Greedy Algorithms

Lecture 21

April 15, 2025

Part I

An Initial Example

Recall: MSTs

Recall the Minimum Spanning Tree problem from last time: Given a weighted graph $G = (V, E)$, find the spanning tree $T \subseteq E$ minimizing $\sum_{e \in T} w(e)$.

Recall: MSTs

Recall the Minimum Spanning Tree problem from last time: Given a weighted graph $G = (V, E)$, find the spanning tree $T \subseteq E$ minimizing $\sum_{e \in T} w(e)$.

Claim

The edge e with the smallest weight in G must be in its MST.

Recall: MSTs

Recall the Minimum Spanning Tree problem from last time: Given a weighted graph $G = (V, E)$, find the spanning tree $T \subseteq E$ minimizing $\sum_{e \in T} w(e)$.

Claim

The edge e with the smallest weight in G must be in its MST.

Claim

For any $S \subseteq V$, let e be the smallest edge that has exactly one endpoint in S. Then e must be in the MST.

Recall: MSTs

Recall the Minimum Spanning Tree problem from last time: Given a weighted graph $G = (V, E)$, find the spanning tree $T \subseteq E$ minimizing $\sum_{e \in T} w(e)$.

Claim

The edge e with the smallest weight in G must be in its MST.

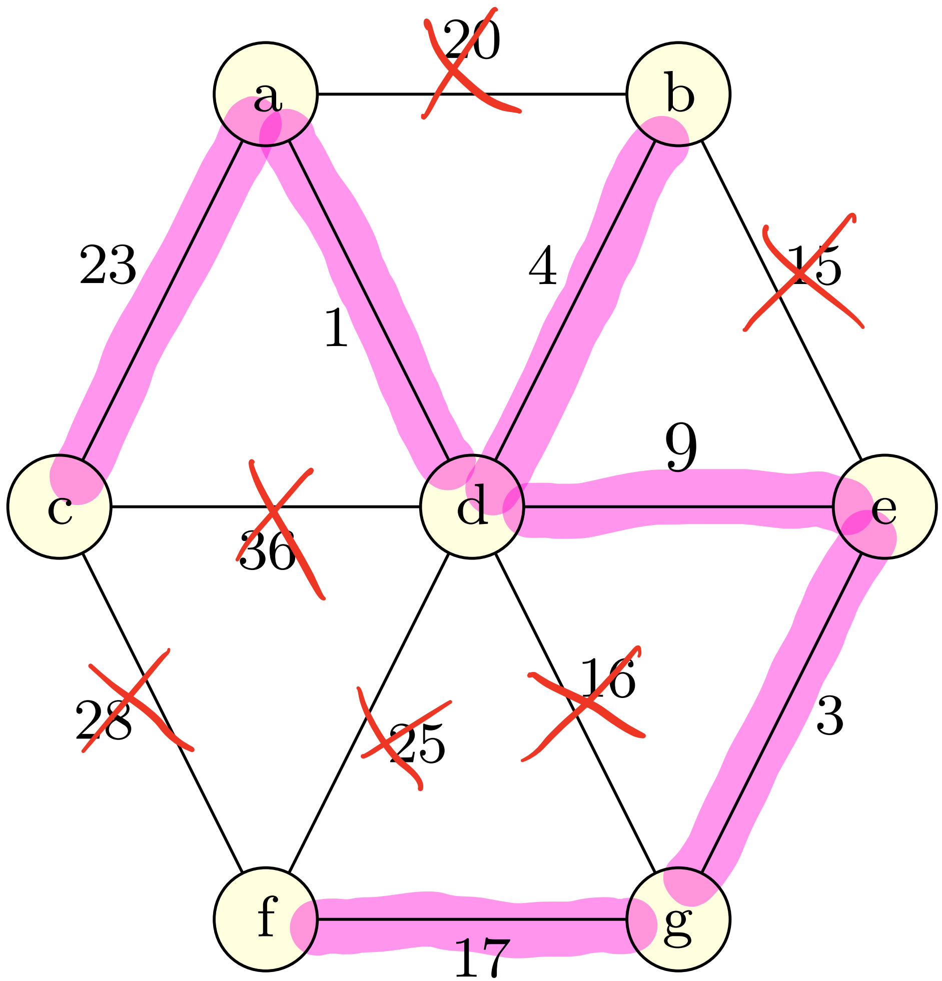
Claim

For any $S \subseteq V$, let e be the smallest edge that has exactly one endpoint in S. Then e must be in the MST.

