「知道错了没」

——如何让队友认错

Chris Xiong

2017-07-21

「知道错了没」

Outline

- 采访
- dp: d(ui)p(ai)
- dp: Knapsack problem
- 如何让队友认错之如何殴打队友

采访

~~a.k.a. 教学质量检查~~

- 上次课讲的内容大家都听懂了吗?

采访

~~a.k.a. 教学质量检查~~

- 上次课讲的内容大家都听懂了吗?
- 什么? 不懂?

采访

~~a.k.a. 教学质量检查~~

- 上次课讲的内容大家都听懂了吗?
- 什么? 不懂?
- 那还记得上次讲的什么么吗?

采访

~~a.k.a. 教学质量检查~~

- 上次课讲的内容大家都听懂了吗?
- 什么? 不懂?
- 那还记得上次讲的什么吗?
- A Water Problem

采访

~~a.k.a. 教学质量检查~~

- 上次课讲的内容大家都听懂了吗?
- 什么? 不懂?
- 那还记得上次讲的什么吗?
- A Water Problem
- 已知

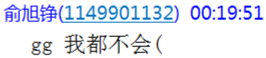
$$f(x+1)=\begin{cases} a & x=0 \\ b & x=1 \\ f(x)+f(x-1)+sin(\frac{\pi x}{2}) & otherwise \end{cases}$$

  对于给定的a, b, n，求$f(n)$。$n \leq 10^{18}$。
- 给大家5分钟的思考时间。

采访

怎么样，是不是不会啊?

- 都怪宇宙智障。



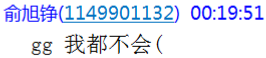
俞旭铮(1149901132) 00:19:51

gg 我都不会(

采访

怎么样，是不是不会啊?

- 都怪宇宙智障。



俞旭铮(1149901132) 00:19:51

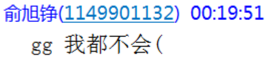
gg 我都不会(

- 提示:

采访

怎么样，是不是不会啊?

- 都怪宇宙智障。



俞旭铮(1149901132) 00:19:51

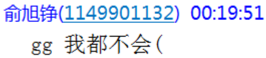
gg 我都不会(

- 提示：周期！！

采访

怎么样，是不是不会啊？

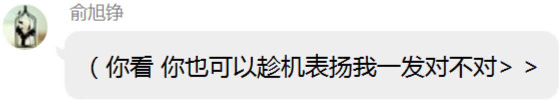
- 都怪宇宙智障。



俞旭铮(1149901132) 00:19:51

gg 我都不会（

- 提示：周期！！
- 还不会的话就去找宇宙智障。



俞旭铮

（你看 你也可以趁机表扬我一发对不对> >

- （听说你想要表扬　厚颜无耻）

d(ui)p(ai)

- WTF is duipai?

d(ui)p(ai)

- WTF is duipai?
- Automated generation of test data and execution of several programs.

d(ui)p(ai)

- WTF is duipai?
- Automated generation of test data and execution of several programs.
- And most importantly, compare their results.

d(ui)p(ai)

- WTF is duipai?
- Automated generation of test data and execution of several programs.
- And most importantly, compare their results.
- ~~A nice way to waste time if you are stuck.~~

A sample script for UNIX-like OS

```
#!/bin/bash
i=0
while(true)
do
        ./170312cgen > test.in
        ./170312ca < test.in > aa.out
        ./170312cb < test.in > bb.out
        diff aa.out bb.out
        if [ $? -ne 0 ]
        then
                break
        fi
        echo $i passed
        let i++
done
```

How to use it?

- Modify the script to your needs.
- Save it as a script, e.g.: "xxx.sh".
- Give it the permission to execute. Run `chmod +x <your_script_name_here>` in a terminal.
- Run it! Type `./<your_script_name_here>` in a terminal.

What does this script do?

- Run the input generator.
- Feed the generated input to the compared program A and gather results from it.
- Do the same thing with program B.
- Check the output. If they differ, terminate the script. Otherwise loop.

