THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

GREEDY ALGORITHMS

KRUSKAL'S ALGORITHM USING UNION FIND
MINIMUM SPANNING TREE
GREEDY ALGORITHMS AND MATROIDS

Design and Analysis of Algorithms

LOGISTICS

- Instructor

  Prof. Amrinder Arora

  amrinder@gwu.edu

  Please copy TA on emails

  Please feel free to call as well

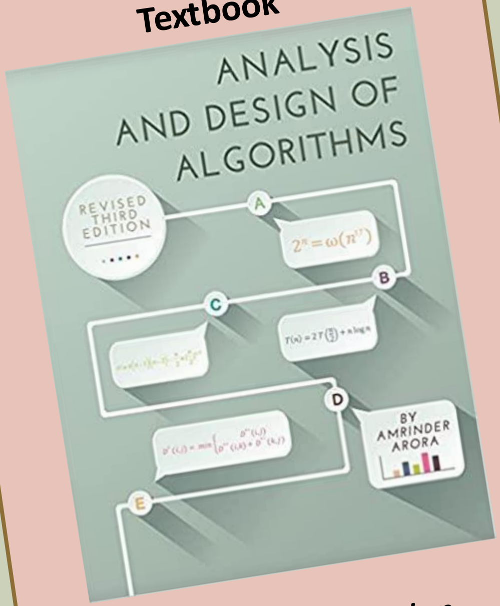
  ☺

- Available for study sessions

  Science and Engineering Hall

  GWU

Textbook



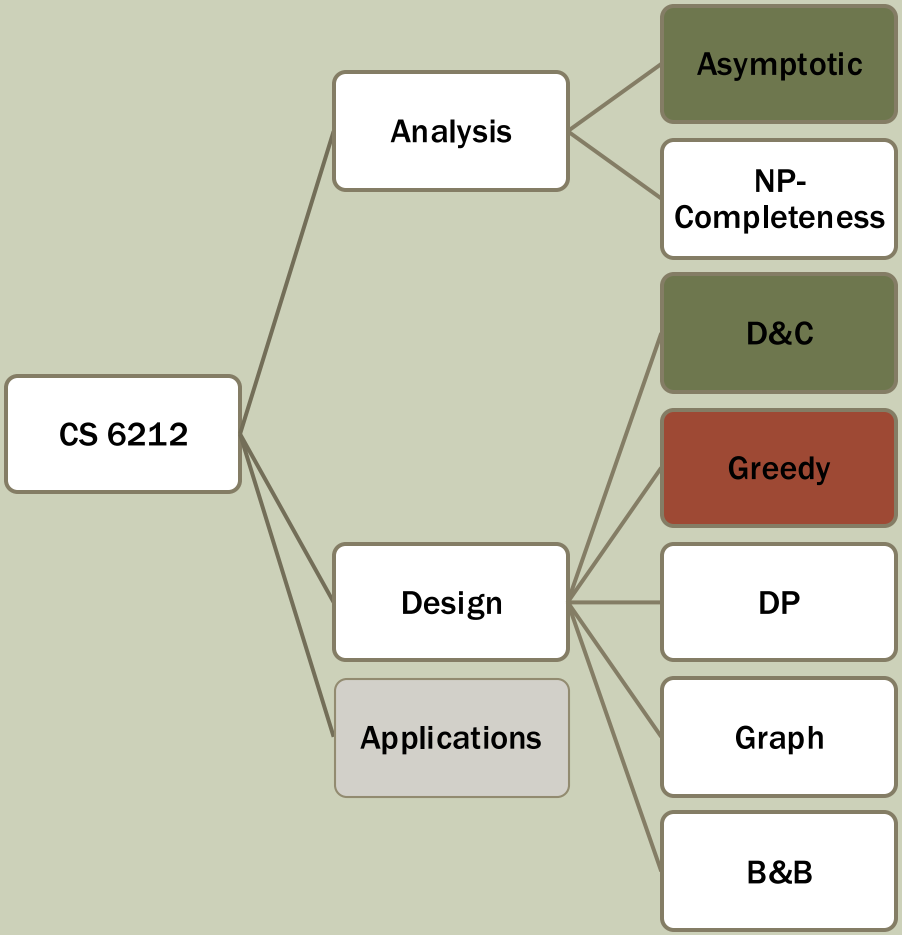
Revised Third Edition

Cognella Academic Publisher

www.cognella.com

ISBN: 978-1793549952

WHERE WE ARE



- ✓ Done
- ✓ Done
- ❖ Starting today..

