Lecture 21: Dynamic Programming III: Text Justification, Parenthesization, Knapsack, Pseudopolynomial Time, Tetris Training

Lecture Overview

- Text Justification
- Parenthesization
- Knapsack
- Pseudopolynomial Time
- Tetris Training

Readings

CLRS 15

Review:

* DP is all about subproblems & guessing

* 5 easy steps:
  (a) define subproblems: count \# subprobs.
  (b) guess (part of solution): count \# choices
  (c) relate subprob. solutions: compute time/subprob.
  (d) recurse + memoize OR build DP table bottom up:
      time = time/subprob. \times \# subprobs
      (check subproblems related acyclically)
  (e) check original problem = a subproblem or solvable from DP table (\implies extra time)

* for sequences, good subproblems are often prefixes OR suffixes OR substrings
Text Justification:
Split text into “good lines”

- obvious (MS Word/Open Office) algorithm: put as many words fit on first line, repeat
- but this can make very bad lines

\[
\begin{align*}
\text{blah blah blah} & \quad \text{blah} \quad \text{blah} \\
\text{\texttt{b l a h}} & \quad \text{vs.} \quad \text{blah} \quad \text{blah} \\
\text{reallylongword} & \quad \text{reallylongword}
\end{align*}
\]

Figure 1: Good vs. Bad Justification

- define \text{badness}(i, j) for line of words \([i : j]\) e.g.,

\[
\begin{align*}
\text{if total length} & > \text{page width} \\
\text{(page width - total length)}^3 & \text{else}
\end{align*}
\]

- goal: split words into lines to \(\min \sum \text{badness}\)

1. subproblem = \(\min \text{badness} \text{ for suffix words} [i : ]\)
   \(\implies \frac{n}{2} \text{ subproblems} = \Theta(n) \text{ where } n = \frac{n}{2} \text{ words}\)

2. guessing = where to end first line, say \(i : j\)
   \(\implies \frac{n}{2} \text{ choices} = n - i = O(n)\)

3. relation:
   - \(\text{DP}[i] = \min (\text{badness}(i, j) + \text{DP}[j] \text{ for } j \text{ in range}(i + 1, n + 1))\)
   - \(\text{DP}[n] = \phi \implies \text{time per subproblem} = O(n)\)

4. total time = \(O(n^2)\)

5. solution = \(\text{DP}[\phi]\)
   \((& \text{ use parent pointers to recover split})\)
Parenthesization:
Optimal evaluation of associative expression - e.g., multiplying rectangular matrices

\[
\begin{align*}
\text{(AB)C} & \text{ costs } \theta(n^2) \\
\text{A(BC)} & \text{ costs } \theta(n)
\end{align*}
\]

Figure 2: Evaluation of an Expression

2. guessing = outermost multiplication \((\cdots)(\cdots)\)
   \[1_{k=1} \cdots 1_k\]
   \[\Rightarrow \# \text{ choices } = O(n)\]
1. subproblems = prefix & suffixes? NO
   cost of substring \(A[i : j]\)
   \[\Rightarrow \# \text{ subproblems } = \Theta(n^2)\]

3. Relation:
   - \(\text{DP}[i,j] = \min(\text{DP}[i,k] + \text{DP}[k,j] + \text{cost of multiplying } (A[i] \cdots A[k-1]) \text{ by } (A[k] \cdots A[j-1]))\) for \(k \in \text{range}(i+1, j)\)
   - \(\text{DP}[i, i+1] = \phi\)
   \[\Rightarrow \text{cost per subproblem } = O(n)\]

4. total time = \(O(n^3)\)

5. solution = \(\text{DP}[0,n]\)
   (& use parent pointers to recover paren.)

Knapsack:
Knapsack of size \(S\) you want to pack
- item \(i\) has integer size \(s_i\) & real value \(v_i\)
- goal: choose subset of items of maximum total value subject to total size \(\leq S\)

First Attempt:
1. subproblem = value for suffix \(i\): \textbf{WRONG}
2. guessing = whether to include item \(i\) \[\Rightarrow \# \text{ choices } = 2\]
3. relation:
• $DP[i] = \max(DP[i+1], v_i + DP[i+1] \text{ if } s_i \leq S^\prime)$

• not enough information to know whether item $i$ fits - how much space is left? GUESS!

1. subproblem = value for suffix $i$:
   given knapsack of size $X$
   $\Rightarrow$ # subproblems = $O(nS)$

3. relation:
   • $DP[i, X] = \max(DP[i+1, X], v_i + DP[i+1, X - s_i] \text{ if } s_i \leq X)$
   • $DP[n, X] = \phi$
     $\Rightarrow$ time per subproblem = $O(1)$

4. total time = $O(nS)$

5. solution = $DP[\phi, S]$
   (& use parent pointers to recover subset)
   AMAZING: effectively trying all possible subsets!

Knapsack is in fact NP-complete! $\Rightarrow$ suspect no polynomial-time algorithm (polynomial in length of input).

What gives?

• here input = $< S, s_0, \cdots, s_{n-1}, v_0, \cdots, v_{n-1} >$

• length in binary: $O(\lg S + \lg s_0 + \cdots) \approx O(n \lg \ldots)$

• so $O(nS)$ is not “polynomial-time”

• $O(nS)$ still pretty good if $S$ is small

• “pseudopolynomial time”: polynomial in length of input & integers in the input

Remember:

polynomial - GOOD
exponential - BAD
pseudopoly - SO SO
Tetris Training:

- given sequence of $n$ Tetris pieces & a board of small width $w$
- must choose orientation & $x$ coordinate for each
- then must drop piece till it hits something
- full rows do not clear
  without these artificialities WE DON'T KNOW! (but: if $w$ large then NP-complete)
- goal: survive i.e., stay within height $h$

[material below covered in recitation]

First Attempt:

1. subproblem = survive in suffix $i$? **WRONG**
2. guessing = how to drop piece $i$ $\implies \exists$ choices = $O(w)$
3. relation: $DP[i] = DP[i + 1]$ ?! not enough information!
   What do we need to know about prefix $i$?

   1. subproblem = survive? in suffix $i$:
      given initial column occupancies $h_0, h_1, \ldots, h_{w-1}$
      $\implies \exists$ subproblems = $O(n \cdot h^w)$
   3. relation: $DP[i, h] = \max(DP[i, m]$ for valid moves $m$ of piece $i$ in $h)$
      $\implies$ time per subproblem = $O(w)$
   4. total time = $O(nwh^w)$
   5. solution = $DP[\emptyset, \emptyset]$
      (& use parent pointers to recover moves)