From: **Jason Li** jasonmli02@gmail.com
Subject: Solutions to contest #4
Date: February 5, 2016 at 6:20 PM
To: 15-295, Carnegie Mellon 15-295@googlegroups.com

JL

Here are main solution ideas to this week's contest:

--- Div2 (this week was a bit hard, sorry about that) ---
B: This problem is basically solving a system of linear equations. Let the unknown sum be X. Then, since each column sums to X, you can rewrite the unknown element in the column as X - (sum of rest). You also know that the unknown diagonal entries sum to X. That gives you enough information to solve for X.
C: This is a dynamic programming problem. Your states are just the sets of used-up sticks (so there are 2^n of them). For each state, try all possible triangle formations. This idea of using sets as dynamic programming states may seem contrived, but it's a standard technique, and once you've seen it once you should be able to apply it in the future. (The n=15 constraint was also a hint towards an exponential-time solution)
There are probably more efficient greedy solutions, but this one runs in time and is sure to work. Note that here, sets are represented as numbers with N binary digits (possibly with leading zeros), where 0/1 at position i determines whether or not element i is included in the set.

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

int n;
ll a[20];
int dp[1<<20];

int main() {
    cin >> n;
    for(int i = 0; i < n; i ++) cin >> a[i];
    sort(a, a + n);

    for(int i = 0; i < (1<<n); i ++) {
        vector<int> v;
        for(int j = 0; j < n; j ++) if(i & (1 << j)) v.push_back(j);
        int m = v.size();
        for(int x = 0; x < m; x ++) {
            for(int y = x + 1; y < m; y ++) {
                for(int z = y + 1; z < m; z ++) {
                    int xx = v[x], yy = v[y], zz = v[z];
                    //cout << xx << " " << yy << " " << zz << endl;
                    int id = i;
                    if(id & (1 << xx)) id ^= (1 << xx);
                    if(id & (1 << yy)) id ^= (1 << yy);
                    if(id & (1 << zz)) id ^= (1 << zz);
                    int tmp = dp[id];
                    if(a[xx] + a[yy] > a[zz]) tmp ++;
                    if(tmp > dp[i]) dp[i] = tmp;
                }
            }
        }
    }

    //for(int i = 0; i < (1 << n); i ++) cout << i << " = " << dp[i] << endl;
    cout << dp[(1<<n)-1] << endl;
    return 0;
}
```

D: Oops, this problem was quite advanced. There were some accepted brute-force solutions, but the intended solution was probably bipartite matching. Form a bipartite graph: left side are the cubes, right side are the letters in the desired word. Form an edge (cube -> letter) if that cube contains that letter. Then match all letters with cubes.

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 256;
string s;
int n, link[MAXN];
bool used[MAXN], e[MAXN][MAXN];

bool match(int u) {
    for(int i = 0; i < n; i ++) {
        if(used[i] || !e[u][i]) continue;
        used[i] = true;
        if(link[i] == -1 || match(link[i])) {
            link[i] = u;
            return true;
        }
```

```
    }
    return false;
}

int main() {
    cin >> s >> n;
    char op[5];
    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < 6; j ++) {
            cin >> op;
            for(int k = 0; k < s.length(); k ++) {
                if(s[k] == op[0]) e[k][i] = true;
            }
        }
    }
    memset(link, -1, sizeof(link));
    int ans = 0;
    for(int i = 0; i < 26; i ++) {
        memset(used, 0, sizeof(used));
        if(match(i)) ans ++;
    }
    if(ans == s.length()) cout << "YES" << endl; else cout << "NO" << endl;
    return 0;
}
```

E: The trick here is matrix multiplication, which can be done in log(M) time using repeated squaring. Remember that Fibonacci numbers can be represented using the matrix:
1 1
1 0
That was covered in 15-251 when I took it. This problem is just an extension of this idea. Again, this trick may seem unintuitive, but when you've seen it once, you're ready for it the next time.

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

const ll mod = 1000000009;
int n, c, len;
ll m;
ll f[40];
int k[40];

struct Matrix {
    ll d[25][25];
    Matrix() {
        for(int i = 0; i < n; i ++) for(int j = 0; j < n; j ++) d[i][j] = 0;
    }
    void print() {
        printf("------\n");
        for(int i = 0; i < len; i ++) {
            for(int j = 0; j < len; j ++) {
                printf("%d ", d[i][j]);
            }
            printf("\n");
        }
        printf("------\n\n");
    }
    Matrix operator * (const Matrix &_p) const {
        Matrix ans;
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                ans.d[i][j] = 0;
                for(int k = 0; k < n; k ++) {
                    ans.d[i][j] += d[i][k] * _p.d[k][j];
                    ans.d[i][j] %= mod;
                }
            }
        }
        return ans;
    }
};