GREEDY METHOD

- A technique to build a complete solution by making a sequence of “best selection” steps
- Selection depends upon actual problem
- Focus is simply on “what is best step from this point”

APPLICATIONS

- Applications of greedy method are *very* broad.
- Examples:
  - Sorting
  - Merging sorted lists
  - Knapsack
  - Minimum Spanning Tree (MST)
  - Hoffman Encoding

Application # 1

SORTING USING GREEDY METHOD

- Select the minimum element
- Move it to the beginning
- Continue doing it for the remaining array

Given array a[1..n] of unsorted numbers

- For i = 1 to n-1
  - For j = i+1 to n
    - If (a[i] > a[j]) swap (a[i], a[j])

INSERTION SORT, EXAMPLE RUN..

- 1, 5, 4, 19, 2, 90, 3
- Objective: To sort the array
- 1, 2, 4, 3, 5, 19, 90
- =================

TIME COMPLEXITY ANALYSIS

- How long does it take to sort using greedy method?
- Is it optimal?

Application # 2

MERGING SORTED LISTS

- Input: n sorted arrays of lengths L[1], L[2],…,L[n]
- Problem: To merge all the arrays into one array as fast as possible. Which pair to merge every time?

- We observe that:
  - The final list will be a list of length L[1] + L[2] + … + L[n]
  - The final list will be same regardless of the sequence in which we merge lists
  - However, the time taken may not be the same.

MERGING TWO LISTS

- List 1 of size 7: {1, 2, 5, 21, 23, 44, 64}
- List 2 of size 12: {1, 4, 15, 16, 17, 19, 34, 38, 56, 63, 69, 89}

- Merged list of size 19 (in time 19):
- {1, 1, 2, 4, 5, 15, 16, 17, 19, 21, 23, 34, 38, 44, 56, 63, 64, 69, 89}

- You can actually prove that merging can take up to n1 + n2 – 1 in the worst case. O(n1 + n2) time.

EXAMPLE

- 5 Lists of sizes: 20M, 25M, 30M, 35M, 40M
- Finally, when it is merged, we will have ONE list of size 150M.

Option 1: ((((1, 5), 3), 2), 4)

- 20 with 40 → 60 (in 60 units of time)
- 60 with 30 → 90 (in 90 units)
- {25, 35, 90}
- 25 with 90 → 115 (in 115 units of time)
- 115 with 35 → 150 (in 150 units of time)
- Total time = 60 + 90 + 115 + 150 = 415M units of time

- Optimal: 45 + 65 + 85 + 150 = 345M

MERGING SORTED LISTS

- Greedy method: Merge the two shortest remaining arrays.
- To Implement, we can keep a data structure, that allows us to:
  - Remove the two smallest arrays
  - Add a larger array
  - Keep doing this until we have one array

MERGING SORTED LISTS

- Implement using heap
- Build the original heap – O(n) time
- For i = 1 to n-1
  - Remove two smallest elements: 2 log (n)
  - Add a new element log(n) time
- Total time: O(n log n)
  - Here n is the number of sorted lists. n has NOTHING to do with the number of elements in any of the lists – that is entirely outside of our knowledge, we are only given the relative sizes of the lists.

Application # 3

KNAPSACK PROBLEM

- Input: A weight capacity C, and n items of weights W[1:n] and monetary value V[1:n].
- Problem: Determine which items to take and how much of each item so that the total weight is ≤ C, and the total value (profit) is maximized.
- Formulation of the problem: Let x[i] be the fraction taken from item i. $0 \le x[i] \le 1$.
  The weight of the part taken from item i is x[i]*W[i]
  The Corresponding profit is x[i]*V[i]
- The problem is then to find the values of the array x[1:n] so that x[1]V[1] + x[2]V[2] + ... + x[n]V[n] is maximized subject to the constraint that x[1]W[1] + x[2]W[2] + ... + x[n]W[n] ≤ C

KNAPSACK

- Given a list of resources, select some of them, such that:
  - Your benefits are maximized
  - Your cost remains with the budget constraint

- "Cost Benefit Optimization" or "Best Bang for the Buck"

- 5 Million Visitors for 1 Million $

vs.

