

Apunte ICPC

4 de noviembre de 2017

Índice general

Notas previas	I
0.1. Abreviaciones utilizadas	I
1. Estructuras de datos	1
1.1. Fenwick Tree	1
1.1.1. Actualizaciones por rango, consultas puntales	1
1.1.2. Actualizaciones puntuales, consultas por rango	2
1.2. Union-Find	2
1.3. Segment Tree	3
1.3.1. Iterativo	3
1.3.2. Lazy	4
1.3.3. Pair	6
1.4. Wavelet Tree	8
2. Grafos	11
2.1. DFS	11
2.2. Brexit	11
2.3. Kruskal	12
2.4. Single source shortest path	13
2.5. Edmond Blonsson	13
2.6. Flood Fill	15
2.7. Dijkstra	17
3. Matemática	18
3.1. Pascal	18
3.2. Criba	18
4. Geometria	19
4.1. Intersection	19
4.2. Rectangle union	21
4.3. Closest pair	22
4.4. Radial sweep example	22
4.5. Line sweep example	25

5. Flujo	27
5.1. Problemas de asignación	27
5.1.1. Bipartite matching	27
5.2. Dinic	28
6. Programación dinámica	31
6.1. 0/1 Knapsack	31
6.2. Bitonic Sequence	31
6.3. Box Stacking	32
6.4. Break multiple words with no space into space	35
6.5. Burst Balloon	39
6.6. Catalan	41
6.7. Coin changing	41
6.8. Cutting sticks	42
6.9. Distinct subsequence	43
6.10. Divide and conquer	43
6.11. Edit Distance	45
6.12. Sum 2D Rectangle	47
6.13. Sum Dice	48
7. Contenido adicional	50
7.1. Fast input	50
7.2. Usar en caso de emergencia	50

Notas previas

0.1. Abreviaciones utilizadas

```
1 typedef long long ll;
2 //en ciertos casos es necesario cambiar int por ll
3 typedef vector<int> vi;
4 typedef vector<vector<int> > vvi;
5 typedef pair<int,int> ii;
6 typedef vector<vector<ii> > vvii; //util para grafos
7 typedef pair<pair<int,int>,int> iii;
8 #define mp(x,y) make_pair(x,y)
9 #define pb(x) push_back(x)
```

Capítulo 1

Estructuras de datos

1.1. Fenwick Tree

Nota: Ambas implementaciones tienen rangos entre 1 a n.

1.1.1. Actualizaciones por rango, consultas puntales

```
1 struct FenwickTree{
2     vi FT;
3     FenwickTree(int N){
4         FT.resize(N+1,0);
5     }
6
7     int query(int i){
8         int ans = 0;
9         for(;;i-=i&(-i)) ans += FT[i];
10        return ans;
11    }
12
13    int query(int i, int j){
14        return query(j)-query(i-1);
15    }
16
17    void update(int i, int v){
18        for(;;i<FT.size();i+=i&(-i)) FT[i] += v;
19    }
20
21    void update(int i, int j, int v){
22        update(i,v); update(j+1,-v);
23    }
24 };
```

1.1.2. Actualizaciones puntuales, consultas por rango

La consulta $query(a, b)$ corresponde a la sumatoria de los elementos entre los índices a y b .

```

1 struct FenwickTree {
2     vi ft;
3     FenwickTree(){}
4     FenwickTree(int n){
5         ft.assign(n + 1, 0);
6     }
7
8     int query(int b) {
9         int sum = 0;
10        for (; b; b -= b&(-b)) sum += ft[b];
11        return sum;
12    }
13
14    int query(int a, int b) { \\RSQ
15        return query(b) - (a == 1 ? 0 : query(a - 1));
16    }
17
18    void update(int k, int v) {                                // note: n = ft.
19        size() - 1                                             size() - 1
20        for (; k < (int)ft.size(); k += k&(-k)) ft[k] += v;
21    };

```

1.2. Union-Find

Utilizada para trabajar conjuntos disjuntos. Sirve para encontrar componentes conexas en grafos no dirigidos.

```

1 class UnionFind {
2 private:
3     vi p, rank, setSize;
4     int numSets;
5 public:
6     UnionFind(int N) {
7         setSize.assign(N, 1); numSets = N; rank.assign(N, 0);
8         p.assign(N, 0); for (int i = 0; i < N; i++) p[i] = i; }
9     int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i]
10        )); }
11    bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
12    void unionSet(int i, int j) {
13        if (!isSameSet(i, j)) { numSets--;
14            int x = findSet(i), y = findSet(j);
15            // rank is used to keep the tree short
16            if (rank[x] > rank[y]) { p[y] = x; setSize[x] += setSize[y]; }
17            else { p[x] = y; setSize[y] += setSize[x];
18                if (rank[x] == rank[y]) rank[y]++; } }
19
20    int numDisjointSets() { return numSets; }
21    int sizeOfSet(int i) { return setSize[findSet(i)]; }
22 };

