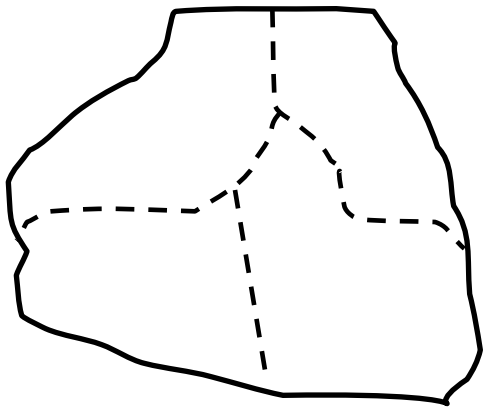
Software Testing Problem

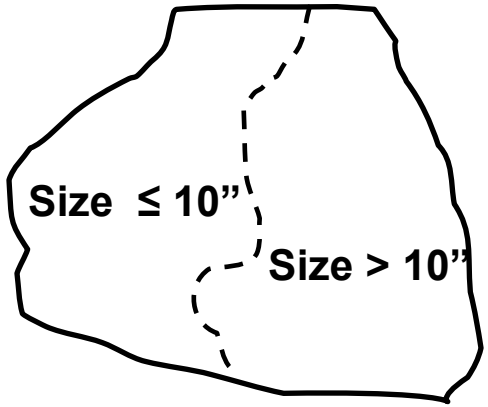
- The input domain of a program is its all possible inputs.
- Identify a subset T of the input domain so that the executions of the program with T reveals all bugs.
- In practice, you are likely to select a subset of T such that it reveals as many errors as possible within the testing budget.

Equivalence Class Testing

- The variable domain is partitioned into disjoint subsets.
- The system behaviour within each subset should be consistent i.e. each point in the subset shall result in the same behaviour of the program.
- Choose one test case (data point) from each sub-set
- Equivalence of outputs could be considered as well and could be used to refine the equivalence partitions of the input space e.g. if two different inputs of program cause outputs that belong to different equivalence classes of outputs, these inputs should be in different equivalence classes of input values, too
- Boundary and special values tests are based on the assumptions that bugs are likely when input or state values are at or very near to minimum or maximum.

Partitioning





Exhaustive Testing

How long would it take (approximately) to test exhaustively the following program?

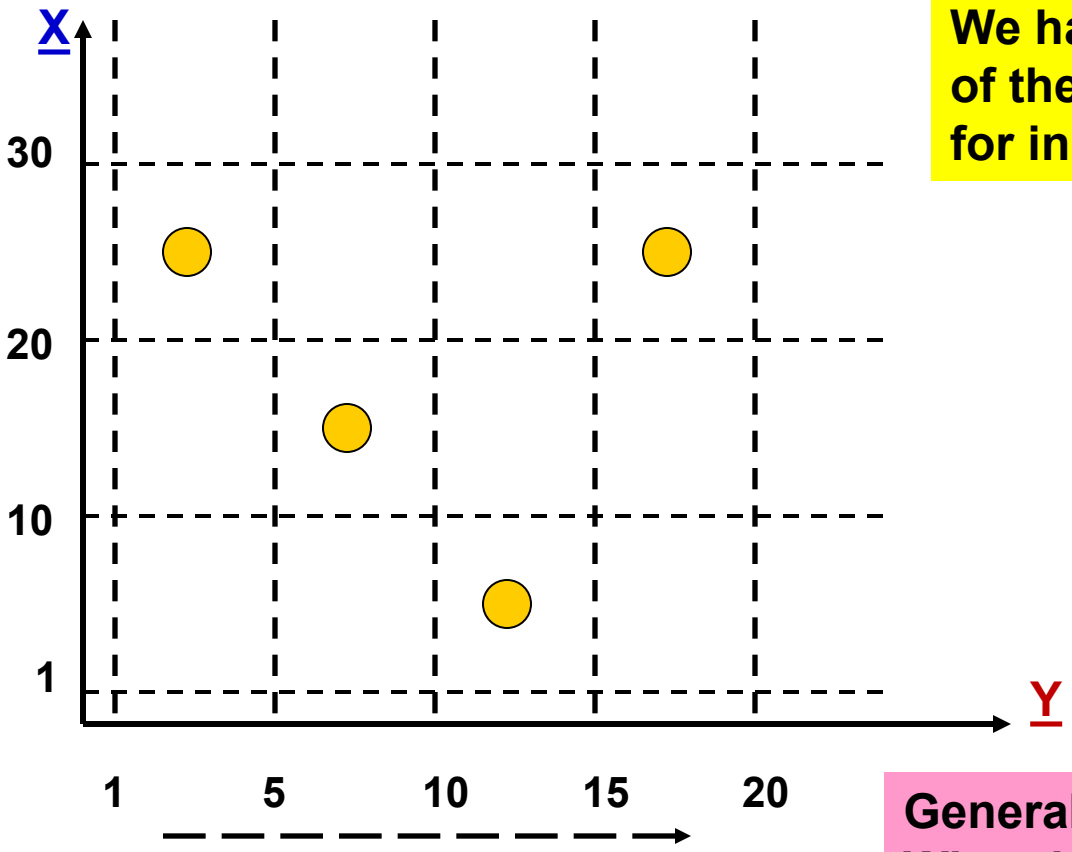
- int sum(int a, int b) { return a + b; }
- 2^32 x 2^32 = 2^64 ~ 10^19 tests
- If each test takes 10^(-9) secs, it is still 10^10 secs

