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 permitted] Partition problem (Problem Q4.4)

Graphs:

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

Lab:

- Questions with racecar last week?
- Create linked list module & test it
- Create minimal-effort stack module & test it

Events:

- Mid-semester test in Week 5, worth 10%.
- Assignment 1, due around Week 6, worth 10%.

Recurrences: mergesort (Q4.2)

Mergesort is a divide-and-conquer sorting algorithm made up of three steps (in the recursive case):

1. Sort the left half of the input (using mergesort)
2. Sort the right half of the input (using mergesort)
3. Merge the two halves together (using a merge operation)

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.*

Answer:

Recurrences: Q4.1 extended

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

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

Solving Recurrences using Substitution:

- Use the recurrence, and substitute each appearance of n with something smaller, for instance, with $n-1$ (for a,b,c) or $n/2$ (for d,e)
- Repeat a few substitutions until seeing a clear pattern
- Use the pattern to get down to $T(1)$

4.1.b

Answer:

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

=

=

=

=

=

=

=

=

4.1.d

Answer:

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

=

=

=

=

=

=

=

4.1.a

Answer:

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

=

=

=

=

=

=

=

=

4.1.c

Answer:

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

=

=

=

=

=

=

=

=

4.1.e

Answer:

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

$=$

$=$

$=$

$=$

$=$

$=$

$=$

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).

Perhaps the most difficult part is to generate S = the set of all possible cases…

Examples:

Notes:

- In many exhaustive search cases, recursive procedures are easier to build than 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'.

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 S' in POWERSET(S) do
    if (t = sum of elements in S') then
      return YES
  return NO


