Overview of Today's Lecture

I. Warmup, Review, Questions, Homework, etc.
II. Greedy Approach in Bin Packing
III. Coin Changing
IV. Interval Problems
V. Huffman Code
VI. Minimum Spanning Tree

Next...

[Start] [End]

I. Review, Questions, Homework, etc.

Warmup, Review, Questions, Homework, etc.

- Next week is in-class Midterm.

  Covers all the material up to Chapter 4.

  Study guide: read our solutions to the homeworks,

  because we like to build upon them.

  – We will publish an ”estimated grade” based on midterm and hwk1-hwk5.

  – Don’t worry because grades are curved (not absolute).

- Hwk5 is due this Thursday. No late hwk

Next...

[Start] [End]

II

Introduction to Greedy Approach

- What is the Greedy Approach?
  - Think of an algorithm as sequence of decisions
  - At each step, do what seems best for this step
  - Moreover, *no regrets!!*

THIS IS THE GREEDY PARADIGM?

Simple! Perhaps too simple?

Introduction to Greedy Approach

To be specific: for $i = 1, 2, \ldots$

–the ith Step has a set X_i of choices.

–there is a "gain function" $G : X_i \to \mathbb{R}$

–to choose $x_i \in X_i$ to maximize $G(x)$.

greedy $\equiv$ LOCAL optimization

Introduction to Greedy Approach

Surprising, this sometimes give OPTIMAL solutions!

In reality:

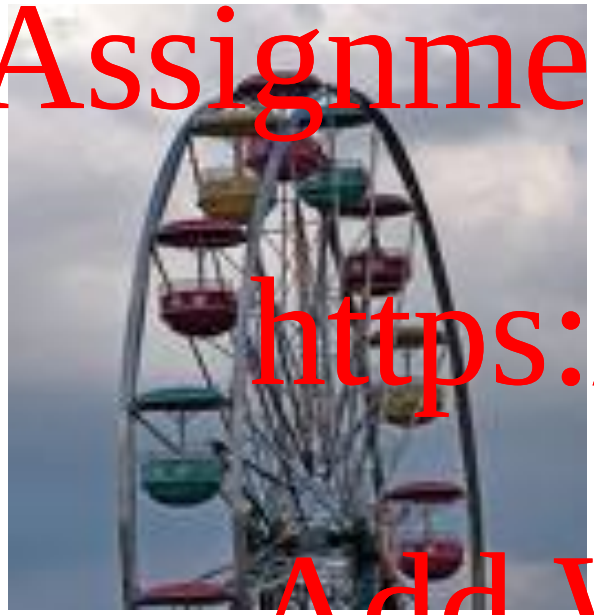
There is usually some global information used

(some global sorting)

–how to choose the ordering

–how to define the greedy function $G(x)$

Joy Rides and Bin Packing



Ferris wheel

Riders join a queue

Load riders into current car if possible

Each car has a maximum load

Start new car if max load will be exceeded

Joy Rides and Bin Packing

- Example

Maximum weight is $M = 400$

Queue: $(30, 190, 80, 210, 100, 80, 50)$

Solution 1 (Greedy) drop the last 0 digit

$(3, 19, 8), (21, 10), (8, 5)$

Solution 2 (Non-greedy)

$(3, 19), (8, 21, 10), (8, 5)$

Different solution, but “no improvement”.

Joy Rides and Bin Packing

- Policies

Online Policy on "First-Come First-Decide"

The decision might be: *wait for the next car*

First-Come First-Ride Policy (FCFR)

These two policies are independent.

FCFR is captured by "linear bin packing" formulation

Given $w = (w_1, \ldots, w_n)$, you can only insert "breakpoints" to choose your cars.

THEOREM: the greedy method is optimal among all FCFR policies.

Sketch: proof by contradiction...

Bin Packing

- Optimality

W.l.o.g., assume $M = 1$ (normalized) and queue is $w = (w_1, \ldots, w_n)$.

Let $A(w)$ be the number of cars used by algorithm A.

E.g., $w = (0.3, 0.2, 0.6, 0.5)$

FCFR algorithm, $G_1(w) = 3$

Sorted FCFR algorithm, $G_2(w) = 2$

E.g., $w = (0.3, 0.7, 0.3, 0.7)$

FCFR algorithm, $G_1(w) = 2$

Sorted FCFR algorithm, $G_2(w) = 3$

So, G_1, G_2 are incomparable!

Bin Packing

- Bin Packing:

Let $Opt(w)$ be the number of cars used by the optimal algorithm (with arbitrary policy)

Bin Packing Problem : compute $Opt(w)$

It is an NP-complete Problem (last lecture in this class).

Such problems are not known to have polynomial-time algorithms (we believe they do not exist).

Joy ride problem with FCFR constraint is Linear Bin Packing.

Bin Packing

- How Good is Linear Bin Packing?

THEOREM

(1) $Opt(w) \geq 1 + \lfloor G_1(w)/2 \rfloor$

(2) For all n, there exist w such that $Opt(w) = 1 + \lfloor n/2 \rfloor$ and $G_1(w) = n$

Significance? The (trivial) Linear Bin Packing is a factor of 2 from the optimal!

Bin Packing

Proof: let $G_1(w) = k$.

(1) If W_i is the weight of the i-th bin, then $W_i + W_{i+1} > 1$

Thus $\sum_{i=1}^{k} W_i > \lfloor k/2 \rfloor$.

It follows that $Opt(w) \geq 1 + \lfloor k/2 \rfloor$. This proves (1).

