Discrete Optimization: A sample of Problems

Ngày 27 tháng 10 năm 2011

A Brief Introduction to Discrete Optimization

Discrete or Combinatorial Optimization deals mainly with problems where we have to choose an optimal solution from a finite (or sometimes countable) number of possibilities.

A Brief Introduction to Discrete Optimization

Discrete or Combinatorial Optimization deals mainly with problems where we have to choose an optimal solution from a finite (or sometimes countable) number of possibilities.

In this short introduction we shall visit a sample of Discrete Optimization problems, step through the thinking process of developing a solution and completely solve one problem.
Let us start with a short list of problems.

A Brief Introduction to Discrete Optimization

Discrete or Combinatorial Optimization deals mainly with problems where we have to choose an optimal solution from a finite (or sometimes countable) number of possibilities.

In this short introduction we shall visit a sample of Discrete Optimization problems, step through the thinking process of developing a solution and completely solve one problem.
Let us start with a short list of problems.

Example

You have a collection of 10000 *objects. Each object has a "value"* v_n *(say 44,500 VND).*

A Brief Introduction to Discrete Optimization

Discrete or Combinatorial Optimization deals mainly with problems where we have to choose an optimal solution from a finite (or sometimes countable) number of possibilities.

In this short introduction we shall visit a sample of Discrete Optimization problems, step through the thinking process of developing a solution and completely solve one problem.
Let us start with a short list of problems.

Example

You have a collection of 10000 *objects. Each object has a "value"* v_n *(say 44,500 VND).*

1. *Can you find a subset of objects whose total value is 2,000,000,000 VND?*

A Brief Introduction to Discrete Optimization

Discrete or Combinatorial Optimization deals mainly with problems where we have to choose an optimal solution from a finite (or sometimes countable) number of possibilities.

In this short introduction we shall visit a sample of Discrete Optimization problems, step through the thinking process of developing a solution and completely solve one problem.
Let us start with a short list of problems.

Example

You have a collection of 10000 *objects. Each object has a "value"* v_n *(say 44,500 VND).*

1. *Can you find a subset of objects whose total value is 2,000,000,000 VND?*
2. *Can you partition the collection into two sub collections of equal value?*

Remark

Both problems are very simple and easy to understand.

Remark

Both problems are very simple and easy to understand.

Remark

Both problems are very simple and easy to understand.
Both seem to have a very “simple” mathematical solution:
Try all possibilities.

Remark

Both problems are very simple and easy to understand.
Both seem to have a very “simple” mathematical solution:
Try all possibilities.
The solutions are not practical even with the fastest computers.

Remark

This situation is typical of many discrete optimization problems. The number of options from which an optimal solution to be chosen is way to big.

Remark

Both problems are very simple and easy to understand.
Both seem to have a very "simple" mathematical solution:
Try all possibilities.
The solutions are not practical even with the fastest computers.

Remark

This situation is typical of many discrete optimization problems. The number of options from which an optimal solution to be chosen is way to big.

Remark

Both problems are very simple and easy to understand.
Both seem to have a very "simple" mathematical solution:
Try all possibilities.
The solutions are not practical even with the fastest computers.

Remark

This situation is typical of many discrete optimization problems. The number of options from which an optimal solution to be chosen is way to big.

For instance, both problems can be solved by testing all possible subsets of objects.

Remark

Both problems are very simple and easy to understand.
Both seem to have a very "simple" mathematical solution:
Try all possibilities.
The solutions are not practical even with the fastest computers.

Remark

This situation is typical of many discrete optimization problems. The number of options from which an optimal solution to be chosen is way to big.

For instance, both problems can be solved by testing all possible subsets of objects.

There are "only" 2^{10000} *subsets... :-(*

A sample of solvable scheduling problems

Question (Scheduling to minimize lateness)

A single resource is available to process jobs (for instance a printer in an office, a big crane in a building site, etc.). n jobs are to be processed by the resource. Once a job starts, it cannot be interrupted. Processing jobs starts at time 0. Each job has a deadline d_i *and processing time* p_i*. We need to schedule the jobs so that the lateness* $(f_i - d_i)$*, the difference between the finishing time and deadline will be minimized.*

A sample of solvable scheduling problems

Question (Scheduling to minimize lateness)

A single resource is available to process jobs (for instance a printer in an office, a big crane in a building site, etc.). n jobs are to be processed by the resource. Once a job starts, it cannot be interrupted. Processing jobs starts at time 0. Each job has a deadline d_i and processing time p_i. We need to schedule the jobs so that the lateness $(f_i - d_i)$, the difference between the finishing time and deadline will be minimized.

