Greedy Algorithms

Design and Analysis of Algorithms
Greedy algorithms, coin changing problem

Fundamental Techniques

- There are some *algorithmic tools* that are quite specialised. They are good for problems they are intended to solve, but they are not very *versatile*.
- There are also more fundamental (general) *algorithmic tools* that can be applied to a *wide variety* of different data structure and algorithm design problems.

The Greedy Method

- An optimisation problem (OP) is a problem that involves searching through a set of configurations to find one that *minimises* or *maximizes* an objective function defined on these configurations
- The greedy method solves a given OP going through a *sequence of (feasible) choices*
- The sequence starts from well-understood starting configuration, and then iteratively makes the decision that seems best *from all* those that are currently possible.

The Greedy Method

- The greedy approach does *not always* lead to an *optimal solution*.
- The problems that have a greedy solution are said to posses the greedy-choice property.
- The *greedy approach* is also used in the context of hard (*difficult to solve*) problems in order to generate an approximate solution.

What is a greedy algorithm?

- **Greedy algorithm**: "an algorithm always makes the choice that looks best at the moment"
- Human beings use greedy algorithms a lot
  - How to maximize your final grade of this class?
  - How to become a rich man?
  - How does a casher minimize the number of coins to make a change?

What is a greedy algorithm?

- How to maximize your final grade of this class?

MaximizeFinalGrade(quizzes and tests){
  if(no quiz and no test) return;
  DoMyBest(current quiz or test); //Greedy choice
  MaximizeFinalGrade (quizzes and tests – current one);
}

– This algorithm works very well for students
  - Why is it correct?

7

What is a greedy algorithm?

Assuming that your performance of each quiz and test are independent

What if you did your b... in the current quiz?
- You have th... ...ximum final grade
- **The greedy choi... ...f certain optimal solution**

Greedy-choice property!

What if you did not maximize the grades of the rest of the quizzes and tes..?
- You get a l...
- **The optim... ...ain optimal solutions to subpro...ems**

Optimal substructure

8

What is a greedy algorithm?

- To guarantee that a greedy algorithm is correct 2 things have to be proved:
  - **Greedy-choice property**: "we can assemble a globally optimal solution by making locally greedy(optimal) choices."
    - i.e. The greedy choice is always part of certain optimal solution
  - **Optimal substructure**: "an optimal solution to the problem contains within it optimal solutions to subproblems."
    - i.e. global optimal solution is constructed from local optimal solutions

9

What is a greedy algorithm?

- How to become a rich man?
- Problem: maximize the money I have on 30/03/2017 09:00 PM
- A greedy algorithm:

Rich(certain time period P){
  Collect as much money as I can in the current 3 hours; //Greedy choice
  Rich(P-3 hours);
}

10

What is a greedy algorithm?

- if **Rich** is implemented by Dr MAQ
- **Rich** (between now and 30/03/2017 9:00 pm)
- What are the choices I have in the most recent 3 hours?
  - Finish this lecture like all the other instructors
    - Money collected: 0
  - Go to underground, be a beggar, repeatedly say "hey generous man, gimme 100 ! "
    - Money collected: 1000 (since k*100, k<=10)

11

What is a greedy algorithm?

- What are the choices I have in the most recent 3 hours?
  - Rob Dir.
    - Money collected: 0 (got killed by guards)
  - Rob my students
    - Money collected: about 3000

12

What is a greedy algorithm?

- Which one is the greedy choice?
  - Teach algorithms
    - Money collected: 0
  - Be a beggar
    - Money collected: 1000 (since k*100, k<=10)
  - Rob BOA
    - Money collected: 0 (got killed by cops)
  - Rob my students //The greedy choice
    - Money collected: about 3000

13

What is a greedy algorithm?

- What happened if I robbed you?
  - Students
    - Report the criminal immediately
    - Or report it after your final
  - The instructor
    - Cops confiscate the illicit 3000, i.e. -3000
    - Get fired, and lose the salary of this month, i.e. about -200 K
- After making this greedy choice, what is the result of **Rich**

14

What is a greedy algorithm?

Rich (between no [illegible]

Collect as much [illegible] ours;

Rich (between 6pm [illegible]);

}

Fail to achieve the optimal solution!

- Greedy choice: 3000
- However there is a influence on the optimal solution to the sub-problem, which prevents the instructor from arriving the richest solution :
  - the best of **Rich** (between 6pm today and 30/03/2017 9 pm) will be around -203K

15

What is a greedy algorithm?

- Why the greedy **Rich** algorithm does not work?
  - After robbing you, I have no chance to be to get the richest solution
  - i.e. the **greedy choice property** is violated

16

What is a greedy algorithm?

- How to become a rich man?
  - +10000 Teach algorithms
    - Money collected: 0
  - Got fired 1000 -200K Be a beggar
    - Money collected: 1000 (since k*100, k<=10)
  - Got killed -infinity Rob Dir
    - Money collected: 0 (got killed by guards)
  - Rob my students
    - Money collected: about 3000

In this problem, we do not have greedy property

So, greedy choice does not help

And it is very consistent with what you see now

17

Coin changing problem

- An example:
  - A hot dog and a drink Costs 468
  - Plus tax it is: 468+100 = 568
  - Often, we give the cashier 500 + 100 notes
  - He/She need to give back, 38 rupees as change
- Generally, you never see he/she gives you 38 rupees of coinage 1.
- What is algorithm here?

