Exam 26 October 2012, 8:00–13:00, Sparta:A–B

EDAN55 Advanced Algorithms

The exam consists of 4 large questions; each consisting of a number of smaller subquestions.

1. The exam is "open book," so you can bring whatever material you want, including textbooks, a dictionary, and your own course notes.
2. You can bring an electronic calculator.
3. We try to minimise the dependencies among subquestions. In particular, you can solve them in any order. Also, you are free to *use* the result of subquestion x to answer subquestion y, even if you didn't answer x.
4. Scoring: Answering "I don't know" (and nothing else) scores $\frac{1}{4}$ of a subquestion's points. An empty or wrong answer scores 0 points.
5. You can answer in Swedish or English.

Some tips:

1. Shorter is better.
2. An example is better than a failed attempt at explaining something in general.
3. Drawings, pseudocode, and formulas are good. "Wall of text" is bad.
4. Admit ignorance.
5. Be tidy.

Good luck!

(blank)

Question 1, Approximation

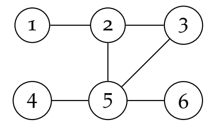

Figure 1: An instance to Max Tripartition.

The *Max Tripartition* problem is defined as follows. Given an undirected graph $G = (V, E)$, find a partition of V into three nonempty sets such that the number of edges (called the size of the cut) between them is maximised. Formally, a tripartition conists of three nonempty sets $A, B, C \subseteq V$ called *parts* such that $A \cap B = A \cap C = B \cap C = \emptyset$ and $A \cup B \cup C = V$. Define the the cutsize c as

$$c(A, B, C) = \left|\{\, uv \in E \colon u \in A \wedge v \in B \vee u \in A \wedge v \in C \vee u \in B \wedge v \in C \,\}\right| .$$

We want to find a tripartition of maximum cutsize.

►*1a* (1 pt.) Find a maximum tripartition of the graph in fig. 1.[1]

[1] Your answer is a drawing showing the tripartition and an integer (the corresponding cutsize).

Consider the following simple algorithm for Max Tripartition. Let $V = \{v_1, \ldots, v_n\}$ and for vertex subset $S \subseteq V$ let $d(u, S) = \left|\{\, v \in S \colon uv \in E \,\}\right|$ denote the number of edges between u and S.

Initially, set $A = \{v_1\}$, $B = \{v_2\}$, and $C = \{v_3\}$.
For every i with $4 \le i \le n$,
 if $d(v_i, A) \le d(v_i, B)$ and $d(v_i, A) \le d(v_i, C)$ then add v_i to A,
 else if $d(v_i, B) \le d(v_i, A)$ and $d(v_i, B) \le d(v_i, C)$ then add v_i to B,
 else add v_i to C.

►*1b* (1 pt.) Run the algorithm on the graph in fig. 1. [2]

[2] Your answer is the resulting sets A and B and C and the cutsize.

►*1c* (1 pt.) Give an example where the algorithm finds a cut of size only $\frac{2}{3}$OPT. [3]

[3] Your answer is a concrete graph, an optimum solution to that instance and the solution found by the algorithm.

►*1d* (3 pts.) Show that the algorithm is guaranteed to find a cut of size at least $\frac{2}{3}$OPT of the optimum. [4]

[4] Your answer is a short proof. It includes a lower bound on the solution found by the algorithm and an upper bound on the optimum solution.

►*1e* (2 pts.) Prove that unless P equals NP, there cannot be an algorithm for Max Tripartion whose solution is at most a factor $(1 + \epsilon)$ below OPT and that runs in polynomial time for any choice of $\epsilon > 0$. You can freely use that Max Tripartition is NP-hard.[5]

[5] You answer is a short proof.

►*1f* (1 pt.) Modify the algorithm and analysis so that it works for the *Max k-Cut* problem, where the input consists of a graph and an integer k, and we want to find a partition into k parts, maximising the edges between them. (Max Tripartition is Max k-Cut for $k = 3$.)[6]

[6] You answer contains an algorithm and a brief analysis of its approximation guarantee; the most important part is to clearly state the approximation factor.

