Algorithms Lab

October 3, 2012

Outline

Outline

1. Hints
2. Templates
3. Memory layout
4. Performance

Know standard techniques

- backtrack
- greedy
- divide & conquer
- dynamic programming ...

Exercise 1 - Longest Path

- Modify DFS/BFS – for a given vertex v, find the longest path which starts in v
- Iterate over all v – $\mathcal{O}(n^2)$ in total ... too slow
- Modify DFS further
- Challenge – implement without a recursion

Exercise 1 - Longest Path

- Modify DFS/BFS – for a given vertex v, find the longest path which starts in v
- Iterate over all v – $\mathcal{O}(n^2)$ in total ... too slow
- Modify DFS further
- Challenge – implement without a recursion

Exercise 1 - Longest Path

- Modify DFS/BFS – for a given vertex v, find the longest path which starts in v
- Iterate over all v – $\mathcal{O}(n^2)$ in total ... too slow
- Modify DFS further
- Challenge – implement without a recursion

Exercise 2 - The Beasts

- 50 points – backtrack
  - keep track of occupied columns/diagonals
- 100 points – backtrack with randomization, ...
- Try not to look on wikipedia for a solution!

Exercise 2 - The Beasts

- 50 points – backtrack
  - keep track of occupied columns/diagonals
- 100 points – backtrack with randomization, ...
- Try not to look on wikipedia for a solution!

Exercise 2 - The Beasts

- 50 points – backtrack
  - keep track of occupied columns/diagonals
- 100 points – backtrack with randomization, ...
- Try not to look on wikipedia for a solution!

Exercise 3 - Burning Coins

- 2 coins - easy
- 3 coins - still easy
- 4+ coins - draw a tree of all possible game scenarios
- Straightforward recursive implementation – 30 points
- Observation - (left, right, left) and (left, left, right) give the same resulting sequence
- Challenge - implement without a recursion

Exercise 3 - Burning Coins

- 2 coins - easy
- 3 coins - still easy
- 4+ coins - draw a tree of all possible game scenarios
- Straightforward recursive implementation – 30 points
- Observation - (left, right, left) and (left, left, right) give the same resulting sequence
- Challenge - implement without a recursion

Exercise 3 - Burning Coins

- 2 coins - easy
- 3 coins - still easy
- 4+ coins - draw a tree of all possible game scenarios
- Straightforward recursive implementation – 30 points
- Observation - (left, right, left) and (left, left, right) give the same resulting sequence
- Challenge - implement without a recursion

Exercise 3 - Burning Coins

- 2 coins - easy
- 3 coins - still easy
- 4+ coins - draw a tree of all possible game scenarios
- Straightforward recursive implementation – 30 points
- Observation - (left, right, left) and (left, left, right) give the same resulting sequence
- Challenge - implement without a recursion

Exercise 3 - Burning Coins

Exam question!

- $\approx$ 110 students
- 31 - 100 points
- 16 - 30 points

Exercise 4 - Pinball

- In disguise

$$\max_{i<j}\left\{\sum_{t=1}^{i-1} p_t + (j-i+1)p_j + \sum_{t=j+1}^{n} p_t\right\}$$

- equivalently

$$\max_{i<j}\{S_n - (S_j - S_{i-1}) + (j-i+1)p_j\}$$

- simple $\mathcal{O}(n^2)$ solution - iterate over all i, j – 10 points

Exercise 4 - Pinball

- In disguise

$$\max_{i<j}\left\{\sum_{t=1}^{i-1} p_t + (j-i+1)p_j + \sum_{t=j+1}^{n} p_t\right\}$$

- equivalently

$$\max_{i<j}\{S_n - (S_j - S_{i-1}) + (j-i+1)p_j\}$$

- simple $\mathcal{O}(n^2)$ solution - iterate over all i, j – 10 points

Exercise 4 - Pinball

- In disguise

$$\max_{i<j}\left\{\sum_{t=1}^{i-1} p_t + (j-i+1)p_j + \sum_{t=j+1}^{n} p_t\right\}$$