- 9 Million Visitors for 3 Million $

3 OPTIONS

- **Policy 1**: Choose the lightest remaining item, and take as much of it as can fit.
- **Policy 2**: Choose the most profitable remaining item, and take as much of it as can fit.
- **Policy 3**: Choose the item with the highest price per unit weight (V[i]/W[i]), and take as much of it as can fit.

- Exercise: Prove by a counter example that Policy 1 does not guarantee an optimal solution. Same with Policy 2. Policy 3 always gives an optimal solution

EXAMPLE

Item #	***1***	***2***	***3***	***4***	***5***
V ($)	3	5	10	11	9
W (lb)	1	2	5	6	7
V/W	3	2.5	2	1.83	1.28

- Capacity = 7
- Solution:
  1. All of items {1, 2} and a fraction of item 3
  2. But, how to handle this problem instance if we cannot take "fractional" portions of items.

EXAMPLE 2

Item #	***1***	***2***	***3***	***4***	***5***
V ($)	4	5	9	12	7
W (lb)	5	2	6	6	10
V/W	0.8	2.5	1.5	2	0.7

- Capacity = 10
- Optimal Solution Value: 5 + 12 + 3 = 20.

IS GREEDY ALGORITHM FOR INTEGER KNAPSACK PROBLEM OPTIMAL?

- No, in fact, it can be as bad as you want to make it to be.
  - Example?
- A simple fix can make this algorithm only as bad as a ratio of 2.
  - How?

Application # 4

MINIMUM SPANNING TREE

- Definitions
  - A spanning tree of a graph is a tree that has all nodes in the graph, and all edges come from the graph
  - Weight of tree = Sum of weights of edges in the tree
- Statement of the MST problem
  - Input : a weighted connected graph G=(V,E). The weights are represented by the 2D array (matrix) W[1:n,1:n], where W[i,j] is the weight of edge (i,j).
  - Output: Find a minimum-weight spanning tree of G.

GREEDY ALGORITHM

- Selection Policy: Minimum weighted edge that does NOT create a cycle.
- Procedure ComputeMST(in:G, W[1:n,1:n]; out:T)

    Sort edges: e[1], e[2], .. e[m].

    Initialize counter j = 1

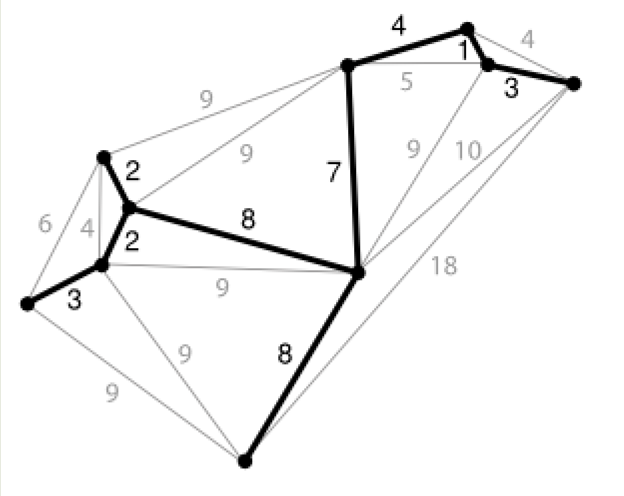
    Initialize tree T to empty

    While (number of edges in Tree < n-1) {

        Does adding an edge e[j] create a cycle?

        If No, add edge e[j] to tree T

    }



HOW TO MAKE THIS EFFICIENT?

Sort edges: e[1], e[2], .. e[m].

Initialize counter j = 1

Initialize tree T to empty

While (number of edges in Tree < n-1) {

  Does adding an edge e[j] create a cycle?

  If No, add edge e[j] to tree T

}

HOW TO MAKE THIS EFFICIENT?

Sort edges: e[1], e[2], .. e[m]. **O(m log n)**

Initialize counter j = 1 O(1)

Initialize tree T to empty O(1)

While (number of edges in Tree < n-1) {

Does adding an edge e[j] create a cycle?

