



Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat [edu_assist_pro](#)
Aleks Ignjatov

School of Computer Science and Engineering
University of New South Wales

6. THE GREEDY METHOD

The Greedy Method

Activity selection problem.

Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times f_i . No two activities can take place simultaneously.

Assignment Project Exam Help

Task: Find a *maximum size* subset of compatible activities.



Attempt 1: always choose the shortest activity which has not previously been chosen. Activities in green are the conflicting ones.

Add WeChat edu_assist_pro



- The above figure shows this does not work...
(chosen activities in green, conflicting in red)

The Greedy Method

Activity selection problem.

Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times f_i . No two activities can take place simultaneously.

Assignment Project Exam Help

Task: Find a *maximum size* subset of compatible activities.



Attempt 1: always choose the shortest activity which previously chosen activities, remove the conflicting activi



- The above figure shows this does not work...
(chosen activities in green, conflicting in red)

The Greedy Method

Activity selection problem.

Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times f_i . No two activities can take place simultaneously.

Assignment Project Exam Help

Task: Find a *maximum size* subset of compatible activities.



Attempt 1: always choose the shortest activity which previously chosen activities, remove the conflicting activi



- The above figure shows this does not work...
(chosen activities in green, conflicting in red)

The Greedy Method

Activity selection problem.

Instance: A list of activities (a_i , $0 \leq i \leq n$) with starting times s_i and finishing times f_i . No two activities can take place simultaneously.

Task: Find a *maximum size* subset of compatible activities.

- Att the fe this way we minimally restrict our next choice....



- As appealing this idea is, the above figure shows this again does not work ...

The Greedy Method

Activity selection problem.

Instance: A list of activities (a_i , $0 \leq i \leq n$) with starting times s_i and finishing times f_i . No two activities can take place simultaneously.

Task: Find a *maximum size* subset of compatible activities.

- Att the fe this way we minimally restrict our next choice....



- As appealing this idea is, the above figure shows this again does not work ...

The Greedy Method

The correct solution: Among the activities which do not conflict with the previously chosen activities always chose the one with the earliest end time.

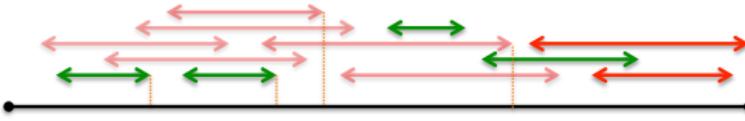
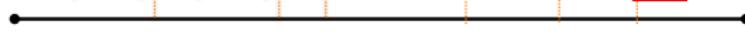
Assignment Project Exam Help



<https://eduassistpro.github.io>



Add WeChat edu_assist_pro

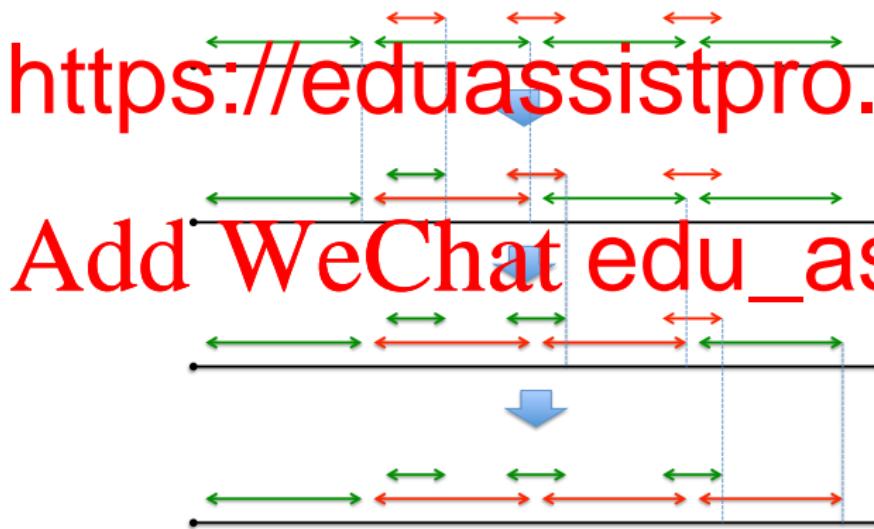


Proving optimality of a greedy solution

Show that any optimal solution can be transformed into the greedy solution with equal number of activities:

- Find the first place where the chosen activity violates the greedy choice.
- Show that replacing that activity with the greedy choice produces a better solution.

Assignment Project Exam Help
Continue in this manner till you "morph" your optimal solution into the greedy solution, thus proving the greedy solution is also optimal.

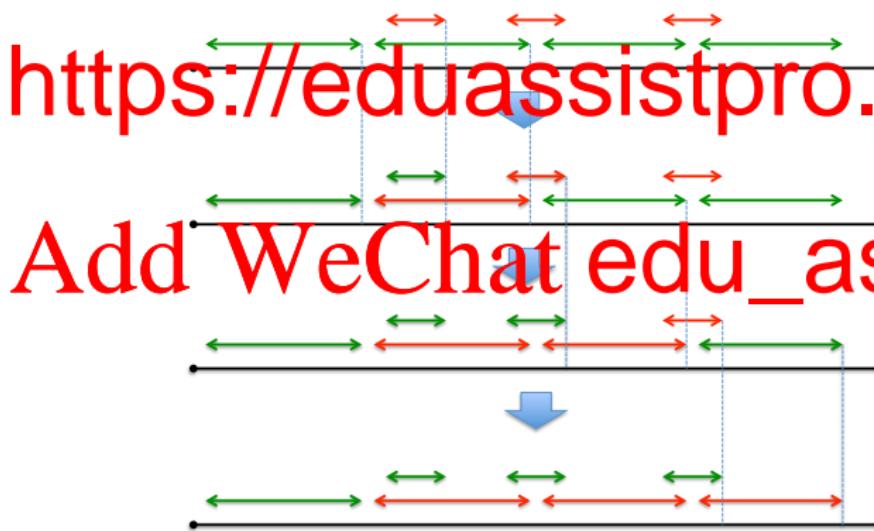


<https://eduassistpro.github.io>
Add WeChat edu_assist_pro

Proving optimality of a greedy solution

Show that any optimal solution can be transformed into the greedy solution with equal number of activities:

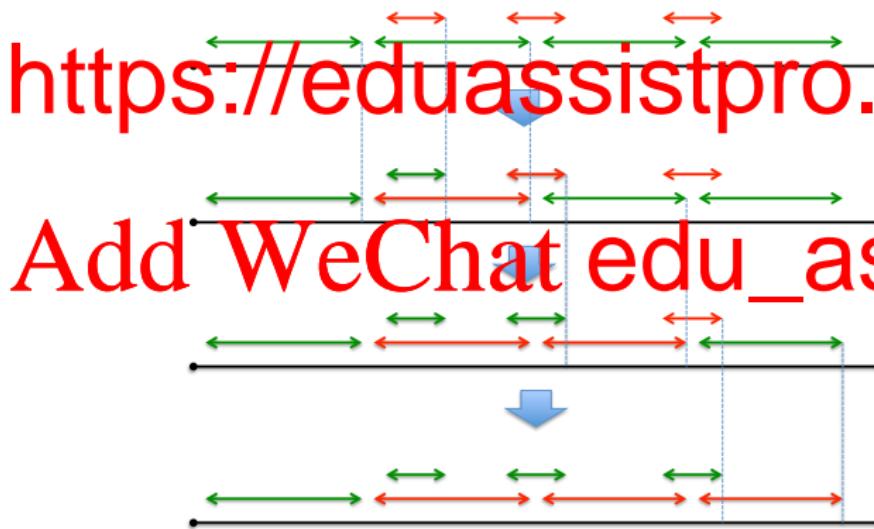
- Find the first place where the chosen activity violates the greedy choice.
 - Show that replacing that activity with the greedy choice produces a better solution.
- Continue in this manner till you "morph" your optimal solution into the greedy solution, thus proving the greedy solution is also optimal.



Proving optimality of a greedy solution

Show that any optimal solution can be transformed into the greedy solution with equal number of activities:

- Find the first place where the chosen activity violates the greedy choice.
 - Show that replacing that activity with the greedy choice produces an assignment schedule with the same number of activities.
- Continue in this manner till you "morph" your optimal solution into the greedy solution, thus proving the greedy solution is also optimal.

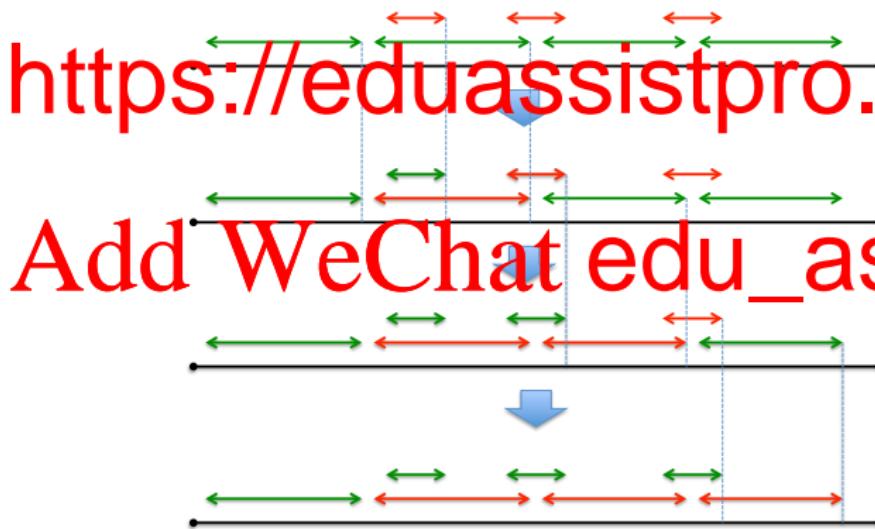


Proving optimality of a greedy solution

Show that any optimal solution can be transformed into the greedy solution with equal number of activities:

- Find the first place where the chosen activity violates the greedy choice.
 - Show that replacing that activity with the greedy choice produces a non-optimal solution.

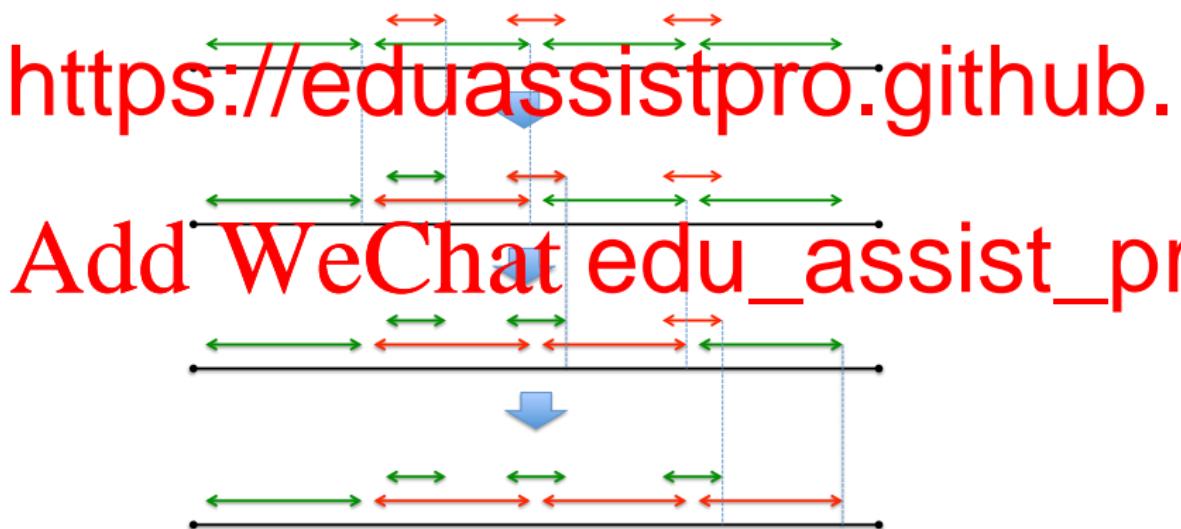
- Continue in this manner till you “morph” your optimal solution into the greedy solution, thus proving the greedy solution is also optimal.



Proving optimality of a greedy solution

Show that any optimal solution can be transformed into the greedy solution with equal number of activities:

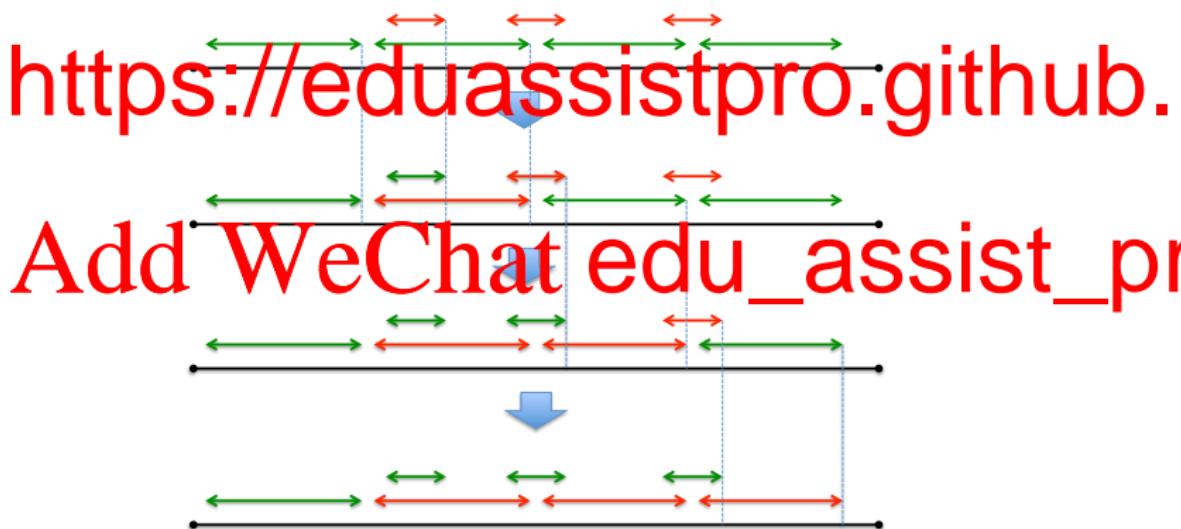
- Find the first place where the chosen activity violates the greedy choice.
- Show that replacing that activity with the greedy choice produces a non-conflicting selection with the same number of activities.
- Continue in this manner till you “morph” your optimal solution into the greedy solution, thus proving the greedy solution is also optimal.



Proving optimality of a greedy solution

Show that any optimal solution can be transformed into the greedy solution with equal number of activities:

- Find the first place where the chosen activity violates the greedy choice.
- Show that replacing that activity with the greedy choice produces a non-conflicting selection with the same number of activities.
- Continue in this manner till you “morph” your optimal solution into the greedy solution, thus proving the greedy solution is also optimal.



- What is the time complexity of the algorithm?

Assignment Project Exam Help

We present activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing times.

- This is done in $O(n \log n)$ time.
- We go through such a sorted list in order, looking for the first activity whose starting time is after the finishing time of the last activity handled so far.
- Every activity is handled only once, so this part of the algorithm runs in $O(n)$ time.
- Thus, the algorithm runs in total time $O(n \log n)$.

- What is the time complexity of the algorithm?

Assignment Project Exam Help

We represent activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing times.

- This is done in $O(n \log n)$ time.
- We go through such a sorted list in order, looking for the first activity whose starting time is after the finishing time of the last activity handled so far.
- Every activity is handled only once, so this part of the algorithm runs in $O(n)$ time.
- Thus, the algorithm runs in total time $O(n \log n)$.

- What is the time complexity of the algorithm?

Assignment Project Exam Help

- We represent activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing times

- This <https://eduassistpro.github.io/>
- We go through such a sorted list in order, looking for the first activity whose starting time is after the finishing time of the last activity.
- Every activity is handled only once, so this part of the algorithm runs in $O(n)$ time.
- Thus, the algorithm runs in total time $O(n \log n)$.

- What is the time complexity of the algorithm?

Assignment Project Exam Help

- We represent activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing times

- This <https://eduassistpro.github.io>.
- We go through such a sorted list in order, looking for the first activity whose starting time is after the finishing time of the last activity.
- Every activity is handled only once, so this part of the algorithm runs in $O(n)$ time.
- Thus, the algorithm runs in total time $O(n \log n)$.

- What is the time complexity of the algorithm?

Assignment Project Exam Help

We represent activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing times.

- This part of the algorithm runs in total time $O(n \log n)$.
- We go through such a sorted list in order, looking for the first activity whose starting time is after the finishing time of the last activity handled so far.
- Every activity is handled only once, so this part of the algorithm runs in $O(n)$ time.
- Thus, the algorithm runs in total time $O(n \log n)$.

- What is the time complexity of the algorithm?

We represent activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing times.

- We represent activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing times.

- This part of the algorithm runs in $O(n \log n)$ time.
- We go through such a sorted list in order, looking for the first activity whose starting time is after the finishing time of the last activity handled.
- Every activity is handled only once, so this part of the algorithm runs in $O(n)$ time.
- Thus, the algorithm runs in total time $O(n \log n)$.

The Greedy Method

Activity selection problem II

- Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times $f_i = s_i + d$ thus, all activities are of the same duration.
No two activities can take place simultaneously.

- Tasks:
 - Find the maximum number of activities that can be performed.

- Solution:
 - To find the maximum number of activities that can be performed, we can use a greedy algorithm. We start with an empty set of selected activities and iterate through the list of activities. For each activity, if it does not overlap with the currently selected activities, we add it to the set. This process continues until no more activities can be added.
 - i.e., the previous problem.

- Question: What happens if the activities have different durations?
- Greedy strategy no longer works - we will need a more sophisticated technique.

The Greedy Method

Activity selection problem II

- Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times $f_i = s_i + d$ thus, all activities are of the same duration.
No two activities can take place simultaneously.

- Tasks:
1. Implement greedy algorithm for activity selection problem.

- Solution:
to find the maximum number of non-overlapping activities
i.e., the previous problem.

- Question:
What happens if the activities have different durations?
- Greedy strategy no longer works - we will need a more sophisticated technique.

The Greedy Method

Activity selection problem II

- Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times $f_i = s_i + d$ thus, all activities are of the same duration.
No two activities can take place simultaneously.

- Task: Find the maximum number of non-overlapping activities.

- Solution: <https://eduassistpro.github.io/>
to find the solution
i.e., the previous problem.

- Question: What happens if the activities have different durations?
- Greedy strategy no longer works - we will need a more sophisticated technique.

The Greedy Method

Activity selection problem II

- Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times $f_i = s_i + d$ thus, all activities are of the same duration.
No two activities can take place simultaneously.

- Task: Find the maximum number of non-overlapping activities.

- Solution: Use dynamic programming to find the solution to the previous problem.
i.e., the previous problem.

- Question: What happens if the activities are of different durations?
Add WeChat edu_assist_pro
- Greedy strategy no longer works - we will need a more sophisticated technique.

The Greedy Method

Activity selection problem II

- Instance: A list of activities a_i , $(1 \leq i \leq n)$ with starting times s_i and finishing times $f_i = s_i + d$ thus, all activities are of the same duration.
No two activities can take place simultaneously.

- Task: Find the maximum number of non-overlapping activities.

- Solution: Use dynamic programming to find the solution to the previous problem.
i.e., the previous problem.

- Question: What happens if the activities are of different durations?
Add WeChat edu_assist_pro
- Greedy strategy no longer works - we will need a more sophisticated technique.

More practice problems for the Greedy Method

- Along the long, straight road from Loololong to Goolagong houses are scattered quite sparsely, sometimes with long gaps between two consecutive houses. Telstra must provide mobile phone service to people who live alongside the road, and the range of Telstra's cell base station is 5km. Design an algorithm for placing the minimal number of base stations alongside the road, that is sufficient to cover all houses.

Assignment Project Exam Help

 <https://eduassistpro.github.io>

Add WeChat edu_assist_pro

More practice problems for the Greedy Method

- Once again, along the long, straight road from Loololong (on the West) to Goolagong (on the East) houses are scattered quite sparsely, sometimes with long gaps between two consecutive houses. Telstra must provide mobile phone service to people who live alongside the road, and the range of Telstras cell base station is 5km.

- One

put
not
the E

- His junior associate did exactly the same but started moving very words and claimed his answer is the best.

Add WeChat `edu_assist_pro`

- Is there a placement of houses for which the associate is right?



