Leftovers

Tuesday, April 17, 2018 7:11 PM

Fractional Knapsack

c_i - amount of items taken

$c_i \in \mathbb{Z}^+ \cup \{0\}$ ← bounded

For this problem

$0 \leq c_i \leq 1$

Goal maximize

$$\sum_{i=1}^{n} v_i \cdot c_i \text{ s.t. } C \geq \sum_{i=1}^{n} w_i \cdot c_i$$

Given v_i values and w_i weights.

We can argue we always take an integral weight of an item

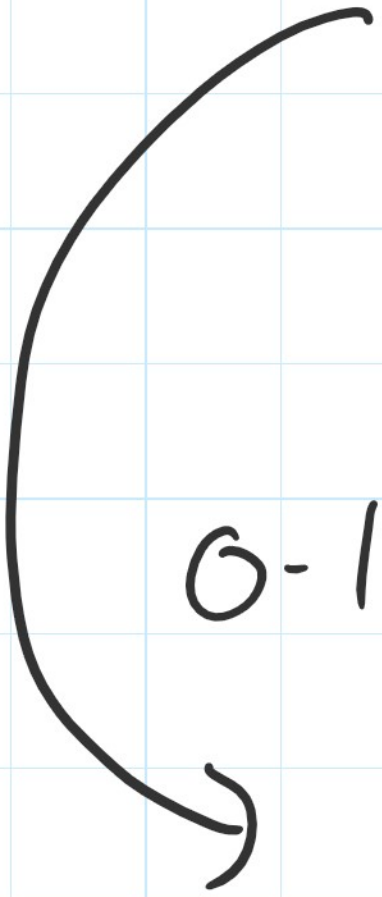
$w_i \in \mathbb{Z}^+$

Fractional knapsack $\epsilon \{0, 1\}$

fractional knapsack

W	V
1	5
2	8
3	9

if C=5 and 0-1 answer is 15



0-1 knapsack

w	V
1	5=5
1	8/2=4
1	8/2=4
1	9/3=3
1	9/3=3
1	9/3=3

if C=5 answer is 19

we can greedily take the largest value

Runtime = $O(\sum_i \log(\sum_i))$

too slow

Instead sorting by the highest value per weight. Always take the highest value per weight if possible, otherwise take as much as possible

Runtime $\in \Theta(|I| \log(|I|))$

Sorting

Swap sorts (Adjacent items only)