18

Coin changing problem

- Coin changing problem (informal):
  - Given certain amount of change: n cents
  - The denominations of coins are: 25, 10, 5, 1
  - How to use the fewest coins to make this change?
- i.e. n = 25a + 10b + 5c + d, what are the a, b, c, and d, minimizing (a+b+c+d)
- Can you design an algorithm to solve this problem?

19

Coin changing problem

- n = 25a + 10b + 5c + d, what are the a, b, c, and d, minimizing (a+b+c+d)
- How to do it in brute-force?
  - At most we use n pennies
  - Try all the combinations where a<=n, b<=n, c<=n, d<=n
  - Choose all the combinations that n = 25a + 10b + 5c + d
  - Choose the combination with smallest (a+b+c+d)

How many combinations? $\Theta(n^4)$

Time complexity is $\Theta(n^4)$

20

Coin changing problem

- n = 25a + 10b + 5c + d, what are the a, b, c, and d, minimizing (a+b+c+d)
- How to do it in divide-and-conquer?

coinD&C(n){

1. if(n<5) return (a=0, b=0, c=0, d=n, n)
2. if(n==5) return (a=0, b=0, c=1, d=0, 1)
3. if(n==10) return (a=0, b=1, c=0, d=0, 1)
4. if(n==25) return (a=1, b=0, c=0, d=0, 1)
5. s= $\min($**coinD&C**$(n-25),$ **coinD&C**$(n-10),$ **coinD&C**$(n-5),$ **coinD&C**$(n-1))$;
6. Increase s.a or s.b or s.c or s.d by 1 according to the coin used in the minimum one
7. return (s.a, s.b, s.c, s.sum+1);

What is the recurrence equation? $T(n) = T(n-25) + T(n-10) + T(n-5) + T(n-1) + 1$

Time complexity? $T(n) \le 4T(n-1)$ and $T(n) \ge 4T(n-25)$, $T(n) = O(4^n) = \Omega(4^{n/25})$

21

Coin changing problem

- n = 25a + 10b + 5c + d, what are the a, b, c, and d, minimizing (a+b+c+d)
- How to do it in dynamic programming?

coinDP(n){

1. **If(solution for n in memo) return memo(n);**
2. if(n<5) return (a=0, b=0, c=0, d=n, n)
3. if(n==5) return (a=0, b=0, c=1, d=0, 1)
4. if(n==10) return (a=0, b=1, c=0, d=1, 1)
5. if(n==25) return (a=1, b=0, c=0, d=1, 1)
6. s= $\min($**coinDP**$(n-25),$ **coinDP**$(n-10),$ **coinDP**$(n-5),$ **coinDP**$(n-1))$;
7. Increase s.a or s.b or s.c or s.d by 1 according to the coin used in the minimum one
8. **Put (s.a, s.b, s.c, s.sum+1) in memo as memo(n);**
9. return (s.a, s.b, s.c, s.sum+1);

How many sub problems? n

If subproblems are solved, how much time to solve a problem? $\Theta(1)$

Time complexity? $T(n)=\Theta(n)$

22

Coin changing problem

- n = 25a + 10b + 5c + d, what are the a, b, c, and d, minimizing (a+b+c+d)
- How to do it by a greedy algorithm?

coinGreedy(n){

if(n>=25) s = **coinGreedy**(n-25); s.a++;
else if(n>=10) s = **coinGreedy**(n-10); s.b++;
else if(n>=5) s = **coinGreedy**(n-5); s.c++;
else s=(a=0, b=0, c=0, d=n, sum=n);
s.sum++;
return s;

Greedy choice
Always choose the possible largest coin

It that greedy algorithm correct?

Time complexity? $T(n)=\Theta(n)$

If n is large, in most of the subproblems it chooses quarter, so it is much faster than dynamic programming $T(n) \approx C(\frac{n}{25})$ and in DP $T(n) \approx Cn$

23

Coin changing problem

- Optimal substructure
  - After the greedy choice, assuming the greedy choice is correct, can we get the optimal solution from sub optimal result?
    - 38 cents
    - Assuming we have to choose 25
    - Is a quarter + optimal **coin**(38-25) the optimal solution of 38 cents?
- Greedy choice property
  - If we do not choose the largest coin, is there a better solution?

24

Coin changing problem

- For coin denominations of 25, 10, 5, 1
  - The greedy choice property is not violated
- For other coin denominations
  - May violate it
  - E.g. 10, 7, 1
  - 15 cents
- How to prove the greedy choice property for denominations 25, 10, 5, 1?
  - Optimal structure --- easy to prove
  - Greedy choice property

Coin changing problem

- 1. Prove that with coin denominations of “5, 1”, it has the greedy choice property
- Proof:

  Apply greedy choice: n = 5 + 5c + d

  In a optimal solution if there is a nickel, the proof is done

  If there is no nickel: n = d’=5 + d’’

  Need to prove that: 1+d’’ <= d’

  d’=5+d’’ > 1+d’’
- For “5, 1”, it has greedy choice property, greedy algorithm works