```

1.3. Segment Tree

1.3.1. Iterativo

```

1  struct prodsign {
2      int sgn;
3      prodsign() {sgn = 1;}
4      prodsign(int x) {
5          sgn = (x > 0) - (x < 0);
6      }
7      prodsign(const prodsign &a,
8              const prodsign &b) {
9          sgn = a.sgn*b.sgn;
10     }
11 };
12
13 // Maximum Sum (SPOJ)
14 struct maxsum {
15     int first, second;
16     maxsum() {first = second = -1;}
17     maxsum(int x) {
18         first = x; second = -1;
19     }
20     maxsum(const maxsum &a,
21           const maxsum &b) {
22         if (a.first > b.first) {
23             first = a.first;
24             second = max(a.second,
25                         b.first);
26         } else {
27             first = b.first;
28             second = max(a.first,
29                         b.second);
30         }
31     }
32     int answer() {
33         return first + second;
34     }
35 };
36
37 // Range Minimum Query
38 struct rminq {
39     int value;
40     rminq() {value = INT_MAX;}
41     rminq(int x) {value = x;}
42     rminq(const rminq &a,
43           const rminq &b) {
44         value = min(a.value,
45                     b.value);
46     }
47 };
48
49
50 template<class node> class ST {
51     vector<node> t;
52     int n;
53

```

```

54 public:
55     ST(vector<node> &arr) {
56         n = arr.size();
57         t.resize(n*2);
58         copy(arr.begin(), arr.end(), t.begin() + n);
59         for (int i = n-1; i > 0; --i)
60             t[i] = node(t[i<<1], t[i<<1|1]);
61     }
62
63     // 0-indexed
64     void set_point(int p, const node &value) {
65         for (t[p += n] = value; p > 1; p >>= 1)
66             t[p>>1] = node(t[p], t[p^1]);
67     }
68
69     // inclusive exclusive, 0-indexed
70     node query(int l, int r) {
71         node ans1, ansr;
72         for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
73             if (l&1) ans1 = node(ans1, t[l++]);
74             if (r&1) ansr = node(t[--r], ansr);
75         }
76         return node(ans1, ansr);
77     }
78 };

```

1.3.2. Lazy

```

1  struct RSQ {
2      static intt const neutro = 0;
3      static intt op(intt x, intt y) {
4          return x + y;
5      }
6      static intt
7          lazy_op(int i, int j, intt x) {
8          return (j - i + 1)*x;
9      }
10 };
11
12 struct RMinQ {
13     static intt const neutro = 1e18;
14     static intt op(intt x, intt y) {
15         return min(x, y);
16     }
17     static intt
18         lazy_op(int i, int j, intt x) {
19         return x;
20     }
21 };
22
23
24 template<class t> class SegTreeLazy {
25     vector<intt> arr, st, lazy; int n;
26
27     void build(int u, int i, int j) {
28         if (i == j) {
29             st[u] = arr[i];

```



```

30         return;
31     }
32     int m = (i+j)/2, l = u*2+1, r = u*2+2;
33     build(l, i, m);
34     build(r, m+1, j);
35     st[u] = t::op(st[l], st[r]);
36 }
37
38 void propagate(int u, int i, int j, intt x) {
39     st[u] += t::lazy_op(i, j, x);
40     if (i != j) {
41         lazy[u*2+1] += x;
42         lazy[u*2+2] += x;
43     }
44     lazy[u] = 0;
45 }
46
47 intt query(int a, int b, int u, int i, int j) {
48     if (j < a or b < i)
49         return t::neutro;
50     int m = (i+j)/2, l = u*2+1, r = u*2+2;
51     if (lazy[u])
52         propagate(u, i, j, lazy[u]);
53     if (a <= i and j <= b)
54         return st[u];
55     intt x = query(a, b, l, i, m);
56     intt y = query(a, b, r, m+1, j);
57     return t::op(x, y);
58 }
59
60 void update(int a, int b, intt value,
61 int u, int i, int j) {
62     int m = (i+j)/2, l = u*2+1, r = u*2+2;
63     if (lazy[u])
64         propagate(u, i, j, lazy[u]);
65     if (a <= i and j <= b)
66         propagate(u, i, j, value);
67     else if (j < a or b < i) return; else {
68         update(a, b, value, l, i, m);
69         update(a, b, value, r, m+1, j);
70         st[u] = t::op(st[l], st[r]);
71     }
72 }
73
74 public:
75     SegTreeLazy(vector<intt>& v) {
76         arr = v;
77         n = v.size();
78         st.resize(n*4+5);
79         lazy.assign(n*4+5, 0);
80         build(0, 0, n-1);
81     }
82
83     intt query(int a, int b) {
84         return query(a, b, 0, 0, n-1);
85     }
86

```

```

87     void update(int a, int b, intt value) {
88         update(a, b, value, 0, 0, n-1);
89     }
90 };