What happens if we just repeatedly add the lightest edge anyway?

Greedy MST

Idea: add the lightest edge possible.



Kruskal's Pseudocode

```
Kruskal(G):
    Set T = ∅
    Sort edges in increasing order of weight
    for each edge e (in order by weight):
        if e connects two different components of (V, T):
            Add e to T
    return T
```

$O(E \log V)$

Correctness?

consider any edge e that Kruskal's adds to T,
e must connect two connected components: C_1, C_2
claim: e must be the lightest edge with exactly one endpoint in C_1
$\Rightarrow$ e must be in the MST

Greedy Algorithms, In General

General structure: repeatedly make “small” or “incremental” decisions about what to put in your final output.

Greedy Algorithms, In General

General structure: repeatedly make "small" or "incremental" decisions about what to put in your final output.

- Contrast with DP: instead of "considering all the options" and taking the best, just make a decision.

Greedy Algorithms, In General

General structure: repeatedly make "small" or "incremental" decisions about what to put in your final output.

- Contrast with DP: instead of "considering all the options" and taking the best, just make a decision.
- Decision is "greedy" in that it only considers what is best in the current state, without considering what will happen later.

Greedy Algorithms, In General

General structure: repeatedly make "small" or "incremental" decisions about what to put in your final output.

- Contrast with DP: instead of "considering all the options" and taking the best, just make a decision.
- Decision is "greedy" in that it only considers what is best in the current state, without considering what will happen later.

Often argue correctness by an "exchange argument":

Greedy Algorithms, In General

General structure: repeatedly make "small" or "incremental" decisions about what to put in your final output.

- Contrast with DP: instead of "considering all the options" and taking the best, just make a decision.
- Decision is "greedy" in that it only considers what is best in the current state, without considering what will happen later.

Often argue correctness by an "exchange argument":

- Consider any solution S other than the greedy one.

Greedy Algorithms, In General

General structure: repeatedly make "small" or "incremental" decisions about what to put in your final output.

- Contrast with DP: instead of "considering all the options" and taking the best, just make a decision.
- Decision is "greedy" in that it only considers what is best in the current state, without considering what will happen later.

Often argue correctness by an "exchange argument":

- Consider any solution S other than the greedy one.
- Find the "first" decision where S differs from greedy.

Greedy Algorithms, In General

General structure: repeatedly make "small" or "incremental" decisions about what to put in your final output.

- Contrast with DP: instead of "considering all the options" and taking the best, just make a decision.
- Decision is "greedy" in that it only considers what is best in the current state, without considering what will happen later.

Often argue correctness by an "exchange argument":

- Consider any solution S other than the greedy one.
- Find the "first" decision where S differs from greedy.
- Show that one can "exchange" the decision made by S for the greedy one *without making the solution worse*.

A Word of Warning

Greedy algorithms are nice in that they generally are simple to describe an efficient to run—but it is *very* easy to come up with a *wrong* greedy optimization!

A Word of Warning

Greedy algorithms are nice in that they generally are simple to describe an efficient to run—but it is *very* easy to come up with a *wrong* greedy optimization!

In 374, greedy algorithms always require a formal proof of correctness.

A Word of Warning

Greedy algorithms are nice in that they generally are simple to describe an efficient to run—but it is *very* easy to come up with a *wrong* greedy optimization!

In 374, greedy algorithms always require a formal proof of correctness.

Our advice is to always come up with an algorithm you *know* works first (eg using DP), and only then try to optimize with greedy.

