

# Assignment Project Exam Help

Algorithms: https://powscoler.com

### Add Wechat powcoder

School of Computer Science and Engineering University of New South Wales

DYNAMIC PROGRAMMING



#### Dynamic Programming

# As siemining pynamic Programming: Xii and ptimae p solution to the problem from optimal solutions for (carefully chosen) smaller size subproblems.

- Subjutting and the state of t
- Efficiency of DP comes from the fact that the sets of subproblems needed to so Add problems the Chat powpcoder only once Add utility e Chat powpcodering larger problems.

#### Dynamic Programming

# A Stie Chiline Dynamic Price Interior Solution to the problem from optimal solutions for (carefully chosen) smaller size subproblems.

- Subjections are chosen in a way which allows recursive construction of optimal solutions to such a by the state of the solutions to smaller size subproblems.
- Efficiency of DP cames from the fact that the sets of subproblems needed to so Add problems from the fact that the sets of subproblems needed once Add problems from the fact that the sets of subproblems needed to so Add problems.

4□ > 4□ > 4□ > 4□ > 4□ > 3

#### Dynamic Programming

# A Spic Children by mark Programming Nia Inqui na elp solution to the problem from optimal solutions for (carefully chosen) smaller size subproblems.

- Subproblems are chosen in a way which allows recursive construction of optimal solutions to such submillens to toptimal solutions to smaller size subproblems.
- Efficiency of DP comes from the fact that the sets of subproblems needed to sold larger problems havil propagate subproblems solved only once and its solution is stored in a table for number use of solving larger problems.

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

## Assignment Project Exam Help

of finding a subset with the **largest possible number** of compatible activities, but the Greedy Method **does not** work for the present problem.

- We shttps://powcoder.com
  non-decreas g sequence will assume that  $f_1 \leq f_2 \leq \ldots \leq f_n$ .
- For every  $i \leq n$  we solve the following subproblems:

- $\bullet$   $\sigma_i$  consists of non-overlapping activities;
- $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
- ③  $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

## Assignment Project Exam Help

of finding a subset with the **largest possible number** of compatible activities, but the Greedy Method **does not** work for the present problem.

- We shttps://powcoder.com
- For every  $i \leq n$  we solve the following subproblems:

- ①  $\sigma_i$  consists of non-overlapping activities;
- $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
- 3  $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

## A S Task: Find a subset of compatible activities of maximal total direction Project Exam Help Remember, we used the Greedy Method to solve a somewhat similar problem

Remember, we used the Greedy Method to solve a somewhat similar problem
of finding a subset with the largest possible number of compatible
activities, but the Greedy Method does not work for the present problem.

## • We shttps://powcoder.com

• For every  $i \leq n$  we solve the following subproblems:

- ①  $\sigma_i$  consists of non-overlapping activities;
- $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
- ③  $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

## A S Task: Find a subset of compatible activities of maximal total direction Project Exam Help Remember, we used the Greedy Method to solve a somewhat similar problem

- Remember, we used the Greedy Method to solve a somewhat similar problem
  of finding a subset with the largest possible number of compatible
  activities, but the Greedy Method does not work for the present problem.
- We satisfied by the point of the property of the point of the property of t
- For every  $i \leq n$  we solve the following subproblems:

- ①  $\sigma_i$  consists of non-overlapping activities;
- $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
- 3  $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

# A S Task: Find a subset of compatible activities of maximal total deration Help Remember, we used the Greedy Method to solve a somewhat similar problem

- Remember, we used the Greedy Method to solve a somewhat similar problem
  of finding a subset with the largest possible number of compatible
  activities, but the Greedy Method does not work for the present problem.
- We satisfied by the point of the property of the point of the property of t
- For every  $i \le n$  we solve the following subproblems: **Subproblem** i(i) and a subsequence  $\sigma_i$  of the sequence of adjuities  $S_i = a_1, a_2, \ldots, a_i$  such that i(i) and i(i) ano
  - $\bullet$   $\sigma_i$  consists of non-overlapping activities;
  - $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
  - 3  $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

# A S Task: Find a subset of compatible activities of maximal total deration Help Remember, we used the Greedy Method to solve a somewhat similar problem

- Remember, we used the Greedy Method to solve a somewhat similar problem
  of finding a subset with the largest possible number of compatible
  activities, but the Greedy Method does not work for the present problem.
- We satisfy a Sig the Otti iver the Civing time it in non-decreasing sequence so will assume that  $f_1 \leq f_2 \leq \ldots \leq f_n$ .
- For every  $i \le n$  we solve the following subproblems: Subproblem P(i) and a subsequence  $\sigma_i$  of the sequence of adjuities  $S_i = (a_1, a_2, \dots, a_i)$  such that
  - $\sigma_i$  consists of non-overlapping activities;
  - $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
  - ③  $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

# A S Task: Find a subset of compatible activities of maximal total deration Project Exam Help Remember, we used the Greedy Method to solve a somewhat similar problem

- Remember, we used the Greedy Method to solve a somewhat similar problem of finding a subset with the largest possible number of compatible activities, but the Greedy Method does not work for the present problem.
- We sattly a Sig the Stip W which the stip time it is non-decreasing sequence, so will assume that  $f_1 \leq f_2 \leq \ldots \leq f_n$ .
- For every  $i \le n$  we solve the following subproblems: • Subproblem P(i) and a subsequence  $\sigma_i$  of the sequence of adjuities  $S_i = (a_1, a_2, \dots, a_i)$  such that I and I are the sequence of adjuities I and I are the sequence of I and I are the sequence I are the sequence I and I are the sequence I are the sequence I and I are the sequence I are the sequence I and I are the sequence I are the sequence I and I are the sequence I are the sequence I and I are the sequen
  - $\bullet$   $\sigma_i$  consists of non-overlapping activities;
  - $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
  - 3  $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

## A S Task: Find a subset of compatible activities of maximal total direction problem. Remember, we used the Greedy Method to solve a somewhat similar problem.

- Remember, we used the Greedy Method to solve a somewhat similar problem of finding a subset with the largest possible number of compatible activities, but the Greedy Method does not work for the present problem.
- We satisfy a Sig the Otti iver the Civing time it in non-decreasing sequence so will assume that  $f_1 \leq f_2 \leq \ldots \leq f_n$ .
- For every  $i \le n$  we solve the following subproblems: Subproblem P(i) and a subsequence  $\sigma_i$  of the sequence of adjuities  $S_i = (a_1, a_2, \dots, a_i)$  such that I and I are I and I and I are I and I are I are I and I are I are I and I are I and I are I and I are I are I are I and I are I are I and I are I and I are I are I and I are I and I are I are I are I and I are I are I are I and I are I are I and I are I are I and I are I are I are I are I are I and I are I are I are I and I are I are I are I and I are I and I are I are I are I and I are I are I are I are I and I are I are I are I are I are I and I are I and I are I are I are I are I are I and I are I are I and I are I are I are I are I are I and I are I are I and I are I are I and
  - $\bullet$   $\sigma_i$  consists of non-overlapping activities;
  - $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
  - $\bullet$   $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

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

### ASSIGNMENT Project Exam Help

- Remember, we used the Greedy Method to solve a somewhat similar problem of finding a subset with the largest possible number of compatible activities, but the Greedy Method does not work for the present problem.
- We sattly a fig the other will assume that  $f_1 \leq f_2 \leq \ldots \leq f_n$ .
- For every  $i \le n$  we solve the following subproblems: • Subproblem f(i) and a subsequence  $\sigma_i$  of the sequence of adjuities  $S_i = (a_1, a_2, \dots, a_n)$  such  $G_{i}$  and  $G_{i}$  are  $G_{i}$  and  $G_{i}$  and  $G_{i}$  are  $G_{i}$  and  $G_{i}$  and  $G_{i}$  are  $G_{i}$  are  $G_{i}$  and  $G_{i}$  are  $G_{i}$  are  $G_{i}$  and  $G_{i}$  are  $G_{i}$  and  $G_{i}$  are  $G_{i}$  are  $G_{i}$  are  $G_{i}$  are  $G_{i}$  and  $G_{i}$  are  $G_{i}$  are  $G_{i}$  are  $G_{i}$  and  $G_{i}$  are  $G_{i}$  are
  - $\bullet$   $\sigma_i$  consists of non-overlapping activities;
  - $\circ$   $\sigma_i$  ends with activity  $a_i$ ;
  - $\bullet$   $\sigma_i$  is of maximal total duration among all subsequences of  $S_i$  which satisfy 1 and 2.
- Note: the role of Condition 2 is to simplify recursion.

- Let T(i) be the total duration of the optimal solution S(i) of the subproblem P(i).
- Assignment Project Exam Help

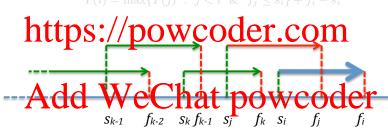
$$\begin{array}{c} \textbf{https://powcoder.com} \\ \textbf{Add WeChat powcoder} \\ s_{k-1} \ f_{k-2} \ s_k \ f_{k-1} \ s_j \ f_k \ s_i \ f_j \ f_i \end{array}$$

• In the table, for every i, besides T(i), we also store  $\pi(i) = j$  for which the above max is achieved:

$$\pi(i) = \arg\max\{T(j) : j < i \& f_j \le s_i\}$$

- Let T(i) be the total duration of the optimal solution S(i) of the subproblem P(i).

# • For S(1) we choose $a_1$ ; thus $T(1) = f_1 - s_1$ ; a substituting the substitution of the substitutio



• In the table, for every i, besides T(i), we also store  $\pi(i) = i$  for which the

$$\pi(i) = \arg\max\{T(j) : j < i \& f_j \le s_i\}$$

- Let T(i) be the total duration of the optimal solution S(i) of the subproblem P(i).
- For S(1) we choose  $a_1$ ; thus  $T(1) = f_1 s_1$ ;

  Security 13 supergraph that properties of the properties of the

$$T(i) = \max\{T(j) : j < i \& f_j \le s_i\} + f_i - s_i$$

$$https://powcoder.com$$

$$Add WeChat powcoder$$

$$s_{k-1} f_{k-2} s_k f_{k-1} s_j f_k s_i f_j f_i$$

• In the table, for every i, besides T(i), we also store  $\pi(i) = j$  for which the above max is achieved:

$$\pi(i) = \arg\max\{T(j) : j < i \& f_j \le s_i\}$$

- Let T(i) be the total duration of the optimal solution S(i) of the subproblem P(i).
- For S(1) we choose  $a_1$ ; thus  $T(1) = f_1 s_1$ ;

  Securing that the properties of the propertie

$$T(i) = \max\{T(j) : j < i \& f_j \le s_i\} + f_i - s_i$$

$$https://powcoder.com$$

$$Add WeChat powcoder$$

$$s_{k-1} f_{k-2} s_k f_{k-1} s_j f_k s_i f_j f_i$$

• In the table, for every i, besides T(i), we also store  $\pi(i) = j$  for which the above max is achieved:

$$\pi(i) = \arg \max \{ T(j) : j < i \& f_j \le s_i \}$$

ullet Why does such a recursion produce optimal solutions to subproblems P(i)?

- We claim: the truncated subsequence  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  is an optimal solution to subproblem  $P(k_{m-1})$ , where  $k_{m-1} < i$ .
- Why https://powcoder.comsed to prove the optimal prof the grady solutions!
- If there were a sequence  $S^*$  of a larger total duration than the duration of sequence  $\widehat{S}$  decreased at the content of the sequence  $\widehat{S}$  with a longer total duration than the total duration of sequence S, contradicting the optimality of S.
- Thus, the optimal solution  $S = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}}, a_{k_m})$  for problem P(i)  $(= P(a_{k_m}))$  is obtained from the optimal solution  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  for problem  $P(a_{k_{m-1}})$  by extending it with  $a_{k_m}$

• Why does such a recursion produce optimal solutions to subproblems P(i)?

- We claim: the truncated subsequence  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  is an optimal solution to subproblem  $P(k_{m-1})$ , where  $k_{m-1} < i$ .
- Why https://powcoder.comsed to prove the optimal prof the graphy solutions!
- If there were a sequence  $S^*$  of a larger total duration than the duration of seque  $\widehat{A}$  decided at which at powcoder for subproblem P(i) with a longer total duration than the total duration of sequence S, contradicting the optimality of S.
- Thus, the optimal solution  $S = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}}, a_{k_m})$  for problem P(i)  $(= P(a_{k_m}))$  is obtained from the optimal solution  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  for problem  $P(a_{k_{m-1}})$  by extending it with  $a_{k_m}$

• Why does such a recursion produce optimal solutions to subproblems P(i)?

- We claim: the truncated subsequence  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  is an optimal solution to subproblem  $P(k_{m-1})$ , where  $k_{m-1} < i$ .
- Why https://powcoder.comsed to prove the optimal prof the graphy solutions!
- If there were a sequence  $S^*$  of a larger total duration than the duration of seque  $\widehat{Add}$  at  $\widehat{Add}$  we chart powcoder for subproblem P(i) with a longer total duration than the total duration of sequence S, contradicting the optimality of S.
- Thus, the optimal solution  $S = (a_{k_1}, a_{k_2}, \ldots, a_{k_{m-1}}, a_{k_m})$  for problem P(i)  $(= P(a_{k_m}))$  is obtained from the optimal solution  $S' = (a_{k_1}, a_{k_2}, \ldots, a_{k_{m-1}})$  for problem  $P(a_{k_{m-1}})$  by extending it with  $a_{k_m}$

ullet Why does such a recursion produce optimal solutions to subproblems P(i)?

- We claim: the truncated subsequence  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  is an optimal solution to subproblem  $P(k_{m-1})$ , where  $k_{m-1} < i$ .
- whyhttps://powcoder.comsed to prove the optimal p of the grady solutions!
- If there were a sequence  $S^*$  of a larger total duration than the duration of seque  $\widehat{Add}$  at  $\widehat{Add}$  see the  $\widehat{Add}$  sequence  $\widehat{S}$  subproblem P(i) with a longer total duration than the total duration of sequence S, contradicting the optimality of S.
- Thus, the optimal solution  $S = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}}, a_{k_m})$  for problem P(i)  $(= P(a_{k_m}))$  is obtained from the optimal solution  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  for problem  $P(a_{k_{m-1}})$  by extending it with  $a_{k_m}$

• Why does such a recursion produce optimal solutions to subproblems P(i)?

- We claim: the truncated subsequence  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  is an optimal solution to subproblem  $P(k_{m-1})$ , where  $k_{m-1} < i$ .
- Why he apply he sime to W Gate a surent wild W sed to prove the optimality of the greedy solutions!
- If there were a sequence  $S^*$  of a larger total duration than the duration of seque  $\widehat{A}$  decided at which at powcoder for subproblem P(i) with a longer total duration than the total duration of sequence S, contradicting the optimality of S.
- Thus, the optimal solution  $S=(a_{k_1},a_{k_2},\ldots,a_{k_{m-1}},a_{k_m})$  for problem P(i)  $(=P(a_{k_m}))$  is obtained from the optimal solution  $S'=(a_{k_1},a_{k_2},\ldots,a_{k_{m-1}})$  for problem  $P(a_{k_{m-1}})$  by extending it with  $a_{k_m}$

ullet Why does such a recursion produce optimal solutions to subproblems P(i)?

- We claim: the truncated subsequence  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  is an optimal solution to subproblem  $P(k_{m-1})$ , where  $k_{m-1} < i$ .
- Why he ap She's me cu W Gate a surent wild will sed to prove the optimality of the greedy solutions!
- If there were a sequence  $S^*$  of a larger total duration than the duration of sequence S' are also unding with activity  $a_{i,m}$  and obtain a colorion for subproblem P(i) with a longer total duration than the total duration of sequence S, contradicting the optimality of S.
- Thus, the optimal solution  $S = (a_{k_1}, a_{k_2}, \ldots, a_{k_{m-1}}, a_{k_m})$  for problem P(i)  $(= P(a_{k_m}))$  is obtained from the optimal solution  $S' = (a_{k_1}, a_{k_2}, \ldots, a_{k_{m-1}})$  for problem  $P(a_{k_{m-1}})$  by extending it with  $a_{k_m}$

ullet Why does such a recursion produce optimal solutions to subproblems P(i)?

- We claim: the truncated subsequence  $S' = (a_{k_1}, a_{k_2}, \dots, a_{k_{m-1}})$  is an optimal solution to subproblem  $P(k_{m-1})$ , where  $k_{m-1} < i$ .
- Why he apply he sime to W Gate a surent wild W sed to prove the optimality of the greedy solutions!
- If there were a sequence  $S^*$  of a larger total duration than the duration of sequence S' are also undirectivity  $a_{i,m}$  and obtain a colorion for subproblem P(i) with a longer total duration than the total duration of sequence S, contradicting the optimality of S.
- Thus, the optimal solution  $S=(a_{k_1},a_{k_2},\ldots,a_{k_{m-1}},a_{k_m})$  for problem P(i)  $(=P(a_{k_m}))$  is obtained from the optimal solution  $S'=(a_{k_1},a_{k_2},\ldots,a_{k_{m-1}})$  for problem  $P(a_{k_{m-1}})$  by extending it with  $a_{k_m}$

• Continuing with the solution of the problem, we now let

```
T_{max} = \max\{T(i) : i \leq n\};
```

# Assignment $\Pr_{\text{roject}}^{\text{last}} = \underset{\text{last}}{\text{arg max}} \{T(i) : i \leq n\}.$

### • Thushttps://powcoder.com(π(last)),....

- Why is such solution optimal, i.e., why looking for optimal solutions of P(i)
- Add WeChat powcoder
- Time complexity: having sorted the activities by their finishing times in time their optimal solutions (to be looked up in a table). Thus:  $T(\bar{n}) = \mathbb{P}(n^2)$

• Continuing with the solution of the problem, we now let

$$T_{max} = \max\{T(i) : i \le n\};$$

 $last = arg \max\{T(i) : i \le n\}.$ 

Second which should be plined enough which says for problem from the table of partial solutions, because in the  $i^{th}$  slot of the table, besides T(i), we also store  $\pi(i) = j$ , (j < i) such that the optimal solution of P(i) extends the optimal solution of subproblem P(j).

