CS320 Algorithms: Theory and Practice

Course Introduction

“For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.” - Francis Sullivan

Course Objectives

Algorithms:
- Design – strategies for algorithmic problem solving
- Your algorithmic toolbox: Graph algorithms, greedy, divide and conquer, dynamic programming, reductions
- Reasoning about algorithm correctness
- Analysis of time and space complexity
- Implementation – create an implementation that respects the runtime analysis

Grading

Assignments 40% (25% programming, 15% written)
Quizzes 10%
Midterm 20%
Final 30%
Course staff

Asa Ben-Hur (instructor)  
Zhisheng Xu (TA)

Implementation

Programs will be written in Python

- A concise and powerful interpreted language
- Powerful data structures
  - tuples, lists, dictionaries
  - Simple, easy to learn syntax
  - Highly readable, compact code
- Supports object oriented and functional programming
- An extensive standard library
- Strong support for integration with other languages (C, C++, Java)

Why I love Python

I am more productive!

- Machine performance vs. programmer performance

Makes programming fun

image from: http://www.mindview.net/pub/eckel/LovePython.zip

Which version?

2.x or 3.x? Stick with 2.x for now.

Python 3 is a non-backward compatible version that removes a few "warts" from the language.

Does anyone else use python?

One of the three "official languages" in google.  
Peter Norvig, Director of Research at Google:  
"Python has been an important part of Google since the beginning, and remains as as the system grows and evolved. Today dozens of Google engineers use Python, and we’re looking for more people with skills in this language"  

Yahoo groups, Yahoo maps -- 100% python

How will we learn Python?

- Course website has links to Python tutorials and other resources
- Special lab times to help you get up to speed
HW #1

Play with graphs and adjacency matrices

Our first problem

Problem: Matching residents to hospitals

- Each resident has a preference list of hospitals
- Each hospital has a preference list of residents

We would like a method for assigning residents to hospitals that is “stable”.

NRMP (National Resident Matching Program)

- Matching system in use since the early 50s.
- Addressed issues of competition between hospitals.
- Handles over 38,000 residents yearly.

Simplifying the problem

Matching students/companies problem a bit messy:
- Hospital may look for multiple residents
- Students looking for a single residency
- Maybe there are more residencies than students, or fewer residencies than students

Formulate a “bare-bones” version of the problem

The Stable Matching Problem

Problem: Given \( n \) men and \( n \) women where

- Each man lists women in order of preference
- Each woman lists men in order of preference

Find a stable matching of all men and women

Note that the preference lists are a strict total order

Stable Matching Problem

Goal: Given a set of preferences among hospitals and students, design a stable matching process.

Unstable pair: resident \( x \) and hospital \( y \) are an unstable pair if:

- \( x \) prefers \( y \) to its assigned hospital.
- \( y \) prefers \( x \) to one of its admitted applicants.

Note that \( (x,y) \) is not part of the current matching

Stable assignment: Assignment with no unstable pairs.

Stable matching problem: Given the preference lists of \( n \) men and \( n \) women, find a stable matching if one exists.
Formulation

Men: $M = \{m_1, \ldots, m_n\}$  
Women: $W = \{w_1, \ldots, w_n\}$  
The cartesian product $M \times W$ is the set of all ordered pairs.  
A perfect matching $S$ is a set of pairs (subset of $M \times W$) such that each individual occurs in exactly one pair.  
How many perfect matchings are there?

Instability

Given a perfect matching, eg  
$S = \{(m_1, w_1), (m_2, w_2)\}$  
But $m_1$ prefers $w_2$ and $w_2$ prefers $m_1$  
$(m_1, w_2)$ is an instability for $S$  
Notice $(m_1, w_2)$ is not in $S$!  
$S$ is a stable matching if:  
* $S$ is perfect  
* and there is no instability wrt $S$

Example 1

$m_1$: $w_1, w_2$  
$m_2$: $w_1, w_2$  
$w_1$: $m_1, m_2$  
$w_2$: $m_1, m_2$  

1. $\{(m_1, w_1), (m_2, w_2)\}$  
2. $\{(m_1, w_2), (m_2, w_1)\}$  
which is stable/unstable?