Part II

Minimum Waiting Time

Problem Statement

Problem (Minimum Waiting Time)

Input: *A set of* n *jobs lengths to be scheduled on machine.*
Goal: *Order the jobs to minimize the time spent waiting, totaled over all jobs.*

Problem Statement

Problem (Minimum Waiting Time)

Input: *A set of* n *jobs lengths to be scheduled on machine.*
Goal: *Order the jobs to minimize the time spent waiting, totaled over all jobs.*

Example: say we have lengths $L = [4, 13, 10, 1, 374]$

in order: total wait time is

$0 + 4 + 17 + 27 + 28 = 76$

Greedy Attempts

Given job lengths, how do we (greedily) pick the order?

Example lengths: $L = [4, 13, 10, 1, 374]$

idea: sort jobs by increasing length

example: 1, 4, 10, 13, 374

total wait: 0 + 1 + 5 + 15 + 28 = 49

Exchange Intuition

Greedy idea: order jobs in increasing order of length.

Why is ordering **[4, 13, 10, 1, 374]** sub-optimal?

job 10: wait time decreases by 13

job 13: wait time increased by 10

every other job is unaffected!

$\Rightarrow$ improved our solution by 3.

Exchange Argument

Theorem

Ordering by increasing length gives the optimal solution.

Exchange Argument

Theorem

Ordering by increasing length gives the optimal solution.

Consider any order other than increasing length. In this order, there must be an index i such that $L[i] > L[i+1]$.

Exchange Argument

Theorem

Ordering by increasing length gives the optimal solution.

Consider any order other than increasing length. In this order, there must be an index i such that $L[i] > L[i+1]$.

Consider flipping the order of jobs i and $i+1$.

Exchange Argument

Theorem

Ordering by increasing length gives the optimal solution.

Consider any order other than increasing length. In this order, there must be an index i such that $L[i] > L[i+1]$.

Consider flipping the order of jobs i and $i+1$.

- Waiting time for job i ~~decreases~~ increases by $L[i+1]$
- Waiting time for job $i+1$ ~~increases~~ decreases by $L[i]$
- *All* the other jobs are unaffected

Exchange Argument

Theorem

Ordering by increasing length gives the optimal solution.

Consider any order other than increasing length. In this order, there must be an index i such that $L[i] > L[i+1]$.

Consider flipping the order of jobs i and $i+1$.

- Waiting time for job i decreases by $L[i+1]$
- Waiting time for job $i+1$ increases by $L[i]$
- *All* the other jobs are unaffected

Since $L[i] > L[i+1]$, this gives a strictly better solution!

(So the order we started with was not optimal.)

Part III

Job Scheduling

Problem Statement

Problem (Job Scheduling)

Input: *A set of n jobs with start and finish times to be scheduled on a resource (eg: classes in a classroom).*

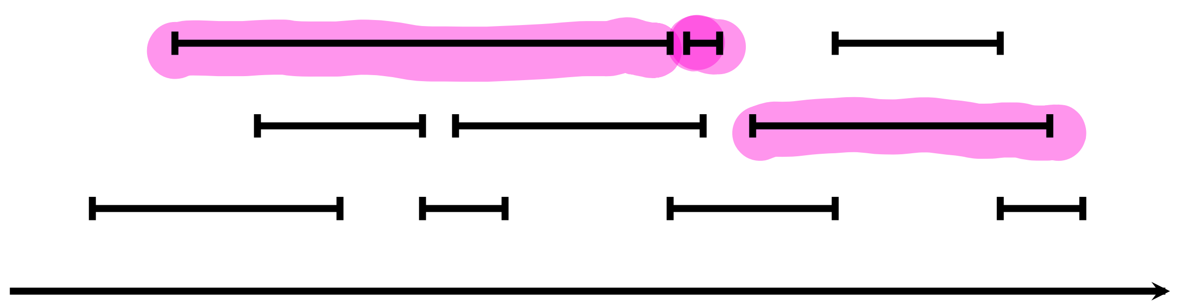
Goal: *Schedule as many jobs as possible (Two jobs with overlapping intervals cannot both be scheduled.)*

