DP

- Longest increasing subsequence (LIS):

```
vi data, dp, ans;
dp.resize(data.size(), 1); ans.resize(data.size(), -1);
ii best = {0,0};
for(int i = 0; i < data.size(); ++i){
        for(int j = 0; j < i; ++j){
                if(data[j] < data[i]){
                        //dp[i] = max(dp[i], dp[j] + 1);
                        if (dp[j]+1 > dp[i]){
                                dp[i] = dp[j]+1;
                                ans[i] = j;
                        }
                }
        }
        if (best.x < dp[i]) best = {dp[i], i};
}
int index = best.y; si print;
while(index > -1){
        print.insert(data[index]);
        index = ans[index];
}
printf("Size of LIS: %d\n",best.x);
for (auto x : print) printf("%d\n", x);
```

- Knapsack Problem w/ retrieve of items:

```
vector<vi > dp;
vector<int > weigth, value;
si uses;
int bt(int i, int j){//Get items
        if (i-1 < 0) return 0;
        if (dp[i][j] == dp[i-1][j]){
                return bt(i-1,j);
        }
        uses.insert(i);
        return bt(i-1, j-weigth[i]) + 1;
}
//Knapsack
weigth.push_back(0); value.push_back(0);
dp.resize(n_items+1, vi(bag+1));
while(n_items--){
        int w, v; cin >> w >> v;
        weigth.push_back(w);
        value.push_back(v);
}

for (int i = 0; i < dp.size(); i++){
        for (int j = 0; j < dp[0].size(); j++){
                int a,b;
                (i > 0) ? a = dp[i-1][j] : a = 0;
                (i > 0 && j-weigth[i] >= 0) ? b = dp[i-1][j-weigth[i]] + value[i] : b = 0;
                dp[i][j] = max(a,b);
        }
}
printf("Max value: %d\n", dp[dp.size()-1][dp[0].size()-1]);
printf("Number of itens in the bag: %d\n", bt(dp.size()-1, dp[0].size()-1));
for (int i : uses)
        printf("Weigth: %d Value: %d\n", weigth[i], value[i]);
```

-Knapsack Problem Recursively:

```
int S, N;// Size of bag & Number of items
int dp[2000][2000];
vector<ii > items; // (weight, price)

int knapsack(int pos, int avaible){
        if (pos == items.size()) return 0;

        if (dp[pos][avaible] != -1) return dp[pos][avaible];

        int solution_include = -1;
        if (avaible >= items[pos].x){
                solution_include = knapsack(pos+1, avaible - items[pos].x) + items[pos].y;
        }
        int solution_exclude = knapsack(pos+1, avaible);

        return dp[pos][avaible] = max(solution_include, solution_exclude);
}

knapsack(0, S);
```

- Longest Common Subsequence (LCS):

- Recursive:

```
string str1, str2;
int dp[1005][1005];

int LCS(int i, int j){
        if (dp[i][j] != -1) return dp[i][j];

        if (str1[i] == '#' || str2[j] == '#')
                return dp[i][j] = 0;
        else if (str1[i] == str2[j])
                return dp[i][j] =  1 + LCS(i+1, j+1);
        else
                return dp[i][j] = max(LCS(i+1,j), LCS(i,j+1));
}
cin >> str1 >> str2;
str1 += "#"; str2 += "#";
memset(dp, -1, sizeof(dp));
printf("Size of LCS: %d\n", LCS(0,0));
```

- Iterative:

```
string X, Y;
int lcs(int m, int n ){
   int L[m + 1][n + 1];
   int i, j;

   for (i = 0; i <= m; i++){
      for (j = 0; j <= n; j++){
         if (i == 0 || j == 0)
            L[i][j] = 0;
         else if (X[i - 1] == Y[j - 1])
            L[i][j] = L[i - 1][j - 1] + 1;
         else
            L[i][j] = max(L[i - 1][j], L[i][j - 1]);
      }
   }

   return L[m][n];
}
```

- Cutting problem:

```
int N; si abc;
int dp[4005]; memset(dp, -1, sizeof(dp));
//Cut N elements into pieces of size a or b or c
int cut(int pieces, int rem){
        if (rem == 0)
                return dp[rem] = max(pieces, dp[rem]);
        if (dp[rem] != -1)
                dp[rem] = max(pieces, dp[rem]);
        else{
                int best = 0;
                for (auto x : abc){
                        if (rem >= x)
                                best = max(best, cut(pieces+1, rem-x));
                }
                return dp[rem] = best;
        }
}
printf("Number of sizes: %d\n", cut(0, N));
```

