Name: Trevor Buck

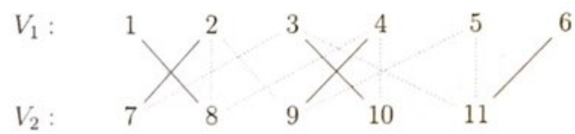
ID: 109081318

**CSCI 3104** **Profs. Grochow & Layer**

**Problem Set 10** **Spring 2019, CU-Boulder**

Quick links 1a 1b 1c 2a 2b 2c 3a 3b

1. A *matching* in a graph $G$ is a subset $E_M \subseteq E(G)$ of edges such that each vertex touches at most one of the edges in $E_M$. Recall that a bipartite graph is a graph $G$ on two sets of vertices, $V_1$ and $V_2$, such that every edge has one endpoint in $V_1$ and one endpoint in $V_2$. We sometimes write $G = (V_1, V_2; E)$ for this situation. For example:




The edges in the above example consist of all the lines, whether solid or dotted; the solid lines form a matching.

The *bipartite maximum matching* problem is to find a matching in a given bipartite graph $G$ , which has the maximum number of edges among all matchings in $G$.

Name: Trevor Buck

ID: 109081218



(a) (6 pts total) Prove that a maximum matching in a bipartite graph $G = (V_1, V_2; E)$ has size at most $\min\{|V_1|, |V_2|\}$.

$V_1$ has size $n$
$V_2$ has size $m$

3 options:

① $n < m$

maximum matching

$V_{11} \leftrightarrow V_{21}$

$V_{12} \leftrightarrow V_{22}$

$V_{13} \leftrightarrow V_{23}$

⋮

$V_{1n} \leftrightarrow V_{2n}$

⋮

Stop here because every element in $V_1$ is matched.

#matches = $n$

② $n = m$

$V_{11} \leftrightarrow V_{21}$

$V_{12} \leftrightarrow V_{22}$

$V_{13} \leftrightarrow V_{23}$

⋮

$V_{1n} \leftrightarrow V_{2m}$

stop because both sets of vertices are all matched

#matches = $n = m$

③ $n > m$

" "

" "

⋮

$V_{1m} \leftrightarrow V_{2m}$

Stop because every element in $V_2$ is matched

#matches = $m$

* IN EVERY CASE, the maxmum number in every possible outcome is bounded by the lower number ∎

Name: Trevor Buck
ID: 109081318



(b) (6 pts total) Show how you can use an algorithm for max-flow to solve bipartite maximum matching on undirected simple bipartite graphs. That is, give an algorithm which, given an undirected simple bipartite graph $G = (V_1, V_2; E)$, (1) constructs a directed, weighted graph $G'$ (which need not be bipartite) with weights $w : E(G') \to \mathbb{R}$ as well as two vertices $s, t \in V(G')$, (2) solves max-flow for $(G', w), s, t$, and (3) uses the solution for max-flow to find the maximum matching in $G$. Your algorithm may use any `max-flow` algorithm as a subroutine.

