

## 6.00: Introduction to Computer Science and Programming

# Problem Set 5: Financing a Presidential Campaign

Handed out: Tuesday, October 16, 2007.

DUE:

- **Problems #1 AND #2: 11:00am Friday, October 19, 2007.**
- **Problem #3: 11:00am Tuesday, October 23, 2007.**

## Introduction

Your best friend is running for president of the United States. Your friend worked really hard to raise money for her campaign. She has a list of states where she can potentially win. Each state has some number of electoral votes. Your friend wants to maximize the total number of votes she can get. Each state also has a different population. In order to win the electoral votes of a state, your friend needs to spend at least \$1 per person in that state promoting herself. She needs your help to figure out in what states to run her campaign to get the most electoral votes on her budget.

### Workload

Please let us know how long you spend on each problem. We want to be careful not to overload you by giving out problems that take longer than we anticipated.

## Part 1

### Problem #1: Useful functions

Your friend was nice enough to provide you with information about all states in a Python-friendly format: **ps5.py** contains a skeleton for your assignment. It defines a `state_list` variable as a tuple of dictionaries. In this problem we'll learn to use that data structure. Please download `ps5.py` and add your code to that.

- **A)** Write a function that takes a list of states as a parameter and returns the total population in these states.

```
>>> total_pop(state_list[:10])
85250095
```

- **B)** Write a function that takes a list of states as a parameter and returns the total number of electoral votes in these states.

```
>>> total_evotes(state_list[:23])
317
```

- **C)** Write a function that takes a list of states as a parameter and returns the state dictionary with the smallest population per electoral vote. If multiple states have the same population per electoral vote, returning any of them is correct.

```
>>> cheapest_evotes(state_list[10:21])
{'evotes': 4, 'name': 'Hawaii', 'pop': 1285498}
```

### Problem #2: Exhaustive enumeration

Your friend, being the hardworking person she is, already started to write a Python program that tries all possible combinations of states under the budget to find the combination with the most electoral votes. However, she got distracted with other more important campaign matters and never finished the program.

**Complete and test this function according to her specification below.** Note that she was using exhaustive enumeration to compute the result. Your friend assured you that the function needs no more than 5 additional lines of code.

```
def finance_campaign_ee(budget, free_states):
    """
    Takes a budget, in dollars, and a list of available states, as a
    list of dictionaries.
```

```

Computes and returns the list of states that maximizes the total
number of electoral votes such that these states can be acquired
on the budget. Returns an empty list if no states can be acquired.
"""
cheap_states=[]
for s in free_states:
    if s['pop'] <= budget:
        cheap_states.append(s)

# Base case
if len(cheap_states)==0:
    res_list=[]
# Recursive case
else:
    curr_state=cheap_states[0]
    other_states=cheap_states[1:]

    inc_states=finance_campaign_ee( budget-curr_state['pop'],
                                   other_states)

    inc_states.append(curr_state)
    inc_evotes=total_evotes(inc_states)

    excl_states=finance_campaign_ee( budget, other_states )
    excl_evotes=total_evotes(excl_states)

    # ... your code goes here...

return res_list

```

**Note:** the exhaustive enumeration method attempts to examine all possibilities. That may take a long time for a large number of states. When debugging your code, make sure to use only part of the list as your input.

Sample output:

```

>>> finance_campaign_ee(18000000, state_list[:10])
[{'evotes': 10, 'name': 'Minnesota', 'pop': 5167101},
 {'evotes': 7, 'name': 'Iowa', 'pop': 2982085},
 {'evotes': 3, 'name': 'South Dakota', 'pop': 781919},
 {'evotes': 13, 'name': 'Virginia', 'pop': 7642884},
 {'evotes': 3, 'name': 'Wyoming', 'pop': 515004}]

```

Once you're done, you can run `print_finance_campaign_ee(budget, free_states)`, which pretty-prints the output of `finance_campaign_ee()` with some additional information, like so:

```

>>> print_finance_campaign_ee(50000000, state_list[:10])
States where to run the campaign:
Michigan, Minnesota, Iowa, South Dakota, New York, Virginia, Arkansas, Wyoming
Total number of electoral votes acquired: 90
Total budget required: 49301691

```

## Part 2

### Problem #3: Branch and Bound

Unfortunately, the exhaustive enumeration approach used by your friend takes too long to compute the results (try using a budget of \$90,000,000 on the complete state list). The computation takes so long because it examines all possible combinations of states: 2 to the power of the number of states, to be exact. That is unacceptable in the fast-paced environment of presidential elections.

**Implement a faster version of the above function using the branch and bound method. Complete the function below:**

```

def finance_campaign_bb(budget, free_states):
    """
    Takes a budget, in dollars, and a list of available states, as a
    list of dictionaries.

    Computes and returns the list of states that maximizes the total
    number of electoral votes such that these states can be acquired
    on the budget. Returns an empty list if no states can be acquired.
    """

```

```
"""
# ... your code goes here...
```

Hints:

- Be sure to understand how `finance_campaign_ee()` from the previous problem works.
- Remember that the key insight of branch and bound is pruning suboptimal combinations early.
- Feel free to define a helper function to make things easier.
- Test the correctness of your code by comparing results with your exhaustive enumeration implementation.
- Running `finance_campaign_bb(90000000, state_list)` should take less than a minute on your computer.

Now we'll compare the runtime of this implementation to that of the previous exhaustive enumeration approach.

**Add code to `print_finance_campaign_ee` which measures and prints the time required to do the exhaustive-enumeration computation.**

When you are done, the output of `print_finance_campaign_ee` might look like the following:

```
>>> print_finance_campaign_ee(50000000, state_list[:10])
States where to run the campaign:
  Michigan, Minnesota, Iowa, South Dakota, New York, Virginia, Arkansas, Wyoming
Total number of electoral votes acquired: 90
Total budget required: 49301691
Execution time: 0.06407 seconds
```

**Write a new function which measures the time needed for the branch-and-bound implementation. Compare the runtimes of the two implementations.**

## Hand-in Procedure

### 1. Save

All your code should be in a single file called `ps5.py`.

### 2. Time and collaboration info

At the start of the file, in a comment, write down the number of hours (roughly) you spent on this problem set, and the names of whomever you collaborated with. For example:

```
# Problem Set 5
# Name: Jane Lee
# Collaborators: John Doe
# Time: 1:30
#
```

### 3. Submit

Upload the file to your workspace. If there is some error uploading to your workspace, email the file to the 6.00 staff.

You may upload (or email) new versions of the problem set until the **11am deadline**, but anything uploaded after that will be ignored.