More practice problems for the Greedy Method

- Once again, along the long, straight road from Loololong (on the West) to Goolagong (on the East) houses are scattered quite sparsely, sometimes with long gaps between two consecutive houses. Telstra must provide mobile phone service to people who live alongside the road, and the range of Telstras cell base station is 5km.

- One put a t not a the E
- His junior associate did exactly the same but started moving very words and claimed his answer is right
- Is there a placement of houses for which the associate is right?



More practice problems for the Greedy Method

- Once again, along the long, straight road from Loololong (on the West) to Goolagong (on the East) houses are scattered quite sparsely, sometimes with long gaps between two consecutive houses. Telstra must provide mobile phone service to people who live alongside the road, and the range of Telstras cell base station is 5km.

- One
put a t
not a
the E

<https://eduassistpro.github.io>

- His junior associate did exactly the same but started moving westwards and claimed that his method

[Add WeChat edu_assist_pro](#)

- Is there a placement of houses for which the associate is right?



More practice problems for the Greedy Method

- Once again, along the long, straight road from Loololong (on the West) to Goolagong (on the East) houses are scattered quite sparsely, sometimes with long gaps between two consecutive houses. Telstra must provide mobile phone service to people who live alongside the road, and the range of Telstras cell base station is 5km.

- One
put a
not a
the E

<https://eduassistpro.github.io>

- His junior associate did exactly the same but started moving westwards and claimed that his method is better.

[Add WeChat edu_assist_pro](https://eduassistpro.github.io)

- Is there a placement of houses for which the associate is right?



More practice problems for the Greedy Method

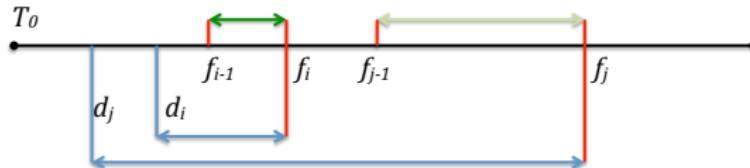
Minimising job lateness

- **Instance:** A start time T_0 and a list of jobs a_i , $(1 \leq i \leq n)$, with duration times t_i and deadlines d_i . Only one job can be performed at any time; all jobs have to be completed. If a job a_i is completed at a finishing time $f_i \geq d_i$ then we say that it has incurred lateness $l_i = f_i - d_i$.

- Tasks are due by their deadlines.
- Solutions must respect the deadlines.

- Optimality: Consider any optimal solution. Will it form a minimum if the deadline is shifted by ϵ before

Add WeChat [edu_assist_pro](https://eduassistpro.github.io)



More practice problems for the Greedy Method

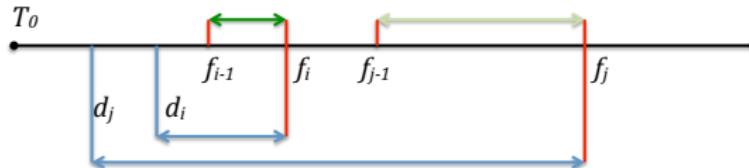
Minimising job lateness

- **Instance:** A start time T_0 and a list of jobs a_i , $(1 \leq i \leq n)$, with duration times t_i and deadlines d_i . Only one job can be performed at any time; all jobs have to be completed. If a job a_i is completed at a finishing time $f_i \geq d_i$ then we say that it has incurred lateness $l_i = f_i - d_i$.

- Tasks late
- Solutions to deadlines.

- Optimality: Consider any optimal solution. Will it form a minimum if job a_i is scheduled before

Add WeChat `edu_assist_pro`



More practice problems for the Greedy Method

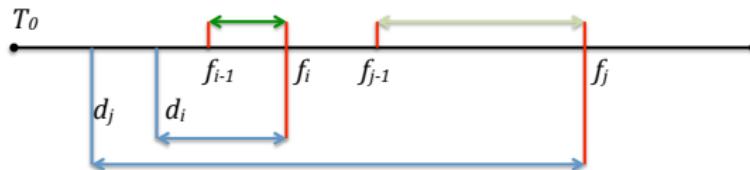
Minimising job lateness

- **Instance:** A start time T_0 and a list of jobs a_i , $(1 \leq i \leq n)$, with duration times t_i and deadlines d_i . Only one job can be performed at any time; all jobs have to be completed. If a job a_i is completed at a finishing time $f_i \geq d_i$ then we say that it has incurred lateness $l_i = f_i - d_i$.

- Tasks late
- Solutions deadlines.

- Optimality: Consider any optimal solution. Will it form a minimum if job a_i is scheduled before

Add WeChat `edu_assist_pro`



More practice problems for the Greedy Method

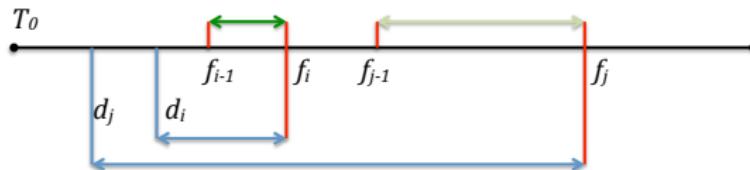
Minimising job lateness

- **Instance:** A start time T_0 and a list of jobs a_i , $(1 \leq i \leq n)$, with duration times t_i and deadlines d_i . Only one job can be performed at any time; all jobs have to be completed. If a job a_i is completed at a finishing time $f_i \geq d_i$ then we say that it has incurred lateness $l_i = f_i - d_i$.

- Tasks late
- Solutions deadlines.

- **Optimality:** Consider any optimal solution. What form an inversion if job a_i is scheduled before job a_j ?

Add WeChat edu_assist_pro



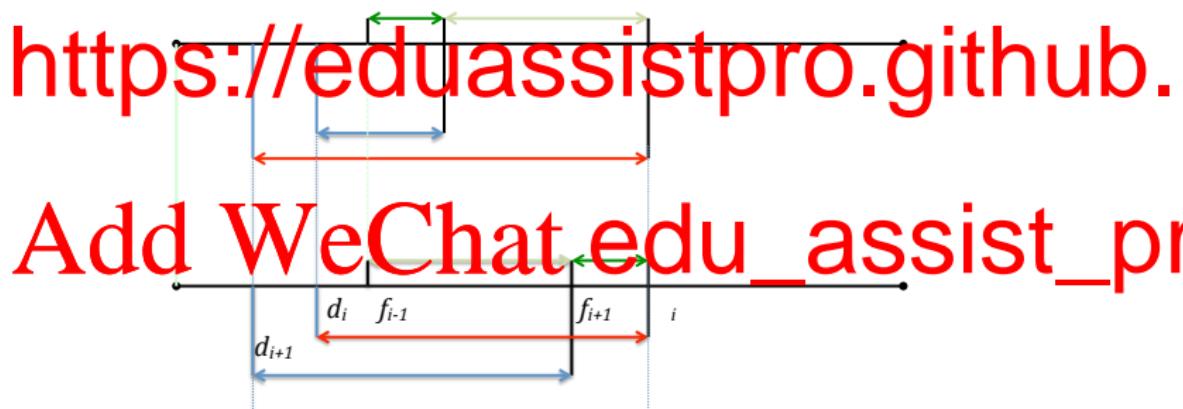
More practice problems for the Greedy Method

Minimising job lateness

- We will show that there exists a scheduling without inversions which is also optimal.

Assignment Project Exam Help

Assignment Project Exam Help
Assignment Project Exam Help

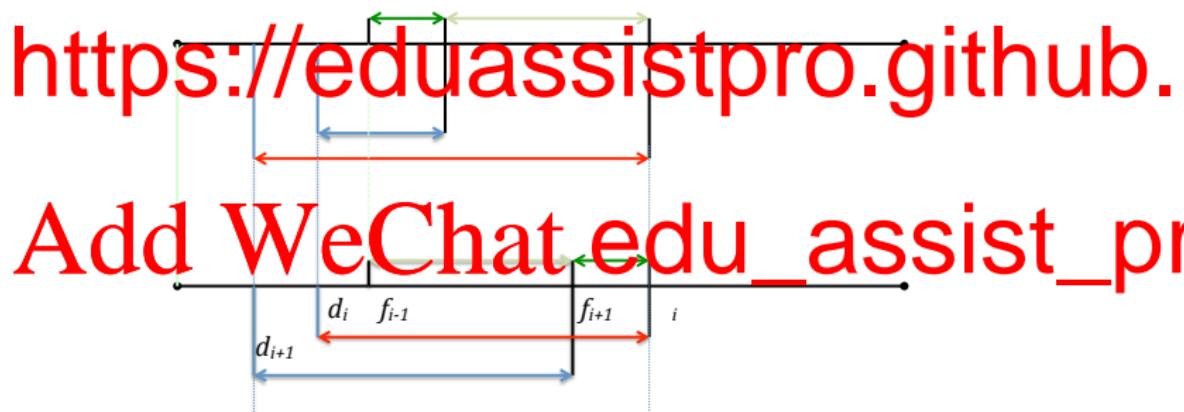


- Note that swapping adjacent inverted jobs reduces the larger lateness!

More practice problems for the Greedy Method

Minimising job lateness

- We will show that there exists a scheduling without inversions which is also optimal.
- Recall the BubbleSort. If we manage to eliminate all inversions between adjacent jobs, eventually all the inversions will be eliminated.



- Note that swapping adjacent inverted jobs reduces the larger lateness!

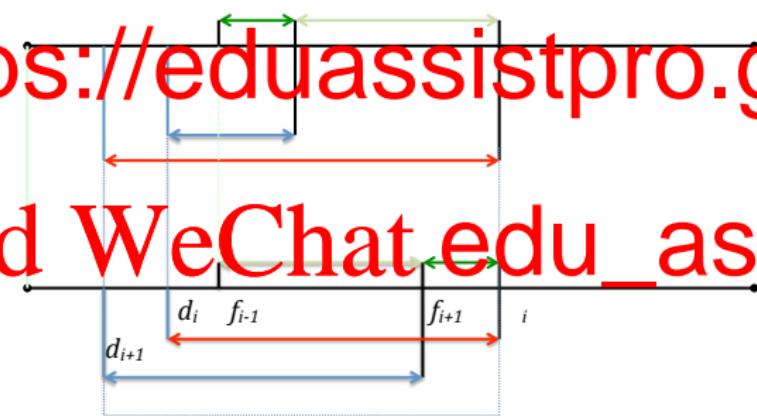
More practice problems for the Greedy Method

Minimising job lateness

- We will show that there exists a scheduling without inversions which is also optimal.
 - Recall the BubbleSort. If we manage to eliminate all inversions between adjacent jobs, eventually all the inversions will be eliminated.

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr



- Note that swapping adjacent inverted jobs reduces the larger lateness!

Tape storage

Assignment Project Exam Help

- Instance: A list of n files j_1 of lengths l_j , which have to be stored on a tape. Each file is equally likely to be needed. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task is minimised if

<https://eduassistpro.github.io>

- Solution: If the files are stored in order l_1, l_2, \dots, l_n , then the expected time is proportional to

Add WeChat edu_assist_pro

$$nl_1 + (n-1)l_2 + \dots + 2l_{n-1} + l_n$$

- This is minimised if $l_1 \leq l_2 \leq l_3 \leq \dots \leq l_n$.

Tape storage

Assignment Project Exam Help

- Instance: A list of n files j_1 of lengths l_j , which have to be stored on a tape. Each file is equally likely to be needed. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task is min

<https://eduassistpro.github.io>

- Solution: If the files are stored in order l_1, l_2, \dots, l_n , then the expected time is proportional to

Add WeChat edu_assist_pro

$$nl_1 + (n-1)l_2 + \dots + 2l_{n-1} + l_n$$

- This is minimised if $l_1 \leq l_2 \leq l_3 \leq \dots \leq l_n$.

Tape storage

Assignment Project Exam Help

- **Instance:** A list of n files j_1 of lengths l_j , which have to be stored on a tape. Each file is equally likely to be needed. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task is min

<https://eduassistpro.github.io>

- **Solution:** If the files are stored in order l_1, l_2, \dots, l_n , then the expected time is proportional to

Add WeChat edu_assist_pro

$$l_1 + (l_1 + l_2) + (l_1 + l_2 + l_3) + \dots + nl_1 + (n - 1)l_2 + \dots + (n - 2)l_{n-1} + nl_n$$

- This is minimised if $l_1 \leq l_2 \leq l_3 \leq \dots \leq l_n$.

Tape storage

Assignment Project Exam Help

- **Instance:** A list of n files j_1 of lengths l_j , which have to be stored on a tape. Each file is equally likely to be needed. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task is min

<https://eduassistpro.github.io>

- **Solution:** If the files are stored in order l_1, l_2, \dots, l_n , then the expected time is proportional to

Add WeChat edu_assist_pro

$$l_1 + (l_1 + l_2) + (l_1 + l_2 + l_3) + \dots + nl_1 + (n - 1)l_2 + \dots + (n - 2)l_{n-1} + nl_n$$

- This is minimised if $l_1 \leq l_2 \leq l_3 \leq \dots \leq l_n$.

More practice problems for the Greedy Method

Tape storage II

- Instance: A list of n files J_i of lengths l_i and probabilities to be needed p_i , $\sum_{i=1}^n p_i = 1$, which have to be stored on a tape. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task

time is

minimised

<https://eduassistpro.github.io>

- Solution

time is

proportional to

1 2 \dots n

Add WeChat edu_assist_pro

- We now show that this is minimised if the files are ordered in a decreasing order of values of the ratio p_i/l_i .

More practice problems for the Greedy Method

Tape storage II

- Instance: A list of n files J_i of lengths l_i and probabilities to be needed p_i , $\sum_{i=1}^n p_i = 1$, which have to be stored on a tape. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task

minimise

time is

<https://eduassistpro.github.io>

- Solution

is proportional to

1 2 \dots n

time is

Add WeChat [edu_assist_pro](https://eduassistpro.github.io)

- We now show that this is minimised if the files are ordered in a decreasing order of values of the ratio p_i/l_i .

More practice problems for the Greedy Method

Tape storage II

- Instance: A list of n files j_i of lengths l_i and probabilities to be needed p_i , $\sum_{i=1}^n p_i = 1$, which have to be stored on a tape. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task

minimise

<https://eduassistpro.github.io>

- Solution

is proportional to

1 2 \dots n

time is

read time is

$p_1 l_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + \dots + l_n) p_n$

- We now show that this is minimised if the files are ordered in a decreasing order of values of the ratio p_i/l_i .

More practice problems for the Greedy Method

Tape storage II

- Instance: A list of n files j_i of lengths l_i and probabilities to be needed p_i , $\sum_{i=1}^n p_i = 1$, which have to be stored on a tape. To retrieve a file, one must start from the beginning of the tape and scan it until the file is found and read.

- Task: minimize total time is proportional to
- Solution: total time is proportional to

$p_1 l_1 + ((l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + \dots + l_n) p_n)$

- We now show that this is minimised if the files are ordered in a decreasing order of values of the ratio p_i/l_i .

More practice problems for the Greedy Method

- Let us see what happens if we swap two adjacent files f_k and f_{k+1} .
- The expected time before the swap and after the swap are, respectively,

$$E = l_1 p_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k) p_k$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k + l_{k+1} - l_k) p_k + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

Assignment Project Exam Help

and

$$E' = l_1 p$$

$$+ (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1}$$

$$\text{https://eduassistpro.github.io}$$

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive if $l_k p_{k+1} > l_{k+1} p_k$, i.e., if $p_k/l_k < p_{k+1}/l_{k+1}$.
- Consequently, if $p_k/l_k > p_{k+1}/l_{k+1}$ then swapping decreases the expected time just in case there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k .
- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

More practice problems for the Greedy Method

- Let us see what happens if we swap two adjacent files f_k and f_{k+1} .
- The expected time before the swap and after the swap are, respectively,

$$E = l_1 p_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k) p_k$$

$$(l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k + l_{k+1}) p_{k+1}$$

and

$$E' = l_1 p_1$$

$$+ (l_1$$

<https://eduassistpro.github.io>

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive if $l_k p_{k+1} > l_{k+1} p_k$, i.e., if $p_k/l_k < p_{k+1}/l_{k+1}$.
- Consequently, if $p_k/l_k > p_{k+1}/l_{k+1}$ then swapping decreases the expected time just in case there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k .
- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

More practice problems for the Greedy Method

- Let us see what happens if we swap two adjacent files f_k and f_{k+1} .
- The expected time before the swap and after the swap are, respectively,

$$E = l_1 p_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k) p_k \\ + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + l_3 + \dots + l_k) p_k$$

and

$$E' = l_1 p_1$$

$$+ (l_1$$

<https://eduassistpro.github.io>

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive if $l_k p_{k+1} > l_{k+1} p_k$, i.e., if $p_k/l_k < p_{k+1}/l_{k+1}$.
- Consequently, if $p_k/l_k > p_{k+1}/l_{k+1}$ then swapping decreases the expected time just in case there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k .
- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

More practice problems for the Greedy Method

- Let us see what happens if we swap two adjacent files f_k and f_{k+1} .
- The expected time before the swap and after the swap are, respectively,

$$E = l_1 p_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k) p_k \\ + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + l_3 + \dots + l_n) p_n$$

and

$$E' = l_1 p_1 + (l_1 + l_2) p_2 + \dots + (l_1 + l_2 + \dots + l_{k-1} + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + \dots + l_n) p_n$$

<https://eduassistpro.github.io>

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive if $p_k/l_k < p_{k+1}/l_{k+1}$.
- Consequently, if $p_k/l_k > p_{k+1}/l_{k+1}$ then swapping decreases the expected time just in case there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k .
- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

More practice problems for the Greedy Method

- Let us see what happens if we swap two adjacent files f_k and f_{k+1} .
- The expected time before the swap and after the swap are, respectively,

$$E = l_1 p_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k) p_k \\ + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + l_3 + \dots + l_n) p_n$$

and

$$E' = l_1 p_1 + (l_1 + l_2) p_2 + \dots + (l_1 + l_2 + \dots + l_{k-1} + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + \dots + l_n) p_n$$

<https://eduassistpro.github.io>

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive if $p_k/l_k < p_{k+1}/l_{k+1}$.
- Consequently, if there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k , then swapping decreases the expected time just in case there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k .
- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

More practice problems for the Greedy Method

- Let us see what happens if we swap two adjacent files f_k and f_{k+1} .
- The expected time before the swap and after the swap are, respectively,

$$E = l_1 p_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k) p_k \\ + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + l_3 + \dots + l_n) p_n$$

and

$$E' = l_1 p_1 + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + l_3 + \dots + l_n) p_n$$

<https://eduassistpro.github.io>

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive if $p_k/l_k < p_{k+1}/l_{k+1}$.
- Consequently, $E > E'$ if and only if $p_k/l_k < p_{k+1}/l_{k+1}$. This swap decreases the expected time just in case there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k .
- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

More practice problems for the Greedy Method

- Let us see what happens if we swap two adjacent files f_k and f_{k+1} .
- The expected time before the swap and after the swap are, respectively,

$$E = l_1 p_1 + (l_1 + l_2) p_2 + (l_1 + l_2 + l_3) p_3 + \dots + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_k) p_k \\ + (l_1 + l_2 + l_3 + \dots + l_{k-1} + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + l_3 + \dots + l_n) p_n$$

and

$$E' = l_1 p_1 + (l_1 + l_2) p_2 + \dots + (l_1 + l_2 + \dots + l_{k-1} + l_{k+1}) p_{k+1} + \dots + (l_1 + l_2 + \dots + l_n) p_n$$

<https://eduassistpro.github.io>

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive if $p_k/l_k < p_{k+1}/l_{k+1}$.
- Consequently, $E > E'$ if and only if $p_k/l_k < p_{k+1}/l_{k+1}$. This swap decreases the expected time just in case there is an inversion: a file f_{k+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_k with a smaller ratio p_k/l_k .
- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

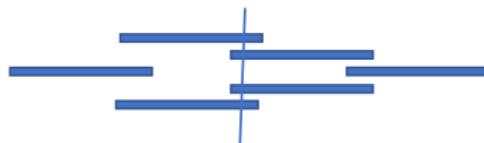
More practice problems for the Greedy Method

- Let X be a set of n intervals on the real line. We say that a set P of points stabs X if every interval in X contains at least one point in P ; see the figure below. Describe and analyse an efficient algorithm to compute the smallest set of points that stabs X . Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$ representing the left and right endpoints of the intervals in X .

Assignment Project Exam Help

<https://eduassistpro.github.io>

- Add WeChat `edu_assist_pro`



- Hint: the interval which ends the earliest has to be stabbed.
- What is the best place to stab it?

