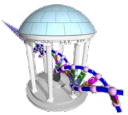
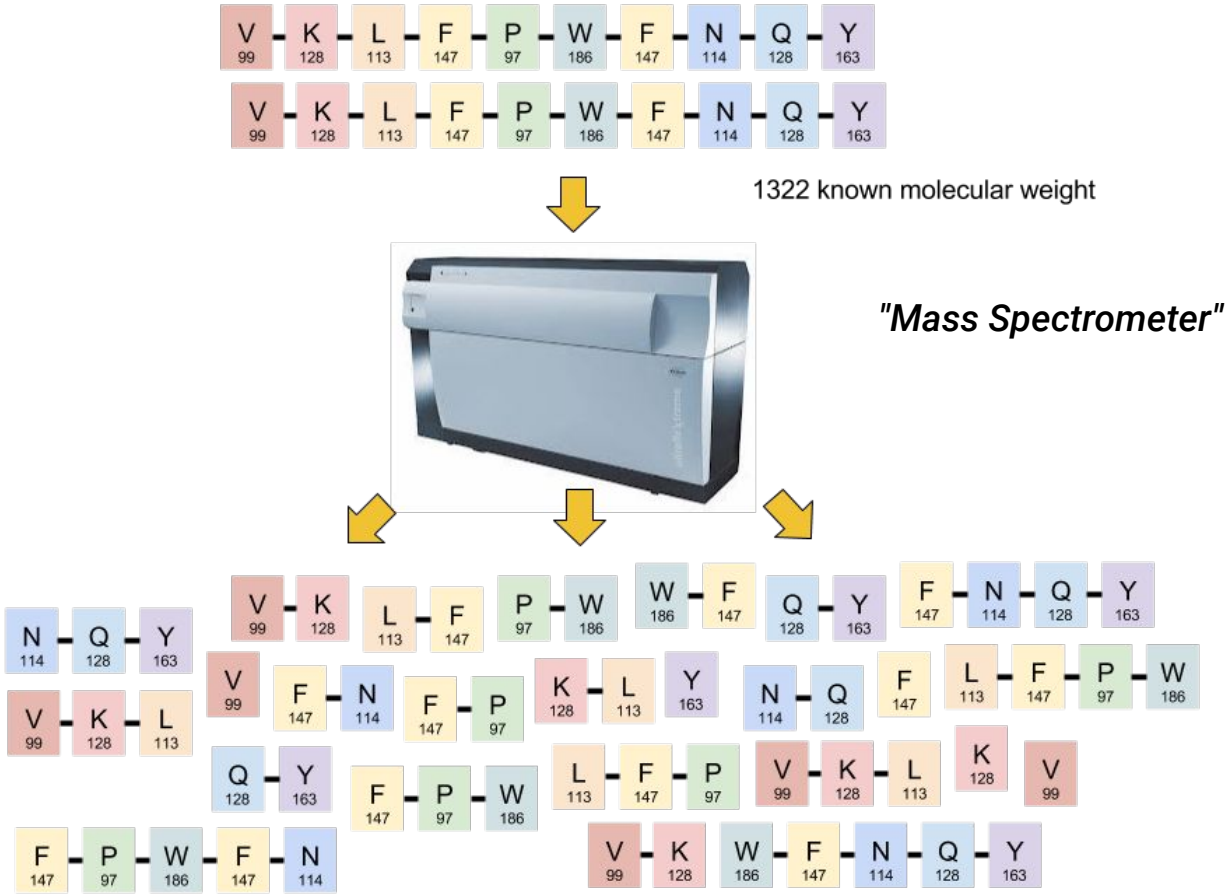
Comp 555 - BioAlgorithms - Spring 2022



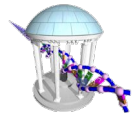
- We've discussed sequencing DNA, but there's another sequence in town
- To reconstruct it, we must first break it into pieces