Problem Statement

Problem (Job Scheduling)

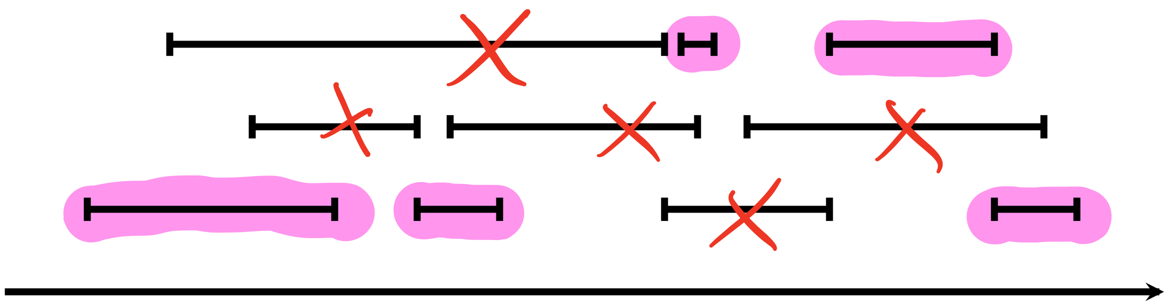
Input: *A set of* n *jobs with start and finish times to be scheduled on a resource (eg: classes in a classroom).*

Goal: *Schedule as many jobs as possible (Two jobs with overlapping intervals cannot both be scheduled.)*

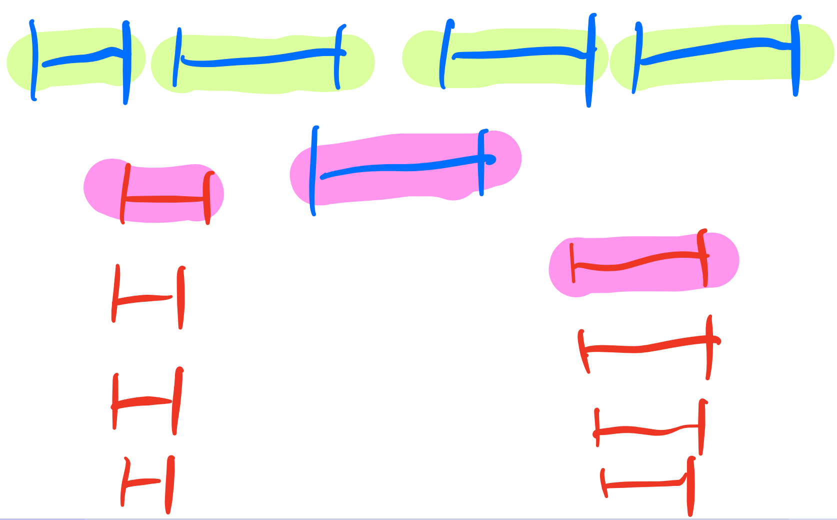


Greedy Attempts

Given job intervals, how do we (greedily) pick which to schedule?

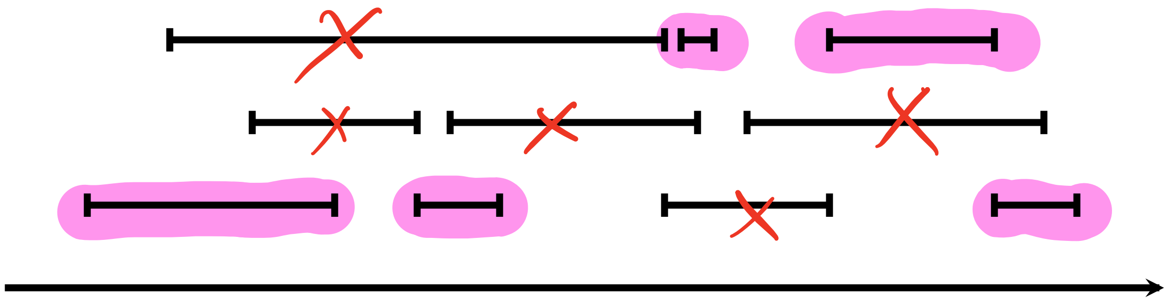


idea: take class with fewest conflicts



Greedy Attempts

Given job intervals, how do we (greedily) pick which to schedule?