If No, add edge e[j] to tree T

}

Suppose this takes f(n,m) time

Suppose this takes g(n,m) time

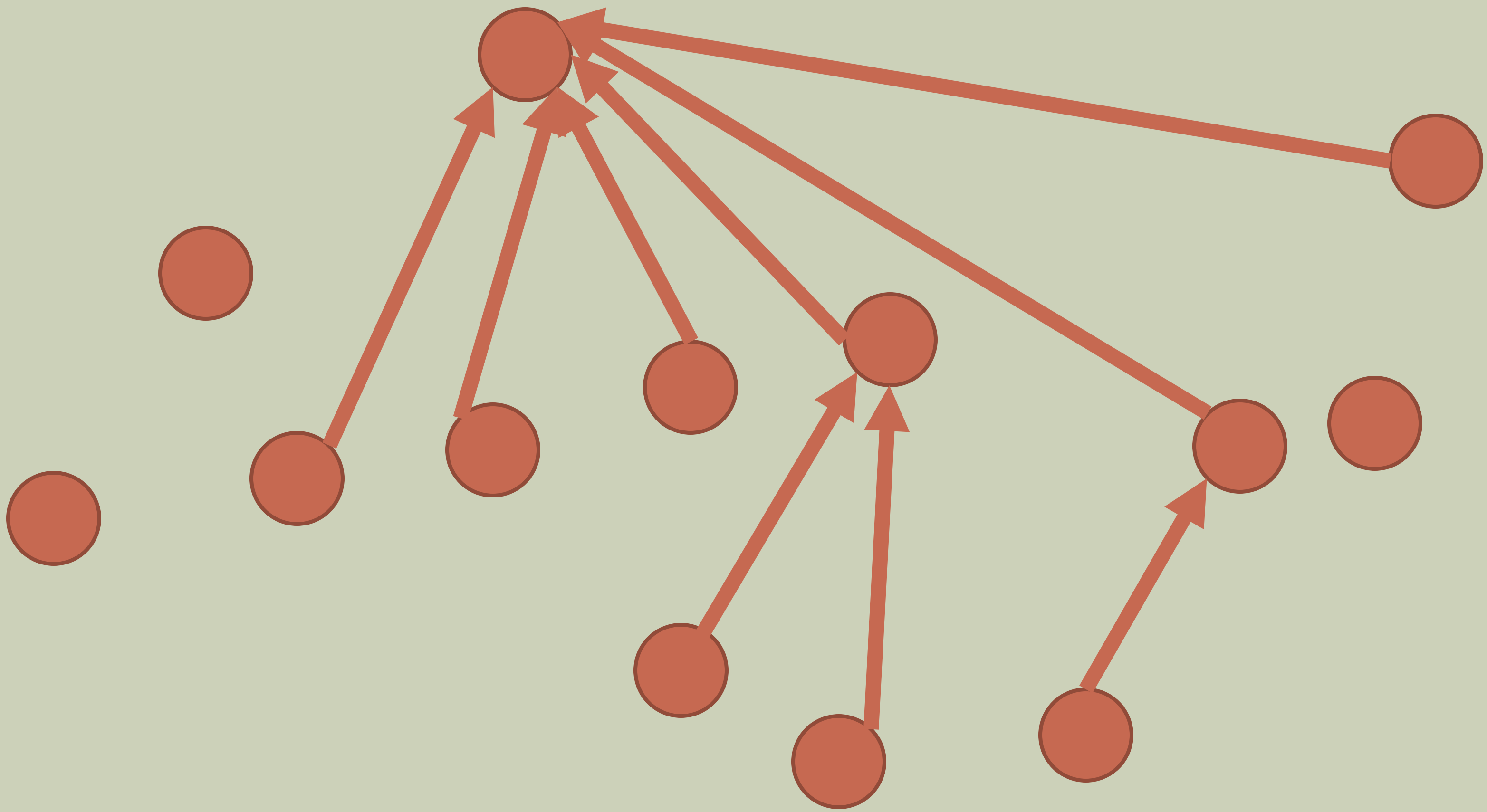
Then, total time complexity becomes:

m log n + f(n,m) * m + g(n,m) * n

UNION FIND DATA STRUCTURE

- Each set is marked by a leader – the root node
- When calling "find" on a set's member, it returns the leader
- Leader maintains a rank (or height)
- When doing a union, make the tree with smaller height (or rank) to be a child of the tree with the larger height
- Note that this is NOT a binary tree.

