Network Flows

Syntax notes:

f(a,b): flow from a to b cap(a,b): capacity from a to b

Lemma: For any flow f and any cut (S,T), $size(f) \leq cap(S,T)$.

1. The flow across the cut $(S,T) \leq cap(S,T)$.
2. The flow across cut (S,T) = flow across cut (S-1, T+1).
3. The flow across cut({s}, V-{s}) = size(f)

Correctness of Ford-Fulkerson

Theorem: If residual network G_f has no augmenting path, then f is a max size flow.

Proof

Let S be the set of vertices reachable by directed path in G_f.

Let T=V S

All edges from S to T are saturated (and don't exist in the residual network). Can't reach from S to T via residual network.

This implies the flow across cut (S,T) = cap (S, T).

size(f) = f(S,T) = cap(S,T) because f(u,v) = cap(u,v) for all u in S, v in T

size of any flow $\leq$ cap(S,T).

If there is a cut where you can't get across via the residual network, that means it is the minimum capacity cut.

Max Flow - Min Cut Theorem

Size of the maximum flow = capacity of the minimum capacity cut.

Proof

Use proof of correctness of Ford-Fulkerson.

Integrality Theorem

If all capacities are integers then there exists a max flow such that every edge has integer flow.

Proof

Very simple proof by induction on number of augmentations of Ford-Fulkerson, since every iteration you add 1 unit of flow.

Maximum matching in a bipartite graph.

Matching

Matching in a graph is a set of edges in a graph such that no two edges in matching have a common endpoint.

Bipartite graph

Bipartite graph verticies can be partitioned into V_1 and V_2 so that for all edges, one endpoint is in V_1 and the other in V_2.

Can partition a square into a bipartite graph by coloring opposing corners.

Pentagons, not so much. Odd cycles are bad.

Maximum matching in a bipartite graph

How do you create a flow network through a bipartite graph such that it will give you the maximum matching?

You can take the bipartite graph and connect every node in each group to a vertex for that group. Set all edges to have capacity 1. Then calculate flow. Edges with flow > 0 are a maximum matching. Since every vertex has a maximum flow of 1 through it, there won't be any edges sharing a vertex.

Applications of Network Flows

Max matching in Bipartite Graphs

Claim:

1. If M is a matching in G then there exists flow f in F (flow network from G) such that size(f) = |M|. $size(f*) \geq |M*|$
2. If f is an integer valued flow in F, then there exists a matching M in G of size size(f). $size(f*) \leq |M*|$.

Let $M = \{(u,v) \in G | f(u,v) = 1\}$ M forms a matching. Why? (capacities of edges from s are 1 and from t are 1.

Pennant Race Problem

Given a team A, a list of other teams $T_1, T_2, \ldots, T_n$, win-loss record for each team, and list of games remaining to be played.

Determine if it is possible for team A to win at least as many games as any other team by the end of the season.

Output is yes/no.

	Wins
A	3
T_1	4
T_2	6
T_3	5
T_4	4

Remaining games: (A, T1) (T1, T2) (T2, T3) (A, T3) (T2, T4) (A, T4) (T1, T2)

We can remove all games that A plays since we can say that A wins all of them.

Assume team A wins all remaining games. Let w = # A's wins (after A wins all remaining games). Let w_i = # wins of T_i.

If $w < w_i$ for some i then A has no hope.

Greedy Approach

Doesn't work. Can't just do an O(n) summation or anything similar.

Runtime of Ford-Fulkerson

Bipartite matching $O(|V| * |E|)$.

There's a more modern and faster way of doing this, but way more complicated. From 2012. That lets you calculate the network flow in a flow network with V vertices and E edges, $O(|V| * |E|)$.

Pennant Race Problem

w = #A’s wins (assuming A wins all remaining games)

w_i = #T_i’s wins

$\{(T_i, T_j)\}$ = games remaining to be played

If $w_i > w$ then no hope for A. Done.

