Lecture #35

gramming and memoization.

Git.

Dynamic Programming

Garcia):

n a list with an even number of non-negative integers.

er in turn takes either the leftmost number or the .

get the largest possible sum.

rting with (6, 12, 0, 8), you (as first player) should take ever the second player takes, you also get the 12, for a

ur opponent plays perfectly (i.e., to get as much as pos- an you maximize your sum?

s with exhaustive game-tree search.

Obvious Program

akes it easy, again:

```
n(int[] V) {
l, i, N = V.length;
 0, total = 0; i < N; i += 1) total += V[i];
estSum(V, 0, N-1, total);


rgest sum obtainable by the first player in the choosing
n the list V[LEFT .. RIGHT], assuming that TOTAL is the
 all the elements in V[LEFT .. RIGHT].  */
n(int[] V, int left, int right, int total) {
 > right)
 0;

= total - bestSum(V, left+1, right, total-V[left]);
= total - bestSum(V, left, right-1, total-V[right]);
 Math.max(L, R);
```

$C(0) = 1,\ C(N) = 2C(N-1)$; so $C(N) \in \Theta(2^N)$

Still Another Idea from CS61A

is that we are recomputing intermediate results many

emoize the intermediate results. Here, we pass in an ($N =$ `V.length`) of memoized results, initialized to -1.

```
n(int[] V, int left, int right, int total, int[][] memo) {
 > right)
 0;
(memo[left][right] == -1) {
= total - bestSum(V, left+1, right, total-V[left], memo);
= total - bestSum(V, left, right-1, total-V[right], memo);
eft][right] = Math.max(L, R);

emo[left][right];
```

nber of recursive calls to `bestSum` must be $O(N^2)$, for th of V, an enormous improvement from $\Theta(2^N)$!

Iterative Version

recursive version, but the usual presentation of this as *dynamic programming*—is iterative:

```
n(int[] V) {
memo = new int[V.length][V.length];
total = new int[V.length][V.length];
 i = 0; i < V.length; i += 1)
][i] = total[i][i] = V[i];
 k = 1; k < V.length; k += 1)
nt i = 0; i < V.length-k-1; i += 1) {
l[i][i+k] = V[i] + total[i+1][i+k];
L = total[i][i+k] - memo[i+1][i+k];
R = total[i][i+k] - memo[i][i+k-1];
[i][i+k] = Math.max(L, R);

emo[0][V.length-1];
```

figure out ahead of time the order in which the memo- will fill in `memo`, and write an explicit loop.

e needed to check whether result exists.

y bother unless it's necessary to save space?

Longest Common Subsequence

d length of the longest string that is a subsequence of other strings.

ngest common subsequence of
`lls␣sea␣shells␣by␣the␣seashore`" and
`ld␣salt␣sellers␣at␣the␣salt␣mines`"

`␣sells␣␣the␣sae`" (length 23)

sting, for example.

rsive algorithm:

```
 of longest common subsequence of S0[0..k0-1]
[0..k1-1] (pseudo Java) */
 lls(String S0, int k0, String S1, int k1) {
= 0 || k1 == 0) return 0;
0-1] == S1[k1-1]) return 1 + lls(S0, k0-1, S1, k1-1);
urn Math.max(lls(S0, k0-1, S1, k1), lls(S0, k0, S1, k1-1);
```

but obviously memoizable.

ıoized Longest Common Subsequence

```
ngest common subsequence of S0[0..k0-1]
-1] (pseudo Java) */
tring S0, int k0, String S1, int k1) {
 new int[k0+1][k1+1];
 : memo) Arrays.fill(row, -1);
 k0, S1, k1, memo);


nt lls(String S0, int k0, String S1, int k1, int[][] memo) {
 k1 == 0) return 0;
1] == -1) {
 == S1[k1-1])
1] = 1 + lls(S0, k0-1, S1, k1-1, memo);

1] = Math.max(lls(S0, k0-1, S1, k1, memo),
              lls(S0, k0, S1, k1-1, memo));

][k1];
```

ill the memoized version be?

ıoized Longest Common Subsequence

```
ngest common subsequence of S0[0..k0-1]
-1] (pseudo Java) */
tring S0, int k0, String S1, int k1) {
 new int[k0+1][k1+1];
 : memo) Arrays.fill(row, -1);
 k0, S1, k1, memo);


nt lls(String S0, int k0, String S1, int k1, int[][] memo) {
 k1 == 0) return 0;
1] == -1) {
 == S1[k1-1])
1] = 1 + lls(S0, k0-1, S1, k1-1, memo);

1] = Math.max(lls(S0, k0-1, S1, k1, memo),
              lls(S0, k0, S1, k1-1, memo));

][k1];
```

ill the memoized version be? $\Theta(k_0 \cdot k_1)$

se Study in System and Data-Structure Design

ibuted version-control system, apparently the most pop- currently.

, it stores snapshots (*versions*) of the files and direc- re of a project, keeping track of their relationships, es, and log messages.

ıted, in that there can be many copies of a given repos- upporting indepenent development, with machinery to reconcile versions between repositories.