### • Thushttps://powcoder.com(\pi(last)),....

- Why is such solution optimal, i.e., why looking for optimal solutions of P(i) which must end with  $a_i$  did not cause us to miss the optimal solution without such a additional variation.
- Such a Add we Chat powcoder

  Consider the optimal solution without such additional requirement, and assume it ends with activity  $a_k$ ; then it would have been obtained as the optimal solution of problem P(k).
- Time complexity: having sorted the activities by their finishing times in time  $O(n \log n)$ , we need to solve n subproblems P(i) for solutions ending in  $a_i$ ; for each such interval  $a_i$  we have to find all preceding compatible intervals and their optimal solutions (to be looked up in a table). That,  $T(a) = O(n^2)$

• Continuing with the solution of the problem, we now let

$$T_{max} = \max\{T(i) : i \le n\};$$

 $last = arg max\{T(i) : i \le n\}.$ 

Special lower operation the optimal sequence when the table of partial solutions, because in the  $i^{th}$  slot of the table, besides T(i), we also store  $\pi(i) = j$ , (j < i) such that the optimal solution of P(i) extends the optimal solution of subproblem P(j).

- Thus that temps in the converge is in the converge in the c
- Why is such solution optimal, i.e., why looking for optimal solutions of P(i) which must end with  $a_i$  did not cause us to miss the optimal solution without such an additional very energy  $\mathbf{x}^{*}$
- Such a Add tight we chat powcoder

  Consider the optimal solution without such additional requirement, and assume it ends with activity  $a_k$ ; then it would have been obtained as the optimal solution of problem P(k).
- Time complexity: having sorted the activities by their finishing times in time  $O(n \log n)$ , we need to solve n subproblems P(i) for solutions ending in  $a_i$ ; for each such interval  $a_i$  we have to find all preceding compatible intervals and their optimal solutions (to be looked up in a table). Thus,  $T(\mathbb{R}) = \mathbb{E}(n^2)$

• Continuing with the solution of the problem, we now let

 $T_{max} = \max\{T(i) : i \le n\};$ 

 $last = arg \max\{T(i) : i \le n\}.$ 

Second with the optimal enumed which salve for problem from the table of partial solutions, because in the  $i^{th}$  slot of the table, besides T(i), we also store  $\pi(i) = j$ , (j < i) such that the optimal solution of P(i) extends the optimal solution of subproblem P(j).

- Thus the tense in the converge is the first tense of the tense of t
- Why is such solution optimal, i.e., why looking for optimal solutions of P(i) which must end with  $a_i$  did not cause us to miss the optimal solution without such an additional requirement. The power of the p
- Consider the optimal solution without such additional requirement, and assume it ends with activity  $a_k$ ; then it would have been obtained as the optimal solution of problem P(k).
- Time complexity: having sorted the activities by their finishing times in time  $O(n \log n)$ , we need to solve n subproblems P(i) for solutions ending in  $a_i$ ; for each such interval  $a_i$  we have to find all preceding compatible intervals and their optimal solutions (to be looked up in a table) That  $T(h) = O(n^2)$

• Continuing with the solution of the problem, we now let

$$T_{max} = \max\{T(i) : i \le n\};$$

 $last = arg \max\{T(i) : i \le n\}.$ 

See cut low represents the optimal entency which salve for partial solutions, because in the  $i^{th}$  slot of the table, besides T(i) we also store  $\pi(i) = j$ , (j < i) such that the optimal solution of P(i) extends the optimal solution of subproblem P(j).

- Thus the tense in the converge is the first late ( )  $(\pi(\text{last})), \dots$
- Why is such solution optimal, i.e., why looking for optimal solutions of P(i) which must end with a<sub>i</sub> did not cause us to miss the optimal solution without such an additional requirement?
  Consider the optimal solution without such additional requirement, and
- Consider the optimal solution without such additional requirement, and assume it ends with activity  $a_k$ ; then it would have been obtained as the optimal solution of problem P(k).
- Time complexity: having sorted the activities by their finishing times in time  $O(n \log n)$ , we need to solve n subproblems P(i) for solutions ending in  $a_i$ ; for each such interval  $a_i$  we have to find all preceding compatible intervals and their optimal solutions (to be looked up in a table). Thus,  $P(\mathbb{R}) = \mathbb{R}(n^2)$

• Continuing with the solution of the problem, we now let

 $T_{max} = \max\{T(i) : i \le n\};$ 

 $\operatorname{last} = \operatorname{arg\,max}\{T(i) \ : \ i \leq n\}.$ 

A Special particle of the primal enumed which salva for problem from the table of partial solutions, because in the  $i^{th}$  slot of the table, besides T(i), we also store  $\pi(i) = j$ , (j < i) such that the optimal solution of P(i) extends the optimal solution of subproblem P(j).

- Thus the tense in the converge is the first tense of the tense of t
- Why is such solution optimal, i.e., why looking for optimal solutions of P(i) which must end with a<sub>i</sub> did not cause us to miss the optimal solution without such an additional very remember 100 Consider the optimal solution without such additional requirement, and
- Consider the optimal solution without such additional requirement, and assume it ends with activity  $a_k$ ; then it would have been obtained as the optimal solution of problem P(k).
- Time complexity: having sorted the activities by their finishing times in time  $O(n \log n)$ , we need to solve n subproblems P(i) for solutions ending in  $a_i$ ; for each such interval  $a_i$  we have to find all preceding compatible intervals and their optimal solutions (to be looked up in a table). Thus,  $T(n) = O(n^2)$

• Longest Increasing Subsequence: Given a sequence of n real numbers A[1..n], determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence are strictly

- Solution: For each  $i \leq n$  we solve the following subproblems:
- Sub roblem P(i): Find a subsequence of the sequence A[1..i] of max https://powcoder.comand which ends with P(i): Find a subsequence of the sequence A[1..i] of max https://powcoder.com
- Recursion: Assume we have solved the subproblems for all j < i, and that Add we Chatipowcoder
- We now look for all A[m] such that m < i and such that A[m] < A[i].
- Among those we pick m which produced the longest increasing subsequence ending with A[m] and extend it with A[i] to obtain the longest increasing subsequence which ends with A[i]:

• Longest Increasing Subsequence: Given a sequence of n real numbers A[1..n], determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence are strictly

- Solution: For each  $i \leq n$  we solve the following subproblems:
- Subtroblem P(i):/Jind a subsequence of the sequence A[1..i] of maxhttps://powcoder.com
  and which ends with P.
- Recursion: Assume we have solved the subproblems for all j < i, and that Add we Chatipowcoder
- We now look for all A[m] such that m < i and such that A[m] < A[i].
- Among those we pick m which produced the longest increasing subsequence ending with A[m] and extend it with A[i] to obtain the longest increasing subsequence which ends with A[i]:

• Longest Increasing Subsequence: Given a sequence of n real numbers A[1..n], determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence are strictly

- Solution: For each  $i \leq n$  we solve the following subproblems:
- Subproblem P(i): Find a subsequence of the sequence A[1..i] of maximum lings in which the last of the fly including and which ends with A[i].
- Recursion: Assume we have solved the subproblems for all j < i, and that Add where Chatipowcoder
- We now look for all A[m] such that m < i and such that A[m] < A[i].
- Among those we pick m which produced the longest increasing subsequence ending with A[m] and extend it with A[i] to obtain the longest increasing subsequence which ends with A[i]:

• Longest Increasing Subsequence: Given a sequence of n real numbers A[1..n], determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence are strictly

- Solution: For each  $i \leq n$  we solve the following subproblems:
- Subproblem P(i): Find a subsequence of the sequence A[1..i] of maximum lings in which the last of the fly including and which ends with A[i].
- Recursion: Assume we have solved the subproblems for all j < i, and that we have just martable S the values  $S[j] = \ell_j$ , which are the lengths  $\ell_j$  of maximal increasing sequences which each wind [j]
- We now look for all A[m] such that m < i and such that A[m] < A[i].
- Among those we pick m which produced the longest increasing subsequence ending with A[m] and extend it with A[i] to obtain the longest increasing subsequence which ends with A[i]:

• Longest Increasing Subsequence: Given a sequence of n real numbers A[1..n], determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence are strictly

- Solution: For each  $i \leq n$  we solve the following subproblems:
- Subproblem P(i): Find a subsequence of the sequence A[1..i] of maximum lings in which with A[i].
- Recursion: Assume we have solved the subproblems for all j < i, and that we have just martable S the values  $S[j] = \ell_j$ , which are the lengths  $\ell_j$  of maximal increasing sequences which enough  $S[j] = \ell_j$ .
- We now look for all A[m] such that m < i and such that A[m] < A[i].
- Among those we pick m which produced the longest increasing subsequence ending with A[m] and extend it with A[i] to obtain the longest increasing subsequence which ends with A[i]:

• Longest Increasing Subsequence: Given a sequence of n real numbers A[1..n], determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence are strictly

- Solution: For each  $i \leq n$  we solve the following subproblems:
- Subproblem P(i): Find a subsequence of the sequence A[1..i] of maximum lings in who the with A[i].
- Recursion: Assume we have solved the subproblems for all j < i, and that we have just martable S the values  $S[j] = \ell_j$ , which are the lengths  $\ell_j$  of maximal increasing sequences which enough  $S[j] = \ell_j$ .
- We now look for all A[m] such that m < i and such that A[m] < A[i].
- Among those we pick m which produced the longest increasing subsequence ending with A[m] and extend it with A[i] to obtain the longest increasing subsequence which ends with A[i]:

$$\begin{split} \ell_i &= \max\{\ell_m \ : \ m < i \ \& \ A[m] < A[i]\} + 1 \\ \pi(i) &= \arg\max\{\ell_m \ : \ m < i \ \& \ A[m] < A[i]\} \end{split}$$

### Assignment Project Exam Help

subsequence ending with A[i] and  $\pi(i) = m$  such that the optimal solution for P(i) extends the optimal solution for P(m).

- So, https://powcoder.com/sequence of the sequence A[1..i] which ends with A[i].
- Finally, from all such subsequences we pick the longest one Add WeChat powcoder
- The end point of such a sequence can be obtained as

end = 
$$\arg\max\{\ell_i : i \le n\}$$



$$\ell_i = \max\{\ell_m : m < i \& A[m] < A[i]\} + 1$$
  
$$\pi(i) = \arg\max\{\ell_m : m < i \& A[m] < A[i]\}$$

Assignment Project Exam Help We store in the  $i^{th}$  slot of the table the length  $\ell_i$  of the longest increasing subsequence ending with A[i] and  $\pi(i) = m$  such that the optimal

- So, https://powcoder.com sequence of the sequence A[1..i] which ends with A[i].
- Finally, from all such subsequences we pick the longest one Add WeChat powcoder

solution for P(i) extends the optimal solution for P(m).

• The end point of such a sequence can be obtained as

end = 
$$\arg\max\{\ell_i : i \leq n\}$$



$$\ell_i = \max\{\ell_m : m < i \& A[m] < A[i]\} + 1$$
  
$$\pi(i) = \arg\max\{\ell_m : m < i \& A[m] < A[i]\}$$

### Assignment Project Exam Help the length $\ell_i$ of the longest increasing

- We store in the  $i^{th}$  slot of the table the length  $\ell_i$  of the longest increasing subsequence ending with A[i] and  $\pi(i) = m$  such that the optimal solution for P(i) extends the optimal solution for P(m).
- So, we late P is for P is P in P
- Finally, from all such subsequences we pick the longest one Add WeChat powcoder
- The end point of such a sequence can be obtained as

end = 
$$\arg\max\{\ell_i : i \leq n\}$$



$$\ell_i = \max\{\ell_m : m < i \& A[m] < A[i]\} + 1$$
  
$$\pi(i) = \arg\max\{\ell_m : m < i \& A[m] < A[i]\}$$

### Assignment Project Exam Help We see in the 1<sup>th</sup> slot of the table the length $\ell_i$ of the longest increasing

- We store in the  $i^m$  slot of the table the length  $\ell_i$  of the longest increasing subsequence ending with A[i] and  $\pi(i) = m$  such that the optimal solution for P(i) extends the optimal solution for P(m).
- So, we late P is for P is P in P
- The end point of such a sequence can be obtained as

end = 
$$\arg \max\{\ell_i : i \le n\}$$



$$\ell_i = \max\{\ell_m : m < i \& A[m] < A[i]\} + 1$$
  
$$\pi(i) = \arg\max\{\ell_m : m < i \& A[m] < A[i]\}$$

### Assignment Project Exam Help We see in the 1<sup>th</sup> slot of the table the length $\ell_i$ of the longest increasing

- We store in the  $i^{in}$  slot of the table the length  $\ell_i$  of the longest increasing subsequence ending with A[i] and  $\pi(i) = m$  such that the optimal solution for P(i) extends the optimal solution for P(m).

- The end point of such a sequence can be obtained as

end = 
$$\arg\max\{\ell_i : i \leq n\}$$



# As We can now reconstruct the longest monotonically increasing requence Assign M for the level of Assign M and Assign M for Assign M and Assign M for Assign M and Assign M for Assign M

• Againttps://powcoder.com restrictive because if the optimal solution ends with some A[m], it would have been constructed as the solution for P(m).

### Add WeChat powcoder

• Exercise: (somewhat tough, but very useful) Design an algorithm for solving this problem which runs in time  $n \log n$ .

# As We can now reconstruct the longest monotonically increasing Henry $\pi$ and $\pi$ end, $\pi$ (end), $\pi$ ( $\pi$ (end)), . . .

• Again, the partial force squeezed A[m], it would have been constructed as the solution for P(m).

### Add WeChat powcoder

• Exercise: (somewhat tough, but very useful) Design an algorithm for solving this problem which runs in time  $n \log n$ .

# As We can now reconstruct the longest monotonically increasing Henry $\pi$ and $\pi$ end, $\pi$ (end), $\pi$ ( $\pi$ (end)), . . .

- Again, the partial force squeezed A[m], it would have been constructed as the solution for P(m).
- Time And daty: We Chat powcoder
- Exercise: (somewhat tough, but very useful) Design an algorithm for solving this problem which runs in time  $n \log n$ .

# As We can now reconstruct the longest monotonically increasing requence Assign M for the level of Assign M and Assign M for Assign M and Assign M for Assign M and Assign M for Assign M

- Again, the patient of the sequence of the patient of the sequence of the patient of the patient of the sequence of the patient of the pati
- Time And ty: We Chat powcoder
- Exercise: (somewhat tough, but very useful) Design an algorithm for solving this problem which runs in time  $n \log n$ .

- Making Change. You are given n types of coin denominations of values (1) = (2) + (2)
  - Soluhttps://powcoder.comle containing C many slots, so that an optimal solution for an amount i is stored in slot i.
  - If CAdd WeChat powcoder
  - Assume we have found optimal solutions for every amount j < i and now want to find an optimal solution for amount i.

- Making Change. You are given n types of coin denominations of values  $(1) \circ (2) \circ (2)$ 
  - Solution by Rear sio polytecoder with Table containing C many slots, so that an optimal solution for an amount i is stored in slot i.
  - If CAdd WeChat powcoder
  - Assume we have found optimal solutions for every amount j < i and now want to find an optimal solution for amount i.

- Making Change. You are given n types of coin denominations of values v(1) = v(2) + v(n) (Thinteger's) Assuming v(1) = 1, so that for dan makes change for any given integer amount C with as few coins as possible, assuming that you have an unlimited supply of coins of each denomination.
  - Solution processing of the solution for an amount i is stored in slot i.
  - If C = 1, the slutty sevial parts power and the property of the slutty sevial parts and the property of the slutty sevial parts and the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts are property
  - Assume we have found optimal solutions for every amount j < i and now want to find an optimal solution for amount i.

- Making Change. You are given n types of coin denominations of values v(1) = v(2) + v(n) (Thinteger's) Assume v(1) = 1, so that for dan make change for any integer amount C with as few coins as possible, assuming that you have an unlimited supply of coins of each denomination.
- Solution processing of the solution for an amount i is stored in slot i.
- If C = 1, the slutty sevial parts power and the property of the slutty sevial parts and the property of the slutty sevial parts and the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts and the slutty sevial parts are property of the slutty sevial parts are property
- Assume we have found optimal solutions for every amount j < i and now want to find an optimal solution for amount i.

• We consider optimal solutions opt(i-v(k)) for every amount of the form i-v(k), where k ranges from 1 to n. (Recall  $v(1),\ldots,v(n)$  are all of the

# available denominations.) SSIGNMENT Project Exam Help SSIGNMENT Project Exam Help

• we https://powcoder.com.pt(i - v(m))

# 

- Consider an optimal solution for amount  $i \leq C$ ; and say such solution includes

• We consider optimal solutions opt(i-v(k)) for every amount of the form i-v(k), where k ranges from 1 to n. (Recall  $v(1),\ldots,v(n)$  are all of the available denominations.)

### ssignment Project Exam Help

- constructing recursively!) we pick one which uses the fewest number of coins, say this is opt(i - v(m)) for some  $m, 1 \le m \le n$ .
- we https://powcoder.com

# $\begin{array}{c} \operatorname{opt}(i) = \min\{\operatorname{opt}(i-v(k)) \ : \ 1 \leq k \leq n\} + 1 \\ \mathbf{Add} \ \ \mathbf{WeChat} \ \ \mathbf{powcoder} \\ \bullet \ \ \text{Why does this produce an optimal solution for amount } i \leq C? \end{array}$

- Consider an optimal solution for amount  $i \leq C$ ; and say such solution includes

• We consider optimal solutions opt(i-v(k)) for every amount of the form i-v(k), where k ranges from 1 to n. (Recall  $v(1), \ldots, v(n)$  are all of the available denominations.)

