Library
Created by clax members

Parser

◆ 四則演算の例

```
typedef pair<int, const char*> parsed;
parsed fact(const char *p);
parsed term(const char *p);
parsed expr(const char *p);

parsed fact(const char *p) {
  if(isdigit(*p)) {
    int t = *(p++)-'0';
    while(isdigit(*p)) t = t*10 + *(p++)-'0';
    return parsed(t,p);
  } else if (*p == '(') {
    parsed r = expr(p+1);
    if(*r.second!=')') {
      return parsed(INT_MAX, "ERROR");
    }
    return parsed(r.first, r.second+1);
  } else {
    return parsed(INT_MAX, "ERROR");
  }
}

parsed term(const char *p) {
  parsed r = fact(p);
  while(*r.second == '*' || *r.second == '/') {
    char op = *r.second;
    int tmp = r.first;
    r = fact(r.second+1);
    if(op=='*') r.first = tmp * r.first;
    else if(op == '/') r.first =  tmp / r.first;
  }
  return r;
}

parsed expr(const char *p) {
  parsed r = term(p);
  while(*r.second=='+' || *r.second=='-') {
    char op = *r.second;
    int tmp = r.first;
    r = term(r.second+1);
    if(op=='+') r.first = tmp + r.first;
    else if(op=='-') r.first = tmp - r.first;
  } else {
    return parsed(INT_MAX, "ERROR");
  }
  return r;
}

int main() {
  string str;
  while(cin >> str) {
    parsed ans = expr(str.c_str());
    if(ans.second!="ERROR") {
      cout << ans.first << endl;
    }
  }
  return 0;
}
```

Dynamic Programming

◆ LIS: O(n^2)

```
int lis(vector<int> a) {
  int size = a.size();
  int table[size], max = 0;
  for(int i = 0; i < size; i++) table[i] = 1;
  for (int i = 0; i < size - 1; i++) {
    for(int j = i + 1; j < size; j++) {
      if((a[j] > a[i]) && (table[i] + 1 > table[j])){
        table[j] = table[i] + 1;
        if(table[j] > max) max = table[j];
      }
    }
  }
  if(max==0 && size!=0) max = 1;
  return max;
}
```

◆ LCS: 最長共通部分を求め、そのサイズを返します。

```
int lcs(char *a, char *b) {
  int m = strlen(a);
  int n = strlen(b);
  for(int i=0;i<=m;i++) table[i][0] = 0;
  for(int i=0;i<=n;i++) table[0][i] = 0;
  for(int i = 1; i <= m; i++) {
    for(int j = 1; j <= n; j++) {
        int val1=max(table[i-1][j],table[i][j-1]), val2=table[i-1][j-1];
        if(a[i-1] == b[i-1]) val2++;
        table[i][j] = max(val1,val2);
    }
  }
  return table[m][n];
}
```

◆ LIS: O(n log k)　k:増加列の長さ

```
class Node{
public:
  int val, id, prev;
  Node(){}
  Node(int v, int i, int p): val(v), id(i), prev(p) {}
  bool operator< (const Node &n) const {
    return val<n.val;
  }
};

void lis(){
  vector<Node> top;
  top.push_back( Node(nList[0],0,-1) );
  nodes[0] = Node(nList[0],0,-1);

  for(int i=1;i<nNum;i++){
    if(top.back().val<nList[i]){
      int prevId = top.back().id;
      top.push_back( Node(nList[i],i,prevId) );
      nodes[i] = Node(nList[i],i,prevId);
    }
    if(top.front().val>=nList[i]){
      top[0] = Node(nList[i],i,-1);
      nodes[i] = Node(nList[i],i,-1);
    }
    else{
      Node toCmp = Node(nList[i],0,0);
      vector<Node>::iterator pos =
lower_bound(top.begin(),top.end(),toCmp);
      if(pos->val==toCmp.val) continue;
      else {
            int index = pos-top.begin()-1;
            top[index+1] = Node(nList[i],i,top[index].id);
            nodes[i] = Node(nList[i],i,top[index].id);
      }
    }
  }

  vector<int> path;
  for(int i=top.back().id;i!=-1;i=nodes[i].prev)
    path.push_back(nodes[i].val);

  for(int i=path.size()-1;i>=0;i--)
    cout << path[i] << endl;

}
```