Coin changing problem

- 2. Prove that with coin denominations of “10, 5, 1”, it has the greedy choice property
- Proof:

  Apply greedy choice: n = 10 + 10b + 5c + d
  - In a optimal solution if there is a dime, the proof is done
  - If there is no dime : n = 5c’ + d’
    - Since 5c’ + d’>=10
    - with the conclusion of the previous slide, c’>=2
    - 5c’ + d’ = 10 + 5(c’-2) + d’ and c’+d’ > 1+c’-2+d’
    - it cannot be a optimal solution
- For “10, 5, 1”, it has greedy choice property, greedy algorithm works

Coin changing problem

- 3. Prove that with coin denominations of “25, 10, 5, 1”, it has the greedy choice property
- Proof:

  Apply greedy choice: n = 25 + 25a + 10b + 5c + d
  - In a optimal solution if there is a quarter, the proof is done
  - If there is no quarter : n = 10b’+5c’+d’
    - Since 10b’+5c’+d’ >= 25
    - if 25<=n<30, with the conclusion of previous slide, b’>=2, c’>=1
      - 10b’+5c’+d’ = 25 + 10(b’-2) + 5(c’-1) + d’ and b’+c’+d’>1+b’-2+c’-1+d’
      - it cannot be a optimal solution
    - if n>=30, with the conclusion of previous slide, b’>=3
      - 10b’+5c’+d’ = 25 + 10(b’-3) + 5(c’+1) + d’ and b’+c’+d’>1+b’-3+c’+1+d’
      - it cannot be a optimal solution

For “25, 10, 5, 1”, it has greedy choice property, greedy algorithm works

Greedy Algorithms

Coming up

- Casual Introduction: Two Knapsack Problems
- An Activity-Selection Problem
- Greedy Algorithm Design
- Huffman Codes

(Chap 16.1-16.3)

2 Knapsack Problems

1. 0-1 Knapsack Problem:

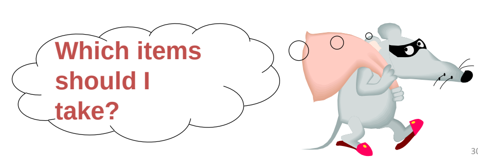
A thief robbing a store finds n items.

i^{th} item: worth v_i dollars

w_i pounds

W, w_i, v_i are integers.

He can carry at most W pounds.



2 Knapsack Problems

2. Fractional Knapsack Problem:

A thief robbing a store finds n items.

i^{th} item: worth v_i dollars
w_i pounds

W, w_i, v_i are integers.

He can carry at most W pounds.

He can take fractions of items.

2 Knapsack Problems

By Greedy Strategy

Both problems are similar. But Fractional Knapsack Problem can be solved in a greedy strategy.

Step 1. Compute the value per pound for each item
Eg. gold dust: $10000 per pound **(most expensive)**
Silver dust: $2000 per pound
Copper dust: $500 per pound

Step 2. Take as much as possible of the most expensive (ie. Gold dust)

Step 3. If the supply of that item is exhausted (ie. no more gold) and he can still carry more, he takes as much as possible of the item that is next most expensive and so forth until he can't carry any more.

Knapsack Problems

By Greedy Strategy

We can solve the Fractional Knapsack Problem by a greedy algorithm:

Always makes the choice that looks best at the moment.

ie. A locally optimal Choice

To see why we can't solve 0-1 Knapsack Problem by greedy strategy, read Chp 16.2.

Activity-Selection Problem

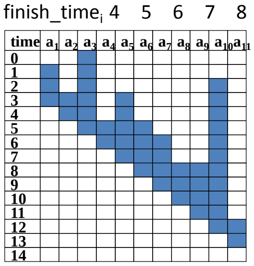
For a set of proposed activities that wish to use a lecture hall, select a maximum-size subset of "compatible activities".

- Set of activities: $S=\{a_1,a_2,\ldots a_n\}$
- Duration of activity a_i: $[start\_time_i, finish\_time_i)$
- Activities ***sorted in increasing order of finish time***:

i	1	2	3	4	5	6	7	8	9	10	11
$start\_time_i$	1	3	0	5	3	5	6	8	8	2	12
$finish\_time_i$	4	5	6	7	8	9	10	11	12	13	14

Activity-Selection Problem

i	1	2	3	4	5	6	7	8	9	10	11
$start\_time_i$	1	3	0	5	3	5	6	8	8	2	12
$finish\_time_i$	4	5	6	7	8	9	10	11	12	13	14

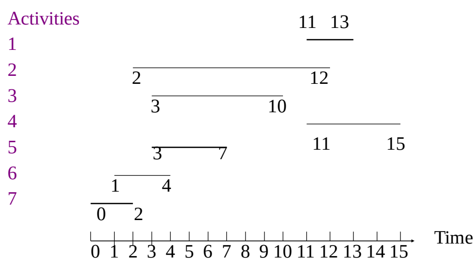


Compatible activities:
$\{a_3, a_9, a_{11}\}$,
$\{a_1,a_4,a_8,a_{11}\}$,
$\{a_2,a_4,a_9,a_{11}\}$

Activity Selection

- **Given a set *S* of *n activities* with *start* time s_i and *finish* time f_i of activity *i***
- **Find a maximum size subset *A* of compatible activities (maximum *number* of activities).**
- **Activities are compatible if they do not overlap**
- **Can you suggest a greedy choice?**

