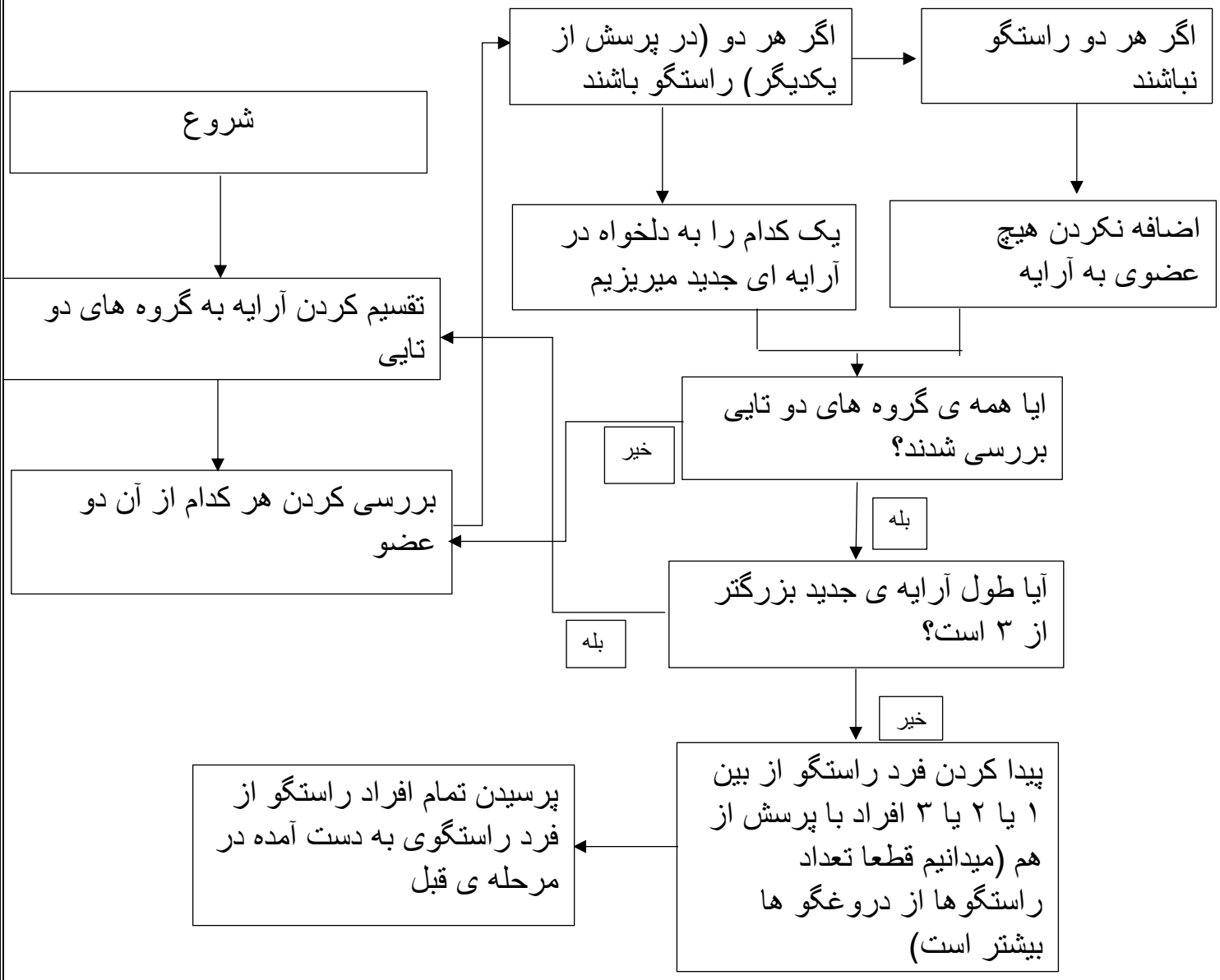
DivideAndConquer_Toidentify_a_knight_from_n_people

پروژه ی شوالیه

فلو چارت الگوریتم :



نحوه ی کار الگوریتم :

ابتدا تعداد کل افراد را از کاربر میگیریم . سپس به طور رندوم و به طوری که تعداد افراد راستگو بیشتر از نصف افراد باشد ، افراد راستگو و دروغگو را بوجود می اوریم . سپس به دلیل راحتی کار اگر طول ارایه توانی از ۲ نباشد، طول ان را با اضافه کردن مقداری عضو بی تاثیر به توانی از ۲ تبدیل مبکنیم . سپس یا استفاده از الگوریتم تقسیم و غلبه ، ان ها را به گروه های دوتایی تقسیم می کنیم . در هر گروه بررسی میکنیم : اگر هر دو عضو(هنگام پرسش از یک دیگر) راستگو خطاب شوند ، یک کدام را به عنوان رندوم به ارایه ای جدید می افزاییم . در غیر این صورت (یک نفر دروغگو و یک راستگو، دو دروغگو،دوبی تاثیر) هیچ کدام را نمی افزاییم . حال اگر یکی بی تاثیر و دیگری غیر بی تاثیر (راستگو یا دروغگو) باشد ، فرد دومی را به ارایه نیز می افزاییم . حال اگر طول ارایه ی جدید به دست امده بزرگتر از سه بود ، دوباره ارایه را به تابع پاس میدهیم و مراحل بالا تکرار می شوند . در غیر این صورت از ارایه ی نهایی (ارایه با طول کمتر مساوی ۳) به راحتی می توانیم با پرسش و با O(1) ، یکی از افراد راستگو را تشخیص دهیم . سپس چون این فرد راستگو است ، با O(n) بقیه ی راستگویان را تشخیص می دهیم .

