CSE 417 Algorithms and Complexity
Lecture 18, Winter 2023
Dynamic Programming

Announcements
• Dynamic Programming Reading:
  – 6.1-6.2, Weighted Interval Scheduling
  – 6.4 Knapsack and Subset Sum
  – 6.6 String Alignment
    • 6.7* String Alignment in linear space
  – 6.8 Shortest Paths (again)
  – 6.9 Negative cost cycles
• How to make an infinite amount of money
• Homework 7

Dynamic Programming
• The most important algorithmic technique covered in CSE 417
• Key ideas
  – Express solution in terms of a polynomial number of sub problems
  – Order sub problems to avoid recomputation

Recursion vs Iteration

Counting Rabbits
0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, . . .

F₀ = 0; F₁ = 1; Fₙ = Fₙ₋₁ + Fₙ₋₂

Fib(n){
  if (n <= 1)
    return n;
  else
    return Fib(n-1) + Fib(n-2);
}
Fibonacci with Memoization

```java
Fib(n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return Fib(n-1) + Fib(n-2);
}
```

Reordering computation

```java
Fib(n) {
    int[] F = new int[n+1];
    F[0] = 0;
    F[1] = 1;
    for (i = 2; i <= n; i++)
        F[i] = F[i-1] + F[i-2];
    return F[n];
}
```

Dynamic Programming

- Weighted Interval Scheduling
- Given a collection of intervals I₁,…,Iₙ with weights w₁,…,wₙ, choose a maximum weight set of non-overlapping intervals

```
<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
```

Optimality Condition

- Opt[ j ] is the maximum weight independent set of intervals I₁, I₂, …, Iᵢ
  - Where p[ j ] is the index of the last interval which finishes before Iᵢ starts

```
Algorithm
```
MaxValue(j) =
    if j = 0 return 0
    else
        return max ( MaxValue(j-1), wᵢ + MaxValue(p[ j ]))
```

Worst case run time: 2ⁿ

```
A better algorithm
```
M[ j ] initialized to -1 before the first recursive call for all j

```
MaxValue(j) =
    if j = 0 return 0;
    else if M[ j ] != -1 return M[ j ];
    else
        M[ j ] = max (MaxValue(j-1), wᵢ + MaxValue(p[ j ]));
        return M[ j ];
```
Iterative Algorithm

MaxValue(n)
    int[] M = new int[n+1];
    M[0] = 0;
    for (int i = 1; i <= n; i++){
        M[ j ] = max(M[j-1], wj + M[p[ j ]]);
    }
    return M[n];

Algorithm Summary

- O(n) time algorithm for finding maximum weight independent set of intervals
- Key idea: Creating an Opt function to express optimal set of I_1, I_2, ..., I_k in terms of optimal set of I_1, I_2, ..., I_{k-1}
- Organize computation to avoid recomputation

Fill in the array with the Opt values


Computing the solution

Record which case is used in Opt computation