Example 2

$m_1$: $w_1, w_2$  
$m_2$: $w_2, w_1$  
$w_1$: $m_1, m_2$  
$w_2$: $m_1, m_2$  

1. $\{(m_1, w_1), (m_2, w_2)\}$  
2. $\{(m_1, w_2), (m_2, w_1)\}$  
which is / are unstable/stable?  
Conclusion?
A naive algorithm

for S in the set of all perfect matchings:
  if S is stable: return S
return None

Is this algorithm correct?
What is its running time?

Towards an algorithm

initially: no match

An unmatched man m proposes to the woman w highest on his list.
Will this be part of a stable matching?

Towards an algorithm

initially: no match

An unmatched man m proposes to the woman w highest on his list.
Will this be part of a stable matching?
Not necessarily: w may like some m' better

So w and m will be in a temporary state of engagement: w is prepared to change her mind when a man higher on her list proposes

A couple of questions...

* Given a preference list, does a stable matching exist?
* Can we efficiently construct a stable matching if there is one?

While not everyone is matched...

An unmatched man m proposes to the woman w highest on his list that he hasn't proposed to yet.
If w is free, they become engaged
If w is engaged to m':
  If w prefers m' over m, m stays free
  If w prefers m over m' (m,w) become engaged
The Gayle-Shapley algorithm

Initialize each person to be free.
while (some man is free and hasn’t proposed to every woman)
  Choose such a man \( m \) such that \( w \) is highest-ranked woman on \( m \)’s list to whom \( m \) has not yet proposed
  if (\( w \) is free)
    \((m,w)\) become engaged
  else if (\( w \) prefers \( m \) to her fiancé \( m' \))
    \((m,w)\) become engaged, \( m' \) becomes free
  else
    \( m \) remains free


A few non-obvious questions:
How long does it take?
Does the algorithm return a stable matching?
Does it even return a perfect matching?

Observations
Each woman remains engaged from the first proposal and the sequence of partners gets better.
Each man proposes to less and less preferred women and will not propose to the same woman twice.

Claim. The algorithm terminates after at most \( n^2 \) iterations of the while loop.

At each iteration a man proposes (only once) to a woman he has never proposed to, and there are only \( n^2 \) possible pairs \((m,w)\).
When the loop terminates, the matching is perfect.

**Proof:** By contradiction. Assume there is a free man, \( m \).
Because the loop terminates, \( m \) proposed to all women.
But then all women are engaged, hence there is no free man.
\[ \rightarrow \text{Contradiction} \]

Initialize each person to be free.

\[
\text{while (some man is free and hasn’t proposed to every woman)}
\]

\[
\text{Choose such a man } m.
\]

\[
\begin{align*}
& n \text{ = highest-ranked woman on } m \text{’s list to whom } m \text{ has not yet proposed.} \\
& \text{if } (n \text{ is free}) \\
& (m, n) \text{ become engaged.} \\
& \text{else if } (n \text{ prefers } m \text{ to her fiancé } m’). \\
& (m, n) \text{ become engaged, } m’ \text{ becomes free.} \\
& \text{else } m \text{ remains free.}
\end{align*}
\]

\[
\text{Proof of Correctness: Stability}
\]

**Claim:** No unstable pairs.

**Proof:**

- Consider an arbitrary pair \((m, w)\) which is not in the Gale-Shapley matching \( S^* \).

<table>
<thead>
<tr>
<th>new proposal is decreasing</th>
<th>order of preference</th>
<th>( S^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1: ( m ) never proposed to ( w ).</td>
<td>( m ) prefers his GS partner to ( w )</td>
<td>( m, w )</td>
</tr>
<tr>
<td>( \rightarrow (m, w) \text{ is stable}. )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  Case 2: \( m \) proposed to \( w \).

  | received the highest proposal | \( m \) prefers \( w \) to her fiancé \( m’ \). |
  | \( \rightarrow (m, w) \text{ is stable.} \) |

In either case \((m, w)\) is not an instability wrt \( S^* \).

\[
\text{Summary}
\]

**Stable matching problem.** Given \( n \) men and \( n \) women and their preferences, find a stable matching if one exists.

**Gale-Shapley algorithm.** Guaranteed to find a stable matching for any problem instance.

**Q.** How to implement the GS algorithm efficiently?

**Q.** If there are multiple stable matchings, which one does GS find?

**Which solution?**

- \( m_1 : w_1, w_2 \)
- \( m_2 : w_2, w_1 \)
- \( w_1 : m_2, m_1 \)
- \( w_2 : m_1, m_2 \)

Two stable solutions

1. \((m_1, w_1), (m_2, w_2)\)
2. \((m_2, w_2), (m_1, w_1)\)

GS will always find one of them (which?)

