Lecture02

CS 350 – Algorithms and Complexity

Paul Doliotis
Adjunct Assistant Professor
Portland State University

Why Algorithms? An Example

- Let’s discuss a problem that software engineers have to solve when developing a web search engine.
- Every day, there is a list A of web pages that have already been visited.
  - "visiting" a web page means that our program has downloaded that web page and processed it, so that it can show up in search results.
- Every day, there is also a list B of links to web pages that are still not processed.

Why Algorithms? An Example

- Question: which links in list B are NOT in A?
- Why is this a useful question?

Why Algorithms? An Example

- Question: which links in list B are NOT in A?
- Why is this a useful question?
  - Most links in B had already been seen in A.
  - It is a **huge waste of resources** to revisit those links

Why Algorithms? An Example

- Recap:
  - A set A of items
  - A set B of items
  - Define setdiff(B, A) to be the set of items in B that are not in A.
- Question: how do we compute setdiff(B, A).
- Any ideas?

setdiff(B, A) – First Version

```
setdiff(B, A):
   result = empty set
   for each item b of B:
      found = false
      for each item a of A:
         if (b == a) then found = true
      if (found == false) add b to result
   return result.
```

- What can we say about how fast this would run?

setdiff(B, A) – First Version

```
setdiff(B, A):
    result = empty set
    for each item b of B:
        for each item a of A:
            if (b == a) then add b to result
    return result.
```

- This needs to compare each item of B with each item of A.
- If we denote the size of B as |B|, and the size of A as |A|, we need |B| * |A| comparisons.

setdiff(B, A) – First Version

```
setdiff(B, A):
   result = empty set
   for each item b of B:
      for each item a of A:
         if (b == a) then add b to result
   return result.
```

- This needs to compare each item of B with each item of A.
- If we denote the size of B as |B|, and the size of A as |A|, we need |B| * |A| comparisons.
- This is our first analysis of **time complexity**.

setdiff(B, A) – First Version - Speed

- We need to perform |B| * |A| comparisons.
- What does this mean in practice?
- Suppose A has 1 billion items.
- Suppose B has 1 million items.
- We need to do 1 quadrilion comparisons.

setdiff(B, A) – First Version - Speed

- We need to perform |B| * |A| comparisons.
- What does this mean in practice?
- Suppose A has 1 billion items.
- Suppose B has 1 million items.
- We need to do 1 quadrilion comparisons.
- On a computer that can do 1 billion comparisons per second, this would take 11.6 days.
  - This is very optimistic, in practice, it would be at least several months.
  - **CAN WE DO BETTER?**

setdiff(B, A) – Second Version

```
setdiff(B, A):
   result = empty set
   sort A and B in alphabetical order
   i = 0; j = 0
   while (i < size(B)) and (j < size(A)):
      if (B[i] < A[j]) then:
         add B[i] to the result
         i = i+1
      else if (B[i] > a[i]) then j = j+1
      else i = i+1; j = j+1
   while i < size(B):
         add B[i] to result
         i = i+1
   return result
```

Application to an Example

- Suppose:
  - B = {January, February, March, April, May, June, July, August, September, October, November, December}
  - A = {May, August, June, July}
- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = August, B[i] = April.
  - B[i] < A[j]
  - we add B[i] to the result
  - i increases by 1.
- result = {April}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = August, B[i] = August.
  - B[i] equals A[j]
  - i and j both increase by 1.
- result = {April}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = July, B[i] = December.
  - B[i] < A[j]
  - we add B[i] to the result
  - i increases by 1.
- result = {April, December}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = July, B[i] = February.
  - B[i] < A[j]
  - we add B[i] to the result
  - i increases by 1.
- result = {August, December, February}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = July, B[i] = January.
  - B[i] < A[j]
  - we add B[i] to the result
  - i increases by 1.
- result = {August, December, February, January}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = July, B[i] = July.
  - B[i] equals A[j]
  - i and j both increase by 1.
- result = {August, December, February, January}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = June, B[i] = June.
  - B[i] equals A[j]
  - i and j both increase by 1.
- result = {August, December, February, January}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = May, B[i] = March.
  - B[i] < A[j]
  - we add B[i] to the result
  - i increases by 1.
- result = {August, December, February, January, March}

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- A[j] = May, B[i] = May.
  - B[i] equals A[j]
  - i and j both increase by 1.
- result = {August, December, February, January, March}
- What happens next?

