CS 577: Introduction to Algorithms

Discussion 14- Review

Will Martin

UW-Madison

May 3, 2023

Upcoming Homeworks and Quiz

- HW 13- Randomization
  - Out: 04/27
  - Due: 05/4 @ 11:59pm
- Final Exam
  - 05/10 @ 7:45 - 9:45 am
  - Open book
  - Max grade: 50/40
  - 5 problems
    1. Graph/greedy
    2. Divide and Conquer
    3. Dynamic Programming
    4. Network Flow
    5. Intractability

May 2023

	1	2	**3**	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

June 2023

				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Discussion 14- Review

General Test Taking Advice

- Skim through the entire exam first. Find the problems you are most comfortable answering
- Questions will not be labeled with their type, should be obvious
- Partial credit is possible, write something down
- Expect to have enough time to solve the problems, not re-learn how to solve the problems by looking at notes
- If asked to write an algorithm:
  - Just words without incredibly detailed pseudocode can earn full points
  - If your algorithm uses a well-known algorithm (Dijkstra's, Merge Sort, BFS/DFS, Ford-Fulkerson, etc.) you do not need to rewrite the pseudocode for those algorithms
- You will be asked to prove correctness/runtime

Study Tips

- Prepare a note sheet
- Print out discussion reviews and bring them to the exam
- Review homework problems with solutions posted on Canvas
- Practice exam on Canvas

Review Topics

Graph/Greedy

2-Colorable

Given a graph, determine whether the graph is 2-colorable. Specifically, can each node be assigned one of 2 colors so that no two neighboring nodes share a color?

Graph/Greedy

Solution

1. BFS, assigning "red" to the first layer, "blue" to the second layer, "red" to the third layer, etc.
2. Go over all the edges and check whether the two endpoints of this edge have different colors.

Runtime: $O(|V| + |E|)$

Graph/Greedy

Greedy Proof Strategy Review

Stays Ahead Proof:

1. Establish time step
2. Use induction over time step to show some measure is the same or better over all time steps
3. Use the property shown in the induction to claim the optimality

Graph/Greedy

Greedy Proof Strategy Review

Exchange Argument Proof:

1. Determine characteristic properties of greedy solution that may not be shared by optimal solution
2. Define exchanges needed to transform optimal solution into greedy solution without changing optimality of optimal solution
3. Use induction to transform optimal solution into greedy solution

Graph/Greedy

Trip Planning

Your goal is to follow a pre-set route from New York to Los Angeles. You can drive 500 miles in a day, but you need to make sure you can stop at a hotel every night (all possibilities are premarked on your map). You'd like to stop for the fewest number of nights possible.

How should you plan?

Prove the optimality of your algorithm.

Graph/Greedy

Solution

Algorithm:
Go as far as you can every night

Proof:
Stays Ahead Proof

Graph/Greedy

Solution

Proof:

1. Establish time step
    - Night i
2. Use induction over time step to show some measure is the same or better over all time steps
    - Measure: distance traveled
    - Base case: $i = 1$. Greedy chooses the farthest out hotel so must be at least as far as optimal
    - Inductive Step: Assume $i = k$ holds, show for $i = k + 1$. When selecting the hotel on night $k + 1$ choose any hotels within 500 mi of k hotel. Since k hotel by greedy is at least as far as optimal solution, optimal solution can't find a hotel further along than greedy for $k + 1$.

Graph/Greedy

Solution

Proof:

3. Use the property shown in the induction to claim the optimality
    - The distance at the last hotel by the greedy algorithm must be at least as far as optimal solution so an extra night is not needed. Therefore greedy solution is optimal.

Divide and Conquer

Maximum Subarray Sum

Given a one dimensional array that may contain both positive and negative integers, find the largest sum of a contiguous subarray.

For example, if the given array is

```
[-2, -5, 6, -2, -3, 1, 5, -6]
```

then the maximum subarray sum is 7 using this subarray

```
[6, -2, -3, 1, 5]
```

Divide and Conquer

Solution

Naive method: Using two loops try every subarray and compare to an overall maximum. This method has a $O(n^2)$ runtime.

Divide and Conquer

Solution

Divide and Conquer method: Run the following algorithm:

1. Divide the given array in two halves
2. Return the maximum of following three
   - Maximum subarray sum in left half (Make a recursive call)
   - Maximum subarray sum in right half (Make a recursive call)
   - Maximum subarray sum such that the subarray crosses the midpoint

Divide and Conquer

Solution

Divide and Conquer method:

- To find the maximum subarray that crosses the midpoint find the maximum sum starting from midpoint and ending at some point on left of mid
- Then find the maximum sum starting from mid + 1 and ending with some point on right of mid + 1
- Add sums
- $O(n)$ time.

