15-853:Algorithms in the Real World

Parallelism: Lecture 3

Parallel techniques and algorithms

- Contraction

Parallel Techniques

Some common themes in "Thinking Parallel"

1. Working with collections.
   - map, selection, reduce, scan, collect
2. Divide-and-conquer
   - Even more important than sequentially
   - Merging, matrix multiply, FFT, …
3. Contraction
   - Solve single smaller problem
   - List ranking, graph contraction, Huffman codes
4. Randomization
   - Symmetry breaking and random sampling

Technique 3: Contraction

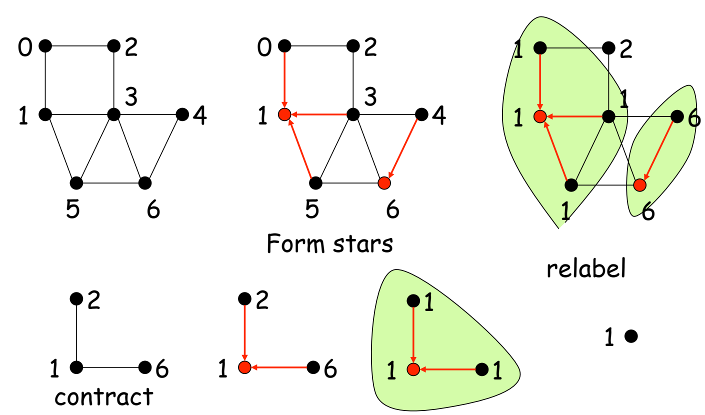
Consists of:

- Do some work to make a smaller problem
- Solve smaller problem recursively
- Use result to create solution of full problem

The code for scan was based on this, i.e.:

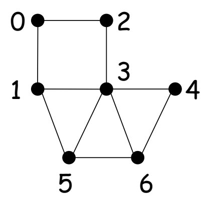
- Pairwise add neighbors in array
- Solve scan that is half as large
- Use results along with original values to generate overall result

Contraction : Graph Connectivity



Graph Connectivity

Representing a graph as an edge list:

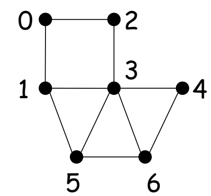



```
E = [(0,1),(0,2),(1,0),(1,3),(1,5),(2,0),(2,3),
     (3,1),(3,2),(3,4),(3,5),(3,6),(4,3),(4,6),
     (5,1),(5,3),(5,6),(6,3),(6,4),(6,5)]
```

Here every edge is represented once in each direction

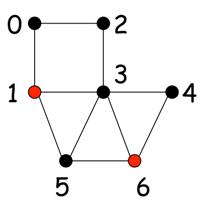
Graph Connectivity

Use an array of pointers, one per vertex to point to parent in connected tree. Initially everyone points to self.




```
L = [0,1,2,3,4,5,6]   (initially)
L = [1,1,1,1,6,1,1]   (possible final)
```

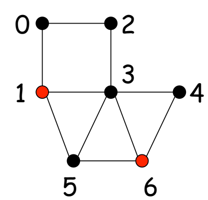

Graph Connectivity



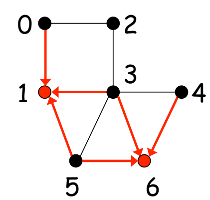
Randomly flip coins

```
FL = {coinToss(.5) : x in [0:#L]};
  FL = [0, 1, 0, 0, 0, 0, 1]
```


Graph Connectivity



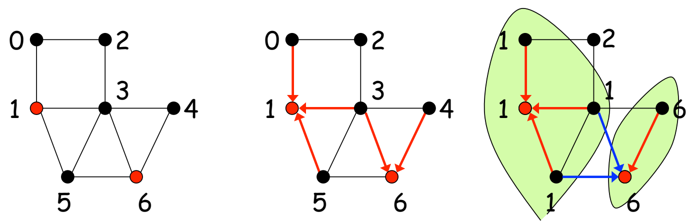
Randomly flip coins



Every edge link from black to red

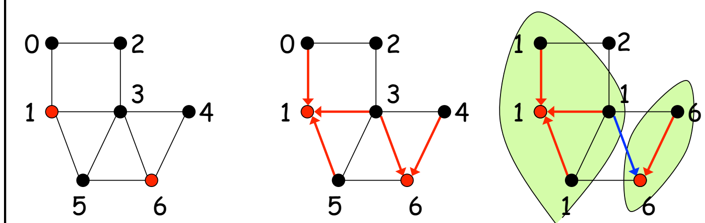
```
FL = [0, 1, 0, 0, 0, 0, 1]
H = {(u,v) in E | not(Fl[u]) and Fl[v]}
 H = [(0,1), (3,1), (5,1), (3,6), (4,6), (5,6)]
```

