Algorithm Design, Analysis & Complexity

Lecture 2 - Greedy Algorithm

Koushik Pal

University of Toronto

May 11, 2021

Greedy Algorithm

A greedy algorithm for an optimization problem is one that makes a locally optimal choice (a choice that looks best at the moment) at every step hoping it leads to a globally optimal solution.

Indication that greedy algorithm might apply to a problem:

Definition

A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.

Techniques of proving that a greedy algorithm works:

- Stays ahead approach
- Exchange argument approach
- Lower bound approach

First Example - Interval Scheduling

Problem (Interval Scheduling)

Given $R = \{r_1, r_2, \ldots, r_n\}$ *a set of* n *requests, with each request specifying a start time* s_i *and a finish time* t_i*, select a subset* $A \subseteq R$ *of* **mutually compatible** *requests of the* **maximum** *size.*

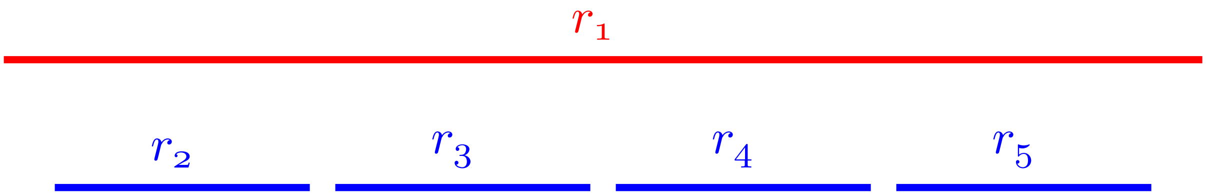
[Two requests are compatible if they don't overlap in time.]

Approach: Select requests greedily based on a "simple" rule.

Possible greedy choices

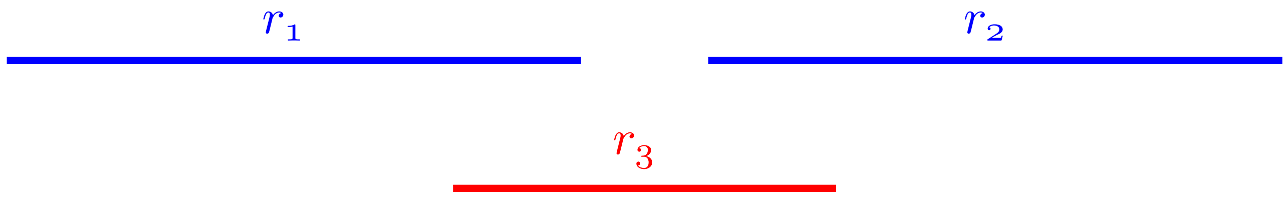
- Select available request that starts the earliest
- Select available request that requires the smallest interval of time to complete
- Select available request with the fewest number of conflicts
- Select available request that finishes the earliest

Choice 1 - Earliest start time



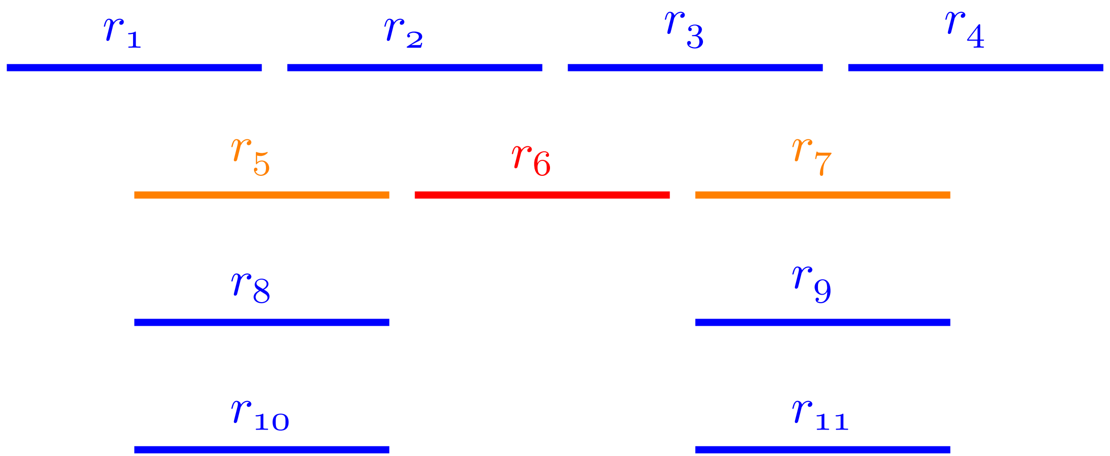
This criteria yields $A = \{r_1\}$.
But the correct answer is $A = \{r_2, r_3, r_4, r_5\}$.

