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 concatenation
    - Text, Solved Exercise, Page 309

Announcements

- Wednesday class will meet in CSE 305.
Billboard Placement

• Maximize income in placing billboards
  \((p_i, v_i), \ v_i: \text{value of placing billboard at position} \ p_i\)
• Constraint:
  – At most one billboard every five miles
• Example
  – \{(6,5), (7,6), (12, 5), (14, 1)\}

Opt\([k]\)

• What are the sub problems?

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

• How is the solution determined from sub problems?

Solution

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

// the last valid location for a billboard, if one placed at \(P[k]\)

\[
\text{for } k := 1 \text{ to } n \\
\text{ while } (P[j] < P[k] \ - \ 5) \\
\quad j := j + 1; \\
\quad j := j - 1; \\
\quad \text{Opt}[k] = \text{Max}(\text{Opt}[k-1], v[k] + \text{Opt}[j]);
\]

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 a 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]
• What are the sub problems?

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

Solution
for k := 1 to n
    Opt[k] := infinity;
    for j := 1 to n-1
        Opt[k] := Min(Opt[k], Opt[j] + 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 := 1 to n-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 = \{b_1, …, b_n\}, construct an approximation of the string S by using copies of strings in B.

B = \{abab, bbbaaa, ccbb, ccaacc\}
S = abacccbbbaabbcbbccaaabab
Formal Model

• Strings from B assigned to non-overlapping positions of s
• Strings from B may be used multiple times
• Cost of $\delta$ for unmatched character in s
• Cost of $\gamma$ for mismatched character in s
  – $\text{MisMatch}(i, j)$ – number of mismatched characters of $b_j$, when aligned starting with position $i$ in $s$.

Opt[$k$]

• What are the sub problems?

Opt[$k$] = fun(Opt[0],…,Opt[$k$-1])

• How is the solution determined from sub problems?

for $i := 1$ to $n$
    Opt[$i$] = Opt[$i$-1] + $\delta$;
    for $j := 1$ to $|B|$;
    $p = i - \text{len}(b_j);$
    Opt[$i$] = min(Opt[$i$], Opt[$p-1$] + $\gamma$ MisMatch($p$, $j$));

Solution