idea: take the class that ends earliest

Greedy Pseudocode

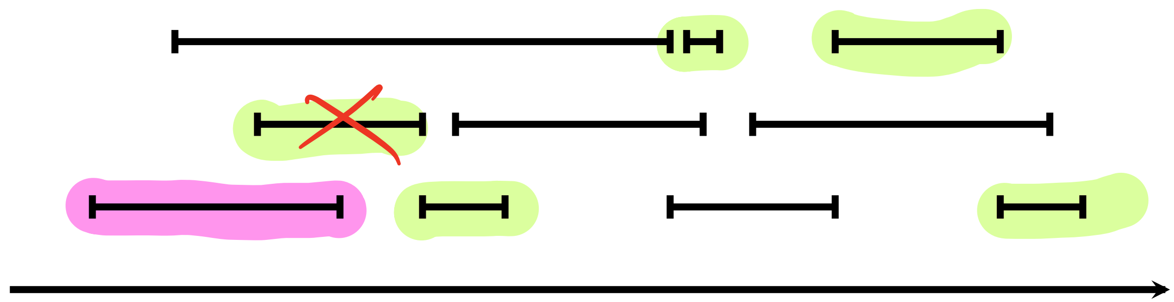
```
GreedyScheduling(J[1..n]):
    Set G = ∅ // Jobs to schedule
    Set lastAdded = −∞ // End time of last job scheduled
    Sort J by increasing end time          O(n log n)
    for each job j ∈ J:
        if j starts after lastAdded:
            Add j to G                      } → O(n)
            Set lastAdded to the end time of j
    return G
```

Efficiency? $O(n \log n)$

Exchange Intuition

Greedy idea: pick job with the earliest *finish* time.

Why can we say that any solution "may as well" use the job with the earliest finish time?



Exchange Warm-Up

Lemma

Let $g_1 = (s_1, f_1)$ *be the job with the earliest finish time. Then some optimal solution uses* g_1. *(Note: there may be multiple optimal solutions)*

Exchange Warm-Up

Lemma

Let $g_1 = (s_1, f_1)$ be the job with the earliest finish time. Then some optimal solution uses g_1. (Note: there may be multiple optimal solutions)

Let S be any optimal solution that does *not* use g_1, and $j_1 = (s'_1, f'_1)$ be the job in S with the earliest finish time.

Exchange Warm-Up

Lemma

Let $g_1 = (s_1, f_1)$ be the job with the earliest finish time. Then some optimal solution uses g_1. (Note: there may be multiple optimal solutions)

Let S be any optimal solution that does *not* use g_1, and $j_1 = (s_1', f_1')$ be the job in S with the earliest finish time.

Since S has no overlaps, *all* jobs in $S - \{j_1\}$ start after $f_1' \geq f_1$.

Exchange Warm-Up

Lemma

Let $g_1 = (s_1, f_1)$ be the job with the earliest finish time. Then some optimal solution uses g_1. (Note: there may be multiple optimal solutions)

Let S be any optimal solution that does *not* use g_1, and $j_1 = (s'_1, f'_1)$ be the job in S with the earliest finish time.

Since S has no overlaps, *all* jobs in $S - \{j_1\}$ start after $f'_1 \geq f_1$.

Thus $S' = (S - \{j_1\}) \cup \{g_1\}$ is a valid solution that (1) uses g_1, and (2) is optimal (since $|S'| = |S|$).

Exchange Argument I

Theorem

The greedy solution is an optimal solution.

Exchange Argument I

Theorem

The greedy solution is an optimal solution.

