09/28

Coin changing problem

- Target amount X, use fewest possible coins
- minimize $a+b+c+d$ s.t $25a+10b+5c+d = X$

10/3/23

Greedy Alg

- Activity selection / Interval scheduling

  Input: a processor, n activities $\{1, \dots, n\}$.

  - each activity has start & finish time (s_t, f_t), $s_t \le f_t$

  Output: Find max subset of activities that can be scheduled.

  - feasible and maximal $A^* \subseteq I$ s.t $\{ i_1 \cap i_2 = 0 \mid i_1, i_2 \in I \}$

  Intuition

  - List the activities in some order
  - Earliest Finish Time

    sort f_t : $f_1 \le f_2 \le \dots \le f_n$

    $A = \{1\}$; $j = 1$

    for $i = 2:n$

    if $s_i \ge f_j$: $A \leftarrow A \cup \{i\}$, $j \leftarrow i$

  Proof of correctness

  - Greedy is clearly feasible — $G \triangleq$ Greedy Solution
  - Intuition: G always stays ahead of other algs. — $OPT \triangleq$ Optimal Solution.

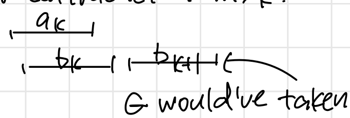
    Notation:

    $G: a_1, a_2, \dots, a_k$

    $OPT: b_1, b_2, \dots, b_m$ prove $m = k$ } Assume sorted order.

    Lemma: $f(a_i) \le f(b_i)$ for $i = 1:k$ $\longrightarrow$ ☆ suppose for contradiction $m > k$.

    $i = k$: G



    Proof: induction on i

    Base case $f(a_1) \le f(b_1)$

    IH: Assume for $i-1$, show for i.

    $f(a_{i-1}) \le f(b_{i-1}) \le s(b_i)$

    $f(a_i) \le f(b_i)$ (choose b_i) or better.

scheduling all activities

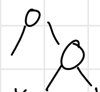
- Input: list of n activities $\{(s_1, f_1), \ldots, (s_n, f_n)\}$
- Output: fewest processors needed to schedule all.

Alg: sort by start time

10/5/23

Huffman Coding

- Prefix-free: no code is prefix for another.
- Tree Represention
  - left 0, right 1, leaf node is code (guarantees prefix-free)
- Optimality
  - Input character set C
  - $f(p)$ is frequency of $p \in C$.
  - Output: binary tree T
  - $d_T(p)$ is depth of p in T
  - $B(T) = \sum_{p \in C} f(p) d_T(p)$
  - Find $\underset{t \in T}{\operatorname{argmin}} B(t)$
- T can only have full nodes
  - If not, can lift one level

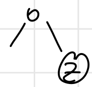


$x_1, x_2, \ldots \ldots x_{n-1}, x_n$

$\downarrow$

$x_1, x_2, \ldots \ldots z$

$f(z) = f_{n-1} + f_n$

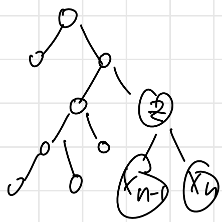


$T_1 \; (x_1, x_2, \ldots, x_{n-2}, z)$

$B(T) \; (x, \ldots, x_{n-2}, x_{n-1}, x_n)$

- Algorithm

```
Q ← C (by frequency min heap)
for i=1 to n-1:
    z ← new node
    x ← left[z] ← deleteMin(Q)
    y ← right[z] ← deleteMin(Q)
    f(z) ← f(x)+f(y)
    insert(Q, z)
```



$B(T_1) = B(T) - (f_{n-1} + f_n)$.

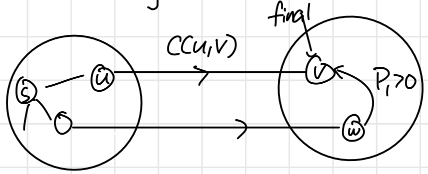
- Lemma: Suppose x, y are lowest frequency chars; then, there is always an opt code in which x, y have longest codes that only differ in last bit
  - Intuition: assume a, b have longer codes in opt. T.