ı is extremely fast (as these things go).

A Little History

Linus Torvalds and others in the Linux community when r of their previous, propietary VCS (Bitkeeper) with- e version.

nentation effort seems to have taken about 2–3 months, he 2.6.12 Linux kernel release in June, 2005.

ame, according to Wikipedia,

ds has quipped about the name Git, which is British ıng meaning "unpleasant person". Torvalds said: "I'm ical bastard, and I name all my projects after myself. ux', now 'git'." The man page describes Git as "the ıtent tracker."

a collection of basic primitives (now called "plumbing") scripted to provide desired functionality.

-level commands ("porcelain") built on top of these to ıvenient user interface.

Major User-Level Features (I)

is of a graph of versions or snapshots (called *commits*) e project.

ructure reflects ancestory: which versions came from

contains

ry tree of files (like a Unix directory).

on about who committed and when.

ge.

o commit (or commits, if there was a merge) from which it was derived.

Conceptual Structure

l components consist of four types of *object:*

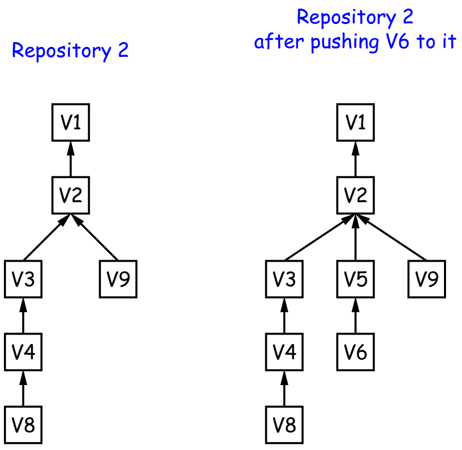
ically hold contents of files.

ectory structures of files.

Contain references to trees and additional information r, date, log message).

erences to commits or other objects, with additional n, intended to identify releases, other important ver- various useful information. (Won't mention further to-

rsion Histories in Two Repositories



Internals

ository is contained in a directory.

nay either be *bare* (just a collection of objects and r may be included as part of a working directory.

the repository is stored in various *objects* correspond- or other "leaf" content), trees, and commits.

e, data in files is *compressed*.

age-collect the objects from time to time to save addi-

---

ontent-Addressable File System

ne way of naming objects that is universal.

names, then, as pointers.

Which objects don't you have?" problem in an obvious

, what is invariant about an object, regardless of repos- *ontents*.

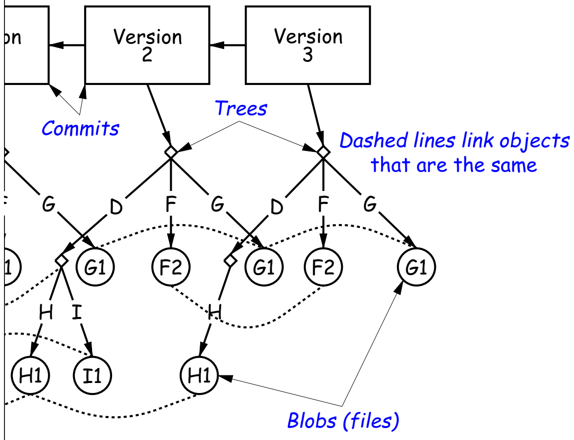
the contents as the name for obvious reasons.

hash of the contents as the address.

at doesn't work!

a: Use it anyway!!

Commits, Trees, Files



Major User-Level Features (II)

has a name that uniquely identifies it to all versions.

can transmit collections of versions to each other.

a commit from repository A to repository B requires nsmission of those objects (files or directory trees) not yet have (allowing speedy updating of repositories).

maintain named *branches*, which are simply identifiers commits that are updated to keep track of the most its in various lines of development.

ts are essentially named pointers to particular commits. branches in that they are not usually changed.

The Pointer Problem

it are files. How should we represent pointers between

ble to *transmit* objects from one repository to another nt contents. How do you transmit the pointers?

transfer those objects that are missing in the target How do we know which those are?

counter in each repository to give each object there a But how can that work consistently for two indepen- ories?

SHA1

A1 (Secure Hash Function 1).

und with this using the `hashlib` module in Python3.

ames in Git are therefore 160-bit hash codes of con-
.

commit in the shared CS61B repository could be fetched
ith

```
ckout e59849201956766218a3ad6ee1c3aab37dfec3fe
```

How A Broken Idea Can Work

o use a hash function that is so unlikely to have a colli-
can ignore that possibility.

ic Hash Functions have relevant property.

ion, f, is designed to withstand cryptoanalytic attacks.
, should have

resistance: given $h = f(m)$, should be computationally
to find such a message m.

e-image resistance: given message m_1, should be infea-
nd $m_2 \neq m_1$ such that $f(m_1) = f(m_2)$.

esistance: should be difficult to find *any* two messages
uch that $f(m_1) = f(m_2)$.

roperties, scheme of using hash of contents as name is
likely to fail, even when system is used maliciously.