UNION FIND != BINARY TREE



UNION FIND – PATH COMPRESSION

- When doing a find, follow that up by compressing the path to the root, by making every node (along the way) point to the root.
- This is not easy to prove, but Union Find with Path compression, when starting with n nodes and m operations, takes $O(m \log^*(n))$ time instead of $O(m \log n)$ time, where the $\log^*$ function is the iterated logarithm (also called the super logarithm) and is an **<u>extremely</u>** slow growing function.
- $\log^*(n)$ is defined as follows:
  - 0, if $n <= 1$
  - $1 + \log^*(\log n)$ if $n > 1$

EXAMPLE OF LOG* VALUES

- Log* (10000)
- 1 + log* 4
- 2 + log* 0.6
- 2

- log*(10^(10^10000))
- = 1 + log*(log(10^10^10000))
- = 1 + log*(10^10000)
- = 1 + 1 + log*(log(10^10000))
- = 2 + log*(10000)
- 3 + log*(4)
- = 4

TIME COMPLEXITY ANALYSIS OF KRUSKAL'S ALGORITHM

- Using 2 Find operations to check if adding an edge will create a cycle or not.
- When adding an edge, use a Union Operation

WHY DOES KRUSKAL'S ALGORITHM WORK?

- Proof by contradiction
- Must practice the writing of this.

TWO BASIC PROPERTIES OF OPTIMAL GREEDY ALGORITHMS

- **Optimal Substructure Property**: A problem has optimal substructure if an optimal solution to the problem contains within it, optimal solutions to its sub problems.
- **Greedy Choice Property**: If a local greedy choice is made, then an optimal solution including this choice is possible.

GREEDY ALGORITHMS AND MATROIDS

- A <u>**subset system**</u> is a set E together with a set of subsets of *E*, called *I*, such that *I* is closed under inclusion. This means that if $X \subseteq Y$ and $Y \in I$, then $X \in I$. (*I* is sometimes referred to as set of independent sets.)
  The "Hereditary Property". Subset of a valid solution, is valid.

- A subset system is a <u>**matroid**</u> if it satisfies the exchange property: If i_1 and i_2 are sets in *I* and i_1 has fewer elements than i_2, then there exists an element $e \in i_2 \setminus i_1$ such that $i_1 \cup \{e\} \in I$.
  The augmentation property or the independent set exchange property. If a larger solution exists, we should be able to add <u>something</u> to the current solution. ("Build solution one step at a time.")

- For any subset system *(E,I)*, the greedy algorithm solves the optimization problem for *(E,I)* if and only if *(E,I)* is a matroid.

AN EXAMPLE OF MATROID

- Consider the set of edges of a graph, and set of "forests" (forest is a set of edges that doesn't have a cycle)
- Subset of that "forest" is also a "forest". This satisfies the hereditary property. So, this is a subset system.
- Consider forest f1, and forest f2. If f1 has less edges than f2, then you can certainly add an edge from f2 to f1 such that f1' will still be a forest.
- **So, the system of forests is a matroid.**

The set of forests in a graph forms a **matroid**. It is known as the **graphic matroid**.

AN EXAMPLE OF A "NON-MATROID"

- Consider a graph, and the set of "cliques" (a clique is a set of vertices that are all connected to each other)
- A sub-set of clique is also a clique.
- So, clique is a subset system.
- Given a clique K1 and a clique K2, suppose K2 has more vertices than K1. It is NOT guaranteed that we can add a vertex from K2 to K1 and keep the K1' as a clique.
- Therefore, the clique system is not a matroid.

WHEN NOT TO USE GREEDY ALGORITHM

- Prone to overuse
  - You shouldn't use this algorithm unless you can prove that the solution is optimal.
  - That is, no points in MT/Final for using greedy algorithm to produce a suboptimal solution, where another algorithmic technique (such as D&C) would have resulted in an optimal solution.
- Why?
  - Optimality has a "business value". Suppose you are trying to maximize the flights that you can schedule using 3 aircrafts.
  - Time complexity merely represents a "cost of computation" of that schedule.
  - If one algorithm runs in 1 minute, but schedules only 7 flights, and another algorithm runs in 2 hours, but schedules 8 flights, which one would you use?

