

1

## Team selection puzzle



- Imagine that your friend is a project manager, and her eam consist of great programmers - if only she could mit an in two shill
-She decides smaller teams
- To minimize fighting within each team.
- She knows who fights with whom (the "CONFLICT
relation"), but how can she do the splitting?
relation"), but how can she do the splitting?
- And is it possible at all to eliminate at least half
the conflicts? If not, why bother...
- Do you think it is possible to split any group into two teams
- to eliminate all conflicts?
- How about eliminating half the conflicts?
- How would you do the splitting?

Suppose this is the graph of the CONFLICT relation for a group. - Here, lines are double-direction arrows, since CONFLICT is symmetric.
 - What do you think is the best split?

3
4


5

## Team selection puzzle



Do you think it is possible to split any group into two teams

- How about eliminating half the conflicts?

In terms of the CONFLICT graph

- Is there a way to split vertices into two groups so that at least half of the edges go between groups?

Graph of the CONFLICT relation - Symmetric and anti-reflexive

- Drawn as an undirected graph


6

## Team selection puzzle <br> 

- Is there a way to split vertices into two groups so that at least half of the edges go between groups?
- Suppose that the probability of every person to be on Team Orange vs. Team Green is $1 / 2$.
- Then the probability for each edge (conflict) to be between people assigned to different teams is: $1 / 2$
- Expected number of edges to go between different colours: $1 / 2$ - total number of edges.
- Therefore, there exists a way to split any group into two teams so that at least half of the conflicts are between people on different teams!

7
Graph of the CONFLICT relation - Symmetric and anti-reflexive

- Drawn as an undirected graph


9


## Team selection puzzle <br> 4

- There exists a way to split any group into two teams so that at least half of the conflicts are between people on different teams

Graph of the CONFLICT relation - Symmetric and anti-reflexive - Drawn as an undirected graph

- But how would we do this split?
- Let's do something random!
- Just assign each person to team Green or team Orange at random with probability $1 / 2$
- So this proof not only tells us that such a split exists, but also gives a (randomized) algorithm that finds it.

- Can prove that it finds it with decent probability
- Can be even made deterministic (that is, it is possible to make
the choices without randomness).

8


10

## How else does it all relate to programming?

You will be using logic to describe what a program should be doing
and proofs, in particular induction to show that it does that correctly

- And occasionally to get algorithms

Sets, relations and especially graphs to model problems and optimize performance.
Recurrence relations to give a guarantee on long it takes in the worst case. With O-notation to talk about the time.

- And combinatorics to calculate possible options.
and probabilities/expectation to estimate how long it might take on average. - And what is a probability of success of a randomized algorithm

Example: search in an array


- Given:
- an array A containing $n$ elements,
- and a specific item $x$ ?
- Goal: find the index of $x$ in $A$, if $x$ is in $A$.
- Which box contains ${ }_{3}$ ? Box 4 .
- Precondition: A is an array containing x


## Algorithm arraySearch $(A, x)$

Input array $\boldsymbol{A}$ of $\boldsymbol{n}$ integers, number x
Output $k$ such that $A[k]=x$
$i=0$
out $=-1$
while out $<0$ do if $A[i]=x$ then out $=i$ $i=i+1$
return out

- Postcondition: Returned k such that $\mathrm{A}[\mathrm{k}]=\mathrm{x}$


## Logic: specifications

- Precondition: what should be true before a piece of code (or the whole algorithm) starts
- E.g.: A is an array of numbers and A is not empty and x is a number.
- Postcondition: what should be true after a program (piece of code) finished.
- E.g. If the program returned value $k$, then $A[k]=x$
- or $\mathrm{k}=-1$, if x is not in A .

15

99
The good news about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do.

Ted Nelson

## Correctness of algorithms

- Prove that if the program starts with precondition being true, it ends with postcondition being true
- If x is in the array, then the program should return its index.
- If there is a loop, prove its correctness by induction - Called loop invariant
- If the program is recursive, prove its correctness by strong induction
- If all recursive calls return correct answers, the program returns a correct answer.