// return the powerset of the set S
function POWERSET(S)
  if (S is ∅)return {∅}
  ??? need to call POWERSET  for a smaller set
  ???
```

Complexity analysis : Suppose that T(n) and P(n) are running time of the first and the 2nd alg.

P(n)=

T(n)=

Q4.4 [optional, time permitted]: 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 for 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?

???

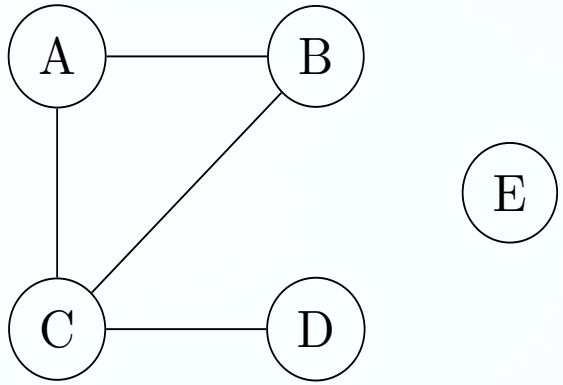
Graphs

- Formal definition: `G= (V, E)`
- concepts:
  - *vertex (node), edge*
  - *graphs: dense, sparse, directed(di-graph), undirected, cyclic, acyclic, DAG, weighted, unweighted*
- Representation of graphs
  - adjacency matrix
  - adjacency lists
  - set V of vertices + set E of edges

Q4.5 a) For the graph, give:

1. sets of vertices and edges,
2. nodes that have highest degree ,
3. the representations as adjacency lists, adjacency matrices.

(a) An undirected graph:



1: V={…} E={…}

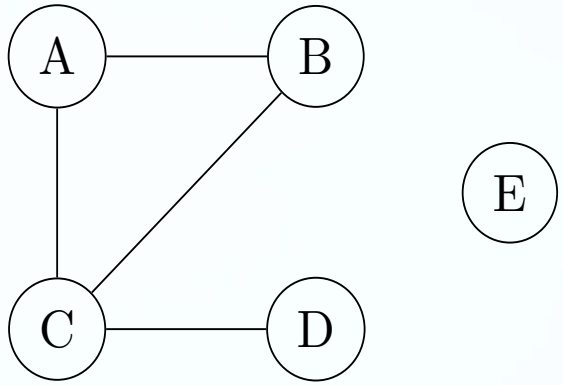
2: ?

3: ?

Q4.5 a) For the graph, give:

1. sets of vertices and edges,
2. nodes that have highest degree ,
3. the representations as adjacency lists, adjacency matrices.

(a) An undirected graph:



Adjacency lists

A →

B →

C →

D →

E →

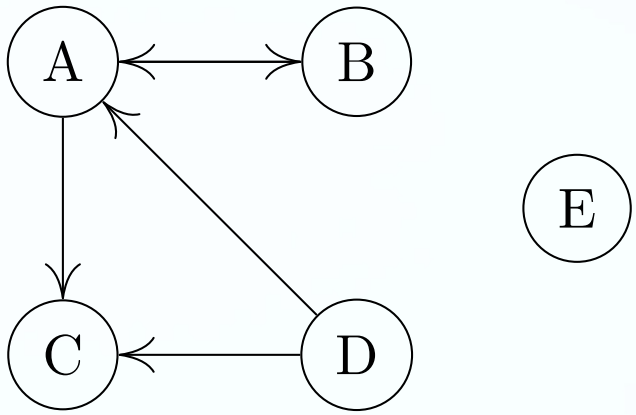
Adjacency Matrix

	A	B	C	D	E
A					
B					
C					
D					
E					

Q4.5 b) For the graph, give:

1. sets of vertices and edges,
2. nodes that have highest degree (in- and out-degree),
3. the representations as adjacency lists, adjacency matrices.

b) a directed graph:



1: V={…}, E={…} 2: ? ?

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
- Often, we need to explicitly note the representation of the graph and use that to compute the complexity of the algorithm.
- 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.

Q4.6 a) determining whether the graph is complete

What is the *time complexity* of the above operations if we use: fill in your solution here!

Adjacency Lists	Adjacency Maxtrices	Set of Vertices & Edges

Q4.6 a) What if m= |E| is *not* available?

Adjacency Lists	Adjacency Maxtrices	Set of Vertices & Edges
needed: if number of elements in each list is available: If not:	needed:	needed:

