Data Structures and Algorithms
(資料結構與演算法)

Lecture 1: Algorithm

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

definition of algorithm

Name Origin of Algorithm

Muhammad ibn Mūsā al-Ḵwārizmī on a Soviet Union stamp

figure licensed from public domain via

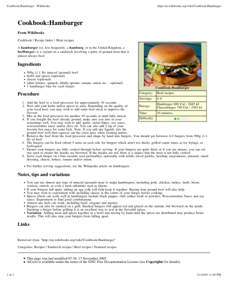
`commons.wikimedia.org/wiki/File:1983_CPA_5426.jpg`

algorithm

- named after **al-Ḵwārizmī** (780–850), Persian mathematician and father of algebra
- algebra: rules to calculate with symbols
- algorithm: instructions to compute with variables

algorithm: recipe-like instructions for computing

Recipe for Cooking Dish



Cookbook:Hamburger - Wikibooks http://en.wikibooks.org/wiki/Cookbook:Hamburger

Cookbook:Hamburger

From Wikibooks

Cookbook | Recipe Index | Meat recipes

A **hamburger** (or, less frequently, a **hamburg**, or in the United Kingdom, a **beefburger**) is a variant on a sandwich involving a patty of ground meat that is almost always beef.

Cookbook:Hamburger

Category:	Beef recipes
Servings:	4-6
Energy:	Hamburger 680 Cal / 2845 kJ Cheeseburger 790 Cal / 3305 kJ
Time:	20 minutes
Difficulty:	

Ingredients

- 500g (1.1 lb) minced (ground) beef
- herbs and spices (optional)
- cheese (optional)
- salad (lettuce, spinach, alfalfa sprouts, tomato, onion etc. - optional)
- 1 hamburger bun for each burger

Procedure

1. Add the beef to a food processor for approximately 10 seconds.
2. Now add your herbs and/or spices to taste. Depending on the quality of your local beef, you may wish to add some beef stock to improve the flavour.
3. Mix in the food processor for another 30 seconds or until fully mixed.
4. If you bought the beef already ground, make sure you mix in your seasonings well. You may wish to add: garlic, onion flakes, soy sauce, worcestershire sauce and/or olive oil. You can also add 2 tsp of your favorite hot sauce for some kick. Note: If you add any liquids, mix the ground beef well then squeeze out the extra juice when forming patties.
5. Remove the beef from the food processor and shape by hand into burgers. You should get between 4-6 burgers from 500g (1.1 lb) of beef.
6. The burgers can be fried (about 5 mins on each side for burgers which aren't too thick), grilled (same times as for frying), or barbequed.
7. Ensure your burgers are fully cooked through before serving. If your burgers are quite thick or if you are unsure, you can cut one open to ensure the insides are browned. If the insides are red, there is a chance that the meat is not fully cooked.
8. Serve each burger on a bun (sesame seed preferably), optionally with relish, sliced pickles, ketchup, mayonnaise, mustard, ranch dressing, cheese, lettuce, tomato and/or onion.

- For further serving suggestions, see the Wikipedia article on hamburgers.

Notes, tips and variations

- You can use almost any type of minced (ground) meat to make hamburgers, including pork, chicken, turkey, lamb, bison, venison, ostrich, or even a meat substitute such as Quorn.
- If your burgers fall apart, adding an egg yolk will help keep it together. Buying lean ground beef will also help.
- You may wish to experiment with including cheese in the centre of your burger before cooking.
- Spices which can work well in hamburgers include black pepper, chili (either fresh or powder), Worcestershire Sauce and soy sauce. Experiment to find good combinations.
- Almost any herb can work, including basil, oregano and parsley.
- Burgers can also be smoked on a grill. Smoked burgers will appear red and glazed on the outside, but browned on the inside. Smoking a burger before grilling it is an excellent way to seal in the flavorful juices.
- **Variation**: Adding meat and spices together in a bowl and mixing by hand until the spices are distributed may produce better results. This will also stop your burgers from falling apart.

Links

Retrieved from "http://en.wikibooks.org/wiki/Cookbook:Hamburger"

Categories: Recipes | Sandwich recipes | Beef recipes | Featured recipes

- This page was last modified 07:38, 17 November 2005.
- All text is available under the terms of the GNU Free Documentation License (see **Copyrights** for details).