Weak Normal Equivalence testing

1. Assumes the 'single fault' or "independence of input variables."
   - e.g. If there are 2 input variables, these input variables are independent of each other.
2. Partition the test cases of each input variable separately into one of the different equivalent classes.
3. Choose the test case from each of the equivalence classes for each input variable independently of the other input variable

Example of : Weak Normal Equivalence testing

Assume the equivalence partitioning of input X is: 1 to 10; 11 to 20, 21 to 30 and the equivalence partitioning of input Y is: 1 to 5; 6 to 10; 11 to15; and 16 to 20



We have covered everyone of the 3 equivalence classes for input X.

For (x, y) we have:
(24, 2)
(15, 8)
(4, 13)
(23, 17)

We have covered each of the 4 equivalence classes for input Y.

General rule for # of test cases?
What do you think?
# of partitions of the largest set?

Strong Normal Equivalence testing

- **This is the same as the weak normal equivalence testing except for**

  "multiple fault assumption"

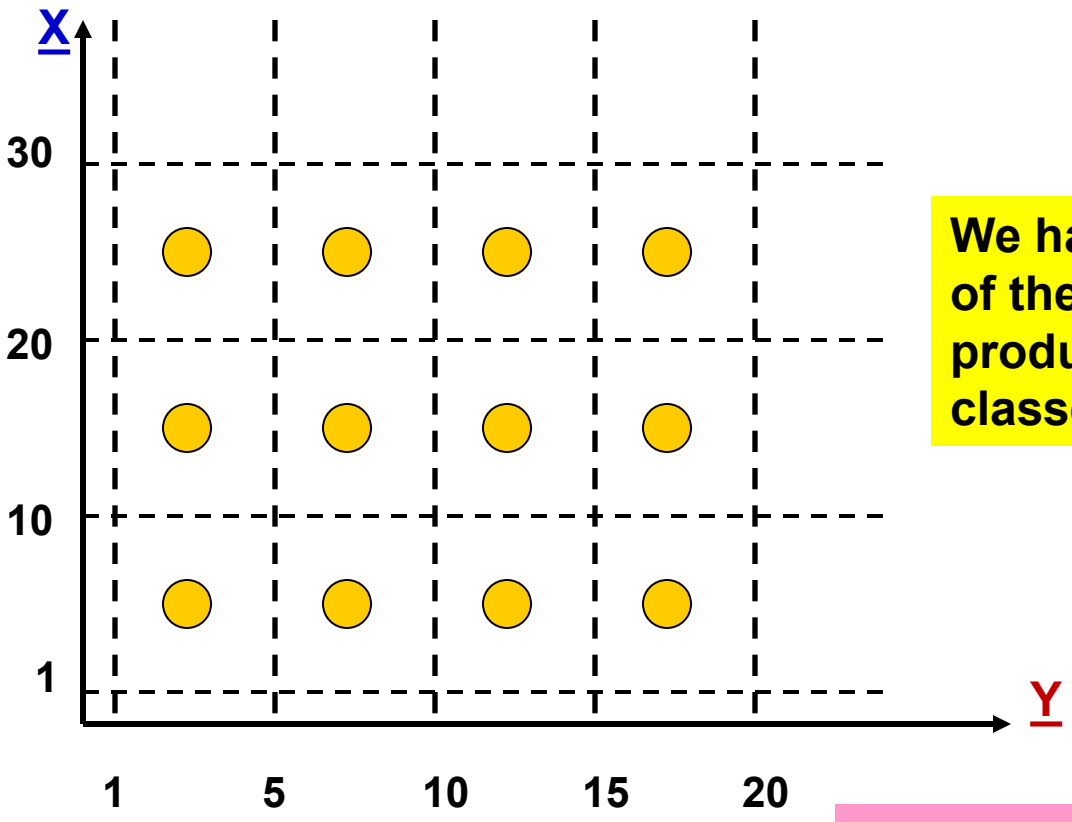
  or

  "dependence among the inputs"

