Midterm 2 Review

CSCI 104

Counting

Important Rules:

- Product rule:
    - If procedure can be broken up into sequence of k tasks
    - n1 ways to do first task, n2 ways to do second task, nk ways to do kth task
    - **n_1 * n_2 * … * n_k ways to do the procedure**

- Sum rule:
    - If procedure can be done in n_1 ways OR n_2 ways
    - n_1 and n_2 have zero overlap
    - **$n_1 + n_2$ ways to do the task**

Counting

Important Rules (cont.):

- Subtraction rule:
  - If procedure can be done in n_1 ways OR n_2 ways
  - n_1 and n_2 have overlap n_3
  - **$n_1 + n_2 - n_3$ ways to do the task**

- Division rule:
  - If procedure can be done in n ways
  - For each way, it is identical to d-1 other ways
  - **n/d ways to do a task**

Counting

Important Rules (cont.):

- Permutation: **ordered** arrangement of r elements from a set of n

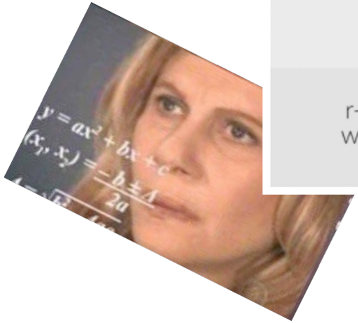
$$ {}_nP_r = \frac{n!}{(n-r)!} $$

- Combination: **unordered** arrangement of r elements from a set of n (n choose r)

$$ {}_nC_r = \binom{n}{r} = \frac{{}_nP_r}{{}_rP_r} = \frac{n!}{r!\,(n-r)!} $$

Counting

	Does order matter?	Is repetition allowed?	Formula
r-permutation without repetition	Yes	No	$\frac{n!}{(n-r)!}$
r-permutation with repetition	Yes	Yes	n^r
r-combination without repetition	No	No	$\frac{n!}{r!\,(n-r)!}$
r-combination with repetition	No	Yes	$\binom{n-1+r}{r}$



Counting

How many different ways are there to distribute 9 cookies to 4 children so that each child gets at least one cookie?

Counting

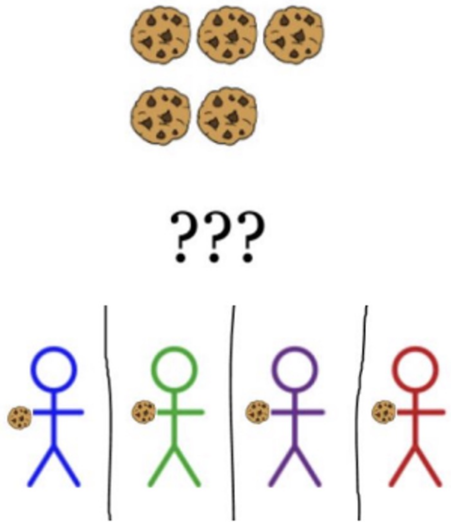
- The cookies are indistinguishable while the children are distinguishable, so Stars and Bars is a good option to use.
    - 9 stars = cookies
    - 3 bars to separate children

- Problem: Using the Stars and Bars equation will give sequences with children getting no cookies.
    - How do we make sure each child gets at least one cookie with this method?

Counting

- We can give each child a cookie, leaving us with 5 cookies (stars) remaining to distribute.

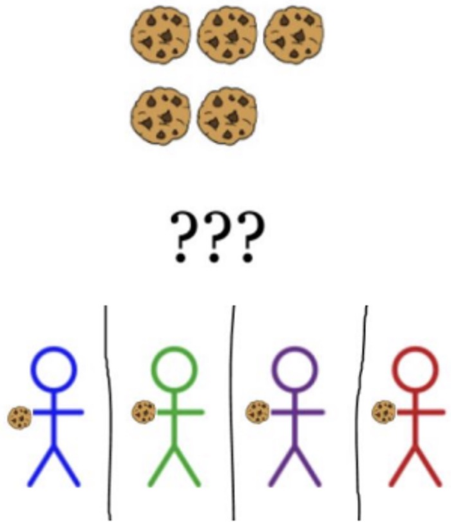
- More importantly, we now have no restrictions on how to distribute those 5 cookies since the given condition will always be fulfilled.



Counting

- We now have no restrictions on how to distribute those 5 cookies since the given condition will always be fulfilled.