Graph Connectivity




```
H = [(0,1), (3,1), (5,1), (3,6), (4,6), (5,6)]
L = L <- H
   L = [1, 1, 2, 1, 6, 1, 6]
```


Graph Connectivity




```
L = [1, 1, 2, 1, 6, 1, 6]
E = {(L[u],L[v]): (u,v) in E | L[u]\=L[v]}
  E = [(1,2),(2,1),(2,1),(1,2),(1,6),(1,6),
       (6,1),(1,6),(6,1),(6,1)]
```

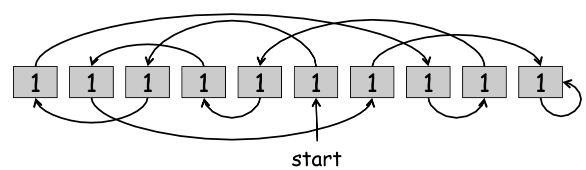

Graph Connectivity

L = Vertex Labels, E = Edge List

```
function connectivity(L, E) =
if #E = 0 then L
else let
  FL = {coinToss(.5) : x in [0:#L]};
  H = {(u,v) in E | not(Fl[u]) and Fl[v]}
  L = L <- H;
  E = {(L[u],L[v]): (u,v) in E | L[u]\=L[v]};
in connectivity(L,E);
```

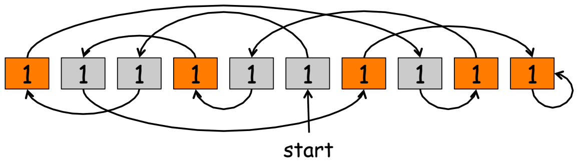
D = O(log n)
W = O(m log n)

List Ranking (again)




```
P = [7, 6, 0, 1, 3, 2, 9, 8, 4, 9]
W = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

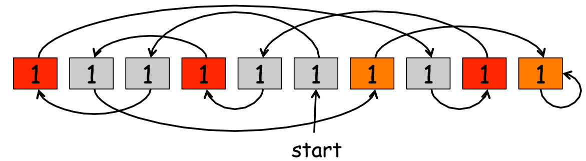
List Ranking



FL = {coinToss(.5) : x in [0:#P]};

FL = [1, 0, 0, 1, 0, 0, 1, 0, 1, 1]

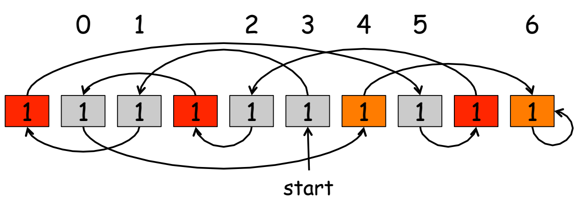
List Ranking



D = {FL[i] and not(FL[P[i]]) : i in [0:#P]};

D = [1, 0, 0, 1, 0, 0, 0, 0, 1, 0]

List Ranking

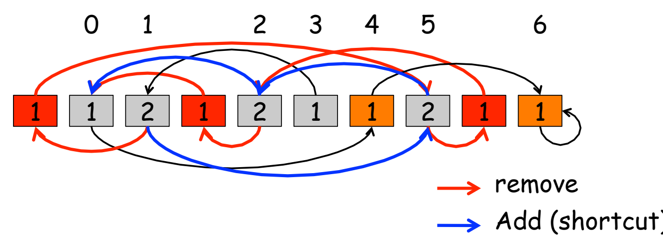


D = [1, 0, 0, 1, 0, 0, 0, 0, 1, 0]

NI = plusScan({not(x) : x in D});

NI = [0, $\underline{0}$, $\underline{1}$, 2, $\underline{2}$, $\underline{3}$, $\underline{4}$, $\underline{5}$, 6, $\underline{6}$]

List Ranking

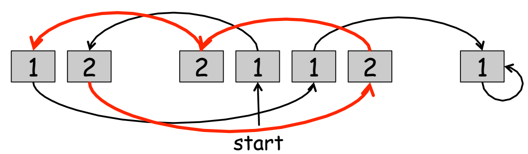


NI = [0, $\underline{0}$, $\underline{1}$, 2, $\underline{2}$, $\underline{3}$, $\underline{4}$, $\underline{5}$, 6, $\underline{6}$]

