Week 4: Exhaustive Search, Recurrences, Graphs

Recurrences:

- Recurrence for mergesort ([optional] Q4.2)
- Solving some recurrences (Q4.1)

Exhaustive search:

- Subset-sum problem (Problem Q4.3)
- [time permitting] Partition problem (Problem Q4.4)

Graphs:

- Representation (Q4.5, Q4.6)
- [time permitting] Sparse & dense graphs (Problem Q4.7)

Lab: **R U OK with the Makefile Task last week?**

- [from last week] Create linked list module & test it
- Create minimal-effort stack module & test it

Coming soon

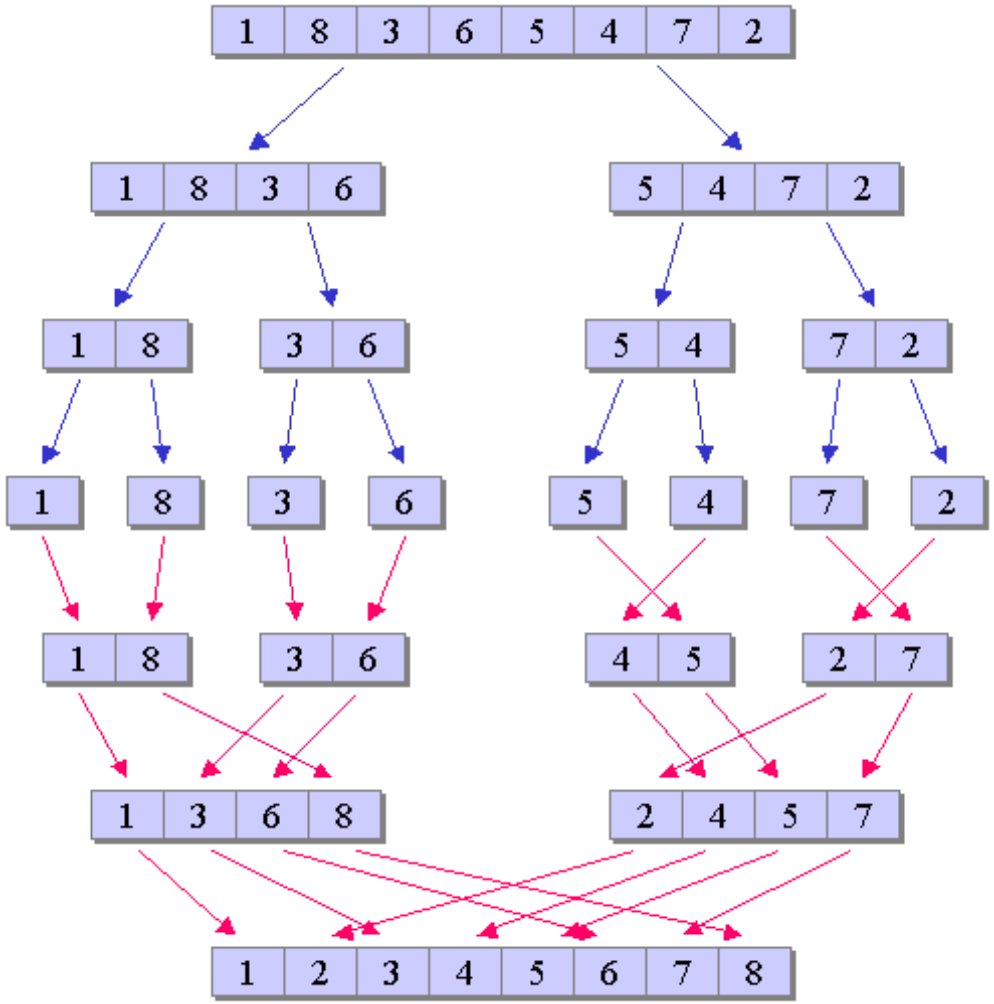
- MST in Week 5, worth 10%, know
  - When, Where
  - Coverage: lectures weeks 1-3 + workshops weeks 2-4
- Assignment 1, due around Week 6, worth 10%.

Recurrences: mergesort (Q4.2) or ... how to get running time of recursive algo

Mergesort is a divide-and-conquer ...

It can be easily implemented as a recursive algo.

The Task: *Construct a recurrence relation to describe the runtime of mergesort sorting n elements. Explain where each term in the recurrence relation comes from.*



How to find complexity of recursive functions?