## Assignment Project Exam Help

- Among all of these optimal solutions (which we find in the table we are constructing recursively!) we pick one which uses the fewest number of coins, say this is opt(i-v(m)) for some  $m, 1 \le m \le n$ .
- We obtain a primal scale of the Cathard Scale of

# Add $\overset{\text{opt}(i) = \min\{\text{opt}(i - v(k)) : 1 \le k \le n\} + 1}{\text{WeChat powcoder}}$

- Why does this produce an optimal solution for amount  $i \leq C$ ?
- Consider an optimal solution for amount  $i \leq C$ ; and say such solution includes at least one coin of denomination v(m) for some  $1 \leq m \leq n$ . But then removing such a coin must produce an optimal solution for the amount i v(m) again by our cut-and-paste argument.

• We consider optimal solutions opt(i-v(k)) for every amount of the form i-v(k), where k ranges from 1 to n. (Recall  $v(1), \ldots, v(n)$  are all of the available denominations.)

## Assignment Project Exam Help

- Among all of these optimal solutions (which we find in the table we are constructing recursively!) we pick one which uses the fewest number of coins, say this is opt(i-v(m)) for some  $m, 1 \le m \le n$ .
- We obtain a primal scape w(i) Cooler by Cooler

## Add $\overset{\text{opt}(i) = \min\{\text{opt}(i - v(k)) : 1 \le k \le n\} + 1}{\text{WeChat powcoder}}$

- Why does this produce an optimal solution for amount  $i \leq C$ ?
- Consider an optimal solution for amount  $i \leq C$ ; and say such solution includes at least one coin of denomination v(m) for some  $1 \leq m \leq n$ . But then removing such a coin must produce an optimal solution for the amount i v(m) again by our cut-and-paste argument.

• We consider optimal solutions opt(i-v(k)) for every amount of the form i-v(k), where k ranges from 1 to n. (Recall  $v(1), \ldots, v(n)$  are all of the available denominations.)

# ssignment Project Exam Help

- Among all of these optimal solutions (which we find in the table we are constructing recursively!) we pick one which uses the fewest number of coins, say this is opt(i-v(m)) for some  $m, 1 \le m \le n$ .
- We obtain spin also with the point of denomination v(m).

## Add $\overset{\text{opt}(i) = \min\{\text{opt}(i - v(k)) : 1 \le k \le n\} + 1}{\text{WeChat powcoder}}$

- Why does this produce an optimal solution for amount  $i \leq C$ ?
- Consider an optimal solution for amount  $i \leq C$ ; and say such solution includes at least one coin of denomination v(m) for some  $1 \leq m \leq n$ . But then removing such a coin must produce an optimal solution for the amount i v(m) again by our cut-and-paste argument.

• We consider optimal solutions opt(i-v(k)) for every amount of the form i-v(k), where k ranges from 1 to n. (Recall  $v(1), \ldots, v(n)$  are all of the available denominations.)

# Ssignment Project Exam Help Among all of these optimal solutions which we find in the table we are

- Among all of these optimal solutions, which we find in the table we are constructing recursively!) we pick one which uses the fewest number of coins, say this is opt(i-v(m)) for some  $m, 1 \le m \le n$ .
- We obtain a primal scale of the contraction v(m).

## Add WeChat powcoder

- Why does this produce an optimal solution for amount  $i \leq C$ ?
- Consider an optimal solution for amount  $i \leq C$ ; and say such solution includes at least one coin of denomination v(m) for some  $1 \leq m \leq n$ . But then removing such a coin must produce an optimal solution for the amount i v(m) again by our cut-and-paste argument.

ullet However, we do not know which coins the optimal solution includes, so we try all the available coins and then pick m for which the optimal solution for

## Assignment Project Exam Help It is enough to store in the ith slot of the table such m and opt(i) because this

• It is enough to store in the  $i^{th}$  slot of the table such m and opt(i) because this allows us to reconstruct the optimal solution by looking at  $m_1$  stored in the  $i^{th}$  slot, then look at  $m_2$  stored in the slot  $i-v(m_1)$ , then look at  $m_2$  stored in the slot  $i-v(m_1)$ . Dowcoder.com

• opt(C) is the solution we need.

### \* Add WeChat powcoder

- Note: Our algorithm is NOT a polynomial time algorithm in the length of the input, because the length of a representation of C is only  $\log C$ , while the running time is nC.
- But this is the best what we can do...



ullet However, we do not know which coins the optimal solution includes, so we try all the available coins and then pick m for which the optimal solution for

### Assignment Project Exam Help

- It is enough to store in the  $i^{th}$  slot of the table such m and opt(i) because this allows us to reconstruct the optimal solution by looking at  $m_1$  stored in the  $i^{th}$  slot, then look at  $m_2$  stored in the slot  $i-v(m_1)$ , then look at  $m_2$  stored in the slot i
- opt(C) is the solution we need.
- \* Add WeChat powcoder
- Note: Our algorithm is NOT a polynomial time algorithm in the length of the input, because the length of a representation of C is only  $\log C$ , while the running time is nC.
- But this is the best what we can do...



ullet However, we do not know which coins the optimal solution includes, so we try all the available coins and then pick m for which the optimal solution for

### Assignment Project Exam Help

- It is enough to store in the  $i^{th}$  slot of the table such m and opt(i) because this allows us to reconstruct the optimal solution by looking at  $m_1$  stored in the  $i^{th}$  slot, then look at  $m_2$  stored in the slot  $i-v(m_1)$ , then look at  $m_2$  stored in the slot  $i-v(m_1)$ , then look at  $m_2$  stored in the slot  $i-v(m_1)$ . DOWCOGET.COM
- opt(C) is the solution we need.
- \* Add WeChat powcoder
- Note: Our algorithm is NOT a polynomial time algorithm in the length of the input, because the length of a representation of C is only  $\log C$ , while the running time is nC.
- But this is the best what we can do...



• However, we do not know which coins the optimal solution includes, so we try all the available coins and then pick m for which the optimal solution for

## Assignment from the from the from the state of the table such m and opt(i) because this

- It is enough to store in the  $i^{th}$  slot of the table such m and opt(i) because this allows us to reconstruct the optimal solution by looking at  $m_1$  stored in the  $i^{th}$  slot, then look at  $m_2$  stored in the slot  $i-v(m_1)$ , then look at  $m_2$  stored in the slot i 100 WCOGET. COM
- $\bullet$  opt(C) is the solution we need.
- Time Amplexity of Wales of the is at powcoder
- Note: Our algorithm is NOT a polynomial time algorithm in the length of the input, because the length of a representation of C is only  $\log C$ , while the running time is nC.
- But this is the best what we can do...



• However, we do not know which coins the optimal solution includes, so we try all the available coins and then pick m for which the optimal solution for

## Assignment for the strumber of crins Exam Help • It is enough to store in the $i^{th}$ slot of the table such m and opt(i) because this

- It is enough to store in the  $i^{th}$  slot of the table such m and opt(i) because this allows us to reconstruct the optimal solution by looking at  $m_1$  stored in the  $i^{th}$  slot, then look at  $m_2$  stored in the slot  $i-v(m_1)$ , then look at  $m_2$  stored in the slot i  $v(m_1)$ ,  $v(m_2)$ ,  $v(m_1)$ ,  $v(m_2)$ ,  $v(m_2)$
- $\bullet$  opt(C) is the solution we need.
- Time Amplexity of Welchingt powcoder
- Note: Our algorithm is NOT a polynomial time algorithm in the **length** of the input, because the length of a representation of C is only  $\log C$ , while the running time is nC.
- But this is the best what we can do...



• However, we do not know which coins the optimal solution includes, so we try all the available coins and then pick m for which the optimal solution for

## Assignment for the strumber of crins Exam Help • It is enough to store in the $i^{th}$ slot of the table such m and opt(i) because this

- It is enough to store in the  $i^{th}$  slot of the table such m and opt(i) because this allows us to reconstruct the optimal solution by looking at  $m_1$  stored in the  $i^{th}$  slot, then look at  $m_2$  stored in the slot  $i v(m_1)$ , then look at  $m_2$  stored in the slot  $i v(m_1)$ , then look at  $m_2$  stored in the slot  $i v(m_1)$ .
- ullet opt(C) is the solution we need.
- Time Amplexity of Welchingt powcoder
- Note: Our algorithm is NOT a polynomial time algorithm in the **length** of the input, because the length of a representation of C is only  $\log C$ , while the running time is nC.
- But this is the best what we can do...



Integer Knapsack Problem (Duplicate Items Allowed) You have n types of items; all items of kind i are identical and of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is plarge as possible. You can take any number of Aris Stagnam Help

- ullet Solution: DP recursion on the capacity C of the knapsack.
- we https://powcoder.com ies i < C.
- Assume we have solved the problem for all knapsacks of capacities j < i.
- We Add We Chat powcoder cities
- Chose the one for which  $opt(i-w_m) + v_m$  is the largest;
- Add to such optimal solution for the knapsack of size  $i-w_m$  item m to obtain a packing of a knapsack of size i of the highest possible value.

Integer Knapsack Problem (Duplicate Items Allowed) You have n types of items; all items of kind i are identical and of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is plarge as possible. You can take any number of the STATIMENT PROJECT EXAM Help

- $\bullet$  Solution: DP recursion on the capacity C of the knapsack.
- we https://powcoder.com is i \le C.
- Assume we have solved the problem for all knapsacks of capacities j < i.
- We Add We Chat powcoder cities
- Chose the one for which  $opt(i-w_m) + v_m$  is the largest;
- Add to such optimal solution for the knapsack of size  $i-w_m$  item m to obtain a packing of a knapsack of size i of the highest possible value.

Integer Knapsack Problem (Duplicate Items Allowed) You have n types of items; all items of kind i are identical and of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is plarge as possible. You can take any number of the straightment Project Exam Help

- ullet Solution: DP recursion on the capacity C of the knapsack.
- We but the Strategy of the polyment of the properties  $i \leq C$ .
- Assume we have solved the problem for all knapsacks of capacities j < i.
- We Add We Chat powcoder cities
- Chose the one for which  $opt(i w_m) + v_m$  is the largest;
- Add to such optimal solution for the knapsack of size  $i-w_m$  item m to obtain a packing of a knapsack of size i of the highest possible value.

Integer Knapsack Problem (Duplicate Items Allowed) You have n types of items; all items of kind i are identical and of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is plarge as possible. You can take any number of the state of the combination of the knapsack and whose value is plarge as possible. You can take any number of the state of the combination of the knapsack and whose value is plarge as possible. You can take any number of the combination of the knapsack and whose value is plarge as possible. You can take any number of the combination of the knapsack and whose value is plarge as possible.

- ullet Solution: DP recursion on the capacity C of the knapsack.
- We but this of optimisation the packet of the properties  $i \leq C$ .
- Assume we have solved the problem for all knapsacks of capacities j < i.
- We Add We Chat powcoder cities
- Chose the one for which  $opt(i w_m) + v_m$  is the largest;
- Add to such optimal solution for the knapsack of size  $i-w_m$  item m to obtain a packing of a knapsack of size i of the highest possible value.

Integer Knapsack Problem (Duplicate Items Allowed) You have n types of items; all items of kind i are identical and of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is plarge as possible. You can take any number of the state of the combination of the knapsack and whose value is plarge as possible. You can take any number of the state of the combination of the knapsack and whose value is plarge as possible. You can take any number of the combination of the knapsack and whose value is plarge as possible. You can take any number of the combination of the knapsack and whose value is plarge as possible.

- ullet Solution: DP recursion on the capacity C of the knapsack.
- We but this point in some parties  $i \leq C$ .
- ullet Assume we have solved the problem for all knapsacks of capacities j < i.
- We not look at optimal solutions pt(i+w) for all knopsecks of capacities i-w and i-w and i-w are the constant of th
- Chose the one for which  $opt(i w_m) + v_m$  is the largest;
- Add to such optimal solution for the knapsack of size  $i-w_m$  item m to obtain a packing of a knapsack of size i of the highest possible value.

Integer Knapsack Problem (Duplicate Items Allowed) You have n types of items; all items of kind i are identical and of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is plarge as possible. You can take any number of the state of the combination of the knapsack and whose value is plarge as possible. You can take any number of the state of the combination of the knapsack and whose value is plarge as possible. You can take any number of the combination of the knapsack and whose value is plarge as possible. You can take any number of the combination of the knapsack and whose value is plarge as possible.

- ullet Solution: DP recursion on the capacity C of the knapsack.
- We but this of optimisation the packet of the properties  $i \leq C$ .
- Assume we have solved the problem for all knapsacks of capacities j < i.
- We not look at optimal solutions pt(itw) for all knapsacks of capacities i-w and v
- Chose the one for which  $opt(i-w_m)+v_m$  is the largest;
- Add to such optimal solution for the knapsack of size  $i w_m$  item m to obtain a packing of a knapsack of size i of the highest possible value.

Integer Knapsack Problem (Duplicate Items Allowed) You have n types of items; all items of kind i are identical and of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is plarge as possible. You can take any number of Aris Stagnam Help

- ullet Solution: DP recursion on the capacity C of the knapsack.
- We but this of optimisation the packet of the properties  $i \leq C$ .
- Assume we have solved the problem for all knapsacks of capacities j < i.
- We not look at optimal solutions pt(i+w) for all knapsacks of capacities i-w and i-w and i-w and i-w are the contractions of the contraction of the cont
- Chose the one for which  $opt(i-w_m)+v_m$  is the largest;
- Add to such optimal solution for the knapsack of size  $i w_m$  item m to obtain a packing of a knapsack of size i of the highest possible value.

Afte Commy steps / obtain the optimal (pinimal) number of coins opt(C).
 https://powcoder.com
 Which coins are present in the optimal solution can again be obtained by

• Which coins are present in the optimal solution can again be obtained by backtracking: if  $\pi(C) = k$  then the first object is  $a_k$  of weight  $w_k$  and value  $v_k$  if  $\pi(C - w_k) = m$  then the second object is  $a_m$  and so on.

Note Add We Chat powcoder good solutions we pick arbitrarily among them.

• Again, our algorithm is **NOT** polynomial in the **length** of the input.

Assignment in the project between the proje

- After C many steps we/obtain the optimal (minimal) number of coins opt(C).

  \*\*Proposition\*\*

  \*\*Propos
- Which coins are present in the optimal solution can again be obtained by backtracking: if  $\pi(C) = k$  then the first object is  $a_k$  of weight  $w_k$  and value  $v_k$  if  $\pi(C w_k) = m$  then the second object is  $a_m$  and so on.
- Note Add We Chat powcoder good solutions we pick arbitrarily among them.
- Again, our algorithm is **NOT** polynomial in the **length** of the input.

Assignment in the project between the proje

• After C many steps we/obtain the optimal (minimal) number of coins opt(C).

\*\*Notice of the initial content of the initial number of the initial nu

• Which coins are present in the optimal solution can again be obtained by backtracking: if  $\pi(C) = k$  then the first object is  $a_k$  of weight  $w_k$  and value  $v_k$ ; if  $\pi(C - w_k) = m$  then the second object is  $a_m$  and so on.

Note Add We Chat powcoder good solutions we pick arbitrarily among them.

• Again, our algorithm is **NOT** polynomial in the **length** of the input.

Assignment in the project between the proje

- After C many steps we/obtain the optimal (minimal) number of coins opt(C). **NUDS:** / **DOWCOGET.COM**
- Which coins are present in the optimal solution can again be obtained by backtracking: if  $\pi(C) = k$  then the first object is  $a_k$  of weight  $w_k$  and value  $v_k$ ; if  $\pi(C w_k) = m$  then the second object is  $a_m$  and so on.
- Note that do might not be charged atterprine; we contain equally good solutions we pick arbitrarily among them.
- Again, our algorithm is **NOT** polynomial in the **length** of the input.

Assignment in the project between the proje

- After C many steps we/obtain the optimal (minimal) number of coins opt(C).

  \*\*DOWCOGET.COM\*\*
- Which coins are present in the optimal solution can again be obtained by backtracking: if  $\pi(C) = k$  then the first object is  $a_k$  of weight  $w_k$  and value  $v_k$ ; if  $\pi(C w_k) = m$  then the second object is  $a_m$  and so on.
- Note that do might not be an quely determined; in case of mantple equally good solutions we pick arbitrarily among them.
- Again, our algorithm is **NOT** polynomial in the **length** of the input.

• Integer Knapsack Problem (Duplicate Items NOT Allowed) You have n items (some of which can be identical); item  $I_i$  is of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available

# ssignment Project Exam Hel

# https://powcoder.com

- Fix now i < n and c < C and assume we have solved the subproblems for:
- all j < i and all knapsacks of capacities from 1 to C;
  AddaeWeChat powcoder



• Integer Knapsack Problem (Duplicate Items NOT Allowed) You have n items (some of which can be identical); item  $I_i$  is of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is as large as passible.

This can example of a "2D" recursion; we will be filling a table of size  $n \times C$ , row by row; subproblems P(i,c) for all  $i \le n$  and  $c \le C$  will be of the form:

chost from items  $I_1, I_2, \dots, I_t$  a subset which fits in a knapsack of capacity c and f to f

- Fix now  $i \leq n$  and  $c \leq C$  and assume we have solved the subproblems for:
  - Add WeChat powcoder



• Integer Knapsack Problem (Duplicate Items NOT Allowed) You have n items (some of which can be identical); item  $I_i$  is of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is as large as p as ble 1

This can example of a "2D" recursion; we will be filling a table of size  $n \times C$ , row by row; subproblems P(i, c) for all  $i \le n$  and  $c \le C$  will be of the form:

- Fix now  $i \leq n$  and  $c \leq C$  and assume we have solved the subproblems for:
  - Add WeChat powcoder

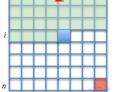
• Integer Knapsack Problem (Duplicate Items NOT Allowed) You have n items (some of which can be identical); item  $I_i$  is of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knip ack and whose value is as large as possible.

