

Data Structures and Algorithms II

Lecture 16: More Geometric Algorithms

- More feedback on assignment 1
- Finding Convex Hull

1

```
bool compress(istream in, ostream out)
// Reads characters from in, and writes out
// compressed version in out
```

Latter should work for both keyboard and file data entry/output.

```
success = compress(cin, cout);
```

- Same points apply to string search

```
int kmpsearch(string searchterm, string docu
// If searchterm is found in document, return
// position in document where found, else
// returns -1
```

- Most people put the various functions in separate files, and listed the function prototypes in the main program.
- It is better to make more use of .h files, so don't have to keep listing function prototypes whenever you want to use that function.

3

Functions, arguments, and header files

- When writing functions, always consider how they can be written to be most re-useable for slightly different purposes.
- A function:

```
void compress(string s)
{
    ifstream output("dna.txt");
    ...
}
```

can only be used to output a compressed version of a string to a specific file!

- Instead, you could either:

```
bool compress(string s, string c)
// Creates compressed version of string s
// in string c, and returns True if
// process succeeds.
```

or

2

```
// kmp.h

#ifndef KMP_H
#define KMP_H

int* next(string searchterm)
// returns a next table array for searchterm

int kmpsearch(string searchterm, string docu
// If searchterm is found in document, return
// position in document where found, else
// returns -1
#endif
```

```
-----

// main.cpp
..
#include "kmp.h"
```

- The .h file is the “publicly accessible” interface to your code. If someone else wanted to use your code they could just use the .h file, and your OBJECT (compiled) code, never looking in your (e.g.) kmp.cpp file.

4

- The .h file therefore should contain the documentation someone would need to use your code (e.g., preconditions)

```
// compress.h
...
bool compress(istream in, ostream out)
// PRE: in and out have both been
//       successfully opened.
// PRE: istream is a stream which
//       contains a sequence of possibly
//       repeating characters, to be read
//       character by character.
// ACTION: characters are read from istream,
//         and a compressed version written
//         to ostream.
// RETURNS: a boolean indicating if process
//          completed successfully.
```

- In general, code worked, it was just often not very re-useable..
- But just because something works, don't assume it's right! e.g.,

5

Labs etc

- Last lab next Monday - anyone can come, but may be limited support.
- Assignment due in next Tuesday.
- Last lecture Thursday; Friday will look at past exam Qs.

7

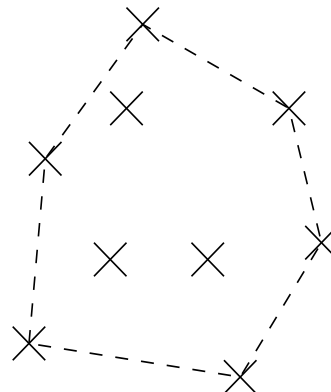
```
int kmp(string s1, string s2)
{
    int* n = initnext(s1);
    ...
}
int* initnext(string s)
{
    int next[20];
    ...
    return next;
}
```

- Testing: Briefly describe unit testing (e.g., initnext) as well as testing of whole.
 - A few people had progs that appeared to work, but which (e.g.,) did not generate a correct next table.
 - Lots of people had a version of "inexact KMP" which would miss certain cases (e.g., AXAAAB in AAAAAAAB, would miss the first match).

6

Back to Geometric Algs: Finding the Convex Hull

Convex hull is natural boundary of set of points. e.g., shortest fence to enclose group of trees.

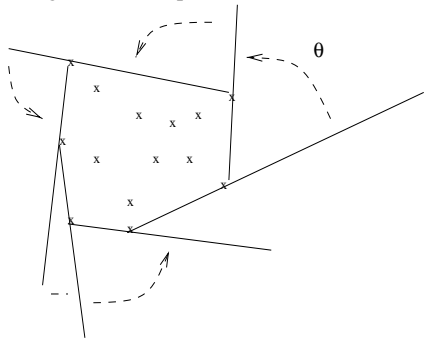


Convex hull is convex polygon enclosing all points, of minimum circumference. What you'd get if you "pulled a string" which was placed around all points.

8

Package Wrapping

This is the most natural algorithm to find convex hull given a set of points.



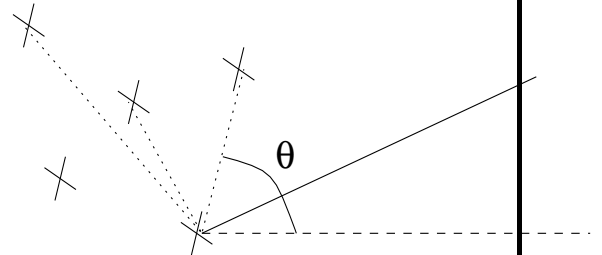
- Start with a point guaranteed to be on hull (e.g., lowest y value).
- Take horizontal line anchored at this point and 'sweep' up until it hits another point. Record point and "anchor" line there.
- Continue sweeping until another point hits. Continue until you get back to start point.

9

Package Wrapping Implementation

In fact.. can just find angle with horizontal for each line from anchor to point in set.

Find point p such that this angle is minimum, (but greater than angle for previous sweep line).



Assume we have a function theta which calculates angle of line with horizontal:

```
float theta(point p1, point p2)
// returns angle between line from p1 and p2 and
// horizontal line.
```

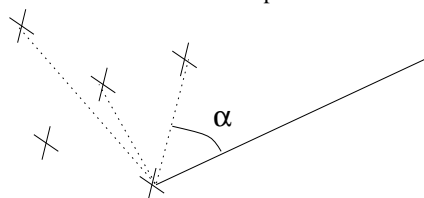
11

Implementation

Requires the following:

Given a line l, an anchor point, and a set of points, how do we find the "next" point that line will hit if swept round?

It's the one such that the angle alpha between line l and the line from anchor to point is a minimum.



So.. try all points, measure angles and return point with lowest such angle as next anchor point.

10

Algorithm

- Find point p[min] with lowest w value.
- Until hull complete (back to p[min]), find next point m on hull by:
 - Looping through other points p[i] to find next one such that angle to horizontal is minimum but greater than current sweep angle v.
 - Make that point be the mth point on the hull.
 - Set current sweep angle v to be "theta" value of new point.

12

Code

```
void wrap(points p[], points wrap[], int n)
{
    .. variable declarations
    // find starting point
    min=0;
    for(i=1; i< n; i++)
        if(p[i].y < p[min].y) min = i;
    wrap[0]=points[min]; m=0; th=0;
    do
    {
        th=360.0; v=th;
        // find the point with lowest angle
        for (i = 0; i <= n; i++)
            if ((theta(wrap[m],p[i]) > v) &&
                (theta(wrap[m], p[i]) < th))
                {next = i; th = theta(p[m], p[i]) }
        wrap[m++]=points[next];
    }
    while(next != points[min])
}
```

Slightly more efficient versions exist, which exploit sorting techniques.

13

Summary

- Assignments/coding: Think of how to make your code useable later/by other people!
- Geometric algorithms: Sometimes thinking “graphically” results in a good algorithm, rather than trying to solve some geometric problem mathematically.

14