دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

درس تحلیل و طراحی الگوریتم‌ها

راه‌حل تکالیف:

سری سوم – برنامه‌نویسی پویا

نام استاد درس:

دکتر بهروز شاهقلی

نام دستیاران آموزشی درس:

رضا رستگاری

جواد جعفری

زهرا تاکی

سارا کهتری

امیرحسین عرب‌پور

میلاد محمدی

نیم‌سال دوم تحصیلی ۱۴۰۰-۱۴۰۱

فهرست

سوال اول: احمد مایه

این سـوال نوعی Largest Sum Contiguous Subarray می باشـد. در ابتدا باید به مقدار p از هر عدد کم کنیم. سپس طرح الگوریتم به شکل زیر است:

فرض کنید جواب مسئله تا عنصر k ام را داریم و این جواب در ans ذخیره شده است. حال عنصر k+1 ام را در نظر می‌گیریم. اگر جمع کردن این عنصر باعث افزایش ans شود، ans را آپدیت می‌کنیم و در غیر این صورت کاری نمی‌کنیم.

```cpp
#include <bits/stdc++.h>
using namespace std;

int num, dp[100000+100], n, p, ans = 0;

int main()
{
   cin >> n >> p;
   dp[0] = 0;
   for (int i=1; i<=n; i++){
      cin >> num;
      dp[i] = max(dp[i-1] + num-p, num-p);
      ans = max(ans, dp[i]);
   }
   cout << ans << endl;
   return 0;
}
```

سوال دوم: اکبر مایه

حل این سوال به طرز فکر سوال احمد مایه نیاز دارد.

راه حل بدیهی آن است که هر ترکیب از دو ستون و دو سطر را در نظر بگیریم تا برخورد آن ها به زیر جدول ها ممکن را بدهند، و سپس با یک حلقه تو در تو مجموعه اعداد داخل آن ها را حساب کنیم. پیچیدگی زمانی این کار از O(6) است.

اگر جدولی جدید بسازیم که در هر خانه آن مجموع اعداد آن سطر (یا آن ستون) تا آن خانه را ذخیره کنیم، میتوانیم با روشی شبیه به سوال قبل و آپدیت کردن بر اساس در نظر گرفتن عنصر جدید، جواب را با O(3) بیابیم.

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

void print(ll **num, ll n){
   for (ll i = 0; i < n + 2; i++){
      for (ll j = 0; j < n + 2; j++){
         cout << num[i][j];
      }
      cout << endl;
   }
}

int main(){
   ll n; cin >> n;
   ll mx = 0; // old friends;
   ll m = 0;
   ll **num = new ll *[n + 2];
   for (ll i = 0; i < n + 2; i++){
      num[i] = new ll[n + 2];
   }
   for (ll i = 0; i < n + 2; i++){
      for (ll j = 0; j < n + 2; j++)
         num[i][j] = 0;
   }
   for (ll i = 1; i < n + 1; i++){
      for (ll j = 1; j < n + 1; j++){
         cin >> num[i][j];
      }
   }

   for (ll i = 1; i < n + 1; i++){
      ll sum = 0;
      for (ll j = 1; j < n + 2; j++){
         num[i][j] += sum;
         sum = num[i][j];
      }
   }
   // print(num,n);
   for (ll c1 = 1; c1 < n + 1; c1++){
      for (ll c2 = c1; c2 < n + 1; c2++){
         m = 0;
         for (ll r = 1; r < n + 1; r++){
            m += (num[r][c2] - num[r][c1 - 1]);
            if (mx < m){
               mx = m;
            }
            if (m < 0){
               m = 0;
            }
         }
      }
   }
   cout << mx << endl;
}
```

سوال سوم: فقط اولویت‌ها مهم هستند

این سوال یک سوال کوله‌پشتی (knapsack) ساده است. در این سوال اولویت‌ها، همان ارزش (value) و زمان همان وزن (weight) است. بعد از بدست آوردن مقدار بهینه با استفاده از الگوریتم کوله‌پشتی، کالاهای انتخاب شده را بدست می‌آوریم.

برای این کار از index جواب در آرایه دو بعدی کوله‌پشتی شروع کرده، و به سمت index آغازی می‌رویم. در هر ردیف در صورتی که کالایی انتخاب شده باشد، باید در ردیف قبلی آن مقدار از مقدار قبلی کم شده باشد.