16

## *

- Precondition: A is an array containing x: $\exists i A[i]=x$

[^0]```
Algorithm arraySearch(A,x)
    Input array }\boldsymbol{A}\mathrm{ of }\boldsymbol{n}\mathrm{ integers, number x
    Output k such that A[k]=x
\existsi\in{0\ldotsn-1} A[i]=x
i=0
i=0
\existsi\in{0\ldotsn-1} A[i]=x\wedgei=0^out = - 1
while out < 0 do
    if }A[i]=x\mathrm{ then
        c
        out] = x
return out
```

Program returned $k$ such that $\mathrm{A}[\mathrm{k}]=\mathrm{x}$

18

- Loop invariant: a condition that is true on each iteration of the loop
- Implied by loop precondition
- Implies the loop postcondition
- Implies next loop iteration is correct
- $\mathrm{I}(\mathrm{k}): i=k \wedge(($ out $=i \wedge A[$ out $]=x) \vee(\exists j>i A[j]=x))$
- Guard condition: condition in the while loop - G= "out <0"

Loop is correct when:
Termination: proof that $\exists \mathrm{k}_{0}$ such that after $\mathrm{k}_{0}$ iterations G becomes false

- for all $\mathrm{k}, \mathrm{G} \wedge \mathrm{I}(\mathrm{k}) \rightarrow \mathrm{I}(\mathrm{k}+1)$
- If $\mathrm{k}_{0}$ is the smallest number such that $\neg G$,
- then $\neg G \wedge I\left(k_{0}\right) \rightarrow$ postcondition

19

Proving the loop invariant by induction on i :

- Base case: I(0)

$$
\begin{aligned}
& -\exists i \in\{0 \ldots n-1\} A[i]=x \wedge i=0 \wedge \text { out }=-1 \\
& \quad \text { Implies } 1(0) \\
& -i=0 \wedge((\text { out }=0 \wedge A[\text { out }]=x) \vee(\exists j>i A[j]=x))
\end{aligned}
$$

```
Mhile out< < do 
    i=i+I
A[out] = x
\(4[\) out \(]=x\)
```

- Assume I(k): $i=k \wedge((o u t=i \wedge A[$ out $]=x) \vee(\exists j>i A[j]=x))$
- Show: if $G$, then $\mathrm{I}(\mathrm{k}+1): i=k+1 \wedge(($ out $=i \wedge A[$ out $]=x) \vee(\exists j>i A[j]=x))$
- $i=k+1$ because of " $i=i+1$ " statement

If $\mathrm{A}[\mathrm{i}]=\mathrm{x}$, then (out $=i \wedge A[$ out $]=x$ ) holds

- Otherwise, $(\exists j>i \quad A[j]=x)$ holds.
- Otherwise, if $\neg G$, postcondition holds:
- in this case, (out $=i \wedge A[o u t]=x$ ) should have been true in $\mathrm{l}(\mathrm{k})$, for $\mathrm{i}=\mathrm{k}$.
- So A[out]=x

20


21


22

## Running time: worst case

- Precondition: A is an array containing x
- Therefore, in the worst scenario need to check all $n$ boxes A[i]
- Running time: $O(n)$
- Ignoring how many steps exactly are inside of the loop, as long as it is constant.


Algorithm array Search $(A, x)$
Input array $A$ of $n$ inte
Agorinm arraysearch $(A, x)$
Input array $A$ of $n$ integers, number $x$
Output $k$ s. Output k such that $A[k]=$
$i=0$
out $=-1$
out $=-1$
while out $<\theta$ do
if $A[i]=x$ then
out $=i$ ${ }_{i=i+1}^{\text {out }}$

- Suppose there is just one $x$ in $A$
- Probability of finding $x$ in each step is $\frac{1}{n}$
- Let random variable X denote the number of loop iterations till $x$ is found
- $E(X)=\sum_{i=1}^{n} i * \operatorname{Pr}(X=i)=\frac{1}{n} \sum_{i=1}^{n} i=(n+1) / 2$
- Expect to find $x$ roughly in the middle of $A$
- Running time $O(n)$


25


[^0]:    Algorithm arraySearch(A, $x$ )
    Input array $\boldsymbol{A}$ of $\boldsymbol{n}$ integers, number x
    Output k such that $\mathrm{A}[\mathrm{k}]=\mathrm{x}$
    $i=0$
    out $=-1$
    while out $<0$ do
    if $A[i]=x$ then
    out $=i$
    $i=i+1$
    return out

    - Postcondition: Returned k such that $\mathrm{A}[\mathrm{k}]=\mathrm{x}: A[k]=x$