(2) Assume n is odd. Consider the n-vector

$$w = (\tfrac{1}{n}, 1, \tfrac{1}{n}, 1, \ldots, 1, \tfrac{1}{n})$$

Then $G_1(w) = n$ but $Opt(w) = 1 + \lfloor n/2 \rfloor$.

Bin Packing

- The Concept of Approximation Ratio

Suppose A is an algorithm for bin packing.

Absolute approximation ratio of A:

$$\alpha_0(A) := \sup_w \frac{A(w)}{Opt(w)}$$

For our Greedy algorithm for linear bin packing, $\alpha_0(G_1) = 3$

The problem is that this $\sup_w$ may be determined by a single value of w (*we like it to be achieved asymptotically*). To overcome this, we define

$$\alpha(A) := \limsup_n a_n = \lim_{n\to\infty} \sup\{a_k : k \geq n\}$$

where $a_n := \sup\left\{\frac{A(w)}{Opt(w)} : Opt(w) = n\right\}$.

Bin Packing

E.g., from our theorem (1) that

$$Opt(w) \geq 1 + \lfloor G_1(w)/2 \rfloor \geq (G_1(w) + 1)/2$$

we conclude that

$$\frac{G_1(w)}{Opt(w)} \leq 2 - \frac{1}{Opt(w)}$$

We mustn't conclude that $\alpha(G_1) < 2$, only conclude $\alpha(G_1) \leq 2$.

What about the lower bound on $\alpha(G_1)$? We know that the bound $2 - \frac{1}{Opt(w)}$ can be achieved by $Opt(w) = n$ for every n. Hence the limit is 2:

$$\boxed{\alpha(G_1) = 2}.$$

First Fit Bin Packing

- How to beat Linear Bin Packing's factor of 2

We now consider the First Fit (FF) bin packing algorithm that can beat Linear Bin Packing: $\alpha(FF) \leq 17/10$.

First Fit Bin Packing

- First Fit Bin Packing


```
First Fit Bin Packing
    INPUT: w = (w_1, ..., w_n)
    Initialize n empty bins (B_1, ..., B_n)
    For i ∈ 1, ..., n
        Place w_i into the first bin B_j that fits
    Return the non-empty bins, (B_1, ..., B_k)
```


E.g., $w = (3,5,4,1,3,2,3)$. $Opt(w) = 3$, and $G_1(w) = 5$. But $FF(w) = 4$.

Do simulation

$$B = ((3,4),(5,1),(3,2),(3))$$

First Fit Bin Packing

i	w_i	B_1	B_2	B_3	B_4
1	3	3			
2	5	3	5		
3	4	3	5		
		4			
4	1	3	5		
		4	1		
5	3	3	5	3	
		4	1		
6	2	3	5	3	
		4	1	2	
7	3	3	5	3	3
		4	1	2	

First Fit Bin Packing

- First Fit Bin Packing

THEOREM

$$FF(w)/Opt(w) \leq 17/10\,.$$

- Complexity?

$T(n) = O(n^2)$

Next...

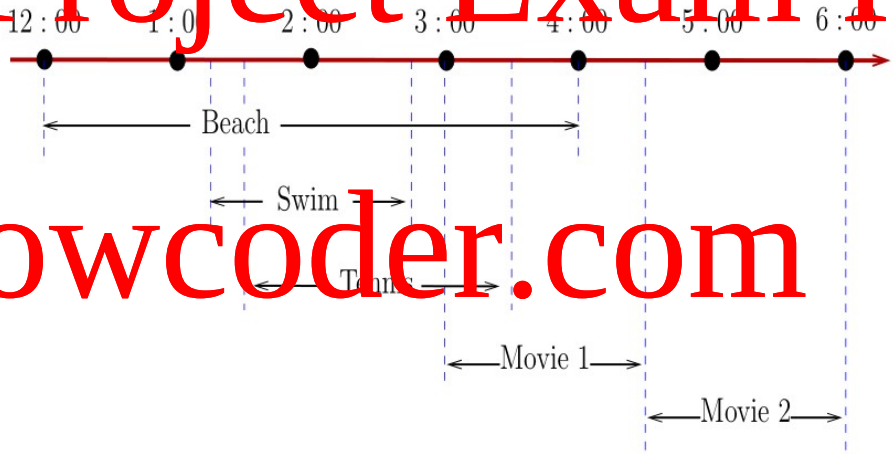
[Start] [End]

III

III. Interval Problems

Introduction to Greedy Approach

- Activities Problem:



beach 12 : 00 – 4 : 00,
swim 1 : 15 – 2 : 45,
tennis 1 : 30 – 3 : 20,
movie 3 : 00 – 4 : 30,
movie 4 : 30 – 5 : 00.

E.g., swimming and tennis are in conflict.

PROBLEM: *Given a set of actitivies,*
find a maximal subset that is conflict-free.

Introduction to Greedy Approach

- Interval Problems

Consider half-open intervals $I_i = [s_i, f_i)$

where $s_i < f_i$ (for $i = 1, \ldots, n$)

It represents time span of the ith activity.

Two activities I_i, I_j conflict if $I_i \cap I_j \neq \emptyset$.

A set $A \subseteq S$ is said to be compatible if no 2 activities in A conflict.