Application to an Example

- After sorting in alphabetical order:
  - B = {April, August, December, February, January, July, June, March, May, November, October, September}
  - A = {August, July, June, May}
- We have reached the end of A.
- We add to result the remaining items of B.
- result = {August, December, February, January, March, November, October, September}
- We are done!!!

setdiff(B, A) – Second Version

```
setdiff(B, A):
   result = empty set
   sort A and B in alphabetical order
   i = 0; j = 0
   while (i < size(B)) and (j < size(A)):
      if (B[i] < A[j]) then:
         add B[i] to the result
         i = i+1
      else if (B[i] > a[i]) then j = j+1
      else i = i+1; j = j+1
   while i < size(B):
         add B[i] to result
         i = i+1
   return result
```

- What can we say about its speed? What takes time?

setdiff(B, A) – Second Version - Speed

```
setdiff(B, A):
   result = empty set
   sort A and B in alphabetical order
   i = 0; j = 0
   while (i < size(B)) and (j < size(A)):
      if (B[i] < A[j]) then:
         add B[i] to the result
         i = i+1
      else if (B[i] > a[i]) then j = j+1
      else i = i+1; j = j+1
   while i < size(B):
         add B[i] to result
         i = i+1
   return result
```

- we need to: sort A and B, and execute the while loops.

setdiff(B, A) – Second Version - Speed

- We need to:
  - sort A
  - sort B
  - execute the while loops.
- How many calculations it takes to sort A?
  - We will learn in this class that the number of calculations is |A| * log(|A|) * some unspecified constant.
- How many iterations do the while loops take?
  - no more than |A| + |B|.

setdiff(B, A) – Second Version - Speed

- We will skip some details, since this is just an introductory example.
  - By the end of the course, you will be able to fill in those details.
- It turns out that the number of calculations is proportional to $|A|\log(|A|) + |B|\log(|B|)$.
  - Unless stated otherwise, all logarithms in this course will be base 2.

setdiff(B, A) – Second Version - Speed

- It turns out that the number of calculations is proportional to $|A|\log(|A|) + |B|\log(|B|)$.
- Suppose A has 1 billion items.
  - $\log(|A|)$ = about 30.
- We need to do at least 30 billion calculations (unrealistically optimistic).
- On a computer that can do 1 billion calculations per second, this would take 30 seconds.
  - This is very optimistic, but compare to optimistic estimate of 11.6 days for first version of setdiff.
  - in practice, it would be some minutes, possibly hours, but compare to several months or more for first version.

setdiff(B, A) – Third Version

- Use Hash Tables.
- At this point, you are not supposed to know what hash tables are.
- By the end of the course, you should be able to implement and evaluate all three versions.

Programming Skills vs. Algorithmic Skills

- The setdiff example illustrates the difference between programming skills and algorithmic skills.
- Before taking this course, if faced with the setdiff problem, you should ideally be able to:
  – come up with the first version of the algorithm.
  – implement that version.
- After taking this course, you should be able to come up with the second and third versions, and implement them.

Programming Skills vs. Algorithmic Skills

- A large number of real-world problems are simply impossible to solve without solid algorithmic skills.
  - A small selection of examples: computer and cell phone networks, GPS navigation, search engines, web-based financial transactions, file compression, digital cable TV, digital music and video players, speech recognition, automatic translation, computer games, spell-checking, movie special effects, robotics, spam filtering, …
- Good algorithmic skills give you the ability to work on many really interesting software-related tasks.
- Good algorithmic skills give you the ability to do more scientific-oriented computer-related work.