◆ Edit Distance: 文字列 A,B に対し、A を最小の操作で B に変換する方法をもとめる。

```
void editDistance(){
  int cost[MAX_SIZE][MAX_SIZE];
  int pi[MAX_SIZE][MAX_SIZE];

  for(int i=0;i<=str1.size();i++){
    cost[i][0]=i;
    pi[i][0]=UP;
  }
  for(int j=0;j<=str2.size();j++){
    cost[0][j]=j;
    pi[0][j]=LEFT;
  }
  pi[0][0]=END;

  for(int i=1;i<=str1.size();i++){
    for(int j=1;j<=str2.size();j++){
      int lCost=cost[i][j-1]+1;
      int uCost=cost[i-1][j]+1;
      int luCost=cost[i-1][j-1]+1;
      if(str1[i-1]==str2[j-1]) luCost--;

      if(uCost<=lCost && uCost<=luCost){
            cost[i][j] = uCost;
            pi[i][j] = UP;
      }
      else if(lCost<=luCost && lCost<=uCost){
            cost[i][j] = lCost;
            pi[i][j] = LEFT;
      }
      else if(luCost<=lCost && luCost<=uCost){
            cost[i][j] = luCost;
            pi[i][j] = LEFT_UP;
      }
    }
  }

  vector<Ope> opeList;
  int r=str1.size(), c=str2.size();

  while(pi[r][c]!=END){
    if(pi[r][c]==LEFT_UP){
      if(str1[r-1]==str2[c-1]){
         opeList.push_back(Ope(COPY,c,' '));
      }
      else
         opeList.push_back(Ope(REPLACE,c,str2[c-1]));
      r--, c--;
    }
    else if(pi[r][c]==UP){
      opeList.push_back(Ope(DELETE,c+1,' '));
      r--;
    }
    else if(pi[r][c]==LEFT){
      opeList.push_back(Ope(INSERT,c,str2[c-1]));
      c--;
    }
  }

}
```

◆ Knapsack:　O(MN)　N: 荷物の数　M: リュックに入る重さ

```
void knapsackRec(int curr, int maxVal[MAX_SIZE][MAX_WEIGHT]){
  const int bValue=bList[curr].value, bWeight=bList[curr].weight;

  if(curr==nBurden) return;

  for(int i=0;i<MAX_WEIGHT;i++){
    if(maxVal[curr-1][i] == -INF) continue;
    maxVal[curr][i] = max(maxVal[curr][i],maxVal[curr-1][i]);
    if(i+bWeight<MAX_WEIGHT)
      maxVal[curr][i+bWeight] = max(maxVal[curr][i+bWeight],maxVal[curr-1][i]+bValue);
  }

  knapsackRec(curr+1,maxVal);
}

void knapsack(){
  int maxVal[MAX_SIZE][MAX_WEIGHT];
  int nQuery;

  for(int i=0;i<MAX_SIZE;i++)
    for(int j=0;j<MAX_WEIGHT;j++)
      maxVal[i][j] = -INF;
  maxVal[0][0] = 0;
  maxVal[0][bList[0].weight] = bList[0].value;

  knapsackRec(1,maxVal);

}
```

