Week 9

p3b: final submission due before 10pm on Thursday, 3/28
p4: available soon
x4: due before 10pm on Monday 4/1
x5: meet with x-team coach for design review 4/1-4/8

W7 : due before 10pm Monday

Module: Week 9 (and start on week 10 before next week)

Team Project - A-Team (x-Team) - 2005 - start next monday.

THIS WEEK:

- DFS and BFS Spanning Trees
- Minimum Spanning Trees
  - Prim's
  - Kruskal's

} same with cost not necessary same ∴ tree

- Topological Ordering
- Dijkstra's Shortest Path algorithm
- Set operations

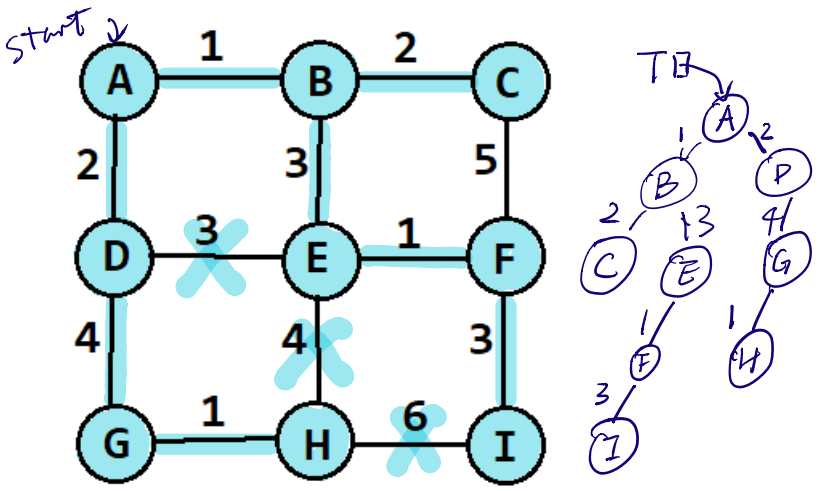
NEXT WEEK

- Linear Sorts (maybe next week)
  - radix
  - flashsort
- Java FX (in-class demo)
- Programming Project Assignment: Part 1, the Design
  (Start organizing your teams now – or keep your x-team for final Team Project)

Prim's Minimum Spanning Tree

minimum spanning tree: minimum cost – lowest sum of weights in S.T.
weighted, connected graph.
all S.T are minimum for unweighted graph
⇒ Prim's is "greedy" algorithm using local best.

1. pick a start vertex
2. make it the root of S.T.
3. at each iteration (to find a new edge) (N-1 edges, for / while loops)
   choose lowest weighted that connects a vertex without creating a cycle.

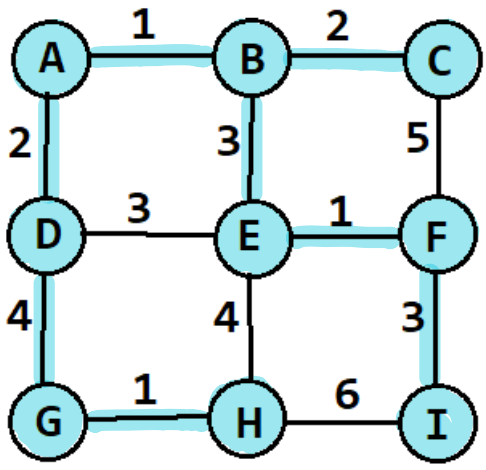


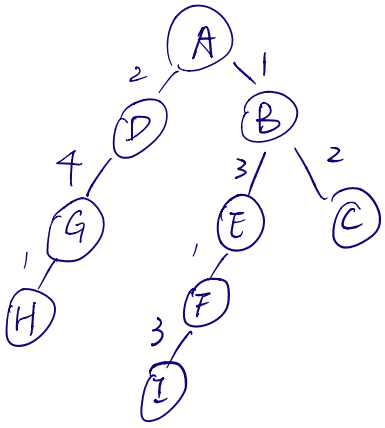
cost: 17

Kruskal's Minimum Spanning Tree

1. add all the vertices to M.S.T
2. Sort edges by weights - Priority Queue
3. At each iteration, add the edge with minimum weight that does NOT creat a cycle
4. pick a vertex to be root

	edge pick
A-B 1	1
B-C 2	5
A-D 2	4
B-E 3	6
~~C-F 5~~	
~~D-E 3~~	⇐create cycle.
E-F 1	2
D-G 4	8
~~E-H 4~~	
F-I 3	7
G-H 1	3
~~H-I 6~~	