Gnome Sort

Bubble Sort

Cocktail Shake Sort

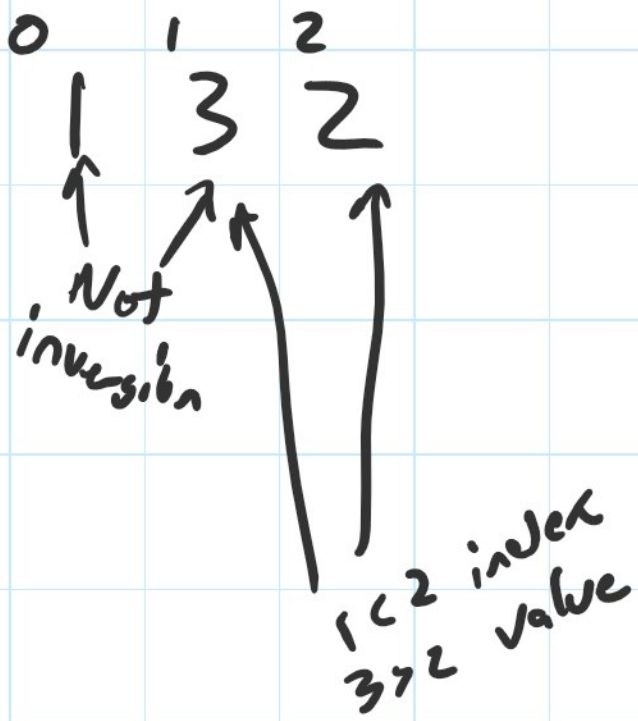
Inversion of an Array

is when we have

elements i, j

such that $i < j$

but $a_i > a_j$



min number of inversions

0 1 2
1 2 3

0

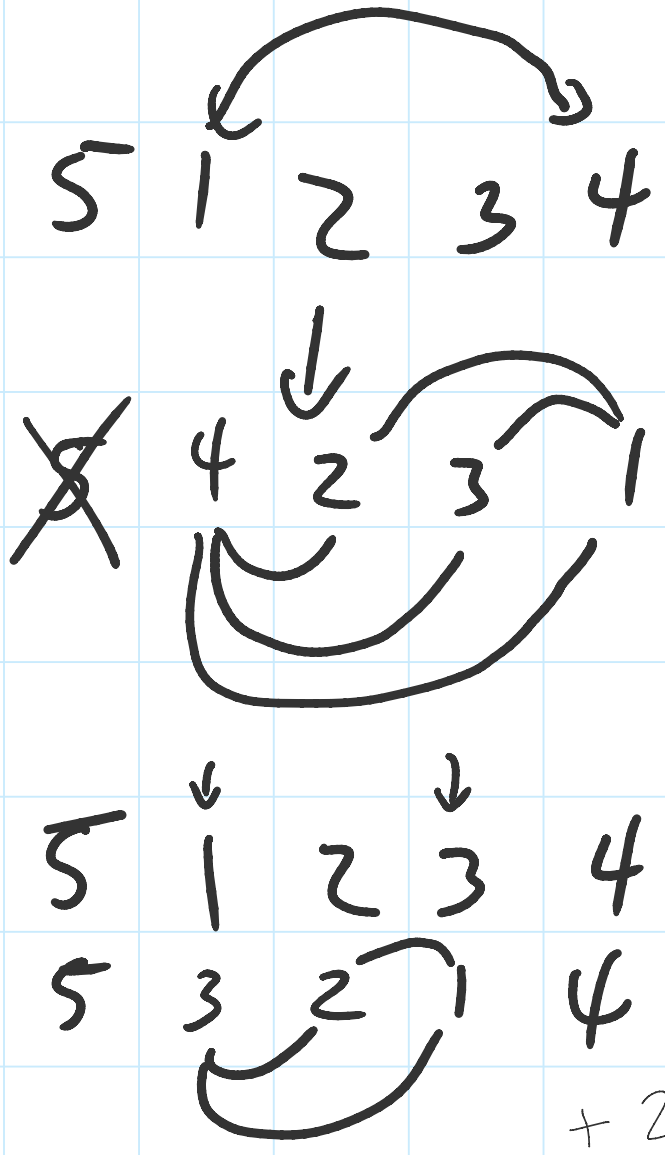
Max number of inversions in an array

5 4 3 2 1

4+3+2+1+0

$$\text{General} = (n-1) + (n-2) + \cdots +$$
$$1 + 0$$

$$\frac{(n)(n-1)}{2} \in O(n^2)$$



Swapping
adjacent elements
Creates or removes
how many inversions

5 4 3 2 1

5 3 4 2 1 +1 +0 inversion

adjacent swaps we can only ever remove (or add) 1 inversion.

What's the min number of adjacent swaps needed to sort an array?

Compute the number of inversions for the given array. The answer is is 1+T.

Why? we can always reduce the inversions if an array is unsorted

Proof.

If we have a positive number of inversions then our array is not sorted. Find an adjacent pair that is out of order. If no pair exist our array is sorted and we have a contradiction. otherwise swap said elements

otherwise swap said elements
and repeat ☒

Corollary.
Inversions is always possible for number of swaps, but we can't do better than it either.

In the worst Case
Invs $\in O(n^2)$

So The best adjacent swap sort has worst case $O(n^2)$

Search Runtimes

$|V| = N$

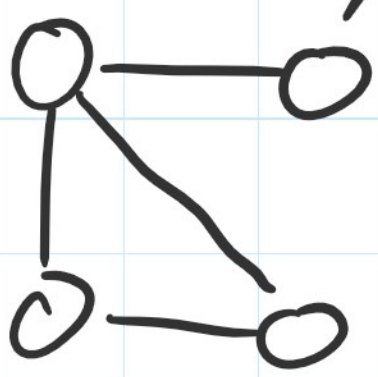
BFS runtime $|E| = M$

$O(|V| + |E|) = O(N+M)$

DFS runtime

$O(|V| + |E|) = O(N+M)$

DFS Theory?