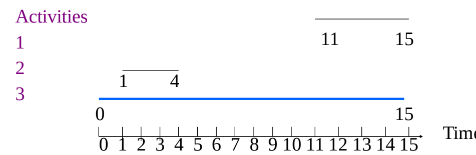
Example



45

Counter Example 1

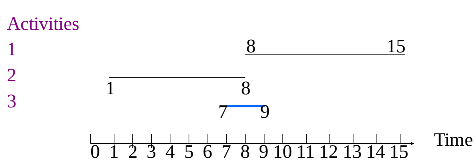
- Select by start time



46

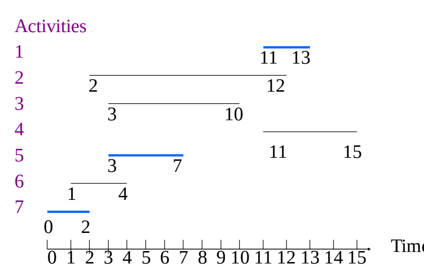
Counter Example 2

- Select by minimum duration



47

Select by finishing time



48

Activity Selection

- **Assume without loss of generality that we number the intervals in order of *finish time*. So $f_1 \leq \ldots \leq f_n$.**
- **Greedy choice: choose activity with minimum finish time**
- **The following greedy algorithm starts with A={1} and then adds all compatible jobs. (Theta(n))**
- **Theta(nlogn) when including sort**

49

Greedy-Activity-Selector(s,f)

```
n <- length[s] // number of activities
A <- {1}
j <- 1 //last activity added
for i <- 2 to n //select
  if s_i >= f_j then //compatible (feasible)
    add {i} to A
    j <- i //save new last activity
return A
```

50

Proof that greedy solution is optimal

1. It is easy to see that there is an **optimal solution** to the problem that makes the greedy choice.

Proof of 1.

Let *A* be an optimal solution. Let activity 1 be the greedy choice. If 1 ∈ *A* the proof is done. If 1 ∉ *A*, we will show that *A'*=*A*-{*a*}+{1} is another optimal solution that includes 1.

Let *a* be the activity with minimum finish time in *A*.

Since activities are sorted by finishing time in the algorithm, $f(1) \le f(a)$. If $f(1) \le s(a)$ we could add 1 to *A* and it could not be optimal. So $s(1) < f(a)$, and 1 and *a* overlap. Since $f(1) \le f(a)$, if we remove *a* and add 1 we get another compatible solution *A'*=*A*-{*a*}+{1} and |*A'*|=|*A*|

Proof that greedy solution is optimal

2. If we combine the optimal solution of the remaining subproblem with the greedy choice we have an optimal solution to the original problem.

Proof of 2.

Let activity 1 be the greedy choice.

Let *S'* be the subset of activities that do not overlap with 1.

$S'=\{i \mid i=1,\dots,n \text{ and } s_i \ge f(1)\}$.

Let *B* be an optimal solution for *S'*.

From the definition of *S'*, *A'*={1}+*B* is compatible, and a solution to the original problem.

Proof that greedy solution is optimal

2. If we combine the optimal solution of the remaining subproblem with the greedy choice we have an optimal solution to the original problem.

Proof of 2 continued.

The proof is by contradiction.

Assume that *A'* is not an optimal solution to the original problem.

Let *A* be an optimal solution that contains 1.

So |*A'*|<|*A*|, and |*A*-{1}|>|*A'*-{1}|=|*B*|.

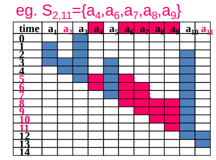
But *A*-{1} is also a solution to the problem of *S'*, contradicting the assumption that *B* is an optimal solution to *S'*.

Activity-Selection Problem

Greedy Strategy Solution

$$c(i,j) = \begin{cases} 0 & \text{if } S_{i,j}=\varnothing \\ \max_{i<k<j} \{c[i,k]+c[k,j]+1\} & \text{if } S_{i,j}\ne\varnothing \end{cases}$$

eg. $S_{2,11}=\{a_4,a_6,a_7,a_8,a_9\}$



Consider any nonempty subproblem $S_{i,j}$, and let a_m be the activity in $S_{i,j}$ with the earliest finish time.

Then,

1. A_m is used in some maximum-size subset of compatible activities of $S_{i,j}$.
2. The subproblem $S_{i,m}$ is empty, so that choosing a_m leaves the subproblem $S_{m,j}$ as the only one that may be nonempty.

Among $\{a_4,a_6,a_7,a_8,a_9\}$, a_4 will finish earliest

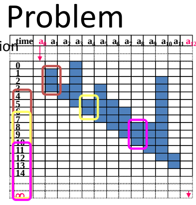
1. A_4 is used in the solution
2. After choosing A_4, there are 2 subproblems: $S_{2,4}$ and $S_{4,11}$. But $S_{2,4}$ is empty. Only $S_{4,11}$ remains as a subproblem.

Activity-Selection Problem

Greedy Strategy Solution

Hence, to solve the $S_{i,j}$:

1. Choose the activity a_m with the earliest finish time.
2. Solution of $S_{i,j}$ = $\{a_m\}$ U Solution of subproblem $S_{m,j}$



That is,

To solve $S_{0,12}$, we select a_1 that will finish earliest, and solve for $S_{1,12}$.

