Blelloch 4

Today: Dynamic Programming

Functional & Parallel

under the hood, support non-functional features (hide side effects)
sophisticated concurrent data structures

What is dynamic programming?

recursion (divide & conquer) with sharing

multiple people can make same call on same args — compute once

Recipe: (1) recursive solution to problem (with sharing)

(2) Calculate the span (based on recursive solution w/o worrying about sharing)

(3) Calculate the work (most dynamic progs are quite parallel)

a) identify unique calls → (same arguments)
b) determine cost at each

(usu. multiply cost)

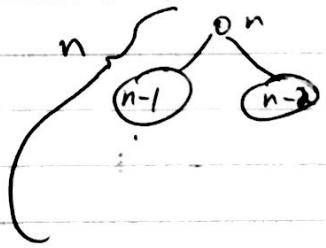
Limit to functions with integer arguments
(easy to test for equality
for purposes of sharing)
also for hashing
(easy to generate hash function)

Examples

index	0	1	2	3	4	5
value	1,	1,	2,	3,	5,	8

Simplest : fibonacci

```
fib n =
    if n ≤ 1
    then 1
    else fib (n-2) + fib (n-1)
```

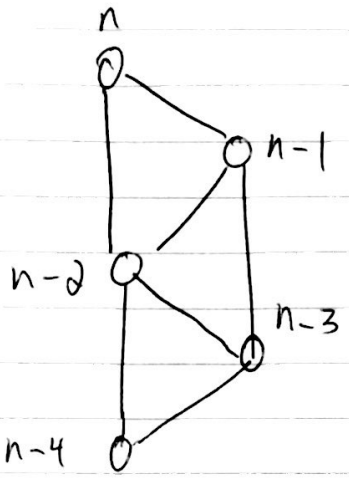


Span = O(n)

Work = n unique calls
× O(1)
= O(n)

(Alternatively: how many unique arguments to fib)

ex w/ sharing



size of DAG = Work

depth of DAG = Span

want (imple) hash table to work in parallel

fill hash table on way back up DAG
(call fill at bottom)

w/o sharing:

work is golden ratio to n

minimum edit distance
MED

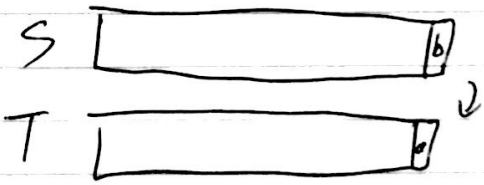
given 2 strings
how many edits to get from one to another

two strings : S, T

how many inserts into S, deletes from T needed for best case

[this is parallel]

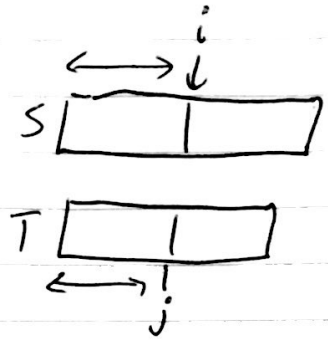
start at either two ends



if differ at last position
either delete 'b'
or insert 'a'

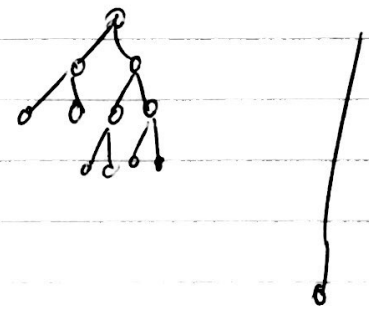
recursively, take best of the two
have to add a cost of 1

imagine as sequences




```
fun MED (S, T)
  = let
      fun MED'(i,j) =
        case i,j of
          (0,_) => j        (j inserts)
        | (_,0) => i
        | _ =>                          (i,j > 0)
            if S[i-1] = T[j-1]          ~> (0-based)
            then MED'(i-1, j-1)
            else 1 + min (MED'(i-1,j)
                          , MED'(i,j-1))
                  ins or del
    in
      MED(|S|, |T|)
    end
```

would take exponential time w/o sharing
(esp. worst case totally diff



2) span $O(|S| + |T|)$ linear

every recursive call
either remove from S
or " " T

every rec. call reduces $|S| + |T|$ by 1

care about $\frac{W}{S}$ (won't work much higher than span)

3) # of unique arguments
work $O(|S| \times |T|)$ (need +1 on each size)

$|S|$ values of i

$|T|$ values of j

constant work per call

parallelism = product over sum

theoretically no one knows a better alg. even sequentially (n^2)

SS

subset sum problem (unweighted knapsack)

given set S of natural numbers

and a $k \in \mathbb{N}$ (sum)

is there a subset of S exactly equal to k

$\exists ?\ X \subset S$ s.t. $\sum_{a \in X} a = k$

(NP-hard problem for any k)

we use 'k' in cost

```
fun SS (S, k) =                          (* assume seq. *)
                                         idea: try both
                                         adding
                                         removing
                                         each element
  let fun ss'(i, k) =
      case (i,k) of
        (0,0) => true   (empty set)
      | (0,_) => false  (k is non-zero)
      |  _    =>        (both i,k non-zero)
          if S[i-1] > k
          then ss'(i-1, k)
          else ss'(i-1, k - S[i-1])   <- include i-th elem
               or ss'(i-1, k)         <- delete i-th elem
  in ss'(|S|, k)
  end
```

2) Span (don't care about sharing)

$= O(|s|)$ (i is going down by 1 on each recursive call)

(so w/o sharing means exponential work)

3) work # unique arguments

$= (|s|+1)(k+1)$

cost of each call $= O(1)$
(not incl. recursive)

~~$= O(sk)$~~

Work $= O(|s| \cdot k)$

parallelism $= O(k)$

often k is a polynomial

(eg. in cryptography)

make k large enough for this brute force alg. too expensive

Longest Increasing Subsequence | LIS

subsequence - not contiguous

number of A's elements in same order

<10, 22, 9, 33, 21, 50, 41, 60, 80>

ex subseq. 50, 60, 80

answer = 6 (ex. <10, 22, 33, 50, 60, 80>)

ideally work = poly, span = linear