◆ Matrix Chain Multiplication: 行列数個の掛け算をする際の計算量を最小に抑える。

```
void output(int left, int right, int best[MAX_SIZE][MAX_SIZE]){
  if(left==right){
    printf("M%d",left);
  }
  else{
    printf("(");
    output(left,best[left][right]-1,best);
    output(best[left][right],right,best);
    printf(")");
  }
}

void matrixMul(Matrix mList[MAX_SIZE]){
  int cost[MAX_SIZE][MAX_SIZE], best[MAX_SIZE][MAX_SIZE];

  for(int i=0;i<nMatrix;i++)
    for(int j=0;j<nMatrix;j++)
      cost[i][j] = INT_MAX;
  for(int i=0;i<nMatrix;i++)
    cost[i][i] = 0;

  for(int i=1;i<nMatrix;i++){
    for(int j=0;j<nMatrix-i;j++){
      for(int k=j+1;k<j+1+i;k++){
        if(cost[j][j+i]>mList[j].row*mList[k-1].column*mList[j+i].column+cost[j][k-1]+cost[k][j+i]){
          cost[j][j+i]=mList[j].row*mList[k-1].column*mList[j+i].column+cost[j][k-1]+cost[k][j+i];
          best[j][j+i] = k;
        }
      }
    }
  }

  output(0,nMatrix-1,best);
  printf("\n");
}
```

グラフ理論

◆ Dijkstra: スタート地点からの最短経路とそのコストを計算。　$O(|V|^2)$

```
void dijkstra(int start){

  int nVisit=0;
  bool visited[MAX_SIZE];
  int cost[MAX_SIZE], pi[MAX_SIZE];

  for(int i=0;i<nNode;i++){
    cost[i] = INF;
    visited[i] = false;
  }
  pi[start] = -1;
  cost[start] = 0;

  while(nVisit!=nNode){
    int minPos, minVal=INF;

    for(int i=0;i<nNode;i++)
      if(minVal>cost[i] && !visited[i]){
        minVal = cost[i];
        minPos = i;
      }

    //グラフは非連結である。
    if(minVal==INF) break;

    visited[minPos] = true;
    nVisit++;

    for(int i=0;i<nNode;i++)
      if(cost[i]>cost[minPos]+adjList[minPos][i]){
        cost[i]=cost[minPos]+adjList[minPos][i];
        pi[i] = minPos;
      }
  }

}
```

◆ Prim: 最小全域木を生成。　$O(|V|^2)$

```
void prim(){
  int pi[MAX_SIZE], nVisited=0;
  double connect[MAX_SIZE];
  bool visited[MAX_SIZE];

  for(int i=0;i<nNode;i++){
    visited[i] = false;
    connect[i] = INF;
  }
  connect[0] = 0;
  pi[0] = -1;

  while(nVisited!=nNode){
    int minPos;
    double minVal=INF;

    for(int i=0;i<nNode;i++)
      if(minVal>connect[i] && !visited[i]){
        minVal = connect[i];
        minPos = i;
      }

    //グラフは非連結である。
    if(minVal==INF) break;

    visited[minPos] = true;
    nVisited++;

    for(int i=0;i<nNode;i++){
      if(visited[i]) continue;
      if(connect[i]>adjList[minPos][i]){
        connect[i] = adjList[minPos][i];
        pi[i] = minPos;
      }
    }
  }

}
```

◆ Floyd Warshall: 全ての二点間の最短距離を求める。 $O(|V|^3)$

```
void floydWarshall(int cost[MAX_SIZE][MAX_SIZE]){
  for(int i=0;i<nNode;i++)
    for(int j=0;j<nNode;j++)
      cost[i][j] = adjList[i][j];

  for(int i=0;i<nNode;i++)
    cost[i][i] = 0;

  for(int k=0;k<nNode;k++)
    for(int i=0;i<nNode;i++)
      for(int j=0;j<nNode;j++)
        cost[i][j] = min(cost[i][j],cost[i][k]+cost[k][j]);
        //二点を通る辺の重みの最大値を最小化する。
        //cost[i][j] = min(cost[i][j],max(cost[i][k],cost[k][j]));
}
```

◆ Topological Sort: 優先順位を元に、優先順位を崩さないように順番に並べる。
$O(|V|+|E|)$

```
void topSortRec(int curr, bool visited[MAX_SIZE], vector<int> &order){
  visited[curr] = true;

  for(int i=0;i<nNode;i++){
    if(visited[i]) continue;
    if(!adjList[curr][i]) continue;
    topSortRec(i,visited,order);
  }

  order.push_back(curr);
}

void topSort(){
  bool visited[MAX_SIZE];
  vector<int> order;

  for(int i=0;i<nNode;i++)
    visited[i] = false;

  for(int i=0;i<nNode;i++){
    if(visited[i]) continue;
    topSortRec(i,visited,order);
  }
}
```

