

# Algorithms Final Cheat Sheet

## Processes and DB

Probability of any success =  $p * (1 - p)^n - 1 \dots$

$p = \Theta(1/n) \dots t = t$ ; prob of fail  $\leq 1/e \dots$

$t = en * \ln(n)$ ; prob of fail  $\leq n^{-c} \dots$  How long until succeeding at least once:  $n^{-1}$

## The Union Bound

$\text{Prob} [\bigcup_{i=1}^{\infty} F_i] \leq \sum_{n=1}^{\infty} \text{Prob}[F_i]$

## Verifying AB=C

$B\bar{r} \rightarrow A(B\bar{r})$  and  $C\bar{r}$ . if  $A(B\bar{r}) \neq C\bar{r}$  then  $AB \neq C$

## Principle of Deferred Decisions

If  $AB \neq C$  and  $\bar{r}$  is chosen uniformly at random with  $r_n$  at 0-1 then  $\text{Prob}(AB\bar{r} = C\bar{r}) \leq 1/2$

## Law of Total Probability

Let  $E_1 \dots E_n$  be mutually disjoint events in the sample space  $\Omega$  and let  $\bigcup_{i=1}^n E_i$  then  $\sum_{i=1}^n \text{Prob}[B|E_i] \text{Prob}[E_i]$

Repeated trials increase the runtime to  $\Theta(kn^2)$  If it returns false, then it this is right, but if it returns true, then it returns so with some probability of mistake.

## Median Value

$\sum_{j=1}^{\infty} j * \text{Prob}[X = j] = (n + 1)/2$  while  $\text{Prob}[X = j] = 1/n$

To take exactly  $j$  steps:  $\text{Prob}[X = j] = (1 - p)^{j-1} p$  1/p for the first success

## Median Value

Linearity of Expectation: Given 2 random vars  $X$  and  $Y$  in the same probability space,  $E[X + Y] = E[X] + E[Y]$

## Memoryless guessing

Expected correct: 1, independent of  $n$

## Memory guessing

$H(n) = \Theta(\log(n))$  [harmonic series]

## Coupon collection:

$E[X_j] = n/(n - j)$ ;  $n$  = number of;  $j$  = collected;

$(n - j)/n$  of getting a new one

$E[X] = nH(n) = \Theta(n \log(n))$

## Conditional Probability

$E[X|\alpha] = \sum_{j=0}^{\infty} j * \text{Prob}[X = j|\alpha]$

## Max 3-SAT

There is a randomized algo with polynomial expected run time that is guaranteed to produce a truth assignment satisfying at least a 7/8 fraction of all clauses. We would need  $8k$  trials to get the satisfying assignment.

```
1: function SELECT( $S, k$ )
2:    $a_i \leftarrow \text{randomvarin } S$ 
3:   for each element  $a_j$  of  $S$  do
4:      $S^-.$ append( $a_j$ ) if  $a_j < a_i$ 
5:      $S^+.$ append( $a_j$ ) if  $a_j > a_i$ 
6:   end for
7:   if  $|S^-| = k - 1$  then
8:     return  $a_i$ 
9:   else if  $|S^-| \geq k$  then
10:    Select( $S^-, k$ )
11:   else
12:    Select( $S^+, k - 1 - |S^-|$ )
13:   end if
14: end function
```

## Quicksort Expected Comparisons

$2n \ln(n) + O(n), \Omega(n \log n)$

## Uniform

$\text{Prob}[h(x) = i] = 1/m$  for all  $x$  and all  $i$

## Universal

$\text{Prob}[h(x) = h(y)] = 1/m$  for all  $x \neq y$

## Near-universal

$\text{Prob}[h(x) = h(y)] \leq 2/m$  for all  $x \neq y$ ;  $E[\text{chainLen}] \leq 2 * \alpha$ ; Runtime:  $\Theta(1 + \alpha)$

