CSE 373

MAY 3RD – HASHING & GRAPHS

ASSORTED MINUTIAE

- Exams are in my office, you can pick them up during any of my office hours.
- HW4 out tomorrow morning
- Regrade requests processed
- H2P2 and H3P1 will be back next week
- Finishing discussion on hashing today, so section tomorrow will be all examples.

MAKE UP ASSIGNMENT

- **Make up assignment end of Week 7**
  - Choose **either** a coding assignment or a write-up assignment
  - Will overwrite your lowest grade for either
  - EC will be possible on these assignments, so it could give bonus points

TODAYS LECTURE

- Hashing considerations
- Introduction to graphs

HASH FUNCTION

- **In reality, good hash functions are difficult to produce**
  - We want a hash that distributes our data evenly throughout the space
  - Usually, our hash function returns some integer, which must then be modded to our table size
  - Needs to incorporate all the data in the keys

HASH FUNCTION

- **You will not have to produce hash functions, but you should recognize good ones**
  - They run in constant time
  - They evenly distribute the data
  - They return an integer
- **These hash functions are chosen in advance, you should not pick a hash function relative to your data**

COLLISIONS

- **Hash table methods are defined by how they handle collisions**
- **Two main approaches**
  - Probing
  - Chaining

COLLISIONS

- **Probing**
  - Linear probing
    - Try the appropriate hash table row first
    - Increase the index by one until a spot is found
    - Guaranteed to find a spot if it is available
    - If the array is too full, its operations reach O(n) time

COLLISIONS

- **Probing**
  - Quadratic Probing
    - Rather than increasing by one each time, we increase by the squares
    - k+1, k+4, k+9, k+16, k+25
    - Certain tables can cause **secondary clustering**
    - Can fail to insert if the table is over half full

COLLISIONS

- **Probing**
  - Secondary Hashing
    - If two keys collide in the hash table, then a secondary hash indicates the probing size
    - Need to be careful, possible for infinite loops with a very empty array

COLLISIONS

- **Chaining**
  - Rather than probing for an open position, we could just save multiple objects in the same position
  - Some data structure is necessary here
  - Commonly a linked list, AVL tree or secondary hash table.
  - Resizing isn't **necessary**, but if you don't, you will get O(n) runtime.

PRIMALITY

- **Array sizes**
  - We normally choose our hash tables to have prime size
  - This is because for any number we pick, so long as it is not a multiple of our table size, they must be coprime
  - Two numbers x and y are **coprime** if they do not share any common factors.
  - If the hash table size and the secondary hash value are coprime, then the search will succeed if there is space available
  - However, many primes cause secondary clustering when used with quadratic probing

LOAD FACTOR

- **When discussing hash table efficiency, we call the proportion of stored data to table size the *load factor*. It is represented by the Greek character lambda (λ).**
  - We've discussed this a bit implicitly before
  - What are good load-factor (λ) values for each of our collision techniques?

LOAD FACTOR

- **Linear Probing?**
- **Quadratic Probing?**
- **Secondary Hashing?**
- **Chaining?**
- What are the tradeoffs?
  - Memory efficiency
  - Failure rate
  - Access times?

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
- **Secondary Hashing?** $0.25 < \lambda < 0.5$
- **Chaining?** $3.0 < \lambda < 10$
  - Because we allow multiple items in each space, we can increase memory efficiency by taking advantage
  - As long as there are a constant number in each space, we get O(1) runtimes.

LOAD FACTOR

- **As with most array data structures, you will need to resize when they get too full**
  - Here, these resizes are often for performance, rather than failure.
  - Hash table maintenance is important
  - Resizing is costly (but still O(n)) because you have to resize the array and rehash every element into the new table.