◆ Bellman Ford:

負の重みのあるグラフにおいて、スタート地点からの最短経路と
そのコストを計算。サイクルの有無を調べる。　Bellman Fordを二回し、
一回目よりも到達コストが低ければ、そこはサイクルに含まれる。　$O(|V|^3)$

```
void bellman(int source, int cost[MAX_SIZE], bool isCycle[MAX_SIZE]){

  int cost2[MAX_SIZE];

  for(int i=0;i<nNode;i++){
    cost[i] = INF;
    cost2[i] = INF;
  }

  cost2[source] = 0;

  for(int i=0;i<nNode*2;i++){
    for(int j=0;j<nNode;j++){
      for(int k=0;k<nNode;k++){
        if(cost2[j]==INF) continue;

        if(cost2[k]>cost2[j]+adjList[j][k])
          cost2[k] = cost2[j]+adjList[j][k];
      }
    }
  }

  for(int j=0;j<nNode;j++)
    cost[j] = cost2[j];

  for(int i=0;i<nNode*2;i++){
    for(int j=0;j<nNode;j++){
      for(int k=0;k<nNode;k++){
        if(cost2[j]==INF) continue;

        if(cost2[k]>cost2[j]+adjList[j][k])
          cost2[k] = cost2[j]+adjList[j][k];
      }
    }
  }

  for(int i=0;i<nNode;i++)
    isCycle[i] = false;

  for(int i=0;i<nNode;i++){
    if(cost2[i]<cost[i])
      isCycle[i]=true;
  }
}
```

◆ Maximum Flow:

始点から終点への最大流を求める。エドモンズ・カープを使用している。 $O(|E|^2|V|)$

```
void addFlow( pair<int,vector<int> > &flowTask )
{

 vector<int> init;
 int visited[MAX_SIZE];
 queue< pair<int,vector<int> > > Q;

 init.push_back(SOURCE);
 for(int i=0;i<MAX_SIZE;i++)
  visited[i] = -INF;

 Q.push( make_pair(INF,init) );

 while(!Q.empty()){
  pair<int,vector<int> > curr = Q.front();
  Q.pop();

  const int currId = curr.second.back();

  if(currId==SINK){
   flowTask = curr;
   break;
  }

  if(visited[currId]>=curr.first) continue;
  visited[currId] = curr.first;

  for(int i=0;i<MAX_SIZE;i++){
   if(capacity[currId][i]<=0) continue;

   pair<int,vector<int> > next = curr;

   next.first = min(next.first,capacity[currId][i]);
   next.second.push_back(i);

   Q.push(next);
  }

 }

}
```

```
void maximumFlow(){
 int total=0;

 while(true){
  pair<int,vector<int> > flowTask;
  flowTask.first = 0;
  addFlow(flowTask);

  if(flowTask.first==0) break;

  for(int i=1;i<flowTask.second.size();i++){
   int s = flowTask.second[i-1];
   int t = flowTask.second[i];

   capacity[s][t]-=flowTask.first;
   capacity[t][s]+=flowTask.first;

   if(t==SINK)
    total+=flowTask.first;
  }
 }

 cout << total << endl;
}
```

◆ Euler Path:

オイラー経路を求める。グラフの中の、度数が奇数である頂点は 2 個以下でなくてはならない。もし度数が奇数ならばそこを始点とする。度数が奇数である頂点が無ければ始点はどこでもよい。 $O(|V|+|E|)$

```
void findEuler(int curr, int cursor, stack<int> &S, pair<int,int> order[MAX_SIZE]){

  for(int i=0;i<nNode;i++){
    if(adjList[curr][i]==0) continue;

    adjList[curr][i]--;
    adjList[i][curr]--;

    S.push(curr);

    findEuler(i,cursor,S,order);

    return;
  }

  if(S.empty()) return;
  int next = S.top();
  S.pop();

  order[cursor] = make_pair(curr,next);

  findEuler(next,cursor+1,S,order);
}
```