- **All the combinations of equivalence classes of the variables must be included.**

Example of : Strong Normal Equivalence testing

Assume the equivalence partitioning of input X is: 1 to 10; 11 to 20, 21 to 30 and the equivalence partitioning of input Y is: 1 to 5; 6 to 10; 11;15; and 16 to 20



We have covered everyone of the 3 x 4 Cartesian product of equivalence classes

General rule for # of test cases? What do you think?

Weak Robust Equivalence testing

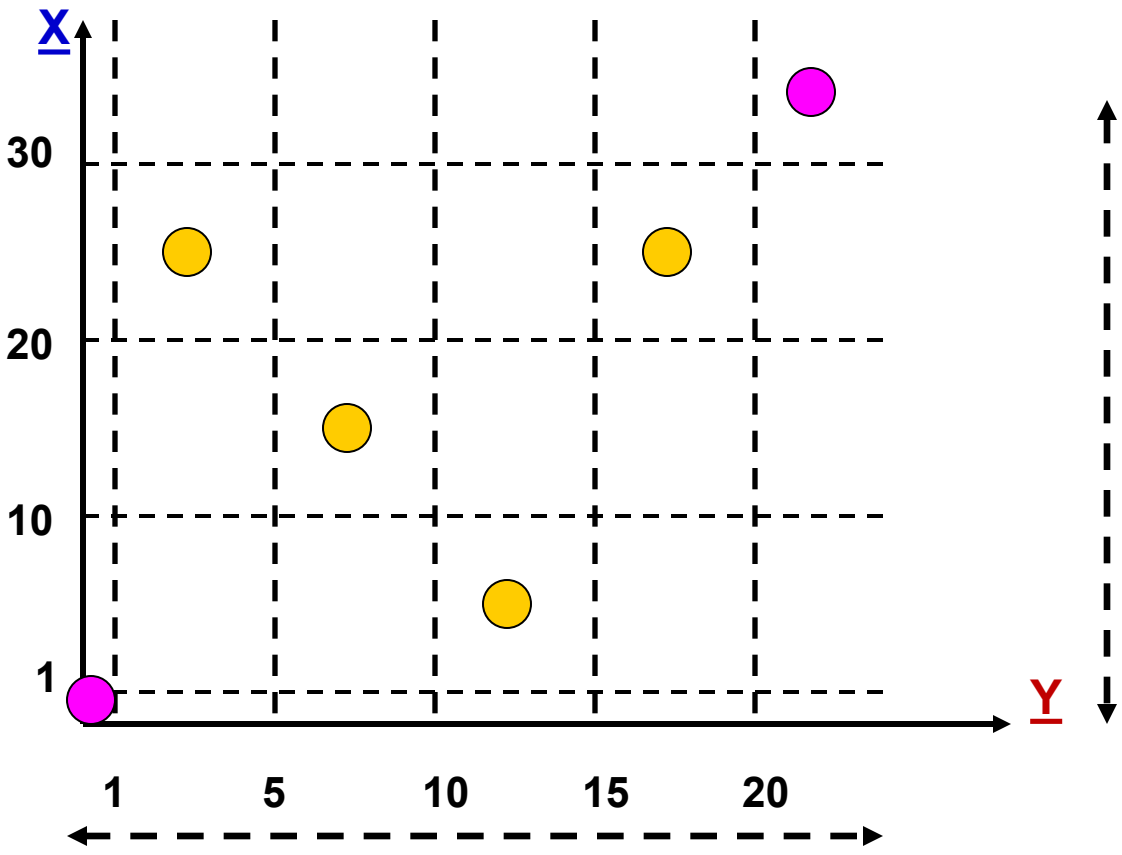
- **Up to now we have only considered partitioning the *valid* input space.**

- **"Weak robust" is similar to "weak normal" equivalence test except that the *invalid* input variables are now considered.**

A note about considering invalid input is that there may not be any definition "specified" for the various, different invalid inputs - - - making definition of the output for these invalid inputs a bit difficult at times. *(but fertile ground for testing*)

Example of : Weak Robust Equivalence testing

Assume the equivalence partitioning of input X is 1 to 10; 11 to 20, 21 to 30 and the equivalence partitioning of input Y is 1 to 5; 6 to 10; 11;15; and 16 to 20



We have covered everyone of the 5 equivalence classes for input X.

