CS1113
Algorithm Analysis & Counting Rules

Lecturer:

Professor Barry O'Sullivan

Office: 2.65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

http://osullivan.ucc.ie/teaching/cs1113/

Introduction to analysing algorithms and data structures

making one choice from multiple sets (the sum rule)

making multiple choices from multiple sets (the product rule)

The story so far

- fundamental concept of computing: carrying out sequences of instructions
- 2nd most important concept: representing and manipulating information
- main challenge has been stating what you want, precisely and unambiguously

Review

We have looked at:

- the basic idea of an algorithm – an ordered deterministic executable terminating set of instructions
- how to describe collections of objects (sets)
- how to represent transformations of collections (functions)
- how to describe relationships between objects in the collections (relations)
- how to specify requirements (logic)
- how to prove solutions meet requirements, and that arguments are correct (logic)
- how to represent and use structured collections of objects (graphs)

and we have looked at examples in databases and specifications for computer systems and in path planning.

Assessing algorithms

In other modules, you are learning how to write algorithms in particular languages, how to represent information, and how to understand computer systems.

To judge how good our solutions are, we need to assess them:

- are our instructions correct?
- how long will it take to carry out the instructions?
- how much space do I need to represent this information?
- how many cases do I have to consider to solve a problem?
- what are good algorithms for different types of relations?
- how do I prove any claims I make?

We will start by learning how to count steps, choices and combinations.

Example: making a single choice

Suppose I want to pick a single student representative for all full-time registered undergraduate computer science students at UCC. Let's say there are 94 students registered for 1st year, 98 students registered for 2nd year, 76 students registered for third year, and 60 students registered for 4th year. Nobody is registered for more than one year (and everybody is registered in at least one of the four years).

How many different possible choices are there?

We simply add together the total number in each category.

94 + 98 + 76 + 60 = 328.

The Sum Rule

In general, if I have to select an object from *n* sets S_1, S_2, ..., S_n, where none of the sets have any elements in common, then there are $|S_1| + |S_2| + ... + |S_n|$ possible selections.

This is called the *sum rule*.

Expressing it in terms of set operations, we have:

> For sets S_1, S_2, ..., S_n such that $\forall i \, \forall j \neq i \; S_i \cap S_j = \{\}$, then $|S_1 \cup S_2 \cup ... \cup S_n| = |S_1| + |S_2| + ... + |S_n|$

Sum Rule and Computational Steps

Suppose I must complete all the actions from one set S_1, then all actions from another set S_2, and so on up to S_n, where all the actions are different. If each action takes 1 second, then the total computation will take $|S_1| + |S_2| + ... + |S_n|$ seconds.

Example: initialising a university records database

for each student X_1 to X_n
  add the student to the database
for each lecturer L_1 to L_m
  add the lecturer to the database
for each employee E_1 to E_t
  add the employee to the database

How many database additions do I need to make?

Example: making multiple choices

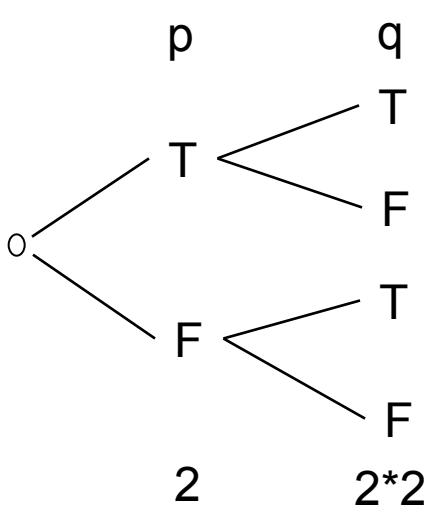
When building a truth table in propositional logic, how many rows do we need?