```
if D[P[i]] then
  (W[i] + W[P[i]], NI[P[P[i]]])
else (W[i],NI[P[i]])
```

List Ranking

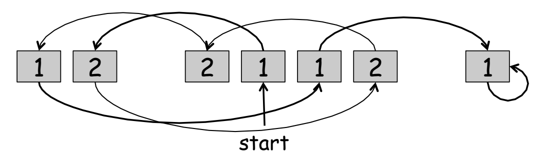
0 1 2 3 4 5 6




```
W = [1, 2, 2, 1, 1, 2, 1]
P = [4, 5, 0, 1, 6, 2, 6]
```


List Ranking

LR = 3 9 5 10 2 7 1

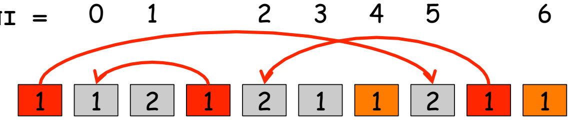



```
LR = listRank(W',P');
```


List Ranking

LR = 3 9 5 10 2 7 1

NI = 0 1 2 3 4 5 6



R = 8 3 9 4 5 10 2 7 6 1

```
If D[i] then LR[NI[P[i]]]+W[i]
    else LR[NI[i]]
```


List Ranking

```
function listRank(W, P) =
  if #P == 1 then [W[0]]
  else let
    FL = {coinToss(.5) : i in [0:#P]};
    D = {FL[i] and not(FL[P[i]]) : i in [0:#P]};
    NI = plusScan({not(x) : x in D});
    (W',P') = unzip {
        if D[P[i]] then (W[i] + W[P[i]], NI[P[P[i]]])
        else (W[i],NI[P[i]])
      : i in [0:#P] | not(D[i])};
    LR = listRank(W',P');
  in {if D[i] then LR[NI[P[i]]]+W[i]
      else LR[NI[i]]
    : i in [0:#P]};
```

Greedy: Huffman Codes

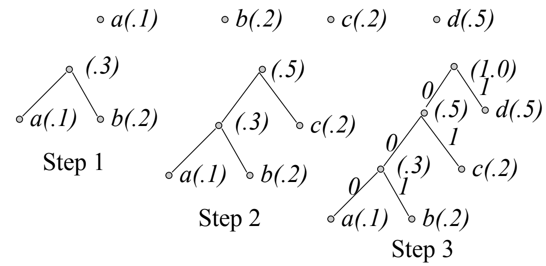
Huffman Algorithm:

Each p in P is a probability and a tree

```
function Huffman(P) =
  if (#P == 1) then return
  else let
    ((p1,t1),(p2,t2),P') = extract2mins(P)
    pt = (p1+p2, newNode(t1,t2)
  in Huffman(insert(pt,P'))
```


Example

p(a) = .1, p(b) = .2, p(c) = .2, p(d) = .5



a=000, b=001, c=01, d=1

Greedy: Huffman Codes

Huffman Algorithm:

How do we do it in parallel?

```
Function Huffman(P) =
  if #P == 1 then return
  else let
    ((p1,t1),(p2,t2),P') = extract2mins(P)
    pt = (p1+p2, newNode(t1,t2)
  in Huffman(insert(pt,P))
```


Primes Sieve

```
function primes(n) =
if n == 2 then [] int
else
  let sqr_primes = primes(ceil(sqrt(float(n))));
      sieves = flatten{[2*p:n:p]: p in sqr_primes};
      flags  = dist(t,n) <- {(i,f): i in sieves};
  in drop({i in [0:n]; flags | flags}, 2) ;
```

$$W(n) = O(n \log\log n)$$

$$\begin{aligned} D(n) &= D(\sqrt{n}) + O(\log n) \\ &= O(\log n) \end{aligned}$$

Parallel Techniques

Some common themes in "Thinking Parallel"

1. Working with collections.
   - map, selection, reduce, scan, collect
2. Divide-and-conquer
   - Even more important than sequentially
   - Merging, matrix multiply, FFT, …
3. Contraction
   - Solve single smaller problem
   - List ranking, graph contraction, Huffman codes
4. Randomization
   - Symmetry breaking and random sampling