Advanced Algorithms

Lecture 14: Randomness in algorithm design

Announcements

- **Mid-term grades out** (Midterm solutions).
- HW 3 due Wednesday (tomorrow)

[please note mild change in last problem.]

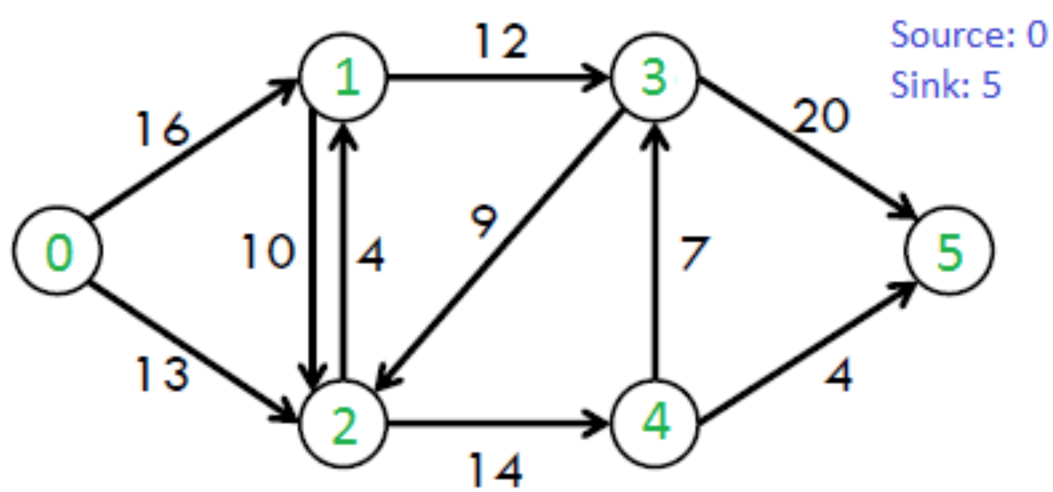
Last two weeks

- Basic graph algorithms
  - Dijkstra's algorithm (O(m+n) log n) time — imitation of BFS
  - DP based, "Bellman-Ford" algorithm — O(n (m+n)) time
  - "Definitions" of flows and cuts in graphs

Maximum flow

communication networks, shipping goods, …

Problem: given a (directed) graph G = (V, E) with edge **capacities** (> 0), source u, sink v, find the max possible "rate" at which one can send "information" from u to v.



→ Greedy procedure gets "stuck"

→ Ford-Fulkerson algorithm solves this problem in time ~ $(m+n)^2$

Min cut problem

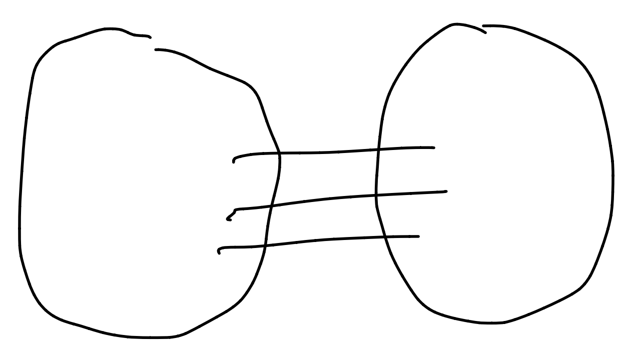
Problem: given a (directed) graph G = (V, E) with edge **costs** (> 0), source u, sink v, find the <u>min</u> possible set of edges to "cut" so that there's no path from u —> v

Blowing bridges…

- Undirected graphs
- Image segmentation

"Sparsest cut"

graph partitioning.



$\geq \frac{n}{4}$.

Flows and cuts

Theorem (easy): G = (V, E) be a weighted directed graph, and u, v be vertices. Let “F” be any flow, interpreting wts as capacities. Let “C” be any cut, interpreting wts as costs. Then F <= C.

Comments

- Max-flow min-cut theorem
- Many applications — e.g., no bottleneck => many edge disjoint paths
- Algorithms for cut == algorithms for flow

Today

Can randomness help in algorithm design?

Toy problem

Problem: given an (unsorted) array A[0], A[1], …, A[n-1], and the *promise* that at least *n*/3 of the A[i] are 0, find one index *i* s.t. A[i]=0

- Generalization of HW problem

– Simple idea: if we pick an i at random, then $\Pr[A[i]=0] \geq 1/3$

Randomized procedure

Pick r random indices $i_1, i_2, \ldots, i_r$
if any of $A[i_t] = 0$, output i_t
else FAIL.

run time $= r$.

Prob. of failure? $\left(\frac{2}{3}\right)^r$

for each index, there is a prob. of $\frac{2}{3}$, these are independent.

for $\frac{1}{1000}$, just need $r \approx 20$.

(decays exponentially).

Key trade-off

- Higher running time, higher probability of success
- <u>Note:</u> don't even read entire input!

[· sampling algos work for a similar reason.]

"Las Vegas" algorithm

(never fail, but can potentially run infinitely).

- While not found: pick random index i and check if A[i]=0

Expected Running Time

(similar to tossing until seeing heads)
Running time is a random variable

Example 2 – checking identities

(checking circuits).

$(x - a_1)(2x - a_2) \cdots \cdots (x - a_d)$.

$$p(x) = (x-7)(x-3)(x-1)(x+2)(2\,x+5)$$
$$q(x) = 2\,x^5 - 13\,x^4 - 21\,x^3 + 127\,x^2 + 121\,x - 210$$

Qn: is $p(x) = q(x)$?