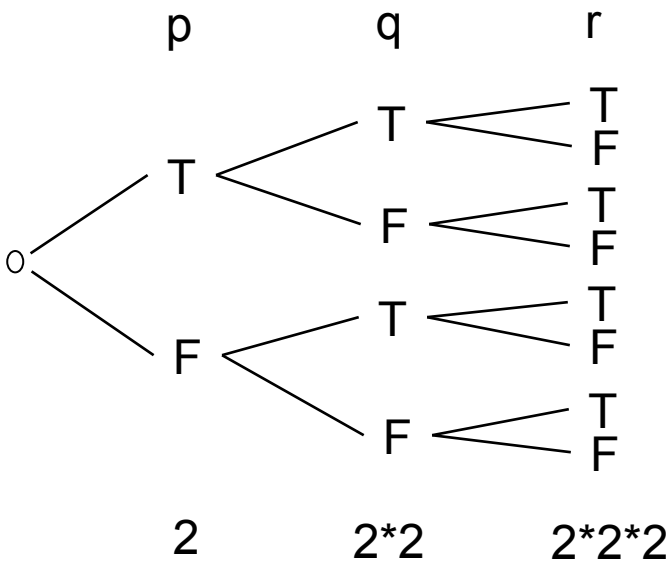
We have to consider all possible assignments of values to the variables. For each variable, there are only two possible values (T or F).

If the statement is $p \wedge q$, then there are two variables. There are two choices for the first variable, and for each of those, there are two choices for the second. Therefore there are 2*2 = 4 choices in total.

For $p \wedge q \wedge r$, there are three variables. So 2 choices for p, and for each of them, 2 choices for q, and for each of them, 2 choices for r. This gives 2*2*2 = 8 choices in total.

We can visualise this by drawing a tree of choices





p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

p	q	r	p∧q	(p∧q)∨r
T	T	T	T	T
T	T	F	T	T
T	F	T	F	T
T	F	F	F	F
F	T	T	F	T
F	T	F	F	F
F	F	T	F	T
F	F	F	F	F

Example

Suppose the department is considering a new system for assigning usernames to students. Each student will be given a two-letter name, consisting of lower-case characters only. How many usernames will this system allow?

There are 26 lower-case letters. Therefore, there are 26 ways to assign the first letter, and for each of these, there are 26 ways to assign the second letter. That means there are 26*26 = 676 possible usernames.

The Product Rule

In general, if I have to select one object from set A and one object from set B, where A has *n* elements and B has *m* elements, then there are *n*m* possible combined selections.

This is called the *product rule*: $|A \times B| = |A| * |B|$

In general, if I have to select one object from each of the sets S_1, S_2, ..., S_t, where set S_i has n_i elements, then there are $n_1 * n_2 * \ldots * n_t$ possible combined selections

$$|S_1 \times S_2 \times \ldots \times S_t| = |S_1| \times |S_2| \times \ldots \times |S_t|$$

We have seen this before: the size of the cartesian product

Example

The standard system architecture for PCs uses bytes consisting of 8 bits. That is, a byte is a sequence of 8 binary digits (i.e. 0s or 1s). How many different bytes are possible?

There are 2 choices for the first bit, and for each of those, 2 choices for the second, etc.,
so there are 2x2x2x2x2x2x2x2 = 2^8 = 256.

In 16-bit arithmetic, for representing integers, we use the first bit for the sign, which leaves 15 bits. That means we can represent 2^{15} positive integers. Since the first integer is 0, that means the largest positive integer we can represent is 2^{15}-1.
So "maxint" = 32767

Example 1

Suppose the computer system requires you to choose a password which consists of a lowercase letter, a digit, a digit, and a lowercase letter, in that order. How many possible passwords are there?

Example 2

Suppose we have two sets – A has 6 elements, and B has 5 elements. How many different functions can we specify from A to B?

Example 3

Suppose we have a set of 5 elements. How many possible subsets are there? In general, for any given set, how many possible subsets are there?

Example: counting actions in a program

```
for each student with id ranging from 1 to n
   obtain student with id=i from the database
   for each project with id ranging from 1 to m
      obtain project with id=j from the database
      if student_i worked on project j
         output (student_i, project_j) to the screen
```

How many times do I read details from the database?

