Ollscoil na hÉireann
The National University of Ireland

Coláiste na hOllscoile, Corcaigh
University College, Cork

Summer Examination 2011

CS4407 Algorithm Analysis

Prof. G. Provan
Prof. J. Bowen (HoD)
Dr Carron Shankland (extern)

Attempt all questions

Total marks: 100

90 minutes

Please answer all questions
Points for each question are indicated by [xx]

1. **[15]** Consider the *UniqueElements* problem, where we check whether all the elements in a given array are distinct.
   a. **[10]** Use the loop invariance approach to analyse this algorithm.
   We assume that we have an array A[0 …n-1] of n elements. Starting from the first element, we check whether this element occurs in the remainder of the array.

```
UniqueElements (A)
        n ← length(A)
        for i ← 0 to n-2 do
            for j ← i+1 to n-1 do
                If  A[i]=A[j] return false
        Return true
```

   Pre-condition: show the loop invariant holds before the first iteration, when i=0. Here, the subarray consists of just A[1], which is (trivially) unique since it has not been compared to anything else.

   Exit step: we exit when we have compared the next-to-last and last elements. Hence the entire array is now checked for uniqueness.

   Post-condition: when we exit, the entire array is now checked for uniqueness.

   Induction step: the body of the outer for loop works by comparing A[i] to A[i+1], A[i+2], A[i+3] and so on by one position to the right until the uniqueness of A[i] is established. This is true for all i from 0 to n-2.

   b. **[5]** Use this approach to specify the complexity of the algorithm.

   In the worst case, the number of array-element comparisons is

$$T_{worst}(n) = \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=1}^{n-1} i = (n-1)n/2 \in O(n^2)$$

2. **[15]** Solve the following recurrence relation using repeated substitution. Do an inductive proof to show your formula is correct.

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

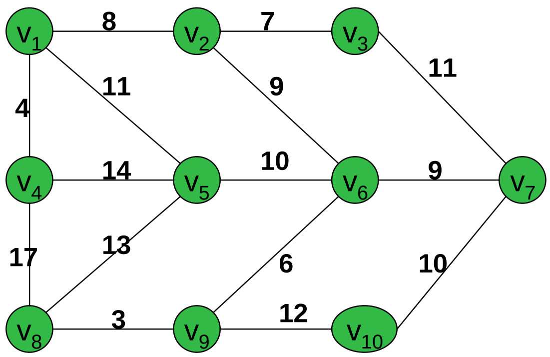
By repeated substitution, we can solve T(n) as:

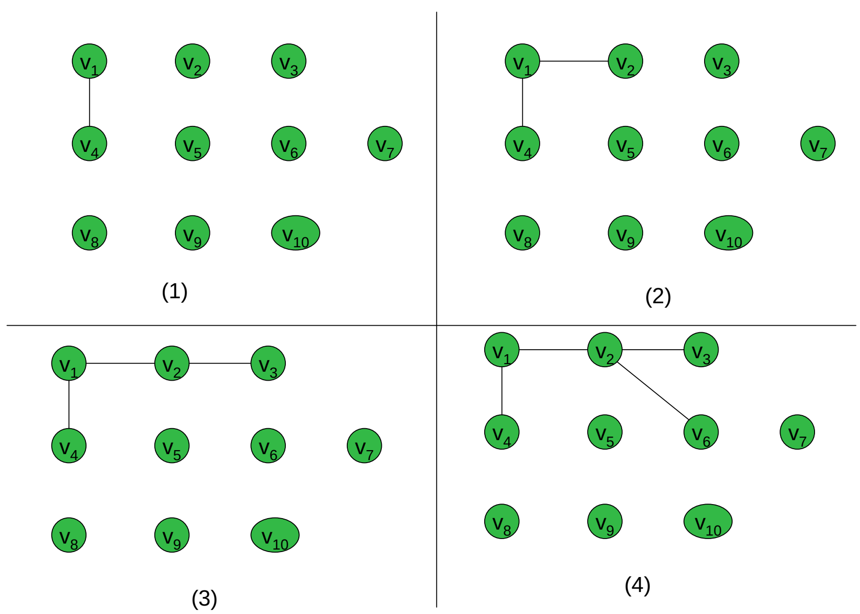
$$\begin{aligned} T(n) &= T(n-1) + O(n) \\ &= (T(n-2) + O(n-1)) + O(n) \\ &= T(n-2) + O(n-1) + O(n) \\ &= T(n-3) + O(n-2) + O(n-1) + O(n) \end{aligned}$$