Choice 2 - Shortest duration



This criteria yields $A = \{r_3\}$.
But the correct answer is $A = \{r_1, r_2\}$.

Choice 3 - Minimum number of conflicts



This criteria yields $A = \{r_5, r_6, r_7\}$.
But the correct answer is $A = \{r_1, r_2, r_3, r_4\}$.

Choice 4 - Earliest finish time

This actually works!

```
procedure IntervalScheduling(R)
    A := ∅
    while R is not empty do
        choose r ∈ R with the smallest finishing time
        add r to A
        delete all requests from R that are incompatible with r
    return A
```

Proof of correctness

Claim 1

A is a compatible set of requests.

Proof.

Obvious. $\square$

Now we will show that A is optimal.

Let φ be an optimal set of requests.

Want to show: $|A| = |\varphi|$.

"Stays ahead approach":
Let $A = \{r_{i_1}, \ldots, r_{i_k}\}$ and $\varphi = \{r_{j_1}, \ldots, r_{j_m}\}$.
Therefore, $|A| = k$, $|\varphi| = m$, and $k \leq m$ (since φ is optimal).

Proof of correctness

Claim 2

For all indices $\ell \leq k$, we have $t_{i_\ell} \leq t_{j_\ell}$.

Proof.

We will prove this by mathematical induction.

Base Case: The case for $\ell = 1$ is obvious by the greedy choice.

Ind. Hyp.: Assume Claim 2 is true for $\ell - 1$, i.e. $t_{i_{\ell-1}} \leq t_{j_{\ell-1}}$.

Ind. Step: Since φ is an optimal solution, and it picks both $r_{j_{\ell-1}}$ and r_{j_ℓ}, it follows that $t_{j_{\ell-1}} < s_{j_\ell}$.
Combining this with the ind. hyp., we obtain $t_{i_{\ell-1}} < s_{j_\ell}$.
In other words, request r_{j_ℓ} is in the set R of available requests at the time when the greedy algo selects r_{i_ℓ}.
By the greedy selection criterion, $t_{i_\ell} \leq t_{j_\ell}$. $\square$

Proof of correctness

Claim 3

A is optimal.

Proof.

We will prove this via contradiction.
If A is not optimal, then an optimal set φ must have more requests than A, i.e., $|\varphi| = m > k = |A|$.
Applying Claim 2 with $\ell = k$, we get $t_{i_k} \leq t_{j_k}$.
Since $m > k$, there is a request $r_{j_{k+1}} \in \varphi$.
Clearly, $s_{j_{k+1}} > t_{j_k} \geq t_{i_k}$.
This implies R is not empty (since $r_{j_{k+1}} \in R$) when r_{i_k} is selected.
On the other hand, R is empty since the greedy algo has stopped.
This yields the required contradiction. $\square$

Efficiency

procedure $\textsc{IntervalScheduling}(R)$
    sort R by the finishing times of the requests
    $r := R[0]$
    $f := t_r$
    $A := [r]$
    for $i = 1 : n-1$ **do**
        $r = R[i]$
        if $s_r \geq f$ **then**
            $A = A \cup \{r\}$
            $f = t_r$
    return A

Complexity: $\Theta(n \lg n)$.

