$T(n)$

$T(n-1)$

$T(n-2)$

$T(0)$

CSCI 241

Lecture 27

Runtime Analysis of Recursive Methods

Max-flow / Min-cut

Greatest Misses

Announcements

- A3 out soonish
- A4 graded next week

Goals

- Be able to solve simple **recurrence relations** to analyze the runtime of recursive methods.

- Get exposure to the **max-flow/min-cut** problem and some of its applications.

- Be able to solve all the hardest quiz problems from throughout the quarter.

Runtime Analysis: Review

- Why? We want a measure of performance that is
  - **Independent** of what computer we run it on.
    Solution: count **operations** instead of clock time.
  - Dependence on **problem size** is made explicit.
    Solution: express runtime as a function of **n**
    (or whatever variables define problem size)
  - **Simpler** than a raw count of operations and focuses on performance on **large problem sizes**.
    Solution: ignore constants, analyze **asymptotic** runtime.

Runtime Analysis: Review

- How?

  1. Count the number of primitive (constant-time) operations that occur over the entire execution of the algorithm.

  2. Drop constants and lower-order terms to find the **asymptotic runtime class**.

Counting Strategies:

1. Simple counting:
   - How long does each line take?
   - How many times does each line happen?
   - Multiply and total it up.
2. Aggregate analysis:
   - Reason about how many times a given line is executed independent of loops/code structure:
     - Example: Radix sort

```
for each bucket:
  for each element:
    // doesn't happen 10*n times
```

     - Example: Prim's algorithm

```
for each vertex:
  for each edge:
    // doesn't happen v*e times
```

Counting Strategies: 3. Recurrences

- Counting is trickier without loop bounds.

```
public int listSize(Node n) {
  if (n == null) {
    return 0;
  }
  return 1 + listSize(n.next);
}
```

Counting Strategies: 3. Recurrences

- Let T(n) be the runtime on a problem of size n.

```
public int listSize(Node n) {
  if (n == null) {        ——— 1
    return 0;             ——— 1   ← (only 1 of these
  }                                  can happen)
  return 1 + listSize(n.next); —— 1 + T(n-1)
}
```

$T(0) = 1 + 1$

$T(n) = 1 + 1 + T(n-1)$

Counting Strategies: 3. Recurrences

- Let T(n) be the runtime on a problem of size n.

```
public int listSize(Node head) {
  if (head == null) {      —— 1
    return 0;              —— 1
  }
  return 1 + listSize(head.next); — 1 + T(n-1)
}
```

$T(0) = 2$

$$
\begin{aligned}
T(n) &= 2 + T(n-1) \\
&= 2 + 2 + T(n-2) \\
&= 2 + 2 + 2 + T(n-3) \\
&\ldots \\
&= 2 + 2 + 2 + \ldots + T(0)
\end{aligned}
$$

(there are n of these in total!)

Counting Strategies: 3. Recurrences

```
/** Return the index of v in A[s..e], or -1 if it
 * doesn't exist. Pre: 0 <= s <= e < A.length */
public int binSearch(int[] A, int v, int s, int e) {
  if ((e - s) == 0) {             1
    return -1;                    1
  }
  int mid = (e+s)/2;              1
  if (A[mid] == v) {              1
    return mid;                   1
  else if (A[mid] < v) {          1
    return binSearch(A, v, s, mid);   T(n/2)
  } else {
    return binSearch(A, v, mid, e);   T(n/2)
  }
}
```

(only 1 of these can happen)

Let n = e-s:

$T(0) = 2$

$$
\begin{aligned}
T(n) &= 4 + T(n/2) \\
&= 4 + 4 + T(n/4) \\
&= 4 + 4 + 4 + T(n/8)
\end{aligned}
$$

Counting Strategies:
3. Recurrences

```
/** Return the index of v in A[s..e], or -1 if it
 * doesn't exist. Pre: 0 <= s <= e < A.length */
public int binSearch(int[] A, int v, int s, int e) {
  if ((e - s) == 0) {           1
    return -1;                  1
  }
  int mid = (e+s)/2;            1
  if (A[mid] == v) {            1
    return mid;                 1
  else if (A[mid] < v) {        1
    return binSearch(A, v, s, mid);   T(n/2)
  } else {
    return binSearch(A, v, mid, e);   T(n/2)
  }
}
```

Let n = e-s:

$T(0) = 2$

$T(n) = 4 + T(n/2)$

$= 4 + 4 + T(n/4)$

$= 4 + 4 + 4 + T(n/8)$

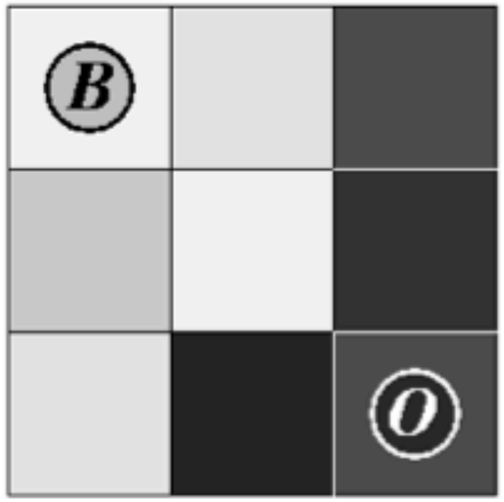
$\vdots$

$= 4 + 4 + 4 + \ldots + T(0)$

$= 4 + 4 + 4 + \ldots + 2$

(there are log(n) of these in total!)

