Applied Algorithm Design: Exam Answers

Prof. Pietro Michiardi

Questions

1. When does a bipartite graph have a perfect matching?

   Answer: If $G = N, E$ and $N = X \cup Y$, where X and Y are disjoint sets of nodes connected by an edge with one end in X and the other end in Y, then you can have a perfect matching if $|X| = |Y|$

2. Explain the salient differences between the Cournot and the Bertrand models of competition.

   Answer: In the former, the strategy set is the production quantity, in the latter the strategy set is the items' price.

3. Give an example of a recursive algorithm.

   Answer: the typical example is the algorithm to compute the factorial of n. Also algorithms for DFS, Fibonacci series, ... are good answers.

4. What is the difference between Prisoners' Dilemma games and Coordination Games?

   Answer: First, specify that we look at 2-player games. In the former, there is a strictly dominated strategy, hence only one equilibirum. In the latter, there are two equilibria that are not symmetric (in the general form). Hence a third-party can guide players toward an equilibrium or the other.

5. Give an example of a vertex centrality measure.

   Answer: examples are PageRank, node degree, ...

6. Discuss the data structures you used in the implementation of the Gale-Shapley algorithm from HomeWork 1.

   Answer: the answer is on the slides from Lecture 2.

7. You have to implement a Web Crawler, and explore the Web following the underlying graph representing Web Pages as vertexes, and hyperlinks between them as edges. Discuss the implications of using DFS or BFS to explore the graph. Which technique is more suitable to you?

   Answer: there is no absolute answer here, but the common practice is to use BFS (avoids over-speicalization).

8. What is the k-means algorithm? What's the typical use of this algorithm?

   Answer: the answer is on the slides from Lecture 2.

9. Is it possible to cheat in the "deferred acceptance" algorithm used to find stable matches?

   Answer: the answer has been discussed in class, as a correction of Homework 1.

10. What is the Maximum Flow of a tree? Draw a tree example and explain.

    Answer: It is dictated by the bottleneck links from the root of the tree to the leaves.

1 Exercise: The Facility Location Problem.

Question 1: Take the k-median problem. What is the worst-case complexity of an algorithm to solve this problem? Think about a brute-force approach.

Answer: The worst case algorithm would simply try all possibilities. This means you have $\binom{n}{k}$ possibilities. This hints that the

complexity class of the algorithm is NP-complete[1]

Question 2: Sketch (even informally) an algorithm to solve the k-median problem.

Answer: you can use a *local search* approach. Initially, just throw in k facilities randomly placed. Then implement a basic operation that is called **swap**: at each iteration of the algorithm, select a facility (which means you select a node of the graph currently hosting a facility) and swap it with another node of the graph that is not hosting a facility. Evaulate the cost function. If the consequence of the swap operation is a reduced cost, then hold the swap and continue, otherwise discard the swap.

Question 3: Sketch (even informally) an algorithm to solve the uncapacitated facility location problem.

Answer: similarly to the local search algorithm for the k-median problem, define a set of basic operations. i) **swap**, as before; ii) **add**, which randomly selects a node that is not a facility and add it to the set of facilities currently deployed; iii) **drop**, which randomly selects a facility to "close". At every iteration of the algorithm, check the costs that correspond to each individual operation and take the one that give lower costs.

2 Exercise: The Cake Cutting Problem.

Question 1: First, attack the problem in a special case. Sketch (in pseudo-code) an algorithm that solves the above cake-cutting problem when $n = 2$.

Answer:

1. Alice cuts the cake into two equal pieces (equal by her measure)
2. Bob chooses whichever piece looks larger (by his measure)
3. Alice takes the remaining one

Question 2: Prove that your algorithm is fair according to the definition given above.

Answer:

[1]This can be formalized by reducing the problem to the vertex cover problem.

Theorem: Cut-and-choose is fair.

Proof: If Alice follows the protocol, she gets exactly 1/2 (by her measure), no matter how Bob behaves. Next Let's consider Bob. When it is time for him to choose, he sees two pieces, one worth W and the other worth $1 - W$ (by his measure). It is guaranteed that either $W \geq 1/2$ or $1 - W \geq 1/2$ (if $W < 1/2$ and $1 - W < 1/2$, then $1 = W + 1 - W < 1/2 + 1/2 = 1$, which is impossible). Therefore, if Bob follows the protocol, he gets at least 1/2 by his measure. **QED.**

Question 3: The general case.

Is there a general algorithm for fair cake-cutting for $n > 2$? Sketch an algorithm for such a general case.
Answer: If $n = 2$, use cut-and-choose. Otherwise: Let first $n-1$ people divide the cake using a recursive call to this procedure. Then the n-th person steps in and asks each of the first $n-1$ people to divide her share into n equal pieces (by her measure). Finally, the n-th person goes around and collects the largest (in his view) of the n pieces from each of the other people.

Is your algorithm for the general case fair? Prove it.

Answer: For first $n-1$ people, yes, since they get $\geq 1/(n-1)$ after recursive call, divide this into pieces each of size $\geq 1/n(n-1)$, and then get $n-1$ of these pieces. For last person, yes, since he gets at least x_i/n from the i-th person, if ith person's whole share is worth x_i. Also, $x_1 + ... + x_{n-1} = 1$, so last person's total share is $x_1/n + ... + x_{n-1}/n \geq 1/n$. Therefore:

Theorem: This protocol is fair.

Proof: By induction on n. We've proved the base case (n=2). The inductive step is exactly what appears in the previous paragraph. **QED**.

What is the running time of your algorithm? Can we achieve a polynomial running time for such a problem?
Answer: in our case it's $n! > 2^n$. Finding a polytime algorithm is very hard... and subject of research.