Fractional knapsack problem: O(n log n) - greedy. Compute value / lb v_i / w_i per item. Thief takes greatest value first, until item is out or run out of room

LCS subsequence: O(n): weighted median city

$\max(LCS(X, Y, m, n-1), LCS(X, Y, m-1, n))$, X & Y are strings, m & n are length, if m or n = 0, return 0

Hashing: When chaining, last insert goes first. remember insert at head

Chaining: Insert(T, x) - Insert x at head of list $T[h(key[x])]$, O(1). Delete(T, x) - search time + O(1)

Avoid collision: Search(T, k) - search key k in $T[h(k)]$ - O(n), n proportional to length of list.

Load factor $\alpha = n/m$ = average # of keys per slot. n = keys in hash table, m = number of slots = # stored in list

Thm: Both successful & unsuccessful search takes $\Theta(1+\alpha)$. For special case with $m = n$, w/ probability at least $1 - 1/n$, longest list is: $O(\ln n / \ln \ln n)$. Avg bin size = α

Div Method: Map each key k into one of the m slots by taking remainder of k divided by m. So $h(k) = k \bmod m$

Adv: Fast since only one div op, disadv = some values the hash depends on subset of bits. Good choice for m is prime, if not too close to power of 2 or 10. K = key to hash. m = size of table

Multiplication: Map each key k to one of the m slots indicated by fractional part of k times $0 < A < 1$.

$h(k) = \lfloor m(kA \bmod 1) \rfloor$. Ex, $m = 1000$, $k = 123$, $A \approx 0.6180339887\ldots$, $h(k) = \lfloor 1000(123 \cdot A \bmod 1) \rfloor = 18$

Value of m is not critical, but slower than div method. $m = 2^p$ choose if want.

Open addressing: Store all n keys in the m slots of the hash table itself. Probe slot h(k). If k is there, or not, then success / fail. If h(k) contains key that is not k, compute index of some other slot based on k & which probe we are on. Keep probe until we either find key k or NIL. Advantage = avoid pointers so less code & we can dedicate memory to table. Assume table never fills, $n \le m$, $\alpha < 1$, uniform hashing, no deletion.

Linear Probe: $h(k,i) = (h(k,0) + i) \bmod m$, k = key & i = probe number, suffers from primary clustering, ↑ find next available

Quadratic: $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$, $c_1 \neq c_2$. | Expected unsuccess search: $1/(1-\alpha)$

Double Hash: $h(k,i) = (h_1(k) + i h_2(k)) \bmod m$. ↑ max i when collision occurs | success search: $(1/\alpha)\ln(1/(1-\alpha))$

Dynamic Prog: Rod cutting problem:

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	30

, can cut 2^{n-1} ways

If optimal solution cuts rod into k pieces, optimal decomp = $n = i_1 + i_2 + \ldots + i_k$, revenue = $r_n = p_{i_1} + p_{i_2} + \ldots p_{i_k}$

$r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2} \ldots r_{n-1} + r_1\}$. Initial cut of rod: 2 pieces of size i & n-i.

Revenue r_i & r_{n-i}. Need to consider all values of i.

$q = \max\{q, p[i] + CUT\text{-}ROD(p, n-i)\}$. Only the remainder is further divided. $O(2^n)$

or $cutRod(n) = \max(price[i] + cutRod(n-i-1))$ for all i in $\{0, 1, \ldots n-1\}$

Matrix Multi: $A_1 = 5 \times 4$, $A_2 = 4 \times 6$, $A_3 = 6 \times 2$. $((A_1 A_2) A_3) = (5 \times 4 \times 6) + (5 \times 6 \times 2) = 180$

BST: worst = O(n), avg O(lg n) $(A_1 (A_2 A_3)) = (4 \times 6 \times 2) + (5 \times 4 \times 2) = 88$

Max Distance BST pair: Do base case: then $\langle dist_{left}, depth_{left} \rangle = MaxDistance(left\ subtree)$, then do same w/ right tree.

$dist = \max(depth_{left} + depth_{right} + 2, dist_{left}, dist_{right})$. $depth = \max(depth_{left}, depth_{right}) + 1$.

return $\langle dist, depth \rangle$

Check if avl: base case - then: $\langle leftIs, leftHeight \rangle = $ is AVL tree(leftTree(tree)). same w/ right,

height = max(leftHeight, rightHeight) + 1

if leftIs & rightIs & |leftHeight − rightHeight| ≤ 1 then

return yes, Height

else no, height

water path:
leftmax[0] = height[0]
for i from 1 to n-1,
leftmax[i] = max(leftMax[i-1], height[i])
rightMax[n-1] = height[n-1].
for i from n-2 to 0, decreasing,
rightmax[i] = max(rightMax[i+1], height[i])

for i from 0 to n-1
h = min(left max[i], right max[i])
~~fsd~~ maxArea = max(maxArea, h·(n-i))

TSP: generate all (n-1)! permutations of cities
calc cost of perm, track min.

