Dynamic programming

9/27/24

Administrivia

- HW 3 (multithreading) due Tuesday 10/1
- Exam 1 out Wednesday
  - Multi-day takehome
  - Open notes and book, closed internet and friends
  - No class next Thursday (10/3)
  - Due early the next week (probably Monday night)
  - Everything thru multithreaded (induction, asymptotic ordering, AVL trees, D&C, multithreaded)

How **not** to compute Fibonacci numbers

```
int fib(int n) {
    if(n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

Instead, solve from small cases

```
int fib(int n) {
    int array[] = new int[n+1];

    array[0] = 0;
    array[1] = 1;
    for(int i=2; i<=n; i++)
        array[i] = array[i-1] + array[i-2];

    return array[n];
}
```

Dynamic programming

- Divide & conquer, but storing subproblem values to avoid recomputing them

- In this context, “programming” = “filling in table”

Properties of dynamic programming problems

- "Optimal substructure"
  - optimal solutions build on optimal solutions to subproblems
- "Overlapping subproblems"
  - subproblems appear many times in a purely recursive solution

Memoization: store subproblem solutions

```
//map to store values we've already computed:
Map<Integer,Integer> memos;  //memos.get(i) is F_i

int fib(int n) {
    if(n <= 1)
        return n;
    if(!memos.containsKey(n))
        memos.put(n, fib(n-1) + fib(n-2));
    return memos.get(n);
}
```

Which approach to use?

- "bottom up": loop-based approach
  - need to find nice order to fill in the table
  - cleaner code since don't need to see if values are already computed
- "top down": memoization
  - may not need to fill in the entire table

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Proposal: Take as many of the largest value size as possible

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Proposal: Take as many of the largest value size as possible

Does this work?

A. Yes

B. No

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Proposal: Take as many of the largest value size as possible

Doesn't work: Length 4 with table above

(2+2 is better than 4)

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Proposal: Take as many as possible of the size with most value per unit length (highest “value density”)

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Proposal: Take as many as possible of the size with most value per unit length (highest “value density”)

Does this work?

A. Yes

B. No

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- Proposal: Take as many as possible of the size with most value per unit length (highest “value density”)

Doesn’t work: Length 4 with

length i	1	2	3
price p_i	1	5	8

Rod cutting

- Given a rod of length n and prices of pieces of different length, what is the best possible total value?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- How can I phrase this recursively?

```
int best_value(int[] p, int n) {
    ...
}
```

Recursive version

```
int best_value(int[] p, int n) {
        if(n == 0)
                return 0
        q = - ∞                 //best value so far
        for i = 1 to n
                q = max(q, p[i] + best_value(p, n-i))
        return q
}
```

Bottom up version

```
int best_value(int[] p, int n) {
        allocate r[0..n]
        r[0] = 0

        for j = 1 to n {
                q = -∞          //best value so far
                for i = 1 to j
                        q = max(q, p[i]+r[j-i])
                r[j] = q
        }
        return r[n]
}
```

Bottom up version

```
int best_value(int[] p, int n) {
        allocate r[0..n]
        r[0] = 0
        for j = 1 to n {
                q = -∞          //best value so far
                for i = 1 to j
                        q = max(q, p[i]+r[j-i])
                r[j] = q
        }
        return r[n]
}
```

How do I get the actual optimal cuts?

Memoization version: setup code

```
int best_value(int[] p, int n) {
        allocate r[0..n]
        r[0] = 0
        for i = 1 to n
                r[i] = -∞

        return best_value_aux(p, n, r)
}
```

Memoized version: main code

```
int best_value_aux(int[] p, int n, int[] r) {
        if(r[n] >= 0)
                return r[n]

        q = - ∞
        for i = 1 to n
                q = max(q, p[i] + best_value_aux(p, n-i, r))
        r[n] = q
        return q
}
```

Rod cutting with cutting cost

Suppose the cost to make a cut is C and your goal is to maximize profit (i.e. revenue minus cutting cost).

```
int max_profit(int[] p, int C, int n) {
```

Rod cutting with cutting cost

Suppose the cost to make a cut is C and your goal is to maximize profit (i.e. revenue minus cutting cost).

int max_profit(int[] p, int C, int n) {

a) Show that maximizing the revenue does not necessarily maximize the profit

Rod cutting with cutting cost

Suppose the cost to make a cut is C and your goal is to maximize profit (i.e. revenue minus cutting cost).

int max_profit(int[] p, int C, int n) {

b) Give an efficient dynamic programming algorithm to maximize profit

Commercial purchasing

Want to purchase a subset of commercial slots available during an online show. You have a sorted list of their locations (x_1, x_2, …, x_n) from the beginning of the show.

- Buying slot x_i brings in revenue r_i
- You don't want to buy commercials starting $\leqslant$2 minutes apart

How do you maximize revenue?