11.06.2022

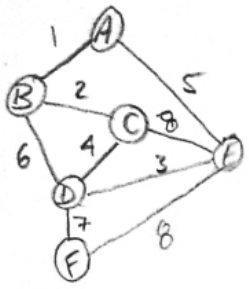
CSC 425 A#2

Sarah Clapoff
(Julia Putko collaborator)

#1 Good

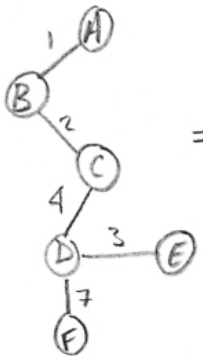
a) We are focusing on how we will store the nontree edges incident to a tree T s.t. they are searched in order of weight when maintaining a MST under deletions.

Eg. Graph G:



⇒

MST T:



nontree edges:
$E = [(A,E,5,NT), (B,D,6,NT), (C,E,8,NT), (E,F,8,NT)]$

ET TOUR:

ABCDFDEDCBA

$A = \{(A,B,1,T), (A,E,5,NT)\}$

$\Rightarrow B = \{(A,B,1,T), (B,C,2,T), (B,D,6,NT)\}$

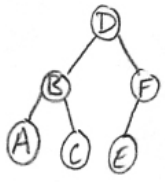
$C = \{(B,C,2,T), (C,D,4,T), (C,E,8,NT)\}$

$D = \{(D,E,3,T), (C,D,4,T), (B,D,6,NT), (D,F,7,T)\}$

$E = \{(D,E,3,T), (A,E,5,NT), (C,E,8,NT), (E,F,8,NT)\}$

$F = \{(D,F,7,T), (E,F,8,NT)\}$

ET TREE:



AND

E BBST:

Similar to the fully dynamic connectivity (FDC), we construct a balanced BST to represent the ET tree:

→ Each node will ~~con~~ store the following keys: $\{((u,v), w, t)\}$

1) (u,v): Every edge incident to the node on level i.
2) w: The weight of (u,v)
3) t: A bit representing whether (u,v) is a tree edge (T) or a nontree edge (NT).

Note: We will only need level i edges, since an edge on level i also appears in all T_j, $j \leq i$ (aka all levels below it).

In addition, to search for minimum edges, we will maintain E BBST. This is a balanced BST consisting of only nontree edges incident to MST T. Each node in E BBST will contain a list of all nontree edges with that weight. (Since this is just another BBST constructed from the ET tour, it has the same time complexities)

where each node in E BBST contains:
- NT edge(s) with that weight,

⇒ $6 = \{(B,D)\}$

$5 = \{(A,E)\}$

$8 = \{(C,E), (E,F)\}$

#1b) ADAPTING HLT's alg. for FDC (sections 3.1 and 4), but we consider replacement edges in order of increasing weight.

DELETE(u,v):

① if (u,v) a non-tree edge, remove from E, and EBBST.
if (u,v) a tree edge:
  $i \leftarrow \ell(u,v)$
① remove (u,v)
  call REPLACE((u,v), i)

REPLACE((u,v), i):
  $T_u, T_v \leftarrow$ Subtrees of MST containing u, v, respectively. // Let $T_u \leq T_v$ [HLT section 3.1]
  $\ell(T_u) = i+1$ // copy T_u (and thus the subtree in ET and EBBST) to level i+1
// Now we have a tree E_u BBST representing all nontree edges incident to T_u. Do a BFS to get them in ascending weight order.
② For e in BFS of E_u:
    if e connects T_u and T_v:
③ ③   insert as replacement edge,
      delete from E_u.
      Return.
    else:
      $\ell(e) = i+1$
// if have traversed E_u and found no replacement edge:
if i == 0:
  No replacement edge in G.
else: replace((u,v), i-1)

11/06/2022 CSC 425 A#2

#1 c) Total cost over m deletions? From HLT Section 2.1: ET operations CUT, JOIN, QUERIES are all $O(\lg n)$

COST FOR ONE DELETION:
(See corresponding #s in 1b))