- Now just need to use with the remaining 5 cookies and 3 bars, giving us:
  [(5 + 3) CHOOSE 3] = **56 ways**



Counting

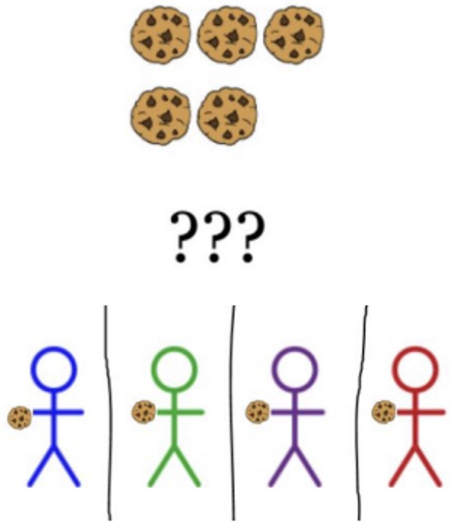
Now let's say the children are indistinguishable. How many different ways are there to distribute 9 cookies to 4 children so that each child gets at least one cookie?

Counting

- We will still give each student one cookie, so now we need to distribute 5 indistinguishable cookies over the 4 indistinguishable students.

- There is no formula for this situation.
    - (5, 0, 0, 0)
    - (4, 1, 0, 0), (3, 2, 0, 0)
    - (3, 1, 1, 0), (2, 2, 1, 0)
    - (1, 1, 1, 2)

- Total: **6 ways**



Probability

Important Rules:

- Probability of event E:
  - S = sample space of equally likely outcomes
  - **$P(E) = |E| / |S|$**

- Complement: probability that event does not occur
  - **$P(\bar{E}) = 1 - P(E)$**

Probability

Important Rules:

- Conditional Probability: probability of B given A

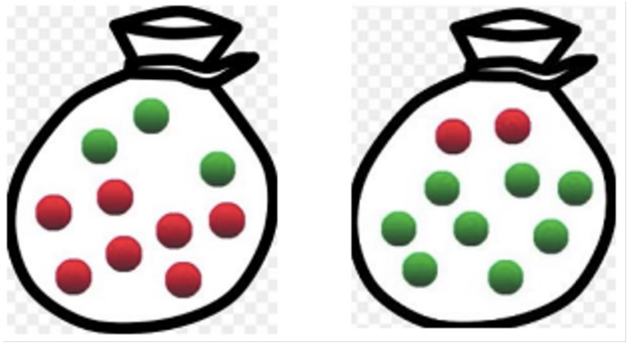
$$\boxed{P(B|A) = \frac{P(A \cap B)}{P(A)}}$$

  - If likelihood of B occurring does not depend on A, then B is independent of A:

$$\boxed{P(B \mid A) = P(B).}$$

Probability

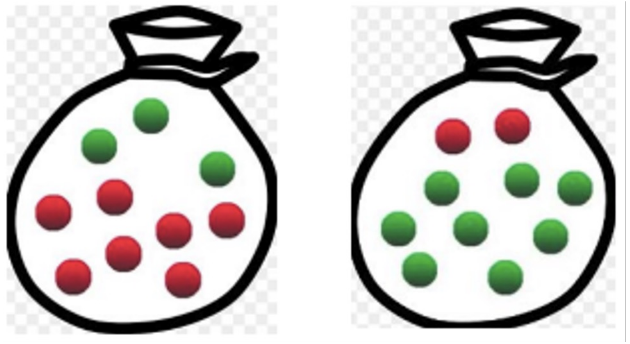
- Suppose there are two bags in a box, which contain the following marbles:
  - Bag 1: 7 red marbles and 3 green marbles.
  - Bag 2: 2 red marbles and 8 green marbles.



- If we randomly select one of the bags and then randomly select one marble from that chosen bag, what is the probability that it's a green marble?

Probability

- Green marble could come from Bag 1 or Bag 2, which will affect the chances of drawing a green marble



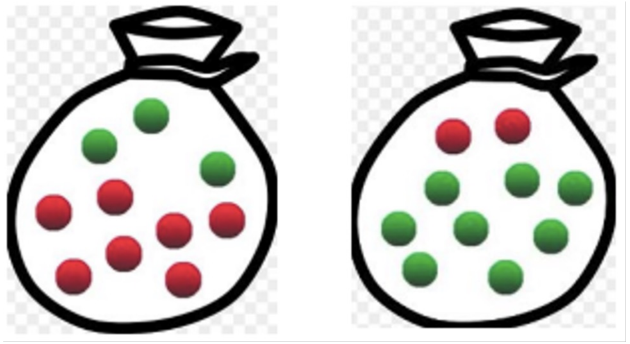
- We need to use the Law of Total Probability:
  - For any partition of the sample space into disjoint events $F_1, \ldots, F_k$:

    $$p(E) = p(E|F_1) * p(F_1) + \ldots + p(E|F_k) * p(F_k)$$

Probability

$$p(G) = p(G|B_1) * p(B_1) + p(G|B_2) * p(B_2)$$

- $p(B_1) = p(B_2) = 0.5$
- $p(G|B_1) = 3 / (7+3) = 3/10 = 0.3$
- $p(G|B_2) = 8 / (8+2) = 8/10 = 0.8$