Just like mergesort:

- First, build the complexity as a recurrence
- Then, “solve” the recurrence, for example:

$$T(n) = 2\,T(n/2) + n$$
$$T(1) = 0$$

$$T(n) = \Theta(\,n\log n\,)$$

How to “solve” a recurrence?

Recurrences: Q4.1 extended, Group Work: (a OR c) AND d

Solve the following recurrence relations, assuming $T(1) = 1$.

Example: $T(n)=T(n-1)+n$

DIY:

a) $T(n)=T(n-1)+4$

b) $T(n)=T(n-1)+n$

c) $T(n)=2T(n-1)+1$

d) $T(n)= 2T(n/2) + n$

e) $T(n)= T(n/2) + 1$

Tips:

- *Don't rush*
- *Be lazy 🤣: delay the tempting sum calculation until the end*

Telescoping: 🌕 → 🌖 → 🌗 → 🌘 → … → 🌑

1. Start from the given recurrence

`T(n) = T(🌖) + f(n)`

work out T(🌖) by replacing each appearance of n in the given recurrence with 🌖, now:

`T(n)= T(🌖) + f(n)`

`= T(🌗) + f(🌖) + f(n)`

2. Continue to work out `T(🌗)` in the same manner, do further *until seeing a pattern*.

3. Parameterize the pattern so that:

`T(n) = T(p(n,k)) + f'(n,k)`

then work out `k` so that `p(n,k)= 1` . Since `T(1)` is given, we just need to substitute `k` into `f'(n,k)` and "beautify" it.

exhaustive search = brute-force search = generate-and-test

exhaustive search:
What?
Why? When?
How?

Are linear search and binary search exhaustive?

exhaustive search = brute-force search = generate-and-test

Exhaustive search= go through all “potential solutions” until an answer is found (or until the end if not found at all).

Difficult Part:
Generate S = the collection of all solution candidates

Notes:
Normally, recursive procedures are easier to build than the iterative ones.

Question 4.3: subset-sum problem

Design an *exhaustive-search algorithm* to solve the following problem: Given a set S of n positive integers and a positive integer t, does there exist a subset $S' \subseteq S$ such that the sum of the elements in S' equals t, i.e.,

$$\sum_{i \in S'} i = t.$$

If so, output the elements of this subset S' *[we relax the output requirement if we can identify* S'*]*

Assume that the set is provided as an array of length n.

An example input may be $S = \{1,8,4,2,9\}$ and $t = 7$, in which case the answer is Yes and the subset would be $S' = \{1,4,2\}$.

What is the time complexity of your algorithm?

Q4.3: approaches = ???

```
// S: set of positive integers, t: an integer
// returns YES if there is a subset S'⊆ S
//   such that the sum of elements in S' equals  t
// return NO if otherwise

function subset_sum(S, t)
```

Q: – What is a possible solution?
– What is the set of all possible solutions?

Q4.3: refining

```
function subset_sum(S, t)
  for each subset S’ of S do
    if (t = sum of elements in S’) then
      return YES
  return NO
```

Q4.3: refining

```
function subset_sum(S, t)
  for each S’ in Powerset(S) do
    if (t = sum of elements in S’) then
      return YES
  return NO
```

```
// find all the subsets, aka. the powerset, of the set S
function Powerset(S)
```

How?

Can we use divide-&-conquer? reduce-&-conquer?

```
    // Do it in your draft paper!
```

Q4.3 Check your algorithm

```
function subset_sum(S, t)
  for each S' in Powerset(S)
    if (t = sum of elements in S')
      return YES
  return NO
```

```
for each S' in POWERSET(S) do
    if ∑_{i∈S'} i = t then
        return YES
return NO
```

```
// returns the powerset of the set S
function Powerset(S)
  if (S is ø)return {ø}
  pick x from S
  S' := S - {x}
  P' := Powerset(S')
  P := P'
  for each element s of P'
    P := P U (s U {x})
  return P
```

$x \leftarrow$ some element of S
$S' \leftarrow S \setminus \{x\}$
$P \leftarrow \text{POWERSET}(S')$
return $P \cup \{s \cup \{x\} \mid s \in P\}$

Complexity =?