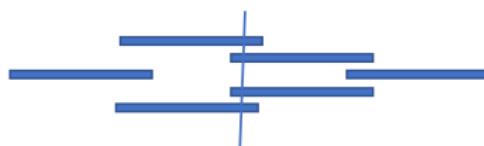
More practice problems for the Greedy Method

- Let X be a set of n intervals on the real line. We say that a set P of points stabs X if every interval in X contains at least one point in P ; see the figure below. Describe and analyse an efficient algorithm to compute the smallest set of points that stabs X . Assume that your input consists of two arrays $L[1..n]$ and $R[1..n]$ representing the left and right endpoints of the intervals in X .

Assignment Project Exam Help

<https://eduassistpro.github.io>

- Is it a good idea to stab the largest possible number of intervals?



- Hint: the interval which ends the earliest has to be stabbed.
- What is the best place to stab it?

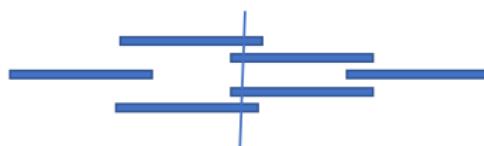
More practice problems for the Greedy Method

- Let X be a set of n intervals on the real line. We say that a set P of points stabs X if every interval in X contains at least one point in P ; see the figure below. Describe and analyse an efficient algorithm to compute the smallest set of points that stabs X . Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$ representing the left and right endpoints of the intervals in X .

Assignment Project Exam Help

<https://eduassistpro.github.io>

- Is it a good idea to stab the largest possible number of intervals?



- Hint: the interval which ends the earliest has to be stabbed.
- What is the best place to stab it?

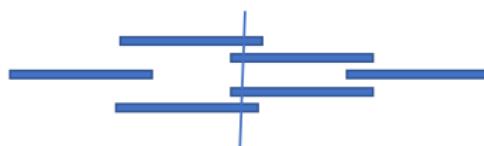
More practice problems for the Greedy Method

- Let X be a set of n intervals on the real line. We say that a set P of points stabs X if every interval in X contains at least one point in P ; see the figure below. Describe and analyse an efficient algorithm to compute the smallest set of points that stabs X . Assume that your input consists of two arrays $L[1..n]$ and $R[1..n]$ representing the left and right endpoints of the intervals in X .

Assignment Project Exam Help

<https://eduassistpro.github.io>

- Is it a good idea to stab the largest possible number of intervals?



- Hint: the interval which ends the earliest has to be stabbed.
- What is the best place to stab it?

The Greedy Method

0-1 knapsack problem

- **Instance:** A list of weights w_i and values v_i for discrete items a_i , $1 \leq i \leq n$, and a maximal weight limit W of your knapsack.

Assignment Project Exam Help

- Tas exce

- Can

<https://eduassistpro.github.io>

- Assume there are just three items with weights and values $(20\text{kg}, \$100)$, $(10\text{kg}, \$120)$ and a knapsack of capacity $W = 30\text{kg}$.

Add WeChat edu_assist_pro

- Greedy would choose $(10\text{kg}, \$60)$ and $(20\text{kg}, \$100)$, while the optimal choice is to take $(20\text{kg}, \$100)$ and $(30\text{kg}, \$120)$!

- So when does the Greedy Strategy work??
- Unfortunately there is no easy rule...

The Greedy Method

0-1 knapsack problem

Assignment Project Exam Help

- Tas
exe

• Can <https://eduassistpro.github.io>

- Assume there are just three items with weights and values (20kg, \$100), (10kg, \$120) and a knapsack of capacity

Add WeChat edu_assist_pro

- Greedy would choose (10kg, \$60) and (20kg, \$100), while the best choice is to take (20kg, \$100) and (30kg, \$120)!

- So when does the Greedy Strategy work??
- Unfortunately there is no easy rule...

The Greedy Method

0-1 knapsack problem

Assignment Project Exam Help

- Instance: A list of weights w_i and values v_i for discrete items a_i , $1 \leq i \leq n$, and a maximal weight limit W of your knapsack.
- Tas exce
- Can <https://eduassistpro.github.io>

- Assume there are just three items with weights and values $(20\text{kg}, \$100)$, $(10\text{kg}, \$120)$ and a knapsack of capacity $W = 30\text{kg}$.

Add WeChat edu_assist_pro

- Greedy would choose $(10\text{kg}, \$60)$ and $(20\text{kg}, \$100)$, while the optimal choice is to take $(20\text{kg}, \$100)$ and $(30\text{kg}, \$120)$!

- So when does the Greedy Strategy work??
- Unfortunately there is no easy rule...

The Greedy Method

0-1 knapsack problem

- Instance: A list of weights w_i and values v_i for discrete items a_i , $1 \leq i \leq n$, and a maximal weight limit W of your knapsack.

Assignment Project Exam Help

- Tas exce
- Can <https://eduassistpro.github.io>
- Assume there are just three items with weights and values $(20\text{kg}, \$100)$, $(30\text{kg}, \$120)$ and a knapsack of capacity $W = 50\text{kg}$. The greedy algorithm would choose $(20\text{kg}, \$100)$ and $(30\text{kg}, \$120)$, while it is better to take $(10\text{kg}, \$60)$ and $(20\text{kg}, \$100)$, which totals 30kg and $\$160$!
- So when does the Greedy Strategy work??
- Unfortunately there is no easy rule...

The Greedy Method

0-1 knapsack problem

- Instance: A list of weights w_i and values v_i for discrete items a_i , $1 \leq i \leq n$, and a maximal weight limit W of your knapsack.

Assignment Project Exam Help

- Tas exce

- Can <https://eduassistpro.github.io>

- Assume there are just three items with weights and values $(20\text{kg}, \$100)$, $(30\text{kg}, \$120)$ and a knapsack of capacity

Add WeChat edu_assist_pro

- Greedy would choose $(10\text{kg}, \$60)$ and $(20\text{kg}, \$100)$, while it is to take $(20\text{kg}, \$100)$ and $(30\text{kg}, \$120)$!

- So when does the Greedy Strategy work??
- Unfortunately there is no easy rule...

The Greedy Method

0-1 knapsack problem

- Instance: A list of weights w_i and values v_i for discrete items a_i , $1 \leq i \leq n$, and a maximal weight limit W of your knapsack.

Assignment Project Exam Help

- Tas exce

- Can <https://eduassistpro.github.io>

- Assume there are just three items with weights and val (20kg, \$100), (30kg, \$120) and a knapsack of capacity

Add WeChat edu_assist_pro

- Greedy would choose (10kg, \$60) and (20kg, \$100), while it is to take (20kg, \$100) and (30kg, \$120)!

- So when does the Greedy Strategy work??

- Unfortunately there is no easy rule...

The Greedy Method

0-1 knapsack problem

- Instance: A list of weights w_i and values v_i for discrete items a_i , $1 \leq i \leq n$, and a maximal weight limit W of your knapsack.

Assignment Project Exam Help

- Tas exce
- Can <https://eduassistpro.github.io>
- Assume there are just three items with weights and val (20kg, \$100), (30kg, \$120) and a knapsack of capacity 50kg. Greedy would choose (10kg, \$60) and (20kg, \$100), while the optimal solution is to take (20kg, \$100) and (30kg, \$120)!
- So when does the Greedy Strategy work??
- Unfortunately there is no easy rule...

Add WeChat edu_assist_pro

Assignment Project Exam Help

- Assume you are given n sorted arrays of different sizes.
- You are allowed to merge any two arrays into a single new sorted array and p

<https://eduassistpro.github.io>

- Design an algorithm to merge n sorted arrays into one sorted array. Provide a justification why your algorithm is optimal.
- This problem is somewhat related to the knapsack problem, which is, arguably, among the most important problems solved by the greedy method!

Assignment Project Exam Help

- Assume you are given n sorted arrays of different sizes.
- You are allowed to merge any two arrays into a single new sorted array and p

<https://eduassistpro.github.io>

- Design an algorithm to merge n sorted arrays into one sorted array. Provide a justification why your algorithm is optimal.
- This problem is somewhat related to the knapsack problem, which is, arguably, among the most important problems solved by the greedy method!

Assignment Project Exam Help

- Assume you are given n sorted arrays of different sizes.
- You are allowed to merge any two arrays into a single new sorted array and p

• Design an algorithm to merge n sorted arrays into one sorted array and provide a justification why your algorithm is optimal.

- This problem is somewhat related to the knapsack problem, which is, arguably, among the most important problems solved by the greedy method!

Assignment Project Exam Help

- Assume you are given n sorted arrays of different sizes.
- You are allowed to merge any two arrays into a single new sorted array and p

• Design an algorithm to merge n sorted arrays into one sorted array and provide a justification why your algorithm is optimal.

- This problem is somewhat related to the next application, which is, arguably, among the most important applications of the greedy method!

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 5

$= 32$.

- To do this we need to give each symbol a binary string of length 5 bits a

- However this is not an economical way: all the symbols have the same length but the symbols are not equally frequent.

- One way of doing this is to assign shorter codes to more frequent symbols. For example, we could assign shorter codes to symbols that appear more frequently, such as 't' have short codes while infrequent ones, such as 'w', 'x', 'y', 'z', have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?

- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.

- Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 5

$= 32$.

- To do this we need 5 bits a

- However this is not an economical way: all the symbols h length but the symbols are not equally frequent.

- One way to do this is to assign shorter codes to more frequent symbols. For example, we could assign shorter codes to more frequent symbols, such as 't' have short codes while infrequent ones, such as 'w', 'x', 'y', 'z', longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?

- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 5

bits a

- However this is not an economical way: all the symbols have the same length but the symbols are not equally frequent.

- One way to do an encoding in a more efficient way is to map 't' have short codes while infrequent ones, such as 'w', 'x', 'y' have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?
- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

Add WeChat [edu_assist_pro](https://eduassistpro.github.io)

Add WeChat [edu_assist_pro](https://eduassistpro.github.io)

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 6

$^{5} = 32$.

- To do this we need 5 bits a

- However this is not an economical way: all the symbols have the same length but the symbols are not equally frequent.

- One way to do this is to assign shorter codes to more frequent symbols. For example, we could assign shorter codes to more frequent symbols, such as 't' have short codes while infrequent ones, such as 'w', 'x', 'y', 'z' have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?

- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 6

$5^5 = 32$.

- To de bits a

- However this is not an economical way: all the symbols have equal length but the symbols are not equally frequent.

- One way to do a better job is to assign shorter codes to more frequent symbols, so that 't' have short codes while infrequent ones, such as 'w', 'x', 'y', 'z', have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?
- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 6

$^{5} = 32$.

- To de bits a

- However this is not an economical way: all the symbols have equal length but the symbols are not equally frequent.

- One would prefer an encoding in which frequent symbols like 't' have short codes while infrequent ones, such as 'w', 'x', 'y', 'z', have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?
- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 6

$^{5} = 32$.

- To do this we need 5 bits a

- However this is not an economical way: all the symbols have the same length but the symbols are not equally frequent.

- One would prefer an encoding in which frequent symbols like 't' have short codes while infrequent ones, such as 'w', 'x', 'y' have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream **UNIQUELY** into segments each corresponding to a code?

- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 6

$^{5} = 32$.

- To do this we need 5 bits a

- However this is not an economical way: all the symbols have the same length but the symbols are not equally frequent.

- One would prefer an encoding in which frequent symbols like 't' have short codes while infrequent ones, such as 'w', 'x', 'y' have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?

- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.

• Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

- One way of doing so is to reserve bit strings of equal and sufficient length, give

plus 6

$^{5} = 32$.

- To do this we need 5 bits a

- However this is not an economical way: all the symbols have the same length but the symbols are not equally frequent.

- One would prefer an encoding in which frequent symbols like 't' have short codes while infrequent ones, such as 'w', 'x', 'y' have longer codes.

- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?

- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

The most important applications of the Greedy Method: The Huffman Code

Assignment Project Exam Help

- We can now formulate the problem:

Give

opti

enc

<https://eduassistpro.github.io>

- Note that this amounts to saying that the
in an “average” text is as small as possible.

1

- Add WeChat edu_assist_pr
- We now sketch the algorithm informally; please see th
and the proof of optimality.

Assignment Project Exam Help

- There are n radio towers for broadcasting tsunami warnings. You are given the (x, y) coordinates of each tower and its radius of range. When a tower will activate, it will send a signal to all other towers within its range.

- You are given a set of n towers. You need to find the minimum number of towers that will eventually get activated and send a tsunami warning signal to all other towers.
- The goal is to determine which towers must be equipped with seismic sensors so that they can detect the activation of other towers.

Add WeChat `edu_assist_pro`

Assignment Project Exam Help

- There are n radio towers for broadcasting tsunami warnings. You are given the (x, y) coordinates of each tower and its radius of range. When a tow will a

- You whe towers will eventually get activated and send a tsu

- The goal is to determine which towers you must equip with seismic sensors.

Assignment Project Exam Help

- There are n radio towers for broadcasting tsunami warnings. You are given the (x, y) coordinates of each tower and its radius of range. When a tow will a

- You whe towers will eventually get activated and send a tsu
- The goal is to design an algorithm which finds the few towers you must equip with seismic sensors.

Add WeChat `edu_assist_pro`

- Someone has proposed the following two algorithms:

Algorithm 1: Find the unactivated tower with the largest radius (if multiple towers have the same radius, select one at random). Activate this tower. Find and remove all activated towers. Repeat.

②

<https://eduassistpro.github.io>

- Give examples which show that neither Algorithm 1 nor Algorithm 2 correctly solve the problem.
- Design an algorithm which correctly solves the problem.
- Solving this problem involves several important concepts which we now revisit.

Greedy Method applied to graphs

- Someone has proposed the following two algorithms:

① Algorithm 1: Find the unactivated tower with the largest radius (if multiple with the same radius, pick the any of them). Activate this tower. Find and remove all activated towers. Repeat.

② <https://eduassistpro.github.io>

- Give examples which show that neither Algorithm 1 nor Algorithm 2 solve the problem correctly.
- Design an algorithm which correctly solves the problem.
- Solving this problem involves several important concepts which we now revisit.

Greedy Method applied to graphs

- Someone has proposed the following two algorithms:

① Algorithm 1: Find the unactivated tower with the largest radius (if multiple with the same radius, pick the any of them). Activate this tower. Find and remove all activated towers. Repeat.

② <https://eduassistpro.github.io>

- Give examples which show that neither Algorithm 1 nor Algorithm 2 solve the problem correctly.
- Design an algorithm which correctly solves the problem.
- Solving this problem involves several important concepts which we now revisit.

- Someone has proposed the following two algorithms:

① Algorithm 1: Find the unactivated tower with the largest radius (if multiple with the same radius, pick the any of them). Activate this tower. Find and remove all activated towers. Repeat.

② <https://eduassistpro.github.io>

- Give examples which show that neither Algorithm 1 or 2 solve the problem correctly.
- Design an algorithm which correctly solves the problem.
- Solving this problem involves several important concepts which we now revisit.

Greedy Method applied to graphs

- Someone has proposed the following two algorithms:

① Algorithm 1: Find the unactivated tower with the largest radius (if multiple with the same radius, pick the any of them). Activate this tower. Find and remove all activated towers. Repeat.

② <https://eduassistpro.github.io>

- Give examples which show that neither Algorithm 1 or 2 solve the problem correctly.
- Design an algorithm which correctly solves the problem.
- Solving this problem involves several important concepts which we now revisit.

Greedy Method applied to graphs

- Someone has proposed the following two algorithms:

① Algorithm 1: Find the unactivated tower with the largest radius (if multiple with the same radius, pick the any of them). Activate this tower. Find and remove all activated towers. Repeat.

② <https://eduassistpro.github.io>

- Give examples which show that neither Algorithm 1 or 2 solve the problem correctly.
- Design an algorithm which correctly solves the problem.
- Solving this problem involves several important concepts which we now revisit.

Greedy Method applied to graphs

- Given a directed graph $G = (V, E)$ and a vertex v , the *strongly connected component* of G containing v consists of all vertices $u \in V$ such that there is a path in G from v to u and a path from u to v .

Assignment Project Exam Help

- Con
vert
dire
- Use BFS to find the set $D \subseteq V$ of all vertices in V which are reachable in G from v .
- Find the set $R \subseteq V$ all vertices which are reachable from v .
- The strongly connected component C of G containing v is just the set $C = D \cap R$.
- Clearly, distinct strongly connected components are disjoint sets.

Greedy Method applied to graphs

- Given a directed graph $G = (V, E)$ and a vertex v , the *strongly connected component* of G containing v consists of all vertices $u \in V$ such that there is a path in G from v to u and a path from u to v .

- How do we find a strongly connected component $C \subseteq V$ containing v ?

- Con
vert
dire

of

he

<https://eduassistpro.github.io>

- Use BFS to find the set $D \subseteq V$ of all vertices in V which are reachable in G from v .
- Find the subset $R \subseteq V$ of all vertices which are reachable from v .
- The strongly connected component C of G containing v is just the set $C = D \cap R$.
- Clearly, distinct strongly connected components are disjoint sets.

Greedy Method applied to graphs

- Given a directed graph $G = (V, E)$ and a vertex v , the *strongly connected component* of G containing v consists of all vertices $u \in V$ such that there is a path in G from v to u and a path from u to v .

- How do we find a strongly connected component $C \subseteq V$ containing u ?

- Convert directed graph to an undirected graph.

<https://eduassistpro.github.io>

- Use BFS to find the set $D \subseteq V$ of all vertices in V which are reachable in G from v .

- Find the set $R \subseteq V$ of all vertices which are reachable from v .

- The strongly connected component C of G containing v is just the set $C = D \cap R$.

- Clearly, distinct strongly connected components are disjoint sets.

Greedy Method applied to graphs

- Given a directed graph $G = (V, E)$ and a vertex v , the *strongly connected component* of G containing v consists of all vertices $u \in V$ such that there is a path in G from v to u and a path from u to v .

Assignment Project Exam Help

- How do we find a strongly connected component $C \subseteq V$ containing u ?

- Convert directed graph to an undirected graph G' by adding edges $(u, v) \in E$ if $(v, u) \in E$.

<https://eduassistpro.github.io>

- Use BFS to find the set $D \subseteq V$ of all vertices in V which are reachable in G' from v .

- Find the subset $R \subseteq V$ of all vertices which are not reachable from v .

- The strongly connected component C of G containing v is just the set $C = D \cap R$.

- Clearly, distinct strongly connected components are disjoint sets.

Greedy Method applied to graphs

- Given a directed graph $G = (V, E)$ and a vertex v , the *strongly connected component* of G containing v consists of all vertices $u \in V$ such that there is a path in G from v to u and a path from u to v .

- How do we find a strongly connected component $C \subseteq V$ containing u ?

- Con
vert
dire
cted
graph
to
the
form
of
 $D \subseteq V$

<https://eduassistpro.github.io>

- Use BFS to find the set $D \subseteq V$ of all vertices in V which are reachable in G from v .
- Find also the set $R \subseteq V$ of all vertices which are reachable from v .
- The strongly connected component C of G containing v is just the set $C = D \cap R$.
- Clearly, distinct strongly connected components are disjoint sets.

Greedy Method applied to graphs

- Given a directed graph $G = (V, E)$ and a vertex v , the *strongly connected component* of G containing v consists of all vertices $u \in V$ such that there is a path in G from v to u and a path from u to v .

- How do we find a strongly connected component $C \subseteq V$ containing u ?

- Con
vert
dire

of

he

<https://eduassistpro.github.io>

- Use BFS to find the set $D \subseteq V$ of all vertices in V which are reachable in G from v .
- Find also the set $R \subseteq V$ of all vertices which are reachable from v .
- The strongly connected component C of G containing v is just the set $C = D \cap R$.
- Clearly, distinct strongly connected components are disjoint sets.

Greedy Method applied to graphs

- Given a directed graph $G = (V, E)$ and a vertex v , the *strongly connected component* of G containing v consists of all vertices $u \in V$ such that there is a path in G from v to u and a path from u to v .

- How do we find a strongly connected component $C \subseteq V$ containing u ?

- Con
vert
dire

of

he

<https://eduassistpro.github.io>

- Use BFS to find the set $D \subseteq V$ of all vertices in V which are reachable in G from v .
- Find also the set $R \subseteq V$ of all vertices which are reachable from v .
- The strongly connected component C of G containing v is just the set $C = D \cap R$.
- Clearly, distinct strongly connected components are disjoint sets.