This can example of a "2D" recursion; we will be filling a table of size  $n \times$ row by row; subproblems P(i, c) for all  $i \le n$  and  $c \le C$  will be of the form:

chose from items  $I_1, I_2, \dots, I_r$  a subset which fits in a knapsack of capacity c and is placed by the Original Property of the Original Prop

- Fix now i < n and c < C and assume we have solved the subproblems for:

  - all j < i and all knapsacks of capacities from 1 to C;
     Add We hat powcoder



• Integer Knapsack Problem (Duplicate Items NOT Allowed) You have n items (some of which can be identical); item  $I_i$  is of weight  $w_i$  and value  $v_i$ . You also have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is as large as possible.

• This can example of a "2D" recursion; we will be filling a table of size  $n \times C$ , row by row; subproblems P(i,c) for all  $i \le n$  and  $c \le C$  will be of the form:

chose from items  $I_1, I_2, \dots, I_t$  a subset which fits in a knapsack of capacity c and if  $I_1$  to  $I_2$  for  $I_3$  to  $I_4$  t

- Fix now  $i \leq n$  and  $c \leq C$  and assume we have solved the subproblems for:
  - **1** all  $j \le i$  and all knapsacks of capacities from 1 to C;
  - · A deday Week hour for by coder



• we now have two options: either we take item  $I_i$  or we do not;

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- if  $opt(i-1, c-w_i) + v_i > opt(i-1, c)$ then  $opt(i, c) = opt(i-1, c-w_i) + v_i;$ else opt(i, c) = opt(i-1, c).
- Final solution will be given by opt(n, C).

• we now have two options: either we take item  $I_i$  or we do not;

Assignment Project Exam Help

https://powcoder.com Add WeChat powcoder

- if  $opt(i-1, c-w_i) + v_i > opt(i-1, c)$ then  $opt(i, c) = opt(i-1, c-w_i) + v_i;$ else opt(i, c) = opt(i-1, c).
- Final solution will be given by opt(n, C).



• we now have two options: either we take item  $I_i$  or we do not;

Assignment Project Exam Help

https://powcoder.com Add WeChat powcoder

- if  $opt(i-1, c-w_i) + v_i > opt(i-1, c)$ then  $opt(i, c) = opt(i-1, c-w_i) + v_i;$ else opt(i, c) = opt(i-1, c).
- Final solution will be given by opt(n, C).



• we now have two options: either we take item  $I_i$  or we do not;

Assignment Project Exam Help

https://powcoder.com Add WeChat powcoder

- if  $opt(i-1, c-w_i) + v_i > opt(i-1, c)$ then  $opt(i, c) = opt(i-1, c-w_i) + v_i;$ else opt(i, c) = opt(i-1, c).
- Final solution will be given by opt(n, C).

• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

# Assignment Project Exam Help

of size S/2 and with each integer  $x_i$  of both size and value equal to  $x_i$ .

- Claim the best part of such knaps of produces optimally balanced part  $\frac{1}{2}$  powcoder. Com  $\frac{1}{2}$  all the integers let  $\frac{1}{2}$  all the
- Why? Since  $S = S_1 + S_2$  we obtain

### Add WeChat powcoder

- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.

• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

As supplied that supplies it the supplied it is a supplied in the supplied in

- Claim the best part power of such knaps of produces optimally balanced part https://powcoder.com S2 all the
- Why? Since  $S = S_1 + S_2$  we obtain

### Add WeChat powcoder

- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.

• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

Assign problem (with duplicat items not allowed) with the knapsak of size S/2 and with each integer  $x_i$  of both size and value equal to  $x_i$ .

- Claim: the best packing of such knapsack produces optimally balanced participh, with  $S_1$  being 11 Lyantegers in the knapsack and  $S_2$  all the integers left out of the knapsack.
- Why? Since  $S = S_1 + S_2$  we obtain

## Add WeChat powcoder

- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.

• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

Assign problem (with duplicat items not allowed) with the knapsak of size S/2 and with each integer  $x_i$  of both size and value equal to  $x_i$ .

- Claim: the best packing of such knapsack produces optimally balanced participh, with  $S_1$  being 11 Lyantegers in the knapsack and  $S_2$  all the integers left out of the knapsack.
- Why? Since  $S = S_1 + S_2$  we obtain

## Add WeChat powcoder

- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.

• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

As supplying the transmitted from the Knapsack problem (with duplicat items not allowed) with the knapsack of size S/2 and with each integer  $x_i$  of both size and value equal to  $x_i$ .

- Claim: the best packing of such knapsack produces optimally balanced participh, with  $S_1$  being 11 Lyantegers in the knapsack and  $S_2$  all the integers left out of the knapsack.
- Why? Since  $S = S_1 + S_2$  we obtain

# Add WeChat-poweoder

- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.

• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

Assign problem (with duplicat items not allowed) with the knapsak of size S/2 and with each integer  $x_i$  of both size and value equal to  $x_i$ .

- Claim: the best packing of such knapsack produces optimally balanced participh, with  $S_1$  being 11 Lyantegers in the knapsack and  $S_2$  all the integers left out of the knapsack.
- Why? Since  $S = S_1 + S_2$  we obtain

# Add WeChat-poweoder

- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.

• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

As super problem (with duplicat items not allowed) with the knapsak of size S/2 and with each integer  $x_i$  of both size and value equal to  $x_i$ .

- Claim: the best packing of such knapsack produces optimally balanced participh, with  $S_1$  being 11 Lyantegers in the knapsack and  $S_2$  all the integers left out of the knapsack.
- Why? Since  $S = S_1 + S_2$  we obtain

# Add WeChat-poweoder

i.e.  $S_2 - S_1 = 2(S/2 - S_1)$ .

- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.

17 / 41

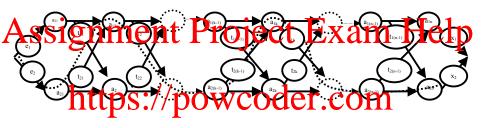
• Balanced Partition You have a set of n integers. Partition these integers into two subsets such that you minimise  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.

As supplying the training of the state of size S/2 and with each integer  $x_i$  of both size and value equal to  $x_i$ .

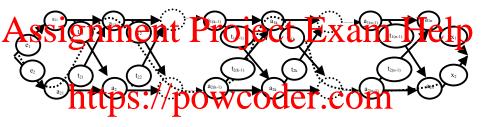
- Claim: the best packing of such knapsack produces optimally balanced participat, with  $S_1$  being 11 Winteger in the knapsack and  $S_2$  all the integers left out of the knapsack.
- Why? Since  $S = S_1 + S_2$  we obtain

# Add WeChat-poweoder

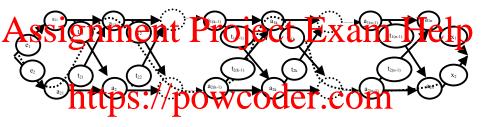
- Thus, minimising  $S/2 S_1$  will minimise  $S_2 S_1$ .
- So, all we have to do is find a subset of these numbers with the largest possible total sum which fits inside a knapsack of size S/2.



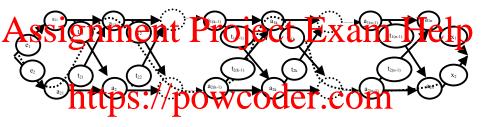
- On the Add mb We Chate powcoder time to complete; on the second assembly line the Pme job takes  $a_{2,k}$  units of time.
- To move the product from station k-1 on the first assembly line to station k on the second line it takes  $t_{1,k-1}$  units of time.
- Likewise, to move the product from station k-1 on the second assembly line to station k on the first assembly line it takes  $t_{2,k-1}$  units of time.



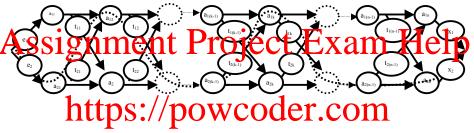
- On the just assembly in the  $k^{th}$  jby takes a . We find the to complete; on the second assembly line the same job takes  $a_{2,k}$  units of time.
- To move the product from station k-1 on the first assembly line to station k on the second line it takes  $t_{1,k-1}$  units of time.
- Likewise, to move the product from station k-1 on the second assembly line to station k on the first assembly line it takes  $t_{2,k-1}$  units of time.



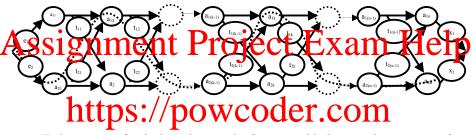
- On the intrasembly in the  $k^{ij}$  place  $a_{2,k}$  units of time to complete; on the second assembly line the same job takes  $a_{2,k}$  units of time.
- To move the product from station k-1 on the first assembly line to station k on the second line it takes  $t_{1,k-1}$  units of time.
- Likewise, to move the product from station k-1 on the second assembly line to station k on the first assembly line it takes  $t_{2,k-1}$  units of time.



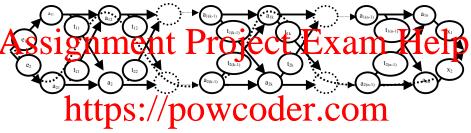
- On the intrasembly in the  $k^{il}$  job takes a few units of time to complete; on the second assembly line the same job takes  $a_{2,k}$  units of time.
- To move the product from station k-1 on the first assembly line to station k on the second line it takes  $t_{1,k-1}$  units of time.
- Likewise, to move the product from station k-1 on the second assembly line to station k on the first assembly line it takes  $t_{2,k-1}$  units of time.



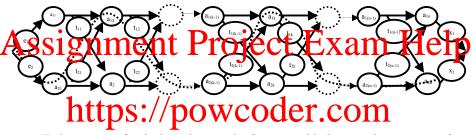
- ullet To bring an unfinished product to the first assembly line it takes  $e_1$  units of time.
- Add WeChat powcoder
- To get a finished product from the first assembly line to the warehouse it takes  $x_1$  units of time;
- $\bullet$  To get a finished product from the second assembly line to the warehouse it takes  $x_2$  units.
- Task: Find a fastest way to assemble a product using both lines as necessary.



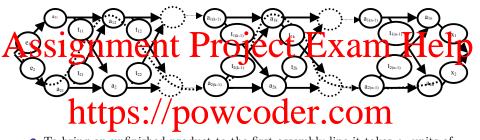
- To bring an unfinished product to the first assembly line it takes  $e_1$  units of time.
- $\bullet$  To bring an unfinished product to the second assembly line it takes  $e_2$  units of time. Add WeChat powcoder
- To get a finished product from the first assembly line to the warehouse it takes  $x_1$  units of time;
- $\bullet$  To get a finished product from the second assembly line to the warehouse it takes  $x_2$  units.
- Task: Find a fastest way to assemble a product using both lines as necessary.



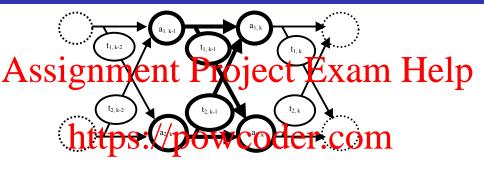
- To bring an unfinished product to the first assembly line it takes  $e_1$  units of time.
- $\bullet$  To bring an unfinished product to the second assembly line it takes  $e_2$  units of time. Add WeChat powcoder
- To get a finished product from the first assembly line to the warehouse it takes  $x_1$  units of time;
- $\bullet$  To get a finished product from the second assembly line to the warehouse it takes  $x_2$  units.
- Task: Find a fastest way to assemble a product using both lines as necessary.



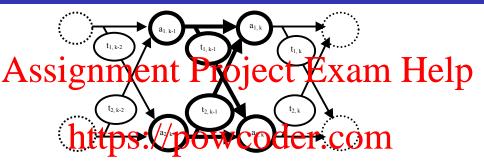
- To bring an unfinished product to the first assembly line it takes  $e_1$  units of time.
- $\bullet$  To bring an uninished product to the second assembly line it takes  $e_2$  units of time. Add We Chat powcoder
- To get a finished product from the first assembly line to the warehouse it takes  $x_1$  units of time;
- ullet To get a finished product from the second assembly line to the warehouse it takes  $x_2$  units.
- Task: Find a fastest way to assemble a product using both lines as necessary.



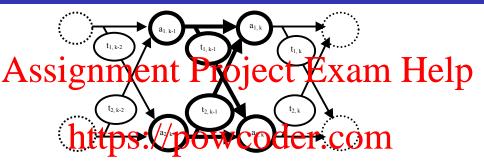
- ullet To bring an unfinished product to the first assembly line it takes  $e_1$  units of time.
- $\bullet$  To bring an uninished product to the second assembly line it takes  $e_2$  units of time. Add We Chat powcoder
- To get a finished product from the first assembly line to the warehouse it takes  $x_1$  units of time;
- ullet To get a finished product from the second assembly line to the warehouse it takes  $x_2$  units.
- Task: Find a fastest way to assemble a product using both lines as necessary.



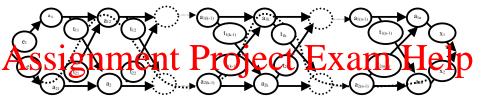
- For each  $k \le n$ , we solve subproblems P(1,k) and P(2,k) by a simultaneous recursion P(1,k) we solve subproblems P(1,k) and P(2,k) by a simultaneous recursion P(1,k) and P(2,k) by a simultaneous P(1,k) by a sim
- P(1,k): find the minimal amount of time  $\overline{m}(1,k)$  needed to finish the first k jobs, such the  $k^{th}$  job is finished on the  $k^{th}$  workstation on the **first** assembly line;
- P(2,k): find the minimal amount of time m(2,k) needed to finish the first k jobs, such the  $k^{th}$  job is finished on the  $k^{th}$  workstation on the **second** assembly line.



- For each  $k \le n$ , we solve subproblems P(1,k) and P(2,k) by a simultaneous recursor P(1,k) by a simultaneous P(1,k) by
- P(1,k): find the minimal amount of time  $\overline{m}(1,k)$  needed to finish the first k jobs, such the  $k^{th}$  job is finished on the  $k^{th}$  workstation on the **first** assembly line;
- P(2,k): find the minimal amount of time m(2,k) needed to finish the first k jobs, such the  $k^{th}$  job is finished on the  $k^{th}$  workstation on the **second** assembly line.



- For each  $k \le n$ , we solve subproblems P(1,k) and P(2,k) by a simultaneous recursor P(1,k) by a simultaneous P(1,k) by
- P(1,k): find the minimal amount of time  $\overline{m}(1,k)$  needed to finish the first k jobs, such the  $k^{th}$  job is finished on the  $k^{th}$  workstation on the **first** assembly line;
- P(2,k): find the minimal amount of time m(2,k) needed to finish the first k jobs, such the  $k^{th}$  job is finished on the  $k^{th}$  workstation on the **second** assembly line.



- · We shttps://powicoder.com
- Recursion:

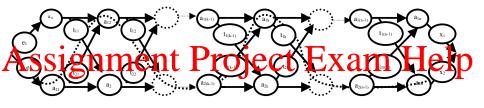
 $m(1,k) = \min\{m(1,k-1) + a_{1,k}, m(2,k-1) + t_{2,k-1} + a_{1,k}\}\$ 

## Add WeChat powcoder

• Finally, after obtaining m(1, n) and m(2, n) we choose

$$opt = \min\{m(1,n) + x_1, m(2,n) + x_2\}.$$

• This problem is important because it has the same design logic as the Viterbi algorithm, an extremely important algorithm for many fields such as speech recognition, decoding convolutional codes in telecommunications et a, covered a communications et a, covered a covered



- We state (15) Sand H (2) Can wind a simulation of the control of t
- Recursion:

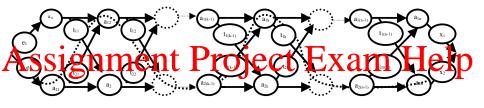
 $m(1,k) = \min\{m(1,k-1) + a_{1,k}, m(2,k-1) + t_{2,k-1} + a_{1,k}\}$ 

## Add WeChat powcoder

• Finally, after obtaining m(1, n) and m(2, n) we choose

$$opt = min\{m(1, n) + x_1, m(2, n) + x_2\}.$$

• This problem is important because it has the same design logic as the Viterbi algorithm, an extremely important algorithm for many fields such as speech recognition, decoding convolutional codes in telecommunications et a, covered a communications et a, covered a communications et a, covered a communications et a, covered a covered



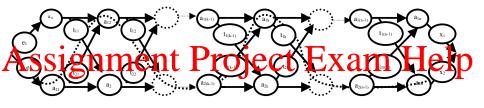
- We state (15) Sand H (2) Can wind a simulation of the control of t
- Recursion:

$$\begin{array}{c} m(1,k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) = \min\{m(1,k-1) + a_{1,k}\} \\ \textbf{A}^{m}(2k) =$$

• Finally, after obtaining m(1, n) and m(2, n) we choose

$$opt = min\{m(1, n) + x_1, m(2, n) + x_2\}.$$

This problem is important because it has the same design logic as the Viterbi algorithm, an extremely important algorithm for many fields such as speech recognition, decoding convolutional codes in telecommunications ets.



- We start (15) Sand H (2) Can wind a representation of the limitally the start of the limitally the limitally the start of the limitally the limitally the start of the limitally the start of the limitally the start of the limitally the limitally the start of the limitally the limita
- Recursion:

$$\begin{array}{c} m(1,k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{n(2)} & \text{with } 2 \textbf{C}^{1} + \text{at} & \textbf{p}^{(1)} \textbf{k} \textbf{V} \textbf{C}^{1} \textbf{C}^{1} \textbf{C}^{1} \\ \end{array}$$

• Finally, after obtaining m(1,n) and m(2,n) we choose

$$opt = min\{m(1, n) + x_1, m(2, n) + x_2\}.$$

This problem is important because it has the same design logic as the Viterbi algorithm, an extremely important algorithm for many fields such as speech recognition, decoding convolutional codes in telecommunications ets.