each Node is visited at most

at most
each edge is incident
to at most 2 nodes
(visit each edge twice)

$O(2|E| + |V|) =$

$O(|E| + |V|)$

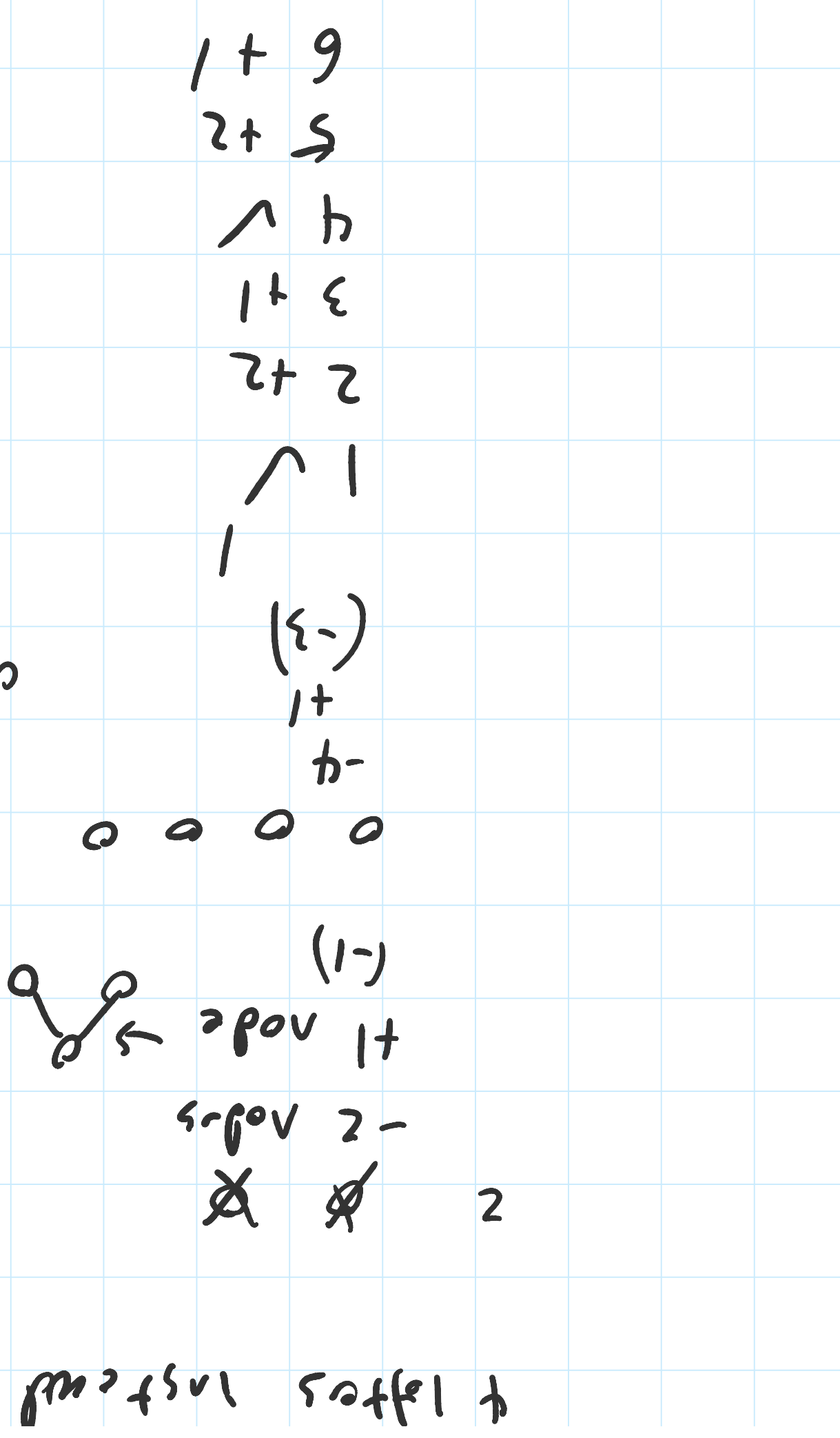
incident means that one of the
endpoints of an edge :
Node x is incident to Edge e if
x is one of e's endpoints