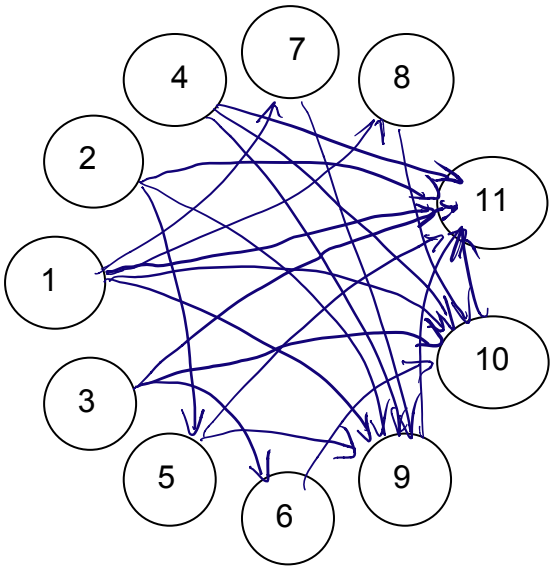


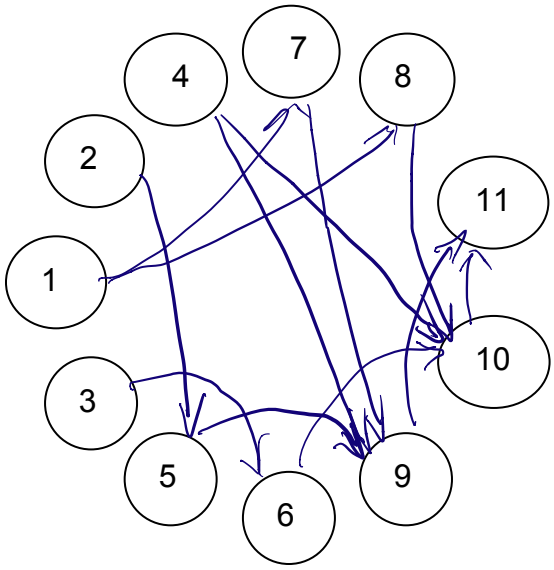


Cost: 17

Topological Ordering

1. get bread
2. get jelly
3. get peanut butter
4. get butter knife
5. open jelly
6. open peanut butter
7. take bread slice 1
8. take bread slice 2
9. use knife to spread jelly on bread slice
10. use knife to spread peanut butter on bread slice
11. put slices together with spreaded sides facing each other





IDEA:

Come up with a list of vertices such that each vertex comes before any successors

Directed Acyclic Graph.

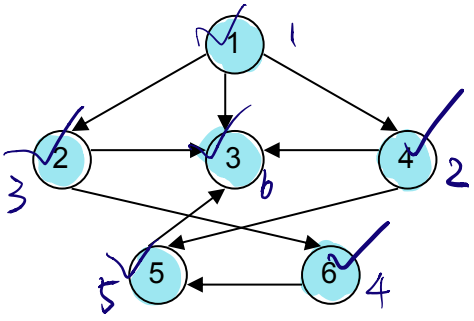
Topological Ordering

Iterative Algorithm (see readings for recursive algorithm)

Num = # vertices
ST = stack
mark all vertices unvisited
For each vertex with no pred
    mark as visited
    push to stack
while stack is not empty
    curr = ST.peek();
    if all the successors are visited;
        pop curr from stack;
        assign num to curr; // find the last item first.
        decrement Num;
    else
        select unvisited successor(u) of curr,
        mark u as visited;
        push u to stack;

* use CS400 convention choose alpha numerically

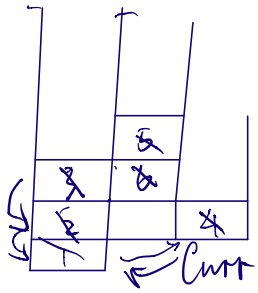
Example



Topological order:

Num = 6 5 4 3 2 1 0

ST



Curr: 1 2 3 3 2 5 6 2 4 1

Dijkstra's Algorithm

- Fastest, single-start, shortest path algorithm for arbitrary directed graph with unbounded non-negative edge weights.
  - Easiest $O(E + V \cdot \log_2 V)$
  - must supply a start vertex
  - arbitrary directed – unknown at start
  - unbounded – don't know
  - non-negative – 0 or greater edges vertices.
  - creates and use a pred list to construct shortest path from start.

