1033 - Generating Palindromes

Category: DP, Palindrome

Given a string S consists of only lowercase letter(1<=|S|<=100). And you have to find the minimum number of characters required to make the given string to a palindrome if you are only allowed to insert characters at any position of the string. Number of test case T(1<=T<=200).

Sample Code O(n^2):

```
string s;
int n,dp[101][101];
int fun(int i,int j)
{
//    if(i==j)return 0;
//    if(j==i+1){
//        if(s[i]==s[j])return 0;
//        return 1;
//    }

    if(i>j)return 0;

    int& ret = dp[i][j];
    if(ret!=-1)return ret;

    ret = 100000;
    if(s[i]==s[j])ret = min(ret,fun(i+1,j-1));
    else {
        ret = min(ret,1+fun(i+1,j));
        ret = min(ret,1+fun(i,j-1));
    }
    return ret;
}
int main()
{
    int tt; cin>>tt;
    for(int ks=1; ks<=tt; ks++)
    {
        cin>>s;
        n = s.size();
        memset(dp,-1,sizeof(dp));
        int ans = fun(0,n-1);
        cout<<"Case "<<ks<<": "<<ans<<endl;
    }
    return 0;
}
```

10453 - Make Palindrome

Category: DP, LCS, Palindrome:

Given a string S (1<=|S|<=1000), you need to add as few characters as possible to make the new string a palindrome. Outputs the minimum number of characters added and the new string.

Sample Code:

```
int la,lb,dp[1005][1005];
string a,b,s;
int lcs(int i, int j){
    if(i==la || j==lb)return 0;
    if(dp[i][j]!=-1)return dp[i][j];
    int ret,ret1,ret2;
    if(a[i]==b[j]) ret = 1+lcs(i+1,j+1);
    else{
        ret1 = lcs(i+1,j); ret2 = lcs(i,j+1);
        ret = max(ret1,ret2);
    }
    return dp[i][j] = ret;
}
void path(int i, int j){
    if(i==la){ for(int k=j; k<lb; k++)s+=b[k]; return; }
    if(j==lb){ for(int k=i; k<la; k++)s+=a[k]; return; }
    int ret = lcs(i,j);
    if(a[i]==b[j]){ s+=a[i]; path(i+1,j+1); }
    else{
        int ret1 = lcs(i+1,j);
        int ret2 = lcs(i,j+1);
        if(ret==ret1){ s+=a[i]; path(i+1,j); }
        else{ s+=b[j]; path(i,j+1); }
    }
}
int main(){
    while(cin>>a)
    {
        b=a; reverse(b.begin(),b.end());
        la = a.size(); lb = b.size();
        memset(dp,-1,sizeof(dp));
        int ls = lcs(0,0);
        int ans = la-ls;
        s = "";
        path(0,0);
        cout << ans << " " << s << endl;
    }
}
```

11151 - Longest Palindrome

Category: DP, LCS, Palindrome

Given a string S(1<=|S|<=1000). You have to find the length of the longest palindrome you can get by removing zero or more characters from it.

Sample Code:

```
string a,b; int dp[1005][1005];
int lcs(int i, int j){
    if(i==a.size() || j==b.size())return 0;
    if(dp[i][j]!=-1)return dp[i][j];
    int ret=0;
    if(a[i]==b[j])ret = max(ret,1+lcs(i+1,j+1));
    else{
        ret = max(ret,lcs(i+1,j));
        ret = max(ret,lcs(i,j+1));
    }
    return dp[i][j] = ret;
}
int main()
{
    int tt; scanf("%d",&tt); getchar();
    for(int ks=1; ks<=tt; ks++){
        getline(cin,a);
        b = a; reverse(b.begin(),b.end());
        memset(dp,-1,sizeof(dp));
        printf("%d\n",lcs(0,0));
    }
    return 0;
}
********************** Iterative DP ************************
int n,dp[1005][1005]; char s[1005];
int main(){
     int T; scanf("%d%*c", &T);
     while(T--){
          gets(s); n = strlen(s);
          for (int i = 1; i <= n; i++){
               for (int j = 1; j <= n; j++){
                    if(s[i-1]==s[n-j]) dp[i][j] = 1+dp[i-1][j-1];
                    else dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
               }
          }
          printf("%d\n", dp[n][n]);
     }
     return 0;
}
```

1025 - The Specials Menu

Category: DP, Palindrome

Given a string S consists of only uppercase letter(1<=|S|<=60). You have to calculate the number of ways that you could remove any letters from the string S, so that it would become a palindrome. Number of test case T(1<=T<=200).

