CSE 2011/3101 Paradigms

Jeff Edmonds

1. Algorithmic Paradigms: Give a 4-5 sentence definition of each of the following algorithmic paradigms.

   (a) An *Iterative* Algorithm.

   - Answer: An *iterative* algorithm takes one step at a time. During each it makes a little progress while maintaining a loop invariant. A loop invariant is an assertion/picture about what the data structure looks like at that moment in the computation. The key steps are:

     Establishing the Loop Invariant:
     $\langle pre{-}cond \rangle$ & $code_{pre\text{-}loop}$ $\Rightarrow$ $\langle loop{-}invariant \rangle$

     Maintaining the Loop Invariant:
     $\langle loop{-}invariant_t \rangle$ & *not* $\langle exit{-}cond \rangle$ & $code_{loop} \Rightarrow \langle loop{-}invariant_{t+1} \rangle$

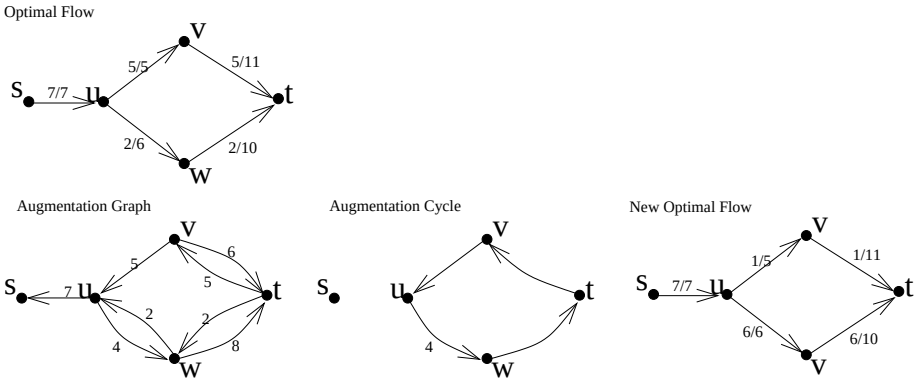
     Ending Obtaining the Postcondition:
     $\langle loop{-}invariant \rangle$ & $\langle exit{-}cond \rangle$ & $code_{post\text{-}loop} \Rightarrow \langle post{-}cond \rangle$

   (b) A *Recursive* Algorithm.

   - Answer: A *recursive* algorithm asks his friends any instance that is smaller and meets the precondition and trusts that the friends gives the correct answer. If the algorithm's instance is sufficiently small, then it must do it on its own. Best not to trace the computation out, i.e. talking about your friend's friend's, i.e. micro managing your friends.

   (c) *Max Flow - Min Cut, Primal Dual* Algorithm.

   - Answer: Max Flow is looking for the maximum amount of water (any commodity) flowing through the edges of a network subject to constraints that the flow along any edge cannot exceed that of the edge's capacity and the total flow into a node must equal the total flow out.
     The algorithm is hill climbing which keeps getting a better solution until a cut is found that proves there can be no better flow.



     Cut separates $\{s\}$ from the rest.

   (d) *Linear Programming*? For example, express Network Flows as a Linear Program.

   - Answer: Maximize some linear equation in your unknowns subject to a set of linear constraints.
     Maximize $\sum_u F_{s,u} - \sum_v F_{v,s}$
     Subject to
     $\forall \langle u, v \rangle$ $0 \leq F_{u,v} \leq C_{u,v}$
     $\forall u \neq s$ or t, $\sum_v F_{u,v} = \sum_v F_{v,u}$

   (e) A *Greedy* Algorithm.

- Answer: A *greedy* algorithm grabs the next best item and commits to a decision about it without concern for any long term consequences. This decision is irrevocable.
  Loop Invariant: We have not gone wrong. There is at least one valid optimal solution S_t that extends the choices A_t made so far by the algorithm.

(f) A *Recursive Backtracking* Algorithm.