Greedy Method applied to graphs

- Let S_G be the set of all strongly connected components of a graph G .

Assignment Project Exam Help

Given a graph $G = (S_G, E^*)$ with its vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$ then there exists a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just at there

<https://eduassistpro.github.io>

- Clear G topological sorting of vertices.
- Topologically sort of a directed acyclic graph G in linear (enumeration) of its vertices $\sigma_i \in S_G$ such that if $(\sigma_i, \sigma_j) \in E^*$ then vertex σ_i precedes σ_j in such an ordering, i.e., $i < j$.
- How do we topologically sort a directed acyclic graphs?

Greedy Method applied to graphs

- Let S_G be the set of all strongly connected components of a graph G .
- We define a graph $\Sigma = (S_G, E^*)$ with vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$, there exists a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just in case there is a directed edge $v \in \sigma_1 \rightarrow u \in \sigma_2$ in G .
- How do we topologically sort a directed acyclic graphs?

Greedy Method applied to graphs

- Let S_G be the set of all strongly connected components of a graph G .

We define a graph $\Sigma = (S_G, E^*)$ with vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$, there is a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just if there is a directed edge $v_1 \rightarrow v_2$ in G for some $v_1 \in \sigma_1$ and $v_2 \in \sigma_2$.

- We define a graph $\Sigma = (S_G, E^*)$ with vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$, there is a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just if there is a directed edge $v_1 \rightarrow v_2$ in G for some $v_1 \in \sigma_1$ and $v_2 \in \sigma_2$.

<https://eduassistpro.github.io>

- Clearly, we can perform a topological sorting of vertices.

- Topologically sort or directed acyclic graphs in linear time by performing a topological sort of the graph Σ (enumeration) of its vertices $\sigma_i \in S_G$ such that if $(\sigma_i, \sigma_j) \in E^*$ then vertex σ_i precedes σ_j in such an ordering, i.e., $i < j$.
- How do we topologically sort a directed acyclic graphs?

Greedy Method applied to graphs

- Let S_G be the set of all strongly connected components of a graph G .

We define a graph $\Sigma = (S_G, E^*)$ with vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$, there is a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just if there is a directed edge $v_1 \rightarrow v_2$ in G for some $v_1 \in \sigma_1$ and $v_2 \in \sigma_2$.

- We define a graph $\Sigma = (S_G, E^*)$ with vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$, there is a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just if there is a directed edge $v_1 \rightarrow v_2$ in G for some $v_1 \in \sigma_1$ and $v_2 \in \sigma_2$.

<https://eduassistpro.github.io>

- Clearly, Σ has a topological sorting of vertices.
- Topological sort of a directed acyclic graph is a linear enumeration (enumeration) of its vertices $\sigma_i \in S_G$ such that if $(\sigma_i, \sigma_j) \in E^*$ then vertex σ_i precedes σ_j in such an ordering, i.e., $i < j$.
- How do we topologically sort a directed acyclic graphs?

Greedy Method applied to graphs

- Let S_G be the set of all strongly connected components of a graph G .

We define a graph $\Sigma = (S_G, E^*)$ with vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$, there is a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just if there is a directed edge $v \rightarrow w$ in G such that $v \in \sigma_1$ and $w \in \sigma_2$.

- We define a graph $\Sigma = (S_G, E^*)$ with vertices S_G and directed edges E^* between the strongly connected components so that if $\sigma_1 \in S_G$ and $\sigma_2 \in S_G$, there is a directed edge $\sigma_1 \rightarrow \sigma_2$ in E^* just if there is a directed edge $v \rightarrow w$ in G such that $v \in \sigma_1$ and $w \in \sigma_2$.

<https://eduassistpro.github.io>

- Clearly, Σ has a topological sorting of vertices.
- Topological sort of a directed acyclic graph is a linear enumeration (enumeration) of its vertices $\sigma_i \in S_G$ such that if $(\sigma_i, \sigma_j) \in E^*$ then vertex σ_i precedes σ_j in such an ordering, i.e., $i < j$.
- How do we topologically sort a directed acyclic graphs?

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge

Assignment Project Exam Help

- remove a node u from S ;

• <https://eduassistpro.github.io>

- remove edge e from the graph;

Add WeChat edu_assist_pro

- if graph has edges left, then return e
- else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge

Assignment Project Exam Help

- remove a node u from S ;

• <https://eduassistpro.github.io>

- remove edge e from the graph;

Add WeChat edu_assist_pro

- if graph has edges left, then return e
else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge
- while S is non-empty do
 - remove a node u from S ;

• <https://eduassistpro.github.io>

- remove edge e from the graph;

• if e has no other incoming edges then add e to L

- if graph has edges left, then return e

else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge

Assignment Project Exam Help

- while S is non-empty do
 - remove a node u from S ;

• <https://eduassistpro.github.io>

- remove edge e from the graph;

Add WeChat edu_assist_pro

- if graph has edges left, then return e

else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge
- while S is non-empty do
 - remove a node u from S ;

:
<https://eduassistpro.github.io>

- remove edge e from the graph;

Add WeChat edu_assist_pro

- if graph has edges left, then return e

else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge
- while S is non-empty do
 - remove a node u from S ;

:
<https://eduassistpro.github.io>

- remove edge e from the graph;

Add WeChat edu_assist_pro

- if u has no other incoming edges
 - if graph has edges left, then return e
 - else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge
- while S is non-empty do
 - remove a node u from S ;

:
<https://eduassistpro.github.io>

- remove edge e from the graph;

- if v has no other incoming edges
 - Add WeChat edu_assist_pro
- if graph has edges left, then return e
 - else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge
- while S is non-empty do
 - remove a node u from S ;

:
<https://eduassistpro.github.io>

- remove edge e from the graph;

- if u has no other incoming edges
 - Add WeChat edu_assist_pro
- if graph has edges left, then return e
else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

Topological sorting

- $L \leftarrow$ Empty list that will contain ordered elements
- $S \leftarrow$ Set of all nodes with no incoming edge
- while S is non-empty do
 - remove a node u from S ;

:
<https://eduassistpro.github.io>

- remove edge e from the graph;

- if u has no other incoming edges
 - if graph has edges left, then return e
 - else return L (a topologically sorted order)

Now it should be easy to use the Greedy Strategy to solve the problem of finding the fewest number of towers which you must equip with seismic sensors, so that all emergency transmission towers get activated.

An augmented Priority Queue

- We will need a priority queue which allows an efficient change of the key of an element in the queue, so we first need to extend the Heap data structure.

Assignment Project Exam Help

- We

way
to n

<https://eduassistpro.github.io>

- Thus, every element of A is of the form $(i, k(i))$ where $k(i)$ is the key of element i .

- Besides A , we may also have an array P of the same length which stores the position of element i in A , thus $A[P[i]] = (i, k(i))$.

- Changing the key of an element i is now an $O(\log n)$ operation: we look up its position $P[i]$ in A , change the key of the element in $A[P[i]]$ and then perform the Heappify operation to make sure the Heap property is being preserved.

An augmented Priority Queue

- We will need a priority queue which allows an efficient change of the key of an element in the queue, so we first need to extend the Heap data structure.

Assignment Project Exam Help

- We will use heaps represented by arrays; the left child of $A[i]$ is stored in $A[2i]$ and the right child in $A[2i + 1]$.

- We
way
to n

<https://eduassistpro.github.io>

- Thus, every element of A is of the form $(i, k(i))$ where $k(i)$ is the key of element i .
- Besides A , we may also have an array P of the same length which stores the position of element i in A , thus $A[P[i]] = (i, k(i))$.
- Changing the key of an element i is now an $O(\log n)$ operation: we look up its position $P[i]$ in A , change the key of the element in $A[P[i]]$ and then perform the Heappify operation to make sure the Heap property is being preserved.

An augmented Priority Queue

- We will need a priority queue which allows an efficient change of the key of an element in the queue, so we first need to extend the Heap data structure.

Assignment Project Exam Help

- We will use heaps represented by arrays; the left child of $A[i]$ is stored in $A[2i]$ and the right child in $A[2i + 1]$.

- We
way
to n

<https://eduassistpro.github.io>

- Thus, every element of A is of the form $(i, k(i))$ where $k(i)$ is the key of element i .

- Besides A , we may also have an array P of the same length which stores the position of element i in A , thus $A[P[i]] = (i, k(i))$.

- Changing the key of an element i is now an $O(\log n)$ operation: we look up its position $P[i]$ in A , change the key of the element in $A[P[i]]$ and then perform the Heappify operation to make sure the Heap property is being preserved.

An augmented Priority Queue

- We will need a priority queue which allows an efficient change of the key of an element in the queue, so we first need to extend the Heap data structure.

Assignment Project Exam Help

- We will use heaps represented by arrays; the left child of $A[i]$ is stored in $A[2i]$ and the right child in $A[2i + 1]$.

- We

way
to n

<https://eduassistpro.github.io>

- Thus, every element of A is of the form $(i, k(i))$ where $k(i)$ is the key of element i .

- Besides A , we may also have an array P of the same length which stores the position of element i in A , thus $A[P[i]] = (i, k(i))$.

- Changing the key of an element i is now an $O(\log n)$ operation: we look up its position $P[i]$ in A , change the key of the element in $A[P[i]]$ and then perform the Heappify operation to make sure the Heap property is being preserved.

An augmented Priority Queue

- We will need a priority queue which allows an efficient change of the key of an element in the queue, so we first need to extend the Heap data structure.

Assignment Project Exam Help

- We will use heaps represented by arrays; the left child of $A[i]$ is stored in $A[2i]$ and the right child in $A[2i + 1]$.

- We

way
to n

<https://eduassistpro.github.io>

s 1

- Thus, every element of A is of the form $(i, k(i))$ where $k(i)$ is the key of element i .

- Besides the array A which represents the heap, we have an array P of the same length which stores the position of each element in A , thus $A[P[i]] = (i, k(i))$.

- Changing the key of an element i is now an $O(\log n)$ operation: we look up its position $P[i]$ in A , change the key of the element in $A[P[i]]$ and then perform the Heappify operation to make sure the Heap property is being preserved.

An augmented Priority Queue

- We will need a priority queue which allows an efficient change of the key of an element in the queue, so we first need to extend the Heap data structure.

Assignment Project Exam Help

- We will use heaps represented by arrays; the left child of $A[i]$ is stored in $A[2i]$ and the right child in $A[2i + 1]$.

- We

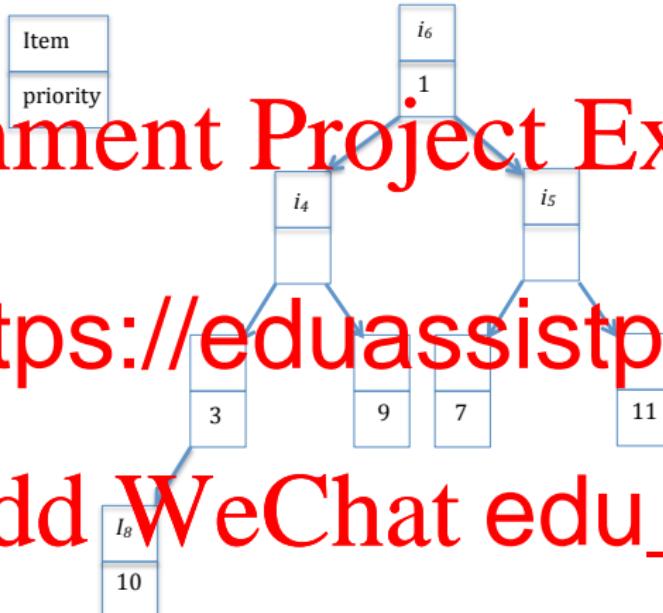
way
to n

<https://eduassistpro.github.io>

s 1

- Thus, every element of A is of the form $(i, k(i))$ where $k(i)$ is the key of element i .
- Besides the array A which represents the heap, we have an array P of the same length which stores the position of each element in A , thus $A[P[i]] = (i, k(i))$.
- Changing the key of an element i is now an $O(\log n)$ operation: we look up its position $P[i]$ in A , change the key of the element in $A[P[i]]$ and then perform the Heappify operation to make sure the Heap property is being preserved.

An augmented Priority Queue



Add WeChat edu_assist_pro

Heap Array							
i_6	i_4	i_5	i_2	i_3	i_1	i_7	i_8
1	2	6	3	9	7	11	10

Below the array indices 1 through 8 are aligned under each element.

Index Array							
i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
6	4	5	2	3	1	7	8

- Some of the most important applications of the Greedy Strategy are in graph algorithms.

Assignment Project Exam Help

- Assume we are given a directed graph $G = (V, E)$ with non-negative weights.

• We are interested in finding shortest paths from a source vertex v to all other vertices.

- For simplicity, we will assume that for every edge (u, v) , there is an edge (v, u) .

• The task is to find for every $u \in V$ the shortest path from v to u .

• This is accomplished by a very elegant greedy algorithm by Edsger Dijkstra already in 1959!

Add WeChat edu_assist_pro

- Some of the most important applications of the Greedy Strategy are in graph algorithms.

Assignment Project Exam Help

- Assume we are given a directed graph $G = (V, E)$ with **non-negative** weights.

• We are interested in finding shortest paths from a source vertex v to all other vertices.

- For simplicity, we will assume that for every edge (u, v) , v is reachable from u .

• The task is to find for every $u \in V$ the shortest path from v to u .

- This is accomplished by a very elegant greedy algorithm by Edsger Dijkstra already in 1959!

- Some of the most important applications of the Greedy Strategy are in graph algorithms.

Assignment Project Exam Help

- Assume we are given a directed graph $G = (V, E)$ with **non-negative** weights.
- We are asked to find the shortest path from a source vertex s to a target vertex t .

<https://eduassistpro.github.io/>

- For simplicity, we will assume that for every edge $(v, u) \in E$, $w(v, u) \geq 0$.

Add WeChat edu_assist_pro

- The task is to find for every $v \in V$ the shortest path from s to v .

- This is accomplished by a very elegant greedy algorithm by Edsger Dijkstra already in 1959!

- Some of the most important applications of the Greedy Strategy are in graph algorithms.
 - Assume we are given a directed graph $G = (V, E)$ with **non-negative** wei

<https://eduassistpro.github.io>

- For simplicity, we will assume that for every v there is a unique path from v to u .

- Add WeChat edu_assist_pr

- The task is to find for every $a \in V$ the set

- Some of the most important applications of the Greedy Strategy are in graph algorithms.

Assignment Project Exam Help

- Assume we are given a directed graph $G = (V, E)$ with **non-negative** weights.

- We are interested in finding shortest paths from a source node s to every other node v .

- For simplicity, we will assume that for every edge (u, v) , v is reachable from u .

- The task is to find for every $u \in V$ the shortest path from s to u .

- This is accomplished by a very elegant greedy algorithm by Edsger Dijkstra already in 1959!

- Some of the most important applications of the Greedy Strategy are in graph algorithms.

Assignment Project Exam Help

- Assume we are given a directed graph $G = (V, E)$ with **non-negative** weights.

- We are interested in finding shortest paths from a source vertex s to every other vertex v .

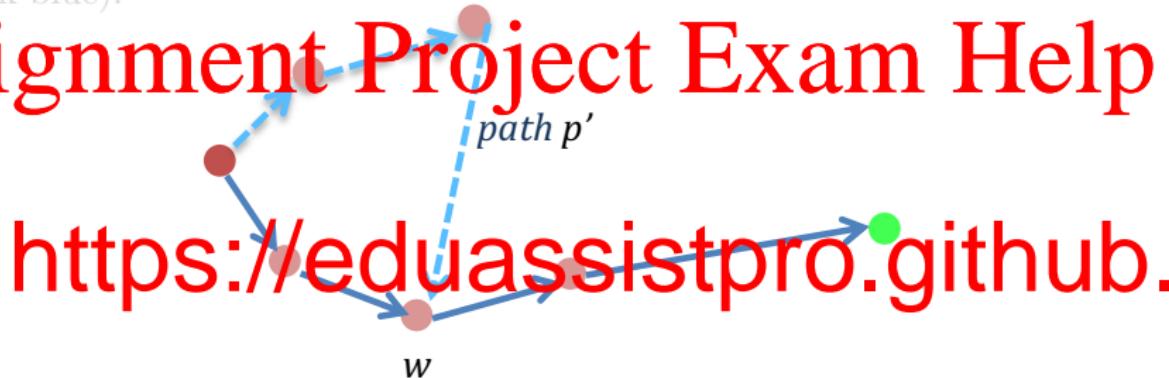
- For simplicity, we will assume that for every edge (u, v) , v is reachable from u .

- The task is to find for every $u \in V$ the shortest path from s to u .

- This is accomplished by a very elegant greedy algorithm by Edsger Dijkstra already in 1959!

Greedy Method applied to graphs: Shortest Paths

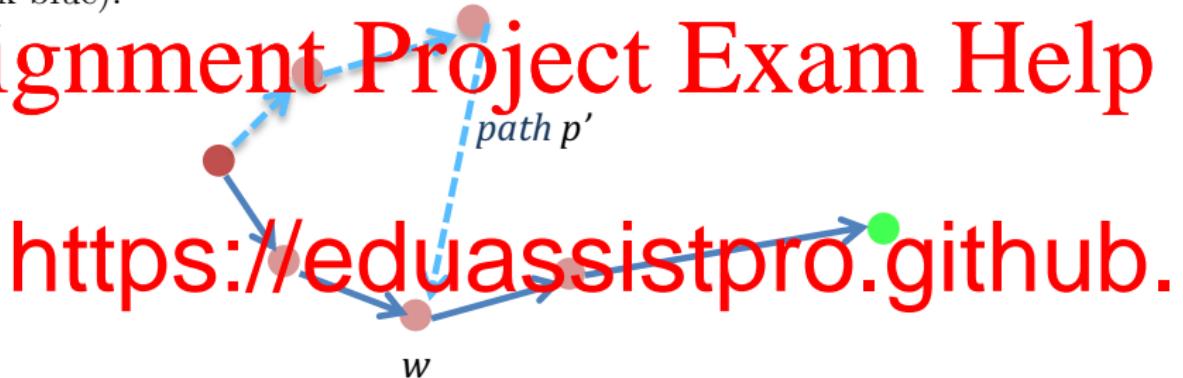
- We first prove a simple fact about shortest paths.
- Consider a shortest path p in G from a vertex v to a vertex z (shown in dark blue).



- We claim that for every vertex w on the path p from v to z , the path from v to w is just the truncation of the shortest path from v to z at w .
- Assume the opposite, that there is a shorter path from v to w (shown in dashed light blue) which is not an initial segment of the shortest path from v to z .
- However, in that case we could remove the part of the shortest path between v and z which is between v and w and replaced it with the light blue shorter path from v to w , thus obtaining a shorter path from v to z .
- Contradiction!

Greedy Method applied to graphs: Shortest Paths

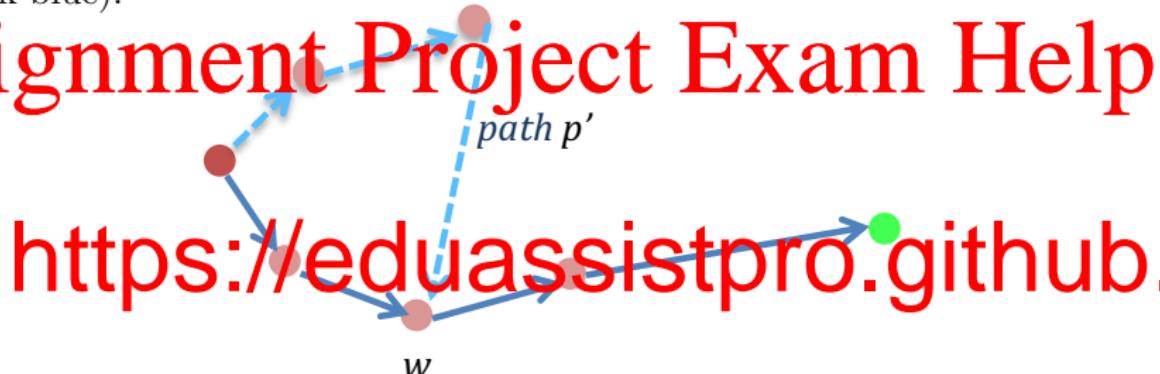
- We first prove a simple fact about shortest paths.
- Consider a shortest path p in G from a vertex v to a vertex z (shown in dark blue).