Explanation

- `while(true)`
  `do`
  `done`
  `break`
- `> <`
  redirection
- `if`
  `then`
  `fi`
  `$?`
  `[, -ne`
- Verification: `diff` / custom program

Alternative approaches

- Write a `C/C++` program instead of a shell script?
- `system()` in `stdlib.h` (`cstdlib`)
- return value of `system()`
- Windows batch file:
  - `IF %ERRORLEVEL% EQU 0(GOTO :loop)`
- ~~Powershell~~?

Writing input generators

- Random?
- Constructed special cases?

Knapsack problem

I suck at this

- Unbounded knapsack problem
- Bounded knapsack problem
  - 0/1 knapsack problem
- NP-complete!
- A No-Dynamic-Programming-At-All variant

Knapsack problem

The No-DP-At-All variant

Fractional knapsack problem (a.k.a. Continuous knapsack problem)

- A knapsack of capacity W.
- N items, each having its weight w_i and value per unit weight v_i.
- Select an amount x_i of each item so that the total weight doesn't exceed the capacity ($\sum_i x_i \leq W$) and maximizing the total value $\sum_i x_i \times v_i$, where $x_i \in \mathbb{R}, x_i \geq 0$.

Knapsack problem

The No-DP-At-All variant

Fractional knapsack problem (a.k.a. Continuous knapsack problem)

- A knapsack of capacity W.
- N items, each having its weight w_i and value per unit weight v_i.
- Select an amount x_i of each item so that the total weight doesn't exceed the capacity ($\sum\limits_i x_i \leq W$) and maximizing the total value $\sum\limits_i x_i \times v_i$, where $x_i \in \mathbb{R}, x_i \geq 0$.
- Greedy.

Knapsack problem

0/1 knapsack problem

- Still a knapsack of capacity W.
- Still N items, each having its weight w_i and value v_i.
- For each item, determine whether to put it in the knapsack so that the total weight doesn't exceed the capacity and the total value is maximum.

Knapsack problem

0/1 knapsack problem

A brute-force solution:

```
def dfs(i,remaining_capacity):
        if(i==0): return 0;
        if(remaining_capacity<0):
                return -inf;
        r1=dfs(i-1,remaining_capacity);
        r2=dfs(i-1,remaining_capacity-w[i])+v[i];
        return max(r1,r2);
```

- Call dfs(N,W) for answer.
- Each non-trivial invocation of dfs branch into two paths.
- Time complexity: $O(2^N)$.
- A minor optimization: replace the second condition statement with `if(remaining_capacity<w[i]): return dfs(i-1,remaining_capacity);`

Knapsack problem

0/1 knapsack problem

A effective optimization: memoization.

```
f=[[-1 for i in range(N)] for j in range(W)]
def dfs(i,remaining_capacity):
        if(i==0): return 0;
        if(remaining_capacity<w[i]):
                return dfs(i-1,remaining_capacity);
        if(f[i][remaining_capacity]!=-1):
                return f[i][remaining_capacity];
        f[i][remaining_capacity]=max(
                        dfs(i-1,remaining_capacity),
                        dfs(i-1,remaining_capacity-w[i])+v[i]);
        return f[i][remaining_capacity];
```

Knapsack problem

0/1 knapsack problem

- For each parameter tuple of dfs, the function may only branch once.
- Time complexity: $O(NW)$.

  It's a pseudo-polynomial algorithm, so the knapsack problem is still NP-complete.

Knapsack problem

0/1 knapsack problem

- For each parameter tuple of dfs, the function may only branch once.
- Time complexity: $O(NW)$.

  It's a pseudo-polynomial algorithm, so the knapsack problem is still NP-complete.
- Why does this work?

Knapsack problem

0/1 knapsack problem

- For each parameter tuple of dfs, the function may only branch once.
- Time complexity: $O(NW)$.

  It's a pseudo-polynomial algorithm, so the knapsack problem is still NP-complete.
- Why does this work?
- Once the result for a specific parameter tuple has been calculated, will it change any further?
- Non-aftereffect property.