Let $G = (g_1, g_2 \ldots, g_k)$ be the greedy solution (sorted by finish time). We can write *any* optimal solution S (sorted by finish time) as $(g_1, \ldots, g_{i-1}, j_i, \ldots, j_m)$—$g_i$ is the *first* greedy interval not in S.

Exchange Argument I

Theorem

The greedy solution is an optimal solution.

Let $G = (g_1, g_2 \ldots, g_k)$ be the greedy solution (sorted by finish time). We can write *any* optimal solution S (sorted by finish time) as $(g_1, \ldots, g_{i-1}, j_i, \ldots, j_m)$—$g_i$ is the *first* greedy interval not in S.

Since S has no overlaps, *all* jobs in $(j_{i+1}, \ldots, j_m)$ start after j_i finishes—which must be after g_i finishes!

Exchange Argument I

Theorem

The greedy solution is an optimal solution.

Let $G = (g_1, g_2 \ldots, g_k)$ be the greedy solution (sorted by finish time). We can write *any* optimal solution S (sorted by finish time) as $(g_1, \ldots, g_{i-1}, j_i, \ldots, j_m)$—$g_i$ is the *first* greedy interval not in S.

Since S has no overlaps, *all* jobs in $(j_{i+1}, \ldots, j_m)$ start after j_i finishes—which must be after g_i finishes!

Thus $S' = (S - \{j_i\}) \cup \{g_i\}$ is an optimal solution that is "closer" to the greedy solution.

Exchange Argument II

Theorem

The greedy solution is an optimal solution.

If $S = (g_1, \ldots, g_{i-1}, j_i, \ldots, j_m)$ is an optimal solution, we can exchange j_i for g_i.

Exchange Argument II

Theorem

The greedy solution is an optimal solution.

If $S = (g_1, \ldots, g_{i-1}, j_i, \ldots, j_m)$ is an optimal solution, we can exchange j_i for g_i.

Applying this repeatedly, we can get an optimal solution that uses the *entire* greedy solution: $S_G = (g_1, \ldots, g_k, j_{k+1}, \ldots, j_m)$.

Exchange Argument II

Theorem

The greedy solution is an optimal solution.

If $S = (g_1, \ldots, g_{i-1}, j_i, \ldots, j_m)$ is an optimal solution, we can exchange j_i for g_i.

Applying this repeatedly, we can get an optimal solution that uses the *entire* greedy solution: $S_G = (g_1, \ldots, g_k, j_{k+1}, \ldots, j_m)$.

Note that greedy only stops when there are no intervals possible to add—so in fact we must have that $S_G = G$!

Sometimes Greed is Bad

Suppose now each job has some reward $r(j)$, and we want to maximize the *total reward* instead of the number of jobs.

What happens if we still pick the job with the earliest finish time?

Sometimes Greed is Bad

Suppose now each job has some reward $r(j)$, and we want to maximize the *total reward* instead of the number of jobs.

What happens if we still pick the job with the earliest finish time?

Well, what about picking the job with the highest reward?

Sometimes Greed is Bad

Suppose now each job has some reward $r(j)$, and we want to maximize the *total reward* instead of the number of jobs.

What happens if we still pick the job with the earliest finish time?

Well, what about picking the job with the highest reward?

Maybe the job whose conflicts have the least total reward?

Sometimes Greed is Bad

Suppose now each job has some reward $r(j)$, and we want to maximize the *total reward* instead of the number of jobs.

What happens if we still pick the job with the earliest finish time?

Well, what about picking the job with the highest reward?

Maybe the job whose conflicts have the least total reward?

Takeaway: Greed is tempting, but not always the right answer.

Exercise: write a DP algorithm for the weighted version.

Takeaway Points

- Greedy algorithms have the advantage of being relatively simple to state, but often are incorrect. *Always* prove correctness.
- *Exchange* arguments are often the key proof ingredient. Start with understanding why the first step of the algorithm is correct: need to show that there is an optimum/correct solution that agrees with the first step of the greedy algorithm.
- Thinking about correctness is also a good way to figure out which of the many greedy strategies is likely to work.