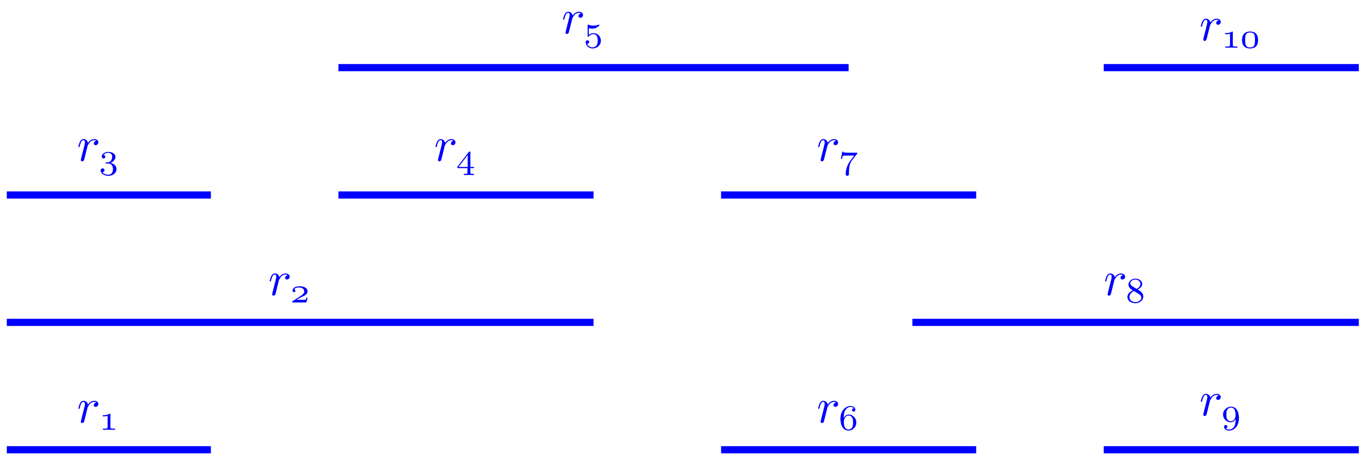
Scheduling all requests

There is a single resource and many requests in **Interval Scheduling**.

Problem (Interval Partitioning)

Given many identical resources and a list of requests, schedule all of them using as few resources as possible.

Example



These requests can be scheduled on 3 resources:

- $A_1 = \{r_1, r_4, r_7, r_{10}\}$
- $A_2 = \{r_2, r_6, r_9\}$
- $A_3 = \{r_3, r_5, r_8\}$

Observation

Definition

The depth of a set of requests is defined as the maximum number of requests that conflict at any given time instant.

Claim 4

In any instance of Interval Partitioning, the number of resources needed is at least the depth of the set of requests.

Proof.

Suppose a set of requests has depth d.
Let $r_1, r_2, \ldots, r_d$ be a set of requests that all conflict at some time instant.
Then each of these requests must be scheduled on a different resource. $\square$

Algorithm

```
procedure IntervalPartitioning(R)
    sort R by their start times
    d := depth of R
    for j = 1 to n do
        for each request r_i that precedes r_j and overlaps it do
            exclude the label of r_i from consideration for r_j
        if there is any label from {1, 2, ..., d} that has not been excluded then
            assign a non-excluded label to r_j
        else
            leave r_j unlabeled
```

Proof of correctness

Claim 5

Every request will receive a label, and no two overlapping requests will receive the same label.

Proof.

Let m requests conflict with request r_j.
This forms a set of $m+1$ overlapping intervals (overlapping at s_j).
Therefore, $m+1 \leq d$, i.e., $m \leq d-1$.
Thus, at least one of the d labels is not excluded, and a label can be assigned to r_j.

Next, consider two conflicting requests r_i and r_j, and r_i precedes r_j in the sorted order.
Then, when r_j is considered, r_i is in the set of requests whose labels are excluded from consideration.
Hence, r_i and r_j have different labels. □

This greedy algorithm is optimal by the Lower Bound Approach.

Complexity

```
procedure IntervalPartitioning(R)
    sort requests by their starting times
    d := 0
    Q := empty priority queue
    for j = 1 to n do
        k = ExtractMin(Q)
        if request r_j is compatible with resource k then
            schedule r_j on k
            update the value of k in Q to t_j
        else
            allocate a new resource d+1
            schedule request r_j on resource d+1
            d = d+1
            enqueue d+1 in Q with value t_j
```

Complexity: $\Theta(n \lg n)$.

Huffman Coding

Problem

Encode a piece of text into a long string of bits such that minimum space is required to store the data.

Basic Idea: For an alphabet of size ≤ 32, map each letter of the alphabet to a sequence of 5 bits. For example,

- $a \mapsto 00000$
- $b \mapsto 00001$
- ...

A string abc then maps to 000000000100010.

Clearly, decoding the original string back from the encoded bit string is unambiguous: map every 5 bits to the corresponding letter of the alphabet.

Space required by the encoding $= 5$ * size of text.

Key observation

Key observation: Frequency of usage of each letter of the alphabet is not the same.

Hence, it is a wastage of space to encode them with equal length of bits.

Better idea: Do variable length encoding with more frequent letters encoded by small number of bits and less frequent letters encoded by large number of bits.

Problem of variable length encoding

Decoding is ambiguous!

Assume our alphabet consists of three letters a, e and t. And we encode them as follows:

- $a \mapsto 01$
- $e \mapsto 0$
- $t \mapsto 1$

Then all the strings eta, aa, etet, aet are encoded by 0101.