Determining a Peptide's Sequence

Molecular Weights are the Puzzle Peices



Structure of a Peptide Chain



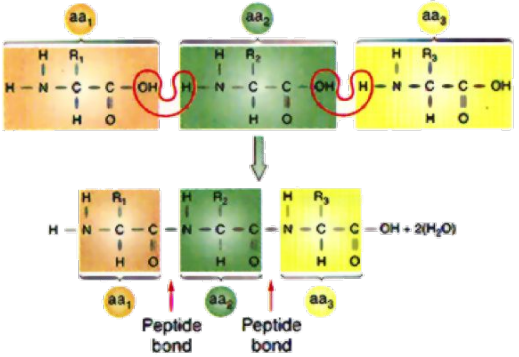
- Peptides are chains of amino acids that are joined by *peptide* bonds
- These bonds reduce the weight of each amino acid by one H_20 molecule
- The result is called a *residue*
- A Mass Spectrograph can precisely measure the molecular weight (and charge and abundance) of any peptide chain
- Since the molecular weight of each of the possible 20 residues is known precisely, one can ask the question, which combination of residues would give a particular weight?
- The problem is ambiguous for the entire molecule
  - Consider all permutations of 'PIT':

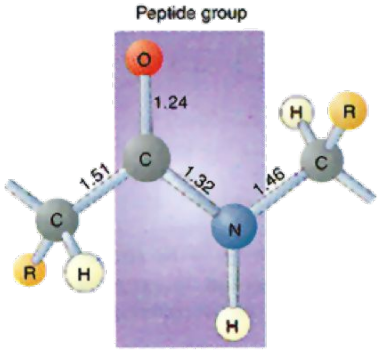
    `'PIT', 'PTI', 'ITP', 'IPT', 'TPI', and 'TIP' all weigh the same`

  - But they differ in their 2-peptide fragments:

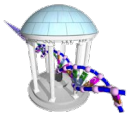
    `'PIT' breaks into 'PI' and 'IT',`
    `while 'PTI' breaks into 'PT' and 'TI'`

  - `However, 'TIP' breaks into fragments 'TI' and 'IP', which have the same weights as 'PI' and 'TI' respectively. Thus, we can't tell 'PIT' from 'TIP'`





An Simplified Peptide Weight table



- The actual molecular weight of an amino acid is a real number.
  This accounts for the relative abundances of atomic isotopes
- Today, we will use a simplified version that assumes only integer molecular weights

  Example:

  Molecular weight of Glycine Amino Acid

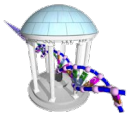
  $$W(C_2H_5NO_2) = 12\times2 + 5\times1 + 14 + 16\times2 = 75$$

  Molecular weight of Glycine Residue (Minus the H_2O lost forming the peptide bond)

  $$W(C_2H_5NO_2 - H_2O) = 57$$

- We can repeat this for all 20 Amino Acids to get a integer molecular weight table,
  which I name *Daltons*

