5CCS2FC2: Foundations of Computing II

Tutorial Sheet 4

Solutions

4.1 The problem **VERTEX-COVER** takes as input a graph $G = (V, E)$ and integer $k > 0$ and returns `True` if there is a set $C \subseteq V$ of size k such that every edge in E connects to some vertex in C. (The set C is called a *vertex-cover*. See `https://en.wikipedia.org/wiki/Vertex_cover`)

Consider the following polynomial reduction from **SAT** to **VERTEX-COVER**:

> **Step 1)** Let F be a formula in conjunctive normal form (CNF) with three literals in each clause,
>
> **Step 2)** Construct a graph $G_F = (V, E)$, where
>
> $$V = \{L^i \;:\; L \text{ is a literal belonding to the } i\text{th clause}\}$$
>
> and
>
> $$(L_1^i, L_2^j) \in E \qquad \Longleftrightarrow \qquad i = j \quad \text{or} \quad L_1 \equiv \neg L_2$$
>
> That is to say that two literals are connected with an edge if they appear in the same clause, or if they are contradictory (*e.g.*, P and $\neg P$).
>
> **Step 3)** Return the pair $\langle G_F, k\rangle$, where k is twice the number of clauses in F, with the property that
>
> $$F \in \textsf{SAT} \qquad \Longleftrightarrow \qquad \langle G_F, k\rangle \in \textsf{VERTEX-COVER}.$$

(i) Construct the graph G_F for the following formula

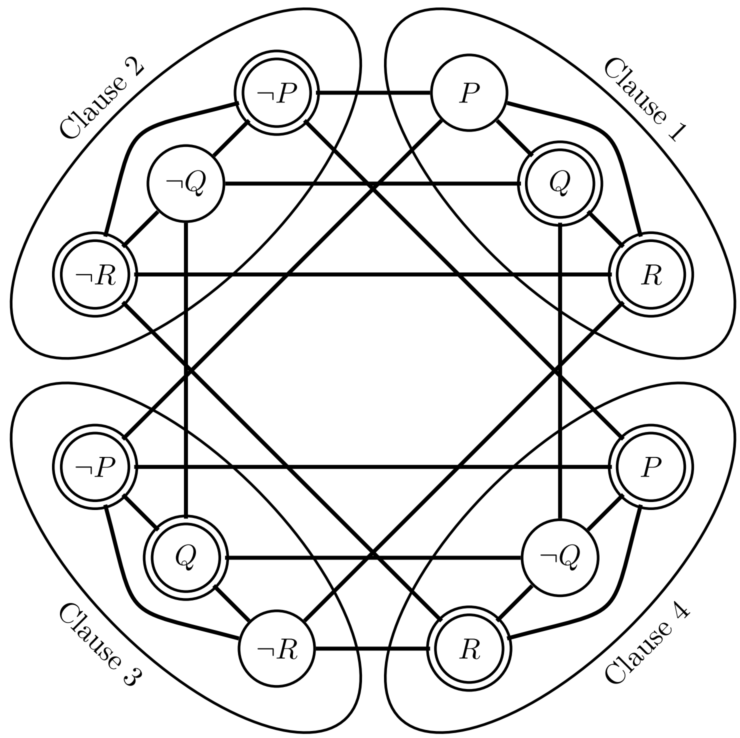
$$F \;=\; (P \vee Q \vee R) \wedge (\neg P \vee \neg Q \vee \neg R) \wedge (\neg P \vee Q \vee \neg R) \wedge (P \vee \neg Q \vee R)$$

(ii) Find a vertex cover of size $k = 8$.

(iii) What can you say about the vertices that do not belong to the vertex cover?

SOLUTION:

(i) The graph G_F is given as follows:



(ii) An example of a vertex cover for G_F of size $k = 8$ is the following:

$$C = \{(Q)^1, (R)^1, (\neg P)^2, (\neg R)^2, (\neg P)^3, (Q)^3, (P)^4, (R)^4\}$$

since every vertex that does not belong to C is adjacent to some vertex in C. (These vertices are indicated in the diagram above with double rings.)

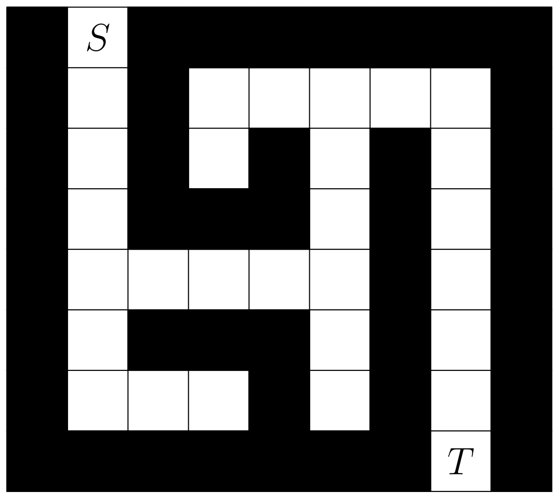
(iii) The vertices in C do not correspond to a satisfying assignment for F, since we have selected (for example) $\neg P$ from clause 2 and P from clase 4. However, those vertices *not* in the vertex cover do form a satisfying assignment for F; namely

$$\boxed{P = 1, \qquad Q = 0, \quad \text{and} \quad R = 0}$$

By requiring that the vertex-cover is of size 8, we guarentee that we select 2 vertices from every clause (since every clause contains 3 vertices that are all connected).

