Hard Problems

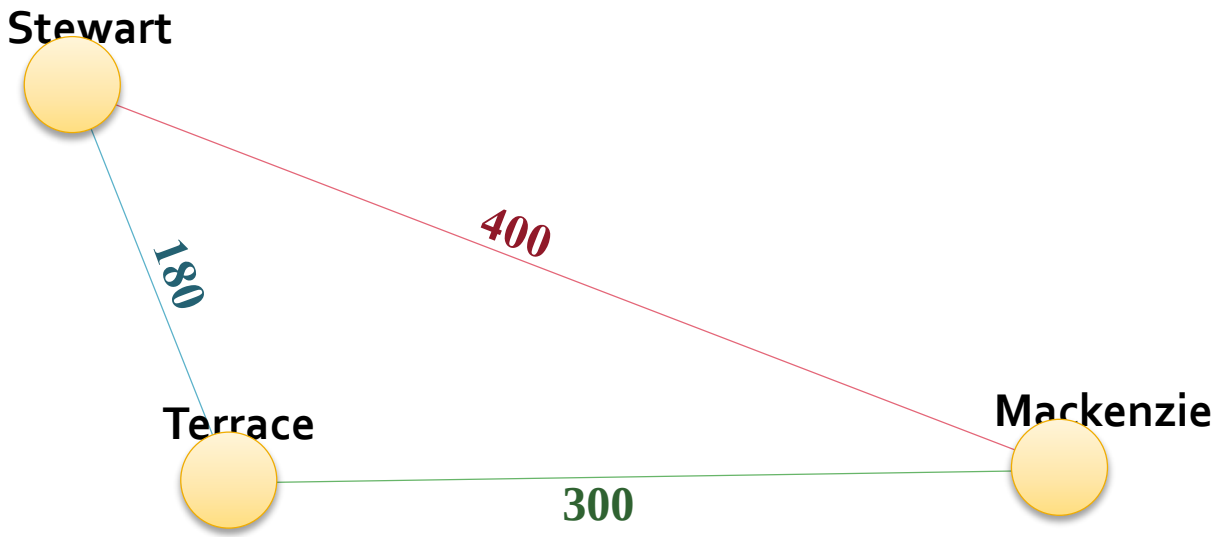
Hard Problems

- Intractability
- Decidability

Intractability

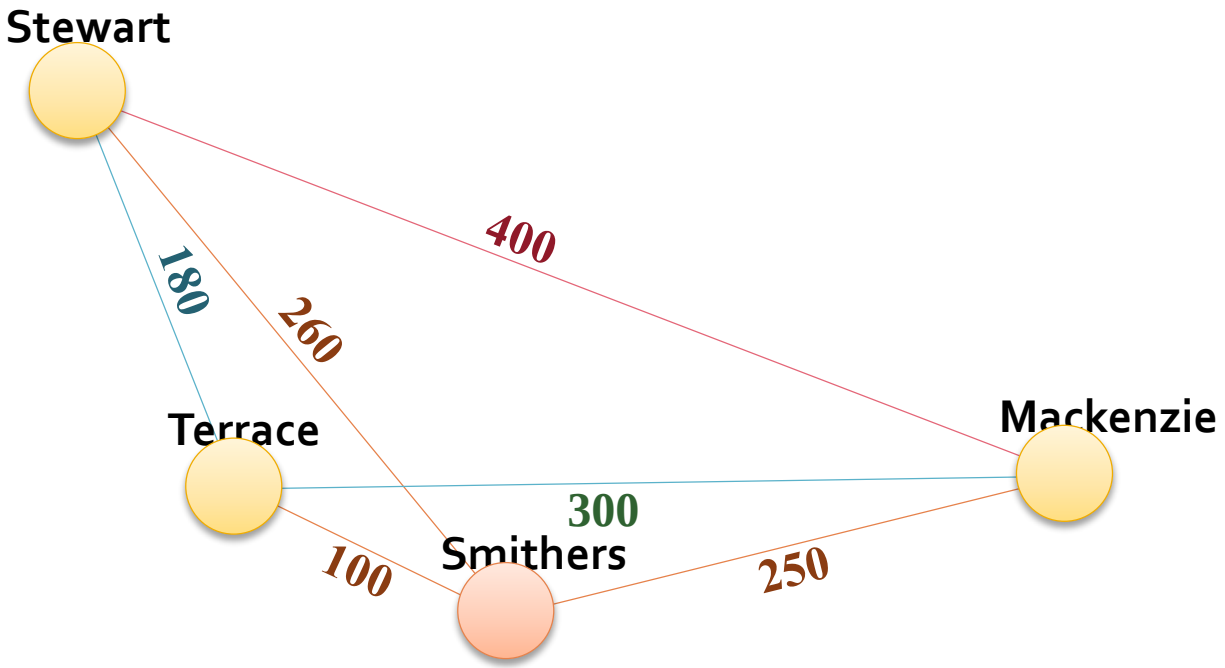
Traveling Salespeople

- If you start in Terrace, visit each town once and then return to Terrace, what's the shortest round trip distance?
  - Hint – this really isn't very difficult …



What About Smithers?

- Let's add another town to the route
  - Now what's the shortest round trip distance?
  - And how many different routes are there?



More Cities

- What is the obvious ("brute force") algorithm for computing the best possible route?
  - Add up the distance for each possible route
  - Pick the route with the cheapest distance
- But how many routes are there if we keeping adding cities?
  - 5 cities?
  - 6 cities?
  - 10 cities?

Factorials

- How many ways are there of visiting 3,038 cities?
  - There are 3,037 cities that could be visited from the home city, and
  - 3,036 cities that could be visited next
    - So that's 3,037 * 3,036 options, then we could visit one of 3,035 …
  - That gives 3,037 * 3,036 * … * 2 * 1 possible routes
  - ½ of these are just the same route in reverse so we have
    - (3,037)! / 2
    - = $1.18*10^{9,259}$
- With *n* cities then there are (*n* - 1)! / 2 different routes

How Big Is $1.18*10^{9,259}$?

- 1179738538 7515997567 2260119227 7856391456 0323307555 1203849566
  6223857826 1179738538 7515997567 2260119227 7856391456 0323307555
  1203849566 6223857826 3070636136 5215602746 9206644453 5541847325
  2220130370 7539740447 2752122520 1291614091 2145091403 9732067302
  9095006463 8027485889 3957751434 6362958528 3224729785 7733992121
  2374426615 2599815480 0036730336 6048625079 0704705660 7432124794
  3014550885 6637280030 2968046729 9055826068 3696889832 4578394493
  3624357068 8998261561 5191767591 0893074108 2349294385 8238155468
  0815656096 6262632631 5157800874 4210580633 3114982327 1001974811
  9228310691 5860575143 4948930468 8396994048 8354803731 4276983065
  4108963747 3238049816 9107195372 4859275350 5646371655 5367102189
  9393738063 4286848780 4356111805 9518212881 9347907702 1377754942
  3220491191 6209294407 5167112719 9696381592 7918733362 1823873063