Table Definitions



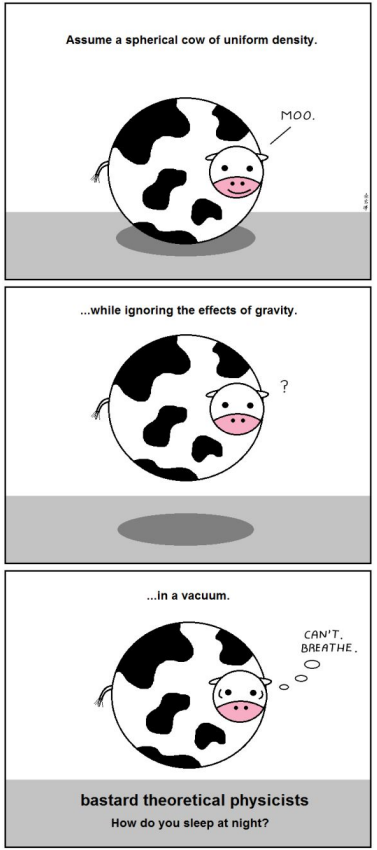
In [1]:
```
AminoAcid = {
    'A': 'Alanine', 'C': 'Cysteine', 'D': 'Aspartic acid', 'E': 'Glutamic acid',
    'F': 'Phenylalanine', 'G': 'Glycine', 'H': 'Histidine', 'I': 'Isoleucine',
    'K': 'Lysine', 'L': 'Leucine', 'M': 'Methionine', 'N': 'Asparagine',
    'P': 'Proline', 'Q': 'Glutamine', 'R': 'Arginine', 'S': 'Serine',
    'T': 'Theronine', 'V': 'Valine', 'W': 'Tryptophan', 'Y': 'Tyrosine',
    '*': 'STOP'
}

AminoAbbrv = {
    'A': 'Ala', 'C': 'Cys', 'D': 'Asp', 'E': 'Glu',
    'F': 'Phe', 'G': 'Gly', 'H': 'His', 'I': 'Ile',
    'K': 'Lys', 'L': 'Leu', 'M': 'Met', 'N': 'Asn',
    'P': 'Pro', 'Q': 'Gln', 'R': 'Arg', 'S': 'Ser',
    'T': 'Thr', 'V': 'Val', 'W': 'Trp', 'Y': 'Tyr',
    '*': 'STP'
}

# Here's a new dictionary!
Daltons = {
    'A':  71, 'C': 103, 'D': 115, 'E': 129,
    'F': 147, 'G':  57, 'H': 137, 'I': 113,
    'K': 128, 'L': 113, 'M': 131, 'N': 114,
    'P':  97, 'Q': 128, 'R': 156, 'S':  87,
    'T': 101, 'V':  99, 'W': 186, 'Y': 163
}
```

In [4]:
```
averageMW = sum(Daltons.values())/20.0
typicalLen = 1322/int(averageMW)
print(averageMW, typicalLen, 20**typicalLen)
```

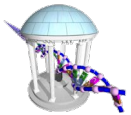
```
118.75 11.203389830508474 376657155762813.56
```

Some Issues with our Table

- We can't distinguish between Leucine (L) and Isoleucine (I). They both weight 113d
- Nor can we distinguish Lysine (K) and Glutamine (Q), which weigh 128d
- For long peptide chains >50, our errors can build up
- In reality, peptides can lose or gain one or more small molecules from their side chains and fractured peptide bonds
    - Gain Hydrogen ions (H, +1 Dalton)
    - Lose Water (H2O, -18 Daltons)
    - Lose Ammonia (NH3, -17 Daltons)
- This leads to measurements that vary around the ideal sums we assume
- Regardless of these caveats, let's keep going



The total molecular weight of our target

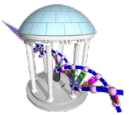


In [5]:
```
TyrocidineB1 = "VKLFPWFNQY"

# The weight of Tyrocidine B1
print(sum([Daltons[res] for res in TyrocidineB1]))
```

```
1322
```

- Generally, we will assume that the peptide's total molecular weight is known
- We will use it as a terminating condition for many of our algorithms that attempt to reconstruct the peptide sequence from a measured set of weights

What weights should we expect?

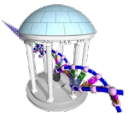


- We will make the optimistic assumption that we will fracture our given peptide chain into all of its constituent parts
- For a 10 peptide chain

10 single peptides **9, 2-peptide chains** **8, 3-peptide chains**
7, 4-peptide chains **6, 5-peptide chains** **5, 6-peptide chains**
4, 7-peptide chains **3, 8-peptide chains** **2, 9-peptide chains**
1, 10-peptide chain

- This gives an upper bound of $\binom{11}{2} = 55$ molecular weights
- In reality both the peptide chains and their weights may not be unique
- The collection of all possible sub-peptide molecular weights from a peptide is called the peptide's *Theoretical Spectrum*