## k-uniform

$\text{Prob}[\bigwedge_{j=1}^k h(x_j) = i_j] = 1/m^k$  for all distinct  $x_1 \dots x_k$  and all  $i_1 \dots i_k$

## Load Factor

$\alpha = n/m$

## Balanced Binary Tree Search

$O(1 + \log(\text{chainLen}))$  with any hash  $O(1 + \log(\alpha))$  for uniform

## Recursively Hash

$O(\log_m n)$

## Expected Search w/ binary probe

$O(1)$

## Formula Satisfiability/SAT

$(a \vee b \vee c \vee \bar{d}) \Leftrightarrow ((b \wedge \bar{c}) \vee \overline{(a \Rightarrow d)} \vee (c \neq a \wedge b))$

## CNF

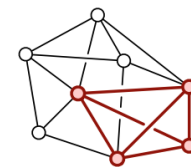
conjunction/AND of several clauses which use OR inside these clauses.

## 3CNF

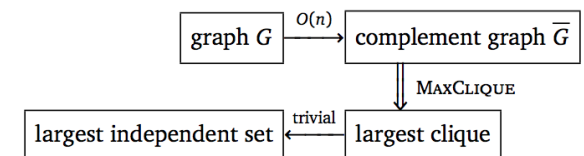
cnf with exactly 3 literals per clause

## Maximum Independent Set (from 3Sat)

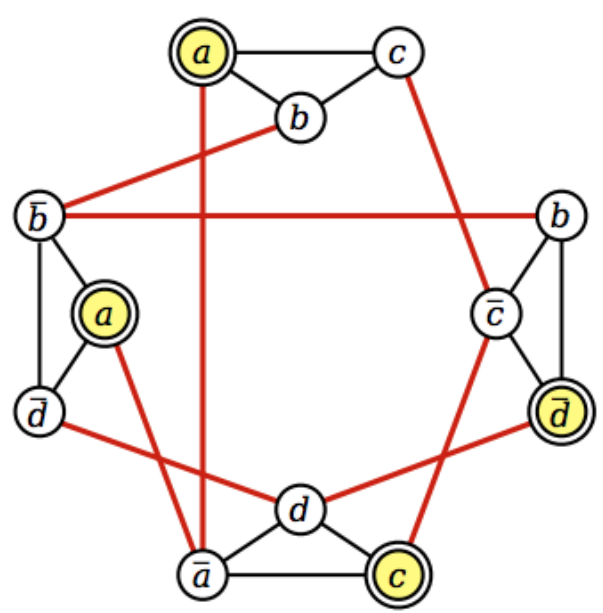
input is a simple, unweighted graph, get the size of the largest/smallest subgraph Make the formula into a graph or vice versa, if it has an independent set of size  $k$ , its possible Any graph has an edge-complement with the same vertices but the opposite set of edges if its not an edge in  $G$ . Its independent in  $G$  if and only if the same vertices define a clique in  $\bar{G}$  (a complete graph). The largest independent is thus the largest clique in the compliment of the graph.



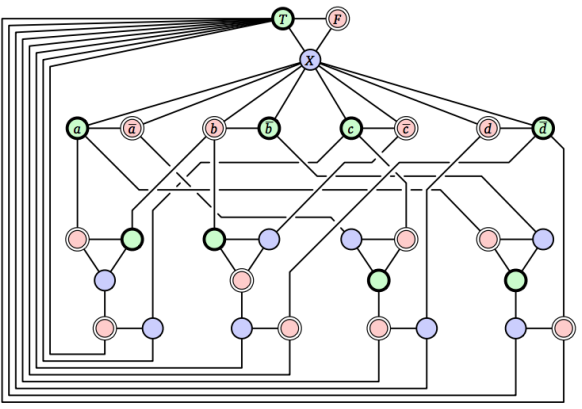
A graph with maximum clique size 4.



3Color [From 3SAT]



The truth gadget and a variable gadget for  $a$ .



