Week 4: Exhaustive Search, Recurrences, Graphs

Recurrences:

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

Exhaustive search:

- Subset-sum problem (Problem 3)
- [time permitted] Partition problem (Problem 4)

Graphs:

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

Lab:

- Working with `dimefox`: using your last week's `racecar` module
- Create linked list module & test it
- Create minimal-effort stack and queue modules & test them

Recurrences: mergesort (Problem 2)

- **Recall**: 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.*

Recurrences: Problem 1

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 $T(1)$

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.
- But recursive procedures are normally less efficient, at least in terms of memory complexity (due to the memory stack for recursive calls)

Problem 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?

P3: approaches = ???

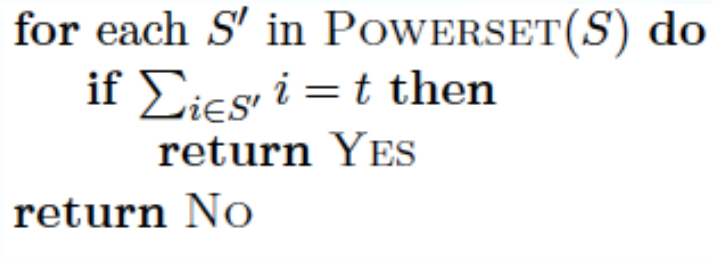
```
// S: set of positive integers, t: an integer
// returns YES if there is a subset S'⊆ S
//   such that the sum of elements in S' equals  t
// return NO if otherwise

function subset_sum(S, t)
```

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

P3: refining

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




```
// return the powerset of the set S
function POWERSET(S)
  if (S is {})return {}
  // reduce to POWERSET of |S|-1 elements
       ???
  // example: how to use POWERSET({1,2}) to get
POWERSET({1,2,3})
```

P3: refining

3	{a,b,c}	use POWERSET({a,b})= { {}, {a}, {b}, {a,b} } to build POWERSET({a,b,c}) { {},{a},{b},{a,b}, {c}, {a,c}, {b,c}, {a,b,c} }

```
// returns the powerset of the set S
function POWERSET(S)
  if (S is {})return {}
  pick x from S
  S' ← S - {x}              // S' is S after removing x
  P ← Powerset(S')
  for each element s of P
    s ← s ∪ {x}             // add x to s
    P ← P ∪ s               // add s to P
  return P
```