① Remove an edge ⇒ CUT the corresponding ET TREES: $O(\lg n)$
⇒ if on level i, also remove from all levels $\leq i$.
Since we are always considering the smaller of T_u, T_v, $l_{max} = \lfloor \lg n \rfloor$. Hence, we may have to remove the edge at most $\lg n$ times.
⇒ $O(\lg^2 n)$

② Replace
⇒ May be called recursively over all levels: $\lg n$
⇒ For each edge e in E_u, we perform a connectivity query ⇒ $O(\lg n)$
⇒ Cost to consider non-replacement edges is covered by increasing the level.
⇒ If we search all e in E_u, then total cost is $O(|E_u| \lg n) = O(\lg n)$
⇒ Total for searching for replacement edge: $O(\lg n) * \lg n = O(\lg^2 n)$

③ Insert replacement edge:
⇒ JOIN the corresponding ET TREES: $O(\lg n)$
⇒ Similar to removing, also, insert edge on trees level $\leq i$. At most $\lg n$ times.
⇒ Total for insert = $O(\lg^2 n)$
⇒ Remove replacement edge from E BBSTs (and thus rebalancing them): $O(\lg n)$
⇒ at most $\lg n$ times for l_{max} levels.
⇒ $O(\lg^2 n)$

SO TOTAL TIME COMPLEXITY FOR ONE DELETION: $O(4\lg^2 n) = O(\lg^2 n)$

OVER m DELETIONS: $O(m \lg^2 n)$

11.06.2022 CSC 425 A#2

#1 d) Can you prove your alg correctly returns a MST?

FROM MINIMUM SPANNING TREE WIKIPEDIA:

MST DEF: A tree made from a connected, weighted, undirected graph, in which all vertices are connected w/out any cycles, and the total edge weight (sum of all edge weights) is minimal.

CLAIM: Given a connected weighted undirected graph G w/ n nodes and m edges, there exists an alg. that maintains a MST under deletions.

PROOF:

Initially, we construct a MST T from G. $\ell(T)=0$.

From T we do a ET tour and construct ET TREE, as well as E BBST, which represents nontree edges sorted in ascending weights.

In DELETE(u,v), when looking for a replacement edge, we consider the non tree edges in order of increasing weight, due to the BFS traversal of E BBST.

This means that the first replacement edge we encounter has the smallest weight and thus the total edge weight of the MST is minimal.

Invariants (i) and (ii) from HLT Section 3.1 (pg. 731) are maintained since this is a specialization of FDC.

Additionally, we look for a replacement edge on the highest possible level.

To maintain a MST, we can define the invariant: [HLT Section 4, pg. 733]

"(iii) Every cycle C has a nontree edge e with $w(e)=\max_{f \in C} w(f)$ and $\ell(e)=\min_{f \in C} \ell(f)$."

This gives:

LEMMA: "Assume (iii) and that F is a min. spanning forest. Then, for any tree edge e, among all replacement edges, the lightest edge is on the max. level." [HLT pg. 733]

→ To show this lemma, take e_1 and e_2 as replacement edges for e, where $w(e_1) < w(e_2)$. Since they are both replacement edges, a cycle is induced - $C=(C_1 \cup C_2)/(C_1 \cap C_2)$. Since F is a min spanning forest, e_1 is the heaviest edge on C_1. So e_2 is the heaviest edge on C. By (iii), e_2 must have the lowest level on C, and e_1 must have a higher level. [HLT pg. 733]

In other words, when we are increasing the level of an edge, we are doing so to the lightest edges first, as those are the first ones considered.