Q4.3 Check complexity

```
function subset_sum(S, t)
  for each S' in Powerset(S)
    if (t = sum of elements in S')
      return YES
  return NO


// returns the powerset of the set S
function Powerset(S)
  if (S is ø)return {ø}
  pick x from S
  S' := S - {x}
  P' := Powerset(S')
  P := P'
  for each element s of P'
    P := P U (s U {x})
  return P
```

$T(n) = \max(P(n), n\,2^n)$

$= O(n2^n)$

Basic operation= ?

P(n)=

$P(n) = \theta(2^n)$

Q4.4 [time permitting]: Partition problem

Design an exhaustive-search algorithm to solve the following problem:

Given a set S can we find a partition (i.e., two disjoint subsets A;B such that all elements are in either A or B) such that the sum of the elements in each set are equal, i.e.,

$$\sum_{i \in A} i = \sum_{j \in B} j.$$

If so, which elements are in A and which are in B?

Again, assume S is given as an array.

For example , if `S = [1, 8, 4, 2, 9]` one valid solution is `A = [1, 2, 9]` and `B = [8, 4].`

Can we make use of the algorithm from Problem 3? What's the time complexity of this algorithm?

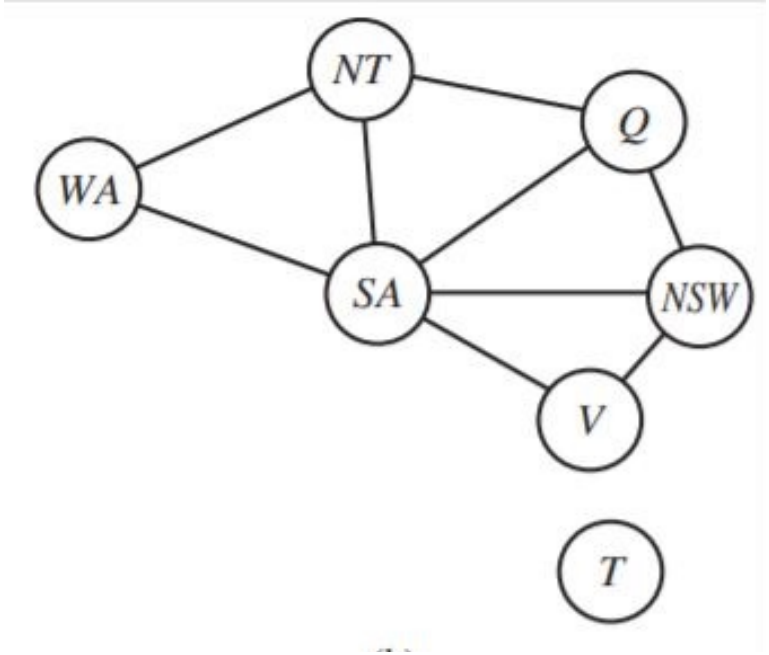
"Reduction" Technique

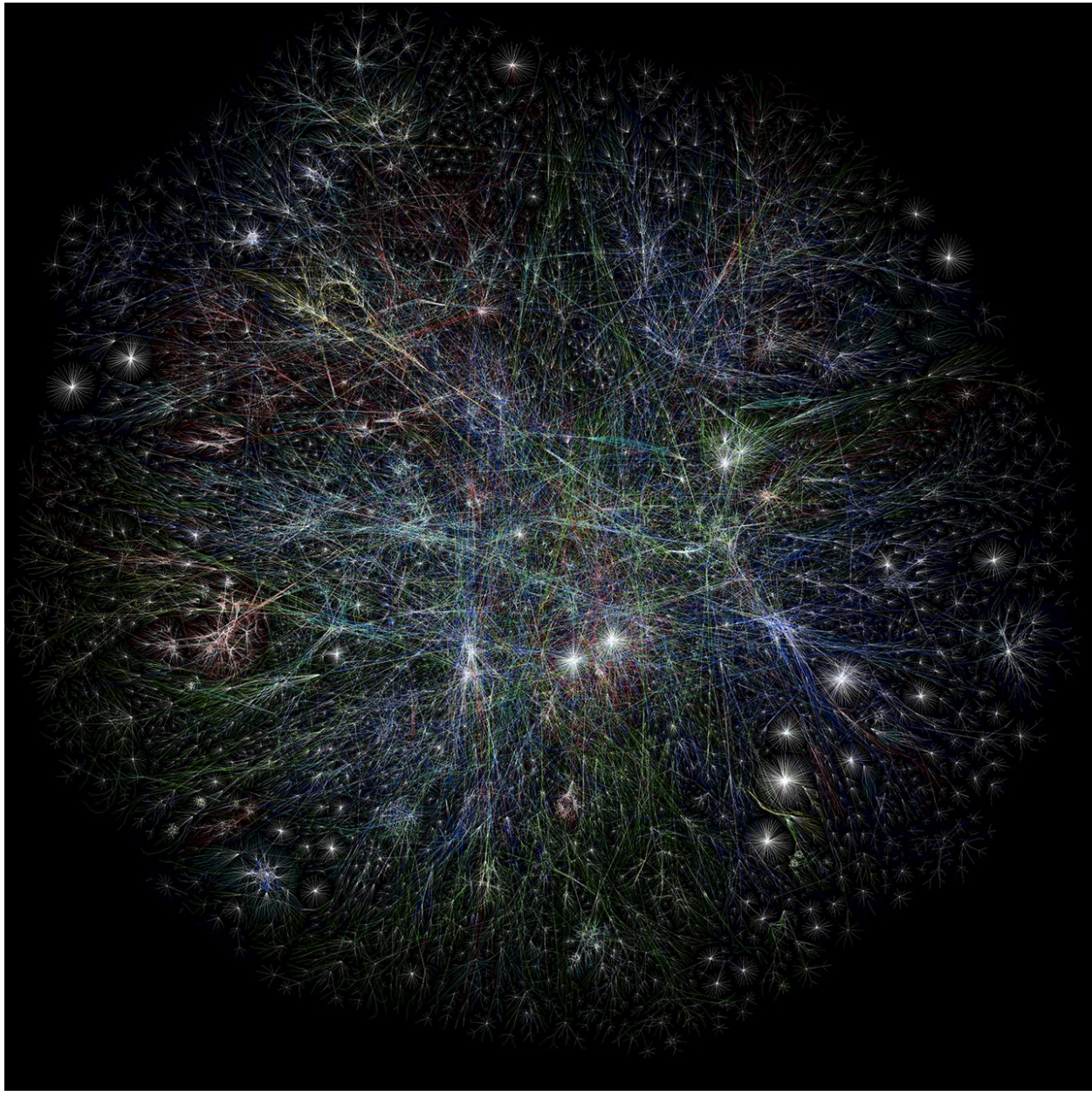
- If we can reduce a current task to a well-known task, we'll have solution "for free".
- In practice, we apply this technique quite frequently, without noticing it.

Graphs: understanding basic concepts

Formal definition: **G= (V, E)**

- *vertex (node), edge*
- *graphs: dense, sparse, directed(di-graph), undirected, cyclic, acyclic, DAG, weighted, unweighted*

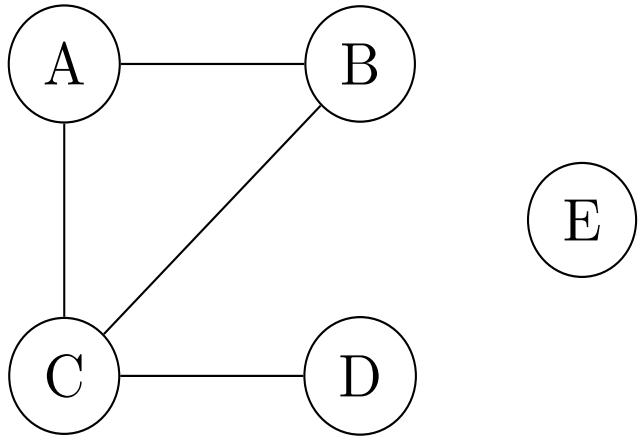




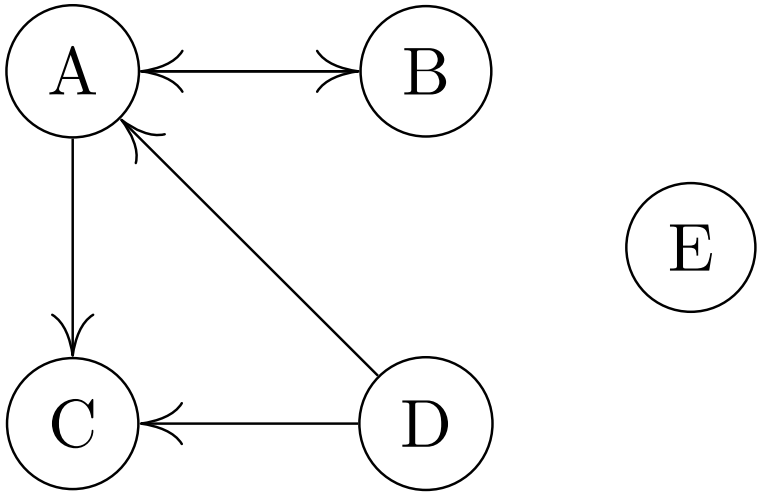
Q4.5 For each graph, give:

1. node that has the highest degree.
2. the representations as
   a) sets of vertices and edges,
   b) adjacency lists,
   c) adjacency matrices.