T1: final, and what's the complexity?

```
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 {})return {}
  pick x from S
  S' ← S - {x}
  P ← Powerset(S')
  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 $P \cup \{s \cup \{x\} \mid s \in P\}$

Problem 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?

Ideas= ???

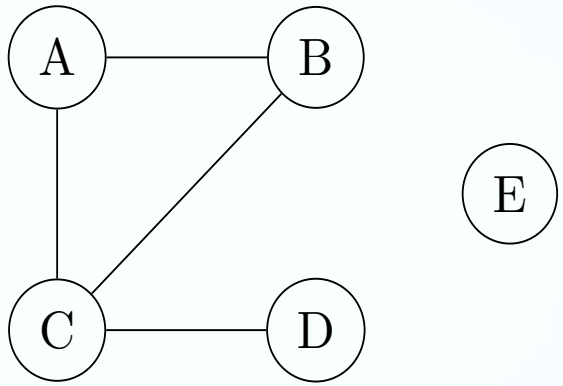
Graphs

- 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

P.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={…}

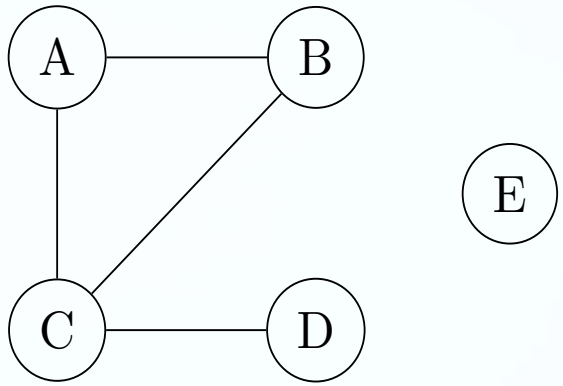
2: ?

3: ?

P.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 →

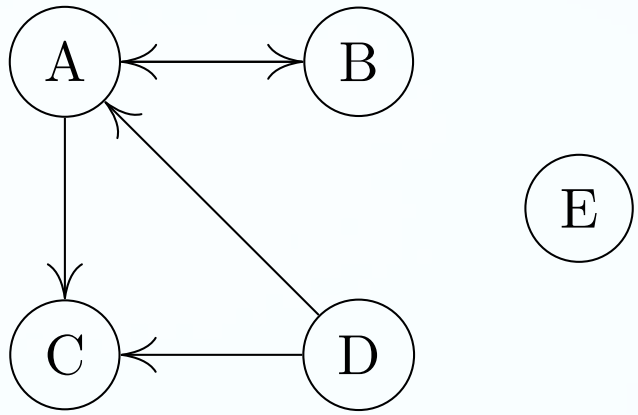
E →

Adjacency Matrix

	A	B	C	D	E
A					
B					
C					
D					
E					

P.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					

PROBLEM 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.

P.6 a) determining whether the graph is complete

What is the *time complexity* of the above operations if we use:

Adjacency Lists	Adjacency Maxtrices	Set of Vertices & Edges

P.6 a) determining whether the graph is complete

What is the *time complexity* of the above operations if we use:

Adjacency Lists	Adjacency Maxtrices	Set of Vertices & Edges
$A \rightarrow B, C$ $B \rightarrow A, C$ $C \rightarrow A, B, D$ $D \rightarrow C$ $E \rightarrow$ if number of elements in each list is available: If not:	$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$	$G = (V, E)$, where $V = \{A, B, C, D, E\}$ $E = \{\{A, B\}, \{A, C\}, \{B, C\}$

What if m= |E| is available?

P.6 a) determining whether the graph is complete