- equivalently

$$\max_{i<j}\{S_n - (S_j - S_{i-1}) + (j-i+1)p_j\}$$

- simple $\mathcal{O}(n^2)$ solution - iterate over all i, j – 10 points

Exercise 4 - Pinball

For each j

- $f_j(i) := S_n - (S_j - S_{i-1}) + (j - i + 1)p_j$
- m_j – smallest i which maximizes f_j

Observation 1

- Consider $j_1 < j_2$
- Assume $p_{j'} \leq p_{j_2}$ for each $j' \in \{j_1, \dots, j_2 - 1\}$
- then $f_{j_2}(m_{j_1}) \geq f_{j_1}(m_{j_1}) \Rightarrow f_{j_2}(m_{j_2}) \geq f_{j_1}(m_{j_1})$

Exercise 4 - Pinball

For each j

- $f_j(i) := S_n - (S_j - S_{i-1}) + (j - i + 1)p_j$
- m_j – smallest i which maximizes f_j

Observation 1

- Consider $j_1 < j_2$
- Assume $p_{j'} \leq p_{j_2}$ for each $j' \in \{j_1, \dots, j_2 - 1\}$
- then $f_{j_2}(m_{j_1}) \geq f_{j_1}(m_{j_1}) \Rightarrow f_{j_2}(m_{j_2}) \geq f_{j_1}(m_{j_1})$

Exercise 4 - Pinball

For each j

- $f_j(i) := S_n - (S_j - S_{i-1}) + (j - i + 1)p_j$
- m_j – smallest i which maximizes f_j

Observation 1

- Consider $j_1 < j_2$
- Assume $p_{j'} \leq p_{j_2}$ for each $j' \in \{j_1, \dots, j_2 - 1\}$
- then $f_{j_2}(m_{j_1}) \geq f_{j_1}(m_{j_1}) \Rightarrow f_{j_2}(m_{j_2}) \geq f_{j_1}(m_{j_1})$

Exercise 4 - Pinball

For each j

- $f_j(i) := S_n - (S_j - S_{i-1}) + (j - i + 1)p_j$
- m_j – smallest i which maximizes f_j

Observation 1

- Consider $j_1 < j_2$
- Assume $p_{j'} \leq p_{j_2}$ for each $j' \in \{j_1, \dots, j_2 - 1\}$
- then $f_{j_2}(m_{j_1}) \geq f_{j_1}(m_{j_1}) \Rightarrow f_{j_2}(m_{j_2}) \geq f_{j_1}(m_{j_1})$

Exercise 4 - Pinball

Instead of every j - set of candidates S for the right end of your obstacle

- go from right to left
- consider a hole only if it updates the current maximum
- for each $j \in S$ – find m_j by iterating $i = 1, \dots, j - 1$
- S can be large ... still $\mathcal{O}(n^2)$ – 30 points

Exercise 4 - Pinball

Instead of every j - set of candidates S for the right end of your obstacle

- go from right to left
- consider a hole only if it updates the current maximum
- for each $j \in S$ – find m_j by iterating $i = 1, \dots, j - 1$
- S can be large ... still $\mathcal{O}(n^2)$ – 30 points

Exercise 4 - Pinball

Observation 2

- Consider $j_1, j_2 \in S$ and $j_1 < j_2$
- remember – from the definition of S we have $p_{j_1} > p_{j_2}$
- then $m_{j_1} \leq m_{j_2}$

Consequence of observation 2

- Consider some $j \in S$ and m_j
- for all $j_1 \in S$, $j_1 < j$: $m_{j_1} \leq m_j$
- for all $j_2 \in S$, $j < j_2$: $m_j \leq m_{j_2}$
- what if j is the median of S?

Exercise 4 - Pinball

Observation 2

- Consider $j_1, j_2 \in S$ and $j_1 < j_2$
- remember – from the definition of S we have $p_{j_1} > p_{j_2}$
- then $m_{j_1} \leq m_{j_2}$

Consequence of observation 2

- Consider some $j \in S$ and m_j
- for all $j_1 \in S$, $j_1 < j$: $m_{j_1} \leq m_j$
- for all $j_2 \in S$, $j < j_2$: $m_j \leq m_{j_2}$
- what if j is the median of S?