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times
  - Can be memory inefficient
  - Probing can fail, and delete with probing mechanisms is difficult
  - Chaining can be a good balance, but there is a lot of overhead maintaining all those data structures

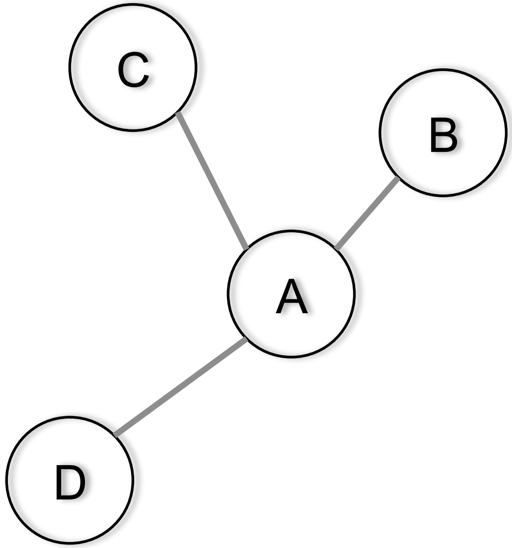
HASH TABLES

- Understand these tradeoffs and how these implementations work
- Section tomorrow will provide practice problems for each of these hash table methods

GRAPHS

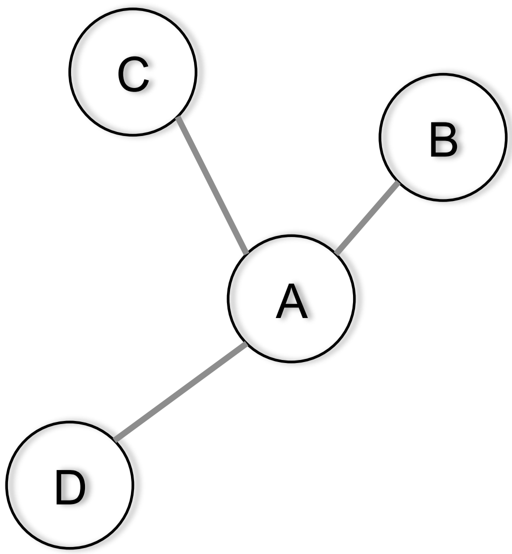
- **A graph is composed of two things**
  - A set of vertices
  - A set of edges (which are vertex tuples)
- **Trees are types of graphs**
  - Each of the nodes is a vertex
  - Each pointer from parent to child is an edge
- **Represented as G(V,E) to indicate that V is the set of vertices and E is the set of edges**

GRAPHS



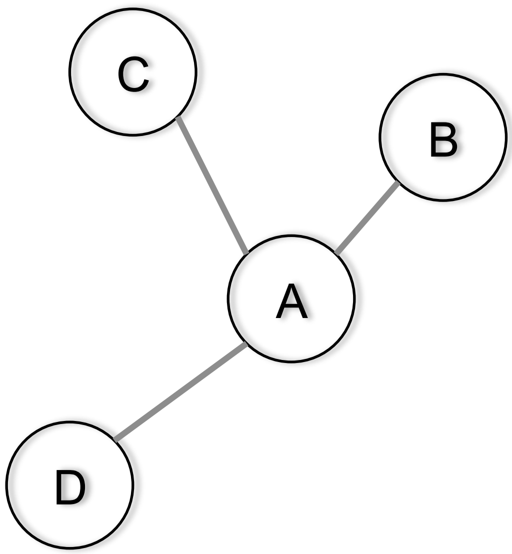
- **What this graphs vertices and edges?**

GRAPHS



- **What this graphs vertices and edges?**
  - V = {A, B, C, D}
  - E = {(A,B) , (A,C), (A,D)}

GRAPHS

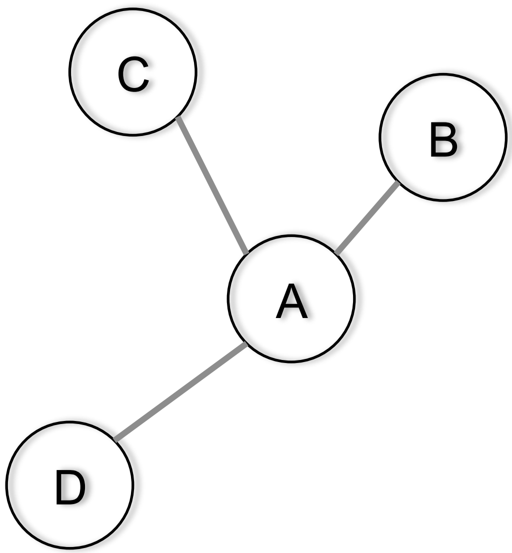


- **What this graphs vertices and edges?**
  - V = {A, B, C, D}
  - E = {(B,A) , (C,A), (D,A)}?