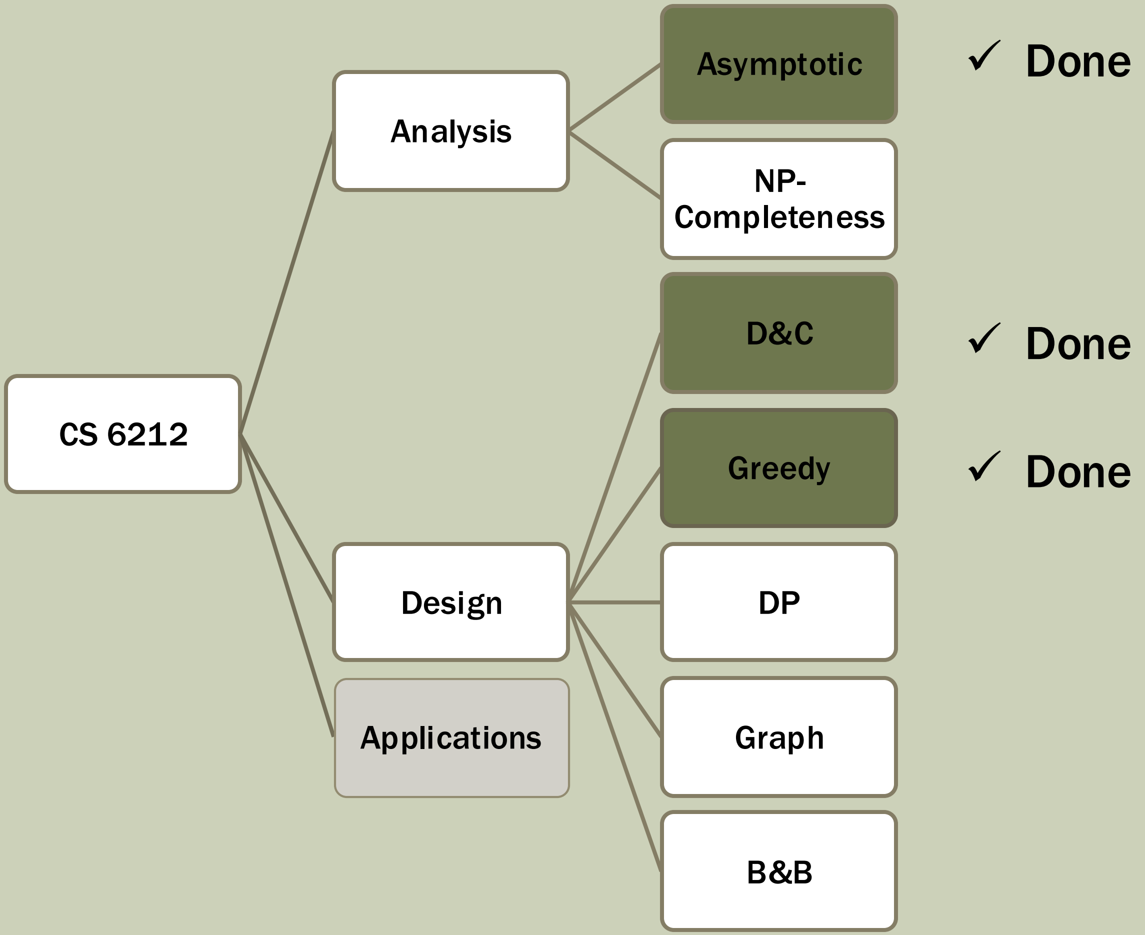
MORE GREEDY ALGORITHM PROBLEMS

- Symbol Encoding
- Interval Scheduling

GREEDY: TO APPLY OR NOT TO APPLY

- Chess
- Sorting
- Shortest path computation
- Knapsack

WHERE WE ARE


CS 6212
Analysis
Design
Applications
Asymptotic
NP-Completeness
D&C
Greedy
DP
Graph
B&B
✓ Done
✓ Done
✓ Done

READING ASSIGNMENT

Application # 5

- Greedy
  - Book – first problem on interval scheduling classes
  - http://en.wikipedia.org/wiki/Huffman_coding
  - http://www.cs.kent.edu/~dragan/AdvAlg05/GreedyAlg-1x1.pdf
- Dynamic Programming
  - Dynamic Programming: Book sections 6.1 – 6.4
  - http://www.yaroslavvb.com/papers/wagner-dynamic.pdf