String reconstruction

Computed document: punctuation missing.
 shortage of words
 We are interested in S(n,m).

STEP 1: define subproblem
S(0,j) = true if x[1..j] is a valid sequence
false otherwise
We are interested in S(n,0).

STEP 2: express recursively
S(i,j) = false if x[i..j] is not a valid sequence
S(i-1,j) or S(i,j-1)
For consistency define S(i,0) = true.

STEP 3: order of solving subproblems
S(n), then S(n-1), then S(n-2), etc.

Longest common subsequence

Two sequences:
ACTG GCTAG
GTCAG T

What is their longest common subsequence?

STEP 1: define subproblem
S(i,j) = LCS of x[1..i] and y[1..j]
We are interested in S(n,m).

STEP 2: express recursively
S(i,j) = S(i-1,j-1) + 1 if x[i] = y[j]
m = S(i-1,j), S(i,j-1) otherwise

STEP 3: order of solving subproblems
Column by column, top to bottom.

Achieving efficiency via dynamic programming vs divide and conquer

Divide-and-conquer
A problem of size n is decomposed into a few subproblems which are
significantly smaller (e.g. n/2, n/4, ...)
Therefore, size of subproblems
decreases geometrically.
Use a recursive algorithm.
Avoid recursion and instead solve the
subproblems one-by-one, saving the
answers in a table, in a clever explicit order.

Dynamic programming
A problem of size n is expressed in
terms of subproblems that are not
much smaller (e.g. n-1, n-2, ...)
A recursive algorithm would take
exponential time.
Saving grace: in total, there are only
polynomially many subproblems.

Divide-and-conquer
Dynamic programming
In order from 1 to n.

STEP 3: order of subproblems

L(j) = cost of words through j in optimal solution

STEP 2: express recursively

We are interested in S(i,n).

STEP 1: define subproblem

Given sequence x[1..n], what is their longest common substring?

Two strings:

A G C T G A C C T G A C
G C A T C A C T G A C

What is their longest common substring?

[Steps to be consecutively, unlike subsequence]

STEP 1: define subproblem

S(i,j) = LCS of x[1..j] and y[1..j]

Two strings:

A G C T G A C C T G A C
Y 1 2 3 4 5 6 7 8 9 10

What is the length of their longest common substring?

[Steps to be consecutively, unlike subsequence]

STEP 1: define subproblem

S(i,j) = LCS of x[1..j] and y[1..j]

Given sequences x[1..n] and y[1..m], what is the length of their longest common substring?

[Steps to be consecutively, unlike subsequence]

STEP 1: define subproblem

S(i,j) = LCS of x[1..j] and y[1..j]

Given sequence x[1..n], what is the length of its LIS?

STEP 1: define subproblem

L(i) = LIS of x[1..i]

We are interested in max(L(i)).

STEP 2: express recursively

L(i+1) = max{L(i) + 1, if x[i+1] ≥ x[i] else S(i+1, j)}

STEP 3: order of subproblems

In order from 1 to n.

6. Expected number of comparisons

= (0.04) x 5 + (0.40 + 0.10) x 2 + (0.05 + 0.08 + 0.10 + 0.23) x 3 = 2.42

Column by column, top to bottom

BEGIN
DO
WHILE
END

Expected number of comparisons

= (0.04) x 5 + (0.40 + 0.10) x 2 + (0.05 + 0.08 + 0.10 + 0.23) x 3 = 2.42

BEGIN
DO
WHILE
END

Keywords frequencies:

BEGIN
DO
ELSE
IF
THEN
WHILE
END

Frequency of keywords:

BEGIN
DO
ELSE
IF
THEN
WHILE
END

Input: a list of positive integers, a[1..n]; integer t

Question: is there some subset of the a[ ] that adds up to t?

While n is to 0 do {
\[ (0.04) x 5 + (0.40 + 0.10) x 2 + (0.05 + 0.08 + 0.10 + 0.23) x 3 = 2.42 \]

Input: a list of positive integers, a[1..n]; integer t

Question: is there some subset of the a[ ] that adds up to t?

While n is to 0 do {
\[ (0.04) x 5 + (0.40 + 0.10) x 2 + (0.05 + 0.08 + 0.10 + 0.23) x 3 = 2.42 \]

Input: a list of positive integers, a[1..n]; integer t

Question: is there some subset of the a[ ] that adds up to t?

While n is to 0 do {
\[ (0.04) x 5 + (0.40 + 0.10) x 2 + (0.05 + 0.08 + 0.10 + 0.23) x 3 = 2.42 \]

Input: a list of positive integers, a[1..n]; integer t

Question: is there some subset of the a[ ] that adds up to t?

While n is to 0 do {
\[ (0.04) x 5 + (0.40 + 0.10) x 2 + (0.05 + 0.08 + 0.10 + 0.23) x 3 = 2.42 \]
Longest common substring

Given sequences $x[1..n]$ and $y[1..m]$, what is their longest common substring?

**STEP 1: define subproblem**
$S(i,j) =$ longest common substring of $x[1..i]$ and $y[1..j]$, ending at $x[i]$ and $y[j]$
We are interested in $\max_{i,j} S(i,j)$.

**STEP 2: express recursively**
$S(i,j) = S(i-1,j-1) + 1$ if $x[i] = y[j]$
0 otherwise

**STEP 3: order of subproblems**
Column by column, top to bottom

```
for i = 0 to n: S[i,0] = 0
for j = 0 to m: S[0,j] = 0
for i = 1 to n:
    for j = 1 to m:
        if $x[i] = y[j]$: $S[i,j] = S[i-1,j-1] + 1$
        else: $S[i,j] = 0$
return $\max_{i,j} S[i,j]$
```

Independent set

A subset of vertices $S \subseteq V$ is an independent set of $G$ if there are no edges between them.

**Input:** Graph $G = (V,E)$

**Output:** An independent set $S \subseteq V$ of maximal size

This is a very hard problem!
Instead, consider a restricted version, in which the graph $G$ is a tree.

```
\begin{center}
\begin{tikzpicture}
    \node at (0,0) (a) {a};
    \node at (1,0) (b) {b};
    \node at (2,0) (c) {c};
    \node at (3,0) (d) {d};
    \node at (4,0) (e) {e};
    \node at (5,0) (f) {f};
    \node at (0,-1) (g) {g};
    \node at (1,-1) (h) {h};
    \draw (a) -- (b);
    \draw (a) -- (g);
    \draw (b) -- (c);
    \draw (c) -- (d);
    \draw (d) -- (e);
    \draw (e) -- (f);
    \draw (g) -- (h);
\end{tikzpicture}
\end{center}
```

All-pairs shortest paths

Given $n$ nodes and distances $d_{ij}$ (which could be any integer or $\infty$) on all edges, find shortest path distances between all pairs of nodes.

**STEP 1: define subproblem**
$D(i,j,k) =$ length of shortest path from $i$ to $j$ with intermediate nodes in $\{1,2,...,k\}$
We are interested in $D(i,j)$.

**STEP 2: express recursively**
$D(i,j,k) = \min\{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)\}$

**STEP 3: order of subproblems**
By increasing $k$.

```
for i,j = 1 to n:
    dist[i,j] = $d_{ij}$
for k = 1 to n:
    for i,j = 1 to n:
        dist[i,j] = $\min\{dist[i,j], dist[i,k] + dist[k,j]\}$
```