GRAPHS

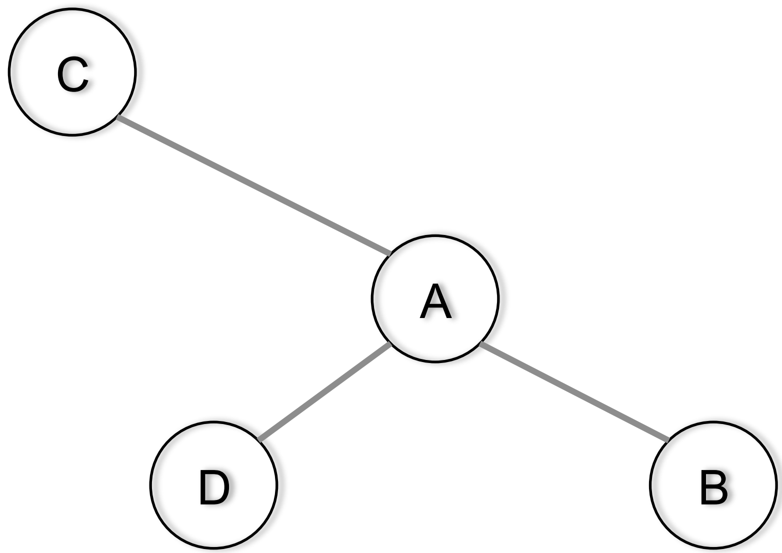
- **In that graph, the order did not matter**
- **This is called an undirected graph**
    - In undirected graphs, an edge from (A,B) means that there must be an edge from (B,A)
    - In implementation, both of these edges need to be present, but if you indicate that a graph is undirected, you do not need to indicate both in your list of edges

GRAPHS



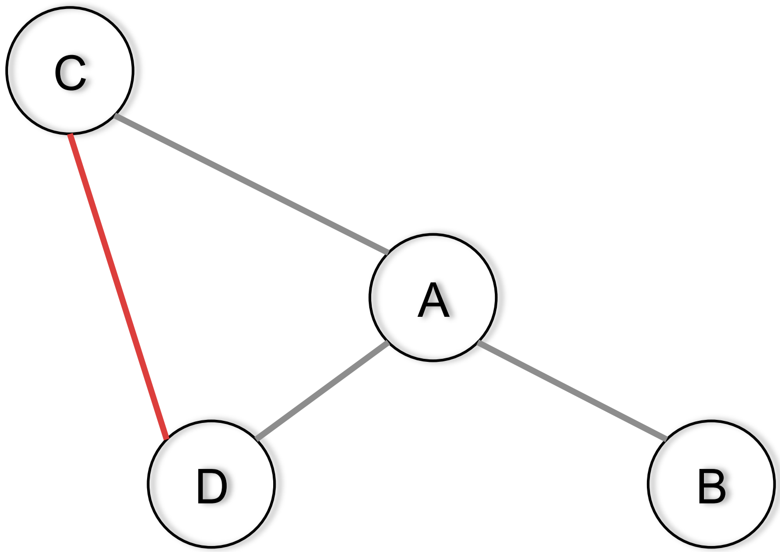
- **Is this graph a tree?**

GRAPHS



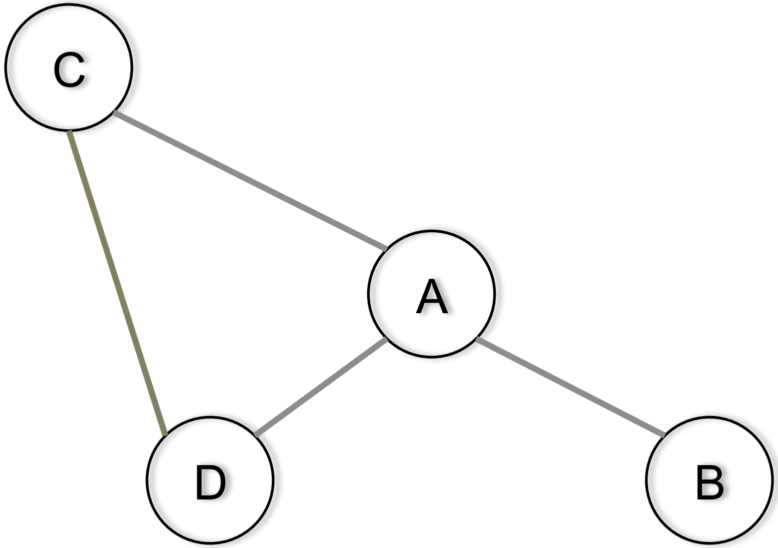
- **Is this graph a tree?**
  - Yes, you can make the shape by moving vertices around