To solve $S_{1,12}$, we select a_4 that will finish earliest, and solve for $S_{4,12}$.

To solve $S_{4,12}$, we select a_8 that will finish earliest, and solve for $S_{8,12}$.

...

Greedy Choices (Locally optimal choice)

To leave as much opportunity as possible for the remaining activities to be scheduled.

Solve the problem in a top-down fashion

Activity-Selection Problem

Greedy Strategy Solution

```
Recursive-Activity-Selector(i,j)
   m = i+1
   // Find first activity in S_i,j
   while m < j and start_time_m < finish_tim
      do m = m + 1
   if m < j
      then return {a_m} U Recursive-Activity-Selector(m,j)
   else return Ø
```

Order of calls:

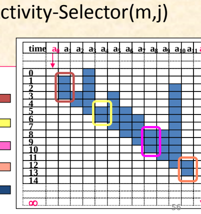
{1,4,8,11} Recursive-Activity-Selector(0,12)

{4,8,11} Recursive-Activity-Selector(1,12)

{8,11} Recursive-Activity-Selector(4,12)

{11} Recursive-Activity-Selector(8,12)

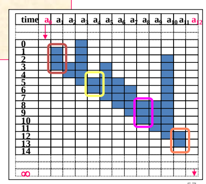
Ø Recursive-Activity-Selector(11,12)



Activity-Selection Problem

Greedy Strategy Solution

```
Iterative-Activity-Selector()
Answer = {a_1}
last_selected=1
for m = 2 to n
    if start_time_m>=finish_time_last_selected
    then Answer = Answer U {a_m}
          last_selected = m
return Answer
```



Activity-Selection Problem

Greedy Strategy Solution

For both Recursive-Activity-Selector and Iterative-Activity-Selector,
Running times are $\Theta(n)$
Reason: each a_m are examined once.

Greedy Algorithm Design

Steps of Greedy Algorithm Design:

1. Formulate the optimization problem in the form: we make a choice and we are left with one subproblem to solve.
2. Show that the greedy choice can lead to an optimal solution, so that the greedy choice is always safe.
3. Demonstrate that an optimal solution to original problem = greedy choice + an optimal solution to the subproblem

Optimal Substructure Property

Greedy-Choice Property

A good clue that that a greedy strategy will solve the problem.

Greedy Algorithm Design

Comparison:

Dynamic Programming	Greedy Algorithms
At each step, the choice is determined based on solutions of subproblems.	At each step, we quickly make a choice that currently looks best. --A local optimal (greedy) choice.
Sub-problems are solved first.	Greedy choice can be made first before solving further sub-problems.
Bottom-up approach	Top-down approach
Can be slower, more complex	Usually faster, simpler

Huffman Coding

Encoding and Compression of Data

- Fax Machines
- ASCII
- Variations on ASCII
  - min number of bits needed
  - cost of savings
  - patterns
  - modifications

Purpose of Huffman Coding

- Proposed by Dr. David A. Huffman in 1952
  - *"A Method for the Construction of Minimum Redundancy Codes"*
- Applicable to many forms of data transmission
  - Our example: text files

The Basic Algorithm

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency!
- Yet all characters are allocated the same amount of space
  - 1 char = 1 byte, be it e or X

The Basic Algorithm

- Any savings in tailoring codes to frequency of character?
- Code word lengths are no longer fixed like ASCII.
- Code word lengths vary and will be shorter for the more frequently used characters.

The (Real) Basic Algorithm

1. Scan text to be compressed and tally occurrence of all characters.
2. Sort or prioritize characters based on number of occurrences in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.

Building a Tree
Scan the original text

- Consider the following short text:

Eerie eyes seen near lake.

- Count up the occurrences of all characters in the text

Building a Tree
Scan the original text

Eerie eyes seen near lake.
- What characters are present?

E e r i space
y s n a r l k .

Building a Tree
Scan the original text

Eerie eyes seen near lake.
- What is the frequency of each character in the text?

Char	Freq.	Char	Freq.	Char	Freq.
E	1	y	1	k	1
e	8	s	2	.	1
r	2	n	2		
i	1	a	2		
space	4	l	1		

Building a Tree
Prioritize characters

- Create binary tree nodes with character and frequency of each character
- Place nodes in a priority queue
  - The lower the occurrence, the higher the priority in the queue

Building a Tree
Prioritize characters

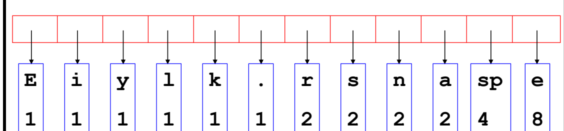
- Uses binary tree nodes

```
public class HuffNode
{
    public char myChar;
    public int myFrequency;
    public HuffNode myLeft, myRight;
}

priorityQueue myQueue;
```