```cpp
#include <bits/stdc++.h>
using namespace std;

int knapsack[2007][2007];
typedef pair<int, int> pi;

int main(){
  int n, c;
  while (cin >> c >> n){
    pi items[n];
    for (int i = 0; i < n; i++){
      cin >> items[i].first >> items[i].second;
    }
    for (int i = 1; i < n + 1; i++){
      for (int w = 0; w < c + 1; w++){
        if ((w - (items[i - 1].second)) >= 0){
          knapsack[i][w] = max(knapsack[i - 1][w],
          knapsack[i - 1][w - (items[i - 1].second)] + (items[i - 1].first));
        }
        else{
          knapsack[i][w] = knapsack[i - 1][w];
        }
      }
    }
    vector<int> path;
    int cap = c;
    int count = 0;
    for (int i = n; i > 0; i--){
      if (knapsack[i][cap] == knapsack[i - 1][cap]){
        continue;}
      else{
        cap -= items[i - 1].second;
        count += 1;
        path.push_back(i - 1);
      }
    }
    cout << count << '\n';
    sort(path.begin(), path.end());
    for (auto vi : path){cout << vi << " ";}
    cout << '\n';
  }
}
```

سوال چهارم: KAUAN

میتوان حروف انگلیسی را راس های یک گراف و هزینه تبدیل را وزن یال بین دو راس در نظر گرفت.

در این صورت سوال به پیدا کردن کوتاه ترین مسیر بین هر جفت راس تبدیل میشود که با توجه به محدودیت ها، الگوریتم فلوید برای آن مناسب میباشد.

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll dist[26][26];
string a,b,ans="";
ll sum=0,n;
int main(){
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  for(ll i=0;i<26;i++){
    for(ll j=0;j<26;j++){
      if(i==j)dist[i][j]=0;
      else
        dist[i][j]=1000000;
    }
  }
  cin >> a >> b >> n;
  ll al = a.length();
  ll bl = b.length();
  for(ll i=0;i<n;i++){
    char u,v;
    ll w;
    cin >> u >> v >> w;
    dist[u-'a'][v-'a'] = min(dist[u-'a'][v-'a'],w);
  }
  if(al!=bl){
    cout<<-1<<endl;
    return 0;
  }
  for(ll k=0;k<26;k++){
    for(ll i=0;i<26;i++){
      for(ll j=0;j<26;j++){
        dist[i][j] = min(dist[i][j],dist[i][k]+dist[k][j]);
      }
    }
  }
  for(ll i=0;i<al;i++){
    if(a[i]==b[i]){
      ans+=a[i];
      continue;
    }
    ll add=1000000;
    char c;
    if(dist[a[i]-'a'][b[i]-'a']== 1000000 && dist[b[i]-'a'][a[i]-'a'] == 1000000 ){
      add = 1000000;
    }
    else if(dist[a[i]-'a'][b[i]-'a'] < dist[b[i]-'a'][a[i]-'a']){
      add=dist[a[i]-'a'][b[i]-'a'];
      c = b[i];
    }
    else{
      add=dist[b[i]-'a'][a[i]-'a'];
      c=a[i];
    }
    for(ll j=0;j<26;j++){
      ll add2 = dist[a[i]-'a'][j]+dist[b[i]-'a'][j];
      if(add2<add){
        add = add2;
        c='a'+j;
      }
    }
    if(add==1000000){
      cout<<-1<<endl;
      return 0;
    }
    sum+=add;
    ans+=c;
  }
  cout<<sum<<endl;
  cout<<ans<<endl;
  return 0;
}
```

سوال پنجم: دوباره کامی

در این سوال باید حالت‌های مختلفی که میتوان پرانتز اول را تشکیل داد، را مشخص کنیم. (نیازی به محاسبه پرانتز دوم نیست زیرا جمع همه اعضا اولیه ثابت است و از روی پرانتز اول میتوانیم جمع پرانتز دوم را مشخص کنیم.)

حالا هر Dp[i][j] اگر true باشد به این معناست که میتوان i عنصر از n+m عنصر را برداشت و به جمع اعداد j رسید.

```cpp
#include <bits/stdc++.h>
using namespace std;

bool dp[100 + 5][100 * 100 + 5];

int main(){
  int N, M;
  while (cin >> N >> M){
    vector<int> ints(N + M + 1);
    int sum = 0;
    for (int i = 1; i <= N + M; ++i){
      cin >> ints[i];
      sum += ints[i];
      // Shift all integers by +50.
      ints[i] += 50;
    }
    memset(dp, false, sizeof(dp));
    dp[0][0] = true;
    // If dp[i][j] is true, that means it is possible to
    // use i out of the N + M integers to sum to j.
    for (int i = 1; i <= N + M; ++i)
      for (int k = min(i, N); k >= 1; --k)
        for (int j = 0; j <= 10000; ++j)
          if (dp[k - 1][j])
            dp[k][j + ints[i]] = true;

    int maximum = -5000;
    int minimum = 5000;
    for (int i = 0; i <= 10000; ++i)
      if (dp[N][i]){
        int nSum = i - 50 * N;
        maximum = max(maximum, nSum * (sum - nSum));
        minimum = min(minimum, nSum * (sum - nSum));
      }
    cout << maximum << " " << minimum << endl;
  }
}
```