  2575153864 1961522657 4794803529 7264286954 3715072093 9202983166
  2690899624 8539224903 7764220269 5680242854 2617385198 9185053984
  6738298670 4597851284 5098155293 7162985941 5305517622 3839555658
  9150095212 8005042061 6150516067 4075113631 2425463386 3282383178
  7288856764 2016284679 2115626322 2246583256 0213094662 6809073774
  5185050562 9930154116 7848498351 3496888294 5960074432 0935270136
  0825499364 7562737506 4501211533 1501232550 4295791699 6118972369
  0410343503 6790720497 5885138806 2713821784 7109944375 5755592660
  4891325767 2698318269 5305726001 8583929769 5229437406 4140255544
  7547356207 8398858174 7523048707 4556995801 1938636499 8023297593
  4056492057 5412152578 9027834148 4537064002 7703010787 1766359178
  7065282374 7335620743 3371195811 1891997775 9275259959 7298858293
  9404092355 9608038026 6782984591 0574407202 6816887612 6679297847
  6579561635 2656912599 8508514645 7573420370 9558628096 1774551234
  1120558982 8612170133 2810378740 9141069606 3448466398 4811431893
  3422570728 3264478137 9605216433 8517528699 0033450472 7803333445
  7632203983 1919534878 1770961466 6577766328 2536477497 1533007115
  7891112267 6378057868 4786969380 1595948321 0786213185 4208358559
  6276299481 9706997996 4846831301 5731803757 9885179761 1641640990
  0949747238 5323968461 5656487838 2145501860 0006086772 1511230101
  6534642308 1729395684 9714393355 8232045525 2903977332 4715270182
  8573355481 4917429133 0436413735 9585907577 6365387951 4653266609
  3506855316 4098691101 2624091655 4200419620 2450700030 2143673348
  0652686673 6863306775 6880415188 6013223685 1903334020 3502856989
  6839602419 6949718977 7363569407 3545175700 3510778109 2475114111
  3071038659 0967697946 0819936674 1215930871 6317365383 2857347264
  0850437237 4868876874 8203589265 5582789956 8990277197 9694455847
  0982024901 1452259530 0650991156 4839105954 0186086255 3325033544
  0143696388 7567405937 1491168422 0269250650 6443213344 3462047262
  9148229726 0481504546 7879249178 4470426613 7253155617 5932158191
  6479159896 9768917999 1102358818 0200442448 6911306561 1756275029
  9081111061 9286385429 8835042868 6325785176 0954352879 8043641348
  2598825402 1887773477 0855569116 5679431057 4229551930 3684103666