Psuedo Code

```
for each vertex V
   initialize V's visited mark to false
   initialize V's total weight to "infinity"
   initialize V's predecessor to null

set start vertex's total weight to 0

create new priority queue pq
pq.insert( [start vertex total weight,start vertex] )

while !pq.isEmpty()
    [C's total weight,C] = pq.removeMin()
   set C's visited mark to true

   for each unvisited successor S adjacent to C
      if S's total weight can be reduced
         S's total weight = C's total weight + edge weight from C to S
         update S's predecessor to C
         pq.insert( [S's total weight,S] )
         (if S already in pq we'll just update S's total weight)
```

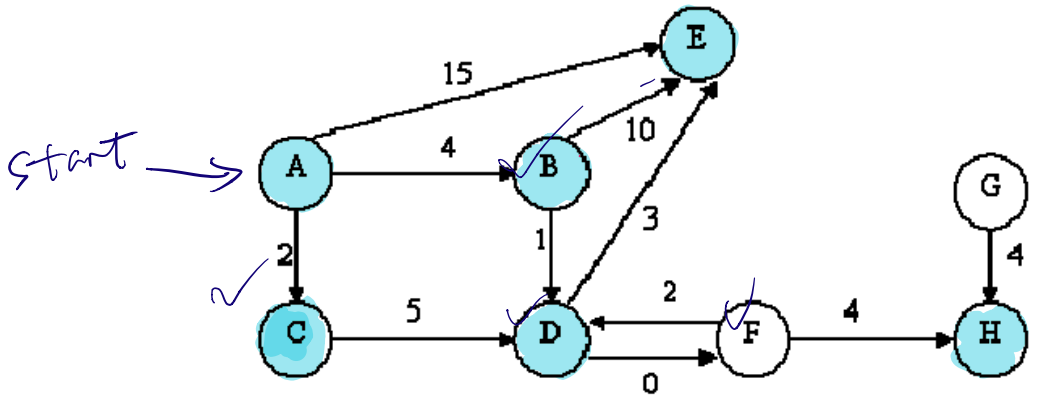
setup D.S.

total weight

running

* construct the path from pred list.

Dijkstra's Practice



Curr	Iteration	PQ Priority Queue (just list smallest to largest)
	0	[0, A]
[0, A]	1	[4, B] [2, C] [15, E]
[2, C]	2	[4, B] [15, E] [7, D]
[4, B]	3	[15, E] [7, D] [5, D] [14, E]
[5, D]	4	[15, E] [7, D] [14, E] [8, E] [5, F]
[5, F]	5	[15, E] [7, D] [14, E] [8, E] [9, H]
[8, E]	6	[15, E] [14, E] [9, H]
[9, H]	7	[15, E] [14, E]

Vertex	Visited	Total Weight	Predecessor
A	T	0	\
B	T	4	A
C	T	2	A
D	T	5	C
E	T	8	B
F	T	7	D
H	T	9	F
G	F	∞	\

Reconstruct shortest path from A to F A → B → D → F Backward

Reconstruct shortest path from C to F

Sets

Definition

- a collection of distinct items
- unordered
- can describe sets by words
- can be finite or infinite in size
- size is number of elements

Examples:

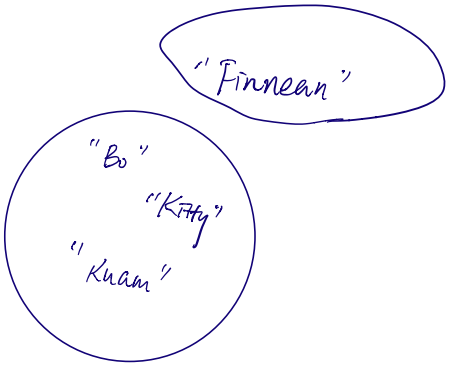
pet = {"Bo", "Kitty", "Kuan"};