Code for computing a Theoretical Spectrum

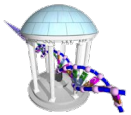


In [7]:
```
def TheoreticalSpectrum(peptide):
    # Generate every possible fragment of a peptide
    spectrum = set()
    for fragLength in range(1,len(peptide)+1):
        for start in range(0,len(peptide)-fragLength+1):
            seq = peptide[start:start+fragLength]
            spectrum.add(sum([Daltons[res] for res in seq]))
    return sorted(spectrum)

print(TyrocidineB1)
spectrum = TheoreticalSpectrum(TyrocidineB1)
print(len(spectrum))
print(spectrum)
```

```
VKLFPWFNQY
51
[97, 99, 113, 114, 128, 147, 163, 186, 227, 241, 242, 244, 260, 261, 283, 291, 333, 340, 357, 388, 389, 405, 430, 44
7, 485, 487, 543, 544, 552, 575, 577, 584, 671, 672, 690, 691, 738, 770, 804, 818, 819, 835, 917, 932, 982, 1031, 106
0, 1095, 1159, 1223, 1322]
```

- Notice there are distinct 51 weights, how many would you expect?

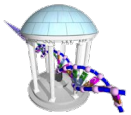
Fragments and their Spectrums



In [11]:
```
peptide = TyrocidineB1
fragList = []
for fragLength in range(1,len(peptide)+1):
    for start in range(0,len(peptide)-fragLength+1):
        seq = peptide[start:start+fragLength]
        fragList.append((sum([Daltons[res] for res in seq]), seq))

print(peptide)
print(len(fragList))
N = 0
lastWeight = 0
for weight, frag in sorted(fragList):
    print("%12s: %4d%s" % (frag, weight, "*" if (weight == lastWeight) else " "), end='')
    N += 1
    if (N % 5 == 0):
        print()
    lastWeight = weight
```

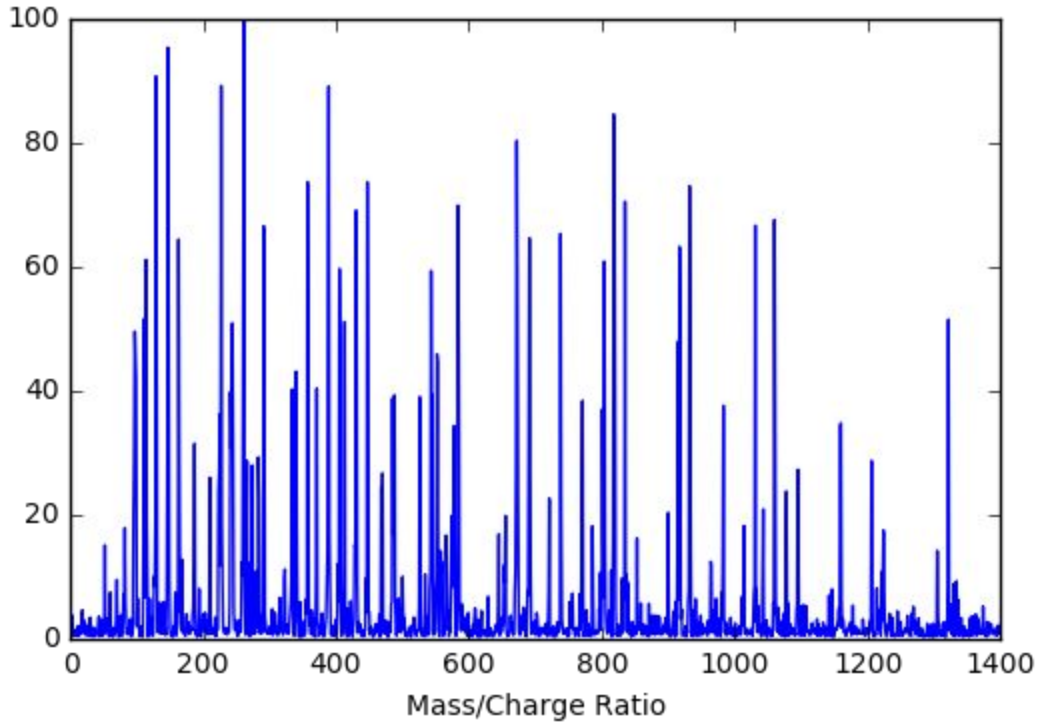
```
VKLFPWFNQY
55
           P:   97           V:   99           L:  113           N:  114           K:  128
           Q:  128*          F:  147           F:  147*          Y:  163           W:  186
          VK:  227          KL:  241          NQ:  242          FP:  244          LF:  260
          FN:  261          PW:  283          QY:  291          WF:  333         VKL:  340
         LFP:  357         KLF:  388         FNQ:  389         NQY:  405         FPW:  430
         PWF:  430*        WFN:  447        KLFP:  485        VKLF:  487        LFPW:  543
        PWFN:  544        FNQY:  552        WFNQ:  575        FPWF:  577       VKLFP:  584
       KLFPW:  671       PWFNQ:  672       LFPWF:  690       FPWFN:  691       WFNQY:  738
      VKLFPW:  770      LFPWFN:  804      KLFPWF:  818      FPWFNQ:  819      PWFNQY:  835
     VKLFPWF:  917     KLFPWFN:  932     LFPWFNQ:  932*    FPWFNQY:  982    VKLFPWFN: 1031
    KLFPWFNQ: 1060    LFPWFNQY: 1095    VKLFPWFNQ: 1159    KLFPWFNQY: 1223   VKLFPWFNQY: 1322
```