(a) An undirected graph:



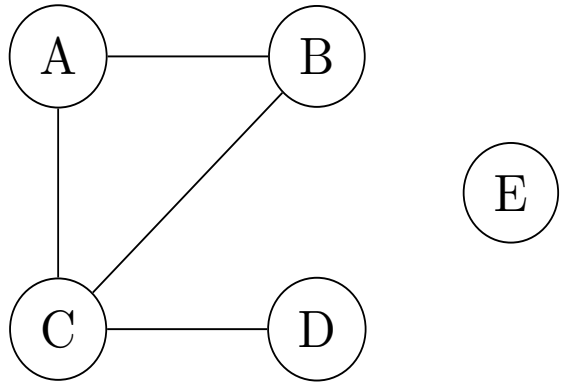
b) a directed graph:



Q4.5 a) For the graph, give:

1. node that has the highest degree.
2. the representations as
   a) sets of vertices and edges,
   b) adjacency lists,
   c) adjacency matrices.

(a) An undirected graph:



1) 2a)

2b) Adjacency lists

A →

B →

C →

D →

E →

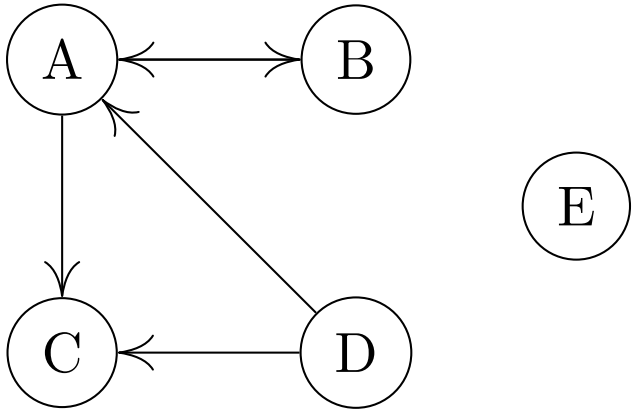
2c) Adjacency Matrix

	A	B	C	D	E
A					
B					
C					
D					
E					

Q4.5 b) For the graph, give:

1. node that has the highest degree.
2. the representations as
   a) sets of vertices and edges,
   b) adjacency lists,
   c) adjacency matrices.

b) a directed graph:



1) 2a)

2b) Adjacency lists 2c) Adjacency Matrix

A →

B →

C →

D →

E →

	A	B	C	D	E
A					
B					
C					
D					
E					

Notes for Graph Algorithms

If a graph acts as an input and/or an output of a graph algorithm, it's normally specified as `G=(V,E)`.

```
function Newgraph(G= (V,E))
    // compute V'
    // compute E'
    G' := (V',E')
    return G'
```

That doesn't imply the "sets of vertices and edges" graph representation.

If we want to discuss the complexity of a graph algorithm:

- we need to explicitly note the representation of the graph and use, or
- we might need to specify the complexity for each of the 3 representations.

Question 4.6

Different graph representations are favourable for different applications. What is the *time complexity* of the following operations if we use (i) adjacency lists (ii) adjacency matrices or (iii) sets of vertices and edges?

a) determining whether the graph is *complete*

b) determining whether the graph has an *isolated node*

Assume that the graphs are undirected and contain *no self-loop* (edge from a node to itself). A complete graph is one in which there is an edge between every pair of nodes. An isolated node is a node which is not adjacent to any other node. Assume that the graph has n vertices and m edges.

4.6 a) time complexity of determining whether the graph is complete when using

Adjacency Lists	Adjacency Matrices	Set of Vertices & Edges

$A \rightarrow B, C$
$B \rightarrow A, C$
$C \rightarrow A, B, D$
$D \rightarrow C$
$E \rightarrow$

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$G = (V, E)$, where

$V = \{A, B, C, D, E\}$

$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}\}$

Assumptions: 1. undirected, no self-loop, n vertices, m edges
2. m is not available [why?]