PROBLEM: Given activities $S = \{1, \ldots, n\}$,

find a compatible set of maximum cardinality

Introduction to Greedy Approach

- Generic Greedy Algorithm

GENERIC GREEDY ACTIVITIES SELECTION:
Input: a set A of intervals
Output: $S \subseteq A$, a set of compatible intervals
▷ *Initialization*
Sort A according to some numerical criterion
Let $(I_1, \ldots, I_n)$ be the sorted sequence.
Let $S = \emptyset$.
▷ *Main Loop*
For $i = 1$ to n
If $S \cup \{I_i\}$ is compatible, add I_i to S
Return(S)

Introduction to Greedy Approach

- What Sorting Criteria?

(a) Sort I_i's in order of increasing finish times:
swim, tennis, beach, movie 1, movie 2

(b) Sort I_i's in order of increasing start times:
beach, swim, tennis, movie 1, movie 2

(c) Sort I_i's in order of duration where the duration of activity I_i is $f_i - s_i$. Note that *movie 1*, *movie 2* and *swim* are tied, but breaking ties arbitrarily:
movie 1, movie 2, swim, beach, tennis

(d) Sort I_i's in order of increasing conflict degree.
The conflict degree of I_i is the number of I_j's which conflict with I_i. If the conflict degree of I_i is zero, clearly we can always add I_i into our set. Thus the ordering is:
movie 2, movie 1 or swim, beach or tennis

What are the solutions following these criteria?

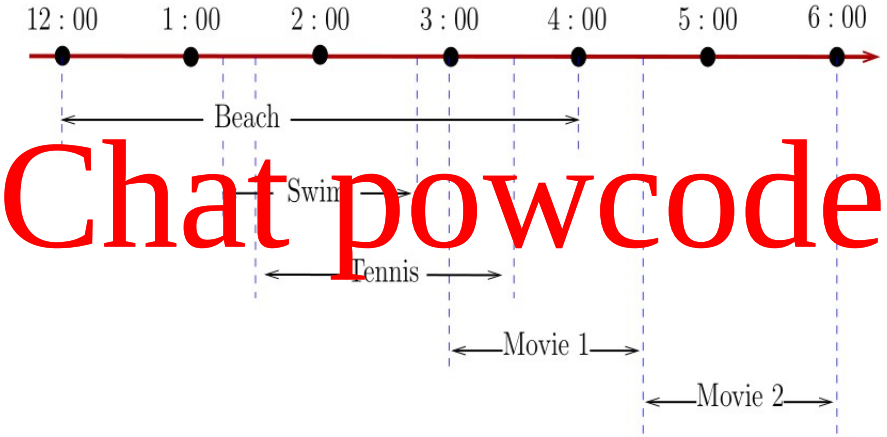
Introduction to Greedy Approach

Sol1: (swim, mov1, mov2)

Sol2: (beach, mov2)

Sol3: (mov1, mov2, swim)

Sol4: (mov2, mov1, swim)



beach	$12:00-4:00$,
swim	$1:15-2:45$,
tennis	$1:30-3:20$,
movie	$3:00-4:30$,
movie	$4:30-6:00$.

Introduction to Greedy Approach

small

THEOREM

Sorting the activities l_i's in order of increasing finish times yields the optimal solution.

How to implement the algorithm?

- *Sorting in $O(n \log n)$ time.*
- *Each $S \cup \{l_i\}$ compatibility test is $O(1)$.*

Extensions: e.g., maximize the time spent on activities.

Give weights to activities: maximize the total weight of solution.

Next...

[Start] [End]

IV

Huffman Code

- Informal Problem

> (P) Given a string s of characters (or symbols) from some alphabet Σ, choose a variable length code C for Σ which minimize the space to encode s.

s can be a file; Σ can be the ascii code

Compare with fixed length encoding, $ASCII : \Sigma \to \{0,1\}^8$

Huffman Code

- Variable Length Encoding

$C : \Sigma \to \{0,1\}^*$

Idea: if a letter $x \in \Sigma$ is more frequent in s, want $C(x)$ to be shorter.

Frequency function: $f_s(x) := \#(x, s)$, number of occurences of x in s.

- Prefix Free Codes:

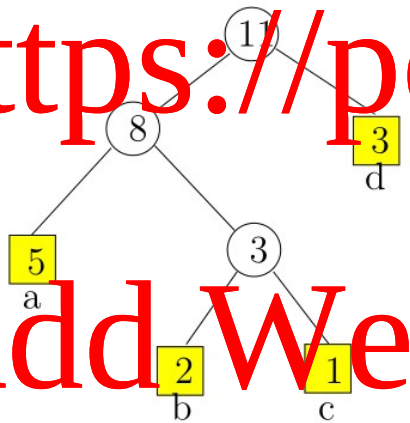
For $x \neq y \in \Sigma$, $C(x)$ is NOT a prefix of $C(y)$.

So if $s = x_1, \ldots, x_n$, then $C(s) = C(x_1)C(x_2)\cdots C(x_n)$ can be uniquely decoded!

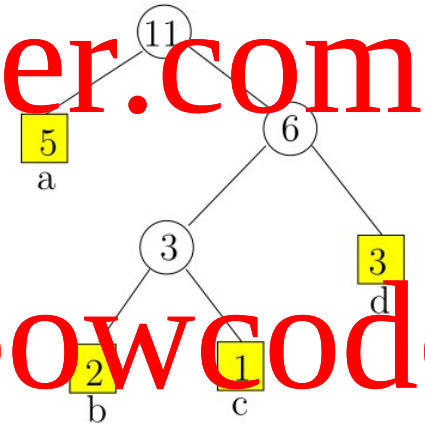
Huffman Code

- Example $s =$ '*abadacadaba*'.
- Then $f_s(a) = 6$, $f_s(b) = 2$, $f_s(c) = 1$, $f_s(d) = 3$.