- We claim that for every vertex w on p , the path from v to w is just the truncation of the shortest path from v to z at w .
- Assume the opposite, that there is a shorter path from v to w (shown in dashed light blue) which is not an initial segment of the shortest path from v to z .
- However, in that case we could remove the part of the shortest path between v and z which is between v and w and replaced it with the light blue shorter path from v to w , thus obtaining a shorter path from v to z .
- Contradiction!

Greedy Method applied to graphs: Shortest Paths

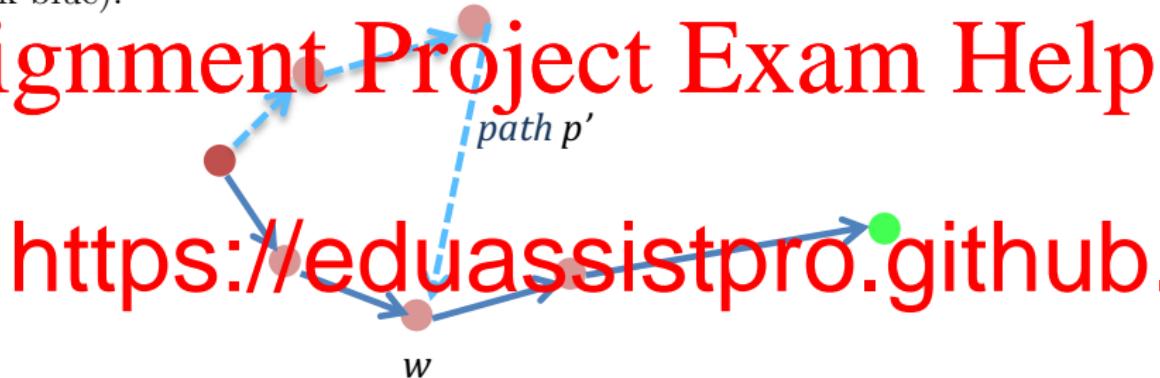
- We first prove a simple fact about shortest paths.
- Consider a shortest path p in G from a vertex v to a vertex z (shown in dark blue).



- We claim that for every vertex w on the path p , the shortest path from v to w is just the truncation of the shortest path from v to z at w .
- Assume the opposite, that there is a shorter path from v to w (shown in dashed light blue) which is not an initial segment of the shortest path from v to z .
- However, in that case we could remove the part of the shortest path between v and z which is between v and w and replaced it with the light blue shorter path from v to w , thus obtaining a shorter path from v to z .
- Contradiction!

Greedy Method applied to graphs: Shortest Paths

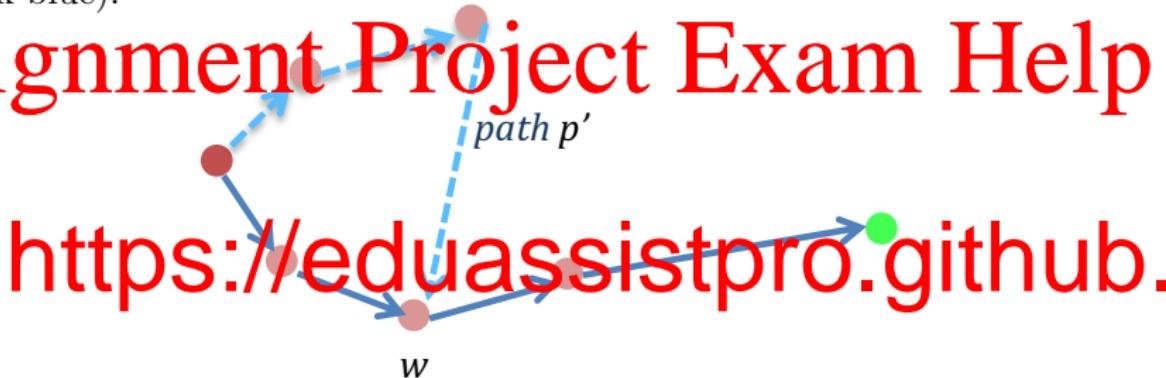
- We first prove a simple fact about shortest paths.
- Consider a shortest path p in G from a vertex v to a vertex z (shown in dark blue).



- We claim that for every vertex w on the path p , the shortest path from v to w is just the truncation of the shortest path from v to z at w .
- Assume the opposite, that there is a shorter path from v to w (shown in dashed light blue) which is not an initial segment of the shortest path from v to z .
- However, in that case we could remove the part of the shortest path between v and z which is between v and w and replaced it with the light blue shorter path from v to w , thus obtaining a shorter path from v to z .
- Contradiction!

Greedy Method applied to graphs: Shortest Paths

- We first prove a simple fact about shortest paths.
- Consider a shortest path p in G from a vertex v to a vertex z (shown in dark blue).

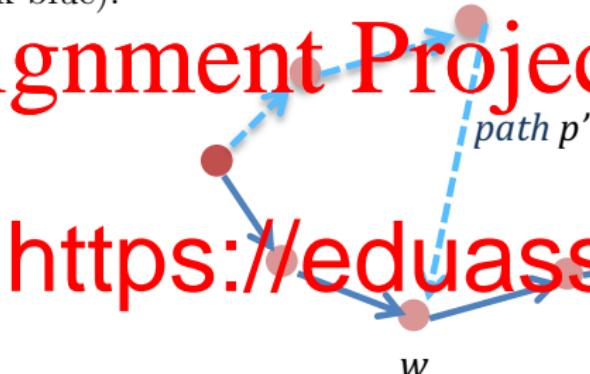


- We claim that for every vertex w on the path p , the shortest path from v to w is just the truncation of the shortest path from v to z at w .
- Assume the opposite, that there is a shorter path from v to w (shown in dashed light blue) which is not an initial segment of the shortest path from v to z .
- However, in that case we could remove the part of the shortest path between v and z which is between v and w and replaced it with the light blue shorter path from v to w , thus obtaining a shorter path from v to z .
- Contradiction!

Greedy Method applied to graphs: Shortest Paths

- We first prove a simple fact about shortest paths.
- Consider a shortest path p in G from a vertex v to a vertex z (shown in dark blue).

Assignment Project Exam Help



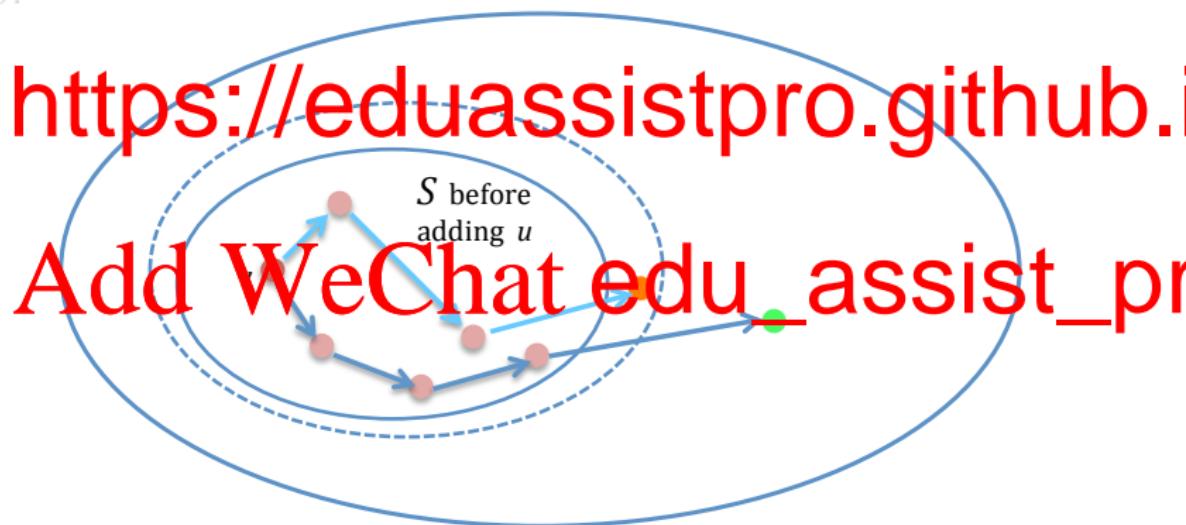
<https://eduassistpro.github.io>

- We claim that for every vertex w on the path p , the shortest path from v to w is just the truncation of the shortest path from v to z at w .
- Assume the opposite, that there is a shorter path from v to w (shown in dashed light blue) which is not an initial segment of the shortest path from v to z .
- However, in that case we could remove the part of the shortest path between v and z which is between v and w and replaced it with the light blue shorter path from v to w , thus obtaining a shorter path from v to z .
- Contradiction!

Dijkstra's Shortest Paths Algorithm

- The algorithm builds a set S of vertices for which the shortest path has been already established, starting with a single source vertex $S = \{v\}$ and adding one vertex at a time.

Assignment Project Exam Help
has the shortest path from v to u with all intermediate vertices already in S .

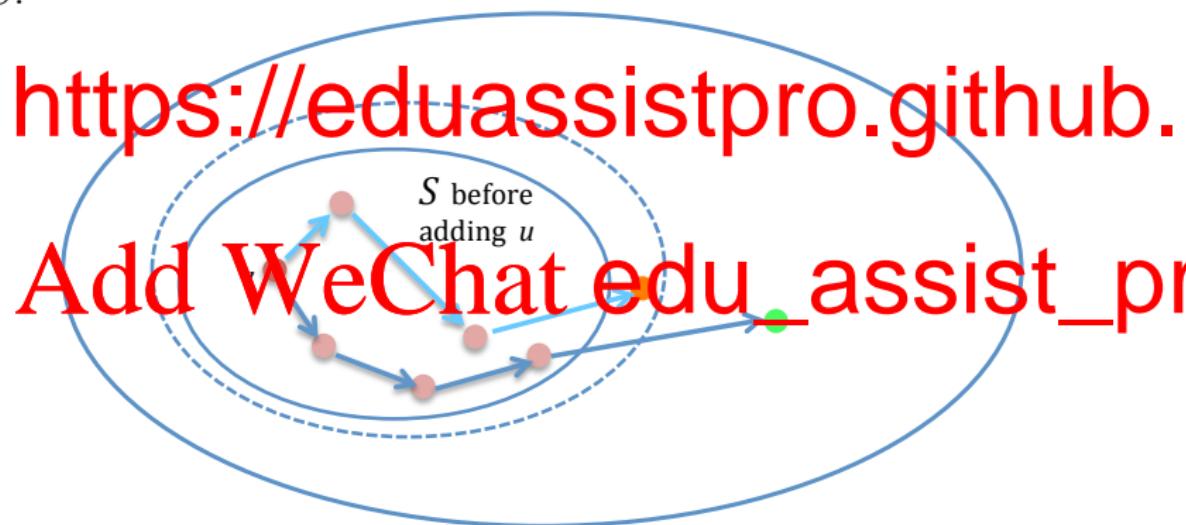


Dijkstra's Shortest Paths Algorithm

- The algorithm builds a set S of vertices for which the shortest path has been already established, starting with a single source vertex $S = \{v\}$ and adding one vertex at a time.

Assignment Project Exam Help

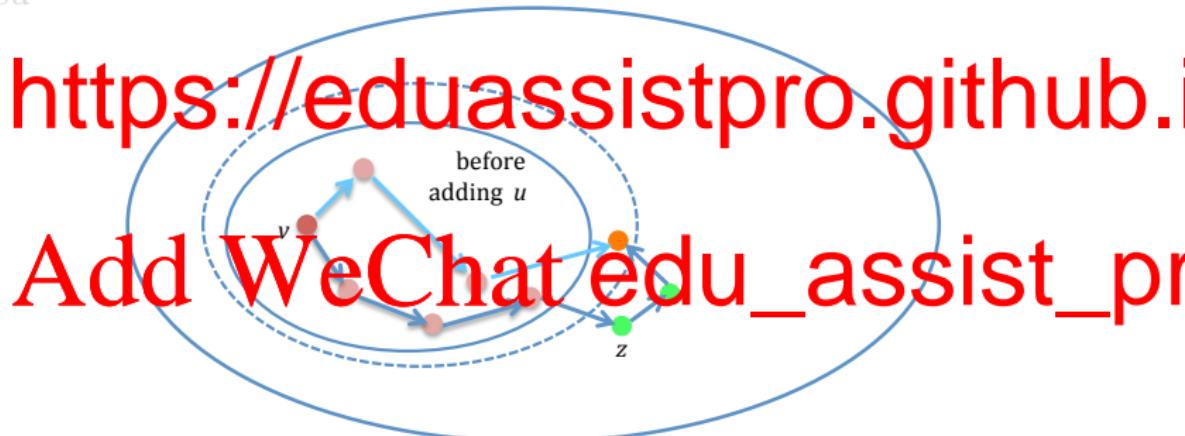
- At each stage of the construction, we add the element $u \in V \setminus S$ which has the shortest path from v to u with all intermediate vertices already in S .



Add WeChat edu_assist_pro

Dijkstra's Shortest Paths Algorithm

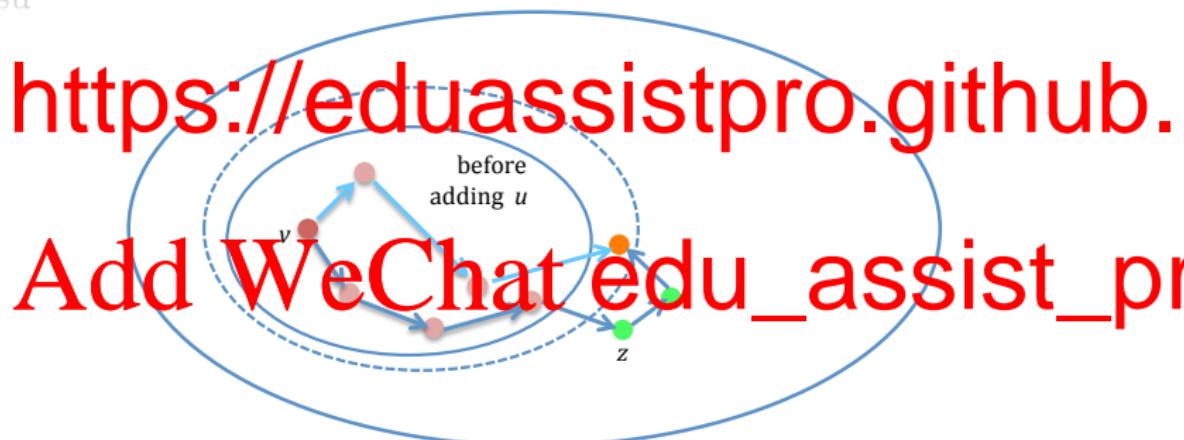
- Why does this produce a shortest path from v to u in G ?
- Assume the opposite, that there exists a shorter path from v to u in G .
By our choice of u , such a path cannot be entirely in S .
- Let z be the first vertex outside S (as it was just prior to addition of u)
on su



- Since there are no negative weight edges the path from v to such z would be shorter than the path from v to u , contradicting our choice of u .

Dijkstra's Shortest Paths Algorithm

- Why does this produce a shortest path from v to u in G ?
- Assume the opposite, that there exists a shorter path from v to u in G . By our choice of u such a path cannot be entirely in S .
- Let z be the first vertex outside S (as it was just prior to addition of u)



- Since there are no negative weight edges the path from v to such z would be shorter than the path from v to u , contradicting our choice of u .

Dijkstra's Shortest Paths Algorithm

- Why does this produce a shortest path from v to u in G ?
 - Assume the opposite, that there exists a shorter path from v to u in G . By our choice of u such a path cannot be entirely in S .
 - Let z be the first vertex outside S (as it was just prior to addition of u) on su

<https://eduassistpro.github.io>

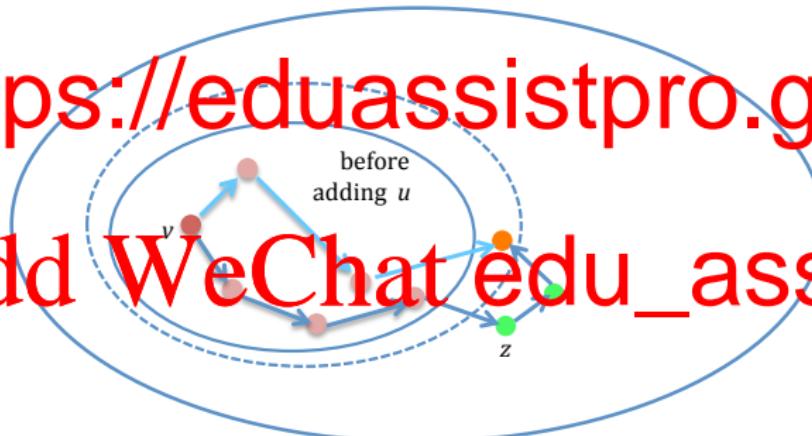
Add WeChat `edu_assist_pr`

Dijkstra's Shortest Paths Algorithm

- Why does this produce a shortest path from v to u in G ?
- Assume the opposite, that there exists a shorter path from v to u in G . By our choice of u such a path cannot be entirely in S .
- Let z be the first vertex outside S (as it was just prior to addition of u)

<https://eduassistpro.github.io>

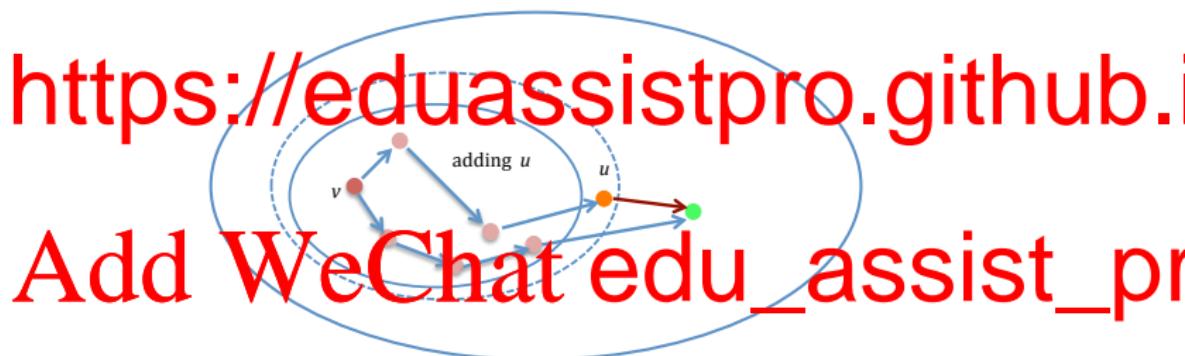
Add WeChat `edu_assist_pro`



- Since there are no negative weight edges the path from v to such z would be shorter than the path from v to u , contradicting our choice of u .

Dijkstra's Shortest Paths Algorithm

- How is this construction done efficiently?
 - Initially all vertices except v are placed in a heap based priority queue with additional Position array, with the weight $w(v, u)$ if $(v, u) \in E$ or ∞ as the key;
- After each step, every element u will be paired with its $l_{S,v}(u)$ of the shortest path from v to u which has all intermediate vertices on such a path in S .



- We always pop the element u from the priority queue which has the smallest key and add it to set S .
- We then look for all elements $z \in V \setminus S$ for which $(u, z) \in E$ and if $l_{S,v}(u) + w(u, z) < l_{S,v}(z)$ we update the key of z to the value $l_{S,v}(u) + w(u, z)$.

Dijkstra's Shortest Paths Algorithm

- How is this construction done efficiently?
 - Initially all vertices except v are placed in a heap based priority queue with additional Position array, with the weight $w(v, u)$ if $(v, u) \in E$ or ∞ as the key;
- Assignment Project Exam Help
- $\text{lhs}_{S,v}(u)$ of the shortest path from v to u which has all intermediate vertices on such a path in S .