4.6 a) time complexity of determining whether the graph has an isolated node when using

Adjacency Lists	Adjacency Matrices	Set of Vertices & Edges

$A \rightarrow B, C$
$B \rightarrow A, C$
$C \rightarrow A, B, D$
$D \rightarrow C$
$E \rightarrow$

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

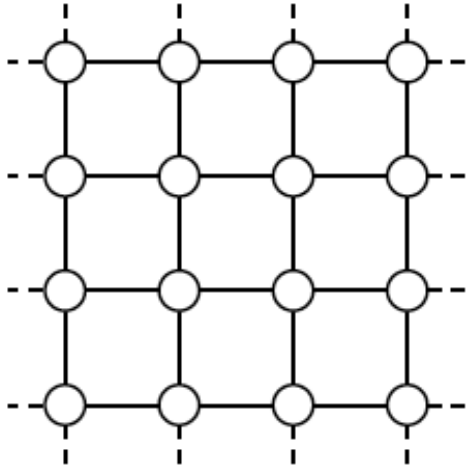
$G = (V, E)$, where

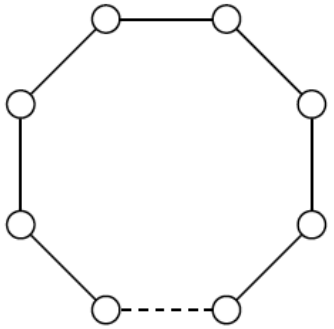
$V = \{A, B, C, D, E\}$

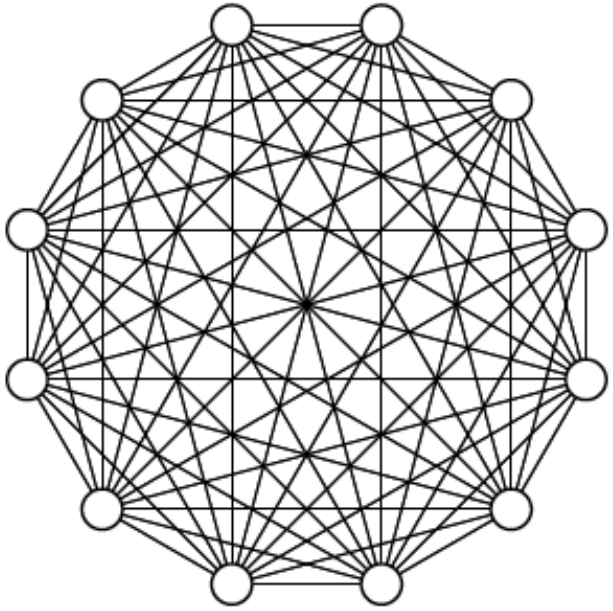
$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}\}$

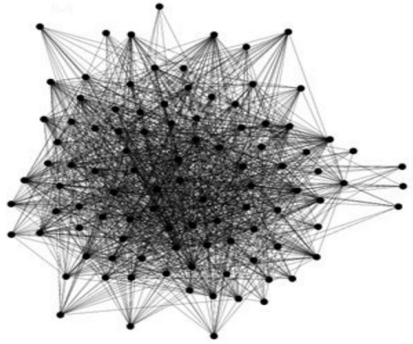
Assumptions: 1. undirected, no self-loop, n vertices, m edges