What a Mass Spectrum looks like

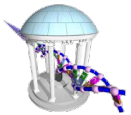


- Peaks appear at frequently occurring mass locations
- Y-axis indicates the relative abundance, sometimes called relative intensity
- The peaks roughly correspond To our mass numbers

[97, 99, 113, 114, 128, 147, 163, 186, 227, 241, 242, 244, 260, 261, 283, 291, 333, 340, 357, 388, 389, 405, 430, 447, 485, 487, 543, 544, 552, 575, 577, 584, 671, 672, 690, 691, 738, 770, 804, 818, 819, 835, 917, 932, 982, 1031, 1060, 1095, 1159, 1223, 1322]



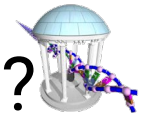
Let’s try a smaller example



In [13]:
```
peptide = 'PLAY'
spectrum = TheoreticalSpectrum(peptide)
print(len(spectrum), spectrum)

fragList = []
for fragLength in range(1,len(peptide)+1):
    for start in range(0,len(peptide)-fragLength+1):
        seq = peptide[start:start+fragLength]
        fragList.append((sum([Daltons[res] for res in seq]), seq))

print(len(fragList))
N = 0
lastWeight = 0
for weight, frag in sorted(fragList):
    print("%12s: %4d%s" % (frag, weight, "*" if (weight == lastWeight) else " "), end='')
    N += 1
    if (N % 5 == 0):
        print()
    lastWeight = weight
```

```
10 [71, 97, 113, 163, 184, 210, 234, 281, 347, 444]
10
           A:   71           P:   97           L:  113           Y:  163          LA:  184
          PL:  210          AY:  234         PLA:  281         LAY:  347        PLAY:  444
```

Can we Invert the Process of creating a Spectrum?



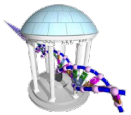
- In essence, the problem of inferring a peptide chain from the set of mass values reported by a Mass Spectrometer is the inverse of the code we just wrote

  Easy Problem: Peptide Sequence → Spectrum

  Hard Problem: Spectrum → Peptide Sequence

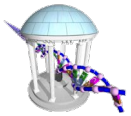
- Why is computing a spectrum from a peptide sequence easy? $O(N^2)$?
- Why is computing a peptide sequence from a spectrum hard? O(?)