So assume $W_i \leq w$ for all i.

Can draw a flow graph with s pointing to each game, and each game pointing to the two participating teams, and the teams pointing to t..

Edges:

$(s, (T_i, T_j))$ with capacity 1

$((T_i, T_j), T_i), ((T_i, T_j), T_j)$ with capacity 1

(T_i, t) with capacity $w_i - w$.

If max flow size = # games to play, then A still has hope.

Runtime $O((\#games * 3 + \#teams) * (\#games + \#teams))$.

Open Pit Mining

Input: Directed acyclic graph $G = (V, E)$ where V = set of tasks.

$E = \{(u, v)|u$ must be done before v$\}$.

a function w(v) that specifies profit from the task.

Find the most profitable set of tasks to perform.

An initial set in a set of vertices that has no edge coming into it from the outside.

Convert problem to a newtork flow problem so that 1. any infinite capacity cut corresponds to an initial set and , 2. a minimum capacity cut corresponds to max profit initial set.

If we connect all the nodes with infinite capacity, and then reverse it, we get finite capacities.

Claim: In this “network”, any finite capacity cut (S,T) defines an initial set T-{t}.

Proof: If cut(S,T) has finite capacity then no original edge is directed into T from S. THus T-{t} is an initial set.

If set U is an initial set, then $T = U \cup \{t\}, S = V - T$ is a cut with no original edge entering T thus finite capacity.

Open Pit Mining

1. Create edge from s to all negative value vertices with capacity `-v`.
2. Create an edge from all of the positive value points to t with capacity `v`.
3. All inner edges have infinite capacity.
4. Solve the flow network, so you can look at minimum cuts.

Infinite edges aren't part of the minimum cut, since it's going the wrong direction.

Want to maximize profit for initial set u.

$$profit = \sum_{u \in U, w(u) > 0} w(u) + \sum_{v \in U, w(v) < 0} w(v)$$

Want to minimize

$$-profit = -\sum_{u \in U, w(u) > 0} w(u) - \sum_{v \in U, w(v) < 0} w(v)$$

Capacity of cut associated with initial set U.

$$T = U \cup \{T\}$$

$$S = V - T$$

$$cap(S, T) = \sum_{u \notin U, w(u) > 0} w(u) + \sum_{v \in U, w(v) < 0} -w(v)$$

These is different than the profit calculation.

Linear Programming Duality

[max flow = min cap cut]

1. maxmize $x_1 + 6x_2$
2. $x_1 \leq 200$
3. $x_2 \leq 300$
4. $x_1 + x_2 \leq 400$
5. $x_1, x_2 \geq 0$

$$(x_1 x_2) = (100, 300)$$

$$value = 1900$$

multiply inequality constraints by multipliers y_1, y_2, y_3 and add them up.

1. y_1's must be non-negative (otherwise, inequality flips)
2. sum weighted inequalities $(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \leq 200y_1 + 300y_2 + 400y_3$ $(y_1 + y_3) = 1, (y_2 + y_3) = 6$
3. Choose "best" multipliers minimize $200y_1 + 300y_2 + 400y_3$ subject to $y_1 + y_3 \geq 1$ $y_2 + y_3 \geq 6$ $y_1, y_2, y_3 \geq 0$

Linear Programming Duality

max $x_1 + 6x_2$

subject to $x_1 \leq 200$, $x_2 \leq 300$ … (same thing as last time)

Duality Theorem

If LP has a bounded optimum then so does its dual and the two optimum values are the same.