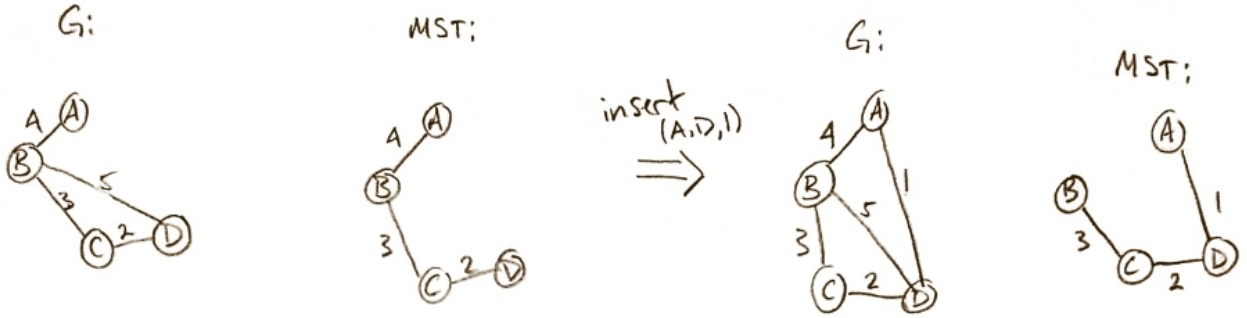
Thus, invariants (i), (ii), (iii) are maintained and the MST will stay minimal under deletions. ■

11.06.2022 CSC 425 A#2

#1e) What goes wrong with insertions?

When inserting an edge $e'=(u,v)$ with weight w', there is the possibility that w' is lighter than the current path connecting u and v.
Then, we would have to update the MST and the corresponding ~~ET~~ TREES.

For Example; take the following



Since the structure of the MST on level 0 is changed, all subtrees and subforests T_i also would have to be updated, and their corresponding ET trees would need to be entirely reconstructed. Hence, we'd need to use a different data structure.

HLT [section 5, pg. 734] present a solution to the fully dynamic MSF problem using a set, A, of subgraphs of G. For each A_i, they maintain a min spanning forest. When inserting and deleting edges, these subgraphs are updated.
Their presented solution has an amortized time of $O(\lg^4 n)$ per insertion or deletion.

11.06.2022 CSC 425 A#2 Sarah Clapoff

#2

Hiring Problem:

In the alg. where you get to hire many times, if the candidates are presented in random order, what is the probability that you hire exactly n times?

To hire exactly n times, candidates must be presented in order of rank (eg. $1,\dots,n$), so that each new candidate will have a rank higher than the current hire (ie. the previous candidate).

There is only one possible way to present the candidates in this way.

The total # of ways to present the candidates is all permutations of the list $(1,\dots,n)$. That is, $n!$

Thus, the probability of hiring exactly n times is $\boxed{P(\text{hiring } n \text{ times}) = \frac{1}{n!}}$

What is the probability you hire exactly twice?

We must hire the first candidate, and in order to ensure we hire twice, that candidate cannot be n. Let i be the rank of the first candidate.

For each $1 \le i < n$, we want:

① The probability that the candidate of rank i will be presented first $\Rightarrow \frac{1}{n}$

② The probability that the next hired candidate will be of rank n:

Since we don't hire candidates with rank $< i$, we want the prob of being presented n out of all $i+1, i+2, \dots, n$ ranks. There are $n-i$ candidates with $i < \text{rank} \le n$, and so $P(\text{Hire } n \text{ next}) = \frac{1}{n-i}$

Thus, for each $1 \le i < n$, the prob of hiring exactly twice is $\left(\frac{1}{n}\right)\left(\frac{1}{n-i}\right)$.