How might you approach this problem?

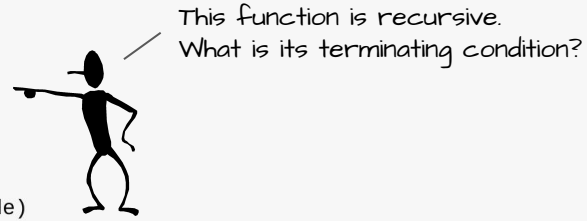


- Can you think of a Brute-Force way of solving this problem?
- Here's one:
    1. For every peptide sequence with the target peptide's molecular weight
    2. Compute the sequence's Theoretical Spectrum
    3. If it matches the one given, report this peptide as a possible solution
- Which step in this algorithm is the hard part?
- How many peptides have a molecular weight of 1322?
    1. How long is the longest peptide under 1322 daltons?
    2. How short is the shortest peptide over 1322 daltons?

A Brute-Force Attempt

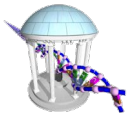


In [16]:
```
def PossiblePeptide(spectrum, prefix=''):
    """ Brute force method of generating all peptide sequences with a desired weight, the max of a given spectrum """
    global peptideList
    if (len(prefix) == 0):
        peptideList = []
    current = sum([Daltons[res] for res in prefix])
    target = max(spectrum)  # our target
    if (current == target):
        peptideList.append(prefix)
    elif (current < target):
        for residue in Daltons.keys():
            PossiblePeptide(spectrum, prefix+residue)

def TestPeptides(candidateList, target):
    filteredList = []
    for peptide in candidateList:
        candidateSpectrum = TheoreticalSpectrum(peptide)
        if (candidateSpectrum == target):
            filteredList.append(peptide)
    return filteredList

spectrum = TheoreticalSpectrum('PLAY')
%time PossiblePeptide(spectrum)
print(len(peptideList), "candidates", "PLAY" in peptideList)
%time matches = TestPeptides(peptideList, spectrum)
print(matches, "PLAY" in matches)
```




```
CPU times: user 3.84 s, sys: 13 ms, total: 3.85 s
Wall time: 3.85 s
3687 candidates True
CPU times: user 80 ms, sys: 0 ns, total: 80 ms
Wall time: 79.8 ms
['PIAY', 'PLAY', 'YAIP', 'YALP'] True
```

Impressions?

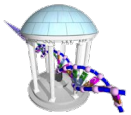


- Not so bad for a first attempt, but how will it perform for longer peptides?
- We are getting the expected answer as well as answers with the indistinguishable amino acids substituted
- We are also getting the sequence reversed? Is this a surprise?
- We could code around this, but for today we'll just include the reversed peptide chain as a possible answer

Could we do better?

- The brute force method does not make good use of the spectrum it is given
- It only ever considers the largest *mass* value from this table
- How might we make use of the other values?

Improving on Brute Force

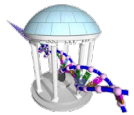


- We could extend our prefix using *only* residues that appear in our spectrum
- The weight of every new prefix that we consider should also be in our spectrum

Actual fragments: P, L, A, Y, PL, LA, AY, PLA, LAY, PLAY

```
A              I              L              P              Y

AI   = LA      IA   = LA      LA   = LA      PI   = PL      YA   = AY
AIP  = PLA     IAP  = PLA     LAP  = PLA     PIA  = PLA     YAI  = LAY
AIPY = PLAY    IAPY = PLAY    LAPY = PLAY    PIAY = PLAY    YAIP = PLAY
AIY  = LAY     IAY  = LAY     LAY  = LAY                    YAL  = LAY
AIYP = PLAY    IAYP = PLAY    LAYP = PLAY                   YALP = PLAY

AL   = LA      IP   = PL      LP   = PL      PL   = PL
ALP  = PLA     IPA  = PLA     LPA  = PLA     PLA  = PLA
ALPY = PLAY    IPAY = PLAY    LPAY = PLAY    PLAY = PLAY
ALY  = LAY
ALYP = PLAY

AY   = AY
AYI  = LAY
AYIP = PLAY
AYL  = LAY
AYLP = PLAY
```