◆ Articulation Point:

削除することでグラフが非連結になってしまう頂点を探す。 $O(|(E)|)$

```
void articRec(int curr, int &nVisited,
                int discover[MAX_SIZE],
                int back[MAX_SIZE],
                bool isArtic[MAX_SIZE],
                bool visited[MAX_SIZE],
                int prev, int nEdge[MAX_SIZE]){

 discover[curr] = nVisited;
 back[curr] = discover[curr];
 visited[curr] = true;
 nVisited++;


 for(int i=0;i<nNode;i++){
  if(prev==i) continue;
  if(!adjList[curr][i]) continue;

  if(!visited[i]){
   nEdge[curr]++;
   articRec(i,nVisited,discover,back,isArtic,visited,
            curr,nEdge);

   back[curr] = min(back[curr],back[i]);

   if(back[i]>=discover[curr])
    isArtic[curr]=true;
  }
  else if(visited[i])
   back[curr] = min(back[curr],discover[i]);
 }

}
```

```
void artic(bool isArtic[MAX_SIZE]){

 int discover[MAX_SIZE], back[MAX_SIZE];
 int nEdge[MAX_SIZE];
 bool visited[MAX_SIZE];

 for(int i=0;i<nNode;i++){
  nEdge[i] = 0;
  visited[i] = false;
  isArtic[i] = false;
 }

 int nVisited=0;

 //グラフが非連結な場合を考慮する
 for(int i=0;i<nNode;i++){
  if(!visited[i]){
   articRec(i,nVisited,discover,back,isArtic,visited,
            -1,nEdge);
   if(nEdge[i]<=1) isArtic[i]=false;
  }
 }

}
```

◆ Bridge:

削除することでグラフが非連結になってしまう辺を探す。 $O(|(E)|)$

```
void findBridgeRec(int curr, int &nVisited, vector< pair<int,int> > &bridgeList,
                   int discover[MAX_SIZE], int back[MAX_SIZE], int prev){
  discover[curr]=nVisited;
  back[curr] = nVisited;
  nVisited++;

  for(int i=0;i<nNode;i++){
    if(i==prev) continue;
    if(!adjList[curr][i]) continue;

    if(discover[i]==-1){
      findBridgeRec(i,nVisited,bridgeList,discover,back,curr);

      back[curr] = min(back[curr],back[i]);
    }
    else if(discover[curr]>discover[i])
      back[curr] = min(back[curr],back[i]);
  }

  if(prev!=-1 && discover[curr]==back[curr])
    bridgeList.push_back( make_pair(min(prev,curr),max(prev,curr)) );
}

void findBridge(vector< pair<int,int> > &bridgeList){
  int discover[MAX_SIZE];
  int back[MAX_SIZE];
  int nVisited=0;

  fill(discover,discover+nNode,-1);

  //グラフが非連結な場合を考慮する
  for(int i=0;i<nNode;i++){
    if(discover[i]==-1)
      findBridgeRec(i,nVisited,bridgeList,discover,back,-1);
  }
}
```

Geometry

◆ 型の定義と、演算子、ベクトルの外積の z 成分。

```
typedef pair<T,T> Point;
typedef complex<T> Vector;

typedef const Point & _Point;
typedef const Vector & _Vector;

Point operator+(_Point p, _Vector v){ // 平行移動
  return Point(p.first+v.real(), p.second+v.imag());
}

void operator+=(Point &p, _Vector v){
  p.first += v.real(); p.second += v.imag();
}

void operator+=(Point &p, _Point q){
  p.first += q.first; p.second += q.second;
}

Vector operator-(_Point p, _Point q){
  return Vector(p.first-q.first, p.second-q.second);
}

void operator/=(Point &p, _double d){
  p.first /= d; p.second /= d;
}

double cross(_Vector v, _Vector w){
  return imag(conj(v)*w);
}

double cross(_Point o, _Point p, _Point q){
  return cross(p-o, q-o);
}
```