- We start (15) Sand H (2) Can wind a representation of the limitally the start of the limitally the limitally the start of the limitally the limitally the start of the limitally the start of the limitally the start of the limitally the limitally the start of the limitally the limita
- Recursion:

$$\begin{array}{c} m(1,k) = \min\{m(1,k-1) + a_{1,k}, & m(2,k-1) + t_{2,k-1} + a_{1,k}\} \\ \textbf{A}^{n(2)} & \text{with } 2 \textbf{C}^{1} + \text{at} & \textbf{p}^{(1)} \textbf{k} \textbf{V} \textbf{C}^{1} \textbf{C}^{1} \textbf{C}^{1} \\ \end{array}$$

• Finally, after obtaining m(1,n) and m(2,n) we choose

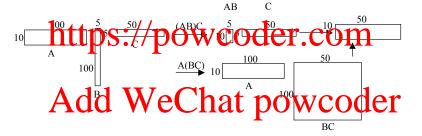
$$opt = min\{m(1, n) + x_1, m(2, n) + x_2\}.$$

• This problem is important because it has the same design logic as the Viterbi algorithm, an extremely important algorithm for many fields such as speech recognition, decoding convolutional codes in telecommunications etc, covered a c

#### Dynamic Programming: Matrix chain multiplication

- For any three matrices of compatible sizes we have A(BC) = (AB)C.
- However, the number of real number multiplications needed to perform in

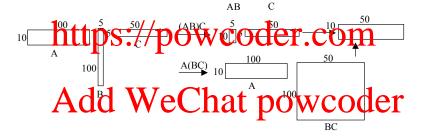
# Assignment Project Exam Help



- To evaluate (AB)C we need  $(10 \times 5) \times 100 + (10 \times 50) \times 5 = 5000 + 2500 = 7500$  multiplications;
- To evaluate A(BC) we need  $(100 \times 50) \times 5 + (10 \times 50) \times 100 = 25000 + 50000 = 75000$  multiplications!

- For any three matrices of compatible sizes we have A(BC) = (AB)C.
- However, the number of real number multiplications needed to perform in order to obtain the matrix induct can be very liferent:

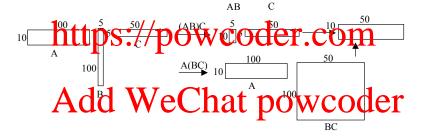
  ASSI SUMMENT = 1000 JECC Exam Help



- To evaluate (AB)C we need  $(10 \times 5) \times 100 + (10 \times 50) \times 5 = 5000 + 2500 = 7500$  multiplications;
- To evaluate A(BC) we need  $(100 \times 50) \times 5 + (10 \times 50) \times 100 = 25000 + 50000 = 75000$  multiplications!

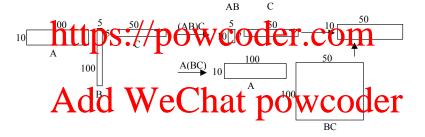
- For any three matrices of compatible sizes we have A(BC) = (AB)C.
- However, the number of real number multiplications needed to perform in order to obtain the matrix induct can be very liferent:

  ASSI SUMMENT = 1000 JECC Exam Help



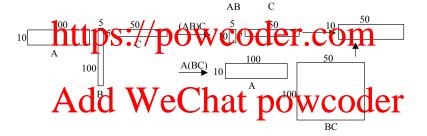
- To evaluate (AB)C we need  $(10 \times 5) \times 100 + (10 \times 50) \times 5 = 5000 + 2500 = 7500$  multiplications;
- To evaluate A(BC) we need  $(100 \times 50) \times 5 + (10 \times 50) \times 100 = 25000 + 50000 = 75000$  multiplications!

- For any three matrices of compatible sizes we have A(BC) = (AB)C.
- However, the number of real number multiplications needed to perform in order to obtain the matrix polyce can be very the rent: Help



- To evaluate (AB)C we need  $(10 \times 5) \times 100 + (10 \times 50) \times 5 = 5000 + 2500 = 7500$  multiplications;
- To evaluate A(BC) we need  $(100 \times 50) \times 5 + (10 \times 50) \times 100 = 25000 + 50000 = 75000$  multiplications!

- For any three matrices of compatible sizes we have A(BC) = (AB)C.
- However, the number of real number multiplications needed to perform in order to obtain the matrix polycycle of Exam Help



- To evaluate (AB)C we need  $(10 \times 5) \times 100 + (10 \times 50) \times 5 = 5000 + 2500 = 7500$  multiplications;
- To evaluate A(BC) we need  $(100 \times 50) \times 5 + (10 \times 50) \times 100 = 25000 + 50000 = 75000$  multiplications!

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignment Project Exam Help

• The total number of different distributions of brackets is equal to the number of bi https://powcoder.com
The total member of different distributions of brackets satisfies the following

# Add WeChat powcoder

- One can show that the solution satisfies  $T(n) = \Omega(2^n)$ .
- Thus, we cannot do an exhaustive search for the optimal placement of the

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### As take the product matrix.

• The total number of different distributions of brackets is equal to the number of bi https://powcoder.com
The total member of different distributions of brackets satisfies the following

# Add WeChat powcoder

- One can show that the solution satisfies  $T(n) = \Omega(2^n)$ .
- Thus, we cannot do an exhaustive search for the optimal placement of the

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### As take number of the Proping in the Xtamper Help

• The total number of different distributions of brackets is equal to the number of birary trees with  $\eta$  leaves.

The total number of different distributions of brackets satisfies the following

# Add WeChat powcoder

- One can show that the solution satisfies  $T(n) = \Omega(2^n)$ .
- Thus, we cannot do an exhaustive search for the optimal placement of the

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignment hProjections in Educations in the decretation of the product matrix.

• The total number of different distributions of brackets is equal to the number of binary trees with pleaves.

\*\*Dowcoder.com\*\*

| The total number of different distributions of brackets is equal to the number of binary trees with pleaves.

• The total number of different distributions of brackets satisfies the following recursion (why?):

# Add Wethat powcoder

- One can show that the solution satisfies  $T(n) = \Omega(2^n)$ .
- Thus, we cannot do an exhaustive search for the optimal placement of the brackets.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### As take number of the product matrix. As the product matrix.

• The total number of different distributions of brackets is equal to the number of binary trees with n leaves.

\*\*Dowcoder.com\*\*

| The total number of different distributions of brackets is equal to the number of binary trees with n leaves.

• The total number of different distributions of brackets satisfies the following recursion (why?):

## Add We'Chat'powcoder

- One can show that the solution satisfies  $T(n) = \Omega(2^n)$ .
- Thus, we cannot do an exhaustive search for the optimal placement of the brackets.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### As take number of the product matrix. As the product matrix is the product matrix.

• The total number of different distributions of brackets is equal to the number of binary trees with pleaves.

\*\*Dowcoder.com\*\*

| The total number of different distributions of brackets is equal to the number of binary trees with pleaves.

• The total number of different distributions of brackets satisfies the following recursion (why?):

### Add We'Chat' powcoder

- One can show that the solution satisfies  $T(n) = \Omega(2^n)$ .
- Thus, we cannot do an exhaustive search for the optimal placement of the brackets.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignment Project Exam Help

• The subproblems P(i, j) to be considered are:

"ground type://powcoder.com

the total number of relatiplication needed to find the product matrix".

- Note: this looks like it is a case of a "2D recursion, but we can actually do it with Add WeChat powcoder
- We group such subproblems by the value of j-i and perform a recursion on the value of j-i.
- At each recursive step m we solve all subproblems P(i, j) for which j i = m.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignmentch Presidentise by Xumer brelp

• The subproblems P(i,j) to be considered are:

"ground types: //powcoder.com
the total number of multiplication needed to find the product matrix".

- Note: this looks like it is a case of a "2D recursion, but we can actually do it with Add WeChat powcoder
- We group such subproblems by the value of j-i and perform a recursion on the value of j-i.
- At each recursive step m we solve all subproblems P(i,j) for which j-i=m.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Scale properties of the product matrix.

• The subproblems P(i, j) to be considered are:

"grohttps://powcoder.com
the total number of inlitiplication needed to find the product matrix".

- Note: this looks like it is a case of a "2D recursion, but we can actually do it with Add WeChat powcoder
- We group such subproblems by the value of j-i and perform a recursion on
- At each recursive step m we solve all subproblems P(i,j) for which i-i=m.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignmentch Pyrajentise Extermer Help

- The subproblems P(i,j) to be considered are:

  "ground to  $S_{i,A_{i,J}}$  DOWn Such  $C_{i,A_{i,J}}$  to  $C_{i,A_{i,J}}$  to
- Note: this looks like it is a case of a "2D recursion, but we can actually do it with Add WeChat powcoder
- We group such subproblems by the value of j-i and perform a recursion on the value of j-i.
- At each recursive step m we solve all subproblems P(i, j) for which j i = m.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignmentch Presidentise by Xtamer Flelp

- The subproblems P(i,j) to be considered are:

  "ground to  $S_{i,A_{i,J}}$  DOWn Such  $C_{i,A_{i,J}}$  to  $C_{i,A_{i,J}}$  to
- Note: this looks like it is a case of a "2D recursion, but we can actually do it with Aimple interview for at powcoder
- We group such subproblems by the value of j-i and perform a recursion on the value of j-i.
- At each recursive step m we solve all subproblems P(i,j) for which j-i=m.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignmentch Projective Extermer Help

- The subproblems P(i,j) to be considered are:

  "ground to  $S_{i,A_{i,J}}$  DOWn Such  $C_{i,A_{i,J}}$  to  $C_{i,A_{i,J}}$  to
- Note: this looks like it is a case of a "2D recursion, but we can actually do it with Aimple interview for at powcoder
- We group such subproblems by the value of j-i and perform a recursion on the value of j-i.
- At each recursive step m we solve all subproblems P(i,j) for which j-i=m.

• Problem Instance: A sequence of matrices  $A_1A_2...A_n$ ;

### Assignmentch Price jeantise by Xtamer Help

- The subproblems P(i,j) to be considered are:

  "ground to  $S_{i,A_{i,J}}$  DOWn Such  $C_{i,A_{i,J}}$  to  $C_{i,A_{i,J}}$  to
- Note: this looks like it is a case of a "2D recursion, but we can actually do it with Aimple interview for at powcoder
- We group such subproblems by the value of j-i and perform a recursion on the value of j-i.
- At each recursive step m we solve all subproblems P(i,j) for which j-i=m.

• Let m(i,j) denote the minimal number of multiplications needed to compute the product  $A_iA_{i+1}...A_{j-1}A_j$ ; let also the size of matrix  $A_i$  be  $s_{i-1} \times s_i$ .

### Assignment Project Exam Help

- Note that both k-i < j-i and j-(k+1) < j-i; thus we have the solutions of the subproblems P(i,k) and P(k+1,j) already computed and stored in slots k-i and j-(1+1) respectively, which procede slot j-1 we are presently filling.
- Note also that the matrix roduct  $A_i \dots A_k$  is a  $s_{i-1} \times s_k$  matrix L and  $A_{k+1} \dots A_k$  is a  $s_k \times s_i$  matrix R.
- Add WeChat powcoder

$$S_{i-1} = S_{i-1}$$

Total number of multiplications: S<sub>i-1</sub> S<sub>j</sub> S<sub>k</sub>

• Let m(i,j) denote the minimal number of multiplications needed to compute the product  $A_iA_{i+1}...A_{j-1}A_j$ ; let also the size of matrix  $A_i$  be  $s_{i-1} \times s_i$ .

## A Souli Scattor, in the chair not be due to principal (outernost elp

- Note that both k-i < j-i and j-(k+1) < j-i; thus we have the solutions of the subproblems P(i,k) and P(k+1,j) already computed and stored in slots k-i and j-(1+1) respectively. Which precede slot j-1 we are presently filling.
- Note also that the matrix product  $A_i \dots A_k$  is a  $s_{i-1} \times s_k$  matrix L and  $A_{k+1} \dots A_j$  is a  $s_k \times s_i$  matrix R.
- Add WeChat poweoder

$$S_{i-1} = S_{i-1}$$

Total number of multiplications: Si-1 Sj Sk

• Let m(i,j) denote the minimal number of multiplications needed to compute the product  $A_iA_{i+1}...A_{j-1}A_j$ ; let also the size of matrix  $A_i$  be  $s_{i-1} \times s_i$ .

## A S Subjection, in turn the chain in the order (4... X, 2.114...4, 1...4)

- Note that both k-i < j-i and j-(k+1) < j-i; thus we have the solutions of the subproblems P(i,k) and P(k+1,j) already computed and stored in slots k-i and j-(k+1) respectively, which precede slot j-1 we are presently filling.
- Note also that the matrix product  $A_i \dots A_k$  is a  $s_{i-1} \times s_k$  matrix L and  $A_{k+1} \dots A_j$  is a  $s_k \times s_k$  matrix R.
- Add WeChat poweoder

$$S_{i-1} = S_{i-1}$$

Total number of multiplications: S<sub>i-1</sub> S<sub>j</sub> S<sub>k</sub>

• Let m(i,j) denote the minimal number of multiplications needed to compute the product  $A_iA_{i+1}...A_{j-1}A_j$ ; let also the size of matrix  $A_i$  be  $s_{i-1} \times s_i$ .

## A S Subjection, in turn the chain in the order (4... X, 2.114...4, 1...4)

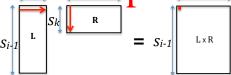
- Note that both k-i < j-i and j-(k+1) < j-i; thus we have the solutions of the subproblems P(i,k) and P(k+1,j) already computed and stored in slots k-i and j-(k+1), respectively, which precede slot j-1 we are presently filling.
- Note also that the matrix product  $A_i \dots A_k$  is a  $s_{i-1} \times s_k$  matrix L and  $A_{k+1} \dots A_j$  is a  $s_k \times s_j$  matrix R.
- To make the contract of the c

Total number of multiplications: S<sub>i-1</sub> S<sub>i</sub> S<sub>k</sub>

• Let m(i,j) denote the minimal number of multiplications needed to compute the product  $A_iA_{i+1}...A_{j-1}A_j$ ; let also the size of matrix  $A_i$  be  $s_{i-1}\times s_i$ .

## 

- Note that both k-i < j-i and j-(k+1) < j-i; thus we have the solutions of the subproblems P(i,k) and P(k+1,j) already computed and stored in slots k-i and j-(k+1), respectively, which precede slot j-1 we are presently filling.
- Note also that the matrix product  $A_i \dots A_k$  is a  $s_{i-1} \times s_k$  matrix L and  $A_{k+1} \dots A_j$  is a  $s_k \times s_j$  matrix R.
- To multiply in  $s_{-1}$  we matrix find an  $s_k \times s_j$  matrix R it takes  $s_{i-1}s_ks_j$  many multiplicated at  $S_i = S_i + S_j = S_i + S_i + S_j = S_i + S_i$



Total number of multiplications: S<sub>i-1</sub> S<sub>j</sub> S<sub>k</sub>

Assignment, Project, Exam Help

- algohttps://powcoder.com
- k for which the minimum in the recursive definition of m(i,j) is achieved can Add WeChat powcoder whole chai
- Thus, in the  $m^{th}$  slot of the table we are constructing we store all pairs (m(i,j),k) for which j-i=m.

Assignment, Project, Exam Help

- Note that the recursion step is a brute force search but the whole algorithm is 0.5 because all Wy up on thems are solved only once, and there are only  $O(n^2)$  many such subproblems.
- k for which the minimum in the recursive definition of m(i,j) is achieved can Add WeChat powcoder whole chailed MeChat powcoder
- Thus, in the  $m^{th}$  slot of the table we are constructing we store all pairs (m(i,j),k) for which j-i=m.

- Note that the recursion step is a brute force search but the whole algorithm is  $O(n^2)$  because all the subproblems are so viding once, and there are only  $O(n^2)$  many such subproblems.
- k for which the minimum in the recursive definition of m(i,j) is achieved can be stored to retrieve the obtain applicament of brackets for the whole chair A.
- Thus, in the  $m^{th}$  slot of the table we are constructing we store all pairs (m(i,j),k) for which j-i=m.

Assignment, Project, Exam Help

- Note that the recursion step is a brute force search but the whole algorithm is  $O(n^2)$  because all the subproblems are so viding once, and there are only  $O(n^2)$  many such subproblems.
- Thus, in the  $m^{th}$  slot of the table we are constructing we store all pairs (m(i,j),k) for which j-i=m.

• Assume we want to compare how similar two sequences of symbols S and  $S^*$  are.

# Assignment Project Exam Help

• Example: how similar are the genetic codes of two viruses.

- Thi https://powcoder.com other.
- A sequence s is a subsequence of another sequence S if s can be the Add WeChat powcoder
- Given two sequences S and  $S^*$  a sequence s is a Longest

• Assume we want to compare how similar two sequences of symbols S and  $S^*$  are.

- Thi https://powcoder.com other.
- A sequence s is a subsequence of another sequence S if s can be the Add WeChat powcoder
- Given two sequences S and  $S^*$  a sequence s is a **Longest**

• Assume we want to compare how similar two sequences of symbols S and  $S^*$  are.

- This part this go if on pious a continent to of the other.
- A sequence s is a subsequence of another sequence S if s can be the Add WeChat powcoder
- Given two sequences S and  $S^*$  a sequence s is a **Longest**

• Assume we want to compare how similar two sequences of symbols S and  $S^*$  are.