Knapsack problem

0/1 knapsack problem

- For each parameter tuple of dfs, the function may only branch once.
- Time complexity: $O(NW)$.

  It's a pseudo-polynomial algorithm, so the knapsack problem is still NP-complete.
- Why does this work?
- Once the result for a specific parameter tuple has been calculated, will it change any further?
- Non-aftereffect property.
- Recursion? Phooey! That will be a lot of context switches!

Knapsack problem

0/1 knapsack problem

- For each parameter tuple of dfs, the function may only branch once.
- Time complexity: $O(NW)$.

  It's a pseudo-polynomial algorithm, so the knapsack problem is still NP-complete.
- Why does this work?
- Once the result for a specific parameter tuple has been calculated, will it change any further?
- Non-aftereffect property.
- Recursion? Phooey! That will be a lot of context switches!
- Time for some black magic!

Knapsack problem

0/1 knapsack problem

The iteration version.

```
f=[[0 for i in xrange(N)] for j in xrange(W)]
for i in xrange(1,N):
    for j in xrange(0,W):
        f[i][j]=max(f[i-1][j],
                f[i-1][j-w[i]]+v[i] if j>=w[i] else 0);
```

Knapsack problem

0/1 knapsack problem

- Recall that in the memoization version, in order to calculate results for $f[i, remaining\_capacity]$ we must already have at least two results for $f[i-1, x]$.
- Why don't we calculate all $f[i-1, x]$ before calculating $f[i, x]$?

Knapsack problem

0/1 knapsack problem

- Recall that in the memoization version, in order to calculate results for $f[i, remaining\_capacity]$ we must already have at least two results for $f[i-1, x]$.
- Why don't we calculate all $f[i-1, x]$ before calculating $f[i, x]$?
- Got the maximum value now! Want the list of selected items?

Knapsack problem

0/1 knapsack problem

- Recall that in the memoization version, in order to calculate results for $f[i, remaining\_capacity]$ we must already have at least two results for $f[i-1, x]$.
- Why don't we calculate all $f[i-1, x]$ before calculating $f[i, x]$?
- Got the maximum value now! Want the list of selected items?
- Traceback.

Knapsack problem

0/1 knapsack problem

- When we are at $i = x$ of the outer loop, all values in $f[y], y < x - 1$ are no longer used.
- If we don't need to traceback, can we save a bit of memory?

Knapsack problem

0/1 knapsack problem

- When we are at $i = x$ of the outer loop, all values in $f[y], y < x - 1$ are no longer used.
- If we don't need to traceback, can we save a bit of memory?
- Yes! Just throw them away!

```
f=[0 for i in xrange(W)]
for i in xrange(1,N):
        for j in xrange(W,w[i],-1):
                        f[j]=max(f[j],f[j-w[i]]+v[i]);
```

Knapsack problem

0/1 knapsack problem

- How does this work?

Knapsack problem

0/1 knapsack problem

- How does this work?
  ($g[i][j]$ denotes the original $f[i][j]$ from the two dimensional iterative solution.)
- When we are at $j = y$ of the inner loop, $f[0..y]$ are values from $g[i-1]$ and $f[y+1..W]$ contains values from $g[i]$.
- Why reverse the inner loop?

Knapsack problem

0/1 knapsack problem

- How does this work?
  ($g[i][j]$ denotes the original $f[i][j]$ from the two dimensional iterative solution.)
- When we are at $j = y$ of the inner loop, $f[0..y]$ are values from $g[i-1]$ and $f[y+1..W]$ contains values from $g[i]$.
- Why reverse the inner loop?
- Because we still need the values with smaller *remaining_capacity* from the last iteration!

Knapsack problem

Unbounded knapsack problem

- Same as the $0/1$ knapsack problem, but each item has unlimited copies.

Knapsack problem

Unbounded knapsack problem

- Same as the 0/1 knapsack problem, but each item has unlimited copies.
- Converting to 0/1 knapsack problem?