$p(G) = (0.3 * 0.5) + (0.8 * 0.5)$

$p(G) = 0.15 + 0.40$

$p(G) = \mathbf{0.55}$

Hashtables

Consider a hash table of size 7 with a loading factor of 0.5, the resize function is 2n + 3, where n is the size of the hash table. *(an insertion may end with the loading factor being >= 0.5; the next insertion would cause the resize).*

When resizing, keys are inserted in the order they appear index-wise in the old hashtable.

Hashtables

The hash function is (3k + 4) % n, using quadratic probing. Insert 3, 11, 10, 6, 8, 23.

Key	**HashFunc(key)**	**Loading Factor Before Insert**	**Probe Sequence**
3	(9 + 4) % 7 = 6	0/7 = 0	6

Hashtables

The hash function is (3k + 4) % n, using quadratic probing. Insert 3, 11, 10, 6, 8, 23.

Key	HashFunc(key)	Loading Factor Before Insert	Probe Sequence
3	(9 + 4) % 7 = 6	0/7 = 0	6
11	(33 + 4) % 7 = 2	1/7 = 0.14	2

Hashtables

The hash function is (3k + 4) % n, using quadratic probing. Insert 3, 11, 10, 6, 8, 23.

Key	HashFunc(key)	Loading Factor Before Insert	Probe Sequence
3	(9 + 4) % 7 = 6	0/7 = 0	6
11	(33 + 4) % 7 = 2	1/7 = 0.14	2
10	(30 + 4) % 7 = 6	2/7 = 0.28	6 → 0

Hashtables

The hash function is (3k + 4) % n, using quadratic probing. Insert 3, 11, 10, 6, 8, 23.

Key	**HashFunc(key)**	**Loading Factor Before Insert**	**Probe Sequence**
3	(9 + 4) % 7 = 6	0/7 = 0	6
11	(33 + 4) % 7 = 2	1/7 = 0.14	2
10	(30 + 4) % 7 = 6	2/7 = 0.28	6 → 0
6	(18 + 4) % 7 = 1	3/7 = 0.42	1

Hashtables

The hash function is (3k + 4) % n, using quadratic probing. Insert 3, 11, 10, 6, 8, 23.

Key	**HashFunc(key)**	**Loading Factor Before Insert**	**Probe Sequence**
3	(9 + 4) % 7 = 6	0/7 = 0	6
11	(33 + 4) % 7 = 2	1/7 = 0.14	2
10	(30 + 4) % 7 = 6	2/7 = 0.28	6 → 0
6	(18 + 4) % 7 = 1	3/7 = 0.42	1
8		**4/7 = 0.57**	

Hashtables

The hash function is (3k + 4) % n, using quadratic probing. Insert 3, 11, 10, 6, 8, 23.

Key	**HashFunc(key)**	**LF**	**Probe**
3	6	0	6
11	2	0.14	2
10	6	0.28	6 → 0
6	1	0.42	1
8		**0.57**	

Move to new table →

New size is 2n + 3 = 17

Key	**HashFunc(key)**	**LF**	**Probe**
10	34 % 17 = 0	0	0
6	22 % 17 = 5	1/17	5
11	37 % 17 = 3	2/17	3
3	13 % 17=13	3/17	13
8	24 % 17 = 7	4/17	7

Hashtables

The hash function is (3k + 4) % n, using quadratic probing. Insert 3, 11, 10, 6, 8, 23.

Key	**HashFunc(key)**	**LF**	**Probe**
10	34 % 17 = 0	0	0
6	22 % 17 = 5	1/17	5
11	37 % 17 = 3	2/17	3
3	13 % 17=13	3/17	13
8	24 % 17 = 7	4/17	7
23	73 % 17 = 5	5/17	5 → 6

Hashtables

Final hashtable:

Hashtable index	**Key**
0	10
3	11
5	6
6	23
7	8
13	3

Hashtables

More questions to consider:

1. What are the benefits of double hashing over things like linear or quadratic probing?
2. No examples of a double collision came up. If there was a double collision, what index do we go to next?
3. Can you explain the benefits of resizing?
4. Are probes ever guaranteed to go to distinct locations? If yes, what are the conditions for this to happen?

