Review HW13-15

Ying Zhao
Department of Computer Science, Tsinghua University

Design Points of GA

- Cast the optimization problem as one in which we make a choice and are left with subproblems to solve.

- Derive a recurrence:
  - Prove that there's always an optimal solution that makes the greedy choice, so that the greedy choice is always safe.
  - Prove optimal substructure: so that the greedy choice and optimal solutions to subproblems $\Rightarrow$ the optimal solution to the problem.
- Implement a top-down solution:
  - make the greedy choice and solve the remaining problem.
  - convert the **recursive** algorithm to an **iterative** algorithm.

- May have to preprocess input to put it into the greedy order.
  - Example: Sorting activities by finish time
  - Running time: $O(n \lg n)$ for sorting

Optimal Substructure

- Let $A_{i,j}$ be a subset of activities in A that start after a_i finishes and finish before a_j starts.
  - $A_{0,n+1}$ is the original input A.
- Suppose an optimal solution of $A_{i,j}$ includes activity a_k.
  - This generates two subproblems:
    - Selecting maximum # of compatible activities from $A_{i,k}$.
    - Selecting maximum # of compatible activities from $A_{k,j}$.
- Suppose $S_{i,j}$ is an opt solution to $A_{i,j}$ and $S_{i,j} = \{S_{i,k}, a_k, S_{k,j}\}$, then $S_{i,k}$ and $S_{k,j}$ must be optimal for $A_{i,k}$ and $A_{k,j}$, respectively.
  - Prove by using the cut-and-paste approach.
  - *Key*: the two subproblems are independent!

    Suppose S' is an opt. solution to $A_{i,k}$ and S'' is an opt. solution to $A_{k,j}$, activities in S' and activities in S'' are compatible to one another.

Final

- Time: 19:00-21:00, Dec 24, 2020 (Beijing Time)
  - Same settings as the midterm for on-line students (Exam Guide)
  - 19:00-19:30 check identity & get ready
  - 19:30-21:00 final exam

- Closed book closed notes
  - Calculator is OK.
  - Formulas (arithmetic series, geometric series) are provided.
  - Pseudocodes (or recurrences) are provided for input/output problems.

Contents

	Monday	Thursday	Readings	Pre-Requisite Self-Checking
11/2-11/6	**Introduction**	**Midterm：Combinatorics**	*Ch1.1*	
11/9-11/13	**Complexity**: Running time Growth of function Reduction	**Incremental Algorithms:** Insertion sort Loop invariant Dutch National Flag	*Ch 3* *Handout on Reduction* *Ch 2.1, 2.2*	1. *Counting the number of executions of statements in a pseudo-code* 2. *Mathematical Induction*
11/16-11/20	**Divide & Conquer-1:** DC fundamentals Merge sort Order statistics: select	**Divide & Conquer-2**: Solving Recurrences: Substitution, Recursion tree	*Ch 2.3, Ch 9.3* *Ch 4.3, 4.4*	1. *Merge sort (ch 6.4)* 2. *Quick sort (ch 7.1)* 3. *Summation Formulas (Appendix A.1)*
11/23-11/27	**Divide & Conquer-3**: Solving Recurrences: Master theorem	**Randomized Algorithms-1:** Probability analysis Balls and bins	*Ch 4.5, 4.6* *Appendix C.3 C.4* *Ch 5.4.2*	*Probability (Appendix C.1 C.2)*
11/30-12/4	**Randomized Algorithms-2:** Indicator Random Variable Hire assistant On-line hire assistant	**Randomized Algorithms-3:** RA fundamentals Randomized select	*Ch 5.1-5.3, 5.4.4* *Ch 8.3* *Ch 9.2*	
12/7-12/11	**Dynamic Programming-1:** Finding Subproblems Maximum Subarray, Knapsack	**Dynamic Programming-2:** Establishing Recurrences Matrix Chain Multiplication	*Ch 4.1* *Ch 15.2, 15.3*	
12/14-12/18	**Dynamic Programming-3:** Implementations: Top-down vs. Bottom-up	**Greedy Algorithms:** Activity selection GA fundamentals	*Ch 16.1, 16.2*	
12/21-12/25	**Review** **Q&A**	**Final：Algorithms**		

Review

- **Analysis of Algorithms:**
  - Running time analysis (asymptotically, best-case, worst-case, average-case, solving recurrences)
  - Correctness analysis (loop Invariant, optimal substructure, greedy property)
  - Lower bound (reduction)
- **Algorithm Methodology (top-down vs bottom-up):**
  - Incremental
    - Loop invariant
  - Divide & Conquer
    - solving recurrences
  - Randomized Algorithm
    - Probability analysis
  - Dynamic Programming
    - Optimal substructure property
  - Greedy Algorithm
    - Optimal substructure property
    - Greedy choice property
- **Typical Algorithms:**
  - Insertion sort, Merge sort, Dutch national flag, Strassen matrix multiplication, Select, on-line hiring, Bucket sort, Matrix-chain multiplication, Knapsack, Activity-selection

HW13-5.2-3

5.2-3

Use indicator random variables to compute the expected value of the sum of n dice.

HW13-5.4-2

5.4-2

Suppose that we toss balls into b bins until some bin contains two balls. Each toss is independent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses?

HW15-16.2-5

16.2-5

Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

Sample Exam

5. Dynamic Programming (18 points)
Let $S = \{a_1, a_2, \ldots, a_n\}$ be a set of n positive integers and let k be an integer. Give an $O(kn)$-time bottom-up dynamic programming algorithm to decide if there is a subset U of S that $\sum_{a_i \in U} a_i = k$. Your algorithm should return T (for 'true') if U exists and F (for 'false') otherwise.

Your answer should include:
1) The recurrence relation and a clear justification for it.
2) Pseudo code for the algorithm.

Let us define $f[i, j]$ to be a 2D boolean array. The state $f[i, j]$ will be **true** if there exists a subset of elements chosen from $S_i = \{a_1, a_2, \cdots, a_i\}$ with sum value of j. Ultimately we want to know the value of $f[n, k]$.

$$f[i,j] = \begin{cases} \textbf{true}, & \text{if } j = 0, \\ \textbf{false}, & \text{if } i = 0 \text{ and } j > 0, \\ f[i-1, j], & \text{if } i > 0 \text{ and } 0 < j < a_i, \\ f[i-1, j] \vee f[i-1, j-a_i], & \text{otherwise.} \end{cases}$$