Sample graph for illustration:

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

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

$G = (V, E)$, where
$V = \{A, B, C, D, E\}$

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

Q4.6 b) determining whether the graph has an isolated node: fill in

Adjacency Lists	Adjacency Maxtrices	Set of Vertices & Edges
needed:	needed:	needed:

Would m help here?

Sample graph for illustration:

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

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

$G = (V, E)$, where
$V = \{A, B, C, D, E\}$

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

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

We consider a graph to be sparse if the number of edges, $m \in \Theta(n)$, dense if $m \in \Theta(n^2)$.

Give examples of types of graphs which are sparse and types which are dense.

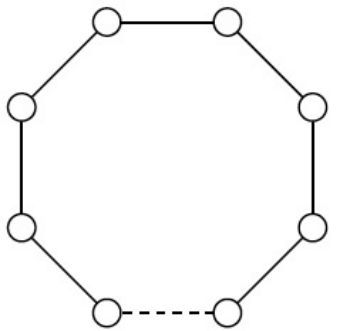
What is the *space complexity* (in terms of n) of a sparse graph if we store it using:

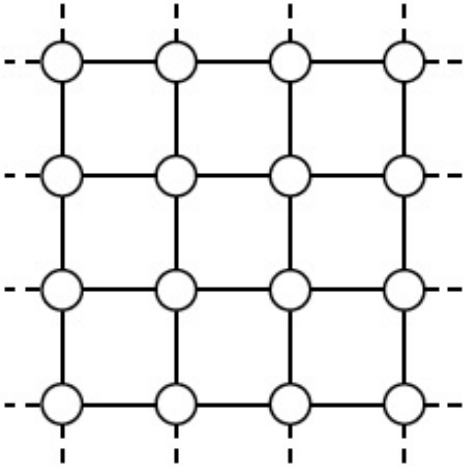
(i) adjacency lists

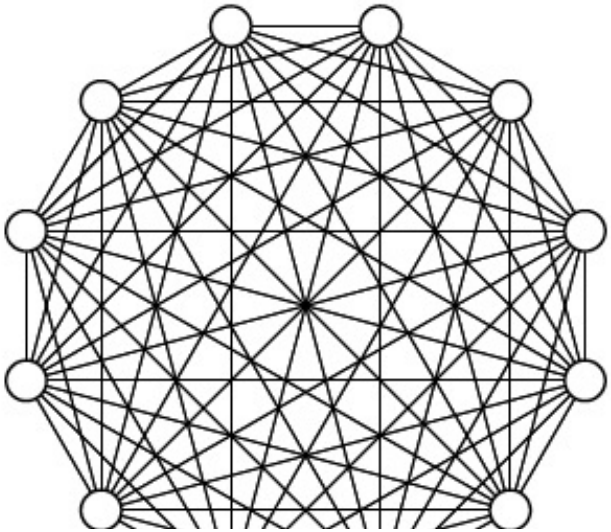
(ii) adjacency matrix

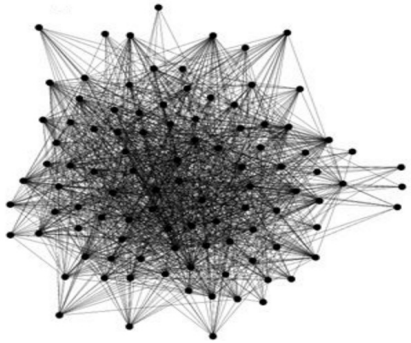
(iii) sets of vertices and edges

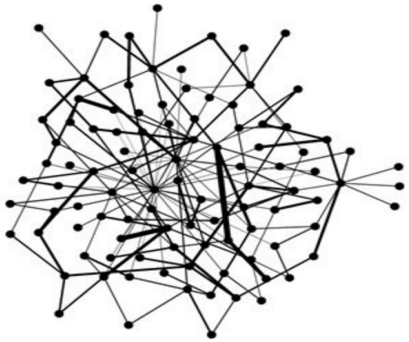
Sparse or Dense?



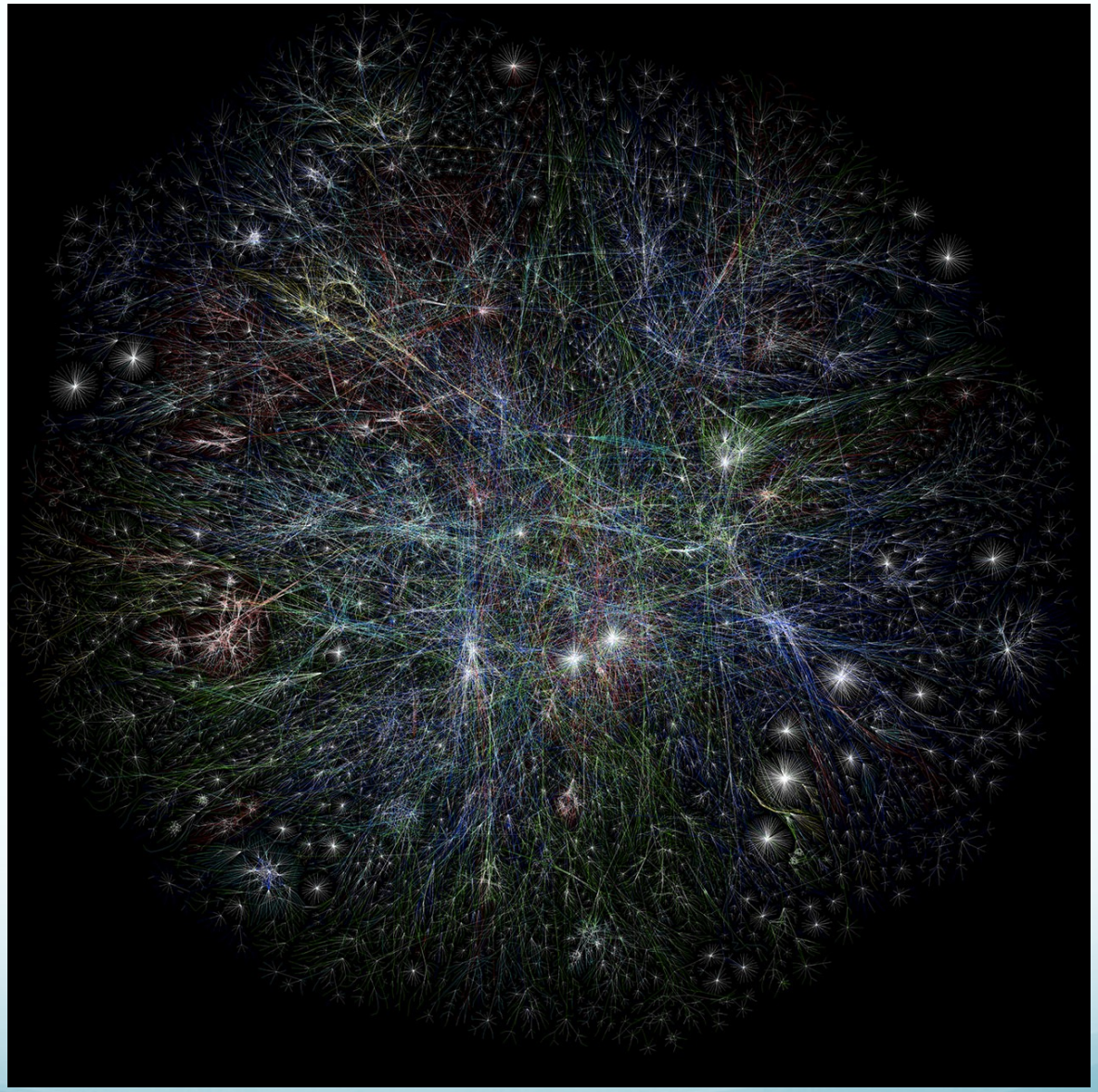








The Internet: Sparse or Dense?



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

We consider a graph to be sparse if the number of edges, $m \in \Theta(n)$, dense if $m \in \Theta(n^2)$.

Give examples of types of graphs which are sparse and types which are dense.

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 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 reference*

MST:

- Know time, procedure & how to start and submit
- Coverage: probably lectures week 1-3 and workshops week 2-4
- Review and give question now (or next week if applicable)

Assignment 1:

- Understand the task before next workshop (Week 5)
- Ask question during Week 5 workshop
- Finish most of the work by Week 6 Workshop
- Ask refinement questions during Week 6 workshop