$f(x) \leq f(a)$, $f(y) \leq f(b)$

10/10 Greedy

Dijkstra's

- All vertices belong to known (true shortest distance found) or unknown



$d(v) \le d(w)$

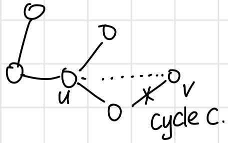
$d(v) < d(w) + C(P_1)$

Only if weights are positive.

Kruskal's Algorithm

Assume opt MST is T, output of kruskal is K.

Let (u,v) be first edge not in T.



One edge in cycle C not in K or else cycle, so (u,v) is cheaper than l, and you could swap (u,v) in for a better tree than T. Contradiction

Horn Clauses

- Boolean variable represents some event.
- 2 kinds of clauses
  1. Implication $(a \wedge b) \longrightarrow c$
     - LHS is n-ary conjunction of positive variables
     - RHS is single boolean variable.
  2. Negative $(\bar{a} \vee \bar{b} \vee \bar{c})$
     - n-ary disjunction of negated variables.
- e.g. $\longrightarrow x$

  $\longrightarrow y$

  $x \wedge u \longrightarrow z$

  $\bar{x} \vee \bar{y} \vee \bar{z}$

  satisfying assignment

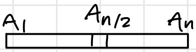
  $x=0, y=1, z=0, u=0.$

- Algorithm
  1. initialize all variables as false
  2. while $\exists$ unsatisfied implication, set RHS to true.
- Implications force positive assignments.

Set Covering
- Input is set $B = \{1, 2, \ldots, n\}$ and family $S = \{S_m \subseteq B\}$.
- Output minimal subset of S that covers B.
- Greedy is not optimal but good option for this hard problem.

10/12/23 Divide & Conquer
- General format
  1. Divide into subproblems
  2. Recursively solve subproblems
  3. Merge the solutions

Binary Search
- Input: array A.



- Output: find x
- $T(n)$ is time complexity of algorithm on input size n
- $T(n) = T(\frac{n}{2}) + O(1)$

Merge Sort
- Input: (A, p, q)
- Output: sorted A between p, q.
- $T(n) = 2T(\frac{n}{2}) + cn \longrightarrow O(n\log n)$

Multiplying 2 large nums
- Input: 2 n-bit numbers x, y
- Output: xy

  x: $a \mid b$ $\longrightarrow x = a2^{n/2} + b$

  y: $c \mid d$ $\longrightarrow y = c2^{n/2} + d$

  shift bits

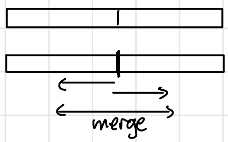
  multiplying $\frac{n}{2}$ bit nums

  $xy = (a2^{n/2} + b) + (c2^{n/2} + d) = ac2^{n} + bc2^{n/2} + ad2^{n/2} + bd$
- $T(n) = 4T(\frac{n}{2}) + O(n) = O(n^2)$

- **Karatsuba's Algorithm**
  - Need to compute $ac, bc, ad, bd \longrightarrow$ too many subproblems
  - Observe $(a-b)(c-d) = (ac+bd) - (ad+bc)$
  - 3 subproblems:
    1. $a \times c$ gives ac
    2. $b \times d$ gives bd
    3. $(a-b)(c-d)$ gives $ad+bc$ using (1), (2)
  - $xy = \underline{ac}\,2^n + \underline{(ad+bc)}\,2^{n/2} + \underline{bd}$
  - $T(n) = 3T(\frac{n}{2}) + O(n) \in O(n^{\log_2 3})$ 1.59