- Answer: On instance I, a *recursive backtracking* algorithm tries various assumptions k about the optimal solution (i.e. tries all answers to the question posed to the little bird.) This assumption k narrows down the set of possible optimal solutions so that the reduced search can be posed as a smaller instance to the same problem. The algorithm recurses having a friend solve this smaller instance. The algorithm combines the advice k from the bird and the friend to produce $Sol_{\langle I,k\rangle}$ which is an optimal solution to his instance I conditioned on it being consist with the bird's advice k. Then it backtracks and tries a different k. In the end, it returns the best of these, namely $best_k(Sol_{\langle I,k\rangle})$.
  For example, if you want to find the shortest distance $Dist(u, v, \ell)$ from u to v with at most ℓ edges then what you try (answer from bird) is the middle node k in such a path. One friend tells you $Dist(u, k, \ell/2)$ which is the shortest distance from u to k with at most $\ell/2$ edges and another friend tells you $Dist(k, v, \ell/2)$ which is that from k on to v. This gives the recurrence relation: $Dist(u, v, \ell) = Min_k\,[Dist(u, k, \ell/2) + Dist(k, v, \ell/2)]$.

(g) A *Dynamic Programming* Algorithm.

- Answer: A *dynamic programming* algorithm fills in a table with a cell for each of a set of subinstances. Each is solved in the same way as one stack frame of the recursive backtracking algorithm. This algorithm is iterative not recursive because by the loop invariant, the needed answers from the friends are already stored in the table.
  For example, the table to store $Dist(u, v, \ell)$ has $n^2 \log n$ entries for $u, v \in [n]$ and $\ell \in \{1, 2, 4, 8, \ldots, n\}$. Each entry requires trying n possible nodes k. The total time is $\Theta(n^3 \log n)$.

(h) *Polynomial Time.*

- Answer: A computational problem is in polynomial time if it can be computed in time $n^{\Theta(1)}$ as a function of the number of bits n to represent the input I.
  Hint: If you don't know the running time, just think of poly time as "feasible". If you think it is a problem that wolfram alpha can solve in a second or two then you can pretty much count on it being a poly time algorithm.

(i) *Nondeterministic Polynomial Time.*

- Answer: Nondeterministic Polynomial Time: The goal of the problem is to find a valid solution S for the given instance I. For example, I could be a circuit and S could be a satisfying assignment. In the yes/no version you only must know whether or not such an S exists. S becomes a witnesses/proof that I is a yes instance. Note that there is no such S to prove that I is a no instance. It takes exponential time to brute force search through all of the 2^n possible solutions S, where n is the number of bits in S. But if one "Nondeterministically" guesses a valid solution, its validity can be checked in polynomial in $|I|$ time (actually in this case, linearly in the number of gates.

(j) *Computable.*

- Answer: A computational problem is said to be computable if there is an algorithm (Turing Machine) that on every input halts with the correct answer. There is no bound on the running time.

(k) *Uncomputable.*

- Answer: If the problem is not computable then it is uncomputable. For every algorithm there is an input on which the algorithm either gives the wrong answer or runs forever.

(l) *Deterministic Worst Case Running Time.* Give the first order logic game and expression for this.

- Answer: The Algorithm/Adversary game for deterministic worst case running time is as follows. Given a problem P, the algorithm designer provides an algorithm A. The adversary seeing this algorithm provides a worst case input instance I. The outcome of the game is the running time of A on I (assuming it gives the correct answer).

$$\exists A,\ \forall I,\ P(I) = A(I) \text{ and } Time(A, I) \leq T(|I|)$$

(m) A *Probabilistic Algorithm.* Give the first order logic expression for this and how are their games different?

- Answer: For a probabilistic algorithm tells the adversary the random algorithm, but not the result of the coin flips $r = \langle 0101011... \rangle$. This amounts to a distribution on deterministic algorithms. The Adversary chooses an input I. The coins are flipped. We take the expected running time assuming the probability of getting the right answer is good.

$$\exists A(I, r),\ \forall I,\ Exp_r/Pr_r/\forall_r,\ P(I) = A(I, r) \text{ and } Time(A, I, r) \leq T(|I|)$$

For example if you have n doors with prizes behind half of them the worst case time is $\frac{n}{2}$ because the adversary can put the prizes where ever the algorithm does not look. The randomized complexity is 2 because after the prizes are placed the algorithm looks in random doors and gets a prize with probability $\frac{1}{2}$.