Two variable codes C_1, C_2 for $\Sigma = \{a, b, c, d\}$ (represented by trees)



COST=11+8+3=22



COST=11+6+3=20

What is the length of this encoding $|C_1(s)|$? Let us calculate it!

$$COST(f, C) := \sum_{x \in \Sigma} |C(x)| \cdot f_s(x)$$

Huffman Code

- The formal Huffman Tree Coding Problem

HC Given $f : \Sigma \to \mathbb{N}$, find an optimal prefix-free code C for f.

The code C is representing by a binary tree T_C whose nodes V_C are binary strings, and each leaf $u \in V_C$ is labeled by a unique symbol $\lambda(u) \in \Sigma$.

Define $W : V_C \to \mathbb{R}$ where $W(u) = \begin{cases} f(\lambda(u)) & u \text{ is a leaf,} \\ W(u_L) + W(u_R) & \text{else.} \end{cases}$

Then $COST(f, C) := \sum_u W(u)$

where u range over internal nodes of T_C.

If T_1, T_2 are two trees, we can merge them by introduce a new root.

Huffman Code

- The Huffman Code Algorithm

HUFFMAN CODE ALGORITHM:
Input: Frequency function $f : \Sigma \to \mathbb{N}$.
Output: Optimal code tree T^* for f.

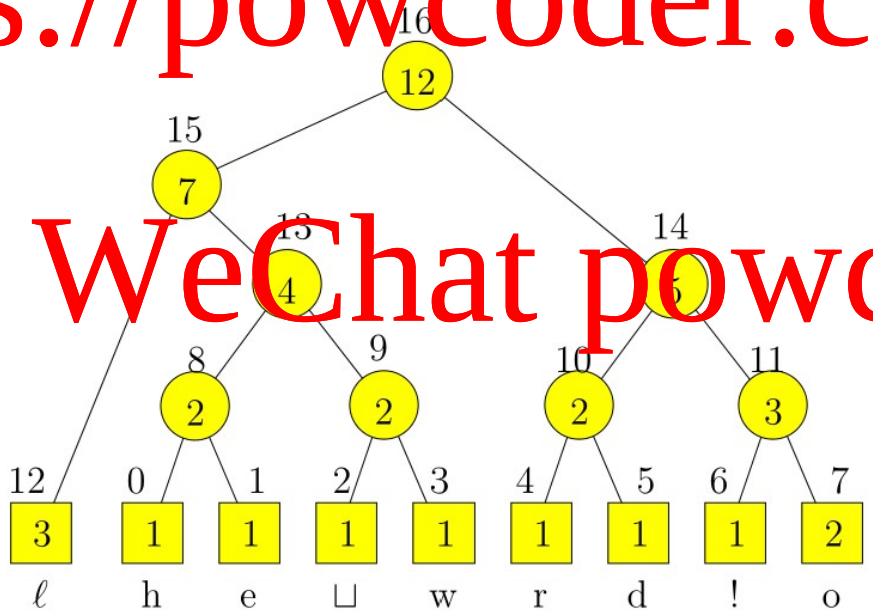
1. Let Q be a set of weighted code trees. Initially, Q is the set of $n = |\Sigma|$ trivial trees, each tree having only one node representing a single character in Σ.
2. While Q has more than one tree
   2.1. Choose $T, T' \in Q$ with the minimum and the next-to-minimum weights, respectively.
   2.2. Merge T, T' and insert the result $T + T'$ into Q.
   2.3. Delete T, T' from Q.
3. Now Q has only one tree T^*. Output T^*.

Note: Q is a priority queue

Huffman Code

- Another Example: $s =$ '*hello* ␣ *world*!'

letter	h	e	l	o	␣	w	r	d	!
frequency	1	1	3	2	1	1	1	1	1



Encoding a Huffman Tree

- Binary Encoding β_T of External Binary Tree T:

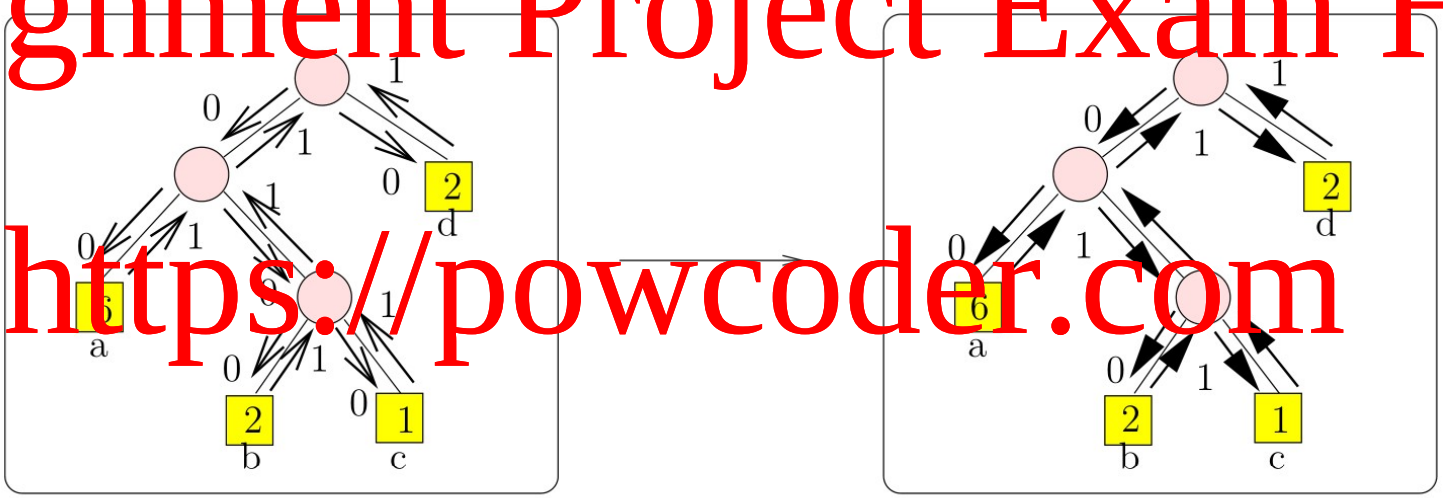


Figure 4: Compressed bit representation for the Huffman tree Figure 2a

Encoding a Huffman Tree

Assume $n \geq 0$ leaves (so $2n-2$ edges)

* Method 1: $4n-4$ bits $(= 2(2n-2))$
* Method 2: $3n-2$ bits $(= (2n-2)+n)$
* Method 3: $2n-1$ bits $(= (3n-2)-(n-1))$
  (replace 01 by 1, for $n-1$ times)

Called the compressed bit representation of T.

These are self-limiting encodings!

E.g., $1, 011, 01011, 00111, 0010111$, etc.

Encoding a Huffman Tree

- Properties β_T, a compressed bit representation
  1. $|\beta_T| = 2n - 1$ with $n - 1$ zeros and n ones.
  2. Any *proper* prefix of β_T has at least as many zeros as ones.
  3. The compressed bit representations forms a prefix-free set.
  4. There is a linear algorithm to check if β_T is valid.

Encoding a Huffman Tree

- Let $C : \Sigma \to \{0,1\}^*$ be a prefix-free code and $\Sigma \subseteq \{0,1\}^N$ for some N. Let T_C be the code tree for C.


```
Protocol to transmit T_C:
  STEP 1:  Transmit the compressed bit representation
     of the shape of T_C
  STEP 2:  Transmit each element of Σ in the in-order
     listing of the leaves of T_C.
```


Total bit bit length: $(2n-1)+N \cdot n$

Encoding a Huffman Tree

- Transmit a string $s \subseteq \Sigma^*$ using static Huffman coding assuming $\Sigma \subseteq \{0,1\}^N$.

(1) Compute the frequency function f_s of s
(2) Compute a Huffman code tree T_C using f_s
(3) STEP 3: Transmit T_C using the above protocol
(4) STEP 4: Compute and transmit $C(s) = C(s_1)C(s_2)\cdots$ using T_C.

ISSUES: need two passes over s – not good if s is very long.

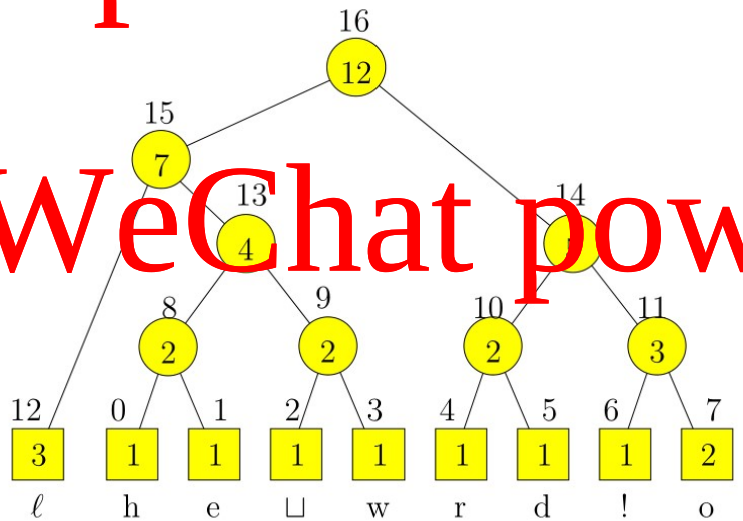
Online Huffman Coding Problem

- Let T be a weighted code tree with $k \geq 0$ internal nodes.

  Assume the nodes are $\{0, 1, \ldots, 2k\}$.

  Call i the **rank** of the node.

  Let node i have weight w_i. E.g.,



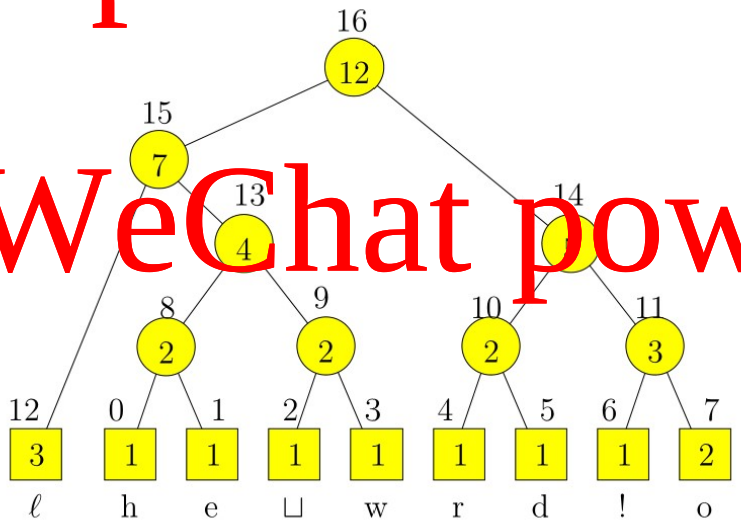
Online Huffman Coding Problem

- Sibling Property

(S0) $w_0 \leq w_1 \leq \cdots \leq w_{2k}$

(S2) $2j$ and $2j+1$ are siblings ($j = 0, \ldots, k-1$)

