http://iti.wtf

Algorithms and Data Structures

Greedy Algorithms: Introduction

Artem A. Golubnichiy

artem@golubnichij.ru

Department of Software of Computer Facilities and Automated Systems

Katanov Khakass State University

STRUCTURE OF THE CLASS

- Maximize Your Salary
- Queue of Patients
- Implementation and Analysis
- Main Ingredients

WHAT'S COMING

- Solve salary maximization problem
- Come up with a greedy algorithm yourself
- Solve optimal queue arrangement problem
- Generalize solutions using the concepts of **greedy choice**, **subproblem** and **safe choice**

MAXIMIZE SALARY

MAXIMIZE SALARY

MAXIMIZE SALARY

LARGEST NUMBER

Toy problem
What is the largest number that consists of digits 9, 8, 9, 6, 1? Use all the digits.

LARGEST NUMBER

Toy problem

What is the largest number that consists of digits 9, 8, 9, 6, 1? Use all the digits.

Examples

16899, 69891, 98961, . . .

LARGEST NUMBER

Correct Answer
99861

GREEDY STRATEGY

{9, 8, 9, 6, 1} → ? ? ? ? ?

GREEDY STRATEGY

Find max

{9, 8, 9, 6, 1}

Find max digit

GREEDY STRATEGY

Find max

{9, 8, 9, 6, 1}

Find max digit

GREEDY STRATEGY

Find max

Append

{9, 8, 9, 6, 1} →

Find max digit

Append it to the number

GREEDY STRATEGY

Find max

Append

{9, 8, 9, 6, 1} → 9

Find max digit

Append it to the number

GREEDY STRATEGY

Find max

Append

{9, 8, 9, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

GREEDY STRATEGY

Find max

Append

{9, 8, 9, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

GREEDY STRATEGY

Find max

Append

{8, 9, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{8, 9, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{8, 9, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{8, 9, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{8, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{8, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{8, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{8, 6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{6, 1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{1} →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{ } →

Remove

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

GREEDY STRATEGY

Find max

Append

{9, 8, 9, 6, 1} → 99861

Remove

Success!

Find max digit

Append it to the number

Remove it from the list of digits

Repeat while there are digits in the list

QUEUE OF PATIENTS

QUEUE OF PATIENTS

Queue Arrangement	
Input:	n patients have come to the doctor's office at 9:00AM. They can be treated in any order. For i-th patient, the time needed for treatment is t_i. You need to arrange the patients in such a queue that the total waiting time is minimized.
Output:	The minimum total waiting time.

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (1, 2, 3):

- First patient doesn't wait

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (1, 2, 3):

- First patient doesn’t wait
- Second patient waits for 15 minutes

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (1, 2, 3):

- First patient doesn’t wait
- Second patient waits for 15 minutes
- Third patient waits for 15 + 20 = 35 minutes

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (1, 2, 3):

- First patient doesn't wait
- Second patient waits for 15 minutes
- Third patient waits for 15 + 20 = 35 minutes
- Total waiting time 15 + 35 = 50 minutes

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (3, 1, 2):

- First patient doesn’t wait

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (3, 1, 2):

- First patient doesn’t wait
- Second patient waits for 10 minutes

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (3, 1, 2):

- First patient doesn’t wait
- Second patient waits for 10 minutes
- Third patient waits for 10 + 15 = 25 minutes

QUEUE OF PATIENTS

Optimal Queue Arrangement

t_1 =15, t_2 =20 and t_3 =10. Arrangement (3, 1, 2):

- First patient doesn’t wait
- Second patient waits for 10 minutes
- Third patient waits for 10 + 15 = 25 minutes
- Total waiting time 10 + 25 = 35 minutes

GREEDY STRATEGY

- Make some greedy choice
- Reduce to a smaller problem
- Iterate

GREEDY CHOICE

- First treat the patient with the maximum treatment time
- First treat the patient with the minimum treatment time
- First treat the patient with average treatment time

GREEDY ALGORITHM

- First treat the patient with the minimum treatment time

GREEDY ALGORITHM

- First treat the patient with the minimum treatment time
- Remove this patient from the queue

GREEDY ALGORITHM

- First treat the patient with the minimum treatment time
- Remove this patient from the queue
- Treat all the remaining patients in such order as to minimize their total waiting time

SUBPROBLEM

Definition
Subproblem is a similar problem of smaller size.

SUBPROBLEM

Examples
• MaximumSalary(1, 9, 8, 9, 6) =

SUBPROBLEM

Examples

- MaximumSalary(1, 9, 8, 9, 6) =

  ‘‘9’’ + MaximumSalary(1, 8, 9, 6)

SUBPROBLEM

Examples

- MaximumSalary(1, 9, 8, 9, 6) =

  ‘‘9’’ + MaximumSalary(1, 8, 9, 6)
- Minimum total waiting time for n patients = $(n - 1) \cdot t_{min} +$

SUBPROBLEM

Examples

- MaximumSalary(1, 9, 8, 9, 6) =

  ‘‘9’’ + MaximumSalary(1, 8, 9, 6)
- Minimum total waiting time for n patients = $(n - 1) \cdot t_{min}$ + minimum total waiting time for $n - 1$ patients without t_{min}

SAFE CHOICE

Definition
A greedy choice is called **safe choice** if there is an optimal solution consistent with this first choice.