We have covered each of the 6 equivalence classes for input Y.

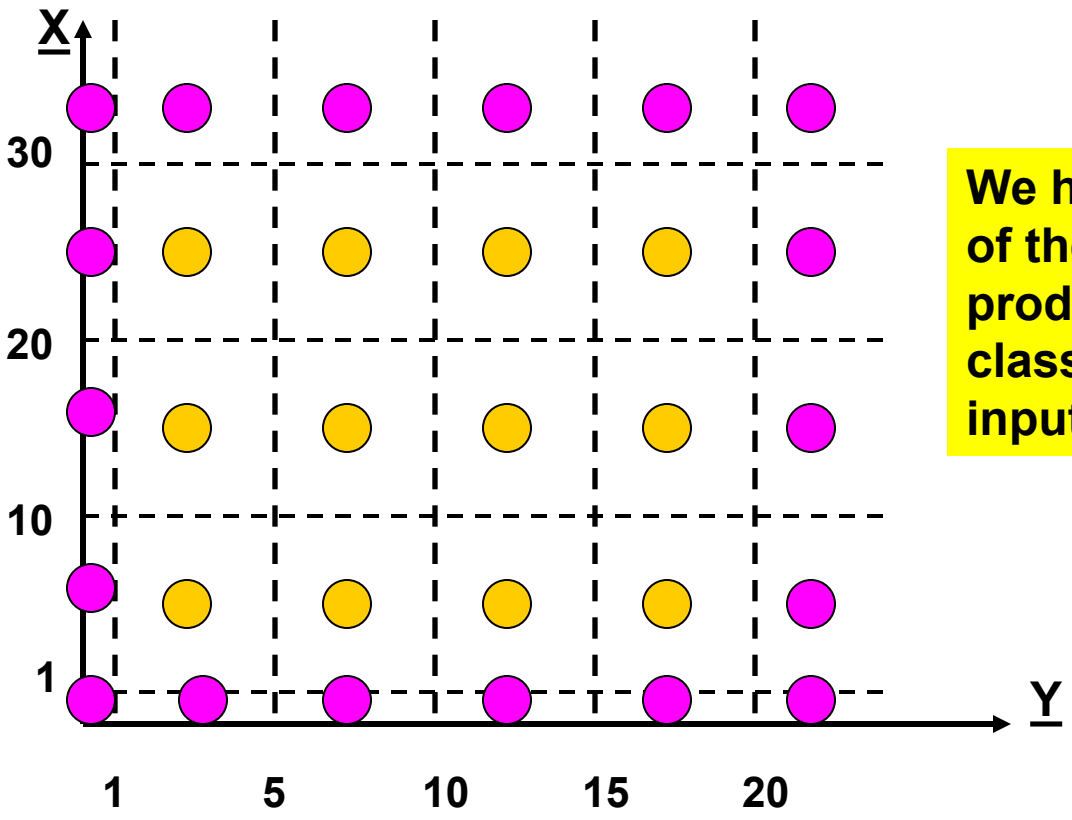
Strong Robust Equivalence testing

- **Does not assume "single fault" - - - assumes dependency of input variables**

- **"Strong robust" is similar to "strong normal" equivalence test except that the invalid input variables are now considered.**

Example of : Strong Robust Equivalence testing

Assume the equivalence partitioning of input X is: 1 to 10; 11 to 20, 21 to 30 and the equivalence partitioning of input Y is: 1 to 5; 6 to 10; 11;15; and 16 to 20



We have covered everyone of the 5 x 6 Cartesian product of equivalence classes (including invalid inputs)

Next Day Program example

- **After receiving Day of the month, Month and the year as three separate inputs, program produces the next on the calendar.**

- **One possible set of partitions are:**
  - **Day : 1 through 31 days**
  - **Month: 1 through 12**
  - **Year: 0001 through 3000**

- **In the above case, if we assume that day of the month, month and year values are independent and invalid values for the day of the month, month and year are prevented via input validation, for weak normal equivalence testing only needs 1 test case from each input. Example:**
  - **(year; month; day) : (2001, 9, 23)**

- ***Clearly lots of bugs could potentially be not exposed if only one potential input is considered in this case.***

- ***What could be a better set of partitions?***

Next Day Program example (cont.)