1 of 1 11/16/05 11:50 PM

a recipe for hamburger on Wikibooks

figure by Gentgeen,

licensed under CC BY-SA 3.0 via Wikimedia Commons

recipe

Wikipedia: *a set of instructions that describes how to prepare or make something, especially a dish of prepared food*

recipe: instructions to complete a (cooking) task

Sheet Music for Playing Instrument

first page of the manuscript of Bach's lute suite in G minor

figure licensed as public domain via

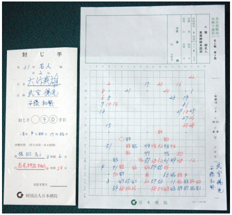
`commons.wikimedia.org/wiki/File:Bachlut1.png`

sheet music

Wikipedia: *handwritten or printed form of musical notation ... to indicate the pitches, rhythms or chords of a song*

sheet music: instructions to play instrument (well)

Kifu for Playing Go



a Japanese kifu

figure by Velobici,

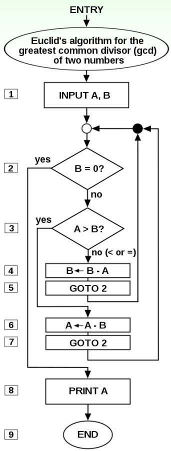
licensed under CC BY-SA 4.0 via Wikimedia Commons

kifu

go game record of steps that describe how the game had been played

kifu: instructions to mimic/learn to play go (professionally)

Algorithm for Computing



flowchart of Euclid's algorithm for calculating the greatest common divisor (g.c.d.) of two numbers

figure by Somepics, licensed under CC BY-SA 4.0 via Wikimedia Commons

algorithm

Wikipedia: *algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation*

algorithm $\sim$ computing recipe: (computable) instructions to solve a computing task efficiently/correctly

pseudo code of algorithm

Pseudo Code for GETMININDEX

C Version

```c
/* return index to min. element
   in arr[0] ... arr[len-1] */
int getMinIndex
    (int arr[], int len){
  int i;
  int m=0;
  for(i=0;i<len;i++){
    if (arr[m] > arr[i]){
      m = i;
    }
  }
  return m;
}
```

Pseudo Code Version

GET-MIN-INDEX(A)

```
m = 1
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

pseudo code: spoken language of programming

Bad Pseudo Code: Too Detailed

Unnecessarily Detailed

```
GET-MIN-INDEX(A)
m = 1
for i = 2 to A.length
        // update if i-th element smaller
        Am = A[m]
        Ai = A[i]
        if Am > Ai
                m = i
        else
                m = m
return m
```

Concise

```
GET-MIN-INDEX(A)
m = 1
for i = 2 to A.length
        // update if i-th element smaller
        if A[m] > A[i]
                m = i
return m
```

goal of pseudo code: communicate efficiently

Bad Pseudo Code: Too Mysterious

Unnecessarily Mysterious

```
GET-MIN-INDEX(A)
x = 1
for xx = 2 to A.length

        if A[x] > A[xx]
                xx = x
return xx
```

Clear

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
        // update if i-th element smaller
        if A[m] > A[i]
                m = i
return m
```

goal of pseudo code: communicate correctly

Bad Pseudo Code: Too Abstract

Unnecessarily Abstract

GET-MIN-INDEX(A)

$m = 1$ // store current min. index
run a loop through A
that updates m in every iteration
return m

Concrete

GET-MIN-INDEX(A)

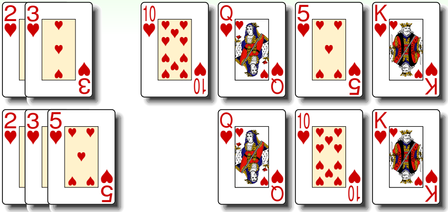
$m = 1$ // store current min. index
for $i = 2$ **to** $A.length$
    // update if i-th element smaller
    if $A[m] > A[i]$
        $m = i$
return m

goal of pseudo code: communicate effectively

From GET-MIN-INDEX to SELECTION-SORT

GET-MIN-INDEX(A, ℓ, r)

```
m = ℓ // store current min. index
for i = ℓ + 1 to r
        // update if i-th element smaller
        if A[m] > A[i]
                m = i
return m
```



Good Pseudo Code

- modularize, just like coding
- depends on speaker/listener
- usually no formal definition