- This part this go if on pious a continent to of the other.
- A sequence s is a subsequence of another sequence S if s can be obtained by deleting some of the symbols of S (while preserving the gracely the remarks symbols). DOWCOUCT
- Given two sequences S and  $S^*$  a sequence s is a Longest

• Assume we want to compare how similar two sequences of symbols S and  $S^*$  are.

- This part this go if on pious a continent to of the other.
- A sequence s is a subsequence of another sequence S if s can be obtained by deleting some of the symbols of S (while preserving the grace of the remarks symbols). DOWCOUCT
- Given two sequences S and  $S^*$  a sequence s is a **Longest** Common Subsequence of  $S, S^*$  if s is a common subsequence of both S and  $S^*$  and is of maximal possible length.

• Instance: Two sequences  $S = \langle a_1, a_2, \dots a_n \rangle$  and  $S^* = \langle b_1, b_2, \dots, b_m \rangle$ .

- We first find the length of the longest common subsequence of  $S, S^*$
- "2D https://powcoder.com the length of  $S_i = \langle a_1, a_2, \dots a_i \rangle$  and  $S_j = \langle b_1, b_2, \dots, b_j \rangle$ .
- Recursion: we fill the table production, so the ordering of subproblems is the lexic Add we Chat powcoder

$$c[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0; \\ c[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } a_i = b, \\ \max\{c[i-1,j],c[i,j-1]\} & \text{if } i,j > 0 \text{ and } a_i \neq b, \end{cases}$$

• Instance: Two sequences  $S = \langle a_1, a_2, \dots a_n \rangle$  and  $S^* = \langle b_1, b_2, \dots, b_m \rangle$ .

- We first find the length of the longest common subsequence of  $S, S^*$
- "2D https://powcoder.com the length of  $S_i = \langle a_1, a_2, \dots a_i \rangle$  and  $S_j = \langle b_1, b_2, \dots, b_j \rangle$ .
- Recursion: we fill that the product row, so the ordering of subproblems is the lexic Add we Chat powcoder

$$c[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0; \\ c[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } a_i = b; \\ \max\{c[i-1,j],c[i,j-1]\} & \text{if } i,j > 0 \text{ and } a_i \neq b; \end{cases}$$

• Instance: Two sequences  $S = \langle a_1, a_2, \dots a_n \rangle$  and  $S^* = \langle b_1, b_2, \dots, b_m \rangle$ .

- We first find the length of the longest common subsequence of  $S, S^*$ .
- "2D https://powcoder.com the length of  $S_i = \langle a_1, a_2, \dots a_i \rangle$  and  $S_j = \langle b_1, b_2, \dots, b_j \rangle$ .
- Recursion: The file that he row, so the ordering of subproblems is the lexic Add we Chat powcoder

$$c[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0; \\ c[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } a_i = b, \\ \max\{c[i-1,j],c[i,j-1]\} & \text{if } i,j > 0 \text{ and } a_i \neq b, \end{cases}$$

• Instance: Two sequences  $S = \langle a_1, a_2, \dots a_n \rangle$  and  $S^* = \langle b_1, b_2, \dots, b_m \rangle$ .

- We first find the length of the longest common subsequence of  $S, S^*$ .
- "2D leggester": for all/legistrandall of particles in the length of the languat common subspace of the threatest sequences  $S_i = \langle a_1, a_2, \dots a_i \rangle$  and  $S_j^* = \langle b_1, b_2, \dots, b_j \rangle$ .
- Recursion: we fill that the product row, so the ordering of subproblems is the lexic Add we Chat powcoder

$$c[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0; \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } a_i = b; \\ \max\{c[i-1,j],c[i,j-1]\} & \text{if } i,j > 0 \text{ and } a_i \neq b; \end{cases}$$

• Instance: Two sequences  $S = \langle a_1, a_2, \dots a_n \rangle$  and  $S^* = \langle b_1, b_2, \dots, b_m \rangle$ .

- We first find the length of the longest common subsequence of  $S, S^*$ .
- "2D leggester": for all/legistrandall legistrandall for plants of the language common subspace of the threatest sequences  $S_i = \langle a_1, a_2, \dots a_i \rangle$  and  $S_j^* = \langle b_1, b_2, \dots, b_j \rangle$ .
- Recursion: we fill the table receive row, so the ordering of subproblems is the lexicolder WeChat powcoder

$$c[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0; \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } a_i = b; \\ \max\{c[i-1,j],c[i,j-1]\} & \text{if } i,j > 0 \text{ and } a_i \neq b; \end{cases}$$

• Instance: Two sequences  $S = \langle a_1, a_2, \dots a_n \rangle$  and  $S^* = \langle b_1, b_2, \dots, b_m \rangle$ .

### Assignment-ProjectsExam Help

- We first find the length of the longest common subsequence of  $S, S^*$ .
- "2D legerster": for all/15 is wanted by the length of the largest common subspace of the threates separated  $S_i = \langle a_1, a_2, \dots a_i \rangle$  and  $S_j^* = \langle b_1, b_2, \dots, b_j \rangle$ .
- $\begin{array}{c} \bullet \ \ {\rm Recursion: \ we \ fill \ the \ table \ row \ hy \ row, \ so \ the \ ordering \ of \ subproblems \ is \ the } \\ {\rm lexicoruplic \ Order \ } \end{array}$

$$c[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0; \\ c[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } a_i = b_j; \\ \max\{c[i-1,j], c[i,j-1]\} & \text{if } i,j > 0 \text{ and } a_i \neq b_j. \end{cases}$$

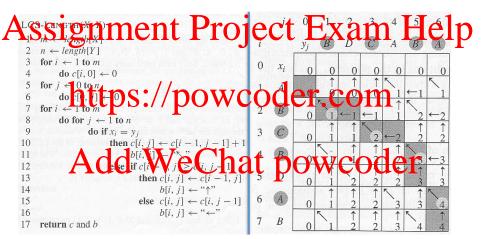
• Instance: Two sequences  $S = \langle a_1, a_2, \dots a_n \rangle$  and  $S^* = \langle b_1, b_2, \dots, b_m \rangle$ .

### Assignment-ProjectsExam Help

- We first find the length of the longest common subsequence of  $S, S^*$ .
- "2D leggester": for all/legistrandall of particles in the length of the languat common subspace of the threatest sequences  $S_i = \langle a_1, a_2, \dots a_i \rangle$  and  $S_j^* = \langle b_1, b_2, \dots, b_j \rangle$ .
- $\begin{array}{c} \bullet \ \ {\rm Recursion: \ we \ fill \ the \ table \ row \ hy \ row, \ so \ the \ ordering \ of \ subproblems \ is \ the } \\ {\rm lexicoruplic \ Order \ } \end{array}$

$$c[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0; \\ c[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } a_i = b_j; \\ \max\{c[i-1,j], c[i,j-1]\} & \text{if } i,j > 0 \text{ and } a_i \neq b_j. \end{cases}$$

Retrieving a longest common subsequence:



• What if we have to find a longest common subsequence of three sequences  $S_1, S_2, S_3$ ?

### Assignment Project Exam Help

 $S_1 = ABCDEGG$ 

 $LCS(S_1, S_2) = ABEC$ 

### https://powcoder.com

 $LCS(LCS(S_1, S_2), S_3) = LCS(ABEG, ACCEDGF) = AEG$ 

### Add WeChat powcoder

But

$$LCS(S_1, S_2, S_3) = ACEG$$

• So how would you design an algorithm which computes correctly  $LCS(S_1, S_2, S_3)$ ?

• What if we have to find a longest common subsequence of three sequences  $S_1, S_2, S_3$ ?

## Assignment Project Exam Help

 $S_1 = ABCDEGG$ 

 $LCS(S_1, S_2) = ABEG$ 

### https://powcoder.com

 $LCS(LCS(S_1, S_2), S_3) = LCS(ABEG, ACCEDGF) = AEG$ 

### Add WeChat powcoder

But

$$LCS(S_1, S_2, S_3) = ACEG$$

 So how would you design an algorithm which computes correctly LCS(S<sub>1</sub>, S<sub>2</sub>, S<sub>3</sub>)?

• What if we have to find a longest common subsequence of three sequences  $S_1, S_2, S_3$ ?

### Assignment Project Exam Help

$$S_{1} = ABCDEGG \qquad LCS(S_{1}, S_{2}) = ABEG$$

$$LCS(S_{1}, S_{3}) = ACEF$$

$$LCS(LCS(S_{1}, S_{2}), S_{3}) = LCS(ABEG, ACCEDGF) = AEG$$

$$LCS(LCS(S_{2}, S_{3}), S_{1}) = LCS(ACEF, ABCDEGG) = ACE$$

$$LCS(LCS(S_{2}, S_{3}), S_{2}) = LCS(ACEF, ABCDEGG) = ACE$$

$$LCS(LCS(S_{3}, S_{3}), S_{3}) = LCS(ACEF, ABCDEGG) = ACE$$

$$LCS(LCS(S_{3}, S_{3}), S_{3}) = LCS(ACEF, ABCDEGG) = ACE$$

But

$$LCS(S_1, S_2, S_3) = ACEG$$

 So how would you design an algorithm which computes correctly LCS(S<sub>1</sub>, S<sub>2</sub>, S<sub>3</sub>)?

• What if we have to find a longest common subsequence of three sequences  $S_1, S_2, S_3$ ?

## Assignment Project Exam Help

$$S_{1} = ABCDEGG \qquad LCS(S_{1}, S_{2}) = ABEG$$

$$S_{2} = ACBFFFG \qquad LCS(S_{1}, S_{3}) = ACEF$$

$$LCS(LCS(S_{1}, S_{2}), S_{3}) = LCS(ABEG, ACCEDGF) = AEG$$

$$LCS(LCS(S_{1}, S_{2}), S_{3}) = LCS(ACEF, ABCDEGG) = ACE$$

$$LCS(LCS(S_{2}, S_{3}), S_{1}) = LCS(ACEF, ABCDEGG) = ACE$$

$$LCS(LCS(S_{1}, S_{2}), S_{3}) = LCS(ACEF, ABCDEGG) = ACE$$

$$LCS(LCS(S_{2}, S_{3}), S_{3}) = LCS(ACEF, ABCDEGG) = ACE$$

But

$$LCS(S_1, S_2, S_3) = ACEG$$

• So how would you design an algorithm which computes correctly  $LCS(S_1, S_2, S_3)$ ?

• Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

Assignment Project Exam Help

- $\bullet$  We again first find the length of the longest common subsequence of  $S,S^*,S^{**}.$
- for a https://powcoder.com ngth of the longest common subsequence of the truncated sequences  $S_i = \langle a_1, a_2, \dots a_i \rangle$ ,  $S_i^* = \langle b_1, b_2, \dots, b_j \rangle$  and  $S_j^{**} = \langle c_1, c_2, \dots, c_l \rangle$ .
- RecurAdd WeChat powcoder

$$d[i,j,l] = \begin{cases} 0, \\ d[i-1,j-1,l-1] + 1 \\ \max\{d[i-1,j,l],d[i,j-1,l],d[i,j,l-1]\} \end{cases}$$



• Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

• Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

• Signment Project Exam Help

- $\bullet$  We again first find the length of the longest common subsequence of  $S,S^*,S^{**}.$
- for a https://powcoder.com and of the longest common subsequence of the truncated sequences  $S_i = \langle a_1, a_2, \dots a_i \rangle$ ,  $S_i^* = \langle b_1, b_2, \dots, b_j \rangle$  and  $S_l^{**} = \langle c_1, c_2, \dots, c_l \rangle$ .
- RecurAdd WeChat powcoder

$$d[i,j,l] = \begin{cases} 0, & \text{if } i = l\\ d[i-1,j-1,l-1] + 1 & \text{if } i,j\\ \max\{d[i-1,j,l],d[i,j-1,l],d[i,j,l-1]\} & \text{other} \end{cases}$$



- Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

   Signment Project Exam Help
  - We again first find the length of the longest common subsequence of  $S, S^*, S^{**}$ .
  - for a https://powcoder.com and of the longest common subsequence of the truncated sequences  $S_i = \langle a_1, a_2, \dots a_i \rangle$ ,  $S_j^* = \langle b_1, b_2, \dots, b_j \rangle$  and  $S_l^{**} = \langle c_1, c_2, \dots, c_l \rangle$ .
  - RecuAdd WeChat powcoder

$$d[i,j,l] = \begin{cases} 0, \\ d[i-1,j-1,l-1] + 1 \\ \max\{d[i-1,j,l],d[i,j-1,l],d[i,j,l-1]\} \end{cases}$$



• Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

• Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

• Task Find a longest common subsequence of  $S, S^*, S^*$ .

- We again first find the length of the longest common subsequence of  $S, S^*, S^{**}$ .
- for an ttps://tpo.wdcqder[i,c] Qmength of the longest common subsequence of the truncated sequences  $S_i = \langle a_1, a_2, \dots a_i \rangle$ ,  $S_j^* = \langle b_1, b_2, \dots, b_j \rangle$  and  $S_l^{**} = \langle c_1, c_2, \dots, c_l \rangle$ .
- RecuAdd WeChat powcoder

$$d[i,j,l] = \begin{cases} 0, \\ d[i-1,j-1,l-1] + 1 \\ \max\{d[i-1,j,l],d[i,j-1,l],d[i,j,l-1]\} \end{cases}$$



- Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

   Instance: Three sequences  $S = \langle a_1, a_2, \dots a_n \rangle$ ,  $S^* = \langle b_1, b_2, \dots, b_m \rangle$  and  $S^{**} = \langle c_1, c_2, \dots, c_k \rangle$ .

   Task Find a longest common subsequence of  $S, S^*, S^*$ .
  - We again first find the length of the longest common subsequence of  $S, S^*, S^{**}$ .
  - for an trps.  $A_i = A_i + A$
  - RecurAndd WeChat powcoder

$$d[i,j,l] = \begin{cases} 0, \\ d[i-1,j-1,l-1] + 1 \\ \max\{d[i-1,j,l],d[i,j-1,l],d[i,j,l-1]\} \end{cases}$$

if i = 0 or j = 0 or l = 0; if i, j, l > 0 and  $a_i = b_j = c_l$ ; otherwise.

◆□▶ ◆□▶ ◆■▶ ◆■▶ ● りへで

#### Dynamic Programming: Shortest Common Supersequence

# Assignment Project Exam Help Task ind a shortest common super sequence S of s, $s^*$ , i.e., the shortest prossible sequence S such that both s and $s^*$ are subsequences of S

• Soluhttps://powcoder.com/s and s and order; for example:

## Add WeChat powcoder

shortest super-sequence S = axbyacazda

#### Dynamic Programming: Shortest Common Supersequence

## $\begin{array}{c} \textbf{A} \overset{\textbf{Instance: Two sequences}}{\textbf{E}} \overset{\textbf{Project}}{\textbf{E}} \overset{\textbf{E}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}}} \overset{\textbf{A}}{\textbf{E}} \overset{\textbf{A}}{\textbf{E}$

- Task-Find a shortest common super-sequence S of  $s, s^*$ , i.e., the shortest possible sequence S such that both s and  $s^*$  are subsequences of S.
- Soluhttps://powcoder.com/s and s and order; for example:

## Add WeChat powcoder

shortest super-sequence S = axbyacazda

#### Dynamic Programming: Shortest Common Supersequence

## $\begin{array}{c} \textbf{A} \overset{\bullet}{\text{\textbf{SS1gnment}}} \overset{\bullet}{\text{\textbf{Project}}} \overset{\bullet}{\text{\textbf{Exam}}} \overset{\bullet}{\text{\textbf{Help}}} \\ \overset{\bullet}{\text{\textbf{Task-Find a shortest common super-sequence}} \overset{\bullet}{\text{\textbf{Sof }}} \overset{\bullet}{\text{\textbf{Sof }}} \overset{\bullet}{\text{\textbf{Soft}}} \overset{\bullet}{\text{\textbf{Exam}}} \overset{\bullet}{\text{\textbf{Help}}} \\ \end{array}$

- Task: Find a shortest common super-sequence S of  $s, s^*$ , i.e., the shortest possible sequence S such that both s and  $s^*$  are subsequences of S.
- Solution: Find the largest common subsequence  $LGS(s,s^*)$  of s and  $s^*$  and then add differing elements of the two sequences at the right places, in any order; for example:

## Add WeChat poweoder

shortest super-sequence S = axbyacazda

• Edit Distance Given two text strings A of length n and B of length m, you want to transform A into B. You are allowed to insert a character, delete a character and to replace a character with another long. An insertion costs of the latest tent tept center outs of the latest tent tept center outs of the latest tent tept center outs of the latest tent tenter outs of the latest tenter of the latest tenter

- Task: find the lowest total cost transformation of A into B.
- Note https://powcoder.com the minimal number of spiroperations required to transform A into B, this number is called the edit distance between A and B.
- If the Add ar Wee Chat op Owcode represents the probability that one sequence mutates to another sequence in the course of DNA copying.
- Subproblems: Let C(i,j) be the minimum cost of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

• Edit Distance Given two text strings A of length n and B of length m, you want to transform A into B. You are allowed to insert a character, delete a character and to replace a character with another lone. An insertion costs of a character with another lone. An insertion costs of a character with another lone. An insertion costs of a character with another lone.

- Task: find the lowest total cost transformation of A into B.
- Note https://powcoder-com the minimal number of spiroperations required to transform A into B; this number is called the edit distance between A and B.
- If the Add ar Wee Chat are probability that one sequence mutates to another sequence in the course of DNA copying.
- Subproblems: Let C(i,j) be the minimum cost of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

• Edit Distance Given two text strings A of length n and B of length m, you want to transform A into B. You are allowed to insert a character, delete a character and to replace a character with another long. An insertion costs of policy and templacement outs of the control of the contro

- Task: find the lowest total cost transformation of A into B.
- Note if the priors have a live state of the minimal number of such operations required to transform A into B; this number is called the edit distance between A and B.
- o If the Add at WeeChatopowcoder presents the probability that one sequence mutates to another sequence in the course of DNA copying.
- Subproblems: Let C(i,j) be the minimum cost of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

• Edit Distance Given two text strings A of length n and B of length m, you want to transform A into B. You are allowed to insert a character, delete a character and to replace a character with another one. An insertion costs of Schletin rests n and replacement outs n.

