Greedy Approach

Week 12

Required Activities

- Check Announcements regularly (every 2-3 days)
- Read FA book: Chapter 4
- Keep working on Assignment 4 due Jan 25

Greedy Approach

- Works in stages where it selects in sequence the item it considers “best” (locally for that stage) regardless of past or future choices
- Often leads to simple and efficient solutions
- Does not guarantee an optimal solution for every algorithm
- Steps:
  - Start with empty set
  - Add items in sequence until reach solution
  - Each iteration has:
    a) Selection procedure – selects next item to add to set that is locally optimal
    b) Feasibility check – checks if set is feasible to reach solution and if not rejects it
    c) Solution check – determines of new set provides solution

Greedy Approach cont.

- Advantage: easy to develop algorithm and code with no repetitious calculation
- Disadvantage: does not always provide the optimum solution
- Examples of greedy algorithms:
  - Prim's (minimum spanning tree)
  - Kruskal's (minimum spanning tree)
  - Dijkstra's shortest path (graph)
  - Scheduling algorithms
  - Huffman code (encoding for data compression)

Minimum Spanning Tree (MST)

- **MST problem:** Given connected, weighted, and undirected graph G remove edges from G such that G remains connected and the sum of the weights on the remaining edges is minimal
- **Prim's Algorithm** – greedy approach produces MST (Algorithm 4.1)
  - Represents graph as adjacency matrix with values as weight on edge
  - Keep values in arrays for distance (same as weight) and nearest
  - Basic algorithm:
    - Selects arbitrary **vertex** from graph to start
    - Adds nearest vertex (connected with min weight) and edge to solution set and remove edge from graph
    - Solution set is always a tree at each repetition
    - Repeats adding until n-1 edges are added
- **Kruskal's Algorithm** - greedy approach produces MST (Algorithm 4.2)
  - Sorts edges in non-decreasing order of weight
  - Basic algorithm:
    - Selects least weighted **edge**, add to solution set, and remove from graph
    - Adds **edge** that connects with min weight (without cycle) and remove edge from graph
    - Repeats adding until n-1 edges are added

Prim's vs Kruskal's

- Prim's initializes with vertex while Kruskal's with edge
- Prim's uses previous node to connect to new while Kruskal's selects the next min edge so has to check for cycle node
- Prim's Algorithm has time complexity $O(n^2)$
- Kruskal's Algorithm when graph is very sparse time complexity is O(nlogn) so faster than Prim's
- Kruskal's Algorithm when graph is highly connected time complexity is n^2logn so Prim's is faster
- The above time complexities are for the algorithms given; they can be improved on based on the data structures used. See textbook p. 169 section 4.1.4 section for examples

Scheduling

- Minimize total time in system such as to minimize wait and serve time for user's access to resource (**interval scheduling** where want to schedule as many jobs as possible in shortest time without overlapping)
- Scheduling with deadlines where serve time is same but there is deadline to yield highest profit so need to maximize profit (job sequencing where only one job can be done at a time)
- Given a set of fixed jobs schedule with as few resources as possible

Scheduling with Deadline

- Each job takes one unit of time
- If job starts before or at its deadline, profit is obtained, otherwise no profit
- Goal is to schedule jobs to maximize the total profit
- Not all the jobs have to be scheduled

Consider this example:

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

Job 1 has deadline 2, therefore it can start at time 1 or 2.
Job 2 has deadline 1, therefore it can start at time 1.
Job 3 has deadline 2, therefore it can start at time 1 or 2.
Job 4 has deadline 1, therefore it can start at time 1.

Schedule	**Total Profit**
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	40 + 30 = 70
[4, 3]	40 + 25 = 65

Scheduling with Deadline

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

Schedule	Total Profit
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	40 + 30 = **70**
[4, 3]	40 + 25 = 65

Definitions:

- A sequence is a **feasible sequence**, if all the jobs in the sequence start by their deadlines. e.g., [4, 1] is a feasible sequence, where as [1, 4] is not.
- A set of jobs is called **feasible set** if there is at least one feasible sequence for the jobs in the set. E.g., {1, 4} is a feasible set because of [4,1], where as {2, 4} is not because there is no sequence with both jobs 2 and 4.
- A feasible sequence with maximum total profit is called **optimal sequence** ➔[4,1].