Algorithms Lab

Exercise 4 - Pinball

Observation 2

- Consider $j_1, j_2 \in S$ and $j_1 < j_2$
- remember – from the definition of S we have $p_{j_1} > p_{j_2}$
- then $m_{j_1} \leq m_{j_2}$

Consequence of observation 2

- Consider some $j \in S$ and m_j
- for all $j_1 \in S$, $j_1 < j$: $m_{j_1} \leq m_j$
- for all $j_2 \in S$, $j < j_2$: $m_j \leq m_{j_2}$
- what if j is the median of S?

Algorithms Lab

Exercise 4 - Pinball

Observation 2

- Consider $j_1, j_2 \in S$ and $j_1 < j_2$
- remember – from the definition of S we have $p_{j_1} > p_{j_2}$
- then $m_{j_1} \leq m_{j_2}$

Consequence of observation 2

- Consider some $j \in S$ and m_j
- for all $j_1 \in S$, $j_1 < j$: $m_{j_1} \leq m_j$
- for all $j_2 \in S$, $j < j_2$: $m_j \leq m_{j_2}$
- what if j is the median of S?

Algorithms Lab

Exercise 4 - Pinball

Observation 2

- Consider $j_1, j_2 \in S$ and $j_1 < j_2$
- remember – from the definition of S we have $p_{j_1} > p_{j_2}$
- then $m_{j_1} \leq m_{j_2}$

Consequence of observation 2

- Consider some $j \in S$ and m_j
- for all $j_1 \in S$, $j_1 < j$: $m_{j_1} \leq m_j$
- for all $j_2 \in S$, $j < j_2$: $m_j \leq m_{j_2}$
- what if j is the median of S?

Algorithms Lab

Testing hints

- Write a slow solution – easy to check if correct
- Handcraft testcases
  - whatever comes to your mind
  - trivial cases – $5, 4, 3, 2, 1$ for Pinball
- Generate testcases
  - random

Testing hints

- Write a slow solution – easy to check if correct
- Handcraft testcases
  - whatever comes to your mind
  - trivial cases – $5, 4, 3, 2, 1$ for Pinball
- Generate testcases
  - random

Testing hints

- Write a slow solution – easy to check if correct
- Handcraft testcases
  - whatever comes to your mind
  - trivial cases – $5, 4, 3, 2, 1$ for Pinball
- Generate testcases
  - random

Algorithms Lab

Testing hints

Random testcases

- might miss border cases – correctness
- might miss worse case – runtime

Example

- Consider second solution for the Pinball problem
- Runtime – $\mathcal{O}(|S| \cdot n)$
- Generate random testcase – assume all points are distinct

$$Pr[\text{update of maximum at position } i] = \frac{1}{n-i+1}$$

$$\mathbb{E}[\#\text{ of updates}] = \sum_{i=1}^{n} 1/i \approx \log n$$

- $|S| \approx \log n \to \mathcal{O}(n \log n)$ running time in expectation

Testing hints

Random testcases

- might miss border cases – correctness
- might miss worse case – runtime

Example

- Consider second solution for the Pinball problem
- Runtime – $\mathcal{O}(|S| \cdot n)$
- Generate random testcase – assume all points are distinct

$$Pr[\text{update of maximum at position } i] = \frac{1}{n-i+1}$$

$$\mathbb{E}[\#\text{ of updates}] = \sum_{i=1}^{n} 1/i \approx \log n$$

- $|S| \approx \log n \to \mathcal{O}(n \log n)$ running time in expectation

Algorithms Lab

Testing hints

Random testcases

- might miss border cases – correctness
- might miss worse case – runtime

Example

- Consider second solution for the Pinball problem
- Runtime – $\mathcal{O}(|S| \cdot n)$
- Generate random testcase – assume all points are distinct

$$Pr[\text{update of maximum at position } i] = \frac{1}{n-i+1}$$

$$\mathbb{E}[\#\text{ of updates}] = \sum_{i=1}^{n} 1/i \approx \log n$$

- $|S| \approx \log n \to \mathcal{O}(n \log n)$ running time in expectation

Algorithms Lab

Testing hints

Random testcases

- might miss border cases – correctness
- might miss worse case – runtime

Example

- Consider second solution for the Pinball problem
- Runtime – $\mathcal{O}(|S| \cdot n)$
- Generate random testcase – assume all points are distinct