Thus, decoding 0101 becomes ambiguous!

Solution

Definition (Prefix Codes)

A prefix code for a set S of letters is a function γ that maps each letter $x \in S$ to some sequence of zeros and ones in such a way that for distinct $x, y \in S$, the sequence $\gamma(x)$ is not a prefix of the sequence $\gamma(y)$.

Example

$S = \{a, b, c, d, e\}$
$\gamma(a) = 11$
$\gamma(b) = 01$
$\gamma(c) = 001$
$\gamma(d) = 10$
$\gamma(e) = 000$

Now, the string abc maps to 1101001, and can be decoded unambiguously as such!

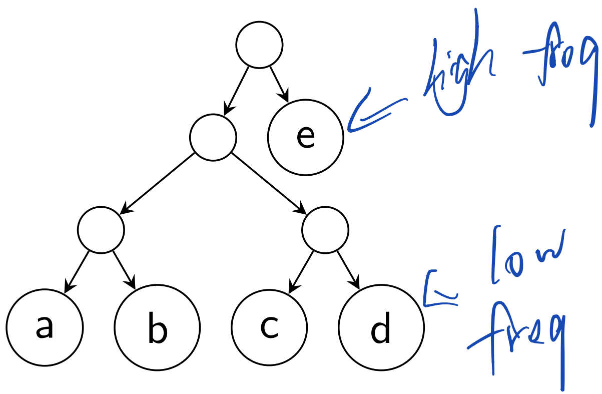
How to build prefix codes?

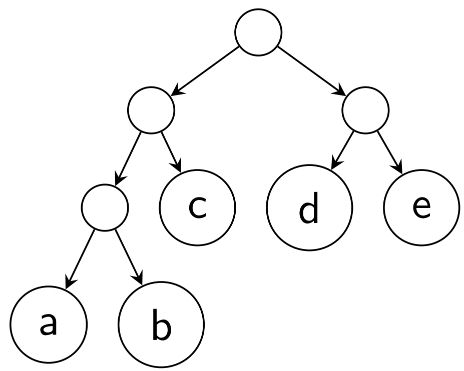
Claim 6

The encoding of S constructed from a binary tree T with the letters of S as the leaves of T and left child encoded as 0 and right child encoded as 1, is a prefix code.

The converse is also true: given a prefix code γ, we can build a binary tree as above.

Example





$\gamma_1(a) = 000, \gamma_1(b) = 001, \gamma_1(c) = 010, \gamma_1(d) = 011, \gamma_1(e) = 1.$

$\gamma_2(a) = 000, \gamma_2(b) = 001, \gamma_2(c) = 01, \gamma_2(d) = 10, \gamma_2(e) = 11.$

Optimal Prefix Codes

Suppose for each letter $x \in S$, there is a frequency f_x (representing the fraction of letters in the text that are equal to x). If total number of letters $= n$, number of occurrences of $x = nf_x$. Also, $\sum_x f_x = 1$.

Let γ be a prefix code for S. Then

$$\text{total encoding length} = \sum_{x \in S} nf_x |\gamma(x)| = n \sum_{x \in S} f_x |\gamma(x)|.$$

Therefore, average number of bits per letter,

$$ABL(\gamma) = \sum_{x \in S} f_x |\gamma(x)|.$$

Example

Let $f_a = 0.4, f_b = 0.3, f_c = 0.2, f_d = 0.05, f_e = 0.05$. Then,
$ABL(\gamma_1) = 0.4 * 3 + 0.3 * 3 + 0.2 * 3 + 0.05 * 3 + 0.05 * 1 = 2.9.$
$ABL(\gamma_2) = 0.4 * 3 + 0.3 * 3 + 0.2 * 2 + 0.05 * 2 + 0.05 * 2 = 2.7.$

Optimal Prefix Code Characterization

Definition

A binary tree is full if each internal node has 2 children.

Claim 7

The binary tree corresponding to the optimal prefix code is full.

Proof.

Assume T is a binary tree corresponding to an optimal prefix code γ, and T has a node u that has exactly one child v. Two cases:

- u root : Define a new tree T' by deleting u and making v the root.
- u not root : Let w be the parent of u. Define a new tree T' by deleting the node u and making v the child of w.

In either case, the number of bits needed to encode any leaf in the subtree rooted at u decreases, but remains intact for other leaves.

Let γ' be the prefix code corresponding to T'. It follows that $ABL(\gamma') < ABL(\gamma)$, which yields the required contradiction. $\square$

Another key observation

A key question: If someone gave you an optimal binary tree T^*, would you be able to label the leaves correctly?

