Secret santa algorithm

Asked 16 years, 1 month ago Modified 7 years ago Viewed 22k times

26

Every Christmas we draw names for gift exchanges in my family. This usually involves mulitple redraws until no one has pulled their spouse. So this year I coded up my own name drawing app that takes in a bunch of names, a bunch of disallowed pairings, and sends off an email to everyone with their chosen giftee.

Right now, the algorithm works like this (in pseudocode):

```
function DrawNames(list allPeople, map
disallowedPairs) returns map
    // Make a list of potential candidates
    foreach person in allPeople
        person.potentialGiftees = People
        person.potentialGiftees.Remove(person)
        foreach pair in disallowedPairs
            if pair.first = person
               person.Remove(pair.second)

    // Loop through everyone and draw names
    while allPeople.count > 0
        currentPerson =
allPeople.findPersonWithLeastPotentialGiftees
        giftee =
pickRandomPersonFrom(currentPerson.potentialGiftees)
        matches[currentPerson] = giftee
        allPeople.Remove(currentPerson)
        foreach person in allPeople
            person.RemoveIfExists(giftee)

    return matches
```

Does anyone who knows more about graph theory know some kind of algorithm that would work better here? For my purposes, this works, but I'm curious.

EDIT: Since the emails went out a while ago, and I'm just hoping to learn something I'll rephrase this as a graph theory question. I'm not so interested in the special cases where the exclusions are all pairs (as in spouses not getting each other). I'm more interested in the cases where there are enough exclusions that finding any solution becomes the hard part. My algorithm above is just a simple greedy algorithm that I'm not sure would succeed in all cases.

Starting with a complete directed graph and a list of vertex pairs. For each vertex pair, remove the edge from the first vertex to the second.

The goal is to get a graph where each vertex has one edge coming in, and one edge leaving.

algorithm language-agnostic graph-theory

edited Nov 7, 2008 at 21:44

asked Nov 7, 2008 at 20:34

Eclipse

45.5k ● 20 ● 116 ● 172

9 Answers

Sorted by: Highest score (default)

11

Just make a graph with edges connecting two people if they are allowed to share gifts and then use a perfect matching algorithm. (Look for "Paths, Trees, and Flowers" for the (clever) algorithm)

Share Improve this answer

Follow

answered Nov 7, 2008 at 22:14

Watson Ladd

186 • 2

Not quite: it's a directed graph, and you don't necessarily want a perfect matching; you just want a subgraph such that every vertex has indegree = outdegree = 1 -- a *cycle cover*. [There are ways to reduce it to a perfect matching problem, but there are also ways to solve it directly.] – ShreevatsaR Nov 8, 2008 at 3:52

@ShreevatsaR: The graph doesn't have to be directed. Just flip a coin to choose a direction for each cycle. (This is assuming that all blacklisted pairs are symmetric.) – Chris Conway Dec 26, 2008 at 5:12

P.S. Treating the graph as undirected rules out any 2-cycles, which is probably desirable. – Chris Conway Dec 26, 2008 at 5:13

What if the black-listed pairs are not symmetric? Is there an algorithm for that? – Adam Dec 14, 2010 at 19:55

10

I was just doing this myself, in the end the algorithm I used doesn't exactly model drawing names out of a hat, but it's pretty damn close. Basically shuffle the list, and then pair each person with the next person in the list. The only difference with drawing names out of a hat is that you get one cycle instead of potentially getting mini subgroups of people who only exchange gifts with each other. If anything that might be a feature.

Implementation in Python:

```
import random
from collections import deque
def pairup(people):
    """ Given a list of people, assign each one a
secret santa partner
    from the list and return the pairings as a
dict. Implemented to always
    create a perfect cycle"""
    random.shuffle(people)
    partners = deque(people)
    partners.rotate()
    return dict(zip(people,partners))
```

Share Improve this answer

Follow

answered Nov 19, 2008 at 21:43

wxs

5,837 ● 5 ● 39 ● 52

6

I wouldn't use disallowed pairings, since that greatly increases the complexity of the problem. Just enter everyone's name and address into a list. Create a copy of the list and keep shuffling it until the addresses in each

position of the two lists don't match. This will ensure that no one gets themselves, or their spouse.

As a bonus, if you want to do this secret-ballot-style, print envelopes from the first list and names from the second list. Don't peek while stuffing the envelopes. (Or you could just automate emailing everyone thier pick.)

There are even more solutions to this problem on this thread.

Share Improve this answer

Follow

edited May 23, 2017 at 11:46

Community Bot

1 ● 1

answered Nov 7, 2008 at 20:37

Bill the Lizard

405k ● 211 ● 572 ● 889

The program just sends off an email to everyone, so the secrecy issue isn't a problem. – Eclipse Nov 7, 2008 at 20:54

That was one of the options I mentioned. – Bill the Lizard Nov 7, 2008 at 21:00

5

Hmmm. I took a course in graph theory, but simpler is to just randomly permute your list, pair each consecutive group, then swap any element that is disallowed with another. Since there's no disallowed person in any given pair, the swap will always succeed if you don't allow

swaps with the group selected. Your algorithm is too complex.