Question (Scheduling to minimize the number of late jobs)

We can have different objectives for the same problem. For instance, we wish to schedule the same jobs so that the number of late jobs will be minimized.

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

1. *Generate all possible orderings (permutations) of the jobs.*

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

1. *Generate all possible orderings (permutations) of the jobs.*
2. *For each ordering calculate the maximum lateness (or the number of late jobs).*

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

1. *Generate all possible orderings (permutations) of the jobs.*
2. *For each ordering calculate the maximum lateness (or the number of late jobs).*
3. *Note that it is very easy to write a program that will calculate these number very fast.*

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

1. *Generate all possible orderings (permutations) of the jobs.*
2. *For each ordering calculate the maximum lateness (or the number of late jobs).*
3. *Note that it is very easy to write a program that will calculate these number very fast.*
4. *Select the optimal ordering.*

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

1. *Generate all possible orderings (permutations) of the jobs.*
2. *For each ordering calculate the maximum lateness (or the number of late jobs).*
3. *Note that it is very easy to write a program that will calculate these number very fast.*
4. *Select the optimal ordering.*
5. *So what is the problem?*

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

1. *Generate all possible orderings (permutations) of the jobs.*
2. *For each ordering calculate the maximum lateness (or the number of late jobs).*
3. *Note that it is very easy to write a program that will calculate these number very fast.*
4. *Select the optimal ordering.*
5. *So what is the problem?*
6. *There are "only" $n!$ permutations to consider!*

Solution candidates

Discussion

The solution to both problems looks mathematically simple:

1. *Generate all possible orderings (permutations) of the jobs.*
2. *For each ordering calculate the maximum lateness (or the number of late jobs).*
3. *Note that it is very easy to write a program that will calculate these number very fast.*
4. *Select the optimal ordering.*
5. *So what is the problem?*
6. *There are “only” $n!$ permutations to consider!*
7. *If $n = 100$ then there are only $100!$ possibilities and $100!$ is so huge, it does not have a name in any language. It is “only” 158-digits long.*

solution cndidates

Question

Can we try to suggest a solution to the minimum lateness problem?

How about scheduling the jobs in order of increasing processing time ?

solution cndidates

Question

Can we try to suggest a solution to the minimum lateness problem?

How about scheduling the jobs in order of increasing processing time ?

solution cndidates

Question

Can we try to suggest a solution to the minimum lateness problem?

How about scheduling the jobs in order of increasing processing time ?
The idea: get rid of the short jobs first.

solution cndidates

Question

Can we try to suggest a solution to the minimum lateness problem?

How about scheduling the jobs in order of increasing processing time ?
The idea: get rid of the short jobs first.
Won't work! Consider 2 jobs:
$J_1(p_1 = 100,\ d_1 = 1000) \quad J_2(p_2 = 500,\ d_2 = 500)$.

solution cndidates

Question

Can we try to suggest a solution to the minimum lateness problem?

How about scheduling the jobs in order of increasing processing time ?
The idea: get rid of the short jobs first.
Won't work! Consider 2 jobs:
$J_1(p_1 = 100,\ d_1 = 1000) \quad J_2(p_2 = 500,\ d_2 = 500)$.
In this schedule J_2 will finish at time 600 or 100 minutes late.

solution cndidates

Question

Can we try to suggest a solution to the minimum lateness problem?

How about scheduling the jobs in order of increasing processing time ?
The idea: get rid of the short jobs first.
Won't work! Consider 2 jobs:
$J_1(p_1 = 100,\ d_1 = 1000) \quad J_2(p_2 = 500,\ d_2 = 500)$.
In this schedule J_2 will finish at time 600 or 100 minutes late.
On the other hand, if we schedule J_2 first it will finish on time at time 500 and J_1 will finish at time 600 with no lateness.

solution cndidates

Question

Can we try to suggest a solution to the minimum lateness problem?