Huffman Encodings

Morse Code
Hard use ILP
Integer Linear Programming

Huffman encoding with
multiple letters

suppose our grammar has
4 letters instead of 2



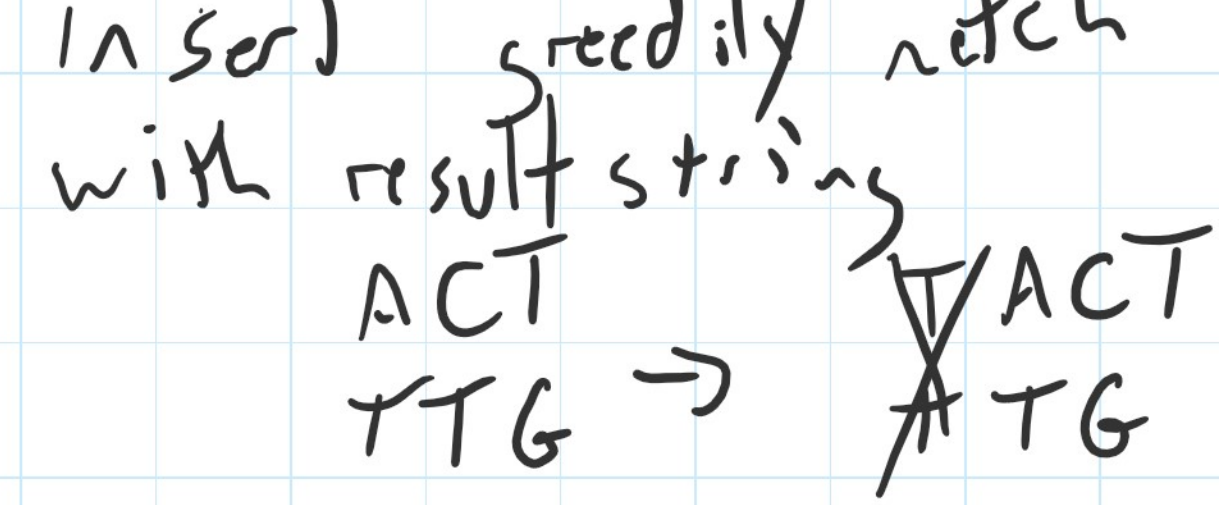
Sort nodes merge lowest
frequency into 1. add node
into set. repeat.

Edit Distance

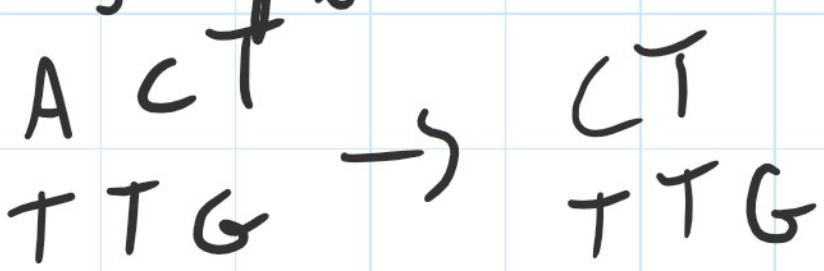
insert character
delete characters
change characters

We can use a DP similar to LCS.

1. for a pair of indices greedily remove characters, if they match
2. insert greedily match with result string

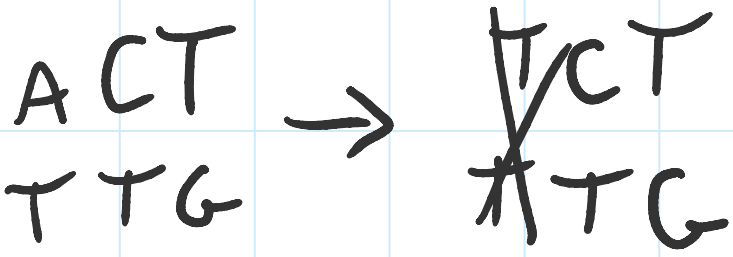


then remove
3. delete single



4. change greedily change

the value to the
correct one



then remove.

At a lower level

1. if(s1[first] == s2[second]){
   edit(first+1, second+1);
   }
2. edit(first, second+1) + insert cost
3. edit(first+1, second) + delete cost
4. edit(first+1, second+1) change cost