The unselected vertex can be assigned to be true without causing any conflicts, since a conflict between, say P and $\neg P$ would manifest itself as an edge that connected two vertices that were not covered. Hence we would not have a vertex-cover.

4.2 Consider the following maze:



(i) Convert the maze into a graph by replacing the cells with vertices and the possible paths with edges.
Hint: Try to minimise the number of vertices required to fully describe the structure of the maze.

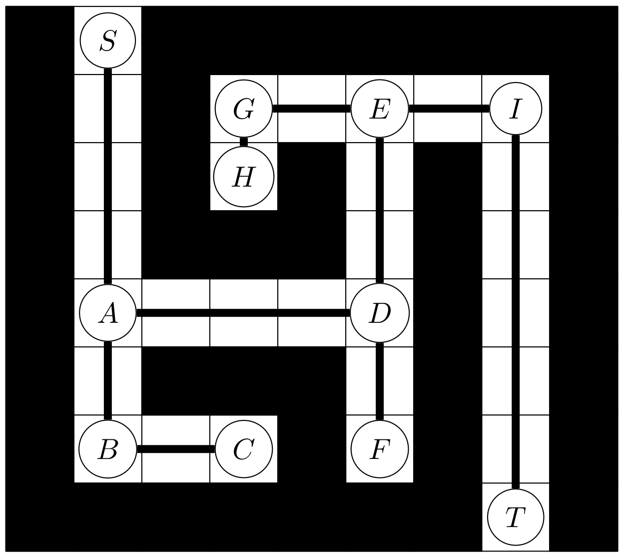
(ii) Find a route through the maze from S to T using Breadth-First-Search (BFS)

(iii) Find a route through the maze from S to T using Depth-First-Search (DFS).

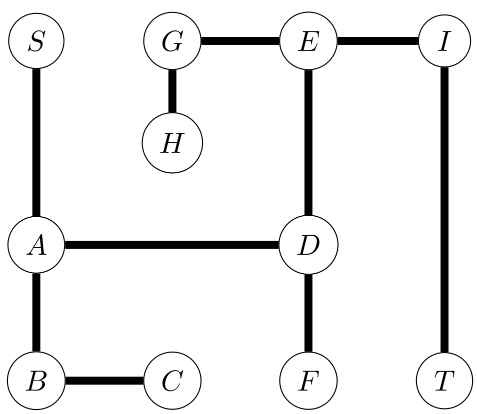
(iv) How does the order in which the vertices are added to the queue/stack affect each of the search algorithms?

SOLUTION:

(i) We need only consider those cells in which we turn a corner. (In fact we only need to consider those cells with more than 2 exposed edges, but it becomes harder to automate this for little additional payoff.)



Extracting the graph, we have the following:



(ii) For the Breadth-First-Search, we have the following stages:

Stage	Dequeue	Enqueue	Current Queue
0	$-$	S	S
1	S	A	A
2	A	B, D	B, D
3	B	C	D, C
4	D	E, F	C, E, F
5	C	$-$	E, F
6	E	G, I	F, G, I
7	F	$-$	G, I
8	G	H	I, H
9	I	T	H, T

Since the purpose of this search was to *discover* the vertex T we can now terminate without needing to dequeue the remaining vertices.

Bonus: This search only demonstrated that there is a path connecting S to T but did not record what the path is. In order to record the path we could start by enqueuing the pair $(S, [])$ where $[]$ is an empty list. At each stage we dequeue a pair $(v, \mathsf{path\_to}(v))$ and enqueue the successors $(u, [\mathsf{path\_to}(v), v])$, for each $(v, u) \in E$.

(iii) For the Depth-First-Search, we have the following stages:

Stage	Pop	Push	Current Stack
0	$-$	S	S
1	S	A	A
2	A	B, D	B, D
3	D	E, F	B, E, F
4	F	$-$	B, E
5	E	G, I	B, E, G, I
6	I	T	B, E, G, T

Again, we need not pop the remaining vertices from the stack since our goal was only to discover the vertex T.

(iii) In the case of the Depth-First-Search, we were fortunate enough that we did not have to discover all the vertices in the graph before discovering T. With the exception of F, we only popped those vertices that formed a path from S to T. However, this was *purely coincidental* due to our choice of labellings for our vertices, and our decision to push vertices to the stack in alphabetical order. In larger graphs, a different ordering may result in a much longer paths than optimal.

By contrast, Breadth-First-Search often takes longer but is guarenteed to provide an optimal path through the graph, since all paths are extended uniformly. Changes to the ordering do not have such a drastic effect on the performance of the search.

Remember to complete the Week 4 Feedback for your TAs:

`https://keats.kcl.ac.uk/mod/feedback/view.php?id=2054168`