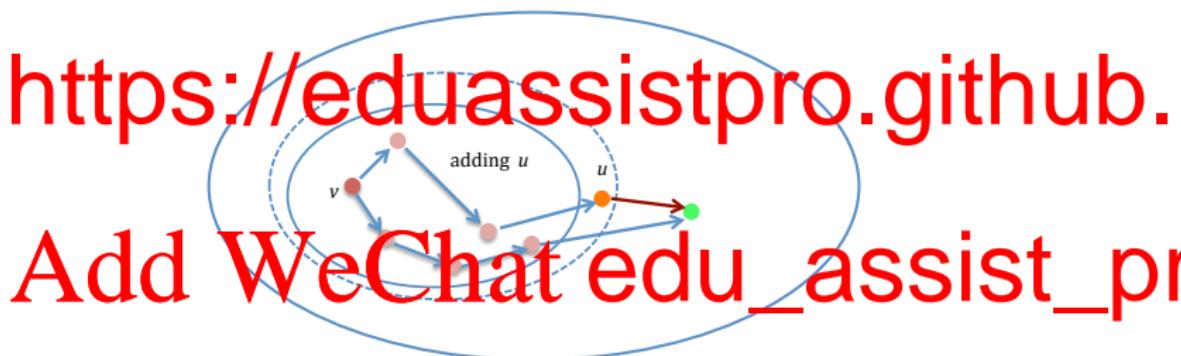
<https://eduassistpro.github.io>

Add WeChat edu_assist_pro

-
- We always pop the element u from the priority queue which has the smallest key and add it to set S .
 - We then look for all elements $z \in V \setminus S$ for which $(u, z) \in E$ and if $\text{lhs}_{S,v}(u) + w(u, z) < \text{lhs}_{S,v}(z)$ we update the key of z to the value $\text{lhs}_{S,v}(u) + w(u, z)$.

Dijkstra's Shortest Paths Algorithm

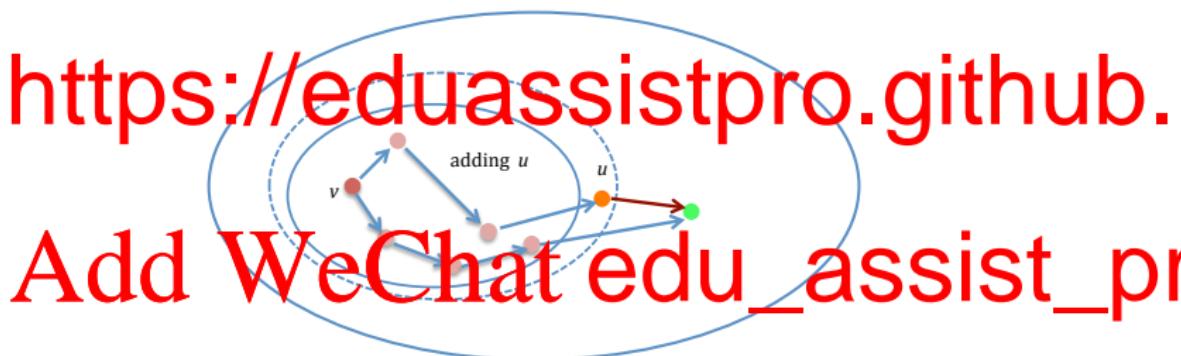
- How is this construction done efficiently?
- Initially all vertices except v are placed in a heap based priority queue with additional Position array, with the weight $w(v, u)$ if $(v, u) \in E$ or ∞ as the key;
- As we progress, the key of each element u will be updated with length $\text{lh}_{S, v}(u)$ of the shortest path from v to u which has all intermediate vertices on such a path in S .



- We always pop the element u from the priority queue which has the smallest key and add it to set S .
- We then look for all elements $z \in V \setminus S$ for which $(u, z) \in E$ and if $\text{lh}_{S, v}(u) + w(u, z) < \text{lh}_{S, v}(z)$ we update the key of z to the value $\text{lh}_{S, v}(u) + w(u, z)$.

Dijkstra's Shortest Paths Algorithm

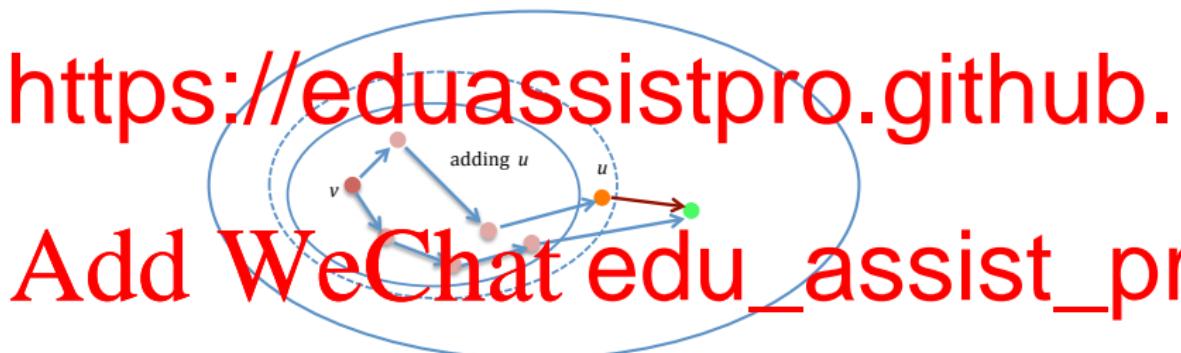
- How is this construction done efficiently?
- Initially all vertices except v are placed in a heap based priority queue with additional Position array, with the weight $w(v, u)$ if $(v, u) \in E$ or ∞ as the key;
- As we progress, the key of each element u will be updated with length $\text{lh}_{S, v}(u)$ of the shortest path from v to u which has all intermediate vertices on such a path in S .



- We always pop the element u from the priority queue which has the smallest key and add it to set S .
- We then look for all elements $z \in V \setminus S$ for which $(u, z) \in E$ and if $\text{lh}_{S, v}(u) + w(u, z) < \text{lh}_{S, v}(z)$ we update the key of z to the value $\text{lh}_{S, v}(u) + w(u, z)$.

Dijkstra's Shortest Paths Algorithm

- How is this construction done efficiently?
- Initially all vertices except v are placed in a heap based priority queue with additional Position array, with the weight $w(v, u)$ if $(v, u) \in E$ or ∞ as the key;
- As we progress, the key of each element u will be updated with length $lh_{S, v}(u)$ of the shortest path from v to u which has all intermediate vertices on such a path in S .

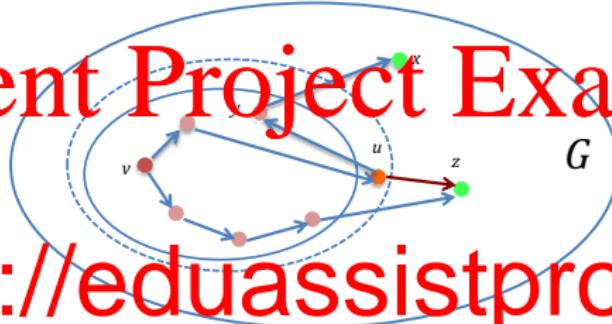


- We always pop the element u from the priority queue which has the smallest key and add it to set S .
- We then look for all elements $z \in V \setminus S$ for which $(u, z) \in E$ and if $lh_{S, v}(u) + w(u, z) < lh_{S, v}(z)$ we update the key of z to the value $lh_{S, v}(u) + w(u, z)$.

Dijkstra's Shortest Paths Algorithm

- Why is this enough; i.e., why the only shortest paths which could have changed have u as the last vertex in S ?

Assignment Project Exam Help



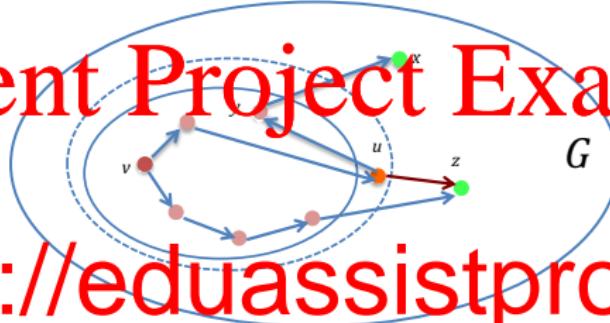
<https://eduassistpro.github.io>

- Assume opposite that the shortest path to a vertex x has changed when u was added, and that instead of u and before x on such a new shortest path.
- However, this is impossible because it would add y with a vertex u on that path not belonging to S .
- If graph G has n vertices and m edges, then each edge is inspected only once, when it is an outgoing edge from the most recently added vertex; updating a vertex key takes $O(\log n)$ many steps.
- Thus, in total, the algorithm runs in time $O(m \log n)$.

Dijkstra's Shortest Paths Algorithm

- Why is this enough; i.e., why the only shortest paths which could have changed have u as the last vertex in S ?

Assignment Project Exam Help



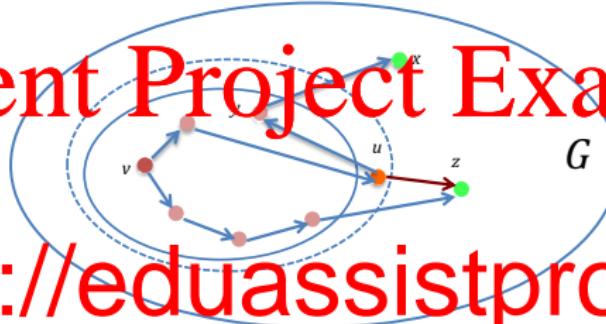
<https://eduassistpro.github.io>

- Assume opposite that the shortest path to a vertex x has changed when u was added, and that instead of u and before x on such a new shortest path.
- However, this is not possible because it would add y with a vertex u on that path not belonging to S .
- If graph G has n vertices and m edges, then each edge is inspected only once, when it is an outgoing edge from the most recently added vertex; updating a vertex key takes $O(\log n)$ many steps.
- Thus, in total, the algorithm runs in time $O(m \log n)$.

Dijkstra's Shortest Paths Algorithm

- Why is this enough; i.e., why the only shortest paths which could have changed have u as the last vertex in S ?

Assignment Project Exam Help



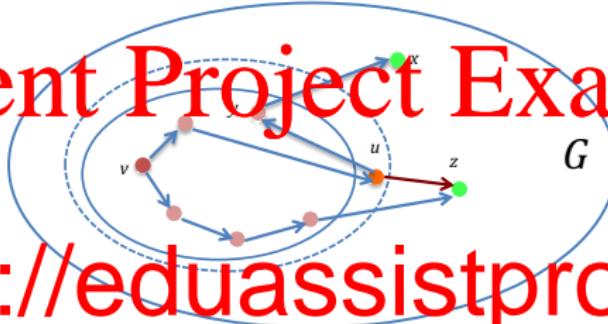
<https://eduassistpro.github.io>

- Assume opposite that the shortest path to a vertex x has changed when u was added, and that instead of u and before x on such a new shortest path.
- However, this is not possible because it would add y with a vertex u on that path not belonging to S .
- If graph G has n vertices and m edges, then each edge is inspected only once, when it is an outgoing edge from the most recently added vertex; updating a vertex key takes $O(\log n)$ many steps.
- Thus, in total, the algorithm runs in time $O(m \log n)$.

Dijkstra's Shortest Paths Algorithm

- Why is this enough; i.e., why the only shortest paths which could have changed have u as the last vertex in S ?

Assignment Project Exam Help



<https://eduassistpro.github.io>

- Assume opposite that the shortest path to a vertex x has changed when u was added, and that instead of u and before x on such a new shortest path.
- However, this is not possible because it would add y with a vertex u on that path not belonging to S .
- If graph G has n vertices and m edges, then each edge is inspected only once, when it is an outgoing edge from the most recently added vertex; updating a vertex key takes $O(\log n)$ many steps.
- Thus, in total, the algorithm runs in time $O(m \log n)$.

Greedy Method for graphs: Minimum Spanning Trees

- **Definition:** A minimum spanning tree T of a connected graph G is a subgraph of G (with the same set of vertices) which is a tree, and among all such trees it is of minimal total length of all edges in T .

Assignment Project Exam Help

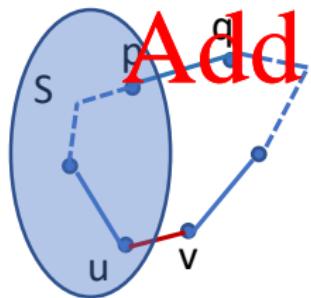
- **Lemma:** Let G be a connected graph with all lengths of edges E of G distinct. As

min

below

<https://eduassistpro.github.io>

- **Proof:**



- Add WeChat edu_assist_pro
- Assume that T is a minimum spanning tree of G that contains such an edge $e = (u, v)$.
 - Since T is a spanning tree, there exists a path from u to v within such a tree.

Greedy Method for graphs: Minimum Spanning Trees

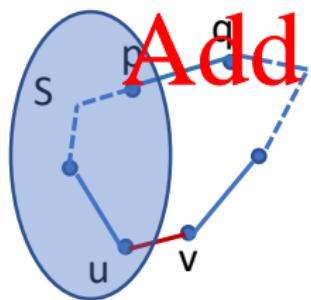
- **Definition:** A minimum spanning tree T of a connected graph G is a subgraph of G (with the same set of vertices) which is a tree, and among all such trees it is of minimal total length of all edges in T .

Assignment Project Exam Help

- **Lemma:** Let G be a connected graph with all lengths of edges E of G distinct. As $\min_{\substack{S \subseteq V \\ S \neq \emptyset}} \sum_{e \in E(S)} \text{length}(e)$ must

<https://eduassistpro.github.io>

- Proof:



- Add WeChat edu_assist_pro
- Assume that T is a minimum spanning tree of G that contains such an edge $e = (u, v)$.
 - Since T is a spanning tree, there exists a path from u to v within such a tree.

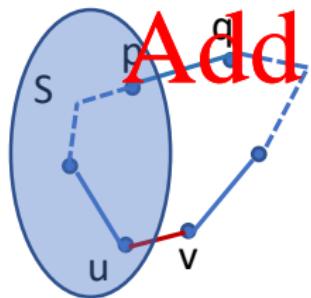
- **Definition:** A minimum spanning tree T of a connected graph G is a subgraph of G (with the same set of vertices) which is a tree, and among all such trees it is of minimal total length of all edges in T .

Assignment Project Exam Help

- **Lemma:** Let G be a connected graph with all lengths of edges E of G distinct. As \min_{below} $\notin S$ and is of e must

<https://eduassistpro.github.io>

- **Proof:**



- Assume for a contradiction that T is a minimum spanning tree that contains such an edge $e = (u, v)$.
- Since T is a spanning tree, there exists a path from u to v within such a tree.

Greedy Method for graphs: Minimum Spanning Trees

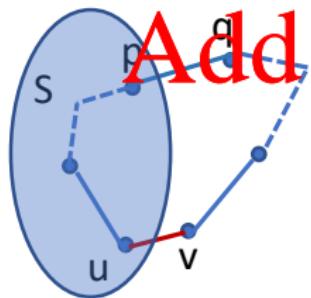
- **Definition:** A minimum spanning tree T of a connected graph G is a subgraph of G (with the same set of vertices) which is a tree, and among all such trees it is of minimal total length of all edges in T .

Assignment Project Exam Help

- **Lemma:** Let G be a connected graph with all lengths of edges E of G distinct. As \min_{below} $\notin S$ and is of e must

<https://eduassistpro.github.io>

- **Proof:**



- Add WeChat edu_assist_pr
- Assume for a contradiction that S is a minimum spanning tree of G and let $e = (u, v)$ be an edge in $G \setminus S$.
 - Since T is a spanning tree, there exists a path from u to v within such a tree.

Greedy Method for graphs: Minimum Spanning Trees



- However, the edge (p, q) belongs to T and must have a length larger than the edge (u, v) (minimum length edge with one end in S and one end outside S).

- Since $u \in S$ and $v \notin S$, this path must have left S at a certain point.
- Assume that p is the last vertex on this path which is in S and $q \notin S$ the vertex immediately after p on that path.

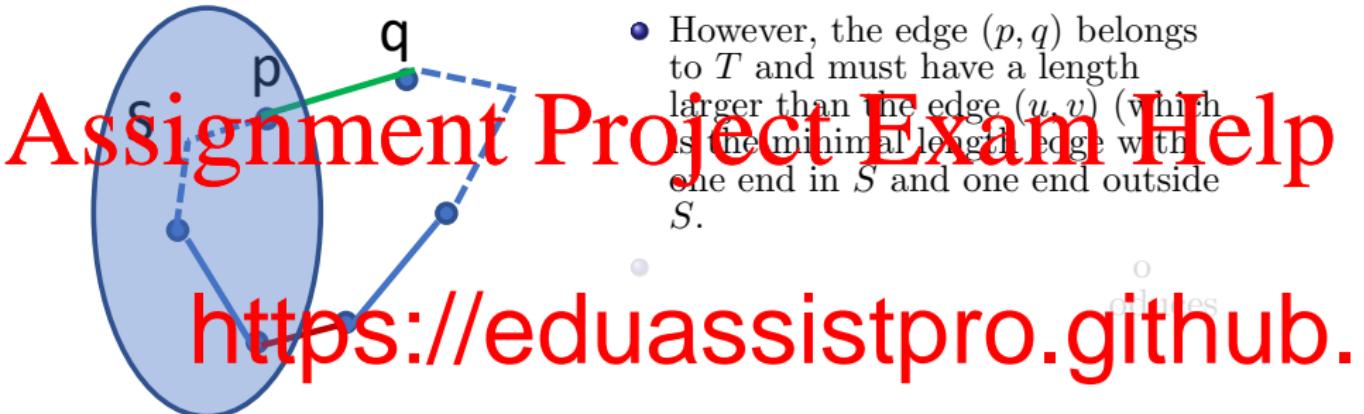
• Howe
with
than th
would r
smaller weight, contradicting our
assumption that T is a minimum
spanning tree.

Greedy Method for graphs: Minimum Spanning Trees



- Since $u \in S$ and $v \notin S$, this path must have left S at a certain point.
- Assume that p is the last vertex on this path which is in S and $q \notin S$ the vertex immediately after p on that path.
- However, the edge (p, q) belongs to T and must have a length larger than the edge (u, v) (minimum length edge with one end in S and one end outside S).
- How ever, with than th would r smaller weight, contradicting our assumption that T is a minimum spanning tree.

Greedy Method for graphs: Minimum Spanning Trees



- However, the edge (p, q) belongs to T and must have a length larger than the edge (u, v) (which is the minimal length edge with one end in S and one end outside S).

- Since $u \in S$ and $v \notin S$, this path must have left S at a certain point.
- Assume that p is the last vertex on this path which is in S and $q \notin S$ the vertex immediately after p on that path.

• Howe
with
than th
would r
smaller weight, contradicting our
assumption that T is a minimum
spanning tree.

Greedy Method for graphs: Minimum Spanning Trees



- However, the edge (p, q) belongs to T and must have a length larger than the edge (u, v) (which is the minimal length edge with one end in S and one end outside S).

• <https://eduassistpro.github.io>

- Since $u \in S$ and $v \notin S$, this path must have left S at a certain point.
- Assume that p is the last vertex on this path which is in S and $q \notin S$ the vertex immediately after p on that path.

• Howe
with
than th
would r
smaller weight, contradicting our
assumption that T is a minimum
spanning tree.

Greedy Method for graphs: Minimum Spanning Trees



- However, the edge (p, q) belongs to T and must have a length larger than the edge (u, v) (which is the minimal length edge with one end in S and one end outside S).

- Since $u \in S$ and $v \notin S$, this path must have left S at a certain point.
- Assume that p is the last vertex on this path which is in S and $q \notin S$ the vertex immediately after p on that path.
- Howe
with
than th
would r
smaller weight, contradicting our assumption that T is a minimum spanning tree.

The Kruskal Algorithm

Assignment Project Exam Help

- We order the edges E in a non-decreasing order with respect to their weight.

• We can use the following URL:

<https://eduassistpro.github.io/>

- An edge e_i is added at a round i of construction whenever its addition does not introduce a cycle in the graph constructed till now.

- If adding an edge does not introduce a cycle, the edge is added to the MST.
- The process terminates when the list of all edges has been exhausted.

The Kruskal Algorithm

Assignment Project Exam Help

- We order the edges E in a non-decreasing order with respect to their weight

• We can use the following algorithm:

<https://eduassistpro.github.io>

- An edge e_i is added at a round i of construction whenever its addition does not introduce a cycle in the graph constructed till now.

- If adding an edge does not introduce a cycle, the edge is added to the MST.
- The process terminates when the list of all edges has been exhausted.

The Kruskal Algorithm

Assignment Project Exam Help

- We order the edges E in a non-decreasing order with respect to their weight

- We build

<https://eduassistpro.github.io>

- An edge e_i is added at a round i of construction whenever its addition does not introduce a cycle in the graph constructed till now.

- If adding an edge does not introduce a cycle, the edge is added to the MST.
- The process terminates when the list of all edges has been exhausted.

The Kruskal Algorithm

Assignment Project Exam Help

- We order the edges E in a non-decreasing order with respect to their weight

- We build a Minimum Spanning Tree

<https://eduassistpro.github.io>

- An edge e_i is added at a round i of construction whenever its addition does not introduce a cycle in the graph constructed till now.