SELECTION-SORT(A)

```
for i = 1 to A.length
        m = GET-MIN-INDEX(A, i, A.length))
        SWAP(A[i], A[m])
return A // which has been sorted in place
```

follow any textbook if you really need a definition

criteria of algorithm

Criteria of Recipe

figure by Larry, licensed under CC BY-NC-ND 2.0 via Flickr

Cocktail Recipe: Screwdriver

inputs: 5 cl vodka, 10 cl orange juice

1. mix inputs in a highball glass with ice
2. garnish with orange slice and serve

output: a glass of delicious cocktail

- input: ingredients
- definiteness: clear instructions
- effectiveness: feasible instructions
- finiteness: completable instructions
- output: delicious drink

algorithm $\sim$ recipe: same five criteria for algorithm

(Knuth, The Art of Computer Programming)

Input of Algorithm

. . . quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects. (Knuth, TAOCP)

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
        // update if i-th element smaller
        if A[m] > A[i]
                m = i
return m
```

one algorithm, many uses (on different legal inputs)

Definiteness of Algorithm

Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously & unambiguously specified. (Knuth, TAOCP)

Clear

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

Ambiguous

```
GET-ZERO-INDEX(A)

for i = 1 to A.length

    if A[m] is almost zero
        return m
// what to return here?
```

definiteness: clarity of algorithm

Effectiveness of Algorithm

> *. . . all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using paper and pencil.* (Knuth, TAOCP)

Effective

GET-MIN-INDEX(A)

$m = 1$ // store current min. index
for $i = 2$ **to** $A.length$
    // update if i-th element smaller
    if $A[m] > A[i]$
        $m = i$
return m

Ineffective

GET-SOFT-MIN(A)

$s = 0$ // sum of exponentiated values
for $i = 1$ **to** $A.length$
    $s = s + \exp(-A[i] \cdot 1126)$
return $-\log(s)/1126$

floating point errors may make some steps ineffective on some computers

Finiteness of Algorithm

An algorithm must always terminate after a finite number of steps ... a very finite number, a reasonable number. (Knuth, TAOCP)

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
        // update if i-th element smaller
        if A[m] > A[i]
                m = i
return m
```

finiteness (& efficiency): often requiring analysis for sophisticated algorithms (to be taught later)

Output of Algorithm

. . . *quantities which have a specified relation to the inputs* (Knuth, TAOCP)

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

output (correctness): needs proving
with respect to requirements

correctness proof of algorithm

Claim

figure by Nick Youngson, licensed CC BY-SA 3.0 via Picpedia.Org

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

Correctness of GET-MIN-INDEX

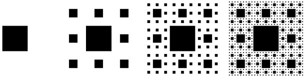
Upon exiting GET-MIN-INDEX(A),

$$A[m] = \min_{1 \leq j \leq n} A[j]$$

with $n = A.length$

claim: mathematical statement that declares correctness

Invariant



invariants when constructing fractals figures by Johannes Rössel,

Correctness of GET-MIN-INDEX

Upon exiting GET-MIN-INDEX(A),

$$A[m] = \min_{1 \le j \le n} A[j]$$

with $n = A.length$

$\Uparrow$

Invariant within GET-MIN-INDEX

Upon finishing the loop with $i = k$, denote m by m_k,

$$A[m_k] \le A[j] \text{ for } j = 1, 2, \ldots, k$$

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

(loop) invariant: property that algorithm maintains

Proof of Loop Invariant

Mathematical Induction

Base

when $i = 2$, invariant true because …

assume invariant true for $i = t - 1$;
when $i = t$,

- $m_t = t$ if $A[m_{t-1}] > A[t]$
  - $A[m_t] = A[t] \leq A[t]$
  - $A[m_t] < A[m_{t-1}] \leq A[j]$ for other j
- $m_t = m_{t-1}$ if $A[m_{t-1}] \leq A[t]$
  - $A[m_t] = A[m_{t-1}] \leq A[t]$
  - $A[m_t] = A[m_{t-1}] \leq A[j]$ for other j

—by mathematical induction,
invariant true for $i = 2, 3, \ldots, k$

$\Rightarrow$

```
GET-MIN-INDEX(A)
m = 1 // store current min. index
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

Correctness of GET-MIN-INDEX

$\Uparrow$

Invariant within GET-MIN-INDEX

Upon finishing the loop with $i = k$, denote m by m_k,

$$A[m_k] \leq A[j] \text{ for } j = 1, 2, \ldots, k$$

proof of (loop) invariants $\Rightarrow$ correctness claim of algorithm

Summary

Lecture 1: Algorithm

- definition of algorithm
  instructions to complete a task by computer
- pseudo code of algorithm
  communicate alg. efficiently/correctly/effectively
- criteria of algorithm
  input, definite, effective, finite, output
- correctness proof of algorithm
  from (loop) invariants to claims