Primal

$I = \{1, 2, \ldots, m\}$

$N = \{1, 2, \ldots, n\}$

$max c_1 x_1 + \ldots + c_n x_n$

such that $a_{i1} x_1 + \ldots + a_{in} x_n \leq b_i for i \in I$

$a_{i1} x_1 + \ldots + a_{in} x_n = b_i for i \notin I$

$x_j \geq 0 for j \in N$

Dual

Min $b_1 y_1 + \ldots + b_m y_m$

such that $a_{j1} x_1 + \ldots + a_{jm} x_m \geq c_j for j \in N$

$a_{j1} x_1 + \ldots + a_{jn} x_n = c_j for j \notin N$

$y_i \geq 0 for i \in I$

Two Player Zero-Sum Games

Rock paper scissors

	rock	paper	scissors
rock	0	-1	1
paper	1	0	-1
scissors	-1	1	0

If Row plays “rock” then Col can win every time.

A mixed strategy is a probability distribution on the actions.

Longest Increasing Subsequence

Using long common subsequence

Can use Longest Common Subsequence by sorting R $O(n \log n)$ and then using LCS(R, sort(R)) in $O(n^2)$.

Faster Solution to LIS

When computing LIS of R[1..k], knowing R[1..k-1] would be useful.

Options:

1. LIS(R[1..k-1]) - not enough
2. Best LIS(R[1..k-1]) - not enough
3. Best IS of sequences 1, 2, 3, …, j (seems like a lot of things to carry)

Witness

$\phi = (x \vee y \vee z)(\bar{x} \vee \bar{y})(x \vee \bar{z})$ <- Conjuntive normal form

$\phi \in SAT \equiv \phi$ has a satisfying truth assignment

Witness is a truth statement.

If it's satisfiable, there has to be a witness.

$\phi \in Co-Sat \equiv \phi$ has no satisfying assignment iff $\phi \in SAT$.

NP-Completeness

L is NP-complete $\equiv$ L is in NP and L is NP-hard.

NP $\equiv \exists$ witness

CoNP $\equiv \forall$ witnesses

$\exists x \forall y$ verifyable in poly time $\equiv \sum_z$

$\forall x \exists y = \Pi_z$

Proofing

You can do a transfer proof, by showing that a NP-foo problem can be transformed into whatever problem, thus it must have an lower limit of NP-foo.

True Quantified Boolean Formulas

$\forall x \exists y \forall z (x \vee y \vee z)(\bar{x} \vee \bar{y})$

Approximation Algorithms

You can make an approximation to an NP problem in P time. This can have provable performance limits. Heuristics however, don't have any guarantees.

An algorithm A is a ρ-approximation algorithm if for every input I with optimal solution value $OPT(I)$.

Approximate Algorithms

Minimum VERTEX COVER

Given an undirected graph $G = (V, E)$ find the smallest set of vertices $S \in V$ such that all edges in G have at least one of their endpoints in S.

This is an NP-hard problem.

Claim

MVC is a 2-approximation for VERTEX COVER.

We don't know how big OPTVC(G) is, but we can lower bound its size.

$|MVC(G)| \leq 2|OPTVC(G)|$

Because edges selected by MVC form a matching, no vertex covers more than one edge in a matching.

Matching Vertex Cover

Pick edge (u,v), remove the edge and all edges adjacent to u or v. Add u, v to vertex cover. Repeat

List Scheduling Approximation 1966 Graham

Given n jobs, job i must execute uninterruptedly for Pi time units.

m machines (identical) each machine can work on one job at a time

Find a schedule of jobs that minimizes completion time (time when last machine finishes).

Approximating Travelling Salesman Problem

TSP: Given a graph with edge weights visit every vertex (and return home) exactly once using smallest total weight.

Christofides Algorithm for TSP 1976

1. Find T=minimum spanning tree of G
2. Compute minimum length complete (perfect) matching M in the complete graph on odd-degree vertices of T.
3. Find Euler tour E in $T \cup M$.
4. Eliminate repeated vertices from E to get TSP tour R.

A TSP tour minus any edge is a spanning tree.

Hardness of Approximation

The general TSP is NP-hard to approximate.

Claim

If $P \neq NP$ then, there is no polytime c-approximation algorithm for TSP.