If reading details from a database is a time-consuming operation, is there a better way of writing this algorithm?

Next lecture ...

More on counting combinations

CS1113
More Counting Rules

Lecturer:

Professor Barry O'Sullivan

Office: 2.65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

http://osullivan.ucc.ie/teaching/cs1113/

More principles for counting actions or choices

combining sum and product rules
inclusion and exclusion
factorials
permutations

pigeonhole principle

Example: combining sum and product

Module codes across University College Skibbereen must start with a department code of either 1 or 2 uppercase letters, and then have 3 digits, where the first digit must be either a 1, 2, 3 or 4. How many different module codes can be represented?

Example (sum rule revision)

"Module codes across University College Skibbereen must start with a department code of either 1 or 2 uppercase letters, and ...."

How many different department codes can be represented?

Sum rule:

$\forall i \, \forall j$ s.t. $j \neq i$ and $S_i \cap S_j = \{\}$,

$|S_1 \cup S_2 \cup ... \cup S_n| = |S_1| + |S_2| + ... + |S_n|$

There are 26 single letters, and 26*26 pairs of letters (using the product rule), so there are 26+676 = 702 possible letter codes

Example (product rule revision)

" ... and then have 3 digits, where the first digit must be either a 1, 2, 3 or 4. "

How many different module codes can be represented for a department?

Product rule: $|S_1 \times S_2 \times \ldots \times S_t| = |S_1| \times |S_2| \times \ldots \times |S_t|$

There are 4 choices for the 1st digit. There are 10 choices for the 2nd digit, and 10 choices for the 3rd digit.

Therefore, there are 4*10*10 = 400 possible module codes.

Example: combining sum and product

Module codes across University College Skibbereen must start with a department code of either 1 or 2 uppercase letters, and then have 3 digits, where the first digit must be either a 1, 2, 3 or 4. How many different module codes can be represented?

We have already seen by the sum rule that there are 702 possible department codes.

We have already seen by the product rule that there are 400 possible numerical codes for each dept.

So, by the product rule, there are 702*400 = 280800 possible module codes in total.

Example

How many UCC students have a 1st year computer science lecture on a Wednesday?

There are lectures for CS1111 and CS1113 on Wednesday, but for no other module. We have to count the number of students across 2 sets, so it looks like the sum rule.

BUT: there are 93 students registered for CS1111, 89 students registered for CS1113, and 85 students are registered for both modules.

So 93 students have a CS1111 lecture, and 89 students have a CS1113 lecture, giving 93+89= 182 entries on the sign-up sheets. But 85 of the students appear in both, so the total number of individual students is 93+89-85 = 97.

Principle of Inclusion and Exclusion

If I have to select one object from two sets A and B, then there are $|A| + |B| - |A \cap B|$ possible selections.

For sets A and B, $|A \cup B| = |A| + |B| - |A \cap B|$

If I have to select one object from 3 sets A, B and C, then there are $|A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$ possible selections.

For sets A,B and C,
$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$

There is a rule for arbitrarily many sets, but we will leave that until later.

Factorial

As our analysis of choices and algorithms gets more complex, we will need to introduce abbreviations for commonly occurring patterns.

The factorial of a number *n* is the multiplication of all the integers from *n* down to 1 and is written *n*!
(and so *n*! = *n**(*n*-1)*(*n*-2)*...*2*1)

3! = 3*2*1 = 6
4! = 4*3*2*1 = 4*(3!) = 24
5! = 5*4*3*2*1 = 5*4! = 120

Exercise: work out the value of the following:
(i) 2!
(ii) 6!

Example

Suppose we have three towns we want to visit: Tralee, Limerick and Waterford. The order in which we do the visits matters (travel time, etc). We will travel by the main roads connecting the towns, but we want to find the most efficient sequence. How many different routes do we need to consider before we are sure we have considered them all?