◆ ベクトルの内積、回転、あるベクトルの周りでの反射

```
double dot(_Vector v, _Vector w){
  return real(conj(v)*w);
}

Vector &rotate(Vector &v, _double lambda){
  v *= polar(1.0, lambda);
  return v;
}

Vector &reflect(Vector &v, _Vector n){
  double vn = cross(v, n);
  double theta = (vn < 0 ? -arg(n) : arg(n));
  rotate(v, theta);
  v.imag() += 2 * vn / abs(n);
  return rotate(v, -theta);
}
```

◆ 2 直線の交点。　r に、交点の座標を格納。

```
bool intersect(_Point p, _Vector v, _Point q, _Vector w, Point &r){
  Vector pq(q - p);
  double c = cross(v, w);
  if(c == 0) return false;
  r = p + (cross(pq, w) / c) * v;
  return true;
}
```

◆ 2 線分の交差判定。

```
bool onSegment(_Point p, _Point q, _Point r){
  Vector v(q-p), w(r-p);
  return (cross(v, w) == 0 && arg(v) == arg(w) && norm(w) <= norm(v));
}

bool isIntersect(_Point p, _Point q, _Point r, _Point s){
  if(cross(p, q, r) * cross(p, q, s) < 0 && cross(r, s, p) * cross(r, s, q) < 0)
    return true;
  // 端点が線分上にあるかどうか
  return (onSegment(p, q, r) || onSegment(p, q, s) ||
          onSegment(r, s, p) || onSegment(r, s, q));
}
```

◆ 多角形の面積

```
typedef vector<Point> Polygon;
typedef const Polygon & _Polygon;

double surface(_Polygon polygon){
  double s = 0;
  int n = polygon.size();
  for(int i = 0; i < n; ++i)
    s += (polygon[i].first  - polygon[(i+1)%n].first) *
         (polygon[i].second + polygon[(i+1)%n].second);
  return fabs(s / 2.0);
}

double surface(_Point p, _Point q, _Point r){
  return fabs(cross(r-p, q-p))/2;
}
```

◆ 多角形の重心

```
// 物理的重心
void physicalGP(_Polygon polygon, Point &g){
  g = Point(0, 0);
  int n = polygon.size();
  for(int i = 0; i < n; ++i)
    g += polygon[i];
  g /= n;
}

// 幾何的重心
void geometricGP(_Polygon polygon, Point &g){
  double s = 0, area;
  int n = polygon.size();
  g = Point(0, 0);
  for(int i = 1; i < n-1; ++i){
    area = surface(polygon.front(), polygon[i], polygon[(i+1)%n]);
    s += area;
    g += area * (polygon[i] + polygon[(i+1)%n]) / 3.0;
  }
  g /= s;
}
```

◆ Convex Hull (Graham's Scan)

```
bool operator<(_Point p, _Point q){
  if(p.second == q.second) return p.first < q.first;
  return p.second < q.second;
}

// < を <= に変えることで、直線上の点を除ける。
double isInvalidTurn(_Point o, _Point p, _Point q){
  return (cross(p-o, q-o) < 0);
}

class PolarAngleSorter{
private:
  Point o;

public:
  PolarAngleSorter(_Point o) : o(o){
  }

  bool operator()(_Point p, _Point q) const{
    Vector v(p-o), w(q-o);
    double c = cross(v, w);
    if(c == 0) return abs(v) < abs(w);
    return c > 0;
  }
}

vector<Point> grahamScan(vector<Point> polygon){
  vector<Point> hull;
  sort(polygon.begin(), polygon.end(),
       PolarAngleSorter(*min_element(polygon.begin(), polygon.end())));

  hull.push_back(polygon[0]);
  hull.push_back(polygon[1]);
  for(int i = 2; i < polygon.size(); ++i){
    while(hull.size() >= 2 && isInvalidTurn(hull[hull.size()-2], hull.back(), polygon[i]))
      hull.pop_back();
    hull.push_back(polygon[i]);
  }
}
```