GRAPHS



- **Is this graph a tree?**

GRAPHS

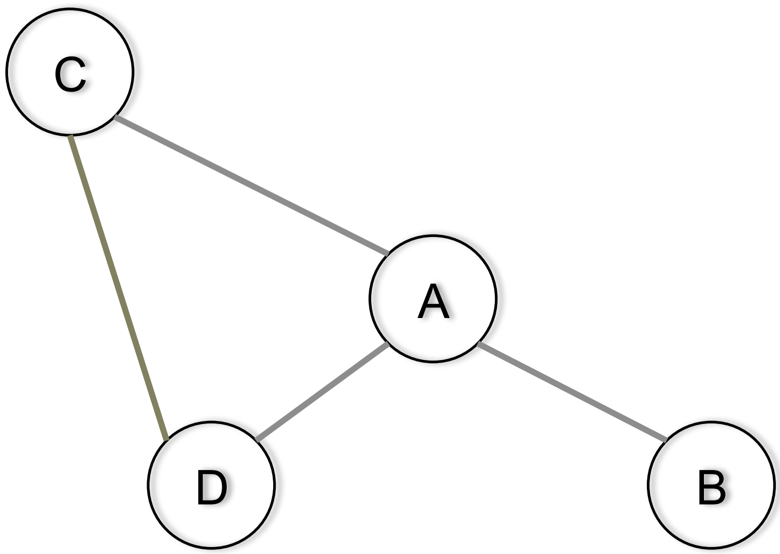


- **Is this graph a tree?**
  - No, there is no way to rearrange these vertices because there is a **cycle**

GRAPHS

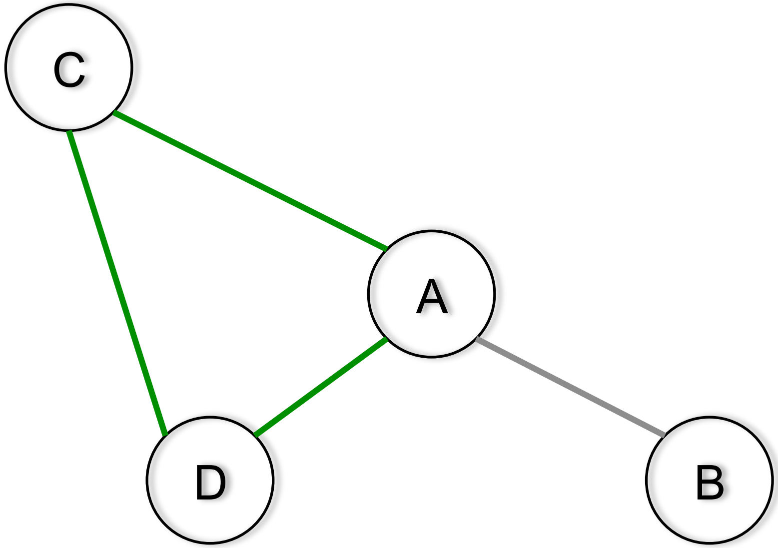
- A path is a set of edges which goes from one vertex to another in a graph
- A cycle is a path that starts and ends on the same vertex.
- A graph is *acyclic* if no cycles exist in the graph

GRAPHS



- **What is the cycle here?**

GRAPHS



- **What is the cycle here?**
  - (A,D) (D,C) (C,A)

GRAPHS

- **In paths and cycles, the sets of edges must pass from one vertex to another, i.e. each edge must share a vertex with some other edge.**
  - (A,B) (B,C) is a path from A to C, while (A,B) (D,C) is not.
  - There is no way to get from B to D

GRAPHS

- **Trees are acyclic graphs**
- **Graphs can be traversed**
  - Breadth-first
  - Depth-first
- **Edges can also have weights**
  - Path finding on a map
  - Route-optimization problems

NEXT CLASS

- More graphs!
- Discuss implementation approaches
- Prepare for path finding problem for next week