There are 6 possible routes:

<Tralee, Limerick, Waterford>
<Tralee, Waterford, Limerick>
<Limerick, Tralee, Waterford>
<Limerick, Waterford, Tralee>
<Waterford, Tralee, Limerick>
<Waterford, Limerick, Tralee>

Calculating the number of routes

For the previous example, we could have calculated the number of routes easily, without writing them all out, using the product rule.

There are three possible choices for the first town. Once we have fixed the first, there are two remaining choices for the second. Once we have fixed the second, there is only one town remaining. So by the product rule, there are 3*2*1 possible sequences.

Using the factorial notation, there are 3! possible sequences.

Permutations

If we have a set of *n* elements, then a sequence of those *n* elements (where each element appears exactly once) is a permutation of the set.

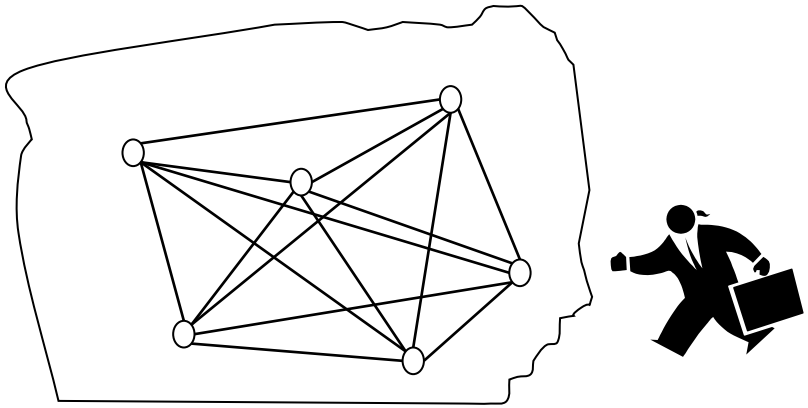
For a set with *n* elements, there are *n*! different possible permutations.

There are *n* choices for the first element, *n-1* for the second, *n-2* for the third, and so on, down to 1 choice for the last.
By the product rule, there are $n*(n-1)*(n-2)*...*1 = n!$ possible permutations.

Travelling Salesman problem

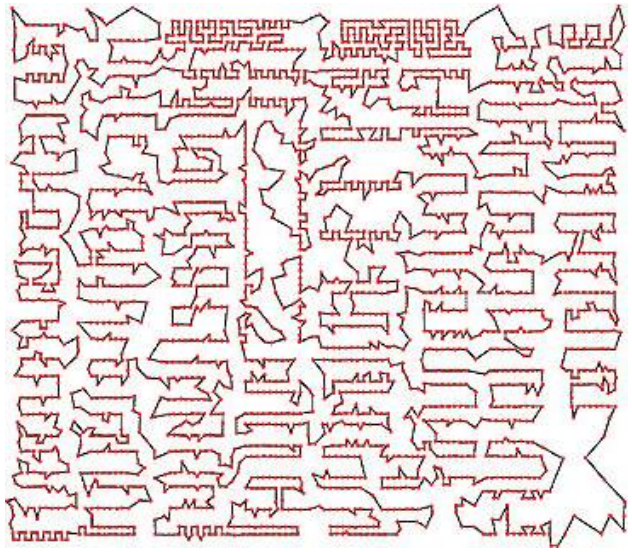
The example of finding routes is part of an important problem that re-appears through computing, known as the travelling salesman problem.