Scheduling with Deadline (Algorithm 4.4)

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

1. Sort according to Profit:

Job	Deadline	Profit
4	1	40
2	1	35
1	2	30
3	2	25

2. J set to [4]
3. K set to [4, 2] rejected because not feasible
4. K set to [4, 1] and is feasible
5. J is set to [4, 1] because K is feasible
6. K is set to [4, 1, 3] rejected because not feasible
7. **Final value of J is [4, 1] so we have Profit 40 + 30 = 70**

Knapsack Problem

- Thief breaks into jewelry store carrying a knapsack in which to place stolen items
- Knapsack has a weight capacity W**,** knapsack will break if weight of stolen items exceeds W, and each item has a value
- Thief's dilemma is to maximize the total value of items stolen while not exceeding the total weight capacity W
- Real application is resource allocation where there are financial constraints such as for example, finding the least wasteful way to cut raw materials or selection of investments and portfolios or luggage with weight limit and need to bring most items with greatest value.

Knapsack Solutions

- **Given:** W max weight allowed, n=num of items, v[0..n] value of items, w[0..n] weight of items, all values are positive

- **Solution: Dynamic Programming (P[1..n][1..n] matrix to calculate solution where P[n][W] has max profit)**

```
array P[n+1][W+1]   // matrix [#items] by [max weight]
Loop i=0 to n
        Loop j=0 to W
                if i==0 or j==0 P[i][j]=0   // not used
                else if w[i-1] > j then P[i][j] = P[i-1][j]  // item weight greater than current weight limit so reject and use previous weight
                else P[i][j] = max(P[i-1][j], P[i-1][j-w[i-1] ] + v[i-1])  // assign higher value of previous or current weight
Return P[n][W]
```

Time complexity **O(nW)** and space complexity **O(nW)**

- **Solution: Greedy Approach (profit has max profit)**

```
Sort for value/weight
profit = 0, currentW = 0
Loop i=0 to n
        if (currentW + w[i] <= W)
                currentW = currentW + w[i]
                profit = profit + v[i]
Return profit
```

Dynamic Solution

v [] = {5, 8, 15}
w[] = {1, 2, 3};
W = 2
n=3

```
Loop i=0 to n
  Loop j=0 to W
    if i==0 or j==0 P[i][j]=0
    else if w[i-1] > j then P[i][j] = P[i-1][j]
    else P[i][j] = max(P[i-1][j], P[i-1][j-w[i-1] ] + v[i-1])
```

i loops 0 to 3 and j loops 0 to 2
i=0, j=0 => P[0][0] = 0 then loops through all j and makes P[0][1..2] = 0
i-1, j= 0 => P[1][0] = 0
i=1, j=1 => w[0] > 1 => **1 > 1 (false)** then P[1][1] = max: P[0][1] or P[0][1-1] + 5 => 0 or 5 => P[1][1] = 5
i=1, j=2 => w[0] > 2 => **1 > 2 (false)** then P[1][2] = max: P[0][2] or P[0][2-1] + 5 => 0 or 5 => P[1][2] = 5
i=2, j=0 => P[2][0] = 0
i=2, j=1 => w[1] > 1 => **2 > 1 (true so reject)** then P[2][1] = P[1][1] = 5
i=2, j=2 => w[1] > 2 => 2 > 2 (false) then P[2][2] = max: P[1][2] or P[1][2-2] + 8 => 5 or 8 => P[2][2] = 8
i=3, j=0 => P[2][0] = 0
i=3, j=1 => w[2] > 1 => **3 > 1 (true so reject)** then P[3][1] = P[2][1] = 5
i=3, j=2 => w[2] > 2 =>3 > 2 **(true so reject)** then P[3][2] = P[2][2] = 8 (profit given weight constraint)

Greedy Solution

v [] = {5, 8, 15}
w[] = {1, 2, 3};
W = 2
n=3

```
profit = 0, currentW = 0
Loop i=0 to n
        if (currentW + w[i] <= W)
                currentW = currentW + w[i]
                profit = profit + v[i]
Return profit
```

value/weight = {5/1, 8/2, 15/3} = {5, 4, 5}
Sort in order of largest profit per unit weight first: {5/1, 15/3, 8/2}

profit=0
currentW=0
i=0; 0+1 <= 2 true ➔ currentW=0 + 1 = 1 profit=0+5
i=1; 1+3 <= 2 false

Return 5 (not optimal solution; may than use fractions but not applicable to all problems)

Questions ?

- Post in the discussions
- Send email to RMcFadden@HarrisburgU.edu
- Respond usually within 48hours