Maximum Subsequence

- Input: Array size n.
- Output: $(i,j) = \underset{(i,j)}{\operatorname{argmax}} \sum_{k=i}^{j} A_i$
- Algorithm



find longest of left and right

find longest from middle

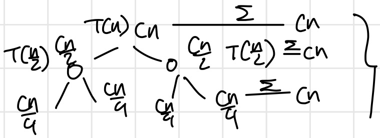
- $T(n) = 2T(\frac{n}{2}) + O(n)$

10/17 Recurrence Solving

1. expansion

$$
\begin{aligned}
T(n) &= 2T(\tfrac{n}{2}) + cn \\
&= 2\left(2T(\tfrac{n}{2^2}) + c\tfrac{n}{2}\right) + cn \\
&= 2^2 T(\tfrac{n}{2^2}) + 2cn \\
&= 2^2\left(2T(\tfrac{n}{2^3}) + c\tfrac{n}{4}\right) + 2cn \\
&= 2^3 T(\tfrac{n}{2^3}) + 3cn \\
&\;\vdots \\
&= 2^i T(\tfrac{n}{2^i}) + icn = n(1) + cn\log_2 n \in O(n\log n)
\end{aligned}
$$

$\frac{n}{2^i} = 1 \Rightarrow n = 2^i \Rightarrow i = \log_2 n$



$\log n$ levels

$\cdot\ T(n) = 4T(\frac{n}{2}) + cn$

$= 4(4T(\frac{n}{2^2}) + c\frac{n}{2}) + cn$

$= 4^2T(\frac{n}{2^2}) + 2cn + cn$

$= 4^2(4T(\frac{n}{2^3}) + c\frac{n}{4}) + 2cn + cn$

$= 4^3T(\frac{n}{2^3}) + 4cn + 2cn + cn$

$\vdots$

$= 4^iT(\frac{n}{2^i}) + cn2^{i-1} \quad = n^2 + cn2^{\log_2 n - 1} \in O(n^2)$

$\frac{n}{2^i} = 1 \Rightarrow i = \log_2 n,$

$4^i = 2^{2i} = 2^{2\log_2 n} = 2^{\log_2 n^2} = n^2$

$\cdot\ T(n) = 2T(\frac{n}{4}) + \sqrt{n} \quad | \ T(1) = 1$

$= 2(2T(\frac{n}{4^2}) + \sqrt{\frac{n}{4}}) + \sqrt{n}$

$= 2^2T(\frac{n}{4^2}) + 2\sqrt{n}$

$= 2^2(2T(\frac{n}{4^3} + \sqrt{\frac{n}{4^2}}) + 2\sqrt{n}$

$= 2^3T(\frac{n}{4^3}) + 3\sqrt{n}$

$\cdots = 2^iT(\frac{n}{4^i}) + i\sqrt{n}$

$n = 4^i = 2^{2i} \Rightarrow 2i = \log_2 n \Rightarrow i = \frac{1}{2}\log n$

$4^i = n,\ 2^i = \sqrt{n}$

$T(n) = \sqrt{n} + \frac{1}{2}\log_2 n\sqrt{n} \in O(\sqrt{n}\log n)$

2. <u>Master Method.</u>

$T(n) = aT(\frac{n}{b}) + \Theta(n^p(\log n)^k)$ <u>Key constraints</u>: $a \geq 1,\ b > 1$ constant, $p, k \geq 0$