- Task: find the lowest total cost transformation of A into B.
- Note if all dorsions have a highest then yet fire cooking for the minimal number of such operations required to transform A into B; this number is called the edit distance between A and B.
- If the Aque ices are sequences of DNA bases and the costs reflect the probabilities of the corresponding mutations, then the minimal rest represents the probability that one sequence mutates into another sequence in the course of DNA copying.
- Subproblems: Let C(i,j) be the minimum cost of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

• Edit Distance Given two text strings A of length n and B of length m, you want to transform A into B. You are allowed to insert a character, delete a character and to replace a character with another one. An insertion casts of the character with another one. An insertion casts of the character with another one.

- Task: find the lowest total cost transformation of A into B.
- Note if all dorstions have a highest then you are cooking for the minimal number of such operations required to transform A into B; this number is called the edit distance between A and B.
- If the Aquelices are signed as of DNA hases and the costs reflect the probabilities of the object ding hatalions then We makin at the probability that one sequence mutates into another sequence in the course of DNA copying.
- Subproblems: Let C(i,j) be the minimum cost of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

• Subproblems P(i,j): Find the minimum cost C(i,j) of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

### Assignment Project Exam Help

 $\textbf{https://powcoder.com} \\ \textbf{https://powcoder.com} \\ \textbf{bttps://powcoder.com} \\ \textbf{bttps://powco$ 

- $\cos t c_D + C(i-1,j)$  corresponds to the option if you recursively transform
- cost Add (WeChat) powcoder [1..i] to B[1..j-1] and then append B[j] at the end,
- the third option corresponds to first transforming A[1..i-1] to B[1..j-1] and
  - ① if A[i] is already equal to B[j] do nothing, thus incurring a cost of only C(i-1, j-1);
  - ② if A[i] is not equal to B[j] replace A[i] by B[j] with a total cost of  $C(i-1,j-1)+c_R$ .

• Subproblems P(i,j): Find the minimum cost C(i,j) of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

### As Significant we again fill the able of solutions $E_{ij}$ for subproblem P(ij) for $E_{ij}$ for subproblem $E_{ij}$

$$\mathbf{https:}^{C(i-1,j)+c_D} \underbrace{\mathbf{C}(i,j-1)+c_I}_{C(i,j-1)+c_R} \mathbf{erif}_{A[i] \neq B[j]}^{C(i-1,j)+c_D}$$

- $\cos t c_D + C(i-1,j)$  corresponds to the option if you recursively transform
- $\operatorname{cost} Add$  [WeChat] powcoder [1..i] to B[1..j-1] and then append B[j] at the enc.
- the third option corresponds to first transforming A[1..i-1] to B[1..j-1] and
  - if A[i] is already equal to B[j] do nothing, thus incurring a cost of only C(i-1, j-1);
  - ② if A[i] is not equal to B[j] replace A[i] by B[j] with a total cost of  $C(i-1,j-1)+c_R$ .

• Subproblems P(i,j): Find the minimum cost C(i,j) of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

$$\begin{array}{c} \mathbf{https:} \\ \mathbf{https:}$$

- cost  $c_D + C(i-1,j)$  corresponds to the option if you recursively transform
- $A[1..iA^1]$  idea B[1W and then append B[j] at the end,
- the third option corresponds to first transforming A[1..i-1] to B[1..i-1] and

• Subproblems P(i,j): Find the minimum cost C(i,j) of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

$$\begin{array}{c} \mathbf{https:} \\ \mathbf{https:}$$

- cost  $c_D + C(i-1,j)$  corresponds to the option if you recursively transform
- $A[1..i \ A] \text{ i.i.t. } B[1] \text{ and then delete } A[i] \text{ oct } C[1] \text{ oct } A[1..i] \text{ to } B[1..j-1] \text{ and then append } B[j] \text{ at the end;}$
- the third option corresponds to first transforming A[1..i-1] to B[1..i-1] and

• Subproblems P(i,j): Find the minimum cost C(i,j) of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

$$\textbf{https://passage of the first of the fir$$

- cost  $c_D + C(i-1,j)$  corresponds to the option if you recursively transform
- $A[1..i \land 1]$  it to B[1] and then delete A[i]:
    $\cot C(1) + c_I$  corresponds to the option of yamilst C[1..i] to B[1..j-1] and then append B[j] at the end;
- the third option corresponds to first transforming A[1..i-1] to B[1..j-1] and

• Subproblems P(i,j): Find the minimum cost C(i,j) of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

$$\textbf{https://passessmall} \\ \textbf{https://passessmall} \\ \textbf{https://passessmall} \\ \textbf{Oweder} \\ \textbf{Oweder} \\ \textbf{Oweder} \\ \textbf{Oscillation} \\ \textbf{Oscillati$$

- cost  $c_D + C(i-1,j)$  corresponds to the option if you recursively transform
- $A[1..i \land 1]$  it to B[1] and then delete A[i]:
    $\cot C(1) + c_I$  corresponds to the option of yamilst C[1..i] to B[1..j-1] and then append B[j] at the end;
- the third option corresponds to first transforming A[1..i-1] to B[1..j-1] and
  - if A[i] is already equal to B[j] do nothing, thus incurring a cost of only C(i-1, j-1);

• Subproblems P(i,j): Find the minimum cost C(i,j) of transforming the sequence A[1..i] into the sequence B[1..j] for all  $i \leq n$  and all  $j \leq m$ .

$$\begin{array}{c} \mathbf{https:} \\ \mathbf{https:}$$

- cost  $c_D + C(i-1,j)$  corresponds to the option if you recursively transform
- $\begin{array}{c} A[1..i \ A] \text{ into } B[1] \text{ And then delete } 4[i] \\ \text{cost } C[1] \text{ corresponds of the opt of flywhist } G[1..i] \text{ to} \end{array}$ B[1..i-1] and then append B[j] at the end;
- the third option corresponds to first transforming A[1..i-1] to B[1..j-1] and
  - if A[i] is already equal to B[j] do nothing, thus incurring a cost of only C(i-1, j-1);
  - 2 if A[i] is not equal to B[j] replace A[i] by B[j] with a total cost of  $C(i-1, j-1) + c_R$ .

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Project Exam Help

- What will be the subproblems?
- May https://powcoder.com the resulting expression is maximised?
- maybe we could consider which the principal operations should be, i.e. A[i..k] Add WeChat powcoder 1.
- But when would such expression be maximised if there could be both positive and negative values for A[i..k] and A[k+1..j] depending on the placement of brackets??
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!
- Exercise: write the exact recursion for this problem.

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Phroject & Exame Help

- What will be the subproblems?
- May https://powcoder.com the resulting expression is maximised?
- maybe we could consider which the principal operations should be, i.e.  $\overset{A[i..k]}{A} \overset{Add}{d} \overset{WeChat powcoder}{}^{+1]}.$
- But when would such expression be maximised of there could be both positive and negative values for A[i...k] and A[k+1...j] depending on the placement of brackets?
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!
- Exercise: write the exact recursion for this problem.

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Project Parame Help

- What will be the subproblems?
- May https://powcoder.com the resulting expression is maximised?
- maybe we could consider which the principal operations should be, i.e. Add WeChat powcoder 1.
- But when would such expression be maximised of there could be both positive and negative values for A[i...k] and A[k+1...j] depending on the placement of brackets?
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!
- Exercise: write the exact recursion for this problem.

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Project Parame Help

- What will be the subproblems?
- May be the surface of the surface
- maybe we could consider which the principal operations should be, i.e.

  Add WeChat powcoder

  1.
- But when would such expression be maximised of there could be both positive and negative values for A[i...k] and A[k+1...j] depending on the placement of brackets?
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!
- Exercise: write the exact recursion for this problem.

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Project Parame Help

- What will be the subproblems?
- May be the surface of the surface
- maybe we could consider which the principal operations should be, i.e. A[i..k] A[k+1].j. Where O is what aver operation is between A[k] and A[k+1].
- But when would such expression be maximised of there could be both positive and negative values for A[i...k] and A[k+1...j] depending on the placement of brackets?
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!
- Exercise: write the exact recursion for this problem.

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Phroject Exame Help

- What will be the subproblems?
- May be the surface power of the the resulting expression is maximised?
- maybe we could consider which the principal operations should be, i.e. A[i..k] A[k+1] J. Here  $\odot$  is what ever operation is between A[k] and A[k+1].
- But when would such expression be maximised if there could be both positive and negative values for A[i..k] and A[k+1..j] depending on the placement of brackets??
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Phroject Exame Help

- What will be the subproblems?
- May be the surface power of the the resulting expression is maximised?
- maybe we could consider which the principal operations should be, i.e. A[i..k] A[k+1] J. Here  $\odot$  is what ever operation is between A[k] and A[k+1].
- But when would such expression be maximised if there could be both positive and negative values for A[i..k] and A[k+1..j] depending on the placement of brackets??
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!
- Exercise: write the exact recursion for this problem.

 $\bullet$  Instance: a sequence of numbers with operations  $+,-,\times$  in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

### Assignment Phroject Exame Help

- What will be the subproblems?
- May be the surface of the surface
- maybe we could consider which the principal operations should be, i.e. A[i..k] A[k+1] J. Here  $\odot$  is what ever operation is between A[k] and A[k+1].
- But when would such expression be maximised if there could be both positive and negative values for A[i..k] and A[k+1..j] depending on the placement of brackets??
- Maybe we should look for placements of brackets not only for the maximal value but also for the minimal value!
- Exercise: write the exact recursion for this problem.

#### Dynamic Programming: Turtle Tower

Sisterical Topic given n urticout the cach purticy of irraiver is we git and is strength. The strength of a turtle is the maximal weight you can put on it without cracking its shell.

- https://powcoder.com
- Add WeChat powcoder

#### Dynamic Programming: Turtle Tower

### istuce me in a fire, if the each until you in live risk and is strength. The strength of a turtle is the maximal weight you can on it without cracking its shell.

- Task Find the largest/possible number of tilrtles which you can stack one on top of the bear svithou of children well er com
- VeChat powcoder

### Dynamic Programming: Turtle Tower

### Sstarch market in the strength of a turtle is the maximal weight you can on it without cracking its shell.

- Task Find the largest/possible number of til rtles which you can stack one on top of the bear svithou of children well er com
- Order turtles in an increasing order of the sum of their weight and their strength, and proceed by recursion.
- eChat powcoder

#### Dynamic Programming: Turtle Tower

# As strength. The strength of a turtle is the maximal weight you can put on it without cracking its shell.

- Task Find the largest possible number of tilrtles which you can stack one on top of the bran without orchivator with the largest possible number of tilrtles which you can stack one on top of the bran without orchivator with the largest possible number of tilrtles which you can stack one on top of the branch without orchivator with the largest possible number of tilrtles which you can stack one on top of the branch without orchivator with the largest possible number of tilrtles which you can stack one on top of the branch without orchivator with the largest possible number of tilrtles which you can stack one on top of the branch without orchivator with the largest possible number of tilrtles which you can stack one on top of the branch without orchivator with the largest possible number of tilrtles which you can stack one on top of the branch without orchivator with the largest possible number of tilrtles which with the largest possible number of the branch with the largest possible number of the branch with the largest possible number of the largest possible number o
- **Hint:** Order turtles in an increasing order of the sum of their weight and their strength, and proceed by recursion.
- You can find a solution to this problem and of another interesting problem on the class website (class resources, file "More Dynamic Programming").

- One of the earliest use of Dynamic Programming (1950's) invented by Bellman.
- Instance: A directed weighted graph G = (V, E) with weights which can be

### Assignment Project Exam Help Goal: Find the shortest path from vertex s to every other vertex t.

- Solution: Since there are no negative weight cycles, the shortest path cannot cont <a href="https://powcoder.com">https://powcoder.com</a> path.
- Thus, every shortest path can have at most |V|-1 edges.
- the Add WeChat powcoder seems to be the Add we Chat powcoder seems to be the Add we c
- Our goal is to find for every vertex  $t \in G$  the value of opt(n-1,t) and the
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \dots, p_k, t)$  then

• One of the earliest use of Dynamic Programming (1950's) invented by Bellman.

### • Instance: A directed weighted graph G = (V, E) with weights which can be ASSIGNITED TO ECUL Weight and a vertex FGoal: Find the shortest path from vertex s to every other vertex t.

- Solution: Since there are no negative weight cycles, the shortest path cannot conthttps://powcoder.com
- Thus, every shortest path can have at most |V|-1 edges.
- the Add WeChat powcoder con the land of th
- Our goal is to find for every vertex  $t \in G$  the value of opt(n-1,t) and the
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \dots, p_k, t)$  then

 $\bullet$  One of the earliest use of Dynamic Programming (1950's) invented by Bellman.

# • Instance: A directed weighted graph G = (V, E) with weights which can be ASSIGNIFICATED Projective total weight and a vertex F

- Goal: Find the shortest path from vertex s to every other vertex t.
- Solution: Since there are no negative weight cycles, the shortest path cannot conthttps://powcoder.com
- Thus, every shortest path can have at most |V| 1 edges.
- \* Subpublic de We Chat powcoder es l'all powcode
- Our goal is to find for every vertex  $t \in G$  the value of  $\operatorname{opt}(n-1,t)$  and the path which achieves such a length.
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \ldots, p_k, t)$  then  $(p_1, p_2, \ldots, p_k, t)$  must be the shortest path from  $p_1$  to t, and  $(v, p_1, p_2, \ldots, p_k)$  must also be the shortest path from v to  $p_k$ .

 $\bullet\,$  One of the earliest use of Dynamic Programming (1950's) invented by Bellman.

# • Instance: A directed weighted graph G = (V, E) with weights which can be A specialty but without cycles progrative total weight and a vertex F to F and the shortest rath from vertex at a graph F and F are the program of the program o

- Goal: Find the shortest path from vertex s to every other vertex t.
- Solution: Since there are no negative weight cycles, the shortest path cannot contain tyles because by the coise a modern that er path.
- Thus, every shortest path can have at most |V| 1 edges.
- \* Subpublic de We Chat powcoder es l'all powcode
- Our goal is to find for every vertex  $t \in G$  the value of  $\operatorname{opt}(n-1,t)$  and the path which achieves such a length.
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \ldots, p_k, t)$  then  $(p_1, p_2, \ldots, p_k, t)$  must be the shortest path from  $p_1$  to t, and  $(v, p_1, p_2, \ldots, p_k)$  must also be the shortest path from v to  $p_k$ .

 $\bullet\,$  One of the earliest use of Dynamic Programming (1950's) invented by Bellman.

# • Instance: A directed weighted graph G = (V, E) with weights which can be Assignified the project of the same and a vertex H and a vertex H and H and H are H and H and H are H are H and H are H are H and H are H and H are H are H are H are H and H are H are H are H and H are H are H and H are H are H are H are H and H are H are H and H are H are H are H and H are H and H are H are H are H are H and H are H are H are H are H are H are H and H are H and H are H and H are H and H are H are H are H are H are H are H ar

- Goal: Find the shortest path from vertex s to every other vertex t.
- Solution: Since there are no negative weight cycles, the shortest path cannot contain tyles because by the coise a mode of that er path.
- Thus, every shortest path can have at most |V| 1 edges.
- \* Add We Chat powcoder s
- Our goal is to find for every vertex  $t \in G$  the value of  $\operatorname{opt}(n-1,t)$  and the path which achieves such a length.
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \ldots, p_k, t)$  then  $(p_1, p_2, \ldots, p_k, t)$  must be the shortest path from  $p_1$  to t, and  $(v, p_1, p_2, \ldots, p_k)$  must also be the shortest path from v to  $p_k$ .

 $\bullet$  One of the earliest use of Dynamic Programming (1950's) invented by Bellman.

# • Instance: A directed weighted graph G = (V, E) with weights which can be As significant without cycles Project Example 1 and a vertex Help

- Goal: Find the shortest path from vertex s to every other vertex t.
- Solution: Since there are no negative weight cycles, the shortest path cannot contain the scause and where the product that explains a shortest path.
- Thus, every shortest path can have at most |V| 1 edges.
- Subproblems: For every  $v \in V$  and every i,  $(1 \le i \le n-1)$ , let  $\mathrm{opt}(i,v)$  be the length of a shortest partition. It will obtain a close edges.
- Our goal is to find for every vertex  $t \in G$  the value of  $\operatorname{opt}(n-1,t)$  and the path which achieves such a length.
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \ldots, p_k, t)$  then  $(p_1, p_2, \ldots, p_k, t)$  must be the shortest path from  $p_1$  to t, and  $(v, p_1, p_2, \ldots, p_k)$  must also be the shortest path from v to  $p_k$ .

 $\bullet$  One of the earliest use of Dynamic Programming (1950's) invented by Bellman.

# • Instance: A directed weighted graph G = (V, E) with weights which can be a separate by the properties of the shortest path from vertex a to every other vertex t.

- Goal: Find the shortest path from vertex s to every other vertex t.
- Solution: Since there are no negative weight cycles, the shortest path cannot contain the scause and where the product that explains a shortest path.
- Thus, every shortest path can have at most |V| 1 edges.
- Subproblems: For every  $v\in V$  and every  $i,\ (1\leq i\leq n-1),\ \mathrm{lgt}\ \mathrm{opt}(i,v)$  be the length of a short of partition V and V will obtain a class  $\mathbf{C}$  ges.
- Our goal is to find for every vertex  $t \in G$  the value of opt(n-1,t) and the path which achieves such a length.
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \ldots, p_k, t)$  then  $(p_1, p_2, \ldots, p_k, t)$  must be the shortest path from  $p_1$  to t, and  $(v, p_1, p_2, \ldots, p_k)$  must also be the shortest path from v to  $p_k$ .