Claim 8

Suppose u and v are leaves of T^* with *depth*$(u) <$ *depth*(v). Further suppose that in a labeling of T^* corresponding to an optimal prefix code, leaf u is labeled with $y \in S$ and leaf v is labeled with $z \in S$. Then $f_y \geq f_z$.

Proof.

If $f_y < f_z$, then by interchanging the labels of u and v, we get that

$$ABL(T^*) - ABL(T') = (\textit{depth}(u) - \textit{depth}(v))(f_y - f_z) > 0,$$

which contradicts the optimality of T^*. □

Corollary

There is an optimal binary tree T^*, in which the two lowest frequency letters are assigned to leaves that are siblings.

Huffman Algorithm

procedure $\textsc{HuffmanEncoding}(S, f)$
    if $S == \{a, b\}$ **then**
        $\gamma(a) = 0$
        $\gamma(b) = 1$
    else
        let y^* and z^* be the two lowest frequency letters
        form a new alphabet $S' = S \setminus \{y^*, z^*\} \cup \{w\}$ with $f_w = f_{y^*} + f_{z^*}$
        $\gamma' = \textsc{HuffmanEncoding}(S', f)$
        extend γ' to a prefix code γ for S as follows:
            $\gamma(y^*) = \gamma'(w)$.append(0)
            $\gamma(z^*) = \gamma'(w)$.append(1)
    return γ

Proof of correctness

Base Case: Works for $|S| = 2$.

Ind. Hyp.: Assume the algorithm gives an optimal tree T' for any alphabet S' of size $n - 1$.

Ind. Step: Let $|S| = n$, and T be the tree returned by the algorithm. Let y^* and z^* be the two least frequency letters that are siblings in T. Define a new tree T' by deleting the two nodes y^* and z^*, and replacing their parent with a node w with $f_w = f_{y^*} + f_{z^*}$.

Claim 9

$ABL(T') = ABL(T) - f_w$.

Proof later.

Also, since T' is the tree returned by the algorithm for a smaller alphabet $S' = S \cup \{w\} \setminus \{y^*, z^*\}$, it follows by induction hypothesis that T' is optimal for S'.

Proof of correctness

Now assume, for a contradiction, that T is not optimal for S.

Let Z be an optimal tree for S, i.e., $ABL(Z) < ABL(T)$. By the corollary, we can assume that y^* and z^* are siblings in Z.

Applying the same transformation of deleting y^* and z^* and replacing their parent by w, we obtain $ABL(Z') = ABL(Z) - f_w$.

Consequently,

$$ABL(Z') = ABL(Z) - f_w < ABL(T) - f_w = ABL(T'),$$

which contradicts the optimality of T'.

Proof of Claim 9

Proof.

$$
\begin{aligned}
& ABL(T) \\
= \; & \sum_{x \in S} f_x \cdot \mathit{depth}_T(x) \\
= \; & f_{y^*} \cdot \mathit{depth}_T(y^*) + f_{z^*} \cdot \mathit{depth}_T(z^*) + \sum_{x \neq y^*, z^*} f_x \cdot \mathit{depth}_T(x) \\
= \; & (f_{y^*} + f_{z^*}) \cdot (1 + \mathit{depth}_{T'}(w)) + \sum_{x \neq y^*, z^*} f_x \cdot \mathit{depth}_{T'}(x) \\
= \; & f_w \cdot (1 + \mathit{depth}_{T'}(w)) + \sum_{x \neq y^*, z^*} f_x \cdot \mathit{depth}_{T'}(x) \\
= \; & f_w + \sum_{x \in S'} f_x \cdot \mathit{depth}_{T'}(x) \\
= \; & f_w + ABL(T') \qquad \square
\end{aligned}
$$

Complexity

Complexity depends on how Line 4 of the algorithm is implemented.

If we store the frequencies in an array, then extracting minimum is $\Theta(n)$. Consequently,

Recurrence: $T(n) = T(n-1) + \Theta(n)$.
Complexity: $\Theta(n^2)$.

If we use a (heap based) priority queue to extract the two lowest frequency terms, then line 4 takes $\Theta(\lg n)$. Consequently,

Recurrence: $T(n) = T(n-1) + \Theta(\lg n)$.
Complexity: $\Theta(n \ln n)$.

Exercise: Solve the two recurrences.

More examples of greedy algorithms

- Single source shortest paths in a graph (Dijkstra)
- Minimum spanning tree (Prim, Kruskal)