SAFE CHOICE

Lemma
To treat the patient with minimum treatment time t_{min} first is a safe choice.

PROOF IDEA

- Is it possible for an optimal arrangement to have two **consecutive** patients in order with treatment times t_1 and t_2 such that $t_1 > t_2$?

PROOF IDEA

- Is it possible for an optimal arrangement to have two **consecutive** patients in order with treatment times t_1 and t_2 such that $t_1 > t_2$?
- It is impossible. Assume there is such an optimal arrangement and consider what happens if we swap these two patients.

PROOF IDEA

- Is it possible for an optimal arrangement to have two **consecutive** patients in order with treatment times t_1 and t_2 such that $t_1 > t_2$?
- It is impossible. Assume there is such an optimal arrangement and consider what happens if we swap these two patients.
- If we swap two consecutive patients with treatment times t1 > t2: Waiting time for all the patients before and after these two doesn't change

PROOF IDEA

- Is it possible for an optimal arrangement to have two **consecutive** patients in order with treatment times t_1 and t_2 such that $t_1 > t_2$?
- It is impossible. Assume there is such an optimal arrangement and consider what happens if we swap these two patients.
- If we swap two consecutive patients with treatment times t1 > t2: Waiting time for all the patients before and after these two doesn't change
- Waiting time for the patient which was first increases by t_2, and for the second one it decreases by t_1

PROOF IDEA

- Is it possible for an optimal arrangement to have two **consecutive** patients in order with treatment times t_1 and t_2 such that $t_1 > t_2$?
- It is impossible. Assume there is such an optimal arrangement and consider what happens if we swap these two patients.
- If we swap two consecutive patients with treatment times t1 > t2: Waiting time for all the patients before and after these two doesn't change
- Waiting time for the patient which was first increases by t_2, and for the second one it decreases by t_1
- Total waiting time increases by $t_2 - t_1 < 0$, so it actually decreases

PROOF IDEA

We have just proved:

Lemma
In any optimal arrangement of the patients, first of any two consecutive patients has smaller treatment time.

SAFE CHOICE PROOF

- Assume the patient with treatment time t_{min} is not the first

SAFE CHOICE PROOF

- Assume the patient with treatment time t_{min} is not the first
- Let $i > 1$ be the position of the first patient with treatment time t_{min} in the optimal arrangement

SAFE CHOICE PROOF

- Assume the patient with treatment time t_{min} is not the first
- Let $i > 1$ be the position of the first patient with treatment time t_{min} in the optimal arrangement
- Then the patient at position $i - 1$ has bigger treatment time — a contradiction

Conclusion

Now we know that the following greedy algorithm works correctly:

- First treat the patient with the minimum treatment time

Conclusion

Now we know that the following greedy algorithm works correctly:

- First treat the patient with the minimum treatment time
- Remove this patient from the queue

Conclusion

Now we know that the following greedy algorithm works correctly:

- First treat the patient with the minimum treatment time
- Remove this patient from the queue
- Treat all the remaining patients in such order as to minimize their total waiting time

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

MinTotalWaitingTime(t, n)