Share Improve this answer

Follow

answered Nov 7, 2008 at 20:40

Brian

25.8k ● 18 ● 86 ● 178

> The person and their spouse would both be disallowed, so the swap isn't guaranteed to work. – Bill the Lizard Nov 7, 2008 at 21:02

> Not true, because the group selected would have both the person and their spouse (otherwise no swap would be needed). – Brian Nov 8, 2008 at 20:09

2

Create a graph where each edge is "giftability" Vertices that represent Spouses will NOT be adjacent. Select an edge at random (that is a gift assignment). Delete all edges coming from the gifter and all edges going to the receiver and repeat.

Share Improve this answer

Follow

answered Nov 10, 2010 at 20:57

Eddie Rowe

21 ● 1

> Wouldn't this create an imperfect result? What if a gifter had a preferred giftee? – Dimitris Sfounis Dec 6, 2014 at 1:36

There is a concept in graph theory called a Hamiltonian Circuit that describes the "goal" you describe. One tip for

2

anybody who finds this is to tell users which "seed" was used to generate the graph. This way if you have to re-generate the graph you can. The "seed" is also useful if you have to add or remove a person. In that case simply choose a new "seed" and generate a new graph, making sure to tell participants which "seed" is the current/latest one.

Share Improve this answer

Follow

edited Dec 31, 2011 at 10:53

McDowell

109k ● 31 ● 206 ● 270

answered Dec 31, 2011 at 6:33

dana

18.1k ● 7 ● 68 ● 90

1

I just created a web app that will do exactly this - http://www.secretsantaswap.com/

My algorithm allows for subgroups. It's not pretty, but it works.

Operates as follows:

1. assign a unique identifier to all participants, remember which subgroup they're in
2. duplicate and shuffle that list (the targets)
3. create an array of the number of participants in each subgroup
4. duplicate array from [3] for targets
5. create a new array to hold the final matches
6. iterate through participants assigning the first target

that doesn't match any of the following criteria:

A. participant == target

B. participant.Subgroup == target.Subgroup

C. choosing the target will cause a subgroup to fail in the future (e.g. subgroup 1 must always have at least as many non-subgroup 1 targets remaining as participants subgroup 1 participants remaining)

D. participant(n+1) == target (n +1)

If we assign the target we also decrement the arrays from 3 and 4

So, not pretty (at all) but it works. Hope it helps,

Dan Carlson

Share Improve this answer

Follow

answered Nov 28, 2008 at 15:39

dancarlson

11 ● 1

1

Here a simple implementation in java for the secret santa problem.

```
public static void main(String[] args) {
    ArrayList<String> donor = new
ArrayList<String>();
    donor.add("Micha");
    donor.add("Christoph");
    donor.add("Benj");
    donor.add("Andi");
    donor.add("Test");
    ArrayList<String> receiver =
(ArrayList<String>) donor.clone();

    Collections.shuffle(donor);
```

```
    for (int i = 0; i < donor.size(); i++) {
        Collections.shuffle(receiver);
        int target = 0;

if(receiver.get(target).equals(donor.get(i))){
            target++;
        }
        System.out.println(donor.get(i) + " => " +
receiver.get(target));
        receiver.remove(receiver.get(target));
    }
}
```

Share Improve this answer

Follow

answered Oct 11, 2012 at 12:50

suizo

531 ● 8 ● 25

0

Python solution here.

Given a sequence of `(person, tags)`, where tags is itself a (possibly empty) sequence of strings, my algorithm suggests a chain of persons where each person gives a present to the next in the chain (the last person obviously is paired with the first one).

The tags exist so that the persons can be grouped and every time the next person is chosen from the group most dis-joined to the last person chosen. The initial person is chosen by an empty set of tags, so it will be picked from the longest group.

So, given an input sequence of:

```
example_sequence= [
    ("person1", ("male", "company1")),
    ("person2", ("female", "company2")),
    ("person3", ("male", "company1")),
    ("husband1", ("male", "company2",
"marriage1")),
    ("wife1", ("female", "company1",
"marriage1")),
    ("husband2", ("male", "company3",
"marriage2")),
    ("wife2", ("female", "company2",
"marriage2")),
]
```

a suggestion is:

['person1 [male,company1]', 'person2 [female,company2]', 'person3 [male,company1]', 'wife2 [female,marriage2,company2]', 'husband1 [male,marriage1,company2]', 'husband2 [male,marriage2,company3]', 'wife1 [female,marriage1,company1]']

Of course, if all persons have no tags (e.g. an empty tuple) then there is only one group to choose from.

There isn't always an optimal solution (think an input sequence of 10 women and 2 men, their genre being the only tag given), but it does a good work as much as it can.

Py2/3 compatible.

```
import random, collections
```

```
class Statistics(object):
    def __init__(self):
        self.tags = collections.defaultdict(int)

    def account(self, tags):
        for tag in tags:
            self.tags[tag] += 1

    def tags_value(self, tags):
        return sum(1./self.tags[tag] for tag in
tags)

    def most_disjoined(self, tags, groups):
        return max(
            groups.items(),
            key=lambda kv: (
                -self.tags_value(kv[0] & tags),
                len(kv[1]),
                self.tags_value(tags - kv[0]) -
self.tags_value(kv[0] - tags),
            )
        )

def secret_santa(people_and_their_tags):
    """Secret santa algorithm.

    The lottery function expects a sequence of:
    (name, tags)

    For example:

    [
        ("person1", ("male", "company1")),
        ("person2", ("female", "company2")),
        ("person3", ("male", "company1")),
```

Share Improve this answer

edited Nov 24, 2017 at 18:05

Follow

answered Nov 24, 2017 at 18:00

tzot