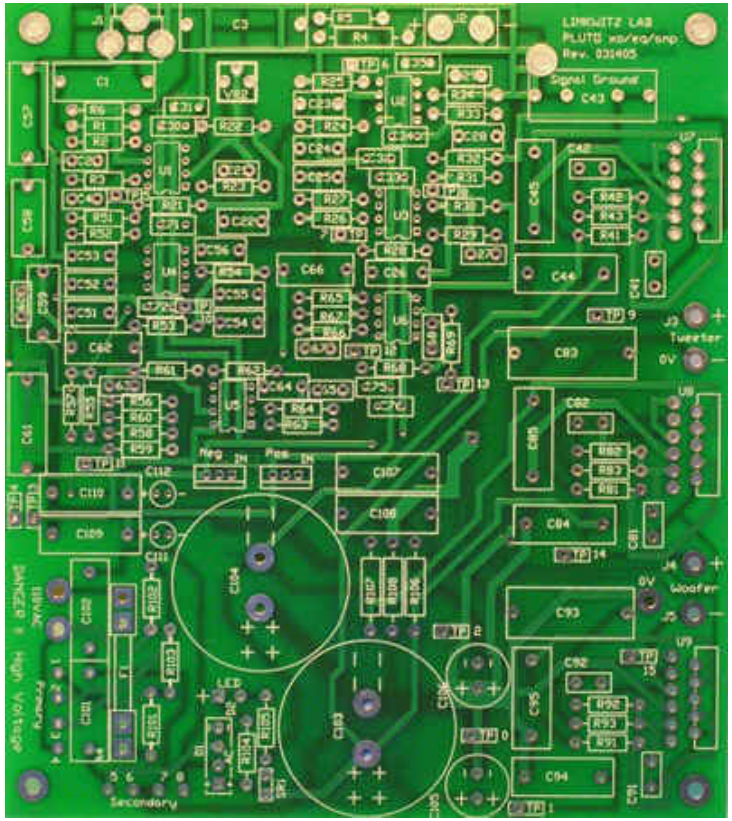
Given a set of cities, and a table showing the distance between any pair of cities, find the shortest route around the cities which does not visit any city twice (except the starting point). In other words, find the permutation which gives the shortest distance (or find the cheapest Hamiltonian Circuit).



Travelling Salesman Problem: Application

In the manufacture of printed circuit boards, each hole has to be drilled or stamped into the board, and each component fixed in place. What is the shortest path for the drill to move around the board?





How hard can it be to solve the TSP?

The obvious way to solve it is to generate each possible permutation, and see how expensive each one is.

But the number of permutations grows very quickly as the number of cities in the problem increases

Suppose we have 20 cities in the problem.
There are then 20! permutations.
$20! \sim 10^{18}$.

There have only been ~ 10^{17} seconds since the Big Bang.

Understanding how long an algorithm will take to run is an important part of software development.

TSP

The travelling salesman problem (TSP) has no known efficient algorithm – that is, as the number of cities increases, the time taken to find the best permutation increases exponentially, even for the best known algorithm. What is more, almost all computer scientists believe no efficient algorithm is possible.

There are many other problems that we can prove are similar to the TSP, and we can prove that if we find an efficient algorithm for one of them, we can do it for all of them.

Understanding the limits of computational power is important.

You will study these issues in 2nd year and 3rd year.

Example: sub-permutations

Suppose an web site wants to conduct a user survey. Each customer is asked to select the top four photographs from the site. There are 30 photographs in total. How many different responses could there be?

Each user has 30 options for the top photograph. For each one of those, there are 29 choices for the 2nd photo, then 28 choices for the 3rd, and 27 choices for the 4th.

So there are 30*29*28*27 = 657720 possible responses.

r-Permutations

An r-permutation is an ordered sequence of r different elements from a set with n elements, where $n \geq r$.

For a set with n elements, there are $n*(n-1)*...*(n-r+1)$ possible r-permutations.

Sometimes, this will be written ${}^{n}P_{r}$

Exercise: what is the value of the following:
(i) ${}^{4}P_{2}$
(ii) ${}^{10}P_{7}$

Exercise: in a race with 7 runners, how many possible finishing orders are there for 1st, 2nd and 3rd?