E.g., $k = 8$



Online Huffman Coding Problem

- Array Representations of T

Rank	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Lc	h	e	␣	w	r	d	!	o	0	2	4	6	ℓ	8	10	12	14
Wt	1	1	1	1	1	1	1	2	2	2	2	3	3	4	5	7	12

Table 2: Compact representation of Huffman tree in Figure 3

* Weight array: `Wt`
* Left-Child array: `Lc`

  `Lc`$[i] = -1$ if no left-child

* Character map array: `Cm`

  where $x \in \Sigma_0 \mapsto \{-1, 0, 1, \ldots, 2k\}$

Online Huffman Coding Problem

How to use `Cm` to compute $C(x)$ (bits in reverse order)

```
C(x):
    u ← Cm[x]
    Output(parity(u))      ◁ parity(u)=1 iff u is odd
    While u > 2k
        u ← parent(u)
        Output(parity(u))
```

Online Huffman Coding Problem

How to restore sibling property after an increment:

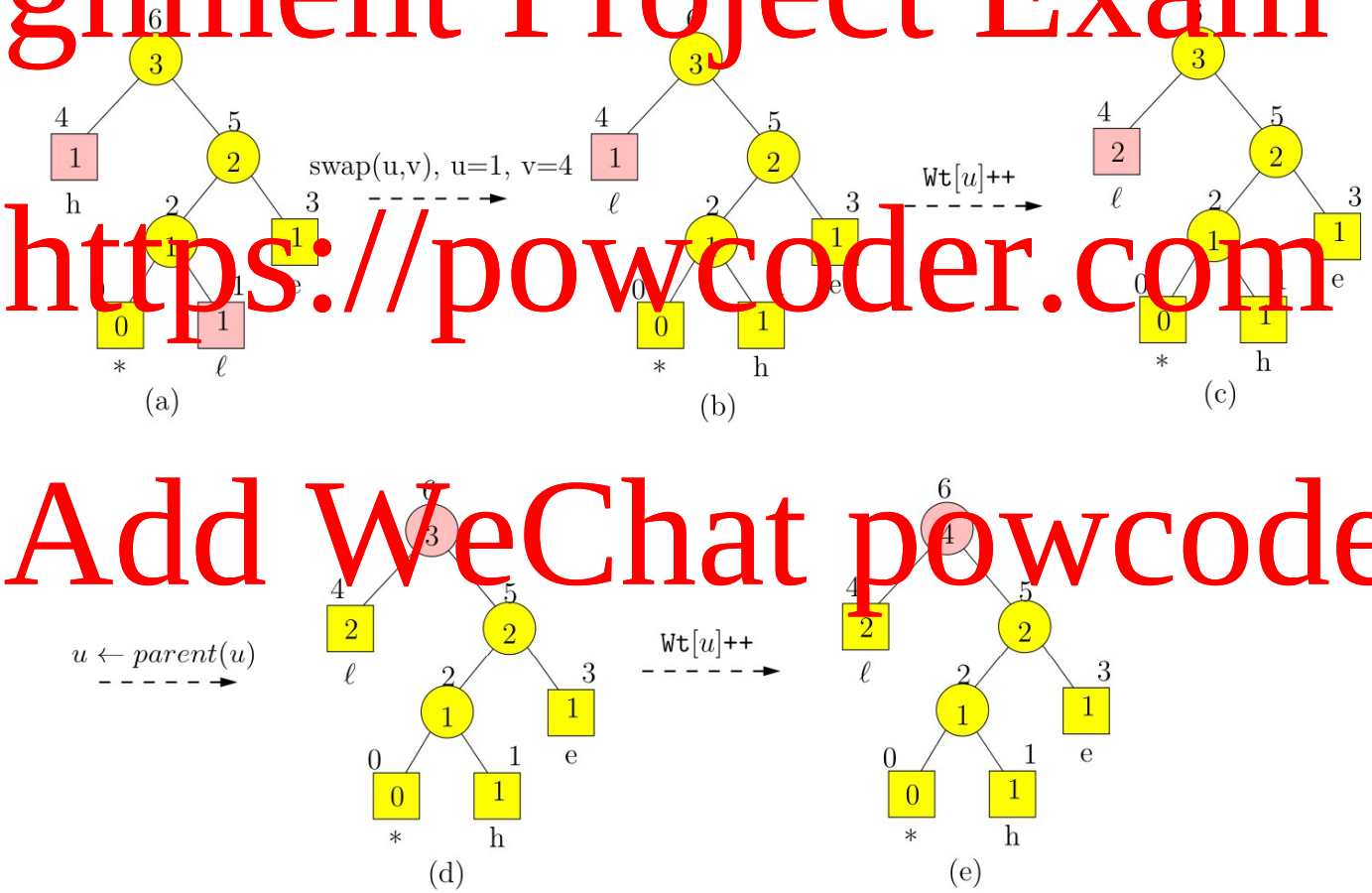


Figure 6: Restoring Huffmanness after incrementing the frequency of letter ℓ

Online Huffman Coding Problem

- Restore(u) (recursive)


```
RESTORE(u)
    ▷ u is a node whose weight is to be incremented
    While (u is not the root) do
1.      ▷ Find node v of largest rank R(v) subject
        ▷ to Wt[v] = Wt[u]. Specifically:
        v ← u
        While (Wt[v+1] = Wt[u])
            v++
2.      If (v ≠ u)
3.          Swap(u, v).   ◁ This swaps the subtrees rooted at u and v.
4.      Wt[u]++.   ◁ Increment the weight of u
5.      u ← parent(u).   ◁ Reset u
6.  Wt[u]++.   ◁ Now, u is the root
```

Online Huffman Coding Problem

Assume the Hello World! Huffman tree.