Max Flow

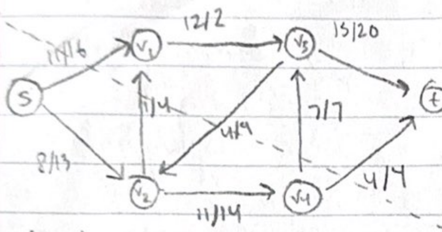
Properties: Capacity constraint: $f(u,v) \le c(u,v)$. skew sym: $f(u,v) = -f(v,u)$. Flow conservation: $\sum_{v \in V} f(u,v) = 0$

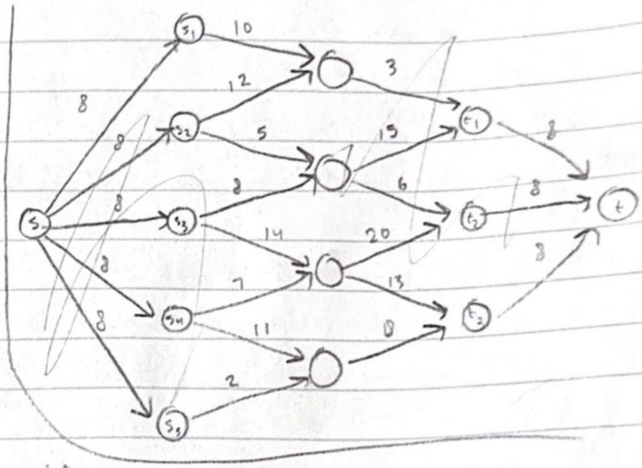
Given G, s, k, t, find a flow f of max value from s to t.

Following is equivalent: 1. f is max flow of G. G_f contains no augmenting path. $|f| = c(S,T)$ for some cut (S,T) of G.

Cut $(\{s, v_2, v_4\}, \{v_1, v_3, t\})$

$f(u,v) / c(u,v)$





$f(S,T) = 11 + 1 + 4 + 7 - 4 = 19$.
$-f(v_3, v_2)$

$c(S,T) = 16 + 4 + 7 + 4 = 31$. Ignore backwards

max Min alg: $mT = \max(\min(mL, T.val), \min(mR, T.val))$
if tree is empty, return $-\infty$
else if root = leaf, return root
else return mT.

FF can be used for negative cap.

FF does not always terminate.
In bipartite graph, max card ≠ max flow
FF does not always increase flow w/ iteration
FF does not always find in polynomial time

Djikstra's: $O(V^2)$ w/ prox matrix, min prio q then $O(V + E \log V)$. Using any augmented path can lead to loop or fail

Tech Meeting: Make new G'. For every edge (u,v) in G, there is two edge in $G' : (u_m, v_t)$ & (u_t, v_m).
Run dijkstra's to find shortest path from S_T to $Vancouver_m$. $O(V \log V + E)$

Max Flow: Suppose capacity for edge $(u,v) \in E$ is decreased by 1. Give $O(V+E)$ alg to update max flow.
f = max flow before. If 0, do nothing. else, $f'(x,y)$ define for $f'(u,v) = f(u,v) - 1$. This violates cap. constraints at u & v unless u = s or v = t. f' has one more flow entering u than leaving, & one less entering v than leaving.
Idea is to reroute so it goes out of u & into v through another path. If impossible, reduce flow from s to u & v to t by one. Look for augmenting path from u to v. If exist, augment flow on that path, else, reduce flow from s to u by augmenting flow from u to s. Also reduce flow from v to t by find aug path t → v & augment flow. $O(V+E)$ using DFS or BFS

Dinner Prob: p families, ith family has a(i) members, q tables, jth table has b(j) cap.
$G(V,E)$: ~~$V = \{s, t\} \cup \{u_i \mid 1 \le i \le p\} \cup \{v_j \mid 1 \le j \le q\}$ $E = \{(s, u_i) \mid 1$~~. Maxflow assigns families to tables. Flow from ui to vj is 0 or 1, latter means member of family i sits at j.

If cap increase, can cause issues if edge is part of min cut. Alg increases flow along edge. If there exists a minimum cut on which edge does not lie, max flow can't increase so no augmenting path in residual network. otherwise, perform one iteration of Ford Fulkerson. If augment path exists it will be found & increased on this iteration, flow values are integral since cap = int. Since flow strictly increase, single iteration increase flow by 1. Use BFS, runs in $O(V+E)$, know to be maximal