Let's show it as a regular number rather than in exponent notation

This is just the first part of the number …

How Big Is $1.18*10^{9,259}$?

- 9659154881 6063169517 3545922819 2470456084 7902101506 6352746694
  9921643825 1588114204 9211969163 6697747150 4558561834 8070025606
  2727837206 7734859183 3839559877 0585045955 5357722828 9061835738
  5741316994 1878289932 6870850320 3770235130 9323427446 2192877954
  7017768711 7661445171 3353249945 9532847642 2633414209 2854165343
  7515774547 8188939951 4631388529 7419856109 4142925717 4757895273
  4080921564 9124311623 4660296778 8510661109 4423156421 7579083217
  5390846290 5045399723 9618843157 1539802659 2543415273 4820290360
  0108076117 6315917227 0158057970 0821463331 9060529853 2397469949
  7732122792 9133202121 6171596678 3133109065 8113443367 8497121410
  0087692377 6224699854 2087970552 8118871547 9810026425 2484100190
  5143284010 6837208909 0720700460 2334053520 0114954183 3246290995
  3578385056 5996050555 8928185955 9536082135 1892841779 9685361965
  6809830270 1556581799 4838279760 9159151196 6272237281 1405452377
  5578634637 4140949581 1698793580 3292610643 7916422644 3024635768
  9878697644 1786525814 3578300255 7411494063 0094301164 7027118509
  7433608760 1143781804 5552582332 6125630109 2053865088 1560719583
  3105774979 8007502403 8371886047 5385192257 7037124506 6680724704
  5846660778 4655581892 1242018830 0512254454 8474894571 7252661894
  9056206906 0562592005 4603024708 8899424226 4042012936 0179809745
  7979881995 8797663321 2571021252 1148187571 7464015645 3506079130
  3640052264 3463681091 0805389383 0055102439 1528209525 7400609407
  4252975505 2160765200 2192347127 0853776415 0408504854 0688383926
  8511702597 0504521188 9999304216 1208884005 1601481280 4179593767
  5668191121 1011457570 7899909958 8711248014 3571539201 7573277683
  6422489858 9388908569 3428228004 4772806478 5801290060 3011487411
  8317215565 6605877801 2035031974 1488362116 9934308722 4583021941
  6962402109 1963239989 1906085359 1001348622 7220801855 0438586247
  7627818922 6786999700 0725433787 2812110137 9871479641 6678331669
  2735447289 5581333914 1264929219 4682277559 5210861824 1934234517
  7862962020 7012711410 6216320543 6467984880 3823624135 9757329238
  3407084711 4672056155 9055800699 2464858427 5654117980 7715816487
  1140957339 7805265754 9367162718 7173050677 9247249288 7610851121
  3818828540 9624815386 0097017774 8859769563 0043572623 0980127673
  3085140446 6592018118 9913214662 7747518553 4300086445 3443344803
  9230411892 3782623928 5683891208 7190697169 1614446919 5075644050
  9701293134 2831768420 7577662825 6550026154 3733403867 4963879027
  1375398772 4730952618 9815992142 0771106715 0893971514 4827758165
  9414868697 0362437695 9740946735 6776105255 6026685550 1886615121
  0392990907 7504560775 6958678471 5186545363 3879462590 1521148086
  2652170895 7072989426 8784285464 4636351418 5969149282 2646361222
  5725843329 5302671981 5251596738 3757045664 0542871990 1891580877
  5422300043 5419684821 8177053524 4725280200 3258491294 0006963695
  8022808829 7530149907 1008107107 2806416393 8864201387 9942637744
  6318776301 6755557577 1738614296 9651500636 5354906486 4799171315
  1744321450 3696036288 2350579267 6666244311 2443202024 2017571295

Still not done …

How Big Is $1.18*10^{9,259}$?