Knapsack problem

Unbounded knapsack problem

- Same as the 0/1 knapsack problem, but each item has unlimited copies.
- Converting to 0/1 knapsack problem?
- Imitating Binary. We can obtaining any multiplicity of items from a combination of 1x, 2x, 4x, 8x, ... of that item.

Knapsack problem

Unbounded knapsack problem

- Same as the 0/1 knapsack problem, but each item has unlimited copies.
- Converting to 0/1 knapsack problem?
- Imitating Binary. We can obtaining any multiplicity of items from a combination of 1x, 2x, 4x, 8x, ... of that item.
- Any other solutions?

Knapsack problem

Unbounded knapsack problem

Another solution:

```
f=[0 for i in xrange(W)]
for i in xrange(1,N):
        for j in xrange(w[i],W):
                f[j]=max(f[j],f[j-w[i]]+v[i]);
```

Wait... Isn't this our final solution for the 0/1 knapsack problem?

Knapsack problem

Unbounded knapsack problem

- Not exactly! Note that the inner loop now iterate from $w[i]$ to W.
- Why?

Knapsack problem

Unbounded knapsack problem

- Not exactly! Note that the inner loop now iterate from $w[i]$ to W.
- Why?
- Let's revisit the reason to iterate in reverse order in 0/1 knapsack problem:
- We still need the values with smaller *remaining_capacity* from the last iteration.

Knapsack problem

Unbounded knapsack problem

- Not exactly! Note that the inner loop now iterate from $w[i]$ to W.
- Why?
- Let's revisit the reason to iterate in reverse order in $0/1$ knapsack problem:
- We still need the values with smaller *remaining_capacity* from the last iteration.
- Why do we need *those values*, instead of the shiney new values we just obtained?

Knapsack problem

Unbounded knapsack problem

- Not exactly! Note that the inner loop now iterate from $w[i]$ to W.
- Why?
- Let's revisit the reason to iterate in reverse order in $0/1$ knapsack problem:
- We still need the values with smaller $remaining\_capacity$ from the last iteration.
- Why do we need *those values*, instead of the shiney new values we just obtained?
- Because these values do not take the current item into consideration, effectively ensuring that every item can be used at most once.
- But now we have unlimited copies of each item!

Knapsack problem

Bounded knapsack problem

- Same as the 0/1 knapsack problem, but each item has C_i copies.
- POJ 1276

Knapsack problem

Bounded knapsack problem

- Same as the 0/1 knapsack problem, but each item has C_i copies.
- POJ 1276
- Still solve by converting to a 0/1 knapsack problem.

Knapsack problem

Bounded knapsack problem

- Same as the 0/1 knapsack problem, but each item has C_i copies.
- POJ 1276
- Still solve by converting to a 0/1 knapsack problem.
- How to limit the maximum number of copies?

Knapsack problem

Bounded knapsack problem

- Same as the 0/1 knapsack problem, but each item has C_i copies.
- POJ 1276
- Still solve by converting to a 0/1 knapsack problem.
- How to limit the maximum number of copies?
- By modifying the largest group so that if all groups are selected, the sum of multiplicity equals to C_i.

Knapsack problem

Bounded knapsack problem

Another "stupid" solution that can also be applied to the unbounded knapsack problem:

- For each item, we have $C_i + 1$ choices.
- We just iterate through these choices to update $f[][]$.
- This solution runs for $O(W\Sigma C_i)$.
- However it can be further optimized to $O(NW)$ using some advanced DP optimization technics.

Knapsack problem

Bounded knapsack problem

Another "stupid" solution that can also be applied to the unbounded knapsack problem:

- For each item, we have $C_i + 1$ choices.
- We just iterate through these choices to update $f[][]$.
- This solution runs for $O(W\Sigma C_i)$.
- However it can be further optimized to $O(NW)$ using some advanced DP optimization technics.
- We are not covering that here today.
- More about knapsack problems: `https://github.com/tianyicui/pack`

如何殴打队友

以下内容仅供娱乐

如何殴打队友

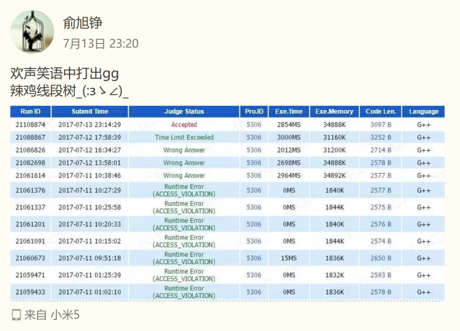
——何时应当考虑殴打队友?