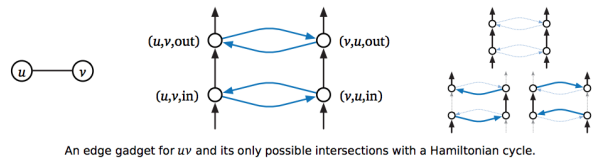
A 3-colorable graph derived from the satisfiable 3CNF formula  $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

Truth gadget:  $T$ ,  $F$ , and  $X$  for true/false/other, variable

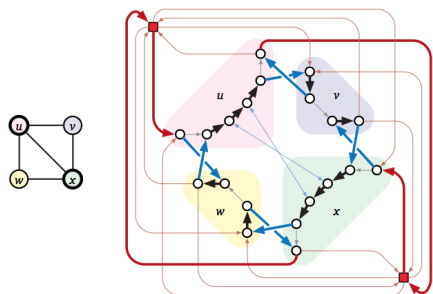
gadget for variable  $a$  connecting and  $\bar{a}$  which must be opposite bools. Clause gadget joining three literal nodes to node  $T$  in the truth gadget using give new unlabeled nodes and ten edges.

Hamiltonian Cycle [From Vertex Cover]

For each vertex  $u$ , all the edge gadgets are connected in  $H$  into a single directed path, a vertex chain.  $H$  has  $d - 1$  additional edges for each  $i$ .  $H$  also contains  $k$  cover vertices,  $1 - k$ , with a directed edge to the first vertex in each vertex chain and a directed edge from the last vertex in each vertex chain. Start at cover vertex 1 and traverse vertex chain for  $vu_2$ , then visit cover vertex 2 and so on and so forth before returning to 1. If  $v$  is a part of the vertex cover, follow the edge from  $(u_i, v, in)$  to  $(u_i, v, out)$ , else, detour from  $(u_i, v, in) > (v, u_i, in) > (v, u_i, out) > (u_i, v, out)$ .  $G$  contains a vertex cover of size  $K$  if and only if  $H$  contains a Hamiltonian cycle



An edge gadget for  $uv$  and its only possible intersections with a Hamiltonian cycle.



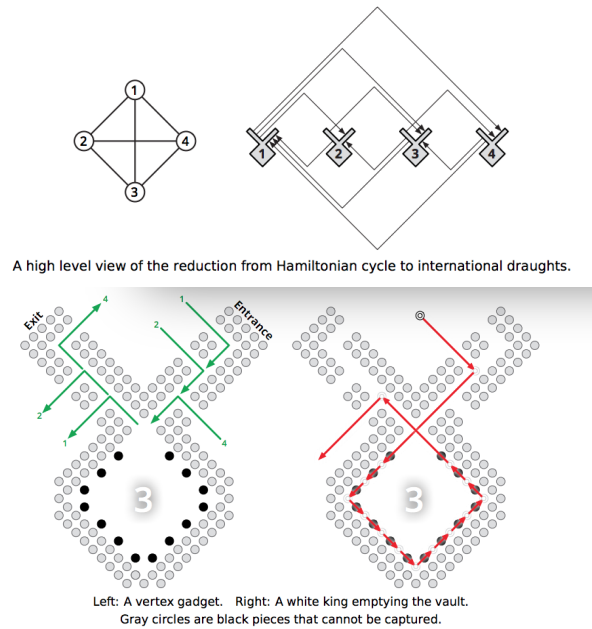
A vertex cover  $\{u, x\}$  in  $G$  and the corresponding Hamiltonian cycle in  $H$ .

Subset Sum

number edges from  $0$  to  $m - 1$ ; set  $X$  contains the integer  $b_i := f^i$  for each edge  $i$ , and the integer  $a_v := f^m + \sum_{i \in \delta(v)} 4^i$  where  $\delta(v)$  is the set of edges that have  $v$  as an endpoint. [  $X$  is a  $(m + 1)$  digit number written in base 4). The  $m$ th digit is 1 if vertex, 0 otherwise.  $t := k * 4^m + \sum_{i=0}^{m-1} 2 * 4^i$

Draughts

White can capture a certain number of black pieces in a single move if and only if  $G$  has a Hamiltonian cycle. Replace edges  $uv$  with  $u \rightarrow v$  and  $v \rightarrow u$ . If there is a path you can capture them all, if there isn't, you can capture at most  $1/2$  of them.