Idea:

If there is just one character in the string, it's a palindrome. If there are more characters left in the string, you can remove character from left fun(i+1, j) or right fun(i, j-1). But in both recursions you will count fun(i+1, j-1). To avoid counting twice, we subtract it from ret. There is only one case left: when s[i] == s[j], you can keep both characters (do not remove them) and then you should call fun(i+1, j-1) to build the middle and we add one because we never count the palindrome consisting of just these two characters.

Sample Code O(n^2):

```
#define ll long long
string s; ll dp[70][70];
ll fun(int i,int j)
{
    if(i==j) return 1;
    if(i>j)return 0;

    ll &ret = dp[i][j];
    if(ret!=-1)return ret;

    // if(s[i]==s[j]) ret = (1+fun(i+1,j-1)) + fun(i+1,j)+fun(i,j-1)-fun(i+1,j-1);
    if(s[i]==s[j]) ret = 1 + fun(i+1,j) + fun(i,j-1);
    else ret = fun(i+1,j) + fun(i,j-1) - fun(i+1,j-1);
    return ret;
}
int main()
{
    int tt; cin>>tt;
    for(int ks=1; ks<=tt; ks++)
    {
        cin>>s;
        int n = s.size();
        memset(dp,-1,sizeof(dp));
        ll ans = fun(0,n-1);
        cout<<"Case "<<ks<<": "<<ans<<endl;
    }
    return 0;
}
```

1044 - Palindrome Partitionin

Category: DP, Palindrome

A palindrome partition is the partitioning of a string such that each separate substring is a palindrome. For example, the string "ABACABA" could be partitioned in several different ways, such as {"A","B","A","C","A","B","A"}, {"A","BACAB","A"}, {"ABA","C","ABA"}, or {"ABACABA"}, among others.

Given a string **s** of uppercase letters with length no more than 1000. Return the minimum possible number of substrings in a palindrome partition of **s**. No. of Test case T(1<=T<=40).

Samle Code $O(n^2+ n^2)$:

```
string s;
int n,dp[1005],ispalindrome[1005][1005];
int palindrome(int i,int j)
{
    if(i>=j)return ispalindrome[i][j]=1;
    if(ispalindrome[i][j]!=-1)return ispalindrome[i][j];
    int ret = 0;
    if(s[i]==s[j])ret = palindrome(i+1,j-1);
    return ispalindrome[i][j] = ret;
}
int fun(int i)
{
    if(i==n)return 0;
    if(dp[i]!=-1)return dp[i];
    int ret = 1000000;
    for(int j=i; j<n; j++){
        if(palindrome(i,j)==1){
            ret = min(ret,1+fun(j+1));
        }
    }
    return dp[i] = ret;
}
int main()
{
    int tt; cin>>tt;
    for(int ks=1; ks<=tt; ks++) {
        cin>>s; n = s.size();
        memset(ispalindrome,-1,sizeof(ispalindrome));
        memset(dp,-1,sizeof(dp));
        cout << "Case " << ks << ": " << fun(0) << endl;
    }
    return 0;
}
```

1205 - Palindromic Numbers

Category: Digit DP, Palindrome

Given two integers **a** and **b** ($0 \le a, b \le 10^{17}$). you have to find the number of palindromic numbers between **a** and **b** (inclusive). Number of test case T **(≤ 200).**

Observation:

1. If we select the half of the palindrome, the next half is already fixed.
2. If our current Number is smaller than the range, after adding next half, the number will always be small.