- 当队友占3个小时键盘什么都没写出来时
- 当队友开一题WA一题时
- 当队友开始表演口技时
- 当队友热身赛开可乐洒了一地时
- 当队友看到树就想重心分解时

如何殴打队友

——正题

- 像现在这么殴打（宣传光辉事迹）
- 比赛时不准碰键盘
- 表演口技时录音
- WA一题灌一瓶可乐，不准洒（大家可以算一下光这张图就要喝多少瓶）



俞旭铮
7月13日 23:20

欢声笑语中打出gg
辣鸡线段树_(:3 ゝ∠)_

Run ID	Submit Time	Judge Status	Pro.ID	Exe.Time	Exe.Memory	Code Len.	Language
21108874	2017-07-13 23:14:29	Accepted	5306	2854MS	34888K	3097 B	G++
21088867	2017-07-12 17:58:39	Time Limit Exceeded	5306	3000MS	31160K	3252 B	G++
21086826	2017-07-12 16:34:27	Wrong Answer	5306	2012MS	31200K	2714 B	G++
21082698	2017-07-12 13:58:01	Wrong Answer	5306	2698MS	34888K	2578 B	G++
21061614	2017-07-11 10:38:46	Wrong Answer	5306	2964MS	34892K	2577 B	G++
21061376	2017-07-11 10:27:29	Runtime Error (ACCESS_VIOLATION)	5306	0MS	1840K	2577 B	G++
21061337	2017-07-11 10:25:58	Runtime Error (ACCESS_VIOLATION)	5306	0MS	1844K	2575 B	G++
21061201	2017-07-11 10:20:33	Runtime Error (ACCESS_VIOLATION)	5306	0MS	1840K	2576 B	G++
21061091	2017-07-11 10:15:02	Runtime Error (ACCESS_VIOLATION)	5306	0MS	1844K	2574 B	G++
21060673	2017-07-11 09:51:18	Runtime Error (ACCESS_VIOLATION)	5306	15MS	1836K	2650 B	G++
21059471	2017-07-11 01:25:39	Runtime Error (ACCESS_VIOLATION)	5306	0MS	1832K	2593 B	G++
21059433	2017-07-11 01:02:10	Runtime Error (ACCESS_VIOLATION)	5306	0MS	1836K	2578 B	G++

来自 小米5

如何殴打队友

——殴打队友时需要注意的地方

- 注意殴打的度——虽然原则上是越重越好，但是如果你的队友是个卜力星人，殴打太重会导致其发射大量宇宙射线，导致「伤敌800，自损1000」的尴尬情形。
- 殴打方式要适当。比如其在表演口技不应该使用灌可乐的手法，因为容易洒一地。
- 适可而止。如果感觉队友能A题了就让其施展一发（没A就接着灌）。

如何殴打队友

Bonus：利用宇宙射线

如果你发现你的队友会发射宇宙射线，那么它可能是可以被利用的。可利用的宇宙射线的发射者是会认错的。这里有一个正面例子和一个反面例子：

- 黄焖蓉　：发射射线导致临近的队伍接连两次CE。
- 宇宙智障：发射射线导致队友高数全部忘光。

如你所见，第一类射线是可以加以利用的；而第二类射线则是「射别人一个也射不中，射自己人一射一个准」的。大家要尽量做好对第二类射线的防护工作。关于这个问题我们下次再说（如果还有下次机会的话）。

如何殴打队友

So... what’s the point?

- 合理利用时间
- 卡题时的处理方式
- 队内的合作
- 其他队伍的影响