EECS 4020 Algorithms

HW3

I. Dynamic Programming

Q1

- Consider a $k \times n$ chessboard
- n pieces of $1 \times k$ bars
  - Can be placed vertically or horizontally

ways to cover the board with the bars ?

Q1 [solution]

Let F_n denote the number of ways

Depending on how to cover the top-left corner:

- If covered by a vertical bar ➔ F_{n-1} ways
- If covered by a horizontal bar ➔ F_{n-k} ways

➔ $F_n = F_{n-1} + F_{n-k}$ ($F_0 = F_1 = \ldots = F_{k-1} = 1$)

Q2

- Given a sequence S of n distinct numbers
- Find a longest subsequence whose numbers are increasing
  - i.e., longest increasing subsequence in S

How to do so?

Q2 [solution]

Method 1 (solving an LCS problem) :

Step 1: Sort S into S^*

Step 2: Compute longest common subsequence between S and S^*

$\boxed{\text{Running time is } \Theta(n \log n) + \Theta(n^2) = \Theta(n^2)}$

Q2 [solution]

Method 2 :

This problem can be solved in $\Theta(n \log n)$ time in various ways, using suitable data structures to help

➔ Search for LIS problem for more details

Q3

- An array of n sushi dishes with different prices
- Select dishes with the following rules:
  - (1) from left to right, and
  - (2) price is increasing

How to maximize total price of selected dishes?

Q3 [solution]

Let $M[k]$ = max price to get with first k dishes,

with the k^{th} dish must be selected

Let $p[k]$ = price of the k^{th} dish

➔

$$M[k] = p[k] + \max\{\, M[j] \mid j < k \text{ and } p[j] < p[k] \,\}$$

Q3 [solution]

Desired answer is:

$$\max \{ M[k] \mid k = 1, 2, \ldots, n \}$$

Each $M[k]$ can be computed in $O(n)$ time

➔ Running time is $O(n^2)$

Q4

- A rooted tree with n nodes
- Placing a guard at a node v can protect all the edges incident to v

How to find min # guards to protect every edge?

Q4 [solution]

- For a node v, let T_v denote subtree rooted at v
- We use

$\text{Best}_0[v]$ = min # guards to protect all edges in T_v with no guard placing at v

$\text{Best}_1[v]$ = min # guards to protect all edges in T_v with a guard placing at v

Q4 [solution]

- Then, we have

$$\mathrm{Best}_0[v] = \sum_{c \text{ is child of } v} \mathrm{Best}_1[c]$$

$$\mathrm{Best}_1[v]$$

$$= 1 + \sum_{c \text{ is child of } v} \min\{\mathrm{Best}_0[c], \mathrm{Best}_1[c]\}$$

Q4 [solution]

Desired answer is:

$$\min \{ \ Best_0[r], Best_1[r] \ \}$$

where r is the root of the tree

Each $Best[v]$ can be computed in $O(1)$ time

➔ Running time is $O(n)$

Q5

- A rooted tree with n nodes
- Each node has a positive value
- Color a node v can get the value of v
- No adjacent nodes can be colored

How to color nodes to get max total value?

Q5 [solution]

- For a node v, let T_v denote subtree rooted at v
- We use

$\text{Best}_0[v]$ = max value we can get from T_v with v not colored

$\text{Best}_1[v]$ = max value we can get from T_v with v colored

Q5 [solution]

- Then, we have

$$\text{Best}_0[v] = \sum_{c \text{ is child of } v} \max\{\text{Best}_0[c], \text{Best}_1[c]\}$$

$$\text{Best}_1[v] = \text{value}(v) + \sum_{c \text{ is child of } v} \text{Best}_0[c]$$

Q5 [solution]

Desired answer is:

$$\max \{ \ Best_0[r], Best_1[r] \ \}$$

where r is the root of the tree

Each $Best[v]$ can be computed in $O(1)$ time

➔ Running time is $O(n)$

Q6

- n objects, with values and integral volumes
- A bag with volume V
- Put an object s to bag can get the value of s
- Total volume cannot exceed V

How to pick objects to get max total value?