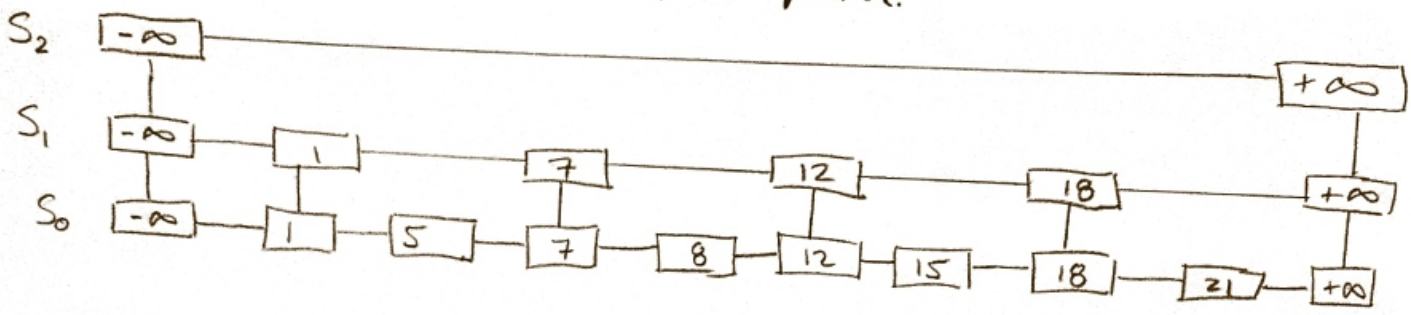
Then, sum over all i for a total probability of:

$$\boxed{P(\text{hire twice}) = \sum_{i=1}^{n-1} \frac{1}{n}\left(\frac{1}{n-i}\right)}$$

11/06/2022 CSC 425 A#2

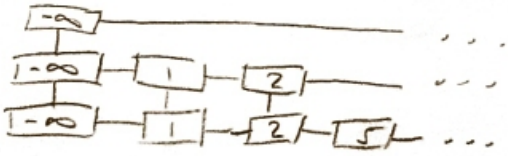
#3 Insert the following #s into an empty skip list: 1, 5, 7, 8, 12, 15, 18, 21
Suppose each time you need to flip a coin, heads and tails alternate.

a) What does your skip list look like? Draw a picture.



b) What in general can you say about what your skip list would look like if your coinflips turned out this way?

Since the coin flips alternate, there will never be an occurence of more than one heads occurring in a row, so the maximum level will be S_1 for #s inserted. Neighbouring elements will not always alternate levels however. For example, after inserting 21 as above, if we inserted 2, then the list would look like:



c) Expected time to search for a random key?

From the SEARCH(k) method, we start at the top level and scan the level to find the largest item $\leq k$. Then we repeat on lower levels. Since our skip list is bound to 2 levels, and on S_1 takes $n/2$ time to scan, we expect the time to search for a random key to be on the order of $\boxed{O(\frac{n}{2})}$

(ie. worst case, we scan to end of S_1 and then go down a level to find the last key $= k$. That is $\frac{n}{2}+1 \Rightarrow O(\frac{n}{2})$)

11.06.2022 CSC 425 A#2

#4a) Prove the recurrence given in the readings for the expected time needed for the Karger-Stein alg.
(4b) → NO ANSWER)

From Sahil Singla's Karger's Min cut alg. notes:

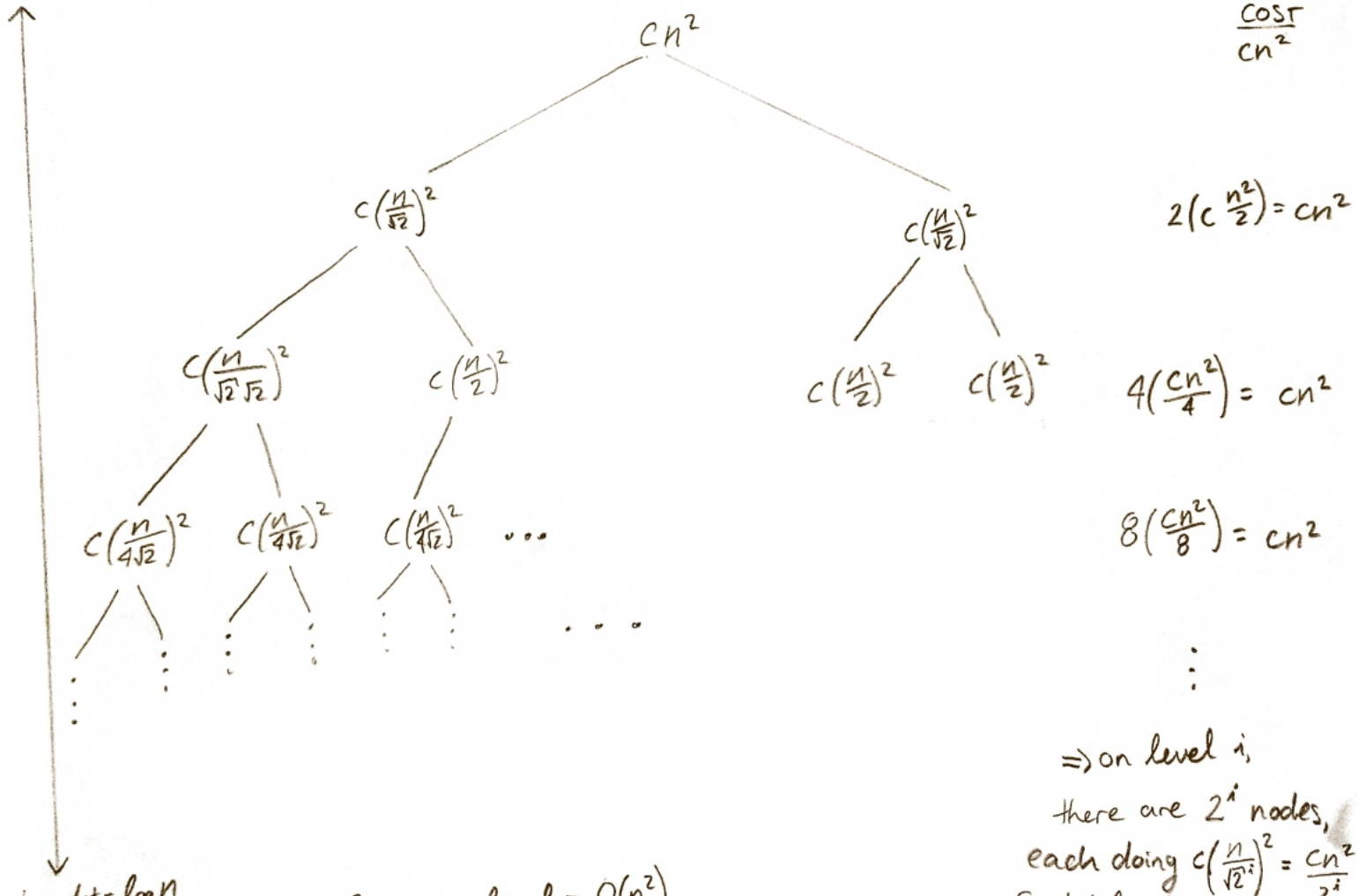
the runtime of the Karger-Stein alg. satisfies the recurrence:

$$T(n) = O(n^2) + 2T\left(\frac{n}{\sqrt{2}}\right)$$

Solving the recurrence gives: $T(n) = O(n^2 \log n)$

We will prove this solution using the recurrence tree.

PROOF: Consider the recurrence $T(n) = O(n^2) + 2T\left(\frac{n}{\sqrt{2}}\right)$. Let c be the constant term for $O(n^2)$. Then the recurrence tree is:



height = log n since this is a binary tree.

Cost per level = $O(n^2)$

# of levels = $O(\log n)$

$\therefore$ Total cost $T(n) = O(n^2 \log n)$

$\Rightarrow$ on level i, there are 2^i nodes, each doing $c\left(\frac{n}{\sqrt{2}^i}\right)^2 = \frac{cn^2}{2^i}$

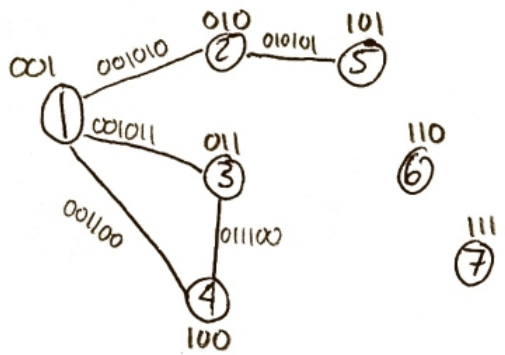
So total cost for each level $= cn^2$.