Only a small change

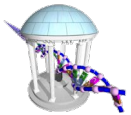


In [19]:
```
def ImprovedPossiblePeptide(spectrum, prefix=''):
    global peptideList
    if (len(prefix) == 0):
        peptideList = []
    current = sum([Daltons[res] for res in prefix])
    target = max(spectrum)
    if (current == target):
        peptideList.append(prefix)
    elif (current < target):
        for residue in Daltons.keys():
            # make sure that this residue appears in our spectrum
            if (Daltons[residue] not in spectrum):
                continue
            # make sure that adding this residue to the sequence we have so far appears in our spectrum
            extend = prefix + residue
            if (sum([Daltons[res] for res in extend]) not in spectrum):
                continue
            ImprovedPossiblePeptide(spectrum, extend)

spectrum = TheoreticalSpectrum('PLAY')
%time ImprovedPossiblePeptide(spectrum)
print(len(peptideList), "PLAY" in peptideList)
print(peptideList)
%time matches = TestPeptides(peptideList, spectrum)
print(matches, "PLAY" in matches)
```

This prunes the search space (a.k.a. Branch-and-bound)

```
CPU times: user 1 ms, sys: 0 ns, total: 1 ms
Wall time: 708 µs
16 True
['AIPY', 'AIYP', 'ALPY', 'ALYP', 'AYIP', 'AYLP', 'IAPY', 'IAYP', 'IPAY', 'LAPY', 'LAYP', 'LPAY', 'PIAY', 'PLAY', 'YAI
P', 'YALP']
CPU times: user 1 ms, sys: 0 ns, total: 1 ms
Wall time: 537 µs
['PIAY', 'PLAY', 'YAIP', 'YALP'] True
```

Impact of a small change



- Provides a HUGE performace difference
- Yet another example of Branch-and-Bound
- We improved both the enumeration and verification phases, but the difference was much more significant in the enumeration step

In [17]:
```
print(', '.join([peptide for peptide in peptideList]))
print(TheoreticalSpectrum('PLAY'))
print(TheoreticalSpectrum('LAPY'))
```

```
AIPY, AIYP, ALPY, ALYP, AYIP, AYLP, IAPY, IAYP, IPAY, LAPY, LAYP, LPAY, PIAY, PLAY, YAIP, YALP
[71, 97, 113, 163, 184, 210, 234, 281, 347, 444]
[71, 97, 113, 163, 168, 184, 260, 281, 331, 444]
```

In [18]:
```
print(sum([Daltons[res] for res in 'AP']))  # Suffix of 'LAP' prefix
print(sum([Daltons[res] for res in 'APY'])) # Suffix of 'LAPY'
print(sum([Daltons[res] for res in 'PY']))  # Suffix of 'LAPY'
```

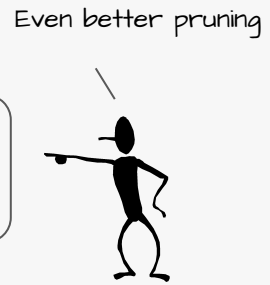
```
168
331
260
```

- There are still differences in the spectrums, yet every prefix was in the spectrum when we added it. What are we missing?
- Suffixes!