How about scheduling the jobs in order of increasing processing time ?
The idea: get rid of the short jobs first.
Won't work! Consider 2 jobs:
$J_1(p_1 = 100,\ d_1 = 1000) \quad J_2(p_2 = 500,\ d_2 = 500)$.
In this schedule J_2 will finish at time 600 or 100 minutes late.
On the other hand, if we schedule J_2 first it will finish on time at time 500 and J_1 will finish at time 600 with no lateness.
Seems like the problem is that we ignore the finish time.

Question

How about scheduling by the shortest difference $d_i - p_i$, these are the jobs that look to have less time to wait. This schedule does consider all data.

Question

How about scheduling by the shortest difference $d_i - p_i$, these are the jobs that look to have less time to wait. This schedule does consider all data.

Discussion

Once again we can show that this rule fails.

Question

How about scheduling by the shortest difference $d_i - p_i$, these are the jobs that look to have less time to wait. This schedule does consider all data.

Discussion

Once again we can show that this rule fails.

Question

How about scheduling by the shortest difference $d_i - p_i$, these are the jobs that look to have less time to wait. This schedule does consider all data.

Discussion

Once again we can show that this rule fails.
To do this we need to show an example where this rule fails to produce the smallest lateness.

Question

How about scheduling by the shortest difference $d_i - p_i$, these are the jobs that look to have less time to wait. This schedule does consider all data.

Discussion

Once again we can show that this rule fails.
To do this we need to show an example where this rule fails to produce the smallest lateness.
Assume that $J_1(p_1 = 100, d_1 = 100), \quad J_2(p_2 = 5, d_2 = 10).$
The suggested schedule will schedule J_1 first causing J_2 to finish at time 110*, 100 minutes delay. On the other hand if we schedule J_2 first J_1 will finish at time* 105 *with a delay of only* 5 *minutes.*

The best schedule for minimizng lateness

There is a somewhat surprising schedule that minimizes the lateness. The surprise is that it ignores the processing time.

Theorem

Performing the jobs by increasing deadline will produce the minimal lateness.

In other words, by presorting the jobs by their deadline d_i we get the optimal schedule. Clearly this can be easily accomplished very fast even for millions of jobs!

Chứng minh.

□

Chứng minh.

1. Let $J_1\ J_2 \dots J_n$ be a schedule.

□

Chứng minh.

1. Let $J_1\ J_2 \ldots J_n$ be a schedule.
2. For this schedule we have:

$$f_m = \sum_{i=1}^{m} p_i$$

.

□

Chứng minh.

1. Let $J_1 \ J_2 \ldots J_n$ be a schedule.
2. For this schedule we have:

$$f_m = \sum_{i=1}^{m} p_i$$

.

3. Assume that for some index $k, \ d_k > d_{k+1}$. (An inversion in the permutation).

□

Chứng minh.

1. Let $J_1\ J_2 \ldots J_n$ be a schedule.
2. For this schedule we have:

$$f_m = \sum_{i=1}^{m} p_i$$

.

3. Assume that for some index $k,\ d_k > d_{k+1}$. (An inversion in the permutation).
4. Let us compare the lateness of this schedule with the schedule: $J_1,\ J_2, \ldots, J_{k-1},\ j_{k+1},\ J_k \ldots, J_n$.

□

Chứng minh.

1. Let $J_1\ J_2 \ldots J_n$ be a schedule.
2. For this schedule we have:

$$f_m = \sum_{i=1}^{m} p_i$$

.

3. Assume that for some index $k,\ d_k > d_{k+1}$. (An inversion in the permutation).
4. Let us compare the lateness of this schedule with the schedule: $J_1,\ J_2, \ldots, J_{k-1},\ j_{k+1},\ J_k \ldots, J_n$.
5. It is easy to see that $f_i - d_i$ remains the same for all jobs different from J_k *and* J_{k+1}.