その他のアルゴリズム

◆ **ユークリッドのアルゴリズム：** 与えれたA,Bの最大公約数を求める。

```
int gcd(int a, int b){
  if(b==0) return a;
  else return gcd(b,a%b);
}
```

◆ **拡張ユークリッド:**

与えられたA,Bに対し、 $AX+BY=GCD(A,B)$ となる、
X,Y,GCD(A,B)を求める。

```
class Data{
public:
  int d,x,y;

  Data(){}
  Data(int d, int x, int y): d(d), x(x), y(y) {}
};

Data extendGCD(int a, int b){
  if(b==0) return Data(a,1,0);

  Data ret = extendGCD(b,a%b);

  return Data(ret.d,ret.y,ret.x-a/b*ret.y);
}
```

◆ **power:**

与えられた n と p に対し、 p^n を求める。 $O(\log n)$

```
double power(double n, int p){
  if(p==0) return 1;
  if(p==1) return n;

  if(p%2 == 0 ){
    double tmp=power(n,p/2);
    return tmp*=tmp;
  }
  else{
    double tmp=power(n,p/2);
    return tmp*=tmp*n;
  }
}
```

◆ シンプソンの公式:

$\int_a^b f(x)dx$ を求める。誤差は h^4 に抑えられる。分割数は偶数にする。

```
double simpson(){
  const static double h=(B-A)/MAX_DIV;
  double sum=0;

  for(int i=0;i<MAX_DIV;i+=2)
    sum+=h*( func(h*i) + 4*func(h*(i+1)) + func(h*(i+2)) )/3;

  return sum;
}
```

◆ Zeller の公式: 年月日の曜日を求める。

```
int zeller(int y, int m, int d){
  if(m<3){ y--; m+=12; }
  return (y + y/4 - y/100 + y/400 + (13*m+8)/5 + d) % 7;
}
```

◆ 閏年判定

```
bool isLeapYear(int y){
  return y%400==0 || (y%4==0 && y%100!=0);
}
```

数表など

◆ カタラン数:

$$C_N = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

ノード数 n に対し、バイナリツリーを構成する方法は C_n 通り。
正 n 角形を n−2 つの三角形に分ける方法は C_{n-2} 通り。

n	C_n
1	1
2	2
3	5
4	14
5	42
6	132
7	429
8	1430
9	4862
10	16796
11	58786
12	208012
13	742900
14	2674440
15	9694845
16	35357670
17	129644790
18	477638700
19	1767263190
20	6564120420
21	24466267020
22	91482563640
23	343059613650
24	1289904147324

◆ カレンダーに関して

各月の日数は、　31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31

閏年の場合、　　31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31

年月日と曜日の組み合わせは 400 年のサイクルで繰り返される。

◆ **ASCII CODE**

0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL
8 BS	9 HT	10 NL	11 VT	12 NP	13 CR	14 SO	15 SI
16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB
24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US
32 SP	33 !	34 ”	35 #	36 $	37 %	38 &	39 '
40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _
96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 DEL

STL

◆ string

```
const CharType *c_str() const;

bool empty() const;

string substr(size_type indx=0, size_type len=npos) const;
```