 $\bullet\,$  One of the earliest use of Dynamic Programming (1950's) invented by Bellman.

# • Instance: A directed weighted graph G = (V, E) with weights which can be a specialty of the property of the property of the shortest path from parters at a graph of the property of the

- Goal: Find the shortest path from vertex s to every other vertex t.
- Solution: Since there are no negative weight cycles, the shortest path cannot contain this locause and whereaster and contain the locause and the the locause an
- Thus, every shortest path can have at most |V| 1 edges.
- Our goal is to find for every vertex  $t \in G$  the value of opt(n-1,t) and the path which achieves such a length.
- Note that if the shortest path from a vertex v to t is  $(v, p_1, p_2, \ldots, p_k, t)$  then  $(p_1, p_2, \ldots, p_k, t)$  must be the shortest path from  $p_1$  to t, and  $(v, p_1, p_2, \ldots, p_k)$  must also be the shortest path from v to  $p_k$ .

• Let us denote the length of the shortest path from s to v among all paths which contain at most i edges by  $\operatorname{opt}(i,v)$ , and let  $\operatorname{pred}(i,v)$  be the immediate predecessor of vertex v on such shortest path.

### Assignment Project Exam Help

 $\begin{aligned} & \text{pred}(i,v) = \begin{cases} & \text{pred}(i-1,v) & \text{if } & \min_{p \in V} \{ \text{opt}(i-1,p) + \text{w}(e(p,v)) \} \geq \text{opt}(i-1,v) \\ & \text{https://powcoder.com} \\ & \text{(here w}(e(p))) & \text{is the w} \\ & \text{ght of the edge } e(p,v) & \text{from vertex } p \text{ to vertex } v. \end{aligned}$ 

- Final solutions: p(n-1, v) for all  $v \in G$ .
- Compadd (NueChat powcoder and for each v, min is taken over all edges e(p,v) in pident to v; thus in each round all edges are inspected.
- $\bullet$  Algorithm produces shortest paths from s to every other vertex in the graph.
- The method employed is sometimes called "relaxation", because we progressively relax the additional constraint on how many edges the shortest paths can contain.

• Let us denote the length of the shortest path from s to v among all paths which contain at most i edges by  $\operatorname{opt}(i,v)$ , and let  $\operatorname{pred}(i,v)$  be the immediate predecessor of vertex v on such shortest path.

## Assignment Project Exam Help

$$\operatorname{pred}(i,v) = \begin{cases} \operatorname{pred}(i-1,v) & \text{if } \min_{p \in V} \left\{ \operatorname{opt}(i-1,p) + \operatorname{w}(e(p,v)) \right\} \geq \operatorname{opt}(i-1,v) \\ \operatorname{pred}(i,v) & \operatorname{pred}(i,v) + \operatorname{w}(e(p,v)) \right\} \end{cases}$$
 (here  $\operatorname{w}(e(p,v))$ ) is the weight of the edge  $e(p,v)$  from vertex  $p$  to vertex  $v$ .)

- Final solutions: p(n-1, v) for all  $v \in G$ .
- ComAdd (WeChat powcoder and for each v, min is taken over all edges e(p,v) in pident to v; thus in each round all edges are inspected.
- $\bullet$  Algorithm produces shortest paths from s to every other vertex in the graph.
- The method employed is sometimes called "relaxation", because we progressively relax the additional constraint on how many edges the shortest paths can contain.

• Let us denote the length of the shortest path from s to v among all paths which contain at most i edges by  $\operatorname{opt}(i,v)$ , and let  $\operatorname{pred}(i,v)$  be the immediate predecessor of vertex v on such shortest path.

## Assignment Project Exam Help

$$\operatorname{pred}(i,v) = \begin{cases} \operatorname{pred}(i-1,v) & \text{if } \min_{p \in V} \left\{ \operatorname{opt}(i-1,p) + \operatorname{w}(e(p,v)) \right\} \geq \operatorname{opt}(i-1,v) \\ \operatorname{pred}(i,v) & \operatorname{pred}(i,v) + \operatorname{w}(e(p,v)) \right\} \end{cases}$$
 (here  $\operatorname{w}(e(p,v))$ ) is the weight of the edge  $e(p,v)$  from vertex  $p$  to vertex  $v$ .)

- Final solutions: p(n-1,v) for all  $v \in G$ .
- Compadd (NueChat powcoder and for each v, min is taken over all edges e(p,v) in pident to v; thus in each round all edges are inspected.
- $\bullet$  Algorithm produces shortest paths from s to every other vertex in the graph.
- The method employed is sometimes called "relaxation", because we progressively relax the additional constraint on how many edges the shortest paths can contain.

• Let us denote the length of the shortest path from s to v among all paths which contain at most i edges by opt(i, v), and let pred(i, v) be the immediate predecessor of vertex v on such shortest path.

## Assignment Project Exam Help

 $\operatorname{pred}(i,v) = \begin{cases} \operatorname{pred}(i-1,v) & \text{if } \min_{p \in V} \left\{ \operatorname{opt}(i-1,p) + \operatorname{w}(e(p,v)) \right\} \geq \operatorname{opt}(i-1,v) \\ \operatorname{pred}(i,v) & \operatorname{pred}(i,v) + \operatorname{w}(e(p,v)) \right\} \end{cases}$  (here  $\operatorname{w}(e(p,v))$ ) is the weight of the edge e(p,v) from vertex p to vertex v.)

- Final solutions: p(n-1, v) for all  $v \in G$ .
- Comparison of (i, v) us a time (v) problem see (v) and for each v, min is taken over all edges e(p, v) incident to v; thus in each round all edges are inspected.
- ullet Algorithm produces shortest paths from s to every other vertex in the graph.
- The method employed is sometimes called "relaxation", because we progressively relax the additional constraint on how many edges the shortest paths can contain.

• Let us denote the length of the shortest path from s to v among all paths which contain at most i edges by  $\operatorname{opt}(i,v)$ , and let  $\operatorname{pred}(i,v)$  be the immediate predecessor of vertex v on such shortest path.

## Assignment Project Exam Help

 $\operatorname{pred}(i,v) = \begin{cases} \operatorname{pred}(i-1,v) & \text{if } \min_{p \in V} \left\{ \operatorname{opt}(i-1,p) + \operatorname{w}(e(p,v)) \right\} \geq \operatorname{opt}(i-1,v) \\ \operatorname{pred}(i,v) & \operatorname{pred}(i,v) + \operatorname{w}(e(p,v)) \right\} \end{cases}$  (here  $\operatorname{w}(e(p,v))$ ) is the weight of the edge e(p,v) from vertex p to vertex v.)

- Final solutions: p(n-1,v) for all  $v \in G$ .
- Comparison of (i, v) us a time (v) problem see (v) and for each v, min is taken over all edges e(p, v) incident to v; thus in each round all edges are inspected.
- ullet Algorithm produces shortest paths from s to every other vertex in the graph.
- The method employed is sometimes called "relaxation", because we progressively relax the additional constraint on how many edges the shortest paths can contain.

• Let us denote the length of the shortest path from s to v among all paths which contain at most i edges by opt(i, v), and let pred(i, v) be the immediate predecessor of vertex v on such shortest path.

## $\textbf{Assignment}_{\text{opt}(i,v)} \underbrace{\textbf{Project}}_{\text{min}(\text{opt}(i-1,v)}, \underbrace{\textbf{Project}}_{\text{net}} \underbrace{\textbf{Exam}}_{\text{Net}} \textbf{Help}$

 $\operatorname{pred}(i,v) = \begin{cases} \operatorname{pred}(i-1,v) & \text{if } \min_{p \in V} \{\operatorname{opt}(i-1,p) + \operatorname{w}(e(p,v))\} \geq \operatorname{opt}(i-1,v) \\ \operatorname{pred}(i,v) & \operatorname{pred}(i,v) + \operatorname{we}(p,v) \\ \operatorname{opt}(i,v) & \operatorname{pred}(i,v) + \operatorname{we}(p,v) \end{cases}$  (here  $\operatorname{w}(e(p,v))$ ) is the weight of the edge e(p,v) from vertex p to vertex v.)

- Final solutions: p(n-1,v) for all  $v \in G$ .
- Compared to t(i), which time a[t] and b[t] and for each v, min is taken over all edges e(p,v) incident to v; thus in each round all edges are inspected.
- Algorithm produces shortest paths from s to every other vertex in the graph.
- The method employed is sometimes called "relaxation", because we progressively relax the additional constraint on how many edges the shortest paths can contain. 38 / 41

• Let again G = (V, E) be a directed weighted graph where  $V = \{v_1, v_2, \dots, v_n\}$  and where weights  $w(e(v_p, v_q))$  of edges  $e(v_p, v_q)$  can be negative, but there are no negative weight cycles.

### Assignment Project Exam Help

vertex  $v_p$  to every vertex  $v_q$  (including back to  $v_p$ )

• Let  $\mathbf{p}(k, v_p, v_q)$  be the length of the shorten path from a vertex  $v_p$  to a vertex  $\mathbf{p}(v_1, v_2, \dots, v_q)$ . (1) • Proposition of the shorten path from a vertex  $v_p$  to a vertex  $v_$ 

Then

### 

- Thus, we gradually **relax** the constraint that the intermediary vertices have to belong to  $\{v_1, v_2, \ldots, v_k\}$ .
- Algorithm runs in time  $|V|^3$ .

• Let again G = (V, E) be a directed weighted graph where  $V = \{v_1, v_2, \dots, v_n\}$  and where weights  $w(e(v_p, v_q))$  of edges  $e(v_p, v_q)$  can be negative, but there are no negative weight cycles.

Assignment Project Exam Help vertex  $v_p$  to every vertex  $v_q$  (including back to  $v_p$ ).

https://powcoder.com

Then

### 

- Thus, we gradually **relax** the constraint that the intermediary vertices have to belong to  $\{v_1, v_2, \ldots, v_k\}$ .
- Algorithm runs in time  $|V|^3$ .

• Let again G = (V, E) be a directed weighted graph where  $V = \{v_1, v_2, \dots, v_n\}$  and where weights  $w(e(v_p, v_q))$  of edges  $e(v_p, v_q)$  can be negative, but there are no negative weight cycles.

Assignment Project Exam Help vertex  $v_p$  to every vertex  $v_q$  (including back to  $v_p$ ).

• Let  $\operatorname{cpt}(k, \gamma_p, v_q)$  be the length of the shortest path from a vertex  $v_p$  to a vertex  $v_p$  such that all/into modific vertices bearing entropy  $\{v_1, v_2, \dots, u_k\}, (1 \le k \le n).$ 

Then

### 

- Thus, we gradually **relax** the constraint that the intermediary vertices have to belong to  $\{v_1, v_2, \ldots, v_k\}$ .
- Algorithm runs in time  $|V|^3$ .

• Let again G = (V, E) be a directed weighted graph where  $V = \{v_1, v_2, \dots, v_n\}$  and where weights  $w(e(v_p, v_q))$  of edges  $e(v_p, v_q)$  can be negative, but there are no negative weight cycles.

Assignment Project Exam Help vertex  $v_p$  to every vertex  $v_q$  (including back to  $v_p$ ).

- Let  $\operatorname{cpt}(k, p_p, v_q)$  be the length of the shortest path from a vertex  $v_p$  to a vertex  $v_p$  such that all/into modify vertices bearing entropy  $\{v_1, v_2, \dots, u_k\}, (1 \le k \le n).$
- $\begin{array}{c} \bullet \text{ Then} \\ \text{opt} \\ & \text{Add} \\ & \text{mM} \\ & \text{peChatpt} \\ & \text{O,WcOder} \\ & v_k, v_q) \end{array} \}$
- Thus, we gradually **relax** the constraint that the intermediary vertices have to belong to  $\{v_1, v_2, \ldots, v_k\}$ .
- Algorithm runs in time  $|V|^3$ .

• Let again G = (V, E) be a directed weighted graph where  $V = \{v_1, v_2, \dots, v_n\}$  and where weights  $w(e(v_p, v_q))$  of edges  $e(v_p, v_q)$  can be negative, but there are no negative weight cycles.

Assignment Project Exam Help vertex  $v_p$  to every vertex  $v_q$  (including back to  $v_p$ ).

- Let  $\operatorname{cpt}(k, p_p, v_q)$  be the length of the shortest path from a vertex  $v_p$  to a vertex  $v_p$  to a vertex  $v_p$  to a vertex  $v_p$  to a limit of  $v_p$  to a vertex  $v_p$  to a  $v_p$  to  $v_p$  to v
- $\begin{array}{c} \bullet \text{ Then} \\ \text{opt} & A_{v} d_{v} d_{v} \\ \end{array} \\ \text{mod } & \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_$
- Thus, we gradually **relax** the constraint that the intermediary vertices have to belong to  $\{v_1, v_2, \ldots, v_k\}$ .
- Algorithm runs in time  $|V|^3$ .



• Let again G = (V, E) be a directed weighted graph where  $V = \{v_1, v_2, \dots, v_n\}$  and where weights  $w(e(v_p, v_q))$  of edges  $e(v_p, v_q)$  can be negative, but there are no negative weight cycles.

Assignment Project Exam Help vertex  $v_p$  to every vertex  $v_q$  (including back to  $v_p$ ).

- Let  $\operatorname{cpt}(k, p_p, v_q)$  be the length of the shortest path from a vertex  $v_p$  to a vertex  $v_p$  to a vertex  $v_p$  to a vertex  $v_p$  to a limit of  $v_p$  to a vertex  $v_p$  to a  $v_p$  to  $v_p$  to v
- $\begin{array}{c} \bullet \text{ Then} \\ \text{opt} & A_{v} d_{v} d_{v} \\ \end{array} \\ \text{mod } & \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right. \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_{v} d_{v} \\ \end{array} \right\} \\ \text{mod } \left\{ \begin{array}{c} A_{v} d_{v} \\ A_$
- Thus, we gradually **relax** the constraint that the intermediary vertices have to belong to  $\{v_1, v_2, \ldots, v_k\}$ .
- Algorithm runs in time  $|V|^3$ .

#### Another example of relaxation:

• Compute the number of partitions of a positive integer n. That is to say the number of distinct multi-sets of positive integers  $\{n_1, \ldots, n_k\}$  which sum up to n, i.e., such that  $n_1 + \ldots + n_k = n$ .

## Assignment Project Exam Help

multi-set.

Hint Let nump(i, j) denotes the number of partitions  $\{j_1,\ldots,j_p\}$  of j, i.e., https://powcoder.com addition, have the property that every element  $j_q$  of each partition satisfies  $j_q \leq i$ . We are looking for nump(n,n) but the recursion is based on relaxation of the allowed size i of the parts of j for all  $i,j \leq n$ . To get a recursive definition of the parts of j for all  $i,j \leq n$ . To get a recursive definition of the parts of j for all j the case of partitions where all compared to the context of the parts of j for all j the case of partitions where all compared to the parts of j for all j the case of partitions where all compared to j for all j the case of partitions where all j the case of partitions where j is the case of partitions where j is j the case of partitions j in j is j the case of partitions j in j is j the case of partitions j in j i

4□▶ 4₫▶ 4½▶ ½ 900

#### Another example of relaxation:

• Compute the number of partitions of a positive integer n. That is to say the number of distinct multi-sets of positive integers  $\{n_1, \ldots, n_k\}$  which sum up to n, i.e., such that  $n_1 + \ldots + n_k = n$ .

As significant that the excent are swellowing of elements count as a single multi-set.

Hint Let  $\operatorname{pump}(i,j)$  lenotes the number of partitions  $\{j_1,\ldots,j_p\}$  of j, i.e.,  $\operatorname{https}(i,j)$  powcoder com addition, have the property that every element  $j_q$  of each partition satisfies  $j_q \leq i$ . We are looking for  $\operatorname{nump}(n,n)$  but the recursion is based on relaxation of the allowed size i of the parts of j for all  $i,j \leq n$ . To get a recursive definition of  $\operatorname{num}(n,n)$  but the case of partitions where all compared to  $\operatorname{num}(n,n)$  but  $\operatorname{num}(n,n)$  but the case of partitions where all compared to  $\operatorname{num}(n,n)$  but  $\operatorname{num}(n,n)$  but the case of partitions where all compared to  $\operatorname{num}(n,n)$  but  $\operatorname{num}(n,n)$  but the recursion is based on relaxation of the allowed size i of  $\operatorname{num}(n,n)$  but the recursion is based on relaxation of the allowed size i of  $\operatorname{num}(n,n)$  but the recursion is based on relaxation of the allowed size i of the parts of i of i

#### Another example of relaxation:

• Compute the number of partitions of a positive integer n. That is to say the number of distinct multi-sets of positive integers  $\{n_1, \ldots, n_k\}$  which sum up to n, i.e., such that  $n_1 + \ldots + n_k = n$ .

Assignment that has a count as a single multi-set.

Hint Let nump(i, j) denotes the number of partitions  $\{j_1, \dots, j_p\}$  of j, i.e., the number of sets such that  $(j_1 + \dots + j_p) = C$  which in addition, have the property that every element  $j_q$  of each partition satisfies  $j_q \leq i$ . We are looking for nump(n,n) but the recursion is based on relaxation of the allowed size i of the parts of j for all  $i, j \leq n$ . To get a recursive definition of nump(i,j) distinguish the case of partitions where all components are  $\leq V$  fand the case where all vas one to replace is of size i.

### Assignment Project Exam Help

You have 2 lengths of fuse that are guaranteed to burn for precisely 1 minute each. Other than that fact, you know nothing; they may burn at different lines at variable was precisely. It follows that the fact, you know nothing; they may burn at different lengths, thick nesses, materials, etc. How can you use these two fuses to time a 45 second interval?

### Add WeChat powcoder