- If adding an edge does not introduce a cycle, the edge is added to the MST.

- The process terminates when the list of all edges has been exhausted.

The Kruskal Algorithm

Assignment Project Exam Help

- We order the edges E in a non-decreasing order with respect to their weight

- We begin with an empty set of edges.

<https://eduassistpro.github.io>

- An edge e_i is added at a round i of construction whenever its addition does not introduce a cycle in the graph constructed till now.

- If adding an edge does introduce a cycle, then the edge is discarded.

- The process terminates when the list of all edges has been exhausted.

The Kruskal Algorithm

Assignment Project Exam Help

- We order the edges E in a non-decreasing order with respect to their weight
- We begin with an empty set of edges.
- An edge e_i is added at a round i of construction whenever its addition does not introduce a cycle in the graph constructed till now.
- If adding an edge does introduce a cycle, the edge is discarded.
- The process terminates when the list of all edges has been exhausted.

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.
Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Consider the set S of edges produced by the Kruskal algorithm up to the moment when edge e was added.
- The set S is a spanning tree of G .
- Until this moment no edge with one end in S has been considered because otherwise it would have formed a cycle.
- Consequently, edge $e = (u, v)$ is the shortest edge connecting a vertex in S and a vertex not in S , and by the previous lemma it must belong to every spanning tree of G .
- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.
- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Add WeChat edu_assist_pro

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.
- Consequently, e connects two components of $G - E(S)$.
- The weight of e is less than or equal to the weight of every other edge connecting two components of $G - E(S)$.
- Until this moment no edge with one end in S has been considered because otherwise it would have formed a cycle.
- Consequently, edge $e = (u, v)$ is the shortest edge connecting two components of $G - E(S)$ and by the previous lemma it must belong to every spanning tree.
- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.
- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Add WeChat edu_assist_pro

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Consequently, u and v were previously in different sets S_u and S_v .
- The set $S_u \cup S_v$ is a spanning tree.
- Until this moment no edge with one end in S_u and one in S_v has been considered because otherwise it would have formed a cycle.
- Consequently, edge $e = (u, v)$ is the shortest edge connecting S_u and S_v .
and by the previous lemma it must belong to every spanning tree.
- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.
- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Con to algo

- The

- Until this moment no edge with one end in S has been considered because otherwise it would form a cycle.

- Consequently, edge $e = (u, v)$ is the shortest and by the previous lemma it must belong to every sp

- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.

- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Con
to
algo
- The

u
 S

<https://eduassistpro.github.io>

- Until this moment no edge with one end in S has been considered because otherwise it would form a cycle.
- Consequently, edge $e = (u, v)$ is the shortest and by the previous lemma it must belong to every spanning tree.
- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.
- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Con to algo

- The

- Until this moment no edge with one end in u has been considered because otherwise it would h S

- forming a cycle.
- Consequently, edge $e = (u, v)$ is the shortest and by the previous lemma it must belong to every sp

- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.

- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Con to algo
- The
- Until this moment no edge with one end in S has been considered because otherwise it would h forming a cycle.
- Consequently, edge $e = (u, v)$ is the shortest and by the previous lemma it must belong to every sp
- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.
- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Con
to
algo

<https://eduassistpro.github.io>

- The
- Until this moment no edge with one end in S has been considered because otherwise it would h
forming a cycle.
- Consequently, edge $e = (u, v)$ is the shortest and by the previous lemma it must belong to every sp
- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.
- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Add WeChat edu_assist_pro

Minimum Spanning Trees: the Kruskal Algorithm

Claim: The Kruskal algorithm produces a minimal spanning tree, and if all weights are distinct, then such a Minimum Spanning Tree is unique.

Proof: We consider the case when all weights are distinct.

- Consider an arbitrary edge $e = (u, v)$ added in the course of the Kruskal algorithm.

- Con to algo

- The / u

- Until this moment no edge with one end in S has been considered because otherwise it would h forming a cycle.

- Consequently, edge $e = (u, v)$ is the shortest and by the previous lemma it must belong to every sp

- Thus, the set of edges produced by the Kruskal algorithm is a subset of the set of edges of every spanning tree.

- But the structure produced by the Kruskal algorithm is clearly cycle-free and it spans the graph, otherwise another edge could be added.

Efficient implementation of the Kruskal Algorithm

- To efficiently implement the Kruskal algorithm, we need a useful data structure for storing sets of elements, called the Union-Find.

Assignment Project Exam Help

① $\text{MAKEUNIONIND}(S)$, which, given a set S returns a structure in

<https://eduassistpro.github.io>

② which v belongs. Such operation should time $O(\log n)$, we explain later.

③ $\text{UNION}(A, B)$, which, given (the labels of) the data structure by replacing sets A and B with the set $A \cup B$. A sequence of k consecutive UNION operations should run in time $O(k \log k)$.

Efficient implementation of the Kruskal Algorithm

- To efficiently implement the Kruskal algorithm, we need a useful data structure for storing sets of elements, called the Union-Find.
- Such a data structure has to support three operations:

① $\text{MAKEUNIONIND}(S)$, which, given a set S returns a structure in

②

which v belongs. Such operation should
time $\mathcal{O}(\log n)$, we explain later.

③

$\text{UNION}(A, B)$, which, given (the labels of) two sets A and B , merges the
data structure by replacing sets A and B with the set $A \cup B$. A
sequence of k consecutive UNION operations should run in time
 $O(k \log k)$.

- To efficiently implement the Kruskal algorithm, we need a useful data structure for storing sets of elements, called the Union-Find.
- Such a data structure has to support three operations:
 - ① $\text{MAKEUNIONIND}(S)$, which, given a set S returns a structure in

<https://eduassistpro.github.io>

- ② which v belongs. Such operation should time $O(\log n)$, we explain later.
- ③ $\text{UNION}(A, B)$, which, given (the labels of) two sets A and B , changes the data structure by replacing sets A and B with the set $A \cup B$. A sequence of k consecutive UNION operations should run in time $O(k \log k)$.

Efficient implementation of the Kruskal Algorithm

- To efficiently implement the Kruskal algorithm, we need a useful data structure for storing sets of elements, called the Union-Find.
- Such a data structure has to support three operations:
 - ① $\text{MAKEUNIONIND}(S)$, which, given a set S returns a structure in

② <https://eduassistpro.github.io>

which v belongs. Such operation should
time $O(\log n)$ as we explain later.

et to

(1) or

- ③ $\text{UNION}(A, B)$, which, given (the labels of) two sets A and B , merges the data structure by replacing sets A and B with the set $A \cup B$. A sequence of k consecutive UNION operations should run in time $O(k \log k)$.

Efficient implementation of the Kruskal Algorithm

- To efficiently implement the Kruskal algorithm, we need a useful data structure for storing sets of elements, called the Union-Find.
- Such a data structure has to support three operations:
 - ① $\text{MAKEUNIONIND}(S)$, which, given a set S returns a structure in

<https://eduassistpro.github.io>

- ② which v belongs. Such operation should time $O(\log n)$ as we explain later.

- ③ $\text{UNION}(A, B)$, which, given (the labels of) two sets A and B , changes the data structure by replacing sets A and B with the set $A \cup B$. A sequence of k consecutive UNION operations should run in time $O(k \log k)$.

Efficient implementation of the Kruskal Algorithm

- Note that we do not give the run time of a single UNION operation but of a sequence of k consecutive such operations.

Assignment Project Exam Help
multiple-choice analysis selected answers; it (essentially) estimates average cost of an operation in a sequence of operations, in this case $\log k$.

- We Uni can a <https://eduassistpro.github.io>
- The simplest implementation of the Union-Find of: **Add WeChat edu_assist_pr**
 - ① an array A such that $A[i] = j$ mea
 - ② an array B such that $B[i]$ contains the number of elements in the set labeled by i (which can be 0) and pointers to the first and last element of a linked list of elements of the set labeled by i .

Efficient implementation of the Kruskal Algorithm

- Note that we do not give the run time of a single UNION operation but of a sequence of k consecutive such operations.

Assignment Project Exam Help

- Such time complexity analysis is called *amortized analysis*; it (essentially) estimates average cost of an operation in a sequence of operations, in this case $\log k$.

- We Uni can a

<https://eduassistpro.github.io>

- The simplest implementation of the Union-Find of:

Add WeChat edu_assist_pr

- ① an array A such that $A[i] = j$ mea

j ;

- ② an array B such that $B[i]$ contains the number of elements in the set labeled by i (which can be 0) and pointers to the first and last element of a linked list of elements of the set labeled by i .

Efficient implementation of the Kruskal Algorithm

- Note that we do not give the run time of a single UNION operation but of a sequence of k consecutive such operations.

Assignment Project Exam Help

- Such time complexity analysis is called *amortized analysis*; it (essentially) estimates average cost of an operation in a sequence of operations, in this case $\log k$.

- We can a <https://eduassistpro.github.io>

- The simplest implementation of the Union-Find of:

Add WeChat edu_assist_pr

- ① an array A such that $A[i] = j$ means i is connected to j ;

- ② an array B such that $B[i]$ contains the number of elements in the set labeled by i (which can be 0) and pointers to the first and last element of a linked list of elements of the set labeled by i .

Efficient implementation of the Kruskal Algorithm

- Note that we do not give the run time of a single UNION operation but of a sequence of k consecutive such operations.

Such time complexity analysis is called *amortized analysis*; it (essentially) estimates average cost of an operation in a sequence of operations, in this case $\log k$.

- We can also implement the Union-Find data structure in $O(n \log n)$ time. This is done by using a disjoint-set data structure.
<https://eduassistpro.github.io/algorithm/union-find.html>
- The simplest implementation of the Union-Find data structure uses:
 - ① an array A such that $A[i] = j$ means that i is connected to j ;
 - ② an array B such that $B[i]$ contains the number of elements in the set labeled by i (which can be 0) and pointers to the first and last element of a linked list of elements of the set labeled by i .

Efficient implementation of the Kruskal Algorithm

- Note that we do not give the run time of a single UNION operation but of a sequence of k consecutive such operations.

Such time complexity analysis is called *amortized analysis*; it (essentially) estimates average cost of an operation in a sequence of operations, in this case $\log k$.

- We can also implement the Union-Find data structure using a disjoint-set forest. This is a collection of sets, each represented by a tree. The root of a tree is the representative element of the set. The sets are disjoint, i.e., they have no common elements.
<https://eduassistpro.github.io/algorithm/kruskal.html>
- The simplest implementation of the Union-Find data structure consists of:
 - ① an array A such that $A[i] = j$ means that i is connected to j ;
 - ② an array B such that $B[i]$ contains the number of elements in the set labeled by i (which can be 0) and pointers to the first and last element of a linked list of elements of the set labeled by i .

- UNION(i, j) of two sets labeled by i and j , respectively, is defined as follows:
 - if number of elements in the set labeled by i is larger or equal to the number of elements in the set labeled by j then labels in array A of array B is

<https://eduassistpro.github.io>

- then the labels of all elements in the set labeled by j and array A is updated accordingly.
- Note that this definition implies that if the set containing an element m , then the new set containing m will have at least twice the number of elements of the set which contained m before the UNION(i, j) operation.

- UNION(i, j) of two sets labeled by i and j , respectively, is defined as follows:
 - if number of elements in the set labeled by i is larger or equal to the number of elements in the set labeled by j then labels in array A of array B is

<https://eduassistpro.github.io>

- then the labels of all elements in the set labeled by j are replaced by the label of the set labeled by i , and array B is updated accordingly.
- Note that this definition implies that if the set containing an element m , then the new set containing m will have at least twice the number of elements of the set which contained m before the UNION(i, j) operation.

- UNION(i, j) of two sets labeled by i and j , respectively, is defined as follows:
 - if number of elements in the set labeled by i is larger or equal to the number of elements in the set labeled by j then labels in array A of array B is

<https://eduassistpro.github.io>

- then the labels of all elements in the set labeled by i are changed to the labels of all elements in the set labeled by j and array B is updated accordingly.
- Note that this definition implies that if the set containing an element m , then the new set containing m will have at least twice the number of elements of the set which contained m before the UNION(i, j) operation.

- Any sequence of k initial consecutive UNION operations can touch at most $2k$ elements of S (which happens if all UNION operations were applied to singleton sets).

Assignment Project Exam Help

- Thus, since each UNION operation touches at most $2k$ elements, we have at most $2k^2$ label changes in A .

ive UNION

- Since each UNION operation containing m at least doubles the size of the set containing that element, the label of the set containing m could change many times.

many

Add WeChat edu_assist_pro

- Thus, since we have at most $2k$ elements in S , $2k$ consecutive UNION operations will have in total fewer than $2k \log 2k$ many label changes in A and each UNION operation changes just a few pointers in B and adds up the sizes of sets.

- Any sequence of k initial consecutive UNION operations can touch at most $2k$ elements of S (which happens if all UNION operations were applied to singleton sets).

Assignment Project Exam Help

- Thus, any sequence of k consecutive UNION operations can touch at most $2k$ elements of S .

- Since each UNION operation containing m at least doubles the size of the set containing that element, the label of the set containing m could change many times.

Add WeChat edu_assist_pro

- Thus, since we have at most $2k$ elements in S , k consecutive UNION operations will have in total fewer than $2k \log 2k$ many label changes in A and each UNION operation changes just a few pointers in B and adds up the sizes of sets.

- Any sequence of k initial consecutive UNION operations can touch at most $2k$ elements of S (which happens if all UNION operations were applied to singleton sets).

Assignment Project Exam Help

- Thus, since we have at most $2k$ elements in S , any sequence of k consecutive UNION operations will have in total fewer than $2k \log 2k$ many label changes in A and each UNION operation changes just a few pointers in B and adds up the sizes of sets.
- Since containing m at least doubles the size of the set containing that element, the label of the set containing m could change many times.
- Thus, since we have at most $2k$ elements in S , any sequence of k consecutive UNION operations will have in total fewer than $2k \log 2k$ many label changes in A and each UNION operation changes just a few pointers in B and adds up the sizes of sets.

Add WeChat edu_assist_pro

- Any sequence of k initial consecutive UNION operations can touch at most $2k$ elements of S (which happens if all UNION operations were applied to singleton sets).

Assignment Project Exam Help

- Thus, since we have at most $2k$ elements, any UNION operation can touch at most $2k$ elements.
- Since each UNION operation containing m at least doubles the size of the set containing that element, the label of the set containing m could change many times.
- Thus, since we have at most $2k$ elements, any sequence of k initial consecutive UNION operations will have in total fewer than $2k \log 2k$ many label changes in A and each UNION operation changes just a few pointers in B and adds up the sizes of sets.

Add WeChat edu_assist_pro

1

Assignment Project Exam Help

- Thus, every sequence of k initial consecutive UNION operations has time complexity of $O(k \log k)$.

- Successive UNION operations are not necessarily independent. It is possible that one UNION operation depends on another.
- See the textbook for an Union-Find data structure that uses path compression, which further reduces the amortized cost of increasing the number of UNION operations from $O(n^2)$ to $O(n \log n)$.

Add WeChat edu_assist_pro

Assignment Project Exam Help

- Thus, every sequence of k initial consecutive UNION operations has time complexity of $O(k \log k)$.

- Suc poss

<https://eduassistpro.github.io>

- See the textbook for an Union-Find data structure

path compression, which further reduces the am
UNION operation (the cost of increasing the co

operations from $O(n^2)$ to $O(n \log n)$)

Add WeChat edu_assist_pro

Assignment Project Exam Help

- Thus, every sequence of k initial consecutive UNION operations has time complexity of $O(k \log k)$.

- Suc poss

<https://eduassistpro.github.io>

- See the textbook for an Union-Find data structure path compression, which further reduces the time complexity of UNION operation at the cost of increasing the time complexity of FIND operation from $O(1)$ to $O(\log n)$.

Add WeChat edu_assist_pro

Efficient implementation of the Kruskal Algorithm

- We now use the previously described Union-Find data structure to efficiently implement the Kruskal algorithm on a graph $G = (V, E)$ with n vertices and m edges.

Assignment Project Exam Help

- We first have to sort m edges of graph G which takes time $O(m \log m)$.

Since

$O($

<https://eduassistpro.github.io>

- As we are sorting the edges, we can make use of the Union-Find data structure to keep track of the connected components. We start by initializing each vertex as its own parent and having a rank of 0. Then, for each edge $e = (v, u)$, we perform two FIND operations: $\text{FIND}(u)$ and $\text{FIND}(v)$. If the two vertices belong to the same component, we skip the edge. Otherwise, we merge the two components by performing a UNION operation. This way, we ensure that we always pick the minimum weight edge that connects two different components.

Add WeChat [edu_assist_pro](https://eduassistpro.github.io)

- For each edge $e = (v, u)$ on the sorted list of edges we use two FIND operations, $\text{FIND}(u)$ and $\text{FIND}(v)$ to determine if vertices u and v belong to the same component.

- We now use the previously described Union-Find data structure to efficiently implement the Kruskal algorithm on a graph $G = (V, E)$ with n vertices and m edges.

Assignment Project Exam Help

- We first have to sort m edges of graph G which takes time $O(m \log m)$.

Since

$O($

<https://eduassistpro.github.io>

- As we are sorting the edges, we can make use of the Union-Find data structure to keep track of the connected components. We start by initializing each vertex as its own parent and having a rank of 0. Then, for each edge $e = (v, u)$, we perform two FIND operations: $\text{FIND}(u)$ and $\text{FIND}(v)$. If the two vertices belong to the same component, we skip the edge. Otherwise, we merge the two components by performing a UNION operation. This way, we ensure that we always pick the minimum weight edge that connects two different components.

Add WeChat [edu_assist_pro](https://eduassistpro.github.io)

- For each edge $e = (v, u)$ on the sorted list of edges we use two FIND operations, $\text{FIND}(u)$ and $\text{FIND}(v)$ to determine if vertices u and v belong to the same component.

- We now use the previously described Union-Find data structure to efficiently implement the Kruskal algorithm on a graph $G = (V, E)$ with n vertices and m edges.

Assignment Project Exam Help

- We first have to sort m edges of graph G which takes time $O(m \log m)$.

Since

$O($

<https://eduassistpro.github.io>

- As we sort the edges, we will be creating connected components which will be merged until all vertices belong to one component. For this purpose we use Union-Find to track the connected components constructed till now.

Add WeChat `edu_assist_pro`

- For each edge $e = (v, u)$ on the sorted list of edges we use two FIND operations, $\text{FIND}(u)$ and $\text{FIND}(v)$ to determine if vertices u and v belong to the same component.

- We now use the previously described Union-Find data structure to efficiently implement the Kruskal algorithm on a graph $G = (V, E)$ with n vertices and m edges.

Assignment Project Exam Help

- We first have to sort m edges of graph G which takes time $O(m \log m)$.

Since

$O($

<https://eduassistpro.github.io>

- As we sort the edges, we will be creating connected components which will be merged until all vertices belong to one component. For this purpose we use Union-Find to track the connected components constructed till now.

Add WeChat `edu_assist_pro`

- For each edge $e = (v, u)$ on the sorted list of edges we use two FIND operations, $\text{FIND}(u)$ and $\text{FIND}(v)$ to determine if vertices u and v belong to the same component.

Assignment Project Exam Help

If they do not belong to the same component (i.e., if $\text{PIND}(u) = i$ and $\text{PIND}(v) = j$, $j \neq i$), we add edge $e = (u, v)$ to the spanning tree being constructed and perform $\text{UNION}(i, j)$ to place u and v into the same component.

- In total, the cost of the algorithm is $O(m \log n)$.
- We also perform $n - 1$ UNION operations, which takes $O(n \log n)$.
- Initial sorting of edges takes $O(m \log m)$ time. Since each UNION operation takes $O(\log n)$ time, the total time complexity is $O(m \log m + n \log n)$.

Add WeChat `edu_assist_pro` for more updates.

Assignment Project Exam Help

If they do not belong to the same component (i.e., if $\text{PIND}(u) = i$ and $\text{PIND}(v) = j$, $j \neq i$), we add edge $e = (u, v)$ to the spanning tree being constructed and perform $\text{UNION}(i, j)$ to place u and v into the same component.

- In total, the cost is $O(m \log n)$.
- We also perform $n - 1$ UNION operations, which takes $O(n \log n)$.
- Initial sorting of edges takes $O(m \log m)$ time. Each UNION operation takes $O(\log n)$ time. Thus, the total time complexity is $O(m \log m + n \log n)$.