Proof

Suppose A is a poly time c-approximation algorithm for TSP. We use A to solve Hamiltonian Cycle.

Black box, reduction: Hamiltonian Cycle

G -> [-> X -> (G’) -> A -> Y ->] -> Y/N

Transform X: Create G’ from G. G’ has all edges

$$w(u,v) = \{1 if (u,v) \in G, c|V| + 1 if (u,v) \notin G\}$$

Transform Y: If $|TSP_A(G')| \leq c|V|$ then output yes, otherwise no.

Why does this work?

Edges not in the original graph are so costly that there is a gap between cost of tour if G contains a Hamiltonian cycle (cost=n), otherwise if G doesn’t contain it the cost has to be greater than $c|V|$.

Online Algorithms

For input sequence $P_1, P_2, \ldots, P_n$ an online algorithm must produce an output given $P_1, P_2, \ldots, P_i$ (without seeing the ones after) for each i.

Example: Page replacement in cache

Loading parts from disk into memory when you need it.

k is cache size (#pages)

At ith page request Pi the cache contains some k pages. If pi is not in cache (page fault) some page must be evicted from cache to make room for pi then pi is added to the cache.

The cost of a page replacement alg A on a sequence $P_1, \ldots P_n$ is $f_A(p_1, \ldots P_n) = \#$ faults on $P_1, \ldots, P_n$.

Online algorithm must decide what page to evice without knowing the future requests.

Algorithms

Least Recently Used (LRU)

Evict page whose most recent request occured furthest in the past.

Least Frequently Used (LFU)

Evict page that has been requested least often.

Marking algorithm

Approximates LRU. Set a bit on each entry in page table when it's accessed. Then you know what's recently been accessed. (with randomization).

FIFO

Evict page that has been in cache the longest.

How do we decide best online algorithm?

1. Worst-case performance

- LRU (new page very time = n)
- Least frequently used, (new page every time = n)
- FIFO (new page every time = n)

2. Average case performance

Assume all incoming pages are equally probable. m = total number of pages possibly requested.

Expected # page faults on sequence of randomly, uniformly, independently chosen pages.

- LRU, LFU, FIFO, expected = $(1 - \frac{k}{m})n$

Average case isn't typically useful, unless analyzing program traces.

3. Competitive Analysis

Have the algorithms compete against each other. How doe the online algorithm performance compare to the optimal/best offline algorithm?

n online algorithm A is c-competitive if for all $P_1 P_2 P_n$, $f_A(p_1, \dots, p_n) \leq c f_{OPT}(p_1, \dots, p_n) + b$

Theorem: LRU and FIFO are k-competitive.

Theorem: If A is a deterministic online algorithm, for paging, then $C \geq k$.

Marking algorithms can do better.

Randomized Online Algorithms

Theorem: If A is a deterministic, c-competitive online algorithm then $c \geq k$ (k= # pages in cache).

Proof: Idea: Find a sequence that is bad for A but not for OPT.

Assume A and OPT have same pages in cache 1,2,3,...,k.

Request page k+1 = a1. A faults and evicts some page, a2.

Request page a2. A faults and evicts some page a3

Request page a3. Evicts page a4, etc

A faults on every page request.

Total number different pages is k+1

How many times does OPT fault?

OPT $a_1*, a_2, a_3, \ldots, a_j, a_{j+1}*$ (* fault)

Claim: $j \geq k+1$

Proof: OPT evicts the page that is requested furthest in the future. Since there are only k+1 different pages, the next k page requests can be kept in cache.

Theorem: LRU is k-competitive

Proof: Let $p_1, p_2, p_3, \ldots, p_n$ be any sequence of page requests. Partition this sequence into contiguous sequences (phases) such that LRU faults on first page of the phase and the phase contains exactly k different pages.