►*1g* (1 pt.) Modify the algorithm and analysis for the *Max Dicut* problem, where the input consists of a *directed* graph, and we want to find a partition into 2 subsets A and B so as to maximise the total number of directed arcs *from A to B*. [7]

[7] You answer contains an algorithm and a brief analysis of its approximation guarantee; the most important part is to clearly state the approximation factor

Question 2, Parameterized Analysis

Consider n points in the plane, such as in figure 2. We consider the problem of covering them with k lines. Formally,

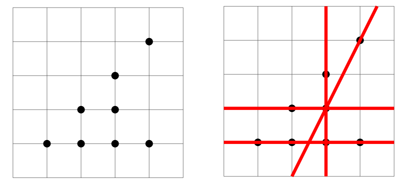

Figure 2: Left: A set of 8 points in the plane. (In general, the points need not be at integer coordinates.) Right: 4 lines covering the points.

Name: Covering Points With Lines

Input: A set S of n points in the Euclidean plane $(x_1, y_1), \dots, (x_n, y_n)$. Integer k.

Output: A set of k lines covering the n points, or "impossible" if no such set exists.

Figure 2 also shows a line cover with $k = 4$ lines.

Throughout this exercise you can assume that it takes constant time to perform basic geometric operations such as constructing a line trough two points, or checking if a given point lies on a given line, or if 3 points are colinear. ("Colinear" means "you can draw a single line through them".)

▶*2a* (1 pt.) Find a line cover of size $k = 3$ for the set of points in fig. 2.[8]

▶*2b* (2 pts.) Write a simple exhaustive search (or "brute force") algorithm and give its running time in terms of n and k. Note that this is not completely trivial—there are infintely many lines through n points, so you can't just say "Check all lines".[9]

We will write a better algorithm. The central observation is that if S contains a set of $k+1$ colinear points or more, then the optimal solution is guaranteed to include the line through all of them.

▶*2c* (1 pt.) Why is this true?

▶*2d* (1 pt.) Give a counterexample that shows that this is not true for k colinear points.

▶*2e* (4 pts.) (Harder.) Write an algorithm based on the above observation, briefly argue for its correctness, and state its running time. (The running time must have the form $f(k) \cdot n^{O(1)}$ for some function f.)

[8] Your answer is a drawing showing the lines.

[9] Your answer is some lines of pseudocode and a running time estimate using asymptotic notation.

Question 3, Exponential Time Algorithms

We consider the Set Cover problem.

Name: Minimum Set Cover (MSC)

Input: A set U of n elements and a collection C of n subsets $S_1, \ldots, S_n \subseteq U$.

Output: A smallest subcollection that covers U, i.e., an index set $J \subseteq \{1, \ldots, n\}$ such that

$$\bigcup_{j \in J} S_j = U\,,$$

with $|J|$ minimal.

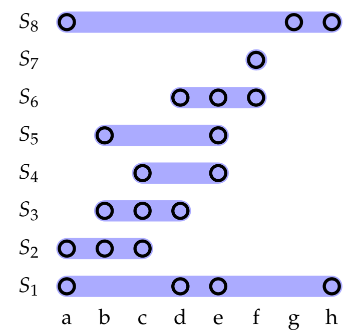

Figure 3: One way of visualising the set cover instance where $U = \{a, b, c, d, e, f, g, h\}$ and $S_1 = \{a, d, e, h\}$, $S_2 = \{a, b, c\}$, $S_3 = \{b, c, d\}$, $S_4 = \{c, e\}$, $S_5 = \{b, e\}$, $S_6 = \{d, e, f\}$, $S_7 = \{f\}$, and $S_8 = \{a, g, h\}$.

For instance, the graph in fig. 3 contains a (nonoptimal) set cover of size 5 given by $J = \{1, 4, 5, 7, 8\}$. (In other words, the union of the sets S_1, S_4, S_5, S_7, and S_8 contains U.)

►*3a* (1 pt.) Find a minimum set cover for the instance in fig. 3.