◆ algorithm

```
size_t count(InIter start, InIter end, const T &val);
size_t count_if(InIter start, InIter end, UnPred pfn);

InIter find(InIter start, InIter end, const T &val);
InIter find_if(InIter start, InIter end, UnPred pfn);

Func for_each(InIter start, InIter end, Func fn);

ForIter upper_bound(ForIter start, ForIter end, const T &val);
ForIter upper_bound(ForIter start, ForIter end, const T &val, Comp cmpfn);
ForIter lower_bound(ForIter start, ForIter end, const T &val);
ForIter lower_bound(ForIter start, ForIter end, const T &val, Comp cmpfn);

const T &max(const T &i, const T &j);
ForIter max_element(ForIter start, ForIter end);
ForIter max_element(ForIter start, ForIter end, Comp cmpfn);
const T &min(const T &i, const T &j);
ForIter min_element(ForIter start, ForIter end);
ForIter min_element(ForIter start, ForIter end, Comp cmpfn);

bool next_permutation(BiIter start, BiIter end);
bool next_permutation(BiIter start, BiIter end, Comp cmpfn);
bool prev_permutation(BiIter start, BiIter end);
bool prev_permutation(BiIter start, BiIter end, Comp cmpfn);

void sort(RanIter start, RanIter end);
void sort(RanIter start, RanIter end, Comp cmpfn);
void partial_sort(RanIter start, RanIter min, RanIter end);
void partial_sort(RanIter start, RanIter mid, RanIter end, Comp cmpfn);

ForIter remove(ForIter start, ForIter end, const T &val);
ForIter remove_if(ForIter start, ForIter end, UnPred pfn);

ForIter replace(ForIter start, ForIter end, const T &old, const T &new);
ForIter replace_if(ForIter start, ForIter end, UnPred pfn, const T &val);

void reverse(BiIter start, BiIter end);

ForIter unique(ForIter start, ForIter end);
ForIter unique(ForIter start, ForIter end, BinPred pfn);
```

チェックリスト

◆ 問題に関して

問題文の見落としはないか。
非連結なグラフが入力として入ってくるか。
素数に1は含まれるか。
case sensitive かどうか。
データのサイズが 14 などといった中途半端なのは、
全探索でぎりぎり間に合うよう設定されたためなのもかもしれない。

◆ 入力に関して

入力は正しくとれているか。
row と column の値が逆になっていないか。
0-origin か 1-origin か。

◆ 出力に関して

出力の最後に'.'を付け忘れていないか。
解が存在しないときは "No Solution" 等と出力すべきではないか 。

◆ 処理に関して

負の値、0 の値を考慮しているか。
ライブラリが間違っている可能性は？
グローバル変数の初期化は忘れてないか。
再起関数で末端についたときに return しているか。
NORTH で row--, SOUTH で row++。
return VALUE; をし忘れていないか。
vector で要素を削除した後、
その場所以降を指していた iterator は無効になる。
関数に vector/list とその iterator を引数にとると vector/list は
新しくコピーされるので iterator は無効となってしまう。
よって参照で渡す。
LCS, EditDistance では初期化の範囲を for(int i=0;i<=N;i++) とする。
Vector や list の要素を erase したあとの添え字の場所は正しいか。

-0.000000 という表示を 0.000000 にしたい時

```
if(-0.0000005<x && x<0.0)
        printf("%.6lf\n",0.0);
else
        printf("%.6lf\n",x);
```

とする。

◆ 凡ミスチェック

using namespace std;を忘れていないか。
配列の範囲は合っているか。
あるグローバル変数と同じ名前の変数をローカル変数として
　宣言してもコンパイルエラーにならないことがある。

余り算による index のアクセスは
```
curr = (curr+1)%n
curr = (curr+n-1)%n
```

```
for(int i=0;i<g[curr].size();i++)
    Q.push(i);
```
ではなく
```
for(int i=0;i<g[curr].size();i++)
    Q.push(g[curr][i]);
```
ではないか。

こういう事をやっていないか。
```
for(int i=0;i<MAX_SIZE;i++)
  for(int j=0;j<MAX_SIZE;i++)
```

```
#define END -1
```
ではなく
```
#define END (-1)
```
とする。

◆ 処理が遅いとき

出発点と目的地が明らかならば、
　始点と終点からの BFS で高速化ができる。
無造作に選ぶよりソートしてから選ぶことで、
　枝刈りの効率は上がるかもしれない。
vector ではなく配列を使うと TLE を免れるかもしれない。
list は vector と比べても処理が遅い。
vector<bool>は処理が遅い。
vector/list/string の size()はサイズが 0 のとき size()-1 とすると、
　無限大の値が帰ってくる。