- 0522168335 1530466367 6727069056 8841118584 3084397251 2161316837
  0232782902 1451628504 5580669889 8533177764 7811346673 0164701687
  0970192544 6441596087 9195022284 3326764637 9929168750 2983104048
  8556116669 8881310995 1790733251 0303010742 1675632532 8490743463
  5606282071 5318123388 8222363342 6562283131 3286514609 6589477443
  9576005717 5139379603 3208451014 4157123017 6357960316 5492149681
  2649929257 9850784968 8431244684 7955132437 8979795976 4272619728
  3320691342 2553664444 5045667341 4907371387 5248099896 4421909771
  9607785379 9279169578 2721103408 2685183301 1882129336 8475449985
  0493838923 2415215053 2081055357 3239842861 3550693731 0499161238
  0974267080 6254564258 9387784844 3730510109 0734522466 6059425533
  0033334060 2414759957 9676405902 7447288655 1777934442 9141797213
  0191840470 5771390342 4253824076 2729408082 8717461432 1632477148
  5043950794 4922809033 4817703984 1584333910 5597235809 7774439072
  7998720099 3027787422 2616006142 8956780117 2051583935 9894308616
  5387209625 1495934559 6646591704 8387926167 1854950354 8232689644
  1229872772 6721451727 8797253951 1668506609 6378075891 7020184510
  7187423089 9412825541 2425600430 8477638378 2317760136 4277308100
  0079749412 1299004567 8544176526 5524541245 8518325879 2285400338
  5110740325 7787855398 1663533716 1609667187 5026295181 7946863928
  1359796500 3447540766 4598835233 7915473714 4019644270 9161407738
  8274526150 0143351543 4663840932 7881648930 6779396542 0185360552
  3350917223 6258211091 9365301247 1939332778 0634697978 4290853661
  3526837293 2343001897 1493279520 6372157566 5618886076 5285065373
  1159854319 2397879991 0852250402 8829766675 5744378819 3210963040
  1098722274 2131250060 2837841597 1829194422 1862739126 5603000915
  6304081568 5687429362 4366706273 2216211612 6703680552 0736077226
  4846737059 4751647866 3955005830 7841819611 3071726775 6179938345
  1329876070 3887899996 4497589984 8145796763 5222795468 1601406474
  4562177144 3855041002 2145463467 3246571315 9246853310 6313615968
  2008457632 1788785989 5264313513 0674528250 4987681047 8283227076
  3942397052 0010183719 5489610941 7637673174 2134964985 9636308580
  6780563277 5776602221 8663312278 9142933473 3376992984 1272959687
  7645286775 0219809057 2419222528 6111215488 0060696397 1661312061
  5686978563 4389660115 4375725736 3025547776 8939135853 0277765425
  5517969902 8123839426 0254622897 4198694165 8337908390 1434232571
  1681879149 3514658977 9632956055 1173080322 3854733428 0686711250
  1993455426 2768220240 2600966238 2733139568 3676816633 2830982747
  1693519546 7299505508 2688865180 0714588049 7486410477 7120533015
  9234527867 7478130912 9070422803 2706730573 8182407194 3344888906
  9371617574 6331307032 4096951212 1067083326 4855437098 1637205255
  3474522629 1356271592 7453438897 9337571507 0624565136 4735480013
  7687461407 4358398558 7626764102 7640702705 1319034987 9634866721
  8391982623 1136828370 1629614763 7266386131 8710123706 9597718438
  4879254670 3919338110 7945824995 5415950611 5770792310 7200323966
  0964957854 6742997022 1388280284 6716925661 5068923056 3997479013

... not yet ...

How Big Is $1.18*10^{9,259}$?

```
5003479310 1572088467 5684485842 6597339564 6198507453 4508488799
5258505826 1200955513 2939378331 0651661968 0071233660 5700591022
0767852628 6122524922 1507303036 3275319141 8772681409 0007850650
0274874049 0460547110 9424199844 2380619096 5048517097 2358209041
4458378742 4220617177 4158658524 9106500910 8578634773 2992000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000
```

done!

How Long Does it Take?