توضیح کد :

- تابع only_one :

این تابع برای زمانی است که کاربر تعداد کاربران را ۱ وارد کند ... واضح است که در آن صورت تنها یک شوالیه داریم .

- تابع only_two :

این تابع برای زمانی است که کاربر تعداد کاربران را ۲ وارد کند ... واضح است که در آن صورت تنها دو شوالیه داریم .

- تابع liar_answer :

این تابع به صورت رندوم true و false تولید میکند . این تابع هم در زمان تعریف شوالیه های راستگو و دروغگو و هم در هنگام پرسش از فردی دروغگو مورد استفاده قرار می گیرد .

- تابع difining_humans :

این تابع برای تعریف شوالیه های راستگو و دروغگو به صورت رندوم به کار می رود . به نحوی که ابتدا تعداد کل افراد توسط کاربر دریافت می شود، سپس یک عدد بیشتر از نصف کل افراد را به طور رندوم پیدا می کنیم . واضح است که باقی افراد دروغگویان هستند . حال با توجه به تعداد شوالیه ها و دروغگوها ، آن ها را به طور رندوم تولید میکنیم و سپس به لیست تمام افراد می افزاییم . در نهایت ما لیستی از افراد راستگو و دروغگو به طور رندوم اما با شرایط صورت سوال (بیشتر از نصف بودن راستگویان) داریم .

- تابع convert_to_power_2 :

این تابع برای راحتی کار در قسمت تقسیم و حل به کار می رود . چون الگوریتم ما به این صورت است که باید هر مرحله تمامی افراد را به صورت گروه های دو دویی تقسیم کنیم ، اگر طول ارایه ی ما توانی از ۲ باشد ، هیچ کدام از عضو ها تکی باقی نمی مانند و کار ساده تر می شود . بنابراین طول ارایه را با اضافه کردن مقداری عضئ خنثی به نزیک ترین توان دو تبدیل میکنیم .

(عضو خنثی را به این صورت نمایش میدهیم : "-")

- تابع find_knight :

این تابع در انتهای الگوریتم به کار می رود . زمانی که ارایه ی نهایی ما حاوی ۱ یا دو یا سه عضو می باشد . در این زمان چون قطعا طبق الگوریتم تعداد راستگویان بیشتر از دروغ گویان است، به راحتی می توان فرد راستگو را تشخیص داد .

- تابع find_one_knight :

همانطور که از اسم تابع کاملا واضح است ، طبق الگوریتم ،این تابع به صورت تقسیم و حل ، ارایه را تا زمانی که به گروه ها دوتایی تقسیم شود ، هی به دو نیم تبدیل می کند ، (مانند الگوریتم merge sort) حال کافی است هر کدام از گروه ها را بررسی کنیم.

- تابع ask_each_other :

در این تابع از گروه های دو دویی سوال می شود ایا دیگری راستگو یا دروغگو است . اگر هر دو راستگو باشند ، یک کدام به ارایه اضافه میشود . همچنین اگر یکی فرد و دیگری خنثی باشد،در غیر این دو صورت هیچ کدام اضافه نخواهند شد .

- تابع find_knight_from_two :

این تابع گروه های دودویی به دست امده از تابع find_one_knight را شکل می دهد و برای بررسی هر کدام به تابع ask_each_other پاس می دهد .

- تابع find_other_knights :

این تابع در مرحه نهایی است . یعنی زمانی که یک فرد راستگو توسط الگوریتم تقسیم و غلبه به دست امده . حال از ان فرد راستگو درباره ی دیگر عضو ها سوال میکنیم که ایا راستگو هستند یا خیر و جواب را چاپ می کنیم .

- قسمت main :

در این قسمت ما بعد از تشکیل راستگوها و دروغگوها ، ان ها را تا زمانی که طول ارایه ی جدیدمان از ۳ بیشتر است را در یک حلقه به تابع پاس می دهیم . لازم به ذکر است که هر بار که ارایه ی جدیدی دریافت میکنیم (با طول بیشتر از ۳)، طول ان را برای راحتی کار به یک عدد توان ۲ می رسانیم .