$\Rightarrow O(n^2)$

∎

11.06.2022 CSC 425 #2

#5 a) Suppose we have a graph with nodes 1,2,3,4,5,6,7 and edges (1,2), (1,3), (1,4), (2,5), (3,4)
Show that if we encode each edge according to the scheme given in class for the monte carlo connectivity algorithm, the XOR sum of the vectors for nodes 1,2,3, and 4 is the name of the edge (2,5).



To get node vectors, XOR all incident edges

① 001010 (1,2)
XOR 001011 (1,3)
000001
XOR 001100 (1,4)
001101

② 001010 (1,2)
XOR 010101 (2,5)
011111

③ 001011 (1,3)
XOR 011100 (3,4)
010111

④ 001100 (1,4)
XOR 011100 (3,4)
010000

XOR Sum of all node vectors:

001101 ①
011111 ②
010111 ③
XOR 010000 ④
010101 ⇒ 010101 = (2,5), as expected.

11.06.2022 CSC 425 A#2

#5 b) Let S be a set of n elements. Show that there exists a way to assign binary strings (names) of length x to each of the n elements s.t. every subset of S of size $\leq k$ has names which XOR to a unique string. Ie. The length x string, which is the XOR, uniquely defines the subset. How big does x have to be? This can be done by a counting argument.

NOTE: I'm not so sure about this one, but here are my ideas:

Since S is a set, it has no edges, and so we aren't able to identify subsets by the way the elements are connected.

For $k \leq n$, the total # of subsets size $\leq k$ is all combinations of i objects, $0 \leq i \leq k$:
Let N be the total # of possible subsets, then:

$$N = \sum_{i=0}^{k} {}_nC_i$$

For example, $n=8$, $k=4$. Then $N = {}_8C_1 + {}_8C_2 + {}_8C_3 + {}_8C_4 = 162$.
There are 162 possible subsets, we will need 162 unique binary strings.
$162 = (10100010)_b \Rightarrow$ 8 digits.
In other words, we can represent up to $2^8 = 256$ unique strings using 8 binary digits.
In this example, since we XOR all element names, strings must be length 8. $x = 8$.

In general, the unique subset name will need to be $2^{\text{length}(N_b)}$ bits long.
Since we are simply XOR-ing element names,

$$x = 2^l, \quad l = \text{length}(N_b), \quad N = \sum_{i=0}^{k} {}_nC_i$$

I am not sure how we could assign the names to each element.
→ Maybe # them in binary 1, ..., n (padded w/ 0's in front) and then XOR with the size of the subset (where the first 4 bits = size).
For example, if $n=8$, $k=4$, we want a subset S' size 4:

S': ② ⑤ ⑧ ⑦

NAMES:
② 0000 0010
XOR 0100 0000
0100 0010

⑤ 0000 0101
XOR 0100 0000
0100 0101

⑧ 0000 1000
XOR 0100 0000
0100 1000

⑦ 0000 0111
0100 0000
0100 0111

S' NAME:
0100 0010 NOT UNIQUE
0100 0101
0100 1000
0100 0111
try XOR 0000 1000
0100 0000
0100 1000
this gives a unique name for we specific size.

I am not sure how we could assign the names to each element.

→ Maybe # them in binary 1, ..., n (padded w/ 0's in front) and then XOR with the size of the subset (where the first 4 bits = size).

For example, if n=8, K=4, we want a subset S' size 4:

S: ② ⑤ ⑧ ⑦

NAMES:

② 00000010
XOR 01000000
01000010

⑤ 0000 0101
XOR 0100 0000
0100 0101

⑧ 0000 1000
XOR 0100 0000
0100 1000

⑦ 0000 0111
0100 0000
0100 0111

S' NAME:

0100 0010 NOT UNIQUE
0100 0101
0100 1000
0100 0111

this gives a unique name, bc we specify size.

0000 1000
try XOR 0100 0000
0100 1000