Building a Tree

- The queue after inserting all nodes

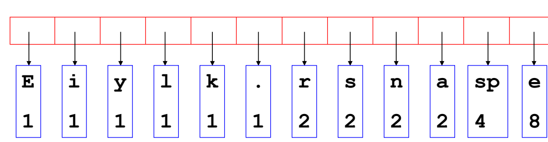


- Null Pointers are not shown

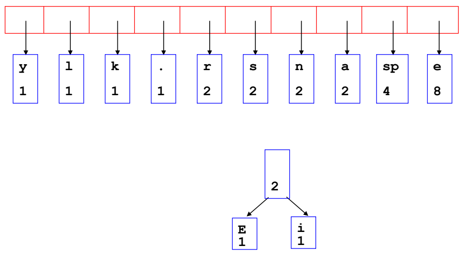
Building a Tree

- While priority queue contains two or more nodes
  - Create new node
  - Dequeue node and make it left subtree
  - Dequeue next node and make it right subtree
  - Frequency of new node equals sum of frequency of left and right children
  - Enqueue new node back into queue

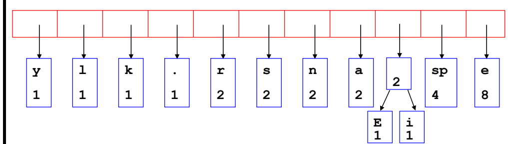
Building a Tree



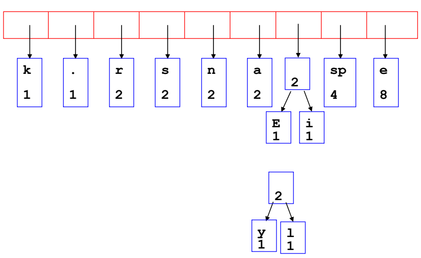
Building a Tree



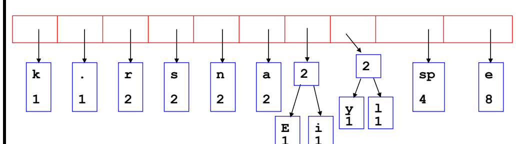
Building a Tree



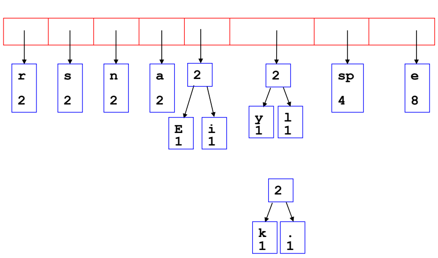
Building a Tree



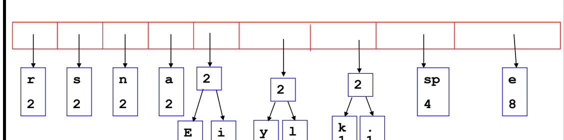
Building a Tree



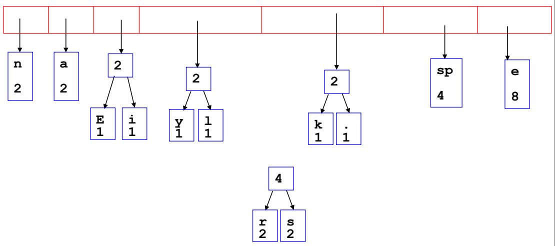
Building a Tree



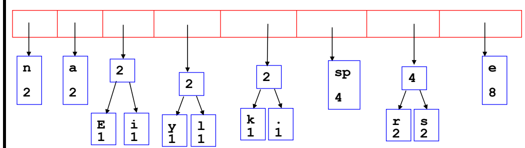
Building a Tree



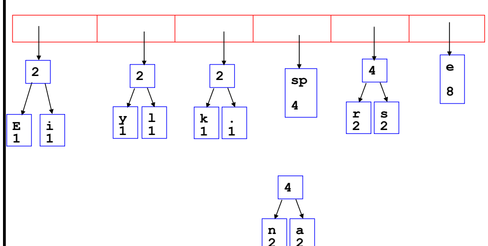
Building a Tree



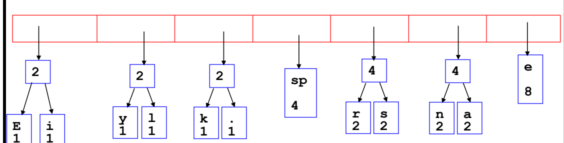
Building a Tree



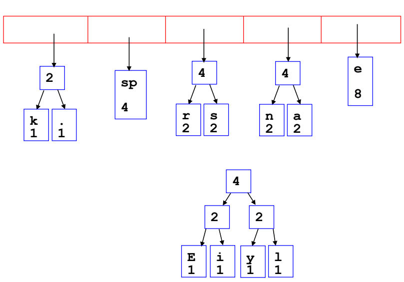
Building a Tree



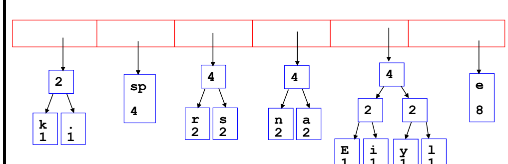
Building a Tree



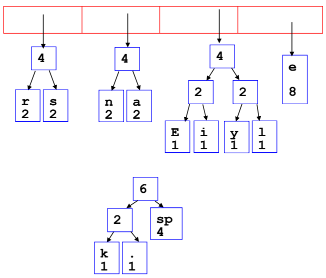
Building a Tree



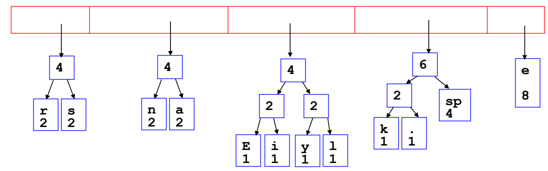
Building a Tree



Building a Tree

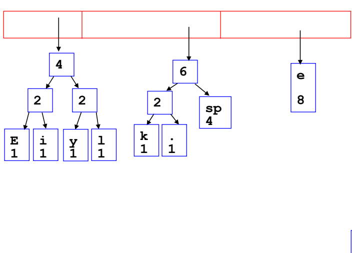


Building a Tree

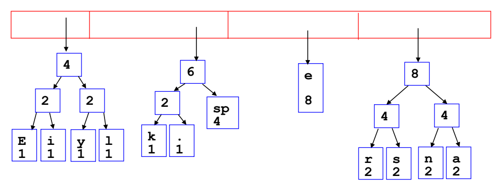


What is happening to the characters with a low number of occurrences?

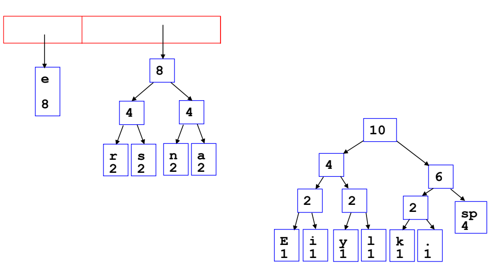