Divide and Conquer

Solution

Recurrence relation:

$$T(n) = 2T(n/2) + O(n), \quad T(1) = 1$$

Solving recurrence relation using unrolling or recursion tree should produce:

$$T(n) = O(nlogn)$$

Dynamic Programming

Coin Change Problem:

You are given an integer array $D = [d_1 ... d_i]$ representing i coins of different denominations and an integer amount, A, representing a total amount of money.

Return the fewest number of coins that you need to make up amount A. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

Dynamic Programming

Solution

Fill a 1D array $C[0...A]$ with $A+1$ values from index 0 to A using the following Bellman equation:

$$C[p] = \begin{cases} 0 & \text{if } p = 0 \\ min_{i:d_i \le p}\{1 + C[p - d_i]\} & \text{if } p > 0 \end{cases}$$

The $p-th$ element of C corresponds to the minimum number of coins needed to provide change for the amount p. Therefore the solution is held in $C[A]$.

Runtime: Filling array C takes $O(A * i)$ time while retrieving the solution takes $O(1)$ time. Therefore, the overall time complexity is $O(A * i)$.

Network Flow

Matrix Rounding

Suppose you are given an array $A[1..m][1..n]$. Elements in the array, $0 \leq x_{ij} \leq 1$. We want to round A to a matrix by replacing every element by $\lfloor x_{ij} \rfloor$ or $\lceil x_{ij} \rceil$ without changing the sum of each row and column.

Network Flow

Solution

- Create node for each row and each column
- Connect each row to each column with edge capacity = 1
- Connect each row to the source with edge capacity = row sum
- Connect each column to the source with edge capacity = column sum
- Remove edges with original $x_{ij} = 0$ or $x_{ij} = 1$
- Decrease row and column sum for edges with original $x_{ij} = 1$

How do you recover the solution from the residual graph?

Network Flow

Box Nesting

Suppose we are given a set of boxes, each specified by their height, width, and depth in centimeters. As you should expect, one box can be placed inside another if the first box can be rotated so that its height, width, and depth are respectively smaller than the height, width, and depth of the second box. Boxes can be nested recursively. Call a box visible if it is not inside another box.

Describe and analyze an algorithm to nest the boxes so that the number of visible boxes is as small as possible.

Network Flow

Solution

- Create a bipartite graph
- Left side and right side consist of nodes representing all of the boxes
- Connect boxes that can fit inside of each other

How do you recover the solution from the residual graph?

Intractability

Directed Disjoint Paths Problem

The DIRECTED DISJOINT PATHS PROBLEM is defined as follows: We are given a directed graph G and k pairs of nodes $(s_1, t_1), \ldots, (s_k, t_k)$. The problem is to decide whether there exist node-disjoint paths $P_1, \ldots, P_k$ so that P_i goes from s_i to t_i.

Show that DIRECTED DISJOINT PATHS PROBLEM is NP-Complete.

Use a reduction from 3-SAT.

Intractability

Solution

Show:
DIRECTED DISJOINT PATHS PROBLEM $\in \texttt{NP}$.

- Given a set of paths $P_1, \ldots, P_k$
- For each path P_i, iterate through the path and flag each node
- If you encounter a node that is already flagged, then the solution is invalid
- Runtime: $O(|V|)$

Intractability

Solution

Show:
Directed Disjoint Paths Problem $\in$ NP-Hard.

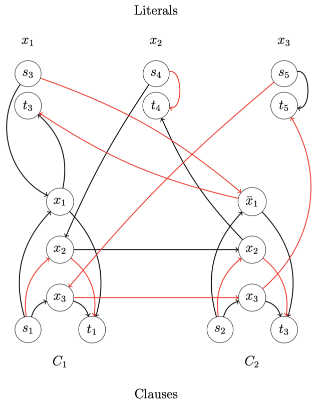
3-SAT $\leq_p$ Directed Disjoint Paths Problem

Intractability

Solution

EXAMPLE:

$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$
$x_1 = T, x_2 = T, x_3 = F$



Intractability

Solution

3-SAT $\leq_p$ Directed Disjoint Paths Problem

Prove:
A set of 3-SAT clauses is satisfiable $\iff$ Directed Disjoint Paths Problem is satisfiable on G with starting and ending pairs $(s_1, t_1), \ldots, (s_k, t_k)$

Intractability

Solution

$(\implies)$

- There is a node-disjoint path from each literal's s to t node. We choose the path to the left if the literal is false and to the right if it is true.
- This eliminated nodes that evaluate to false for the next step.
- There is a node-disjoint path from each clause's s to t node that passes through the literal that satisfies the clause.

Intractability

Solution

($\Longleftarrow$)

- Every clause is satisfied because there is a path from each clause's s to t node.
- Each literal holds the same value throughout all clauses because there is a path from each literal's s to t node