- What if we simply plug in a random integer *x* in interval [1,20]?

and check if $p(x) = q(x)$ for this integer.

$$p(x) = (c_1 \underline{x} - a_1)(c_2 x - \underline{a_2}) \cdots (c_d x - a_d)$$

$$q(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_d x^d.$$

\# of terms in the "expansion" of $\underline{\underline{p(x) \equiv 2^d}}$.

what is

$p(1)$? $\rightsquigarrow$ poly(d)

same for $q(1)$.

One variable identities

Issue: we can have $p(a) = q(a)$ without having

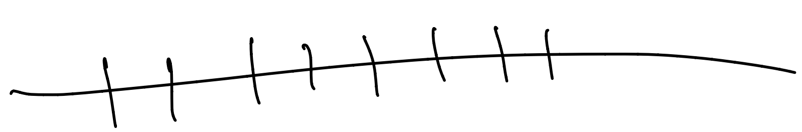
$p(x) \equiv q(x)$.

$r(x) \equiv p(x) - q(x)$ $\longrightarrow$ any a s.t. $p(a) = q(a)$ is a root of $r(x)$

& r is also a polynomial (degree $\leq d$).

$\Downarrow$

there are at most d values of "a" s.t. $p(a) = q(a)$.



General theorem: Schwartz-Zipfel Lemma

Algorithm: pick an interval with $2d$ integers. $\hookrightarrow I$.

- pick a random integer "a" from I and check if $p(a) = q(a)$.

Example 3 – primality

10513

Problem: given an integer $X = a_1 a_2 \ldots a_n$, find if X is prime

$\sqrt{X} \approx 10^{n/2}$

X has no divisors other than 1 and itself.

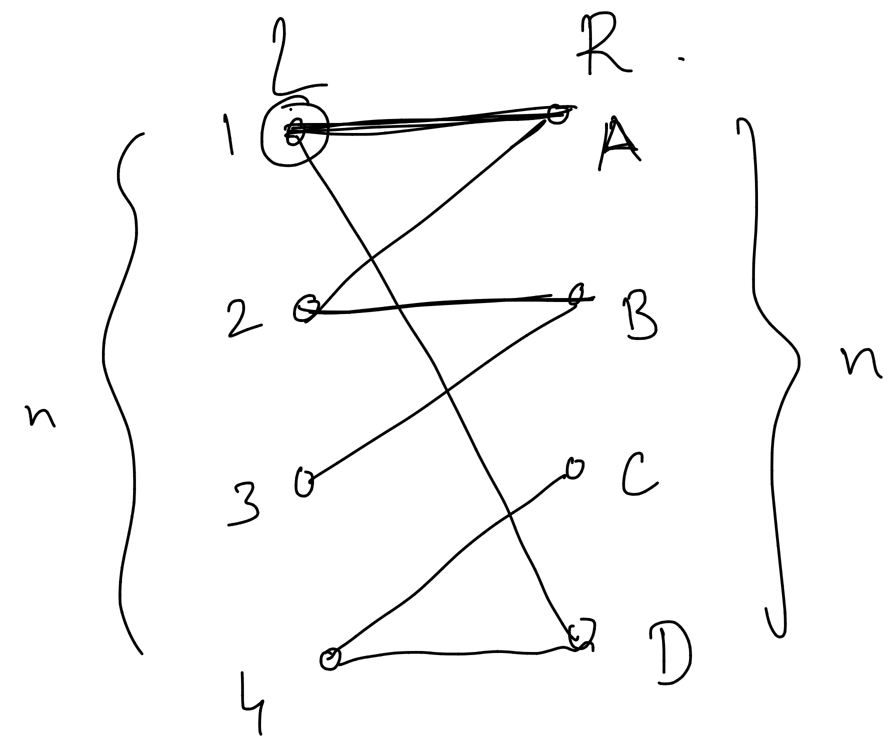
- Classic problem in math/CS
- Can an algorithm run in time poly(n)?
- Miller-Rabin test (link on course webpage).

$r \in [1, \ldots, X]$,

Example 4 – perfect matching

Canonical application of max flow.

Problem: given a bipartite graph G, find if it has a "perfect matching"



- <u>Claim:</u> this reduces to identity testing!

$$M = \begin{array}{c|c} & \overbrace{\begin{matrix} A & B & C & D \end{matrix}}^{n} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} x_{1A} & 0 & 0 & x_{1D} \\ x_{2A} & x_{2B} & 0 & 0 \\ 0 & x_{3B} & 0 & 0 \\ 0 & 0 & x_{4C} & x_{4D} \end{bmatrix} \end{array}$$

$x_{1D} \to 0$.

$1 \to D$
$2 \to A$
$3 \to B$
$4 \to C$.

<u>Obsn:</u> $\det(M)$ is a polynomial in variables $x_{1A}, x_{2A}, \ldots$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdot & a_{2n} \\ & & \vdots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$1, 3, 2$

$$\underline{\underline{\det(A)}} = \sum_{\substack{\text{permutations} \\ \underline{\underline{\sigma \text{ of } \{1,2,\ldots,n\}}}}} (-1)^{\overbrace{\operatorname{sign}(\sigma)}} \underbrace{a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)}}.$$

Perfect matching

Claim: if G has no perfect matching, then determinant of M ($\det(M)$) is $\equiv 0$ &
if G does have a perfect matching, $\det(M) \not\equiv 0$.

Proof: Suppose there was a perfect matching; then the corresponding term in the expansion of the determinant is non-zero.

Examples so far

- Finding hay in a hay stack
- Trade-off between running time and success probability
- (Fairly general) — “boosting”

Randomized algorithms overview

- Data is given, algorithm is randomized (unlike sampling/"ML" analyses)
- Usually concerned about **expected behavior**, behavior "with high probability"

Next few lectures: **general ideas, applications, analysis…**