- About how long would it take to perform $1*10^{9,259}$ calculations?
  - Make a *back of the envelope* calculation
    - Assume we use the Sunway TaihuLight with approximately $1*10^{17}$ instructions per second
    - And $1*10^{9,259}$ different possible routes to evaluate
- Here's an estimate:
  - $1*10^{9,259} / 1*10^{17} = 1*10^{9,242}$
  - $1*10^{9,242} / (60 * 60 * 24 * 365) =$
  - $3*10^{9,234}$ years
    - The sun is projected to turn into a red giant in 5 billion, or $5*10^{9}$, years

So How do we Solve the TSP?

?

- OK, it's not quite that bad …
- There has been a lot of work performed on the TSP, and some specific large problems have been solved
- But, there is no practical, general solution to the problem

So What?

- The TSP is an example of a number of similar problems that don't just relate to salesmen
  - Logistical or transportation problems
    - What is the best bus route?
    - How should supplies be transported to minimize costs?
  - Other related problems
    - How should components be organized on printed circuits to minimize the connections?
  - Solving such problems would be *very* rewarding
- The TSP is an example of a *Operations Research* problem
- It is also an example of an *intractable* problem

Finite Tile Problem

- Finite tile problems
  - Tile an $m * m$ grid with $n = m^2$ tiles
  - Where the edge of the tiles have to match each other
- Solutions to finite tile problem
  - Try every possible combination of the tiles
    - But there are n! combinations
  - Add one tile at a time
    - Until one fits
    - If none fit remove the last tile
    - Known as *backtracking*
    - Rejects failing permutations early so less than n! steps

Subset Sum and Knapsacking

- Subset Sum
  - Given a list of n numbers and a target number t
  - Find a subset of those numbers that sums to t
- Knapsacking
  - Given a list of n items
    - Of weights $w_1, w_2, \ldots, w_n$
    - And values $v_1, v_2, \ldots, v_n$
  - Pack a car whose maximum load is W such that value is maximized

Scheduling and Satisfiability

- Scheduling
  - Given a list of *n* students each of which has up to 5 final exams
  - Devise the minimum schedule so that no exams overlap for any students
- Satisfiability
  - Given a logical expression of length *n*
  - Find a true/false substitution that will yield "true"
    - $d \vee (a \wedge b \wedge (c \vee d \wedge \neg a))$
    - $a \wedge \neg a \wedge b \wedge c \wedge d$

$\vee$ = or (||), $\wedge$ = and (&&), $\neg$ = not (!)

Show Me the Money

- The TSP is one of a class of hard problems where
  - The known solutions are impractically large (k^n or $n!$), but
  - Given an optimal solution, it is easy to verify (or check) a proposed solution
    - e.g. If we know the length of the shortest route it is easy to check if a particular route is that long
- This set of problems is called *NP-hard*
  - It is one of 7 problems identified by the Clay Institute of Mathematics
  - There is a $1,000,000 prize for solving each problem!

Polynomial Time

- A polynomial is an algebraic expression which includes a variable raised to some integral power
  - Like $3x^5$, n^2, $p^3q + y$
- An algorithm's running time is in polynomial time if it is bounded by some finite polynomial in n
  - i.e. n raised to some finite power
  - e.g. $n \log_2 n$ is bounded by n^2, n^2 is bounded by n^3
- Many problems can be classed as class *P* problems
  - Where they can be solved in polynomial time

P vs. Exponential

- We can break problems into two categories
  - Problems that can be solved in polynomial time
  - Problems that can be solved in exponential time
- Is this distinction important?
  - Yes
  - Let's think about the perennial solution to any difficult problem
    - Throw more resources at it

Well, at least for governments …

  - What impact does this solution have on these classes of problems?

Let's Buy a Better Machine

- Imagine that we want to solve larger problems
  - So we buy a computer that is 1,000 times faster
- How much bigger problems can we solve *in the same time*?
  - Linear problems (linear search, printing an array)
    - 1,000 * *n*
  - Quadratic time problems (like selection sort)
    - 31.6 * *n*

    $\sqrt{1000} = 31.6$
  - Five nested loops from 1 to n, i.e. n^5
    - 3.98 * *n*
  - 2^n

    Note the +, not a typo!
    - *n* + 10
  - And what about *n*!
    - *n* + 2

If our old computer could solve a problem of size 200, a computer that's 1,000 times faster can solve a problem of size 210