What is the *time complexity* of the above operations if we use:

Adjacency Lists	Adjacency Maxtrices	Set of Vertices & Edges
$A \rightarrow B, C$ $B \rightarrow A, C$ $C \rightarrow A, B, D$ $D \rightarrow C$ $E \rightarrow$ if number of elements in each list is available: If not:	$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$	$G = (V, E)$, where $V = \{A, B, C, D, E\}$ $E = \{\{A, B\}, \{A, C\}, \{B, C\}$

What if m= |E| is available?

P.6 b) determining whether the graph has an isolated node

What is the *time complexity* of the above operations if we use:

Adjacency Lists	Adjacency Maxtrices	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 $V = \{A, B, C, D, E\}$ $E = \{\{A, B\}, \{A, C\}, \{B, C\}$ if m=|E| available: If not:

T7[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.

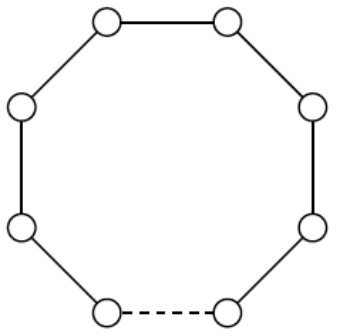
What is the *space complexity* (in terms of n) of a sparse graph if we store it using:

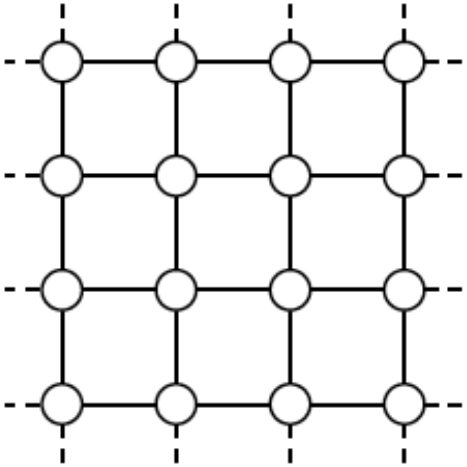
(i) adjacency lists

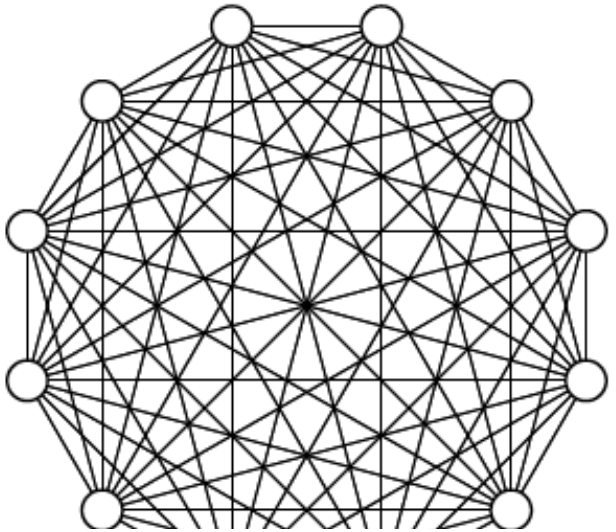
(ii) adjacency matrix

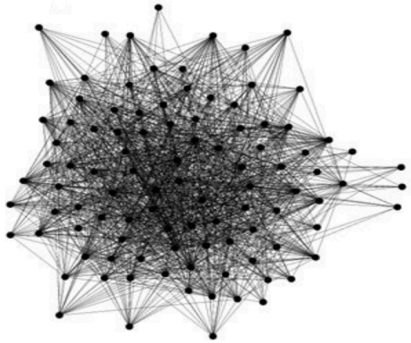
(iii) sets of vertices and edges

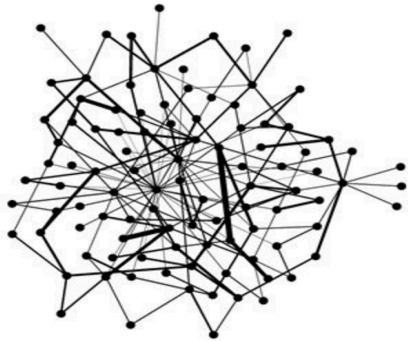
Sparse or Dense?



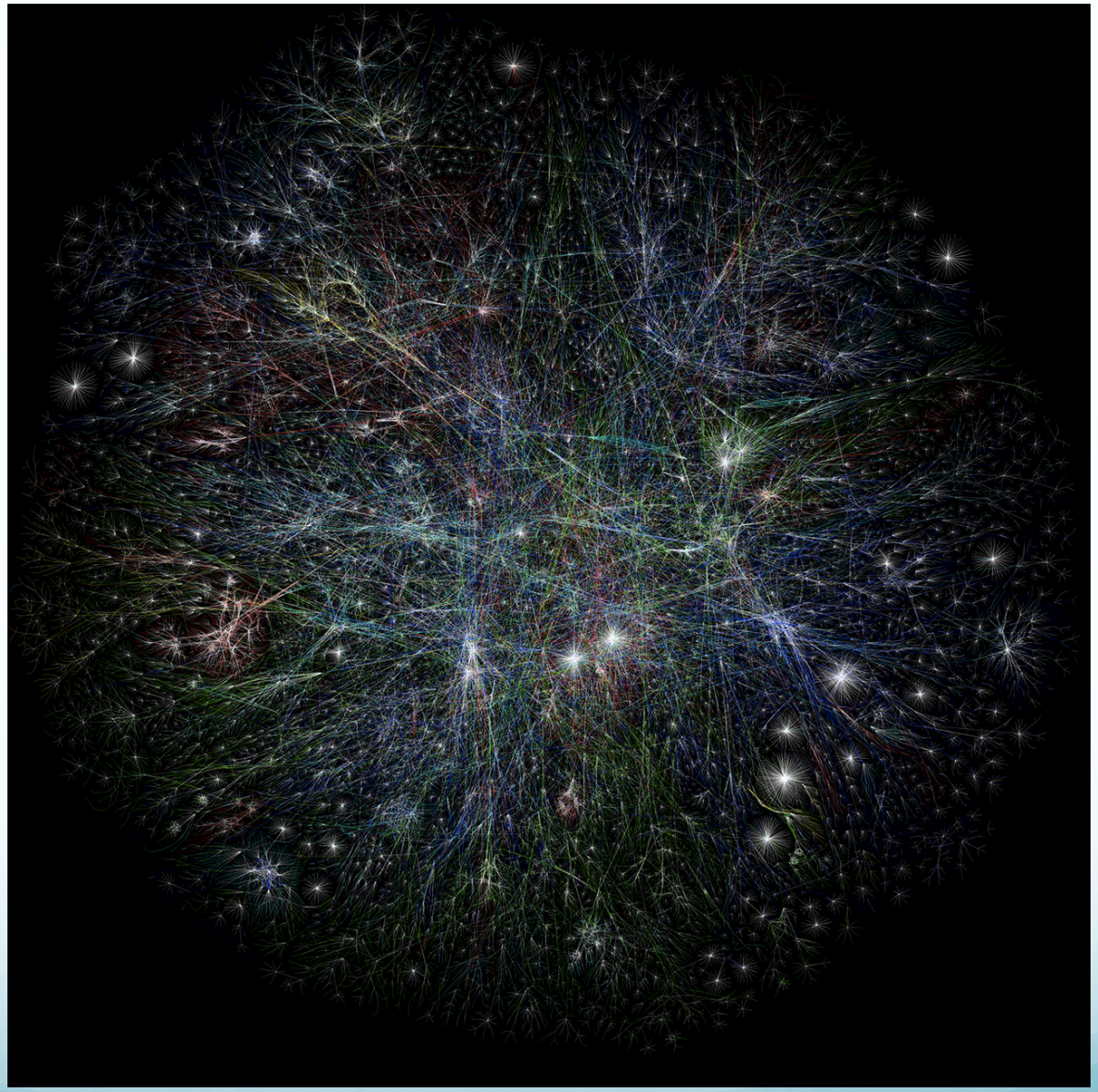








The Internet



Before LAB: notes on MST next week

Lab: also see Lab_Notes.pdf pp. 6—11

Task 1: run racecar (with Makefile) on dimefox

- On minGW (or ?) or Mac Terminal, use command cd appropriately (cd .. for going back to parent's folder) until seeing folder lab_files when running command ls
- Copy the folder lab_files to dimefox :