Building a Tree



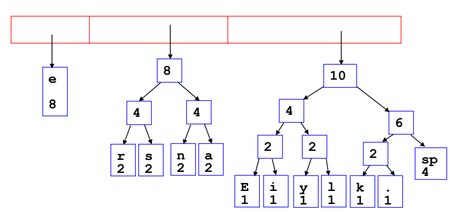
Building a Tree



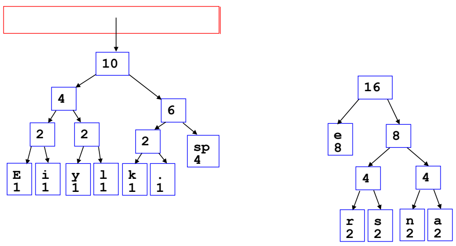
Building a Tree



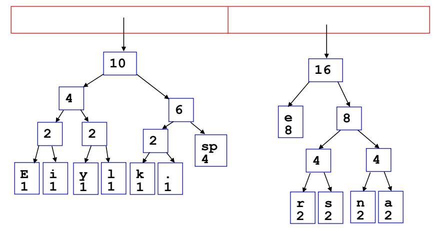
Building a Tree



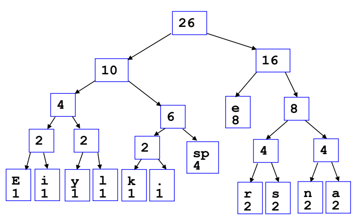
Building a Tree



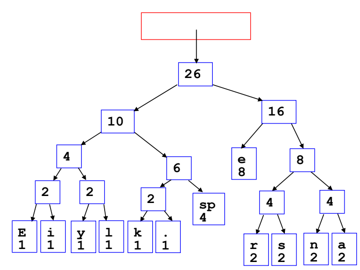
Building a Tree



Building a Tree



Building a Tree



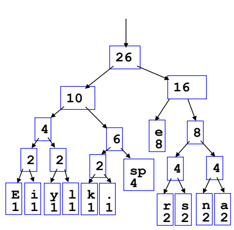
After enqueueing this node there is only one node left in priority queue.

Building a Tree

Dequeue the single node left in the queue.

This tree contains the new code words for each character.

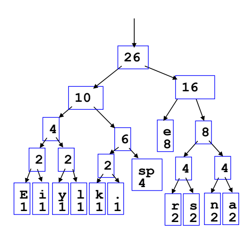
Frequency of root node should equal number of characters in text.



Eerie eyes seen near lake. 🗐 26 characters

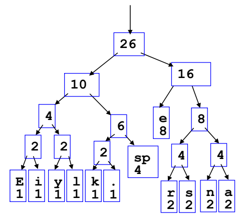
Encoding the File
Traverse Tree for Codes

- Perform a traversal of the tree to obtain new code words
- Going left is a 0 going right is a 1
- code word is only completed when a leaf node is reached



Encoding the File
Traverse Tree for Codes

Char	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
space	011
e	10
r	1100
s	1101
n	1110
a	1111



Encoding the File

- Rescan text and encode file using new code words

Eerie eyes seen near lake.

```
000010110000011001110001010110110100
111110101111110001100111111010010010
1
```

Char	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
space	011
e	10
r	1100
s	1101
n	1110
a	1111

- **Why is there no need for a separator character?**

.