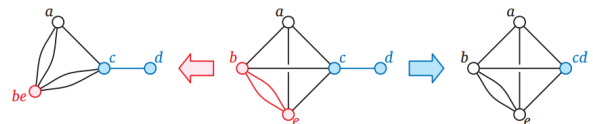


Left: A vertex gadget. Right: A white king emptying the vault. Gray circles are black pieces that cannot be captured.

LIS -> DAG

Every number will be a node. Number-node  $A$  has an edge to number-node  $B$  iff 1.  $B$  comes after  $A$  in the sequence, 2.  $B$  is greater than  $A$  in value. There's a special node end. Every node has an edge to end. Every path on this graph is a valid increasing subsequence. The problem of finding the LIS is now the problem of finding the longest path on this graph. Apply DAG.

Max Flows and Min Cuts



Blind Guess

1: **function** GUESSMINCUT( $G$ )

```

2:   for  $i \leftarrow n, 2$  do
3:       pick a random edge  $e$  in  $G$ 
4:        $G \leftarrow G/e$ 
5:   end for
6:   return the only cut in  $G$ 
7: end function

```

$$P(n) = \frac{2}{n(n-1)}$$

### Repeated Guessing

```

1: function KARGERMINCUT( $G$ )
2:    $minX \leftarrow \infty$ 
3:   for  $i \leftarrow 1, N$  do
4:        $X \leftarrow \text{GUESSMINCUT}(G)$ 
5:       if  $|X| < minX$  then
6:            $minX \leftarrow |X|$ 
7:            $minX \leftarrow X$ 
8:       end if
9:   end for
10:  return  $minX$ 
11: end function

```

Set  $N = c \binom{n}{2} \ln n$  for some constant  $c$ .  $P(n) \geq 1 - \frac{1}{n^c}$ . KARGERMINCUT computes the min cut of any  $n$ -node graph with high probability in  $O(n^4 \log n)$  time.

### Not-So-Blindly Guessing

```

1: function CONTRACT( $G, m$ )
2:   for  $i \leftarrow n, m$  do
3:       pick a random edge  $e$  in  $G$ 
4:        $G \leftarrow G/e$ 
5:   end for
6: end function
7: function BETTERGUESS( $G$ )
8:   if  $G$  has more than 8 vertices then
9:        $G_1 \leftarrow \text{CONTRACT}(G, n/\sqrt{2} + 1)$ 
10:       $G_2 \leftarrow \text{CONTRACT}(G, n/\sqrt{2} + 1)$ 
11:       $X_1 \leftarrow \text{BETTERGUESS}(G_1)$ 
12:       $X_2 \leftarrow \text{BETTERGUESS}(G_2)$ 
13:      return  $\min(X_1, X_2)$ 
14:   else
15:       use brute force
16:   end if
17: end function

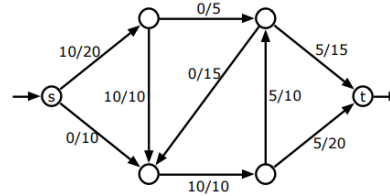
```

$P(n) \geq 1/\log n$ . The running time is  $O(n^2 \log n)$ .

### Flows

A *flow* is a function  $f$  that satisfies the *conservation constraint* at every vertex  $v$ : the total flow *into*  $v$  is equal to the total flow *out* of  $v$ .

A flow  $f$  is *feasible* if  $f(e) \leq c(e)$  for each edge  $e$ . A flow *saturates* edge  $e$  if  $f(e) = c(e)$ , and *avoids* edge  $e$  if  $f(e) = 0$ .