Sample Code:

```
#define ll long long
int ar[20]; ll var, dp[20][2][20];
bool check(int index,int len){
      int id=0, oo[20];  ll d = var;
      for( ; d!=0 ; id++, d/=10) oo[id] = d%10;
      for(int i=id-1; i>index; i--) d = 10*d+oo[i];
      for(int i=index+((len%2)? 2:1);i<len; i++) d=10*d+oo[i];
      if(d <= var) return 1;
      return 0;
}
ll fun(int pos, int flag, int len){
      if(pos == (len/2)-1){
            if(flag && len) return check(pos,len);
            return 1;
      }
      if(flag==0 && dp[pos][flag][len]!=-1) return dp[pos][flag][len];
      ll res=0, hi = 9; if(flag) hi = ar[pos];
      for(int i=0; i<=hi; i++) {
            res += fun(pos-1, flag&&(i==hi), i ? (len==0 ? pos+1 : len): len);
      }
      if(flag) return res;
      return dp[pos][flag][len] = res;
}
ll Solve(ll v){
      var = v; int ind;
      for(ind=0; v!=0; ind++,v/=10) ar[ind]=v%10;
      memset(dp,-1,sizeof(dp));
      return fun(ind-1,1,0);
}
int main(){
      int t; scanf("%d",&t);
      for(int ks=1; ks<=t; ks++){
            ll a,b; scanf("%lld%lld",&a,&b); if(a>b) swap(a,b);
            printf("Case %d: %lld\n", ks, Solve(b) - Solve(a-1));
      }
}
```

Toph – Palindromist

Category: Hashing, Palindrome

Given an initial string S(1<=|S|<=10000) and number of updates U(1<=U<=10000). Each update is of the form: s c k, where s is either "L" or "R" denoting prepending at left side or appending at right side of the string , c is a lowercase letter going to append/prepend and k(1<=k<=100) is the number of time c is added to string.

Then on next U lines, print whether the string formed by the corresponding update operation is a palindrome or not.

Sample Code(Using Hash):

```
#define llu unsigned long long #define base 31LL
llu sum[1000005],cum[1000005];
int main(){
    sum[0]=1; cum[0]=1;
    for(int i=1; i<1000005; i++){
        sum[i] = (sum[i-1]*base);
        cum[i] = (cum[i-1]+sum[i]);
    }
    int tt; scanf("%d",&tt);
    for(int ks=1; ks<=tt; ks++) {
        printf("Case %d:\n",ks);
        string s; cin>>s;   int n = s.size();
        llu fr=0,rv=0;
        for(int i=0; i<n; i++) fr += s[i]*sum[i];
        for(int i=n-1,j=0; i>=0; i--,j++) rv += s[i]*sum[j];

        int qq; scanf("%d",&qq);
        for(int q=1; q<=qq; q++) {
            getchar();
            char tp,ch; llu k; scanf("%c %c %llu",&tp,&ch,&k);
            llu cm = cum[n+k-1]-cum[n-1];
            n = n+k;
            if(tp=='L'){
                fr = (ch*cum[k-1]) + (fr*sum[k]);
                rv = ((ch*cm) + rv);
            }
            else{
                fr = ((ch*cm) + fr);
                rv = (ch*cum[k-1]) + (rv*sum[k]);
            }
            if(fr==rv)printf("Yes\n");
            else printf("No\n");
        }
    }
    return 0;
}
```

1258 - Making Huge Palindromes

Category: KMP, Palindrome

Given a non-empty string S(**1 ≤ length(S) ≤ 10^6**). The given string may or may not be palindrome. Your task is to make it a palindrome. But you are only allowed to add characters at the right side of the string. And of course you can add any character you want, but the resulting string has to be a palindrome, and the length of the palindrome should be as small as possible.

For example, the string is '**bababa**'. You can make many palindromes including **bababababab**, **babababab**, **bababab**
Since we want a palindrome with minimum length, the solution is '**bababab**' cause its length is minimum.

Basic Code:

```
string a,b,s;
int fa[2000005],la,ls;
int failure()
{
    fa[0]=0;
    for(int i=1,j=0; i<ls; i++){

        while(s[i]!=s[j]&&j>0){
            j = fa[j-1];
        }

        if(s[i]==s[j]){ fa[i]=j+1; j++; }
        else{ fa[i]=0; j=0; }
    }
    return fa[ls-1];
}
int main()
{
    int tt; scanf("%d",&tt);
    for(int ks=1; ks<=tt; ks++)
    {
        cin>>a;
        b = a; reverse(b.begin(),b.end());
        s = b+"#"+a;
        la = a.size(); ls=s.size();
        int res = failure();
        int ans = (2*la)-res;
        printf("Case %d: %d\n",ks,ans);
    }
    return 0;
}
```