COMP20007 Design of Algorithms

Week 4 Workshop

Tutorial

1. Solving recurrence relations

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

a. $T(n) = T(n-1) + 4$ $T(n) = T(n-1)+4 = T(n-2)+4+4 = \cdots = T(1) + \underbrace{4+4+\cdots+4}_{n-1} = 4(n-1)+1 = 4n-3$

b. $T(n) = T(n-1) + n$ $T(n) = T(n-1)+n = T(n-2)+n-1+n = \cdots = T(1)+2+\cdots+n = \sum_{k=1}^{n} k = \frac{(1+n)n}{2}$

c. $T(n) = 2T(n-1) + 1$ $T(n) = 2T(n-1)+1 = 2\times 2T(n-2)+1)+1 = 4T(n-2)+3 = 4\times 2T(n-3)+1)+3 = 8T(n-3)+4+2+1$

$= 2^{n-1}T(1) + \sum_{k=0}^{n-2} 2^k = \sum_{k=0}^{n-1} 2^k = \frac{1-2^n}{1-2} = 2^n - 1$

2. Mergesort complexity (Homework)

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)

We can have $T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + \theta(n)$

$= 2T(\frac{n}{2}) + \theta(n)$

$= 4T(\frac{n}{4}) + \theta(n)$

$= \cdots = (\log n)\cdot\theta(n) = \theta(n\log n)$

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

This kind of recurrence can be difficult to solve by expansion. We haven't seen the master theorem yet, but you can look it up and use it to solve this recurrence relation and find the runtime complexity of mergesort if you finish these questions early.

3. Subset-sum problem

Design an exhaustive-search algorithm to solve the following problem: since just enumerate all possible

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.*,

outcomes # outcomes $= 2^n$

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

and just calculte each outcome

If so, output the elements of this subset S'.

So we'll have average length to solve is $\frac{n}{2}$

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]`.

$\Rightarrow$ the time complexity is $\theta(n2^n)$

What is the time complexity of your algorithm?

4. 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.*,

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

$\sum A + \sum B = \sum S$ $\Theta(n)$

$\sum A = \sum B \Rightarrow \sum A = \sum B = \frac{1}{2} \sum S$

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]`.

all we have is calculate sum of S. $\frac{1}{2}\sum S$

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

$\Theta(n2^n)$

5. Graph representations

Consider the following graphs.

a. An undirected graph:

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

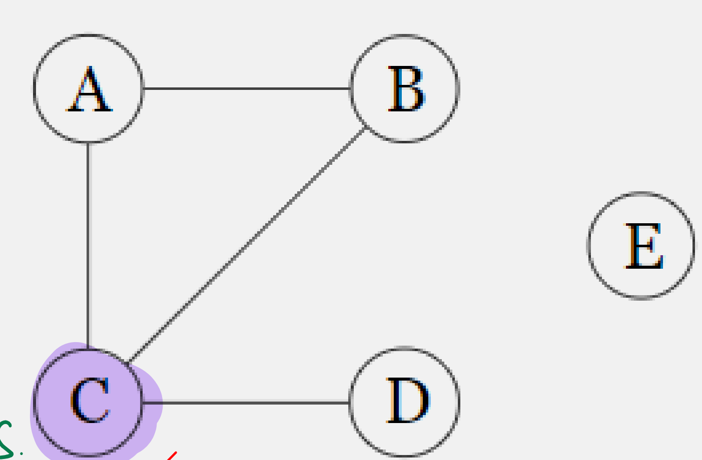
$A \to B \to C$

$B \to A \to C$

$C \to A \to B \to D$

$D \to C$

$E \to$



highest degrees

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	0	0
C	1	1	0	1	0
D	0	0	1	0	0
E	0	0	0	0	0

b. A directed graph:

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

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

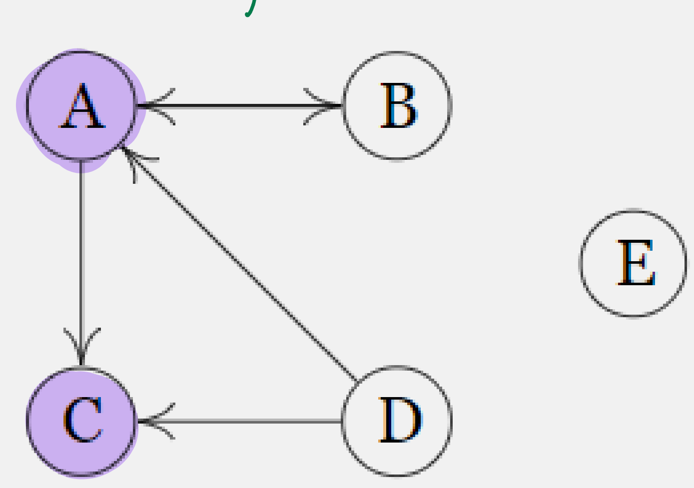
$A \to B \to C$

$B \to A$

$C \to$

$D \to A \to C$

$E \to$



Both have 2 indegree.

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	0	0
C	0	0	0	0	0
D	1	0	1	0	0
E	0	0	0	0	0

Give their representations as:

i. adjacency lists

ii. adjacency matrices

iii. sets of vertices and edges

Which node from (a) has the highest degree? Which node from (b) has the highest in-degree?

6. Graph representations continued

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* ① $\Theta(m) \to O(n^2)$ ② $O(n^2 - n) = O(n^2)$ ③ $O(1)$

b. determining whether the graph has an *isolated node* ① $O(n)$ ② $O(n^2)$ ③ $O(n+m)$

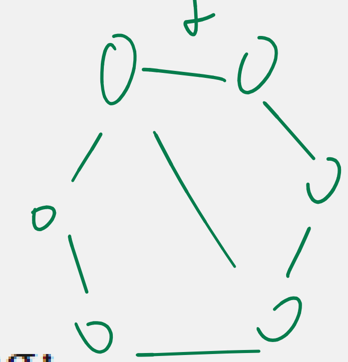
Assume that the graphs are undirected. A *complete* graph is one in which there is an edge between every pair of nodes. An *isolated node* is a node which is not adjacent to any other node.

Assume that the graph has n vertices and m edges.

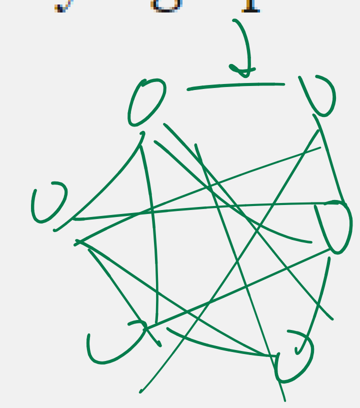
7. Sparse and dense graphs (Homework)

We consider a graph to be *sparse* if the number of edges, m, is order $\Theta(n)$. On the other hand we say a graph is *dense* if $m \in \Theta(n^2)$.

Sparse

① 

Dense

 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 $O(n)$

ii. adjacency matrix $O(n^2)$

iii. sets of vertices and edges $O(m+n) \to O(n+n) = O(n)$