CSE 421
Algorithms
Richard Anderson
Lecture 18
Dynamic Programming

Dynamic Programming
• The most important algorithmic technique covered in CSE 421
• Key ideas
  – Express solution in terms of a polynomial number of sub problems
  – Order sub problems to avoid recomputation

Today - Examples
• Examples
  – Optimal Billboard Placement
    • Text, Solved Exercise, Pg 307
  – Linebreaking with hyphenation
    • Compare with HW problem 6, Pg 317
  – String approximation
    • Text, Solved Exercise, Page 309

Billboard Placement
• Maximize income in placing billboards
  – \( b_i = (p_i, v_i) \); value of placing billboard at position \( p_i \)
• Constraint:
  – At most one billboard every five miles
• Example
  – \{\{(6,5), (8,6), (12, 5), (14, 1)\}\}

Design a Dynamic Programming Algorithm for Billboard Placement
• Compute Opt[1], Opt[2], …, Opt[n]
• What is Opt[k]?

Opt[k] = fun(Opt[0], …, Opt[k-1])
• How is the solution determined from sub problems?

Input \( b_i, …, b_n \), where \( b_i = (p_i, v_i) \), position and value of billboard i
Solution

\[
j = 0; \quad // j \text{ is five miles behind the current position}
\]

\[
\text{for } k := 1 \text{ to } n
\]

\[
\text{while } (P[j] < P[k] - 5)
\]

\[
j := j + 1;
j := j - 1;
\]

\[
\text{Opt}[k] = \text{Max}(\text{Opt}[k-1], V[k] + \text{Opt}[j]);
\]

Optimal line breaking and hyphenation

- Problem: break lines and insert hyphens to make lines as balanced as possible
- Typographical considerations:
  - Avoid excessive white space
  - Limit number of hyphens
  - Avoid widows and orphans
  - Etc.

Penalty Function

- Pen(i, j) – penalty of starting a line at position i, and ending at position j

Optimal line breaking and hyphenation is computed with dynamic programming

- Key technical idea
  - Number the breaks between words/syllables

Opt[k] = fun(Opt[0],...,Opt[k-1])

- How is the solution determined from subproblems?

Solution

\[
\text{for } k := 1 \text{ to } n
\]

\[
\text{Opt}[k] := \text{infinity};
\]

\[
\text{for } j := 0 \text{ to } k-1
\]

\[
\text{Opt}[k] := \text{Min}(\text{Opt}[k], \text{Opt}[j] + \text{Pen}(j,k));
\]
But what if you want to layout the text?
• And not just know the minimum penalty?

Solution
for k := 1 to n
Opt[k] := infinity;
for j := 0 to k-1
temp := Opt[j] + Pen(j, k);
if (temp < Opt[k])
   Opt[k]  = temp;
   Best[k] := j;

String approximation
• Given a string S, and a library of strings B = {b₁, …, bₘ}, construct an approximation of the string S by using copies of strings in B.

B = {abab, bbbaaa, ccb, ccaacc}
S = abacbbbaabbcbbccaabab

Formal Model
• Strings from B assigned to non-overlapping positions of S
• Strings from B may be used multiple times
• Cost of δ for unmatched character in S
• Cost of γ for mismatched character in S
  – MisMatch(i, j) – number of mismatched characters of bᵢ when aligned starting with position i in S.

Design a Dynamic Programming Algorithm for String Approximation
• Compute Opt[1], Opt[2], . . . , Opt[n]
• What is Opt[k]?

Opt[k] = fun(Opt[0],…,Opt[k-1])
• How is the solution determined from sub problems?

Target string S = s₁s₂…sₙ
Library of strings B = {b₁, …, bₘ}
MisMatch(i, j) = number of mismatched characters with bᵢ when aligned starting at position i of S.
Solution

for i := 1 to n
    Opt[k] = Opt[k-1] + \delta;
for j := 1 to |B|
    p = i – len(bj);
    Opt[k] = min(Opt[k], Opt[p-1] + \gamma MisMatch(p, j));