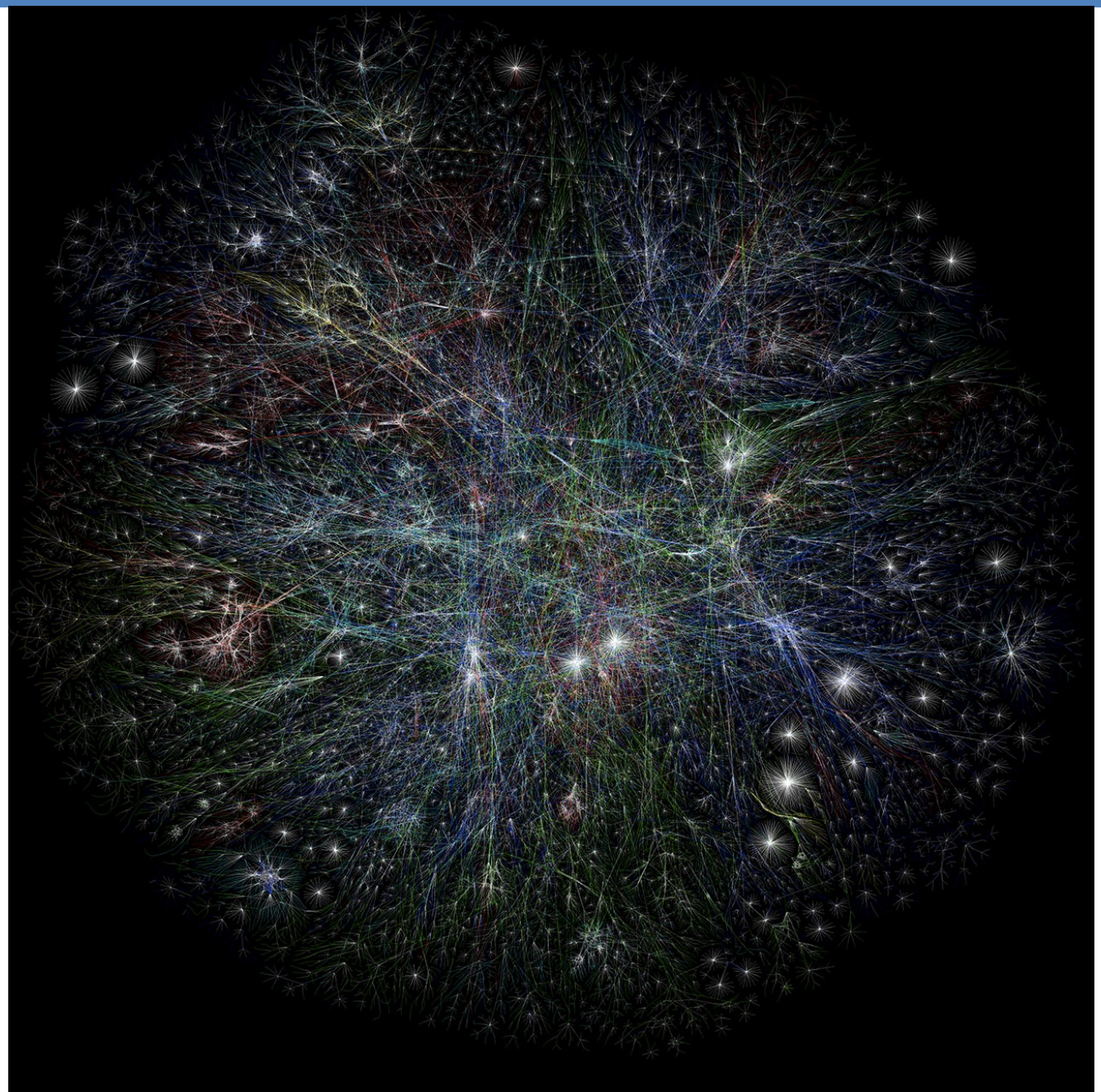
Sparse & Dense Graphs: which one sparse, which one dense? Why?











Q4.7: [time permitting]: Sparse and Dense Graphs

We consider a graph to be sparse if the number of edges, $m \in \Theta(n)$, dense if $m \in \Theta(n^2)$.
(Notes: to be more precise, sparse if $m \in O(n)$, dense if $m \in \boldsymbol{\Omega}(n^2)$*).*

What is the *space complexity* (in terms of n) of a sparse graph if we store it using:

(i) adjacency lists :

(ii) adjacency matrix :

(iii) sets of vertices and edges :

Lab

Task 1: give private questions if in need

Task 2:

Build module linked list with functions `create_list`, `free_list`, `insert_at_front`, `insert_at_end`, `remove_at_front`, `remove_at_end`.

- Create `test_list.c` and `Makefile` with target list for testing linked list
- Using the list module, and with least effort, build module stack
- Create `test_stack.c` and add target stack for testing the stack module

Note: *if needed, google "`listops.c`" and use it as a first draft*

MST:

- Know when & where
- Coverage: lectures week 1-3 and workshops week 2-4
- Review and give questions now (next week is better for Assignment 1 questions)

Assignment 1:

- Understand the task before next workshop (Week 5)
- Q&A during Week 5 workshop
- Finish by Week 6 Workshop
- Ask refinement questions during Week 6 workshop