Case 1: if $p < \log_b a$

$T(n) = \Theta(n^{\log_b a})$

Case 2: if $p = \log_b a$

$T(n) = \Theta(n^p(\log n)^{k+1})$

Case 3: if $p > \log_b a$

$T(n) = \Theta(n^p(\log n)^k)$

<u>eg.</u> 1 $T(n) = T(\frac{n}{2}) + \Theta(n)$

$a=2, b=2, p=1, k=0.$

$\log_b a = 1 = p \Rightarrow T(n) = \Theta(n\log n)$

2. Binary Search $T(n) = T(\frac{n}{2}) + \Theta(1)$

$a=1, b=2, p=0, k=0.$

$\log_b a = 0 = p \Rightarrow T(n) = \Theta(\log n)$

3. $T(n) = 2T(\frac{n}{2}) + \Theta(n\log n)$

$a=2, b=2, p=1, k=1$

$\log_b a = 1 = p \Rightarrow T(n) = \Theta(n(\log n)^2)$

4. $T(n) = 3T(\frac{n}{2}) + \Theta(n)$

$a=3, b=2, p=1, k=0$

$\log_2 3 > p \Rightarrow T(n) = \Theta(n^{\log_2 3})$

5. $T(n) = 4T(\frac{n}{2}) + \Theta(n^2\sqrt{n})$

$a=4, b=2, p=2.5, k=0$

$\log_b a = 2 < p \Rightarrow T(n) = \Theta(n^{2.5})$

HW1 problem 4

- input: n files, lengths l_i, access p_i
- Greedy: sort by $\frac{l_i}{p_i}$
- Suppose optimal ordering is not increasing in $\frac{l_i}{p_i}$
  - pick first place with inversion $\frac{l_i}{p_i} > \frac{l_{i+1}}{p_{i+1}}$ → $l_i p_{i+1} > l_{i+1} p_i$
  - Swap the inversion $1, 2, \dots \frac{l_{i+1}}{p_{i+1}}, \frac{l_i}{p_i}, i+2 \dots n$
  - Cost X for inverted, X' for swap
  - cost from $1 \dots i-1$ and $i+2 \dots n$ is unchanged

$$\mathrm{Cost}(X) - \mathrm{cost}(X') = l_{i+1} p_i - l_i p_{i+1}$$

$$\mathrm{cost}(X') = (\cancel{c} + \cancel{l_i}) p_i + (\cancel{c} + l_i + \cancel{l_{i+1}}) p_{i+1}$$

$$\mathrm{cost}(X) = (\cancel{c} + \cancel{l_{i+1}}) p_{i+1} + (\cancel{c} + \cancel{l_i} + l_{i+1}) p_i$$

10/19 Divide & Conquer

1. Matrix Multiplication

- Naive $O(n^3)$

$$A = \left(\begin{array}{c|c} a_{11} & a_{12} \\ \hline a_{21} & a_{22} \end{array}\right) \quad B = \left(\begin{array}{c|c} b_{11} & b_{12} \\ \hline b_{21} & b_{22} \end{array}\right) \quad C = AB = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)$$

$$\left.\begin{aligned} C_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ C_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ C_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ C_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}\right\} \longrightarrow O(n^3) \; :($$

- Strassen's Algorithm

$$\left.\begin{aligned} P_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ &\vdots \\ P_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}\right\} \quad \begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_4 \\ C_{22} &= P_1 + P_5 - P_2 + P_6 \end{aligned} \longrightarrow \begin{aligned} &T(n) = 7T(\tfrac{n}{2}) + O(n^2) \\ &\in O(n^{\log_2 7}). \end{aligned}$$

2. Quicksort

- inplace: doesn't require extra space
- ideal case pivot splits in half $T(n) = 2T(\frac{n}{2}) + O(n) \in O(n \log n)$
- worst case pivot at end $T(n) = T(n-1) + O(n) \in O(n^2)$
- Assume pivot : [i-1 | pivot | n-i]

$$\mathbb{P}[\text{pivot is } i\text{-splitter}] = \frac{1}{n}, \ \forall i$$

$$T(n) = \mathbb{E}_i[T(i-1) + T(n-i)] + O(n) = O(n) + \frac{1}{n}\sum_{i=1}^{n}[T(i-1) + T(n-i)]$$

· Solving recurrence

$$T(n) = \frac{1}{n}\sum_{i=1}^{n}\left[T(i-1)+T(n-i)\right] + (n+1)$$

$$= \frac{2}{n}\sum_{i=0}^{n-1} T(i) + n + 1$$

$$nT(n) = 2\sum_{i=0}^{n-1} T(i) + n^2 + n$$

$$(n-1)T(n-1) = 2\sum_{i=0}^{n-2} T(i) + (n-1)^2 + (n-1)$$

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2n$$

$$nT(n) = (n+1)T(n-1) + 2n$$

Divide by $n(n+1)$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2}{n+1} = \left(\frac{T(n-2)}{n-1} + \frac{2}{n}\right) + \frac{2}{n+1} = \frac{T(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \cdots$$

copy rest at home