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
  t_min ← +∞
  minIndex ← 0
  for j from 1 to n:
    if treated[j] == 0 and t[j] < t_min:
      t_min ← t[j]
      minIndex ← j
  waitingTime ← waitingTime + (n − i) · t_min
  treated[minIndex] = 1
return waitingTime
```

Lemma

The running time of MinTotalWaitingTime(t, n) is $O(n^2)$.

Lemma

The running time of MinTotalWaitingTime(t, n) is $O(n^2)$.

Proof

- i changes from 1 to n

Lemma

The running time of MinTotalWaitingTime(t, n) is $O(n^2)$.

Proof

- i changes from 1 to n
- For each value of i, j changes from 1 to n

Lemma

The running time of MinTotalWaitingTime(t, n) is $O(n^2)$.

Proof

- i changes from 1 to n
- For each value of i, j changes from 1 to n
- This results in $O(n^2)$

- Actually, this problem can be solved in time O(n log n)

- Actually, this problem can be solved in time O(n log n)
- Instead of choosing the patient with minimum treatment time out of remaining ones n times, sort patients by increasing treatment time

- Actually, this problem can be solved in time O(n log n)
- Instead of choosing the patient with minimum treatment time out of remaining ones n times, sort patients by increasing treatment time
- This sorted arrangement is optimal

- Actually, this problem can be solved in time O(n log n)
- Instead of choosing the patient with minimum treatment time out of remaining ones n times, sort patients by increasing treatment time
- This sorted arrangement is optimal
- It is possible to sort n patients in time O(n log n)

REDUCTION TO SUBPROBLEM

- Make some first choice
- Then solve a problem of the same kind
- Smaller: fewer digits, fewer patients
- This is called a “subproblem”

SAFE CHOICE

- A choice is called safe if there is an optimal solution consistent with this first choice

SAFE CHOICE

- A choice is called safe if there is an optimal solution consistent with this first choice
- Not all first choices are safe

SAFE CHOICE

- A choice is called safe if there is an optimal solution consistent with this first choice
- Not all first choices are safe
- Greedy choices are often unsafe

GENERAL STRATEGY

Problem

GENERAL STRATEGY

greedy choice

Problem →

- Make a greedy choice

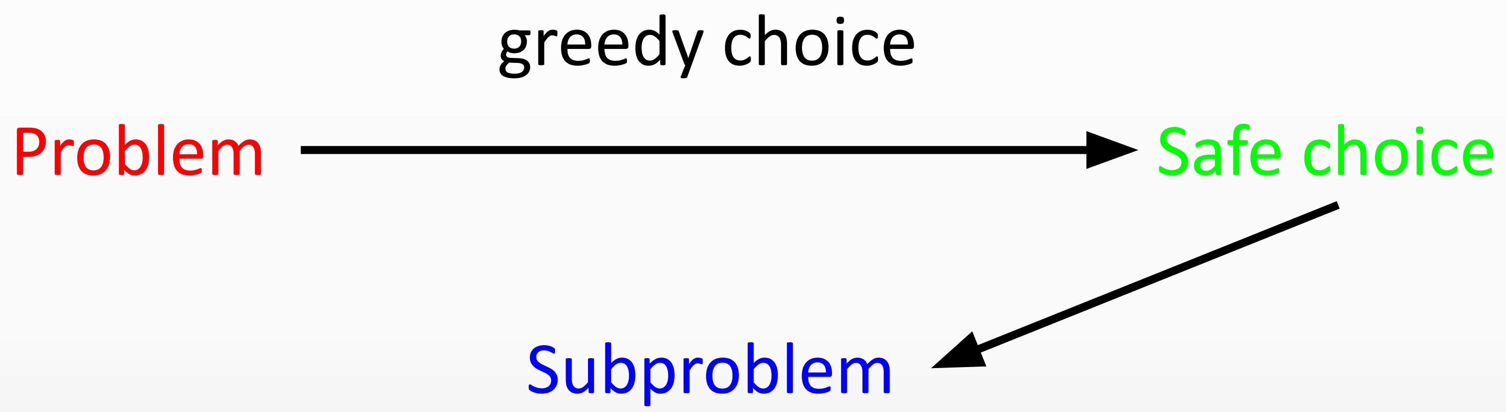
GENERAL STRATEGY

greedy choice

Problem ⟶ Safe choice

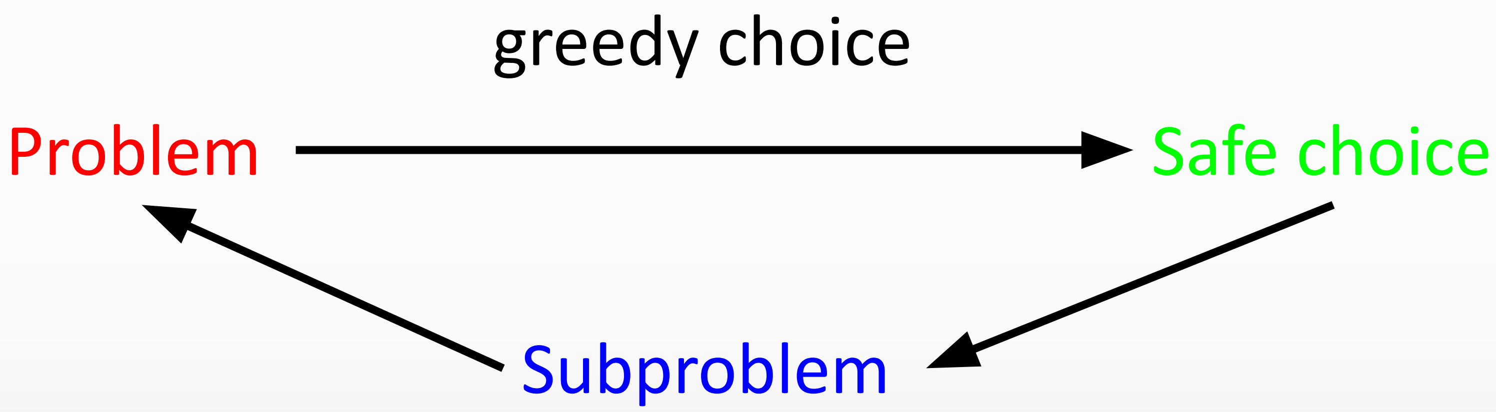
- Make a greedy choice
- Prove that it is a safe choice

GENERAL STRATEGY



- Make a greedy choice
- Prove that it is a safe choice
- Reduce to a subproblem

GENERAL STRATEGY



- Make a greedy choice
- Prove that it is a safe choice
- Reduce to a subproblem
- Solve the subproblem