S = students in CS400

letters = {a..z, A..z};

digits = {0..9}

diagram sets with Venn diagrams



Terminology

subset A is a subset of B if all members in A are in B

proper subset A is a proper subset of B. B is not a subset of A.

superset A is a superset of B: if all members of B are in A

proper superset A is a proper superset of B. B is not a proper superset of A

unit set a set with one item

disjoint sets A is disjoint of B. if no items in A are in B

{1,2,3} is disjoint of {4,5,6}

Notation

https://www.rapidtables.com/math/symbols/Set_Symbols.html

$\emptyset$ empty set

$\mathbb{N}$ nature numbers

$\mathbb{R}$ real, floating decimal point values

powerset
all subset + ϕ + 本身

$\mathbb{U}$ $\mathbf{U}$ universal set

$\mathbf{A} \in \mathbf{letters}$ A is a number of set letters

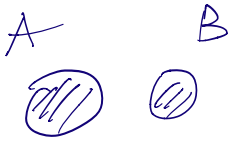
$|\mathbf{A}|$

$\mathbf{A} \subseteq \mathbf{B}$	A is a subset of B	$\mathbf{A} \subset \mathbf{B}$	A is a proper subset of B
$\mathbf{A} \nsubseteq \mathbf{B}$	A is not - - -	$\mathbf{A} \not\subset \mathbf{B}$	A is not - - - -
$\mathbf{A} \supseteq \mathbf{B}$	superset	$\mathbf{A} \supset \mathbf{B}$	proper superset
$\mathbf{A} \not\supseteq \mathbf{B}$	not - - -	$\mathbf{A} \not\supset \mathbf{B}$	not - - - -

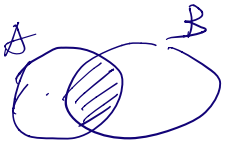
$\mathbf{A}'$ complement of A

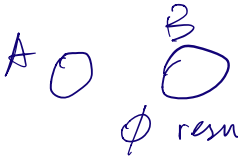
Basic Set Operations

$\mathbf{A} \cup \mathbf{B}$ A union B



$\mathbf{A} \cap \mathbf{B}$ A intersection B

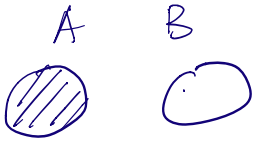




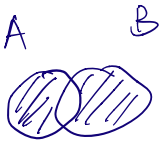
ϕ result.

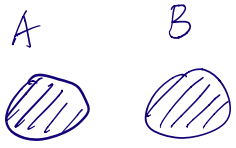
$\mathbf{A} - \mathbf{B}$ } A complement B

$\mathbf{A} \setminus \mathbf{B}$ }



$\mathbf{A}\Delta\mathbf{B} = (\mathbf{A} \setminus \mathbf{B}) \cup (\mathbf{B} \setminus \mathbf{A})$





symmetric differences

SetADT

Operations in java.util.Set

```
boolean add(E e) - add if item is not present
boolean contains(Object o) - true iff o is present
boolean remove(Object o) - remove o if present
boolean isEmpty() - true if no elements
int size() - returns number of elements
```

Implementation

Complexity analysis, if N is number of nodes

	insert	lookup	remove	iteration
Array	O(N)	O(N)	O(N)	ordered (not sorted)
Sorted array	O(N)	O(log₂N)	remove/shift. O(N)	sorted order.
Linked list	O(N) average	O(N)	O(N)	ordered, not sorted
Sorted Linked list	O(N) but O(1)	O(N) av O(N/2) be O(1)	O(N) av O(N/2) be O(1)	Sorted
BST	O(N)	O(N) bOC1) O(H)	O(N) O(H)	sorted (in order traversal)
Balanced search tree	O(log₂N)	O(log₂N)	O(log₂N)	sorted
Hash table (worst case linear)	O(1)	O(1)	O(1)	no particular order

De Morgan's Laws

If A and B are any two sets then:

```
(A union B)'  = A' intersection B'
(A intersection B)' = A' union B'
```

$(A \cup B)' = A' \cap B'$

$\overline{A+B} = A'B'$

$\overline{AB} = A' + B'$

Java's Set is an interface