Bloom Filters

Q: What are the benefits and drawbacks of using a bloom filter?

Bloom Filters

Q: What are the benefits and drawbacks of using a bloom filter?

A:

Benefit: avoids storage of keys (better space efficiency wise)

Drawback: Can be space inefficient if not implemented correctly, not always right…

Bloom Filters

Q: Which is possible, false positives or false negatives?

Bloom Filters

Q: Which is possible, false positives or false negatives?

A: **False positives!** We may accidentally say something exists in the hashtable if all bits are set, but we'll never accidentally say something is not there when it is

Why?

Because we would have set all the bits if we were inserting the item in the first place!

Bloom Filters

Q: Let’s say we have a bloom filter with 19 indices, 3 universal hash functions. 5 of the bits are set. What is the probability of getting a false positive?

Bloom Filters

Q: Let's say we have a bloom filter with 19 indices, 3 universal hash functions. 5 of the bits are set. What is the probability of getting a false positive?

A: Product rule, since the problem can be broken up into 3 "tasks"

hash function 1, 2, 3: all 5/19 chance of hitting set bit

5/19 * 5/19 * 5/19 = 1.8% chance!

Number Theory

Q: Say gcd(a, b) = 1 and gcd(a, c) = 1. What is gcd(a, b*c)?

Number Theory

Q: Say gcd(a, b) = 1 and gcd(a, c) = 1. What is gcd(a, b*c)?

A: **1**. Think about breaking a, b, and c into their prime factors. Since a is coprime with both b and c, when we multiply them together, we don’t gain any factor in the product that will magically make the gcd greater than 1!

Number Theory

Q: Is 257 prime?

Number Theory

Q: Is 257 prime?

A: sqrt(257) = 16 (roughly)

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 → prime!

Number Theory

Q: Is 257 prime?

A: sqrt(257) = 16 (roughly)

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 → prime!

Number Theory

Q: What is the ones digit of 7^{100}?

Number Theory

Q: What is the ones digit of 7^100?

A: Let’s try to find a pattern…

7^ 0 = 1

7^1 = 7

7^2 = 49 = 9

7^3 = 343 = 3

7^4 = 2401 = 1

7^5 = 16807 = 7

7^6 = 117649 = 9

Number Theory

Q: What is the ones digit of 7^100?

A: Pattern repeats in groups of 4: 1 7 9 3 → 1 7 9 3 → etc.

Which number is 100 in the pattern? (i.e. will it be 7 9 3 or 1?)

N = exponent

If n % 4 == 0, last digit 1

If n % 4 == 1, last digit 7

If n % 4 == 2, last digit 9

If n % 4 == 3, last digit 3

N = 100 → 100 % 4 = 0, **last digit is 1**

Number Theory

Q: Given that $5x \equiv 6 \pmod{8}$, find x.

Number Theory

Q: Given that $5x \equiv 6 \pmod{8}$, find x.

A: This means that 6 % 8 == 5x % 8

5x % 8 = 6. Need to find an x that will satisfy this.

Go through multiples of 5, think if remainder 8 == 6

5, 10, 15, 20, 25, 30

X = 6

Coding

Suppose you are given an integer array *nums* of unique elements. **Return all possible subsets** of the array (in other words, the power set). The solution can be in any order, and you must not include duplicate subsets.

```cpp
// Returns all subsets of nums
vector<vector<int>> subsets(vector<int>& nums) {

}
```

- Example:
  - Input: nums = [1,2,3]
  - Output: [[ ], [1], [2], [1,2], [3], [1,3], [2,3], [1,2,3]]
- How many subsets are possible?
  - Each element can be included or not included in the subset
  - For an array of n elements, this would be **2^n total subsets**

Coding

We need to explore all possible combinations of the array's elements -> backtracking!

- We will start building a subset that is initially empty
- We will iterate through the array and add the current number to our subset
    - Recursively build the subset without the letters we have already used (adjust the range of our loop)
    - Backtrack by removing the number we added and proceed to the next iteration of the loop
- In each recursive call, we will have a new subset that will be a part of our solution

Coding

Solution:

```cpp
// helper function for subsets
void dfs(vector<int>& nums, int start, vector<int>& curr, vector<vector<int>>& result) {
    result.push_back(curr);
    for (int i = start; i < nums.size(); i++) {
        curr.push_back(nums[i]);
        dfs(nums, i + 1, curr, result);
        curr.pop_back();
    }
}

// Returns all subsets of nums
vector<vector<int>> subsets(vector<int>& nums) {
    vector<int> curr;
    vector<vector<int>> result;
    dfs(nums, 0, curr, result);
    return result;
}
```