Indian Institute of Technology Kharagpur

**CS29003: Algorithms Laboratory, Spring 2021**

# Assignment 2: Recursive Formulation of Algorithms

2PM – 5PM 19th January, 2021

---

**General Instructions (to be followed strictly)**

Submit a single C/C++ source file.
Do not use global variables unless you are explicitly instructed so.
Do not use Standard Template Library (STL) of C++.
Use proper indentation in your code and comment.
Name your file as <roll_no>_<assignment_no>.
Write your name, roll number, and assignment number at the beginning of your program.

---

Consider the following model for an election. There are $p$ parties, each represented by a candidate. The winning party is decided by simple majority. That is, a party with strictly more than 50% of the votes wins. A winning party, if at all it exists, can form the government. It is possible that no single party wins the election. In that case, 2 or more parties whose vote shares together add up to more than 50% of the votes, can form an alliance/coalition government.

Your task for this assignment is as follows. Read the number of voters $v$ and the number of parties $p$ from the user. Both $v$ and $p$ are positive integers. Let the parties be labelled $1, 2, \ldots, p$. Also input a coalition $C \subseteq \{1, 2, \ldots, p\}$ (this could also be a singleton set). Generate all possible distributions of $v$ votes amongst the $p$ parties. For each distribution, check whether or not it is a winning distribution for the given coalition $C$. Print only the winning distributions of votes for $C$, each followed by the total number of votes obtained by the coalition. (Note that votes are indistinguishable, that is, the order of the votes or who votes for which party does not matter here; only the number of votes per party does.)

(a) Think of a data structure to store each distribution of votes. You may consider using character array.

(b) Write a function *vote_dist* that *recursively* generates all vote distributions. Here is one possible recursive formulation: we distribute the votes to parties $1, 2, \ldots, p$ starting from the left. For the first vote, there are two choices – either it goes to party 1 or party 2. In the first case, we recursively count the number of ways to ditributet $v - 1$ votes amongst $p$ parties and in the latter case, we count number of ways to distribute $v$ votes amongst the remaining $p - 1$ parties. This strategy will generate all possible distributions. More precisely, suppose that $i_v$ votes have been distributed so far among the first $i_p$ parties. Here, $i_p \in \{1, 2, \ldots, p-1\}$ and at this point some or all of first $i_p$ may not have any votes at all. The $(i_v+1)$-th vote either goes to party $i_p$ or to party $i_p+1$. In both cases, call the function recursively with the variables keeping track of $i_v$ and $i_p$ appropriately updated. Decide on an appropriate prototype for the function.

(c) Write a function *print* that given a string or any other data structure representing a vote distribution, decides whether or not it is a winning distribution for the coalition $C$ and if so, prints the number of votes for each party followed by the total number of votes for the coalition.

In the *main()* function,

- Read $v, p$ from the user. Assume (and ensure) that $v \leq 30$ and $p \leq 10$.
- Read the coalition as a subset of $\{1, 2, \ldots, p\}$ – the user enters numbers present in the subset one by one and then enters -1 to indicate end of the subset.
- Call *vote_dist* with the required parameters.

Note that the function *print* should be called from within *vote_dist* whenever you reach the terminating case for the recursion.

**Sample Output 1**

```
#Voters: 7
#Parties: 3

Coalition:
1
3
-1

 1: 7   2: 0   3: 0    Total votes for the coalition: 7
 1: 6   2: 1   3: 0    Total votes for the coalition: 6
 1: 6   2: 0   3: 1    Total votes for the coalition: 7
 1: 5   2: 2   3: 0    Total votes for the coalition: 5
 1: 5   2: 1   3: 1    Total votes for the coalition: 6
 1: 5   2: 0   3: 2    Total votes for the coalition: 7
 1: 4   2: 3   3: 0    Total votes for the coalition: 4
 1: 4   2: 2   3: 1    Total votes for the coalition: 5
 1: 4   2: 1   3: 2    Total votes for the coalition: 6
 1: 4   2: 0   3: 3    Total votes for the coalition: 7
 1: 3   2: 3   3: 1    Total votes for the coalition: 4
 1: 3   2: 2   3: 2    Total votes for the coalition: 5
 1: 3   2: 1   3: 3    Total votes for the coalition: 6
 1: 3   2: 0   3: 4    Total votes for the coalition: 7
 1: 2   2: 3   3: 2    Total votes for the coalition: 4
 1: 2   2: 2   3: 3    Total votes for the coalition: 5
 1: 2   2: 1   3: 4    Total votes for the coalition: 6
 1: 2   2: 0   3: 5    Total votes for the coalition: 7
 1: 1   2: 3   3: 3    Total votes for the coalition: 4
 1: 1   2: 2   3: 4    Total votes for the coalition: 5
 1: 1   2: 1   3: 5    Total votes for the coalition: 6
 1: 1   2: 0   3: 6    Total votes for the coalition: 7
 1: 0   2: 3   3: 4    Total votes for the coalition: 4
 1: 0   2: 2   3: 5    Total votes for the coalition: 5
 1: 0   2: 1   3: 6    Total votes for the coalition: 6
 1: 0   2: 0   3: 7    Total votes for the coalition: 7
```