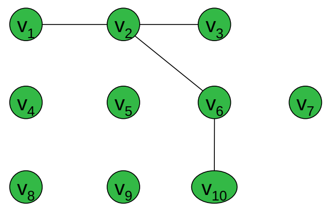
...
$= T(1) + O(2) + ... + O(n-1) + O(n)$
$= O(1 + 2 + ... + n-1 + n)$
$= O(n^2)$

3. **[20]** Given the graph G shown below,

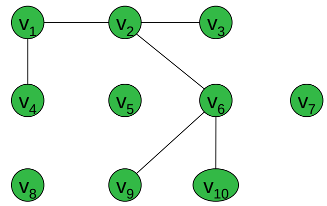
   a. **[15]** Find a minimum spanning tree (MST) for G; show the steps of generating the MST.
   b. **[5]** What is the complexity of this algorithm?



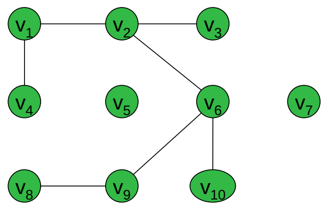




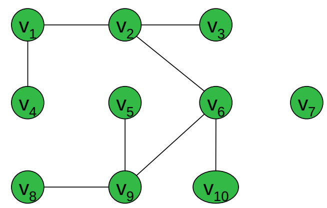
(5)



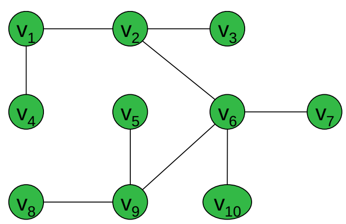
(6)



(7)



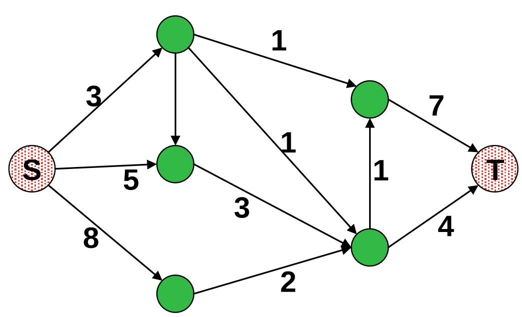
(8)

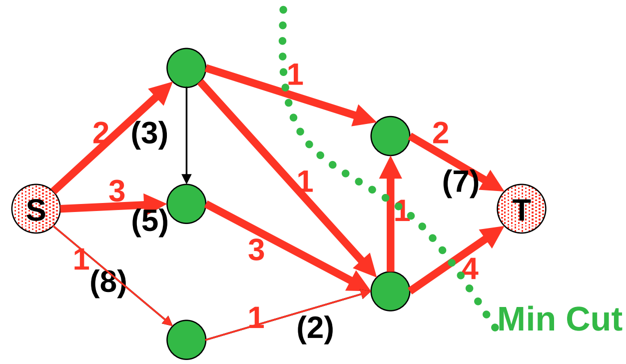


(9)

4. **[15]** Consider a graph G(V,E), with source node S and sink node T.

   a. **[8]** For the instance of a flow network shown below, compute the maximum flow. Give the actual flow as well as its value. Justify your answer.





c. **[4]**We show the solution here, with the flows on each edge defined. Our total flow is 6. We know that 6 is the Max flow, as we can identify a min-cut of 6, and by the max-flow/min-cut theorem, we know that max-flow=min-cut.

d. **[3]**Consider a decision problem defined for such a flow network: Flow:= {(G,S,T,k)|G(V,E) is a flow network, S,T∈V, and the value of a optimal flow from S to T in G is k}.

Is *Flow* in NP? Is *Flow* in P? Justify your answer.

Flow is in NP, since we can explicitly compute an optimal max-flow using a polynomial time algorithm, like Edmonds-Karp. Since P⊆NP, Flow must be in NP. Flow is also in P.

5. **[20]** Prove that SET PACKING (SP) is NP-complete.
INSTANCE: A collection *C* of finite sets over a universal set *U*, and integer *k*.
QUESTION: Does *C* contain *k* disjoint sets?