Matrix quickpow(Matrix mx, ll k) {
    Matrix ans;
```

```
    Matrix ans;
    for(int i = 0; i < len; i ++) ans.d[i][i] = 1;
    while(k > 0) {
        if(k & 1) ans = ans * mx;
        k = k >> 1;
        mx = mx * mx;
    }
    return ans;
}

int main() {
    cin >> n >> m >> c;
    for(int i = 0; i < n; i ++) cin >> f[i];
    for(int i = 0; i < c; i ++) cin >> k[i];
    if(m <= n) {
        cout << f[m-1] << endl;
        return 0;
    }

    Matrix mx;
    for(int i = 0; i < c; i ++) {
        mx.d[0][k[i] - 1] = 1;
    }

    len = n;
    for(int i = 1; i < len; i ++) {
        mx.d[i][i-1] = 1;
    }

    //mx.print();
    mx = quickpow(mx, m - n);

    ll ans = 0;
    for(int i = 0; i < n; i ++) {
        ans = (ans + mx.d[0][i] * f[n-1-i]) % mod;
    }
    cout << ans << endl;
    return 0;
}
```

--- Div1 ---
A: Think of parenthesis sequences as walks up and down a rooted tree. Then, for each subtree, it suffices to count the number of disjoint subtrees within that subtree.

```
#include <bits/stdc++.h>
using namespace std;
char A[100005];
vector<int> ch[100005];
int N;
long long ans;
int go(int x) {
        long long sum = 0, toadd = 0;
        for (auto y : ch[x]) {
                int singles = go(y);
                toadd += sum * singles;
                sum += singles;
        }
        ++sum;
        if (x != N) ans += toadd * sum * 2;
        return sum;
}
int main() {
        scanf("%s", A);
        N = strlen(A);
        stack<int> S;
        S.push(N);
        for (int i = 0; i < N; ++i) {
                if (A[i] == '(') {
                        // Push new child
                        if (!S.empty()) {
                                int tp = S.top();
                                ch[tp].push_back(i);
                        }
                        S.push(i);
                } else {
                        S.pop();
```

```
                }
        }
        go(N);
        printf("%I64d\n", ans);
}
```

B: Dynamic programming, where your state is (your position, guard position)

```
#include <bits/stdc++.h>
using namespace std;
int D[26][26][26][26];
char G[35][35];
int DR[4] = {0, -1, 0, 1};
int DC[4] = {-1, 0, 1, 0};
int N, M, sr, sc, tr, tc, gr, gc;
bool bad(pair<int, int> pos) {
        if (pos.first < 0 || pos.first >= N || pos.second < 0 || pos.second >= M) return true;
        return G[pos.first][pos.second] == 'X';
}
queue<pair<pair<int, int>, pair<int, int> > > Q;
bool dopush(int d, pair<int, int> newme, int r, int c) {
        if (bad(make_pair(r, c))) return false;
        if (D[newme.first][newme.second][r][c] != -1) return true;
        Q.push(make_pair(newme, make_pair(r, c)));
        D[newme.first][newme.second][r][c] = d + 1;
        return true;
}
int main() {
        scanf("%d%d", &N, &M);
        for (int i = 0; i < N; ++i) {
                scanf("%s", G[i]);
                for (int j = 0; j < M; ++j)
                        if (G[i][j] == 'A') {
                                sr = i;
                                sc = j;
                        } else if (G[i][j] == 'P') {
                                tr = i;
                                tc = j;
                        } else if (G[i][j] == 'G') {
                                gr = i;
                                gc = j;
                        }
        }
        memset(D, -1, sizeof(D));
        dopush(-1, make_pair(sr, sc), gr, gc);
        while (!Q.empty()) {
                auto x = Q.front();
                Q.pop();
                auto me = x.first;
                auto you = x.second;
                int dist = D[me.first][me.second][you.first][you.second];
                if (me == you) continue;
                if (me.first == tr && me.second == tc) {
                        printf("%d\n", dist);
                        return 0;
                }
                for (int d = 0; d < 4; ++d) {
                        auto newme = make_pair(me.first + DR[d], me.second + DC[d]);
                        if (bad(newme)) continue;
                        if (newme == you) continue;
                        if (newme.second < you.second) {
                                if (dopush(dist, newme, you.first, you.second - 1)) continue;
                        }
                        if (newme.second > you.second) {
                                if (dopush(dist, newme, you.first, you.second + 1)) continue;
                        }
                        if (newme.first < you.first) {
                                if (dopush(dist, newme, you.first - 1, you.second)) continue;
                        }
                        if (newme.first > you.first) {
                                if (dopush(dist, newme, you.first + 1, you.second)) continue;
                        }
                        dopush(dist, newme, you.first, you.second);
                }
```

```
        }
        printf("-1\n");
}
```

C: Always greedily pick the card that gets you farthest