□

End of proof

1. Let us denote the finishing time for this schedule by g_i.

End of proof

1. Let us denote the finishing time for this schedule by g_i.
2. We have:

$$g_{k+1} - d_{k+1} = \sum_{i=1}^{k-1} p_i + p_{k+1} - d_{k+1} < \sum_{i=1}^{k+1} p_i - d_{k+1} = f_{k+1} - d_{k+1}$$

Similarly:

$$g_k - d_k = \sum_{i=1}^{k+1} p_i - d_k = f_{k+1} - d_k < f_{k+1} - d_{k+1}$$

since $d_{k+1} < d_k$.

End of proof

1. Let us denote the finishing time for this schedule by g_i.
2. We have:

$$g_{k+1} - d_{k+1} = \sum_{i=1}^{k-1} p_i + p_{k+1} - d_{k+1} < \sum_{i=1}^{k+1} p_i - d_{k+1} = f_{k+1} - d_{k+1}$$

   Similarly:

$$g_k - d_k = \sum_{i=1}^{k+1} p_i - d_k = f_{k+1} - d_k < f_{k+1} - d_{k+1}$$

   since $d_{k+1} < d_k$.
3. This means that all delays remain the same except for $g_k, \ g_{k+1}$ which are smaller than the delay $f_{k+1} - d_{k+1}$ which can only decrease the largest delay.

End of proof

1. Let us denote the finishing time for this schedule by g_i.
2. We have:

$$g_{k+1} - d_{k+1} = \sum_{i=1}^{k-1} p_i + p_{k+1} - d_{k+1} < \sum_{i=1}^{k+1} p_i - d_{k+1} = f_{k+1} - d_{k+1}$$

Similarly:

$$g_k - d_k = \sum_{i=1}^{k+1} p_i - d_k = f_{k+1} - d_k < f_{k+1} - d_{k+1}$$

since $d_{k+1} < d_k$.

3. This means that all delays remain the same except for $g_k,\ g_{k+1}$ which are smaller than the delay $f_{k+1} - d_{k+1}$ which can only decrease the largest delay.
4. By the exchange, we removed one inversion in the permutation. Thus by removing all inversions we can only reduce latenesses.

Minimizing the number of delayed jobs

We shall try to explore this problem and try to come up with an optimal solution.

1. First step: Does the previous algorithm produce an optimal schedule?

Minimizing the number of delayed jobs

We shall try to explore this problem and try to come up with an optimal solution.

1. First step: Does the previous algorithm produce an optimal schedule?
    - If no, produce a counter example.

Minimizing the number of delayed jobs

We shall try to explore this problem and try to come up with an optimal solution.

1. First step: Does the previous algorithm produce an optimal schedule?
   - If no, produce a counter example.
   - If yes, can you prove it?

Minimizing the number of delayed jobs

We shall try to explore this problem and try to come up with an optimal solution.

1. First step: Does the previous algorithm produce an optimal schedule?
   - If no, produce a counter example.
   - If yes, can you prove it?
2. Try another algorithm, any suggestion?

Minimizing the number of delayed jobs

We shall try to explore this problem and try to come up with an optimal solution.

1. First step: Does the previous algorithm produce an optimal schedule?
   - If no, produce a counter example.
   - If yes, can you prove it?
2. Try another algorithm, any suggestion?
3. What is the best schedule for the following 8 jobs:

   $J_1(15, 20),\ J_3(20, 40),\ J_4(20, 60),\ J_5(10, 30),$
   $J_6(30, 70),\ J_2(5, 20),\ J_7(15, 50),\ J_8(40, 80).$

Minimizing the number of delayed jobs

We shall try to explore this problem and try to come up with an optimal solution.

1. First step: Does the previous algorithm produce an optimal schedule?
   - If no, produce a counter example.
   - If yes, can you prove it?
2. Try another algorithm, any suggestion?
3. What is the best schedule for the following 8 jobs:

   $J_1(15, 20),\ J_3(20, 40),\ J_4(20, 60),\ J_5(10, 30),$

   $J_6(30, 70),\ J_2(5, 20),\ J_7(15, 50),\ J_8(40, 80).$
4. What can we learn from this example?

Minimizing the number of delayed jobs

We shall try to explore this problem and try to come up with an optimal solution.