Encoding the File
Results

- Have we made things any better?
- 73 bits to encode the text
- ASCII would take 8 * 26 = 208 bits

```
0000101100000110011
1000101011011010011
1110101111110001100
1111110100100101
```

- **If modified code used 4 bits per character are needed. Total bits 4 * 26 = 104. Savings not as great.**

Decoding the File

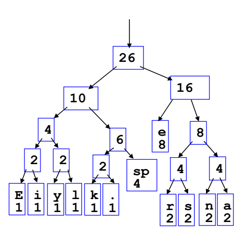
- How does receiver know what the codes are?
- Tree constructed for each text file.
  - Considers frequency for each file
  - Big hit on compression, especially for smaller files
- Tree predetermined
  - based on statistical analysis of text files or file types
- Data transmission is bit based versus byte based

CS 102

Decoding the File

- Once receiver has tree it scans incoming bit stream
- 0 $\Rightarrow$ go left
- 1 $\Rightarrow$ go right

```
101000110111101111
01111110000110101
```



Summary

- Huffman coding is a technique used to compress files for transmission
- Uses statistical coding
  - more frequently used symbols have shorter code words
- Works well for text and fax transmissions
- An application that uses several data structures

Huffman Codes

Huffman Codes
- For compressing data (sequence of characters)
- Widely used
- Very efficient (saving 20-90%)
- Use a table to keep frequencies of occurrence of characters.
- Output binary string.

"Today's weather is nice" ⇒  ⇒ "001 0110 0 0 100 1000 1110"

Huffman Codes

Example:

A file of 100,000 characters. Containing only 'a' to 'e'

	Frequency	Fixed-length codeword	Variable-length codeword
'a'	45000	000	0
'b'	13000	001	101
'c'	12000	010	100
'd'	16000	011	111
'e'	9000	100	1101
'f'	5000	101	1100

eg. "abc" = "000001010"

bits

eg. "abc" = "0101100"

1*45000 + 3*13000 + 3*12000 + 3*16000 + 4*9000 + 4*5000
= bits

Huffman Codes

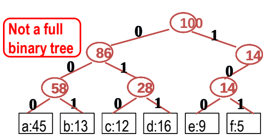
A file of 100,000 characters.

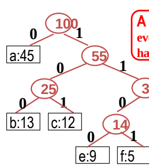
The coding schemes can be represented by trees:

	Frequency (in thousands)	Fixed-length codeword
'a'	45	000
'b'	13	001
'c'	12	010
'd'	16	011
'e'	9	100
'f'	5	101

	Frequency (in thousands)	Variable-length codeword
'a'	45	0
'b'	13	101
'c'	12	100
'd'	16	111
'e'	9	1101
'f'	5	1100

Not a full binary tree

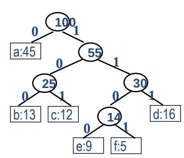




A full binary tree
every nonleaf node has 2 children

Huffman Codes

	Frequency	Codeword
'a'	45000	0
'b'	13000	101
'c'	12000	100
'd'	16000	111
'e'	9000	1101
'f'	5000	1100



Eg. "abc" is coded as "0101100"

To find an **optimal code** for a file:

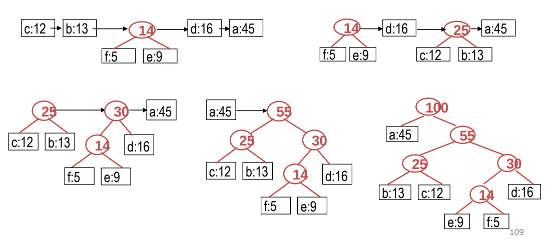
1. The coding must be unambiguous.
   Consider codes in which no codeword is also a prefix of other codeword. => Prefix Codes
   Prefix Codes are unambiguous.
   Once the codewords are decided, it is easy to compress (encode) and decompress (decode).
2. File size must be smallest.
   => Can be represented by a full binary tree.
   => Usually less frequent characters are at bottom
   Let C be the alphabet (eg. C={'a','b','c','d','e','f'})
   For each character c, no. of bits to encode all c's occurrences = $freq_c$*$depth_c$
   File size $B(T) = \Sigma_{c \in C} freq_c * depth_c$

Huffman Codes

How do we find the optimal prefix code?

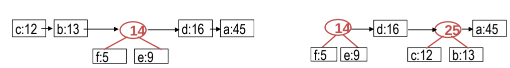
Huffman code (1952) was invented to solve it.
A Greedy Approach.

Q: A min-priority queue f:5 → e:9 → c:12 → b:13 → d:16 → a:45



Huffman Codes

Q: A min-priority queue f:5 → e:9 → c:12 → b:13 → d:16 → a:45



….

```
HUFFMAN(C)
Build Q from C
For i = 1 to |C|-1
      Allocate a new node z
      z.left = x = EXTRACT_MIN(Q)
      z.right = y = EXTRACT_MIN(Q)
      z.freq = x.freq + y.freq
      Insert z into Q in correct position.
Return EXTRACT_MIN(Q)
```

If Q is implemented as a binary min-heap,
"Build Q from C" is O(n)
"EXTRACT_MIN(Q)" is O(lg n)
"Insert z into Q" is O(lg n)
Huffman(C) is O(n lg n)

How is it "greedy"?

Greedy Algorithms

Summary

- Casual Introduction: Two Knapsack Problems
- An Activity-Selection Problem
- Greedy Algorithm Design
  - Steps of Greedy Algorithm Design
  - Optimal Substructure Property
  - Greedy-Choice Property
  - Comparison with Dynamic Programming
- Huffman Codes