```
#include <bits/stdc++.h>
using namespace std;
pair<int, pair<int, int> > A[100005];
int main() {
        int N;
        scanf("%d", &N);
        for (int i = 0; i < N; ++i) {
                A[i].second.second = i + 1;
                scanf("%d%d", &A[i].first, &A[i].second.first);
        }
        int need = A[N - 1].first;
        if (need <= 1) {
                printf("1\n%d\n", N);
                return 0;
        }
        sort(A, A + N);
        int res = 1, best = 1, besttake;
        vector<int> ans;
        for (int i = 0; i <= N; ++i) {
                if (i == N || A[i].first > res) {
                        if (best == res) {
                                printf("No such luck\n");
                                return 0;
                        }
                        // Update res with best
                        res = max(res, best);
                        ans.push_back(besttake);
                        if (res >= need) {
                                if (besttake != N) ans.push_back(N);
                                printf("%d\n", ans.size());
                                for (auto x : ans) printf("%d ", x);
                                printf("\n");
                                return 0;
                        }
                }
                if (best < A[i].second.first || (best == A[i].second.first &&
                        A[i].second.second == N)) {
                        best = max(best, A[i].second.first);
                        besttake = A[i].second.second;
                }
        }
}
```

D: The question reduces to finding the most common string (with some edge cases). Naive string comparison is too slow, so use the fact that strings have at most 15 ones. So they can be represented efficiently as vectors of size <=15 (the set of 1's). allowing easy comparisons. (Alternatively you can just hash the strings; the second solution uses two hashes (10^{-18} chance of failure per comparison).

```
#include <bits/stdc++.h>
using namespace std;
char S[5005];
map<vector<int>, int> num;
int main() {
        int N, M;
        scanf("%d%d", &N, &M);
        while (N--) {
                scanf("%s", S);
                vector<int> v;
                for (int i = 0; i < M; ++i) {
                        if (S[i] == '1') {
                                v.push_back(i);
                                if (v.size() > 15) break;
                        }
                }
                if (v.size() < 8 || v.size() > 15) continue;
                ++num[v];
        }
        vector<int> best;
        for (int i = 0; i < 8; ++i) best.push_back(i);
```

```
        int bestv = 0;
        for (auto &x : num) {
                if (x.second > bestv) {
                        bestv = x.second;
                        best = x.first;
                }
        }
        for (int i = 0, bi = 0; i < M; ++i)
                if (bi < best.size() && best[bi] == i) {
                        printf("1");
                        ++bi;
                } else {
                        printf("0");
                }
        printf("\n");
}
```

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
ll power = 37, mod = 1000000007;

int N, M;
char A[5050][5050];

int ones(char* V)
{
        int cnt = 0;
        for(int i = 0; i < M; i++) cnt += (V[i] == '1');
        return cnt;
}

ll get(char* V, ll* H)
{
        ll res = 0;
        for(int i = 0; i < M; i++)
        {
                res += H[V[i] - '0'];
                res = (res * power) % mod;
        }
        return res;
}
map<ll, int> H;
int main()
{
        scanf("%d %d", &N, &M);
        for(int i = 0; i < N; i++) scanf("%s", A[i]);
        int best = -1, besti = -1;
        ll H1[2], H2[2];
        for(int i = 0; i < 2; i++)
        {
                H1[i] = rand() % mod;
                H2[i] = rand() % mod;
        }
        for(int i = 0; i < N; i++)
        {
                if(!(8 <= ones(A[i]) && ones(A[i]) <= 15)) continue;
                ll hash = ((get(A[i], H1)) << 32LL) + get(A[i], H2);
                H[hash]++;
                if(H[hash] > best) best = H[hash], besti = i;
        }
        if(best == -1)
        {
                printf("11111111");
                for(int i = 8; i < M; i++) printf("0");
        }
        else printf("%s\n", A[besti]);
}
```

E: This is a tricky one. Here's a hint: the answer is the matrix of all 0's if and only if, for each i, the row sum of i equals the column sum of i. In other words, if you view the matrix as a flow graph (where a_{ij} equals the amount of flow from i to j), then the flow is "conservative" (amount of flow into i equals amount of flow out of i, for each i)

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

int N;
ll A[1005][1005];
ll delta[1005];
ll ans[1005][1005];
vector<pair<ll, ll> > pos, neg;

int main()
{
	ios::sync_with_stdio(false);
	cin >> N;
	for(int i = 1; i <= N; i++)
		for(int j = 1; j <= N; j++)
		{
			cin >> A[i][j];
			delta[i] += A[i][j];
			delta[j] -= A[i][j];
		}
	for(int i = 1; i <= N; i++)
	{
		if(delta[i] > 0) pos.push_back(make_pair(i, delta[i]));
		if(delta[i] < 0) neg.push_back(make_pair(i, -delta[i]));
	}
	while(!pos.empty() && !neg.empty())
	{
		ll d = min(pos.back().second, neg.back().second);
		pos.back().second -= d, neg.back().second -= d;
		ans[pos.back().first][neg.back().first] += d;
		if(pos.back().second == 0) pos.pop_back();
		if(neg.back().second == 0) neg.pop_back();
	}
	for(int i = 1; i <= N; i++)
	{
		for(int j = 1; j <= N; j++)
			cout << ans[i][j] << ' ';
		cout << '\n';
	}
}
```