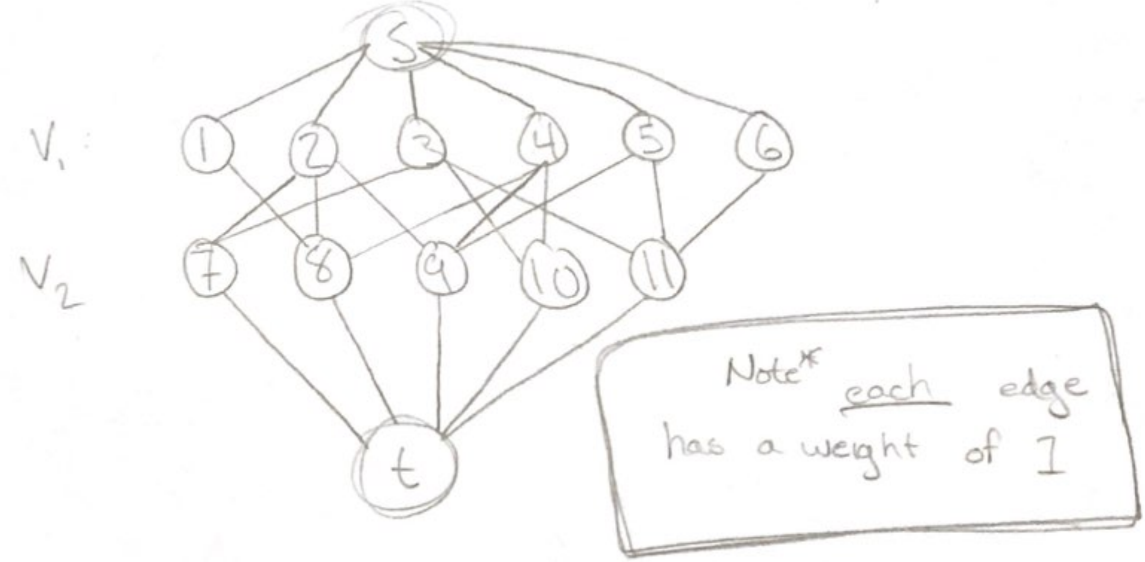
Given: Bi-partite graph $G = (V_1, V_2; E)$

1. I Create $G'$ by adding two vertices which will act as a sink and source.
   II add an edge from 'S (source) to every vertex in $V_1$ with capacity of 1
   III add an edge from T' (sink) to every vertex in $V_2$ with capacity of 1
   IV. For every edge in $G$, recreate that edge in $G'$ with capacity of 1

2. solve for max flow in $G'$ using the max flow algorithm

(3) Because every edge has 'weight = 1'
the max-flow algorithm will take
as many paths as it can find between
$V_1$ and $V_2$. It's important to note that
our algorithm can never return to S
and will only visit T once. (and end there).
The answer will be the wherever
our max flow algorthm goes from
a Vertex in $V_1$ to a vertex in $V_2$.

(c) (7 pts total) Show the weighted graph constructed by your algorithm on the example bipartite graph above.



Name: Trevor Buck
ID: 109081318

**CSCI 3104**
**Problem Set 10**

**Profs. Grochow & Layer**
**Spring 2019, CU-Boulder**

2. In the review session for his Deep Wizarding class, Dumbledore reminds everyone that the logical definition of NP requires that the number of *bits* in the witness $w$ is polynomial in the number of bits of the input $n$. That is, $|w| = poly(n)$. With a smile, he says that in beginner wizarding, witnesses are usually only logarithmic in size, i.e., $|w| = O(\log n)$.

(a) (7 pts total) Because you are a model student, Dumbledore asks you to prove, in front of the whole class, that any such property is in the complexity class P.

- We know that 'log(n)' is a form of poly(n). This means that any property with a time constant of O(logn) is, at worst case, solvable in polynomial time. Making it part of P.

This holds true to the statement

'P ∈ NP'

and, in this special case, NP is even reducible to P.

(b) (6 pts total) Well done, Dumbledore says. Now, explain why the logical definition of NP implies that any problem in NP can be solved by an exponential-time algorithm.

If a problem is verified quickly (ie. in NP) then that problem can be solved by any algorithm that uses a 'process of elimination' approach. The algorithm would just have to try all possible solutions. This is often in exponential time, but it can't be worse than that :)

Take soduku for example. There are only so many possible solutions. You can solve a soduku in exponential time, and then verify it rather quickly in NP time

(c) (6 pts total) Dumbledore then asks the class: "So, is NP a good formalization of the notion of problems that can be solved by brute force? Discuss." Give arguments for both possible answers.

| Arguments for yes | Arguments for no |
|---|---|
| • By applying 'brute force' any algorithm can be solved in exponential time | • There are problems of class 'P' that won't take exponential time to solve |
| • Because if an algorithm is decidable it will always produe an output | • Brute Force can apply to much more algorithms than just 'NP' |

Name: Trevor Buck
ID: 109081318

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

3. (20 pts ) Recall that the *MergeSort* algorithm is a sorting algorithm that takes $\Theta(n \log n)$ time and $\Theta(n)$ space. In this problem, you will implement and instrument MergeSort, then perform a numerical experiment that verifies this asymptotic analysis. There are two functions and one experiment to do this.