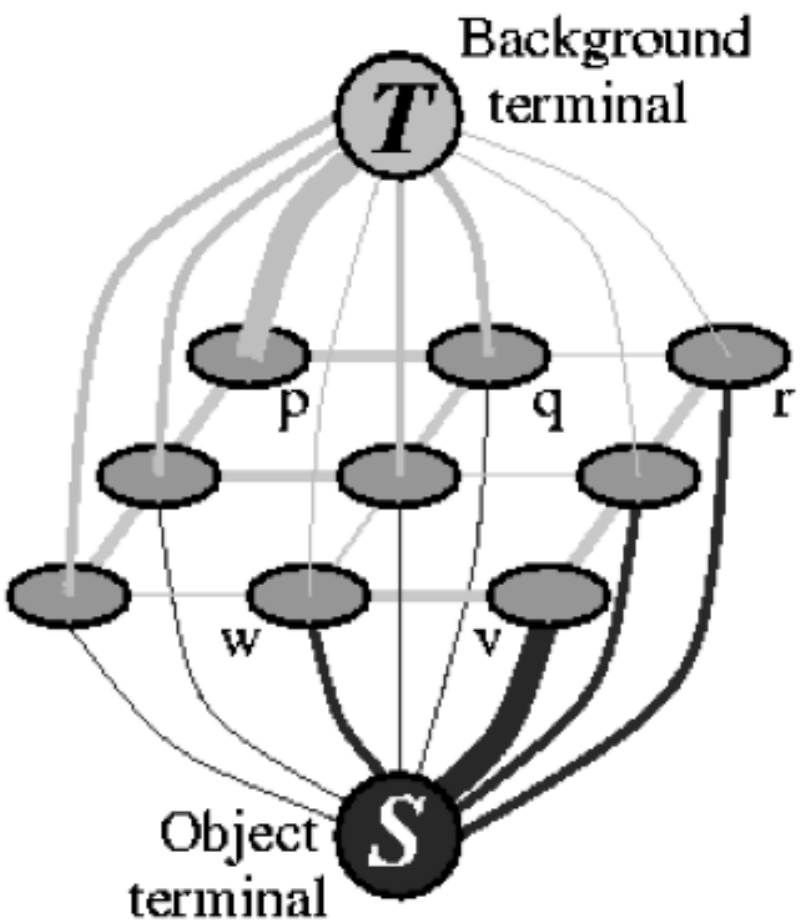
Graph Cuts:
Max Flow/ Min Cut



(a) Image with seeds.

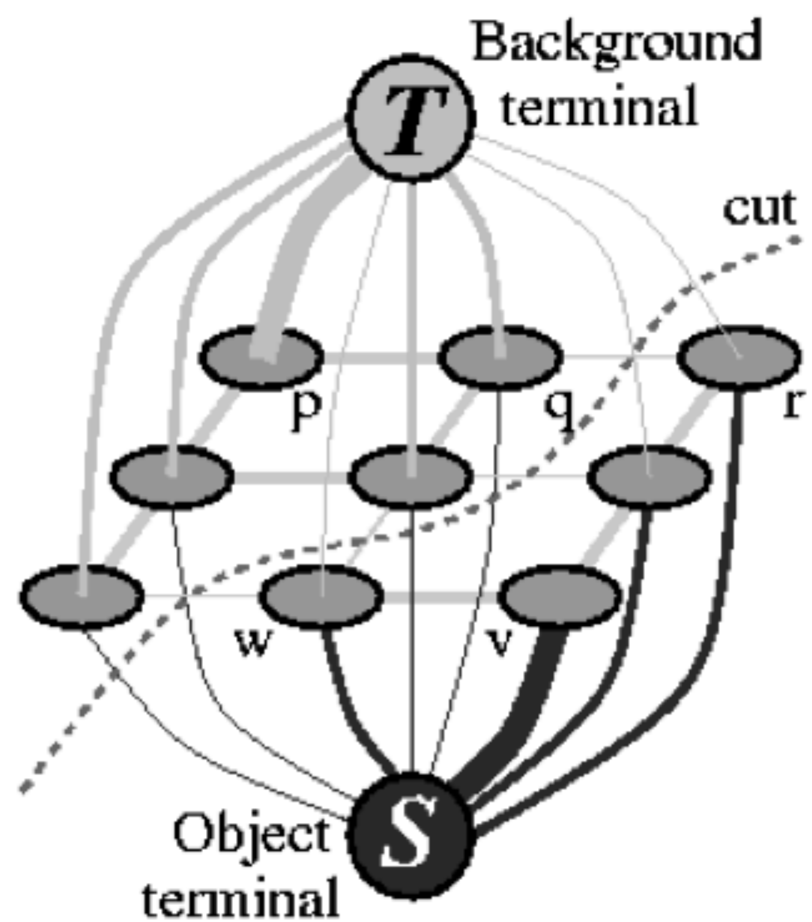
(d) Segmentation results.

⇓ ⇑



(b) Graph.

⇒



(c) Cut.