LRU faults ast most k times per phase (only on 1st occurence of a page in phase)

OPT must have the first page of a phase in cache at beginning of phase. Since the remainder of the phase plus the first page of the next phase consists of k different pages, OPT must fault at least once during these requests.

$$f_{LRU}(p_1, p_2, \ldots, p_N) \leq k\#phases$$

$$f_{OPT}(p_1, p_2, \ldots, p_N) \geq k\#phases - 1$$

Thus, LRU is k-competitive.

Randomized Marking Mouse (RMM)

Mouse hides in one of m hiding places. Every time step, Cat looks in one of the m places. If mouse is there, Mouse must move to a different hiding place.

Cost = # times mouse moves

OPT = min # times mouse must move if mouse knows where cat will look.

For a deterministic mouse there is a sequence of cat probes s.

Mouse Cost(s) $\geq$ (m-1) OPT mouse cost (s)

Final Review - Friday next week 1:30-3:00

RMM

Claim

For all mice A (determinate or randomized) there exists $p_1, ..., p_n E[A_{COST}(p_1, p_2, ..., p_n)] > (\log m) OPT(p_1, p_2, ..., p_n)$

Proof idea

Show that a cat exists that will cause A to move $> \log m$ times more than OPT.

If cat probes at random then no matter what mouse A does, Cat finds it with probability $\frac{1}{m}$. Expected number of times A must move over sequence of t probes is $\frac{t}{m}$. How many Random Cat probes until cat examines all m spots? (coupon collector problem).

$m \ln m$.

So OPT moves once every $m \ln m$ probes while A moves $\frac{m \ln m}{m}$ times. A faults $\geq \ln m$ times OPT.

Hash Functions

Universal Hash functions

\$m = \$size of hash table.

A set of hash functions H that map $U \rightarrow \{0, 1, ..., m-1\}$ is universal if for all distinct keys $k, l \in U$ the number of hash functions $h \in H$ such that $h(k) = h(l)$ is at most $\frac{|H|}{m}$.

Chaining using universal hash functions

Hash n keys into a table T of size m using hash function $h \in_R H$

$\alpha = \frac{n}{m} =$ load factor

Theorem: For key k,

$E[n_{h(k)}] \leq \{\alpha + 1 if key \in T, \alpha if k \notin T\}$

Universal Hash functions

Theorem: For key k, $E[n_{h(k)}] \leq \{\alpha + 1 if k \in T, \alpha otherwise\}$

n= total # items in T m = size of T n/m = load factor = α n_i = # of items in table T bucket i.

Proof: Let $X_{kl} = \{1 if h(k) = h(l), 0 otherwise\}$ Let $Y_k =$ number of keys that hash to same slot as k.

$Y_k = \sum_{l \neq k, l \in T} X_{kl}$

$E[Y_k] = E[\sum_{l \neq k, l \in T} X_{kl}] = \sum E[X_{kl}] = \sum 1/m$

If $k \notin T$ then $n_{h(k)} = Y_k$and $|\{k \in T | l \neq k\}| = n \rightarrow E[Y - k] = n/m$

If $k \in T$ then $n_{h(k)} = Y_k + 1$ and $|\{k \in T | l \neq k\}| = n - 1 \rightarrow E[Y - k] = (n-1)/m.$

Example of universal set of hash functions

$h_{a,b}(k) = ((ak + b) \mod p) \mod m$

p is a prime bigger than any key (fixed)

Choose $a \in \{1, 2, \dots, p-1\}$ and $b \in \{0, 1, \dots, p-1\}$ to select hash function from set.

Cuckoo Hashing

Find operation takes O(1).

Delete operations takes O(1).

Insert operation takes O(1) amortized, expected

This is very weird, seems almost impossible. Occasionally, the table resizes.

Use two hash functions h_1 and h_2.

A key k with either be stored in $h_1(k)$ or in $h_2(k)$. Each slot in table contains one key.

If the slot is full, it tries the backup hashing function, if that also fails, it kicks that node out. If it cycles, it resizes to be larger.

