpenalty = -1;
    int nexttask = -1;  // the task we will schedule next
    for (j=1 to n) {
        if (penalty[j]>maxpenalty) {
            maxpenalty=penalty[j];
            nexttask=j;
            }
        }

    // Schedule nexttask.  Put it as late as possible, but before its
    // deadline.  If all those spots are taken, then put it at the end.
    int spot = deadline[nexttask];
    while ((schedule[spot] != -1) and (spot > 0))
        spot = spot-1;
    if (spot==0) {
        // Nexttask cannot make its deadline, so put nexttask
        // at the end of the schedule.
        totalpenalty = totalpenalty + penalty[nexttask];
        spot = n;
        while (schedule[spot] != -1)
            spot = spot-1;
        }
        schedule[spot] = nexttask;
        penalty[nexttask]=-1;  // indicate that we have already
                               // scheduled this task.
    }

// The answer is in array schedule[1..n], with penalty equal
// to totalpenalty.
```