سوال ششم: ماشین بوق جمع کن

با توجه به اینکه باید تمام بوق ها را جمع کنیم و کوتاهترین دور را بزنیم، میتوانیم مسئله را با travelling salesman حل کنیم.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Collecting Beepers
int n;
pair<int, int> points[13];
int graph[13][13];
int tsp[13][(1 << 13) + 3];

int ctsp(int node, int mask){
   if (tsp[node][mask] != -1)
      return tsp[node][mask];
   if (1 << (node - 1) == mask)
      return tsp[node][mask] = graph[0][node];
   int ans = INT_MAX;
   for (int j = 1; j < n; j++)
      if (j != node && (mask & (1 << (j - 1))))
         ans = min(ans, ctsp(j, mask - (1 << (node - 1))) + graph[j][node]);
   return tsp[node][mask] = ans;
}

int main()
{
   int t, x, y;
   cin >> t;
   while (t--){
      cin >> x >> y;
      cin >> points[0].first >> points[0].second;
      cin >> n;
      n++;
      for (int i = 1; i < n; i++)
         cin >> points[i].first >> points[i].second;
      for (int i = 0; i < n; i++)
         for (int j = i; j < n; j++){
            int dis = abs(points[i].first - points[j].first)
            + abs(points[i].second - points[j].second);
            graph[i][j] = dis;
            graph[j][i] = dis;
         }
      for (int i = 0; i < 13; i++)
         for (int j = 0; j < ((1 << 13) + 3); j++)
            tsp[i][j] = -1;
      int ans = INT_MAX;
      for (int i = 1; i < n; i++)
         ans = min(ans, ctsp(i, ((1 << (n - 1)) - 1)) + graph[i][0]);
      cout << ans << endl;
   }
}
```

سوال هفتم: Lego!

برای حل این سوال لازم است توجه کنیم که مسئله دارای overlapping subproblems است. به همین منظور جدولی را تعیین میکنیم که نمایانگر روش های ساخت یک رشته به طول x است. گر تعداد روش های ساخت رشته های به طول ۱, ۲, ۳ , ... x را داشته باشیم، برای بدست اوردن تعداد روش های ساخت رشته به طول x، کافی است که (در صورت امکان) بررسی کنیم که اگر رشته حک شده روی هر لگو (k) را از رشته x کم کنیم، به چند حالت میتوان رشته باقیمانده را ساخت. اگر رشته باقیمانده را بتوان به dp[i - len(k)]=c حالت ساخت، باید dp[x] += dp[i - len(k)] * count(k) بشود.(count[k] برابر است با تعداد موجود از لگو هایی که رشته k روی ان ها حک شده اند). پس از اجرای این عملیات برای همه لگو ها، مقدار dp[x] به درستی محاسبه شده است و میتوانیم dp[x+1] را محاسبه کنیم.

```cpp
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
int main() {
    int n;
    cin >> n;
    string ans;
    cin >> ans;
    map<string, int> mp;

    for (int i = 0; i < n; ++i) {
        string a;
        int t;
        cin >> a >> t;
        mp[a] = t;
    }

    ll dp[ans.length() + 100];
    memset(dp, 0, sizeof(dp));
    dp[0] = 1;
    for (int i = 1; i <= ans.length(); ++i) {
        string temp;
        for (int j = i; j >= 1; --j) {
            temp.push_back(ans[j - 1]);

            string tt;
            for (int k = temp.length() - 1; k >= 0; --k) {
                tt.push_back(temp[k]);
            }

            if (mp.count(tt) > 0) {
                dp[i] += dp[j - 1] * mp[tt];
                dp[i] %= 1000000007;
            }
        }
    }
    cout << dp[ans.length()] << endl;
}
```

سوال هشتم: بلک سوآن

بهترین اســتراتژی برای اضــافه کردن k تا cout در n خط کد: فرض میکنیم که در k-1 تا cout قبلی، کد ران شــده و به مشــکلی نخورده و باگ برنامه با آخرین و kامین cout مشــخص میشــود. حالا بهترین اســتراتژی برای گذاشــتن coutها در n خط کد این اســت که کل n خط کد را به خطهایی [n/(k+1)] تایی تقسیم کنیم و پایان هر [n/(k+1)] خط یک عبارت cout قرار دهیم.
پس به صورت بازگشتی باید [n/(k+1)]امین خط اخر را صدا بزنیم و همین منوال را دوباره طی کنیم.

$$T(1) = 0$$
$$T(n) = \min_{1 \le k \le n} \{ k*p + T(\lceil n/(k+1) \rceil) \} + r$$

```cpp
#include <bits/stdc++.h>
using namespace std;

// Debugging
inline int up(int n, int div){
   if(n % div == 0)
      return n / div;
   return n / div + 1;
}

long long int n, r, p;
map<int, long long int> memo;

long long int dp(int i){
   if(memo.find(i) != memo.end())
      return memo[i];
   long long int ans = LLONG_MAX;
   for(int k = 2; k <= i; k++)
      ans = min(ans, dp(up(i, k)) + (k-1)*p + r);
   return memo[i] = ans;
}

int main(){
   cin >> n >> r >> p;
   memo[1] = 0;
   cout << dp(n) << endl;
}
```