Greedy Algorithms

- Given a set of activities (s_i, f_i), we want to schedule the maximum number of non-overlapping activities.
- GREEDY-ACTIVITY-SELECTOR (s, f)

1. n = length[s]
2. S = {a_1}
3. i = 1
4. **for** m = 2 **to** n **do**
5. **if** s_m is not before f_i **then**
6. S = S U {a_m}
7. i = m
8. return S

Example

- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14] -- Sorted by finish times
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
- [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]

Why does it work?

- THEOREM

  Let A be a set of activities and let a_1 be the activity with the earliest finish time. Then activity a_1 is in some maximum-sized subset of non-overlapping activities.
- PROOF

  Let S' be a solution that does not contain a_1. Let a'_1 be the activity with the earliest finish time in S'. Then replacing a'_1 by a_1 gives a solution S of the same size.

  Why are we allowed to replace? Why is it of the same size?

Then apply induction! How?

Greedy Algorithms – Huffman Coding

- Huffman Coding Problem
  Example: Release 29.1 of 15-Feb-2005 of TrEMBL Protein Database contains **1,614,107** sequence entries, comprising **505,947,503** amino acids. There are 20 possible amino acids. What is the minimum number of bits to store the compressed database?
  ~2.5 G bits or 300MB.
- How to improve this?
- Information: Frequencies are not the same.

Ala (A) 7.72	Gln (Q) 3.91	Leu (L) 9.56	Ser (S) 6.98
Arg (R) 5.24	Glu (E) 6.54	Lys (K) 5.96	Thr (T) 5.52
Asn (N) 4.28	Gly (G) 6.90	Met (M) 2.36	Trp (W) 1.18
Asp (D) 5.28	His (H) 2.26	Phe (F) 4.06	Tyr (Y) 3.13
Cys (C) 1.60	Ile (I) 5.88	Pro (P) 4.87	Val (V) 6.66

- **Idea:** Use shorter codes for more frequent amino acids and longer codes for less frequent ones.

Huffman Coding

2 million characters in file.
A, C, G, T, N, Y, R, S, M

IDEA 1: Use ASCII Code
Each need at least 8 bits,
Total = 16 M bits = 2 MB

IDEA 2: Use 4-bit Codes
Each need at least 4 bits,
Total = 8 M bits = 1 MB

Percentage Frequencies

IDEA 3: Use Variable Length Codes

A	22	11
T	22	10
C	18	011
G	18	010
N	10	001
Y	5	00011
R	4	00010
S	4	00001
M	3	00000

How to Decode?
Need Unique decoding!
Easy for Ideas 1 & 2.
What about Idea 3?

11010110111001000110000000110

11010110111001000110000000110

2 million characters in file.
Length = ?
Expected length = ?
Sum up products of frequency times the code length, i.e.,
(.22x2 + .22x2 + .18x3 + .18x3 + .10x3 + .05x5 + .04x5 + .04x5 + .03x5) x 2 M bits =
3.24 M bits = .4 MB

Dynamic Programming

- Activity Problem Revisited: Given a set of activities (s_i, f_i), we want to schedule the maximum number of non-overlapping activities.
- New Approach:
  A_i = Best solution for intervals $\{a_1, \ldots, a_i\}$ that includes interval a_i
  B_i = Best solution for intervals $\{a_1, \ldots, a_i\}$ that does not include interval a_i
- Does it solve the problem to compute A_i and B_i?
- How to compute A_i and B_i?

Dynamic Programming

- Activity Problem Revisited: Given a set of n activities $a_i = (s_i, f_i)$, we want to schedule the maximum number of non-overlapping activities.
- New Approach:
  - Observation: To solve the problem on activities $A_n = \{a_1,\ldots,a_n\}$, we notice that either
    - optimal solution does not include a_n (Problem on A_{n-1})
    - optimal solution includes a_n (Problem on A_k, which is equal to A_n without activities that overlap a_n, I.e., a_k is the last activity that finishes before a_n starts.)

COT 5407 10/18/05 7

An efficient implementation

- Why not solve the problem on $A_1,\ldots,A_{n-1},A_n$?
- In what order to solve them?
- Is the problem on A_1 easy?
  - YES, trivial
- Can the optimal solutions to the problems on $A_1,\ldots,A_i$ help to solve the problem on A_{i+1}?
  - YES! Either:
    - optimal solution does not include a_{i+1} (Problem on A_i)
    - optimal solution includes a_{i+1} (you are left with a problem on A_k, which is equal to A_i without activities that overlap a_{i+1}, i.e., a_k is the last activity that finishes before a_{i+1} starts.)

COT 5407 10/18/05 8

Dynamic Programmming: Activity Selection

- Select the maximum number of non-overlapping activities from a set of n activities $A = \{a_1, \ldots, a_n\}$ (sorted by finish times).
- Identify "easier" subproblems to solve.

  $A_1 = \{a_1\}$

  $A_2 = \{a_1, a_2\}$

  $A_3 = \{a_1, a_2, a_3\}, \ldots,$

  $A_n = A$
- Subproblems: Select the max number of non-overlapping activities from A_i

COT 5407 10/18/05 9

Dynamic Programmming: Activity Selection

- Solving for A_n solves the original problem.
- Solving for A_1 is easy.
- If you have optimal solutions $S_1, \ldots, S_{i-1}$ for subproblems on $A_1, \ldots, A_{i-1}$, how to compute S_i?
- The optimal solution for A_i either
  - Case1: does not include a_i or
  - Case 2: includes a_i
- Case 1:
  - $S_i = S_{i-1}$
- Case 2:
  - $S_i = S_k \cup \{a_i\}$, for some $k < i$.
  - How to find such a k? We know that a_k cannot overlap a_i.

Dynamic Programmming: Activity Selection

- DP-ACTIVITY-SELECTOR (s, f)

```
1. n = length[s]
2.N[1] = 1        // number of activities in S1
3.F[1] = 1        // last activity in S1
4.for i = 2 to n do
5.   let k be the last activity finished before si
6.   if (N[i-1] > N[k]) then   // Case 1
7.          N[i] = N[i-1]
8.          F[i] = F[i-1]
9.   else  // Case 2
10.         N[i] = N[k] + 1
11.         F[i] = i
```

How to output S_n?
Backtrack!
Time Complexity?
O(n lg n)

Dynamic Programming Features

- Identification of subproblems
- Recurrence relation for solution of subproblems
- Overlapping subproblems (sometimes)
- Identification of a hierarchy/ordering of subproblems
- Use of table to store solutions of subproblems (MEMOIZATION)
- Optimal Substructure