Now we increment the frequency of '␣' by one:

Operations of Restore (1st potential swap)

Rank	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Lc	n	e	!	w	r	d	␣	o	0	2	4	6	ℓ	8	10	12	15
Wt	1	1	1	1	1	1	1+1	2	2	2	2	3	3	4	5	7	12
After first swap			v				u										

Online Huffman Coding Problem

Operations of Restore (the remaining potential swaps)

Rank	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Lc	h	e	!	w	r	d	␣	o	0	2	4	ℓ	6	8	10	12	15
Wt	1	1	1	1	1	1	1+1	2	2	2	2	3	3+1	4	5	7	12
After second swap												v	u				
Lc	h	e	!	w	r	d	␣	o	0	2	4	ℓ	6	8	10	12	15
Wt	1	1	1	1	1	1	1+1	2	2	2	2	3	3+1	4	5	7+1	12
No third swap																$u = v$	
Lc	h	e	!	w	r	d	␣	o	0	2	4	ℓ	6	8	10	12	15
Wt	1	1	1	1	1	1	1+1	2	2	2	2	3	3+1	4	5	7+1	12+1
No final swap																	$u = v$

Online Huffman Coding Problem

- How to add a new letter: the 0-node

Introduce a unique node of weight 0

It represents all the *yet-unseen* letters in $\Sigma_0 \setminus \Sigma$

When a new letter x is seen, transmit the code of the 0-node, together with the standard code $x \in \{0,1\}^N$

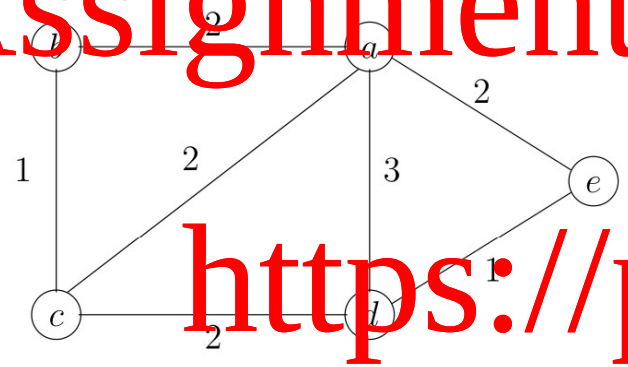
BUT we must increase the ranks of ALL the previous nodes by 2.

Next...

[Start] [End]

V

Minimum Spanning Tree (MST)



Given a graph $G = (V, E; C)$

with cost function $C : E \to \mathbb{R}$

A spanning forest is an acyclic set $T \subseteq E$ of maximal cardinality.

A minimum spanning forest is one of minimum cost.

For simplicity, assume G is connected.

Then we speak of a minimum spanning tree (MST)

$T = \{a{-}b, b{-}c, d{-}e\}$ is acyclic

$T = \{a{-}b, b{-}c, c{-}a, d{-}e\}$ is cyclic

Minimum Spanning Tree (MST)

- Generic Greedy MST Algorithm

Generic Greedy MST Algorithm

Input: $G = (V, E; C)$ a connected bigraph with edge costs.

Output: $S \subseteq E$, a MST for G.

$S \leftarrow \emptyset$.

for $i \leftarrow 1$ to $|V| - 1$ do

1. Greedy Step: find an $e \in E \setminus S$ that is "good for S".
2. $S \leftarrow S + e$.

Output S as the minimum spanning tree.

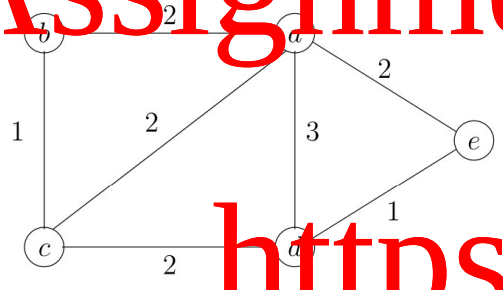
Suppose S is "good"

Problem: find $e \in E \setminus S$ so that e is "good" for S

Necessary for goodness: if $S + e$ is acyclic, call e a candidate

Minimum Spanning Tree (MST)

- Prim's Algorithm



	a	b	c	d	e	$S \subseteq V$	edge
m_0	0	∞	∞	∞	∞	$\emptyset$	
m_1	0	2/a	2/a	3/a	2/a	$\{a\}$	$--$
m_2		2/a	1/b	3/a	2/a	$\{a,b\}$	$a-b$
m_3			1/b	2/c		$\{a,b,c\}$	$b-c$
m_4				2/c	1/d	$\{a,b,c,d\}$	$c-d$
m_5					1/d	$\{a,b,c,d,e\}$	$d-e$