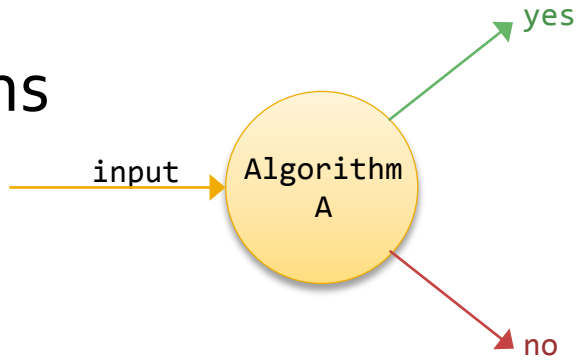
NP Problems

- The TSP is in a class of problems referred to as NP
  - NP does *not* stand for non-polynomial
  - It stands for *nondeterministic* polynomial
- The *nondeterministic* refers to an interesting quality of problems like the TSP
  - Let's say we know the size of the shortest round trip distance
  - Now it's easy to prove whether or not a route is optimal
    - Just add up the lengths of the path
    - A polynomial problem
  - So solutions to NP problems can be *verified* in polynomial time

Decidability

Decision Problems

- Algorithmic problem
  - Set of legal inputs
  - Specification of desired output as a function of input
  - We've seen lots of these in this course
- Decision problem
  - A subset of algorithmic problems where the output is Yes or No
    - Or Accept or Reject

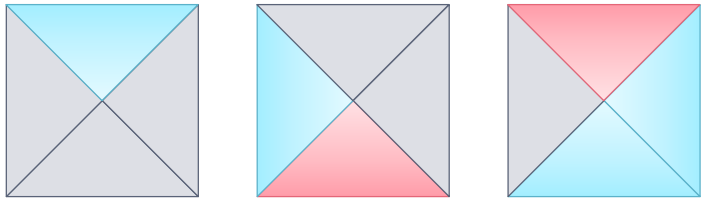


Undecidable Problems

- There are problems that we* haven't solved
- It may for lack of
  - Money – we don't have a big enough machine
  - Time – we are still waiting for the solution
  - Brains – we haven't figured out the algorithm
- Even with unlimited resources there are some problems that we *cannot* solve
  - *Undecidable* problems

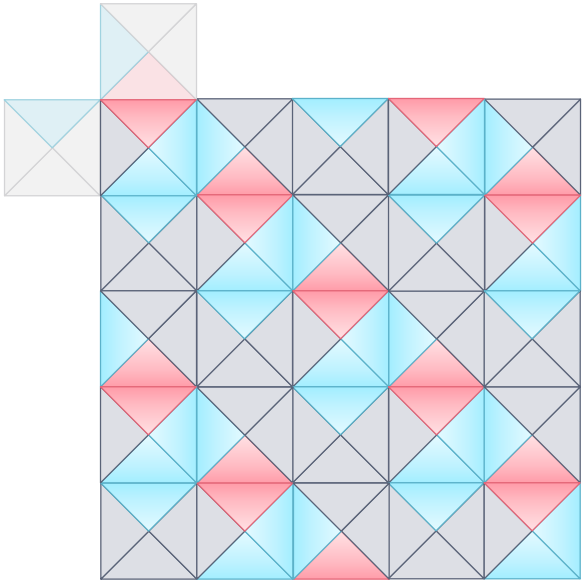
*humanity, not CMPT 125

Tiling the Integer Plane - 1



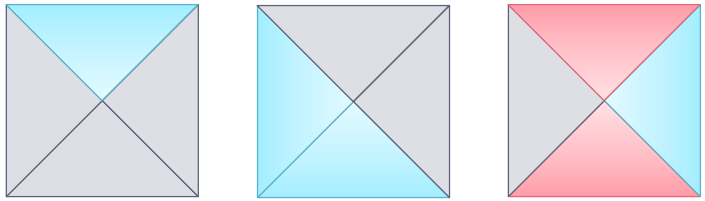
1 2 3

Can we tile the entire (infinite) integer plane with these three tiles?



Yes!

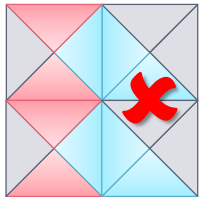
Tiling the Integer Plane - 2



1 2 3

Can we tile the entire (infinite) integer plane with these three tiles?