$$Pr[\text{update of maximum at position } i] = \frac{1}{n-i+1}$$

$$\mathbb{E}[\#\text{ of updates}] = \sum_{i=1}^{n} 1/i \approx \log n$$

- $|S| \approx \log n \to \mathcal{O}(n \log n)$ running time in expectation

Algorithms Lab

Testing hints

Random testcases

- might miss border cases – correctness
- might miss worse case – runtime

Example

- Consider second solution for the Pinball problem
- Runtime – $\mathcal{O}(|S| \cdot n)$
- Generate random testcase – assume all points are distinct

$$Pr[\text{update of maximum at position } i] = \frac{1}{n-i+1}$$

$$\mathbb{E}[\# \text{ of updates}] = \sum_{i=1}^{n} 1/i \approx \log n$$

- $|S| \approx \log n \rightarrow \mathcal{O}(n \log n)$ running time in expectation

Testing hints

Random testcases

- might miss border cases – correctness
- might miss worse case – runtime

Example

- Consider second solution for the Pinball problem
- Runtime – $\mathcal{O}(|S| \cdot n)$
- Generate random testcase – assume all points are distinct

$$Pr[\text{update of maximum at position } i] = \frac{1}{n-i+1}$$

$$\mathbb{E}[\# \text{ of updates}] = \sum_{i=1}^{n} 1/i \approx \log n$$

- $|S| \approx \log n \rightarrow \mathcal{O}(n \log n)$ running time in expectation

Outline

Templates

Example

```
std::vector<int>
```


Templates – goal

Templates solve this problem:

```
void swap_int (int& a, int& b) {
    int t = b;
    b = a;
    a = t;
}
```


Templates – goal

Templates solve this problem:

```
void swap_int (int& a, int& b) {
    int t = b;
    b = a;
    a = t;
}

void swap_float (float& a, float& b) {
    float t = b;
    b = a;
    a = t;
}
```

Function templates

How about if we could say

```cpp
/* for any type T, define a function */
void swap (T& a, T& b) {
    T t = b;
    b = a;
    a = t;
}
```

It is that easy!

```cpp
template<class T>
void swap (T& a, T& b) {
    T t = b;
    b = a;
    a = t;
}
```

Instead of class, you can also use typename as a synonym.

Function templates

How about if we could say

```cpp
/* for any type T, define a function */
void swap (T& a, T& b) {
    T t = b;
    b = a;
    a = t;
}
```

It is that easy!

```cpp
template<class T>
void swap (T& a, T& b) {
    T t = b;
    b = a;
    a = t;
}
```

Instead of class, you can also use typename as a synonym.

Class templates

```cpp
template<class T, class U>
class pair {
public:
    T first;
    U second;
};

pair<int, float> p;
p.first = 1;
p.second = 2;
```

The same goes for struct.

Class templates

```cpp
template<class T, class U>
class pair {
public:
    T first;
    U second;
};

pair<int, float> p;
p.first = 1;
p.second = 2;
```

The same goes for struct.

Special-casing types

Templates can special-case for certain types (or even values). Consider a simple vector:

```cpp
template<class T>
class vector {
    long _size;
    long _capacity;
    T *data;
public:
    T operator[](int i) const {
        return data[i];
    }
    /* rest of logic left as an exercise */
};
```

But if T is `bool`, shouldn't we implement a packed bitfield?

Special-casing types

Templates can special-case for certain types (or even values). Consider a simple vector:


```cpp
template<class T>
class vector {
    T *data;
};
```


But if T is `bool`, shouldn't we implement a packed bitfield?

Special-cased bitfield vector

```cpp
template<>
class vector<bool> {
    long _size;
    long _capacity;
    char *data;
public:
    bool operator[](int i) const {
        return (data[i/8] >> i%8) & 1;
    }
    /* rest of logic left as an exercise */
};
```


Outline

Memory layout and variables

```cpp
int a;
static int b;
extern int c;
std::vector<int> u(1);

int f(int x) {
    int i;
    static int j;
    std::vector<int> v;
    for (int k = 0; k < 100; k++) {
        std::vector<int> w(1);
    }
}
```


Outline

Performance – measures

Real time or **wall time**: Time elapsed on a clock until the task is finished

User time Time the kernel allocated for your program to run

System time Time the kernel did work on your program's behalf

Note

Measurement is influenced by many things and variance usually about 20%.