Q6 [solution]

Let $M[k][v]$

$= $ max total value to get with first k objects,

with total volume exactly v

➔

$$M[k][v] = \max\{ M[k-1][v],$$
$$\text{value}(k) + M[k-1][v - \text{volume}(k)] \}$$

Q6 [solution]

Desired answer is:

$$\max \{ M[n][0], M[n][1], \ldots, M[n][V] \}$$

Each $M[k][v]$ can be computed in $O(1)$ time

➔ Running time is $O(nV)$

II. Greedy Algorithm

Q1

- A car with full tank of gas can travel a distance of d units
- We want to travel from A to B
- Gas stations are along the way

How to minimize # gas stations to visit ?

Q1 [solution]

1. Pick the farthest gas station S within distance d from A
   ➔ this choice is correct [by cut-and-paste]
2. Fill the gas tank to make it full
3. Recursively find the gas stations to visit for the remaining distance from S to B

Q2

- n points are located on the x-axis
- Line-segments of unit-length can be used to cover the points
- Need to cover all the points

How to use minimize # line segments?

Q2 [solution]

1. Cover leftmost point p with a line segment L whose left boundary aligns with p
   ➔ this choice is correct [by cut-and-paste]
2. Remove all points covered by L
3. Recursively find line segments to cover the remaining points

Q3

- n items, each with a weight
- Pack the items in bags
- Weight limit of each bag: W

How to use as few bags as possible ?

Q3 [solution]

- This problem is NP-hard

  ➔ No known efficient algorithm so far

- We will solve the problem by a heuristic

  ➔ Not optimal, but may be good

Q3 [solution]

- Here is the heuristic :

Start with an empty bag B_1. Set $i = 1$.

while (there is an item s not in the bags)

    if (any of the bags $B_1, B_2, \ldots, B_i$ can hold s)

        Put s in that bag;

    else

        Put s in a new empty bag B_{i+1}, and then update i as $i + 1$.

Q3 [solution]

(a) The heuristic may not be optimal :

Consider items in the following input order

0.2W 0.5W 0.7W 0.4W

The heuristic uses 3 bags, while optimal 2

Q3 [solution]

(b) Suppose the heuristic uses m bags

Then, m – 1 bags are more than half full :

If not, 2 bags B and B' are at most half full.

Say, B is created first. Since we can only start a bag when no bags can hold the current item

➔ all items in B' would be in B

➔ contradiction

Q3 [solution]

(c) Based on the result of (b), we see that :

Whenever the heuristic uses m bags,

total weight of all items is $\geq (m - 1) W / 2$,

so any algorithm, including the optimal one,

must use at least $(m - 1) / 2$ bags

➔ # bags used by heuristic is roughly within a factor of 2 from # bags used by optimal

Q4

- n kids, n toys
- Each kid specifies top 3 favorite toys
- Some toys will be donated
- Target: Keep at least one toy for each kid

Show that n / 3 toys can be donated

Q4 [solution]

1. Keep the most popular toy, say t
2. Kids specifying t as a favorite toy are satisfied; remove them from further consideration
3. Recursively keep the toys to satisfy the remaining kids

Q4 [solution]

- How good is the algorithm ?

  Let us divide the process into two phases :

  Phase 1: the toy kept satisfies at least 2 kids

  Phase 2: the toy kept satisfies only 1 kid

Q4 [solution]

Let T_1 and T_2 denote the number of toys kept in the two phases, respectively

In Phase 1, each toy satisfies at least 2 kids,

➔ $$T_1 \le n / 2$$

Q4 [solution]

In Phase 2, for each remaining kid,

- its 3 favorite toys are disjoint with the others' (else, we are still in Phase 1)
- these toys are not kept (else, the kid is removed)
- each is satisfied by one toy

➔ $$T_2 \leq (n - T_1) / 3$$

Q4 [solution]

Combining everything :

$$\begin{aligned} \#\text{ toys kept} &= T_1 + T_2 \\ &\leq T_1 + (n - T_1) / 3 \\ &= n / 3 + 2 T_1 / 3 \quad \leq \quad 2n / 3 \end{aligned}$$

$\Rightarrow$ we can donate $n / 3$ toys