►*3b* (2 pt.) Explain very briefly how MSC can be solved using exhaustive search ("brute force") and state the resulting running time, ignoring polynomial factors.

►*3c* (1 pt.) Explain very briefly why MSC can be solved in polynomial time if every subset $S_i \in C$ in the collection contains at most 2 elements.

►*3d* (4 pts.) Construct a branching ("decrease-and-conquer") algorithm for MSC. You running time must be better than 2^n. Be precise about which branching rules you use; for example by writing the algorithm in some form of pseudocode. Briefly argue why each rule is valid. Give a recurrence relation for the running time of the resulting algorithm; read the solution to your recurrence off Table 1. *Hint:* Use two parameters, $n = |U|$ and $m = |C|$, the number of subsets in the collection.

	b				
a	1	2	3	4	5
1	2	1.62	1.47	1.39	1.33
2		1.42	1.33	1.28	1.24
3			1.26	1.23	1.20
4				1.19	1.17
5					1.15

Table 1: Running times for decrease-and-conquer recurrences of the form $f(N) \leq f(N-a) + f(N-b)$ for small $a \leq b$. For example, the Fibonacci reccurence $F(N) = F(N-1) + F(N-2)$ satisfies $F(N) = O(1.62^N)$.

Question 4, Randomized Algorithms

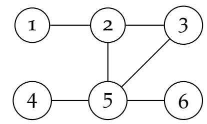

Figure 4: An instance to Vertex Cover.

We consider the well-known Vertex Cover problem.

Name: Minimum Vertex Cover

Input: A simple, undirected graph $G = (V, E)$ with $|V| = n$, $|E| = m$.

Output: A minimum vertex cover, i.e., a vertex subset $S \subseteq V$ such that for every edge $uv \in E$, we have $u \in S$ or $v \in S$ (or both).

Consider the following randomized algorithm for this problem:

1. For every vertex v, pick a priority $p(v)$ uniformly at random from the interval $[0, 1]$.
2. Let S be the set of vertices that have at least one neighbour of higher priority. Formally,

$$S = V - \{\, v \colon p(v) \geq \max_{u \in N(v)} p(u) \,\},$$

where $N(v)$ denotes the neighbours of v.

▶*4a* (1 pt.) Run the algorithm on the graph in fig. 4. Use the random values $0.345, 0.432, 0.165, 0.814, 0.74, 0.524$ for $p(v_1), p(v_2), \ldots$. [10]

▶*4b* (1 pt.) Assume vertex v has degree $d(v)$. Find $\Pr(v \in S)$. [11]

▶*4c* (1 pt.) Consider running the algorithm on the 2-vertex graph u — v. Are the events $u \in S$ and $v \in S$ independent? Why or why not? [12]

▶*4d* (1 pt.) Find the probability that S is a vertex cover.[13]

▶*4e* (2 pt.) Assume that G is d-regular (i.e., every vertex has degree exactly d). Find the expected size of the vertex cover computed by the algorithm. [14]

▶*4f* (1 pt.) Assume G is the n-cycle. (That is, $E = \{\, \{i, i+1\} \colon 1 \leq i < n \,\} \cup \{n, 1\}$.) Find the expected approximation ratio of the algorithm. [15]

▶*4g* (1 pt.) Assume G is the n-star. (That is, $E = \{\, \{1, i\} \colon 2 \leq i \leq n \,\}$.) Find the probability that the algorithm computes a minimum vertex cover. [16]

▶*4h* (1 pt.) Assume G is the n-star. Calculate the number of times that I need to repeat the algorithm before I have constant nonzero probability of finding a minimum vertex cover. [17]

[10] Your answer is a drawing showing S.

[11] Your answer is an expression and an argument for it.

[12] Your answer is the word "yes" or the word "no", followed by an argument.

[13] This is not meant to be a trick question. But if you think your answer is weird, it's probably correct.

[14] Your answer is an expression and an argument for it.

[15] Your answer is an expression and an argument for it.

[16] Your answer is an expression and an argument for it.

[17] Your answer is an expression and an argument for it.