1. First step: Does the previous algorithm produce an optimal schedule?
   - If no, produce a counter example.
   - If yes, can you prove it?
2. Try another algorithm, any suggestion?
3. What is the best schedule for the following 8 jobs:

   $J_1(15,20),\ J_3(20,40),\ J_4(20,60),\ J_5(10,30),$

   $J_6(30,70),\ J_2(5,20),\ J_7(15,50),\ J_8(40,80).$
4. What can we learn from this example?
5. Any suggestion? A heuristic?

1. Lets try the following:

1. Lets try the following:
    - Sort the jobs in increasing finishing time.

1. Lets try the following:
    - Sort the jobs in increasing finishing time.
    - If there is more than one job with the same finishing time select first the one with the shorter processing time.

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.
2. This is a heuristic. If correct, we need a proof.

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.
2. This is a heuristic. If correct, we need a proof.
3. If not, we need a counter example.

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.
2. This is a heuristic. If correct, we need a proof.
3. If not, we need a counter example.
4. Lets check what this heuristic does for the 8 jobs sample:

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.
2. This is a heuristic. If correct, we need a proof.
3. If not, we need a counter example.
4. Lets check what this heuristic does for the 8 jobs sample:
5. It is easy to see that this is not a correct solution.

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.
2. This is a heuristic. If correct, we need a proof.
3. If not, we need a counter example.
4. Lets check what this heuristic does for the 8 jobs sample:
5. It is easy to see that this is not a correct solution.
6. Let the first job be: $J_1(2000, 2000)$ and let $J_k(20, 2010), k = 1, \ldots, 100$.

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.
2. This is a heuristic. If correct, we need a proof.
3. If not, we need a counter example.
4. Lets check what this heuristic does for the 8 jobs sample:
5. It is easy to see that this is not a correct solution.
6. Let the first job be: $J_1(2000, 2000)$ and let $J_k(20, 2010), k = 1, \ldots, 100$.
7. The algorithm will schedule J_1 and there will be 100 late jobs.

1. Lets try the following:
   - Sort the jobs in increasing finishing time.
   - If there is more than one job with the same finishing time select first the one with the shorter processing time.
   - Run through your sorted list. If a job is going to be late, remove it from the list.
2. This is a heuristic. If correct, we need a proof.
3. If not, we need a counter example.
4. Lets check what this heuristic does for the 8 jobs sample:
5. It is easy to see that this is not a correct solution.
6. Let the first job be: $J_1(2000, 2000)$ and let $J_k(20, 2010), k = 1, \ldots, 100$.
7. The algorithm will schedule J_1 and there will be 100 late jobs.
8. On the other hand, we can finish on time 50 jobs and have only 51 late jobs.

Final proposed solution:

Current list = jobs sorted by processing time.

Final proposed solution:

Current list = jobs sorted by processing time.

Select the first job.

Final proposed solution:

Current list = jobs sorted by processing time.

Select the first job.

Remove from the list all jobs that cannot be completed on time to form the new current list.

Final proposed solution:

Current list = jobs sorted by processing time.

Select the first job.

Remove from the list all jobs that cannot be completed on time to form the new current list.

We shall address this algorithm in assignment No. 9.

Belady's scheduling problem

You decide to build your own xe may. You carefully study plans, tools needed. You come up with a list of 20 different tools that you will need. You also figure out that there will be 500 steps to complete the job. Unfortunately you do not have the tools. But Mr. Nguyen is renting tools. Every time you check out a tool, you have to pay Mr. Nguyen $20,000$ VND. Unfortunately he has an irritating policy: he will not allow you to check out more than 5 tools at a time. this means that if you have 5 tools and you need another tool, you'll have to choose one of your current tools, return it and check out the tool you need.

Belady's scheduling problem

You decide to build your own xe may. You carefully study plans, tools needed. You come up with a list of 20 different tools that you will need. You also figure out that there will be 500 steps to complete the job. Unfortunately you do not have the tools. But Mr. Nguyen is renting tools. Every time you check out a tool, you have to pay Mr. Nguyen $20,000$ VND. Unfortunately he has an irritating policy: he will not allow you to check out more than 5 tools at a time. this means that if you have 5 tools and you need another tool, you'll have to choose one of your current tools, return it and check out the tool you need.