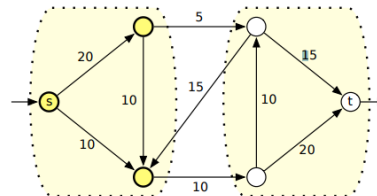


An  $(s, t)$ -flow with value 10. Each edge is labeled with its flow/capacity.

### Cuts

A *cut* is a partition of the vertices into disjoint subsets  $S$  and  $T$  - meaning  $S \cup T = V$  and  $S \cap T = \emptyset$  - where  $s \in S$  and  $t \in T$ .

If we have a capacity function  $c$ , the *capacity* of a cut is the sum of the capacities of the edges that start in  $S$  and end in  $T$ . The definition is asymmetric; edges that start in  $T$  and end in  $S$  are unimportant. The *min-cut problem* is to compute a cut whose capacity is as large as possible.

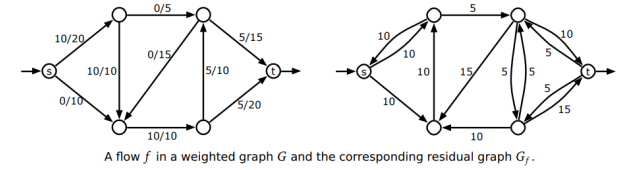


An  $(s, t)$ -cut with capacity 15. Each edge is labeled with its capacity.

**Theorem 1 (Maxflow Mincut Theorem)** In any flow network, the value of the maximum flow is equal to the capacity of the minimum cut.

### Residual Capacity

$$c_f(u \rightarrow v) = \begin{cases} c(u \rightarrow v) - f(u \rightarrow v) & \text{if } u \rightarrow v \in E \\ f(v \rightarrow u) & \text{if } v \rightarrow u \in E \\ 0 & \text{otherwise} \end{cases}$$



### Augmenting Paths

Suppose there is a path  $s = v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_r$  in the residual graph  $G_f$ . This is an *augmenting path*. Let  $F = \min_i c_f(v_i \rightarrow v_{i+1})$  denote the maximum amount of flow that we can push through the augmenting path in  $G_f$ . We can augment the flow into a new flow function  $f'$ :

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) + F & \text{if } u \rightarrow v \in s \\ f(u \rightarrow v) - F & \text{if } v \rightarrow u \in s \\ f(u \rightarrow v) & \text{otherwise} \end{cases}$$

### Ford-Fulkerson

Starting with the zero flow, repeatedly augment the flow along *any* path from  $s$  to  $t$  in the residual graph, until there is no such path.

### Further Work

The fastest known maximum flow algorithm, announced by James Orlin in 2012, runs in  $O(VE)$  time.

### Flow/Cut Applications

#### Edge-Disjoint Paths

A set of paths in  $G$  is *edge-disjoint* if each edge in  $G$  appears in at most one of the paths; several edge-disjoint paths may pass through the same vertex, however. Assign each edge capacity 1. The number of edge-disjoint paths is exactly equal to the value of the flow. Using Orlin's algorithm is overkill; the maximum flow has value at most  $V - 1$ , so Ford-Fulkerson's original augmenting path algorithm also runs in  $O(|f^*| E) = O(VE)$  time.

#### Vertex Capacities and Vertex-Disjoint Paths

If we require the total flow into (and out of) any vertex  $v$  other than  $s$  and  $t$  is at most some value  $c(v)$ , we transform the input into a new graph. We replace each vertex  $v$  with two vertices  $v_{in}$  and  $v_{out}$ , connected by an edge  $v_{in} \rightarrow v_{out}$  with capacity  $c(v)$ , and then replace

every directed edge  $u \rightarrow v$  with the edge  $u_{out} \rightarrow v_{in}$  (keeping the same capacity).