- Submatrix Sum:

```
int N = 1025, M = 1025;
int mat[1026][1026], aux[1026][1026];
void preProcess(){
        for (int i=0; i<N; i++)
    aux[0][i] = mat[0][i];

  for (int i=1; i<M; i++)
    for (int j=0; j<N; j++)
      aux[i][j] = mat[i][j] + aux[i-1][j];

  for (int i=0; i<M; i++)
    for (int j=1; j<N; j++)
      aux[i][j] += aux[i][j-1];
}
//Top-Left, Right-Bottom (i, j)
int sumQuery(int tli, int tlj, int rbi,  int rbj){
   int res = aux[rbi][rbj];
   if (tli > 0)
     res -= aux[tli-1][rbj];
   if (tlj > 0)
     res -=  aux[rbi][tlj-1];
   if (tli > 0 && tlj > 0)
     res +=  aux[tli-1][tlj-1];
   return res;
}
memset(mat, 0, sizeof(mat));
while(n--) mat[x][y] += w;
preProcess();
pair<int, ii> best = {0,{0,0}};//Value, Coords
for (int i = 0; i < 1025; i++){
        for (int j = 0; j < 1025; j++){
                int tli = 0, tlj = 0, rbi = i, rbj = j;
                int res = sumQuery(tli,tlj,rbi,rbj);
                if (res > best.x){
                        best.x = res;
                        best.y.x = i;
                        best.y.y = j;
                }
        }
}
printf("The submatrix sum formed by (%d , %d) is: %d\n", best.y.x, best.y.y, best.x);
```

- Coin exchange problem:

```
ll dp[N];
ll calc(ll n){
	if (n == 0) return 0;
	if (dp[n] != 0) return dp[n];


	ll por2 = n/2, por3 = n/3, por4 = n/4;
	ll dividir = calc(por2) + calc(por3) + calc(por4);

	return	dp[n] = max(n, dividir);
}
printf("", calc(N));
```

- Digit DP [a,b] = g(a,b):

```
int m;
int dp[ms][2][2][m];//ms = num of digits
vi numberA, numberB;

int solve(int pos, int smaller, int bigger, int num){
	if (dp[pos][smaller][bigger][num] != -1) return dp[pos][smaller][bigger][num];

	if (pos == int(numberA.size()))//a e b must be of same size
		return (ll) (num%m == 0);	//divisible by m

	int ans = 0;
	int limSup = (smaller) ? 9 : numberB[pos];
	int limInf = (bigger) ? 0 : numberA[pos];

	for (int digit = limInf; digit <= limSup; digit++){
		int new_smaller = smaller, new_bigger = bigger;

		if (!smaller && digit < limSup)  new_smaller = 1;
		if (!bigger && digit > limInf)  new_bigger = 1;

		ans += solve(pos+1, new_smaller, new_bigger, (num*10 + digit));
	}

	return dp[pos][smaller][bigger][num] = ans;
}

string a, b;
memset(dp, -1, sizeof(dp));
//make
if (a.size() < ms)
	int times = ms-a.size();
	while(times--)
		numberA.push_back(0);
}
if (b.size() < ms){
	int times = ms-b.size();
	while(times--)
		numberB.push_back(0);
}
for (auto x : a)
	numberA.push_back(x-'0');
for (auto x : b)
	numberB.push_back(x-'0');
```

- Digit DP [a,b] = g(0,b)-g(0,a-1):

```
vi number;
int d;
ll dp[ms][105][2][105];

ll solve(int pos, int sum, int smaller, int num){
	if (dp[pos][sum][smaller][num] != -1)
		return dp[pos][sum][smaller][num];
	if (pos == (int) number.size())
		return (ll)((num%d + sum%d) == 0);

	ll ans = 0;
	int lim = (smaller) ? 9 : number[pos];

	for (int digit = 0; digit <= lim; digit++){
		int new_smaller = smaller;
		if (!smaller && digit < lim) new_smaller = 1;
		int newnum = num+(digit*pow(10,number.size()-1-pos));
		ans += solve(pos+1, sum+digit, new_smaller, newnum%d);
	}

	return dp[pos][sum][smaller][num] = ans;
}

//Solve [a,b] = solve(b)-solve(a-1)
//solve a
memset(dp, -1, sizeof(dp));
number.resize(0);
for (auto x : a)
	number.push_back(x-'0');
for (int i = number.size()-1; i >= 0; i--){
	if (number[i] -1 >= 0)		{number[i]--; break;}
	else number[i] = 9;
}
ll A = solve(0, 0, 0, 0);//<---
//solve b
memset(dp, -1, sizeof(dp));
number.resize(0);
for (auto x : b)
	number.push_back(x-'0');
ll B = solve(0, 0, 0, 0);//<---

cout << B-A << endl;
```

g++ -std=c++11 -Wall -Wextra -pedantic-errors main.cpp -o main

- TSP:

```
const int inf = 0x3f3f3f3f, ms = 15;
int n, graph[15][15], dp[15][(1<<V)];//n = 15 vertices
//graph[i][j] means distance (i --> j)
int solve(int pos, int visited){
	if (visited+1 == (1<<(n+1)))
		return graph[pos][0];// 0 if doesnt go back to initial vertice

	int &ans = dp[pos][visited];
	if(~ans) return ans;

	ans = inf;

	for (int i = 0; i < n+1;i++){
		if ( !(visited & (1<<i)) ){
			ans = min(ans, graph[pos][i] + solve(i, visited | (1<<i)));
		}
	}

	return ans;
}

memset(dp, -1, sizeof(dp));
memset(graph, 0, sizeof(graph));

for (int i = 0; i <= n; i++){
	for (int j = 0; j <= n; j++){
		if (j == i) continue;
		cin >> graph[i][j];
	}
}
printf("%d\n", solve(0,0));
```