You carefully look over your plan, redesign each step, make sure that in each step you will not need more than 5 tools. You ilst the tools.

Now you are facing another problem. Design which tools to exchange every time you need a tool you do not have. Your goal of course is to minimize the amount of money you'll have to spend renting the tools.

Now you are facing another problem. Design which tools to exchange every time you need a tool you do not have. Your goal of course is to minimize the amount of money you'll have to spend renting the tools.

For example, how much will you have to pay Mr. Nguyen for renting out the following list of tools (that will only manage to finish $\frac{1}{5}$ of the job):

11, 5, 4, 12, 15, 8, 8, 16, 3, 1, 2, 6, 1, 1, 19, 7, 15, 6, 19, 9, 5, 6, 18, 15, 14, 16, 18, 20, 9, 16, 5, 6, 14, 16, 13, 4, 4, 6, 17, 4, 7, 11, 19, 18, 5, 2, 8, 7, 20, 14, 17, 17, 4, 15, 2, 4, 9, 17, 19, 5, 4, 14, 9, 18, 19, 2, 20, 15, 7, 19, 11, 12, 1, 9, 16, 3, 1, 4, 14, 7, 18, 12, 7, 17, 1, 6, 3, 17, 10, 17, 7, 6, 9, 15, 16, 8, 9, 13, 9, 19

Now you are facing another problem. Design which tools to exchange every time you need a tool you do not have. Your goal of course is to minimize the amount of money you'll have to spend renting the tools.

For example, how much will you have to pay Mr. Nguyen for renting out the following list of tools (that will only manage to finish $\frac{1}{5}$ of the job):

11, 5, 4, 12, 15, 8, 8, 16, 3, 1, 2, 6, 1, 1, 19, 7, 15, 6, 19, 9, 5, 6, 18, 15, 14, 16, 18, 20, 9, 16, 5, 6, 14, 16, 13, 4, 4, 6, 17, 4, 7, 11, 19, 18, 5, 2, 8, 7, 20, 14, 17, 17, 4, 15, 2, 4, 9, 17, 19, 5, 4, 14, 9, 18, 19, 2, 20, 15, 7, 19, 11, 12, 1, 9, 16, 3, 1, 4, 14, 7, 18, 12, 7, 17, 1, 6, 3, 17, 10, 17, 7, 6, 9, 15, 16, 8, 9, 13, 9, 19

For instance, in the first stage you rent tools number $\{11, 5, 4, 12, 15\}$. Then you need to rent tool number 8. Which of the current 5 tools are you going to return?

There seem to be some reasonable options. For instance, we can remove the tool least frequently needed in the future.

There seem to be some reasonable options. For instance, we can remove the tool least frequently needed in the future.

It is also possible that there is no single algorithm that will produce the optimal solution for every input.

There seem to be some reasonable options. For instance, we can remove the tool least frequently needed in the future.

It is also possible that there is no single algorithm that will produce the optimal solution for every input.

In the 1960's L. Belady suggetsed the following procedure:

Evict the tool that will be needed the furthest away in the future.

Surprisingly, this strategy will produce an optimal schedule for any given sequence.

summary

This was a very brief introduaction to discrete optimization problems.

summary

This was a very brief introduaction to discrete optimization problems.

We saw a sample of problems, experienced the thinking process that led us to a complete solution of one problem.

summary

This was a very brief introduaction to discrete optimization problems.

We saw a sample of problems, experienced the thinking process that led us to a complete solution of one problem.

A solution to another problem left as an exercise.

summary

This was a very brief introduaction to discrete optimization problems.

We saw a sample of problems, experienced the thinking process that led us to a complete solution of one problem.

A solution to another problem left as an exercise.

And we are still pondering about building our own xe-may.

summary

This was a very brief introduaction to discrete optimization problems.

We saw a sample of problems, experienced the thinking process that led us to a complete solution of one problem.

A solution to another problem left as an exercise.

And we are still pondering about building our own xe-may.

Time permitting, we will study more discrete optimization problems in this class.