```
scp –r lab_files login@dimefox.eng.unimelb.edu.au:
```

- login into dimefox with command:

```
ssh login@dimefox.eng.unimelb.edu.au
```

- then, on dimefox, run your racecar with:

```
make clean
make
…
```

Task 2: in your laptop or dimefox

- 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*

Tools: 3a. connect to dimefox/nutmeg, how?

- if you are not on uni's ground, make sure to run `VPN` first
- open Terminal (such as `minGW` terminal, or Mac's `Terminal`)
- on the `Terminal` run:

```
ssh login_name@dimefox.eng.unimelb.edu.au
```

then, your terminal will work with dimefox.

Try some unix commands such as:

- `ls` (list the content of current directory)
- `cd` (change current directory),
- `cp` (copy files, run `man cp` for details)
- `mkdir` (make new directory).

At home: (use Google to) learn some basic Unix commands if needed .

Sample session on dimefox:

```
cd
ls
echo Hello Linux > hello.txt
ls
cp hello.txt hello_1.txt
more hello.txt
ls
man cp
emacs helloworld.c
```

Tools: 3b. Remote editing programs in H:

- *Aim*: Avoid time-consuming filecopy from your laptop to `H:`, be able to edit your uni's files remotely from anywhere.
- Tools: in `dimefox` terminal, using `emacs`/`nano`/`vim` etc. While vim seems inconvenient, `emacs` and `nano` could be mastered quickly.
- quick use of `emacs` (try `emacs hello.c` on `dimefox`'s terminal)
  - relatively simple and easy: just remember Ctrl-x Ctrl-s for "Save", Ctrl-x Ctrl-c for "Quit".
  - Cut/Copy/Delete a chunk of code inside the emacs window: Ctrl-Space to mark the start, arrows to expand, Esc-w to copy, Ctrl-w to cut, Ctrl-y to paste the chunk
  - Undo: Ctrl-x Ctrl-u
  - ESC a few times to exit from some mode/command if having troubles
  - how to use `emacs` more effectively and professionally: https://www.digitalocean.com/community/tutorials/how-to-use-the-emacs-editor-in-linux

FAQ

- `minGW`: I got "command not found" when trying `scp` or `ssh`. Why?
- You just missed it when installing `minGW`. Now you need:
  - run `minGW Installation Manager`
  - When the full dialog box opens with a list of software components, scroll down to find, then right-click "msys-openssh-bin" and click "Mark for Installation".
  - Click "Installation" (top left corner), then "Apply Changes", then "Apply".
  - Wait until installer prompts you with a message indicating completion.
- Go back to your `minGW` window, now you can use `scp` and `ssh`