- Day: 1 through 28
- Day :29
- Day: 30
- Day: 31

- Month: those that have 31 days or {1,3,5,7,8,10,12}
- Month: those that have 30 days or {4,6,9,11}
- Month: that has less than 30 days or {2}

- Year: leap years between 0001 and 3000
- Year: non-leap years between 0001 and 3000

i.e. 4 partitions of Day, 3 partitions of month and 2 partitions of year.

If we assume that inputs are independent of each other (weak normal), what faults could be missed?

If we assume the inputs are interdependent (weak robust), what inputs constitute redundant testing?

Next Day Program example (cont.)

If we assume that inputs are independent of each other i.e. weak normal

1. How many test cases ?
2. What errors/faults/bugs could be missed out?

   Example Inputs:

   (Leap Year, Month, Day)

   (leap year, 1, 8)
   (leap year, 2, 28)
   (non-leap year, 3, 31)
   (non-leap year, 6, 29)

If we assume the inputs are interdependent i.e. weak robust

1. How many test cases?
2. Is redundant testing still possible?

Next Day Problem example (cont.)

We should have (2 years x 3 months x 4 days) = 24 test cases

(leap year, 10, 5)	*(non-leap year, 10, 5)*
(leap year, 10, 30)	*(non-leap year, 10, 30)*
(leap year, 10, 31)	*(non-leap year, 10, 31)*
(leap year, 10, 29)	*(non-leap year, 10, 29)*
(leap year, 6, 5)	*(non-leap year, 6, 5)*
(leap year, 6, 30)	*(non-leap year, 6, 30)*
(leap year, 6, 31)	*(non-leap year, 6, 31)*
(leap year, 6,29)	*(non-leap year, 6, 29)*
(leap year, 2, 5)	*(non-leap year, 2, 5)*
(leap year, 2, 30)	*(non-leap year, 2, 30)*
(leap year, 2, 31)	*(non-leap year, 2, 31)*
(leap year, 2, 29)	*(non-leap year, 2, 29)*

Can we still do better?

Next Day Problem example (cont.)

- M1 = {month : 1 .. 12 | days(month) = 30 }
- M2 = {month : 1 .. 12 | days(month) = 31 ∧ month ≠ 12 }
- M3 = {month : {12} }
- M4 = {month : {2} }
- D1 = {day : 1 .. 27}
- D2 = {day : {28} }
- D3 = {day : {29} }
- D4 = {day : {30} }
- D5 = {day : {31} }
- Y1 = leap year
- Y2 = common year

Handles end of month and year better.

Year 2000 could be a separate partition for Y2K bugs.

(In Class Assignment) Search Clients

Determine the equivalence partitions for the keywords, time window and location area based search of the photos from the folder in the external storage by the Photo Gallery app. Assume the following:

- user specifies a time window by picking a StartDate and an EndDate with the date range from MinDate to MaxDate
- for location based search, the user specifies a search area defined by {[TopLeft.Lat, TopLeft.Long], [BottomRight.Lat, BottomRight.Long]}, bounded by {[90, -180], [-90, 180]}
- For keyword based search the user specifies a keyword of up to 10 printable characters and all printable characters (including escape characters) are treated equally

(In Class Assignment) – Search Photos (cont.)

Time Window based search:

- Valid Equivalence Class: MinDate <= StartDate <= EndDate <= MaxDate.
- InValid Equivalence Classes: StartDate = null; EndDate = null; or EndDate < StartDate.

Location Area based search:

- Valid Equivalence Class: 90 >= TopLeft.Lat >= BottomRight.Lat >= -90 and -180 <= TopLeft.Long <= BottomRight.Long <= 180.
- InValid Equivalence Classes: At least one of TopLeft.Lat, TopLeft.Long, BottomRight.Left, BottomRight.Long is null; TopLeft.Lat < BottomRight.Lat; or TopLeft.Long > BottomRight.Long;
- Note: Additional equivalence classes e.g. search along a latitude, or search along a longitude could also be specified.

Keyword based search:

- Valid Equivalence Class: a string of up to 10 printable characters.
- InValid Equivalence Class: Null or empty string.

Independent Testing

- Programmers have a hard time believing they made a mistake
- a vested interest in not finding mistakes
- Design and programming are constructive tasks
- Testers must seek to break the Software

Equivalence Testing

- Several tries may be required before the “right” equivalence relation is discovered
- Coverage reports can be used to know if more partitions needed
- Input validation can eliminate testing for invalid partitions.