Add WeChat edu_assist_pro

Assignment Project Exam Help

If they do not belong to the same component (i.e., if $\text{PIND}(u) = i$ and $\text{PIND}(v) = j$, $j \neq i$), we add edge $e = (u, v)$ to the spanning tree being constructed and perform $\text{UNION}(i, j)$ to place u and v into the same component.

- In total, there are $m - n + 1$ edges added to the spanning tree, each with cost c_e . Thus, the total cost is $\sum c_e$.
- We also perform $n - 1$ UNION operations. Each UNION operation takes $O(\log n)$ time.
- Initial sorting of edges takes $O(m \log m)$ time. Each UNION operation takes $O(\log n)$ time. Thus, we obtain an overall time complexity of $O(m \log m + n \log n)$.

Add WeChat edu_assist_pro

Assignment Project Exam Help

If they do not belong to the same component (i.e., if $\text{PIND}(u) = i$ and $\text{PIND}(v) = j$, $j \neq i$), we add edge $e = (u, v)$ to the spanning tree being constructed and perform $\text{UNION}(i, j)$ to place u and v into the same component.

- In total, there are $n - 1$ edges added to the spanning tree, each with a cost of c_e . Thus, the total cost is $\sum c_e$.
- We also perform $n - 1$ UNION operations. The time complexity for each UNION operation is $O(\log n)$.
- Initial sorting of edges takes $O(m \log m)$ time. Each UNION operation takes $O(\log n)$ time. Thus, the total time complexity is $O(m \log m + n \log n)$.

k-clustering of maximum spacing

- **Instance:** A complete graph G with weighted edges representing distances between the two vertices.
- **Task:** Partition the vertices of G into k disjoint subsets so that the minimal distance between vertices in different sets is as large as possible.

<https://eduassistpro.github.io>

- **Solution:** Sort the edges in an increasing order and start performing the usual Kruskal's algorithm for building a minimal spanning tree. We will obtain k connected components rather than a single one.

Add WeChat edu_assist_pro

- **Proof of optimality:** Let d be the distance between two vertices belonging to the minimal spanning tree which was not added to our k connected components; it is clearly the minimal distance between two vertices belonging to two of our k connected components. Clearly, all the edges included in k many connected components produced by our algorithm are of length smaller or equal to d .

k-clustering of maximum spacing

- **Instance:** A complete graph G with weighted edges representing distances between the two vertices.
- **Task:** Partition the vertices of G into k disjoint subsets so that the minimal

dist
larg
far a

h are as

<https://eduassistpro.github.io>

- **Solution:** Sort the edges in an increasing order and start performing the usual Kruskal's algorithm for building a minimal spanning tree. We will obtain k connected components rather than a single

Add WeChat edu_assist_pro

- **Proof of optimality:** Let d be the distance of the minimal spanning tree which was not added to our k connected components; it is clearly the minimal distance between two vertices belonging to two of our k connected components. Clearly, all the edges included in k many connected components produced by our algorithm are of length smaller or equal to d .

k-clustering of maximum spacing

- **Instance:** A complete graph G with weighted edges representing distances between the two vertices.
- **Task:** Partition the vertices of G into k disjoint subsets so that the minimal

dist
larg
far a

h are as

<https://eduassistpro.github.io>

- **Solution:** Sort the edges in an increasing order and start performing the usual Kruskal's algorithm for building a minimal spanning tree. We will obtain k connected components rather than a single

[Add WeChat](#) [edu_assist_pro](#)

- **Proof of optimality:** Let d be the distance of the minimal spanning tree which was not added to our k connected components; it is clearly the minimal distance between two vertices belonging to two of our k connected components. Clearly, all the edges included in k many connected components produced by our algorithm are of length smaller or equal to d .

k-clustering of maximum spacing

- **Instance:** A complete graph G with weighted edges representing distances between the two vertices.
- **Task:** Partition the vertices of G into k disjoint subsets so that the minimal

dist
larg
far a

h are as

<https://eduassistpro.github.io>

- **Solution:** Sort the edges in an increasing order and start performing the usual Kruskal's algorithm for building a minimal spanning tree. We will obtain k connected components rather than a single

[Add WeChat](#) [edu_assist_pro](#)

- **Proof of optimality:** Let d be the distance of the minimal spanning tree which was not added to our k connected components; it is clearly the minimal distance between two vertices belonging to two of our k connected components. Clearly, all the edges included in k many connected components produced by our algorithm are of length smaller or equal to d .

The Greedy Method

k-clustering of maximum spacing

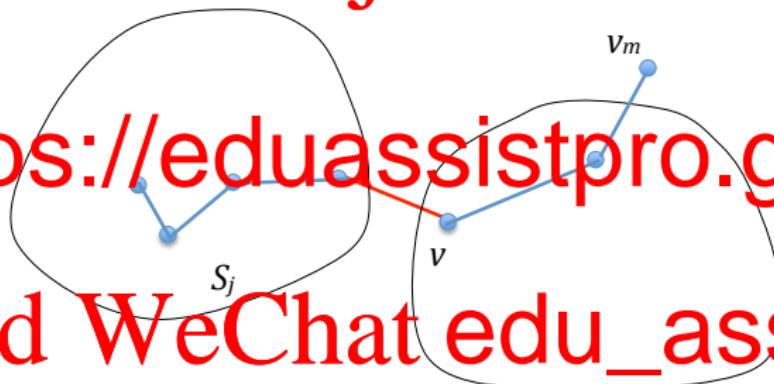
- Consider any partition \mathcal{S} into k subsets different from the one produced by our algorithm.

- This means that there is a connected component produced by our algorithm which contains vertices v_i and v_m . Since $v_i \in S_j$ for some $j \in \mathcal{S}$ and $v_m \notin S_j$.

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat `edu_assist_pro`



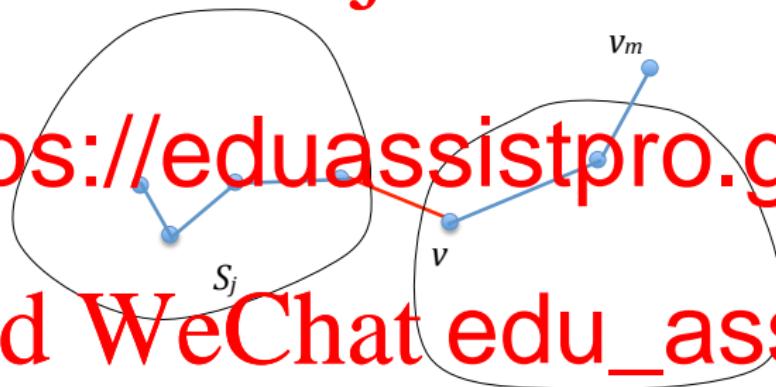
- Since v_i and v_m belong to the same connected component, there is a path in that component connecting v_i and v_m .
- Let v_p and v_{p+1} be two consecutive vertices on that path such that v_p belongs to S_j and $v_{p+1} \notin S_j$.
- Thus, $v_{p+1} \in S_q$ for some $q \neq j$.

The Greedy Method

k-clustering of maximum spacing

- Consider any partition \mathcal{S} into k subsets different from the one produced by our algorithm.
- This means that there is a connected component produced by our algorithm which contains vertices v_i and v_m such that $v_i \in S_j$ for some $S_j \in \mathcal{S}$ and $v_m \notin S_j$.

Assignment Project Exam Help



<https://eduassistpro.github.io>
Add WeChat edu_assist_pro

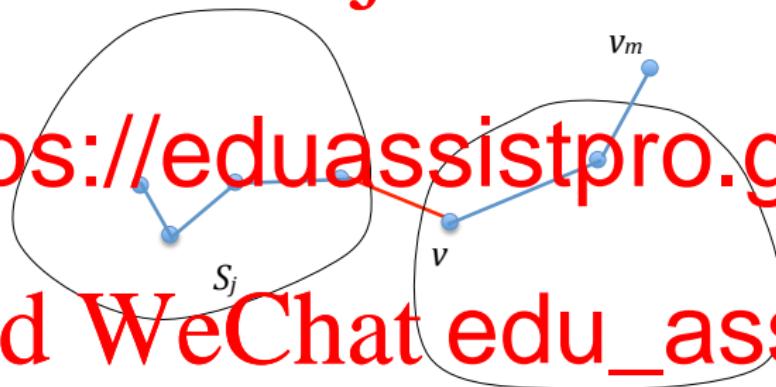
- Since v_i and v_m belong to the same connected component, there is a path in that component connecting v_i and v_m .
- Let v_p and v_{p+1} be two consecutive vertices on that path such that v_p belongs to S_j and $v_{p+1} \notin S_j$.
- Thus, $v_{p+1} \in S_q$ for some $q \neq j$.

The Greedy Method

k-clustering of maximum spacing

- Consider any partition \mathcal{S} into k subsets different from the one produced by our algorithm.
- This means that there is a connected component produced by our algorithm which contains vertices v_i and v_m such that $v_i \in S_j$ for some $S_j \in \mathcal{S}$ and $v_m \notin S_j$.

Assignment Project Exam Help



<https://eduassistpro.github.io>
Add WeChat edu_assist_pro

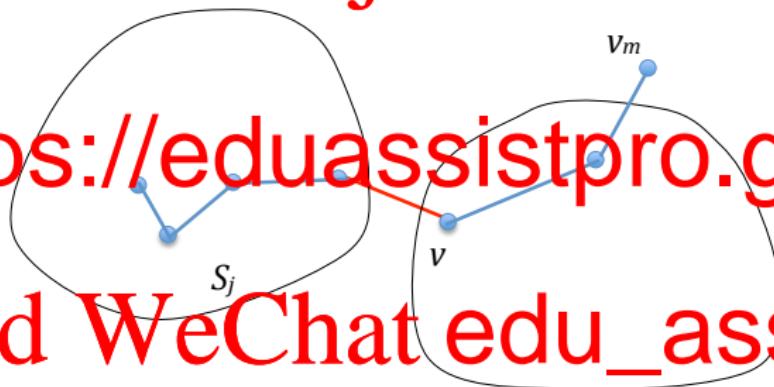
- Since v_i and v_m belong to the same connected component, there is a path in that component connecting v_i and v_m .
- Let v_p and v_{p+1} be two consecutive vertices on that path such that v_p belongs to S_j and $v_{p+1} \notin S_j$.
- Thus, $v_{p+1} \in S_q$ for some $q \neq j$.

The Greedy Method

k-clustering of maximum spacing

- Consider any partition \mathcal{S} into k subsets different from the one produced by our algorithm.
- This means that there is a connected component produced by our algorithm which contains vertices v_i and v_m such that $v_i \in S_j$ for some $S_j \in \mathcal{S}$ and $v_m \notin S_j$.

Assignment Project Exam Help



<https://eduassistpro.github.io>
Add WeChat edu_assist_pro

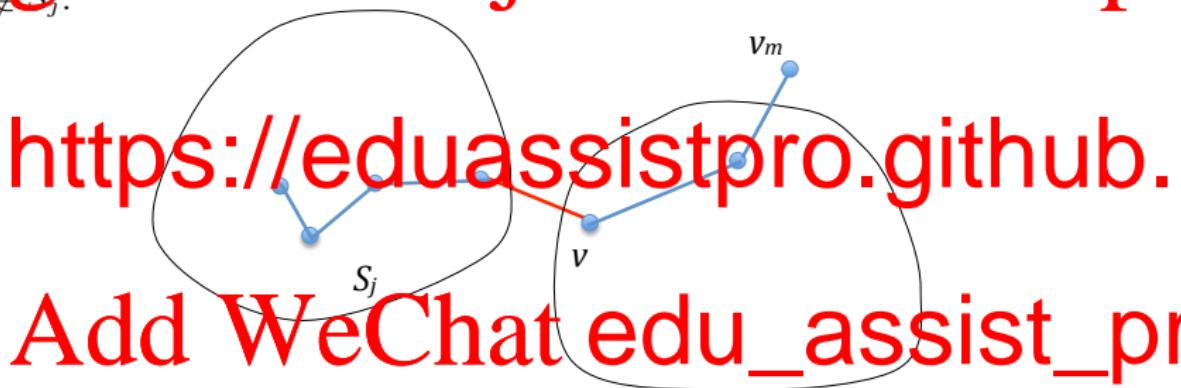
- Since v_i and v_m belong to the same connected component, there is a path in that component connecting v_i and v_m .
- Let v_p and v_{p+1} be two consecutive vertices on that path such that v_p belongs to S_j and $v_{p+1} \notin S_j$.
- Thus, $v_{p+1} \in S_q$ for some $q \neq j$.

The Greedy Method

k-clustering of maximum spacing

- Consider any partition \mathcal{S} into k subsets different from the one produced by our algorithm.
- This means that there is a connected component produced by our algorithm which contains vertices v_i and v_m such that $v_i \in S_j$ for some $S_j \in \mathcal{S}$ and $v_m \notin S_j$.

Assignment Project Exam Help

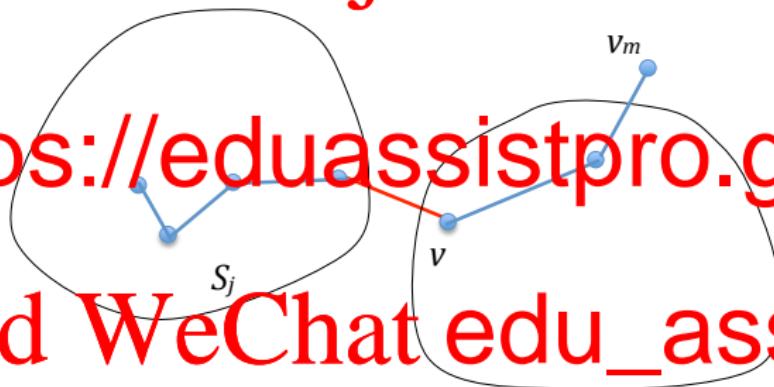


- Since v_i and v_m belong to the same connected component, there is a path in that component connecting v_i and v_m .
- Let v_p and v_{p+1} be two consecutive vertices on that path such that v_p belongs to S_j and $v_{p+1} \notin S_j$.
- Thus, $v_{p+1} \in S_q$ for some $q \neq j$.

The Greedy Method

k-clustering of maximum spacing

- Consider any partition \mathcal{S} into k subsets different from the one produced by our algorithm.
- This means that there is a connected component produced by our algorithm which contains vertices v_i and v_m such that $v_i \in S_j$ for some $S_j \in \mathcal{S}$ and $v_m \notin S_j$.



<https://eduassistpro.github.io>
Add WeChat edu_assist_pro

- Since v_i and v_m belong to the same connected component, there is a path in that component connecting v_i and v_m .
- Let v_p and v_{p+1} be two consecutive vertices on that path such that v_p belongs to S_j and $v_{p+1} \notin S_j$.
- Thus, $v_{p+1} \in S_q$ for some $q \neq j$.

The Greedy Method

- Note that $d(v_p, v_{p+1}) \leq d$ which implies that the distance between these two clusters $S_j, S_q \in \mathcal{S}$ is smaller or equal to the minimal distance d between the k connected components produced by our algorithm.

Assignment Project Exam Help

- Thus, such a partition cannot be a more optimal clustering than the one produced by our algorithm.

• Where <https://eduassistpro.github.io>

- We have $O(n^2)$ edges; thus sorting them by weight

$$O(n^2 \log n^2) = O(n^2 \log n)$$

Add WeChat edu_assist_pro

- While running the (partial) Kruskal algorithm we use a data structure which requires $O(n^2 \log n)$ steps.

-FIND

- So the grand total for the whole algorithm is $O(n^2 \log n)$ many steps.

The Greedy Method

- Note that $d(v_p, v_{p+1}) \leq d$ which implies that the distance between these two clusters $S_j, S_q \in \mathcal{S}$ is smaller or equal to the minimal distance d between the k connected components produced by our algorithm.

Assignment Project Exam Help

- Thus, such a partition cannot be a more optimal clustering than the one produced by our algorithm.

• Why? <https://eduassistpro.github.io>

- We have $O(n^2)$ edges; thus sorting them by weight takes $O(n^2 \log n^2) = O(n^2 \log n)$.

Add WeChat edu_assist_pro

- While running the (partial) Kruskal algorithm we need to maintain a data structure which requires $O(n^2 \log n)$ steps.

-FIND

- So the grand total for the whole algorithm is $O(n^2 \log n)$ many steps.

The Greedy Method

- Note that $d(v_p, v_{p+1}) \leq d$ which implies that the distance between these two clusters $S_j, S_q \in \mathcal{S}$ is smaller or equal to the minimal distance d between the k connected components produced by our algorithm.

Assignment Project Exam Help

- Thus, such a partition cannot be a more optimal clustering than the one proposed.
- When <https://eduassistpro.github.io>
- We have $O(n^2)$ edges; thus sorting them by weight takes $O(n^2 \log n^2) = O(n^2 \log n)$.
- While running the (partial) Kruskal algorithm we use a data structure which requires $O(n^2 \log n)$ steps.
- So the grand total for the whole algorithm is $O(n^2 \log n)$ many steps.

The Greedy Method

- Note that $d(v_p, v_{p+1}) \leq d$ which implies that the distance between these two clusters $S_j, S_q \in \mathcal{S}$ is smaller or equal to the minimal distance d between the k connected components produced by our algorithm.

Assignment Project Exam Help

- Thus, such a partition cannot be a more optimal clustering than the one proposed.
- When <https://eduassistpro.github.io>
- We have $O(n^2)$ edges; thus sorting them by weight takes $O(n^2 \log n^2) = O(n^2 \log n)$.
- While running the (partial) Kruskal algorithm we need to maintain a data structure which requires $O(n^2 \log n)$ steps.
- So the grand total for the whole algorithm is $O(n^2 \log n)$ many steps.

Add WeChat edu_assist_pro

-FIND

The Greedy Method

- Note that $d(v_p, v_{p+1}) \leq d$ which implies that the distance between these two clusters $S_j, S_q \in \mathcal{S}$ is smaller or equal to the minimal distance d between the k connected components produced by our algorithm.

Assignment Project Exam Help

- Thus, such a partition cannot be a more optimal clustering than the one produced.
- When <https://eduassistpro.github.io>
- We have $O(n^2)$ edges; thus sorting them by weight takes $O(n^2 \log n^2) = O(n^2 \log n)$.
- While running the (partial) Kruskal algorithm, we need to maintain a data structure which requires $O(n^2 \log n)$ steps.
- So the grand total for the whole algorithm is $O(n^2 \log n)$ many steps.

Add WeChat edu_assist_pro

-FIND

The Greedy Method

- Note that $d(v_p, v_{p+1}) \leq d$ which implies that the distance between these two clusters $S_j, S_q \in \mathcal{S}$ is smaller or equal to the minimal distance d between the k connected components produced by our algorithm.

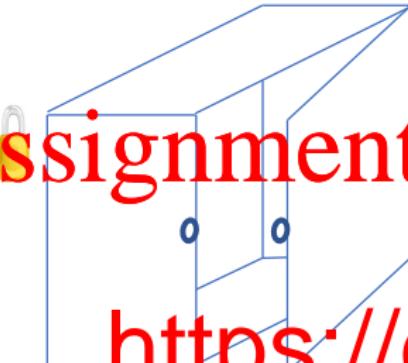
Assignment Project Exam Help

- Thus, such a partition cannot be a more optimal clustering than the one produced by our algorithm.
- When we implement this algorithm, we will have to sort all edges. We have $O(n^2)$ edges; thus sorting them by weight will take $O(n^2 \log n^2) = O(n^2 \log n)$ time.
- While running the (partial) Kruskal algorithm, we will need to maintain a data structure which requires $O(n^2 \log n)$ steps. This data structure is called a Union-Find data structure.
- So the grand total for the whole algorithm is $O(n^2 \log n)$ many steps.

Add WeChat edu_assist_pro

-FIND

PUZZLE!!



The Elbonian postal service mandates that boxes to be sent, if not empty, must be locked, but they do not allow keys to be sent. The key must remain with the sender. You can send padlocks

<https://eduassistpro.github.io>

Bob is visiting Elbonia and well as that both Bob and Alice have wishes to send his teddybear to padlocks and b

Alice who is staying at a different communic hotel. Both Bob and Alice have on the strateg boxes like the one shown on the solutions; on

picture as well as padlocks which can be used to lock the boxes. However, there is a problem.

solution, the other can be called the “OR” solution. The “AND” solution requires 4 mail one way services while the “OR” solution requires only 2.