We can do Even Better

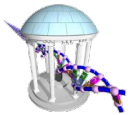
All *suffixes* of each *prefix* that we consider should also be in our spectrum

```
In [21]: def UltimatePossiblePeptide(spectrum, prefix=''):
             global peptideList
             if (len(prefix) == 0):
                 peptideList = []
             current = sum([Daltons[res] for res in prefix])
             target = max(spectrum)
             if (current == target):
                 peptideList.append(prefix)
             elif (current < target):
                 for residue in Daltons.keys():
                     extend = prefix + residue
                     # test every new suffix created by adding this new reside
                     # Note: this includes the residue itself as the length 1 suffix
                     suffix = [extend[i:] for i in range(len(extend))]
                     for fragment in suffix:
                         if (sum([Daltons[res] for res in fragment]) not in spectrum):
                             break
                     else:
                         UltimatePossiblePeptide(spectrum, extend)

         spectrum = TheoreticalSpectrum('PLAY')
         %time UltimatePossiblePeptide(spectrum)
         print(len(peptideList), peptideList, "PLAY" in peptideList)
         %time matches = TestPeptides(peptideList, spectrum)
         print(matches, "PLAY" in matches)

         CPU times: user 1.1 ms, sys: 4 µs, total: 1.11 ms
         Wall time: 1.12 ms
         4 ['PIAY', 'PLAY', 'YAIP', 'YALP'] True
         CPU times: user 113 µs, sys: 0 ns, total: 113 µs
         Wall time: 123 µs
         ['PIAY', 'PLAY', 'YAIP', 'YALP'] True
```



- A little slower, but our list is pruned significantly
- All of theses have identical spectrums

Now let's return to our *Real* peptide



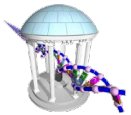
In [23]:
```
spectrum = TheoreticalSpectrum(TyrocidineB1)
%time UltimatePossiblePeptide(spectrum)
print(len(peptideList))
print(TyrocidineB1 in peptideList)
%time matches = TestPeptides(peptideList, spectrum)
print(len(matches))
print(TyrocidineB1 in matches)
```

```
CPU times: user 31.4 ms, sys: 2.2 ms, total: 33.6 ms
Wall time: 31.5 ms
['VKIFPWFNKY', 'VKIFPWFNQY', 'VKLFPWFNKY', 'VKLFPWFNQY', 'VQIFPWFNKY', 'VQIFPWFNQY', 'VQLFPWFNKY', 'VQLFPWFNQY', 'YKN
FWPFIKV', 'YKNFWPFIQV', 'YKNFWPFLKV', 'YKNFWPFLQV', 'YQNFWPFIKV', 'YQNFWPFIQV', 'YQNFWPFLKV', 'YQNFWPFLQV']
16
True
CPU times: user 1.11 ms, sys: 6 µs, total: 1.12 ms
Wall time: 1.13 ms
16
True
```

In [24]:
```
print(TyrocidineB1)
for i, peptide in enumerate(peptideList):
    print(peptide, end=',')
    if (i % 4 == 3):
        print()
```

```
VKLFPWFNQY
VKIFPWFNKY,VKIFPWFNQY,VKLFPWFNKY,VKLFPWFNQY,
VQIFPWFNKY,VQIFPWFNQY,VQLFPWFNKY,VQLFPWFNQY,
YKNFWPFIKV,YKNFWPFIQV,YKNFWPFLKV,YKNFWPFLQV,
YQNFWPFIKV,YQNFWPFIQV,YQNFWPFLKV,YQNFWPFLQV,
```

Great, but our assumptions are a little Naïve



- In reality, Mass Spectometers don't report the Theoretical Spectrum of a peptide
- Instead they report a measured or *Experimental Spectrum*
- This spectrum might *miss* some fragments
- It might also report *false* fragments
  - From Contaminants
  - New peptides formed by unintended reactions between fragments
- The result is that some of the masses that appear may be misleading, and some that we want might be missing
- We need to develop algorithms for reporting candidate protein sequences that are robust to noise

More Next Time