No! 3 has to appear somewhere



Adjacent 3's need blue to the right

Tiling the Plane

- Given a finite set of tiles *T*, can *T* be used to cover the integer plane?
  - Input
    - The set of tiles
  - Algorithm
    - Returns true if the set of tiles can cover the integer plane
    - And false if it cannot
- There is no such algorithm
  - That can determine this for any set of tiles
- The problem is undecidable

The Halting Problem

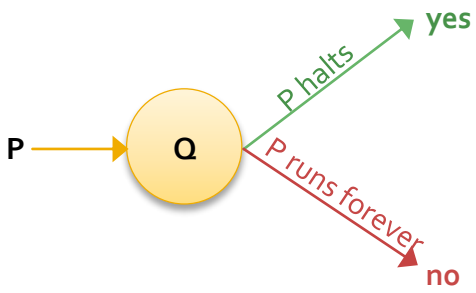
- Given a program *P* and an input *x* determine of *P halts* on input *x*
  - e.g. `while(x != 1) {x = x - 2;}`
    - Depending on the value of *x* this either terminates (halts) or goes into an infinite loop
- Naïve algorithm to solve the halting problem
  - Simulate *P* on input *x*
  - If *P* terminates then return yes
  - Else return no
- But …

Halting Problem is Undecidable

- There can be no algorithm for the halting problem
- This was proven by Alan Turing in 1936
  - The proof is a *proof by contradiction*
- We want to prove that there is no algorithm for the halting problem
  - Start by assuming that such an algorithm exists
  - Then demonstrate that this results in a paradox, or contradiction

Halting Problem Proof - Sketch

- This is an abbreviated version of a proof that there is no solution to the Halting Problem
  - By Alan Turing, 1936
- Assume that a program *Q* determines whether or not its input, program *P*, halts

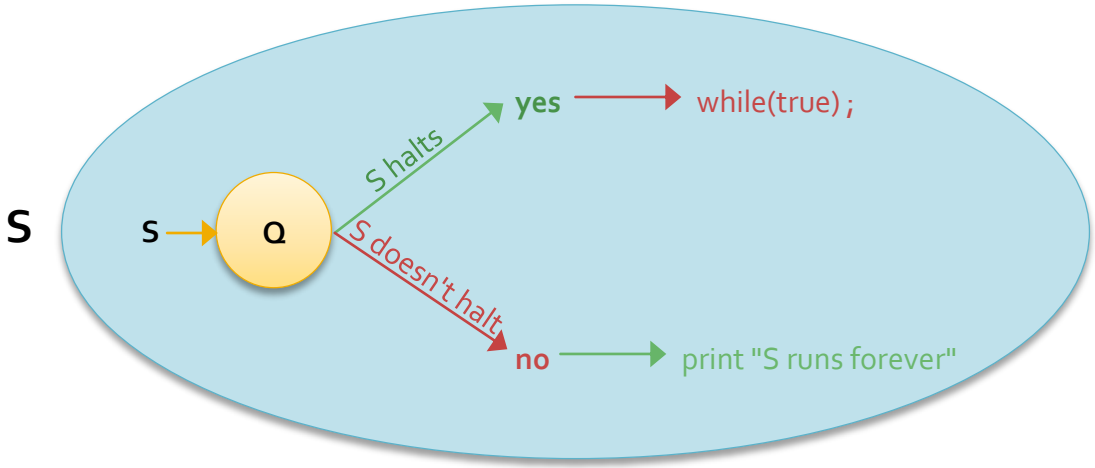


Create Paradox

- We've assumed that *Q* exists
  - Remember that *Q* inspects a program and determines whether or not it halts
- Write a second program called *S* which does the following:
  - Runs *Q* and asks it to inspect *S* (i.e. inspect itself)
  - If `Q(S) == "yes"`
    - Then go into an infinite loop
  - `Else //if Q(S) == "no"`
    - Then print `"Q detects that S does not halt"`

Note that all of these steps are trivial, with the exception of writing *Q*

Contradiction!



- *S* runs *Q* with S as input
  - If Q detects that S halts then S enters an infinite loop
    - i.e. S does not halt
  - If Q detects that S does not halt then S halts
- A contradiction
  - Therefore Q cannot exist

Rice's Theorem

- Can we write program's to
  - Find bugs in other programs?
  - Determine if two programs are equivalent?
  - Determine if a program is malicious (a virus)?
  - Determine if a program always outputs integers
- We cannot write a program that determines any non-trivial property about *all* programs
  - Although we can sometimes achieve this for *most* programs