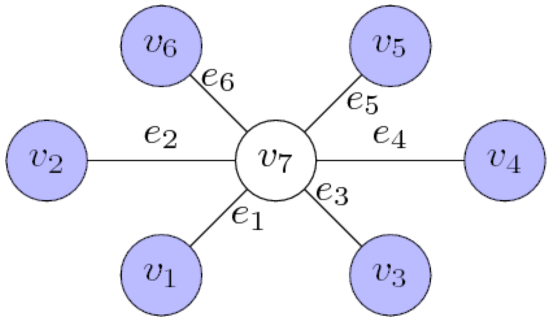
(Assume that you need to define a reduction from one of the following NP-complete problems: HAMILTON CIRCUIT, CLIQUE, INDEPENDENT SET, 3-SAT)

(a) Prove that SET PACKING (SP) is in NP. A certificate consists of a subset $S=\{S_1,\ldots,S_k\}$ of C. We must check 2 things. We must check if $S_i \cap S_j = \varnothing$, $i \neq j$, and $S_i, S_j \in S$.We can clearly do this check in $O(n^2)$ time.

(b) Define a reduction from the INDEPENDENT SET (IS) problem, which is NP-complete.

INSTANCE: Undirected finite graph *G(V,E)* and integer *k*.
QUESTION: Does G contain a set of *k* independent vertices?

Given an instance *G(V,E)* of IS, we generate an instance of SP as follows. First, we generate an element U_i of U corresponding to every edge E_i; and second, we create the set S_i consisting of the edges incident to vertex V_i in IS. Finally, we set the size of Set Packing to be k as well. Clearly, this reduction can be performed in time polynomial in the size of V and E.



Example: Graph of Independent Set

Example: From the construction, a Set Packing of the example above would be $S_1 = \{e_1\}, S_2 = \{e_2\}, S_3 = \{e_3\}, S_4 = \{e_4\}, S_5 = \{e_5\}, S_6 = \{e_6\}, S_7 = \{e_1, e_2, e_3, e_4, e_5, e_6\}$. So, the $S = \{1,2,3,4,5,6\}$.

We need to show that Independent Set $\leq_p$ Set Packing (This construction of Independent Set leads to is a special case of Set Packing).
IS is a Independent Set of size k in G if and only if S is a Set Packing of at most k S_i's, such that $S_i \cap S_j = \emptyset$, $i \neq j$, and $i, j \in S$.

$\Rightarrow$IS has an Independent Set of size k in G if S is a Set Packing of at most k S_i's, such that $S_i \cap S_j = \emptyset$.

Suppose G has an independent set of size at least k, call it IS. Construct S by including exactly the S_i with $v_i \in$IS. Obviously, the size of S is equal to the size of IS, k. Furthermore, since no two vertices in S can share an edge, the sets S_i we picked must be pairwise disjoint, so we have found a valid set packing of at least k sets.

$\Leftarrow$ S is a Set Packing of at most k S_i's, such that $S_i \cap S_j = \emptyset$ if IS has an Independent Set of size k in G

Assume that we are given the new instance of set packing S of size at least k. Then, we define a vertex set IS as consisting exactly of those v_i for which $i \in S$. The size of IS is the same as that of S, k. For any edge e, at most one set S_i with $i \in S$ may contains e, so at most one node v_i can be incident on e. Thus, no two selected nodes are connected by an edge, and IS is in fact an independent set of size at least k.

6. **[20]** Consider a class of graphs G(V,E) which contain an independent set of size ¾|V|. An independent set is a subset V' of vertices such that no two vertices in V' are connected by an edge of G.

   i. **[10]** Provide an approximation algorithm for G that can provably compute an independent set of size at least ½|V|.
   ii. **[10]** Prove that your algorithm can meet such bounds.

(Hint: you may make use of the 2-approximation algorithm for vertex cover that was described in class, i.e., you may assume that this algorithm exists and can be called as a subroutine. A vertex cover of a graph G is a subset of vertices V' such that all edges in G are adjacent to at least one node of V'.)

Algorithm:
//Input: G(V,E)//
Compute complement G'of G
VC←*2-Approx*(G') //return approximate vertex cover of G'//
IndepSet←VC' //compute complement of approx. vertex cover VC//
Return IndepSet

Proof: G(V,E) contains an independent set of size ¾|V|. The complement G'of G thus has a vertex cover of size ¼|V|, by definition of independent set and vertex cover.
We can use the 2-approximation algorithm for vertex cover to find a vertex cover of size at most ½|V| in G'. The complement of this vertex cover is an independent set of size at least ½|V|.