When will the other be found?

**Symmetry**

The stable matching problem is symmetric wrt to men and women, but the GS algorithm is asymmetric.

There is a certain unfairness in the algorithm:
If all men list different women as their first choice, they will end up with their first choice, regardless of the women’s preferences.

**Non-determinism**

- Notice the following line in the GS algorithm:
- **while (some man is free and hasn’t proposed to every woman)**

Choose such a man \( m \)

The algorithm does not specify WHICH

- Still, it can be shown that all executions of the algorithm find the same stable matching (see book).
Extensions: Matching Residents to Hospitals

In this setting, Men = hospitals, Women = med school residents.

Variant 1. Some participants declare others as unacceptable.

Variant 2. Unequal number of men and women.

Variant 3. Limited polygamy. Hospital X wants to hire 3 residents.

Variant 4. The stable roommates problem

http://www.nrmp.org/res_match/about_res/algorithms.html

It has been proven that lying does not get you a better match!

Variant 4. The stable roommates problem

Remember the problem solving paradigm

- Formulate it with precision (usually using mathematical concepts, such as sets, relations, and graphs)
- Design an algorithm
- Prove its correctness
- Analyze its complexity
- Implement respecting the derived complexity

Representative Problems

Interval Scheduling

You have a resource (hotel room, printer, lecture room, telescope, manufacturing facility...)

There are requests to use the resource in the form of start time $s_i$ and finish time $f_i$, such that $s_i < f_i$.

Objective: grant as many requests as possible.

Two requests $i$ and $j$ are compatible if they don’t overlap, i.e., $f_i < s_j$ or $f_j < s_i$.

Input: Set of jobs with start times and finish times.

Goal: Find maximum cardinality subset of compatible jobs.

Interval Scheduling

Algorithmic Approach

The interval scheduling problem is amenable to a very simple solution.

Now that you know this, can you think of it?

Hint: Think how to pick a first interval while preserving the longest possible free time...
Weighted Interval Scheduling

Input: Set of jobs with start times, finish times, and weights.
Goal: Find maximum weight subset of compatible jobs.

Bipartite Matching

Input: Bipartite graph.
Goal: Find maximum cardinality matching.

Stable matching was defined as matching elements of two disjoint sets.

We can express this in terms of graphs.

A graph is bipartite if its nodes can be partitioned in two sets X and Y, such that the edges go from one x in X to a y in Y.

Independent Set

Input: Graph.
Goal: Find maximum cardinality independent set.

A subset of nodes such that no two are joined by an edge.

Can you formulate interval scheduling as an independent set problem?

Independent set problem

- There is no known efficient way to solve the independent set problem.
- What does that mean?
- The only solution we have so far is trying all sub sets and finding the largest independent one.
- How many sub sets of a set of n nodes are there?

Representative Problems / Complexities

- Interval scheduling: $n \log(n)$ greedy algorithm.
- Weighted interval scheduling: $n \log(n)$ dynamic programming algorithm.
- Bipartite matching: polynomial max-flow based algorithm.
- Independent set: NP-complete (no known polynomial algorithm exists).
- Competitive facility location (see book): PSPACE-complete.
Algorithm

Algorithm: effective procedure

- mapping input to output
- effective: unambiguous, executable

Turing defined it as: “like a Turing machine”

program = effective procedure

Is there an algorithm for every possible problem?

Ulam’s problem

```python
def f(n):
    if (n==1) return 1
    elif (odd(n)) return f(3*n+1)
    else return f(n/2)
```

Steps in running f(n) for a few values of n:

1. 2, 1
2. 3, 10, 5, 16, 8, 4, 2, 1
3. 4, 2, 1
4. 6, 3, 10, 5, 16, 8, 4, 2, 1
5. 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
6. 8, 4, 2, 1
7. 9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
8. 10, 5, 16, 8, 4, 2, 1

Does f(n) always stop?

Nobody has found an n for which f does not stop
Nobody has found a proof (so there can be no algorithm deciding this.)
A generalization of this problem has been proven to be undecidable.

A problem P is undecidable, if there is no algorithm that produces P(x) for every possible input x.

The Halting Problem is undecidable

Given a program P and input x
- will P stop on x?

We can prove (cs420):
- the halting problem is undecidable

i.e. there is no algorithm Haltp(x) that for any program P and input x decides whether P stops on x.