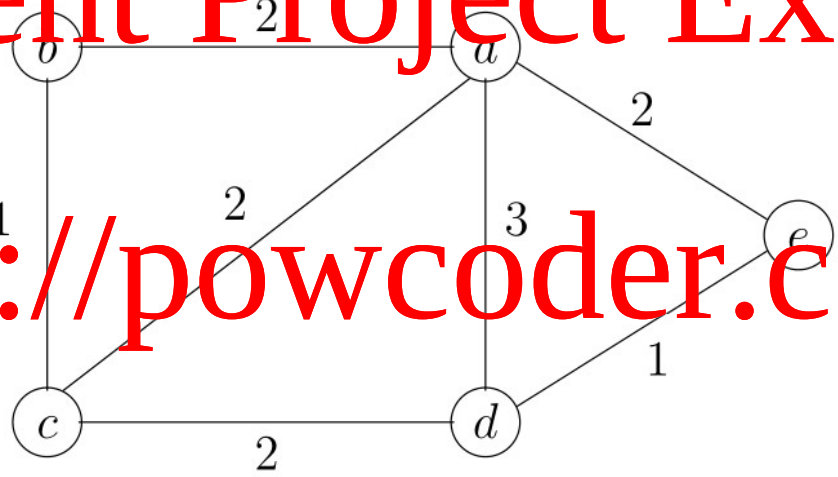
Minimum Spanning Tree (MST)

- Criteria for Goodness

- (Simple) $S + e$ is extendible to some MST.
- (Kruskal) Edge e has the least cost among all the candidates.
- (Boruvka) There is a connected component U of $G' = (V, S)$ such that e has the least cost among all the candidates that extend U. Let us expand this a bit: suppose G' has k components $U_1, \ldots, U_k$. Then for each U_i, there is at least one e_i that extends U_i with least cost. The fact that there are many choices for e can be exploited in parallel algorithms. Note that it is possible that $e_i = e_j$ with $i \neq j$, but we still have at least $\lceil k/2 \rceil$ choices.
- (Prim) This has, in addition to Boruvka's condition, the requirement that the graph $G'' = (V, S + e)$ has only one non-trivial component. [A component is trivial if it has only a single vertex.]

Minimum Spanning Tree (MST)

- Hand Simulation



Kruskal: sort edges $e_1 \leq e_2 \leq \cdots$

Prim: maintain array $d[1..n]$ where $d[i]$ is least cost to connect 1 to i.

Borukva: maintain connected component array $CC[i] = j$ and min-cost $A[CC[i]] = u-v$ extension

Minimum Spanning Tree (MST)

- Boruvka Algorithm

Boruvka's MST Algorithm
Input: $G = (V, E; C)$ a connected graph
Output: MST $S \subseteq E$ of G
$S \leftarrow \emptyset$ ◁ *Initialize a Boruvka-good set of edges*
While $(|S| < n - 1)$
  for each connected component C of G_S ◁ *Do Phase*
    $e \leftarrow \text{argmin}\,\{C(v,u) : v \in C, u \notin C\}$
    $S \leftarrow S + \{e\}$

Maintain connected components $CC[i] = j$

Each phase, number of components is at least halved.

Hence, at most $\lg n$ phases

Minimum Spanning Tree (MST)

- Implementing a Phase of Boruvka in $O(n+m)$ Time
  - Boruvka: maintain connected component array $CC[i] = j$

Matroids

- What is a Matroid?

It is a hypergraph $M = (V, E)$ with V, E non-emety satisfying 2 properties:

(1) Hereditary : if $A \subseteq B \in E$ then $A \in E$. In particular $\emptyset \in E$.

(2) Exchange : if $A, B \in E$ and $|A| \prec |B|$ then there is some $e \in B \setminus A$ such that $A + e \in E$

Matroids

- Matrix Matroids

Let M be a $m \times n$ matrix of real numbers.

Let $C := \{M[1], \ldots, M[n]\}$ be the columns of M.

Let I comprise all linearly independent subset $A \subseteq C$.

Then $M = (C, I)$ is the matroid defined by matrix M.

Recall in Linear Algebra: a set of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ is linearly independent if for all real c_i's,

$\sum_{i=1}^{k} c_i \boldsymbol{v}_i = \boldsymbol{0}$ iff $c_1 = c_2 = \cdots = c_k = 0$.

Matroids

Why is the matrix matroid a matroid?

Clearly hereditary.

Exchange property: if $|A| < |B|$ then $A \cup B$ has rank strictly greater than $|A|$, and so we can find some $b \in B$ so that $A + b$ has rank $|A| + 1$.

E.g., let $M = \begin{bmatrix} 1 & -1 & 2 \\ 0 & 1 & 1 \\ -1 & 0 & -1 \end{bmatrix}$. Then the matroid of M is $H = (\{1,2,3\}, I)$ where $I = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{1,3\}\}$.

Matroids

- Graphic Matroids

Let $G = (V, E)$ be a bigraph.

Let I denote the set of all acyclic subsets of S.

The hypergraph $H = (E, I)$ is a graphic matroid of G.

See Lecture Notes for proof that I has the exchange property.

Matroids

- The bases of a matroid (V, E) are those $A \in E$ that is not properly contained in another independent set.

It follows from exchange property: *all bases have the same cardinality* called the rank of the matroid.

- Given a matroid $(V, E; C)$ with cost function $C : V \to \mathbb{R}_{>0}$, find a base of maximum cost.

Kruskal's algorithm for Maximum base:


```
Sort V in decreasing cost order: v_1 ≥ v_2 ≥ .. ≥ v_n.
Initialize S ← ∅.
For i = 1, ..., n
    Add v_i to S if it preserves independence.
```

"Algebra is generous,
she often gives more than is asked of her." — Jean Le Rond D'Alembert (1717-83)