Computing the maximum number of *vertex-disjoint* paths from  $s$  to  $t$  in any directed graph simply involves giving every vertex capacity 1, and computing a maximum flow. SATandCNFSAT

## NP-Hardness

**Definition 2**  $P$  is the set of decision problems that can be solved in polynomial time. Intuitively,  $P$  is the set of problems that can be solved quickly.

**Definition 3** NP is the set of decision problems with the following property: if the answer is Yes, then there is a proof of this fact that can be checked in polynomial time. Intuitively, NP is the set of decision problems where we can verify a Yes answer quickly if we have the solution in front of us.

**Definition 4** co-NP is essentially the opposite of NP. If the answer to a problem in co-NP is No, then there is a proof of this fact that can be checked in polynomial time.

Every decision problem in  $P$  is also in NP and also in co-NP.

**Definition 5** A problem  $\Pi$  is NP-hard if a polynomial-time algorithm for  $\Pi$  would imply a polynomial-time algorithm for every problem in NP.

**Definition 6** A problem  $\Pi$  is NP-complete if it is both NP-hard and an element of NP.

**Theorem 7 (Cook-Levin Theorem)** Circuit satisfiability is NP-complete.

To prove that problem  $A$  is NP-hard, reduce a known NP-hard problem to  $A$ .

**Definition 8** A many-one reduction from one language  $L' \subseteq \Sigma^*$  is a function  $f : \Sigma^* \rightarrow \Sigma^*$  such that  $x \in L'$  iff  $f(x) \in L$ . A language  $L$  is NP-hard iff, for any language  $L' \in NP$ , there is a many-one reduction from  $L'$  to  $L$  that can be computed in polynomial time.

## NP-Hard Problems

- SAT
- 3SAT
- Maximum Independent Set: find the size of the largest subset of the vertices of a graph with no edges between them
- Clique: Compute the number of nodes in its largest complete subgraph
- Vertex Cover: Smallest set of vertices that touch every edge in the graph
- Graph Coloring: Find the smallest possible number of colors in a legal coloring such that every edge has two different colors at its endpoints
- Hamiltonian Cycle: find a cycle that visits each vertex in a graph exactly once
- Subset Sum: Given a set  $X$  of positive integers and an integer  $t$ , determine whether  $X$  has a subset whose elements sum to  $t$
- Planar Circuit SAT: Given a boolean circuit that can be embedded in the plane so that no two wires cross, is there an input that makes the circuit output True
- Not All Equal 3SAT: Given a 3CNF formula, is there an assignment of values to the variables so

that every clause contains at least one True literal *and* at least one False literal?

- Exact 3-Dimensional Matching: Given a set  $S$  and a collection of three-element subsets of  $S$ , called *triples*, is there a sub-collection of disjoint triples that exactly cover  $S$ ?
- Partition: Given a set  $S$  of  $n$  integers, are there subsets  $A$  and  $B$  such that  $A \cup B = S$ ,  $A \cap B = \emptyset$ , and  $\sum_{a \in A} a = \sum_{b \in B} b$ ?
- 3Partition: Given a set  $S$  of  $3n$  integers, can it be partitioned into  $n$  disjoint three-element subsets, such that every subset has exactly the same sum?
- Set Cover: Given a collection of sets  $S = \{S_1, S_2, \dots, S_m\}$ , find the smallest sub-collection of  $S_i$ 's that contains all the elements of  $\bigcup_i S_i$
- Hitting Set: Given a collection of sets  $S = \{S_1, S_2, \dots, S_m\}$ , find the minimum number of elements of  $\bigcup_i S_i$  that hit every set in  $S$
- Hamiltonian Path: Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?
- Longest Path: Given a non-negatively weighted graph  $G$  and two vertices  $u$  and  $v$ , what is the longest simple path from  $u$  to  $v$  in the graph? A path is *simple* if it visits each vertex at most once.
- Steiner Tree: Given a weighted, undirected graph  $G$  with some of the vertices marked, what is the minimum-weight subtree of  $G$  that contains every marked vertex?