Cuckoo graph for n items

Vertices = {0,1,…,m-1}

Edges $= \{(h_1(k), h_2(k)) | k \in T\}$

Assume $h_1(k)$ and $h_2(k)$ are uniform and independent random edges.

We add n random edges to the graph. We want to know if there's a cycle. Insertion will succeed if there's no cycle sn the cuckoo graph.

First consider paths

Lemma: For c > 1, and $m \geq 2cn$.

Pr[cuckoo graph has path of length l from i to j] $leq \frac{1}{mc^l}$.

Final Exam Mon April 10

1 sheet of notes, 2-sided

"guess what's on the exam and study those"

Definitely: Dynamic Programming, Linear Programming, Flows

Cuckoo Graph

Cycles are dangerous. Want to know how long the chains are.

n elements hashed

Lemma: For c>1 and $m \geq 2cn$, probability cuckoo graph contains a shortest path of length $l \geq 1$ between vertices i, and j is $\leq \frac{1}{mc^l}$.

(Assume $h_1(A)$ and $h_2(A)$ are random, uniform table slots).

Proof: by induction on l

Base $l = 1$ Edge (i, j) exists in graph with prob $\leq n\frac{2}{m^2} = \frac{1}{c*m}$

For $l > 1$ shorted path from i to j has length l only i fthere exists p and

1. There exists a shorted pat from i to p of length l-1 (occurs with probability $\leq \frac{1}{m*c^{l-1}}$).
2. and, there exists the edge (p,j) (occurs with prob $\leq \frac{1}{mc}$).

These two together occur with probability, $\leq \frac{1}{m^2*c^l}$

Sum over possible nodes $p \rightarrow n \leq \frac{ml}{m^2c^l}$

Propability that k and k' hash to the same path ("bucket") of cuckoo graph is probability of a path from h1(k) or h2(k) to h1(k') or h2(k')

$\leq 4\sum_{l=1}^{\infty} \frac{[}{1} mc^l = \frac{4}{m}\frac{1}{c-1} = O(\frac{1}{m})$.

Rehash means choose new hash functions and rehash all keys.

Probability rehash occurs:

- $\leq$ probility hashing creates cuckoo graph with a cycle
- $\leq \sum_{i=1}^{m} Pr[cycleinvolivngi]$
- $\leq \sum_{i=1}^{m}\sum_{l=1}^{\infty} \frac{1}{mc^l} \leq \frac{1}{2}$ for $c \geq 3$.

Expected number of rehashes is ≤ 2.

Final Review

Topics

Convex Hulls

- Jarvis March, Graham Scan
- Chan's Alg - Output sensitive $O(n \log h)$

(Linear) Algebraic Decision Tree Model

- Element Uniqueness $\Omega(n \log n)$

Linear Programming

Network Flow

- $O(|V||E|)$ Orlin
- Bipartite matching
- Pennant Race
- Open pit mining

Linear Programming Duality

- Zero sum games

Dynamic programming

- Longest common subsequent
- Longest increasing subsequence
- Edit distance $O(nm)$

NP-completeness

- NP, NP-hard, NP-complete
- Clique, Vertex Cover, 3SAT
- Independent Set
- Partition
- Hamiltonian Cycle
- TSP
- Knapsack

Approxmation Algorithms

- List Scheduling (Greedy)
- Vertex Cover (MVC)
- Christofides Algorithm for $\triangle$TSP, metric-TSP

Hardness of Approximations

- General-TSP is NP-hard to approximate.
- Chromatic Number is NP-hard to additively approximate.

Online Algorithms

- Paging
- k-server
- (online coloring, not on exam)

Examples of competitive analysis, compete to something stronger and then show how closely you are. Can't differentiate using standard analysis. Close to optimal for all inputs.

- Randomized Marking Mouse $O(\log m)$-competitive

Hashing

- Universal set of hash functions
- Cuckoo Hashing
- Analysing properties of Cuckoo graphs