Computing r-permutations

It is normal to express the number of r-permutations using factorials:

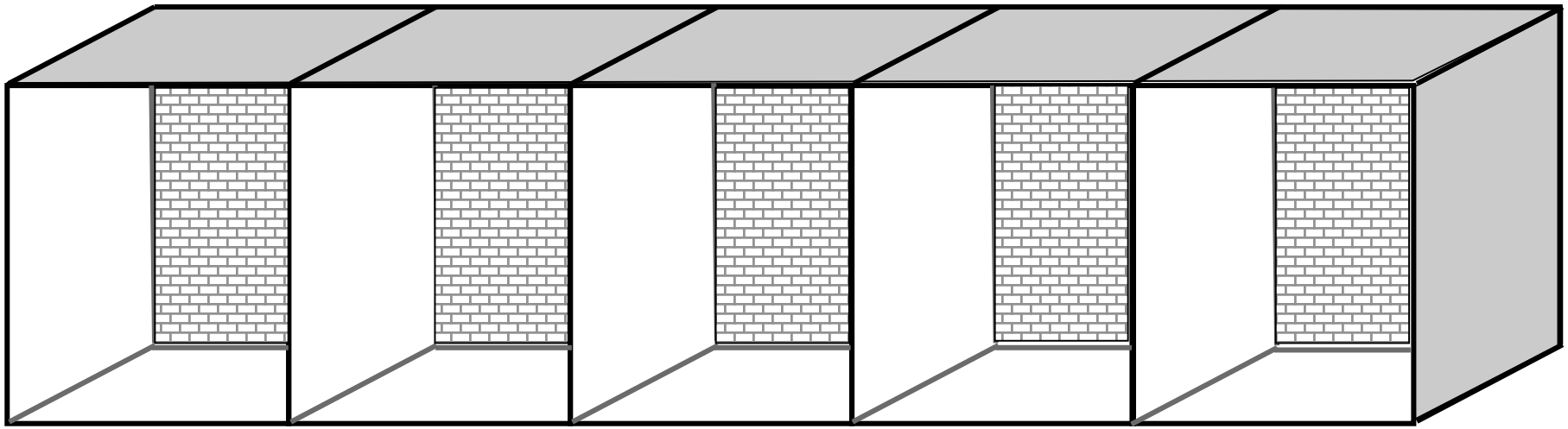
${}^{n}P_{r} = n! / (n-r)!$

We can do this because
$n! = n*(n-1)*(n-2)*...(n-r+1)*(n-r)*(n-r-1)* ...*2*1$
and so
$n! / (n-r)! = n*(n-1)*(n-2)*...(n-r+1) = {}^{n}P_{r}$

E.g. $9! / 4! = 9*8*7*6*5*4*3*2*1 / 4*3*2*1$
$= 9*8*7*6*5$
$= 9*(9-1)* ... * (9-4+1)$





The pigeonhole principle

Suppose I have a 6 pigeons. Each pigeon sleeps in a box in my pigeonloft. I have only 5 boxes. Therefore, at least one box contains 2 or more pigeons.

This reasoning gives us the pigeonhole principle (which is surprisingly useful in analysing algorithms):

If I want to place *k+1* objects into *k* boxes, and *k* is a positive integer, then at least 1 box must have 2 or more objects.

If *N* objects are placed into *k* boxes, then at least 1 box has at least *ceil(N/k)* objects (i.e. the first integer bigger than or equal to *N/k*)

Example: assigning jobs to processors

One of the main tasks in managing computational resources is load balancing – ensuring that the workload is spread as evenly as possible over the processors for maximum efficiency.

Suppose we have 6 processors, and 20 jobs. Suppose the minimum running time for any job is 5 seconds. I need to return the results of all jobs within 15 seconds. Can I do it?

By the pigeonhole principle, at least one processor must have *ceil(20/6)* jobs scheduled on it. So at least one processor must have at least 4 jobs scheduled. The best we can hope for is that each of those 4 jobs takes the minimum time to run of 5 seconds. So that processor will take at least 4*5=20 seconds to return all results. Therefore we cannot complete the task.

Next lecture ...

combinations where the order doesn't matter