Performance – measures

Real time or **wall time**: Time elapsed on a clock until the task is finished

User time Time the kernel allocated for your program to run

System time Time the kernel did work on your program's behalf

Note

Measurement is influenced by many things and variance usually about 20%.

User/system time example

Consider

```
for (int i = 0; i < 1000000; i++)
    std::cout << i << "\n";
```

vs.

```
for (int i = 0; i < 1000000; i++)
    std::cout << i << std::endl;
```

	real [ms]	user [ms]	sys [ms]
`"\n"`	99	97	2
`endl`	241	123	117

User/system time example

Consider

```
for (int i = 0; i < 1000000; i++)
    std::cout << i << "\n";
```

vs.

```
for (int i = 0; i < 1000000; i++)
    std::cout << i << std::endl;
```

	real [ms]	user [ms]	sys [ms]
`"\n"`	99	97	2
`endl`	241	123	117

Measurement on judge

Speed

The judge measures only something vaguely resembling user time.

- For the purposes of this tutorial we call this unit *ticks* (t).
- The "actual" time measurements were done on my laptop.
  - Clock speed boosted 3.3 GHz
  - Length of one clock cycle thus 0.3 ns

Measurement on judge

Speed

The judge measures only something vaguely resembling user time.

- For the purposes of this tutorial we call this unit *ticks* (t).
- The "actual" time measurements were done on my laptop.
  - Clock speed boosted 3.3 GHz
  - Length of one clock cycle thus 0.3 ns

Measurement on judge

Speed

The judge measures only something vaguely resembling user time.

- For the purposes of this tutorial we call this unit *ticks* (t).
- The "actual" time measurements were done on my laptop.
  - Clock speed boosted 3.3 GHz
  - Length of one clock cycle thus 0.3 ns

Measurement on judge

Speed

The judge measures only something vaguely resembling user time.

- For the purposes of this tutorial we call this unit *ticks* (t).
- The "actual" time measurements were done on my laptop.
  - Clock speed boosted 3.3 GHz
  - Length of one clock cycle thus 0.3 ns

Performance: simple operations

X-ing up a 1000-element array takes (per element)...

op	ns	nt
add	0.17	0.26
multiply	0.56	0.52
divide	6.0	1.2

Remember that one clock cycle is SI0.3ns!

Performance: input/output

Doing I/O with `int` typed 1–3 digit numbers takes...

op	ns	nt
`cin` sync	240	297
`cin` nosync	79	100
`scanf`	95	124
`cout` sync	84	118
`cout` nosync	90	107
`cout` sync `endl`	130	142
`cout` nosync `endl`	110	120
`printf`	81	106

Performance: function calls

Calling a simple function...

op	ns	nt
directly	1.9	2.1
indirectly	2.2	2.3
inlined	0.23	0.28

Performance: memory allocations

Allocating a few bytes of memory on the heap...

op	ns	nt
`new`	31	41
`new`, `delete`	35	52
`malloc`	27	38
`malloc`, `free`	32	48

Performance: vector

Storing elements in a vector of type...

type	ns	nt
`int`	0.23	0.20
`char`	0.28	0.35
`bool`	0.89	1.65

Performance: set

In a set, doing...

op	ns	nt
`insert`	180	130
`insert` with many dups	90	31
`insert`, `remove`	280	65

Recall that `new` is about 30 ns.

Performance: vector

Sum up the elements in a. . . (per element)

type	ns	nt
`vector<int>`	0.6	0.85
`set<int>`	5.9	4.2

Performance: vector

Sum up the elements in a. . . (per element)

type	ns	nt
`vector<int>`	0.6	0.85
`set<int>`	5.9	4.2

Algorithms Lab

Performance: synchronization

Locking and then unlocking a mutex. . .

type	ns
only one user	4.7
heavily contended	140

Performance: summary