   (i) `MergeSort(A,n)` takes as input an unordered array $A$, of length $n$, and returns both an in-place sorted version of $A$ and a count $t$ of the number of atomic operations performed by `MergeSort`.

   (ii) `randomArray(n)` takes as input an integer $n$ and returns an array $A$ such that for each $0 \leq i < n$, $A[i]$ is a uniformly random integer between 1 and $n$. (It is okay if $A$ is a random permutation of the first $n$ positive integers.)

Name: Trevor Buck

ID: 109081318

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

(a) (10 pts total) From scratch, implement the functions `MergeSort` and `randomArray`. You may not use any library functions that make their implementation trivial. You may use a library function that implements a pseudorandom number generator in order to implement `randomArray`.

Submit a paragraph that explains how you instrumented `MergeSort`, i.e., explain which operations you counted and why these are the correct ones to count.

```
int * randomArray (n) {
    int * r = (int*) malloc(len * sizof(int));
    Int i:

    for (i=0; i<n; i++) {
        r[i] = random ();
    }
    return r;
}
```

```
void merge (int arr[], int i, int mid, int j) {
    int *temp [length];   ↙ global
    int L = i;
    int r = j;
    int m= mid+1;
    int k = L;
    while (L ≤ mid && m<=r) {
        if (arr[L] <= arr[mid]) { temp[k++] = arr[L++] }
        else {temp[k++] = arr[m++]; }
    }
    while (l<= mid) { temp[k++] = arr[L++]}
    while (m <= r) { temp [k++] = arr [m++]}
    return *temp
}
```

---

```
void mergeSort (int arr[], int i=0, int j ) {
    int mid = 0
    if (i<j) {
        mid = (i+j) /2 ;
        mergeSort(arr, i, mid); ①
        mergeSort(arr, mid +1, j); ②
        merge (arr, i, mid, j); ③
    } operations += 3 ;
}
```

3 atomic operations per call

(b) (10 pts total) For each of $n = \{2^4, 2^5, \ldots, 2^{26}, 2^{27}\}$, run `MergeSort(randomArray(n),n)` fives times and record the tuple $(n, \langle t \rangle)$, where $\langle t \rangle$ is the average number of operations your function counted over the five repetitions. Use whatever software you like to make a line plot of these 24 data points; overlay on your data a function of the form $T(n) = A\, n \log n$, where you choose the constant $A$ so that the function is close to your data.

Hint 1: To increase the aesthetics, use a log-log plot.

Hint 2: Make sure that your *MergeSort* implementation uses only two arrays of length $n$ to do its work. (For instance, don't do recursion with pass-by-value.)

```
Length of array: 16  -> Num of ops: 45
Length of array: 32  -> Num of ops: 138
Length of array: 64  -> Num of ops: 327
Length of array: 128  -> Num of ops: 708
Length of array: 256  -> Num of ops: 1473
Length of array: 512  -> Num of ops: 3006
Length of array: 1024  -> Num of ops: 6075
Length of array: 2048  -> Num of ops: 12216
Length of array: 4096  -> Num of ops: 24501
Length of array: 8192  -> Num of ops: 49074
Length of array: 16384  -> Num of ops: 98223
Length of array: 32768  -> Num of ops: 196524
Length of array: 65536  -> Num of ops: 393129
Length of array: 131072  -> Num of ops: 786342
Length of array: 262144  -> Num of ops: 1572771
Length of array: 524288  -> Num of ops: 3145632
Length of array: 1048576  -> Num of ops: 6291357
Segmentation fault (core dumped)
```

→ $A = 0.70$ (for 16)

→ $A = 0.46$ (for 4096)

→ $A = 0.299$ (for 1048576)

* note couldn't get a graph to work

$\boxed{A \approx 0.5 \text{ on average}}$