```

1.3.3. Pair

```

1  #include <bits/stdc++.h>
2  #define inf 0x7fffffff
3  #define optimizar_io ios_base::sync_with_stdio(0);cin.tie(0);
4
5  using namespace std;
6
7  typedef long long int ll;
8  typedef pair <int,int> par;
9  typedef vector< par > vi;
10 class SegmentTree {
11 public:
12     SegmentTree(const vi&_A){
13         arr = _A; n = (int)arr.size();
14         tree.resize(4*n);
15         lazy.resize(4*n);
16         for(int i = 0; i < 4*n; i++){
17             lazy[i] = 0;
18             tree[i] = make_pair(0,i);
19         }
20         build_tree(1,0,n-1);
21     }
22     par rmq(ll i,ll j) {return query_tree(1,0,n-1,i,j);}
23     void update(ll i,ll j, ll val){update_tree(1,0,n-1,i,j,val);}
24 private:
25     vi arr, tree;
26     vector <long long int > lazy;
27     ll n;
28     void build_tree(ll node, ll a, ll b) {
29         if(a > b) return;
30         if(a == b) {
31             tree[node] = arr[a];
32             return;
33         }
34         build_tree(node*2, a, (a+b)/2);
35         build_tree(node*2+1, 1+(a+b)/2, b);
36         tree[node] = (tree[node*2].first < tree[node*2 +1].first) ?
37             tree[node*2] : tree[node*2+1];
38     }
39     void update_tree(ll node, ll a, ll b, ll i, ll j, ll value) {
40         if(lazy[node] != 0) {
41             tree[node].first += lazy[node]; //+=
42             if(a != b) {
43                 lazy[node*2] += lazy[node]; //+=
44                 lazy[node*2+1] += lazy[node]; //+=
45             }
46             lazy[node] = 0;
47         }
48         if(a > b || a > j || b < i) return;

```

```

50     if(a >= i && b <= j) {
51         tree[node].first += value; //+=
52         if(a != b) {
53             lazy[node*2] += value; //+=
54             lazy[node*2+1] += value; //+=
55         }
56         return;
57     }
58     update_tree(node*2, a, (a+b)/2, i, j, value);
59     update_tree(1+node*2, 1+(a+b)/2, b, i, j, value);
60     tree[node] = (tree[node*2].first < tree[node*2+1].first) ? tree[
        node*2] : tree[node*2+1];
61 }
62
63 par query_tree(ll node, ll a, ll b, ll i, ll j) {
64     //Cuidado con -inf que est como entero en caso de usar long
        long
65     if(a > b || a > j || b < i) return make_pair ((ll)inf,-1); //(-
        inf,max) (inf,min)
66     if(lazy[node] != 0){
67         tree[node].first += lazy[node]; //+=
68         if(a != b) {
69             lazy[node*2] += lazy[node]; //+=
70             lazy[node*2+1] += lazy[node]; //+=
71         }
72         lazy[node] = 0;
73     }
74
75     if(a >= i && b <= j) return tree[node];
76     par q1 = query_tree(node*2, a, (a+b)/2, i, j);
77     par q2 = query_tree(1+node*2, 1+(a+b)/2, b, i, j);
78     par res = (q1.first < q2.first) ? q1 : q2;
79     return res;
80 }
81 };
82
83 int main() {
84     //optimizar_io
85     int a,b,c,d,aux,aux2;
86     scanf("%d_%d_%d_%d",&a,&b,&c,&d);
87     vi conexo,disconexo;
88     conexo.resize(a);
89     disconexo.resize(a);
90     vector <vector <int> > adj;
91     adj.resize(a);
92     for(int i=0;i<b;i++){
93         cin>>aux >> aux2;
94         adj[aux-1].push_back(aux2-1);
95         adj[aux2-1].push_back(aux-1);
96     }
97     for(int i = 0; i < a; i++){
98         conexo[i] = make_pair(adj[i].size(),i);
99         disconexo[i] = make_pair(a-1-adj[i].size(),i);
100 }
101 SegmentTree amigo(conexo);
102 SegmentTree enemigo(disconexo);
103 vector <bool> respuesta;

```

```

104     respuesta.resize(a,true);
105     par temp;
106     while(1){
107         temp = amigo.rm(0,a-1);
108         if(temp.first >= c || c == 0){
109             temp = enemigo.rm(0,a-1);
110             if(temp.first >= d || d == 0)break;
111         }
112         respuesta[temp.second] = false;
113         for(int i = 0; i < adj[temp.second].size() ; i++){
114             amigo.update(adj[temp.second][i],adj[temp.second][i],-1);
115             enemigo.update(adj[temp.second][i],adj[temp.second][i],+1);
116         }
117         enemigo.update(0,a-1,-1);
118         amigo.update(temp.second,temp.second,50000000);
119         enemigo.update(temp.second,temp.second,50000000);
120     }
121     ll imprimir = 0;
122     for(int i = 0; i < a; i++)if(respuesta[i])imprimir++;
123     printf("%lld\n",imprimir);
124     return 0;
125 }

```

1.4. Wavelet Tree

```

1  typedef vector<int>::iterator iter;
2
3  class WaveTree {
4      vector<vector<int>> r0; int n, s;
5      vector<int> arrCopy;
6
7      void build(iter b, iter e, int l, int r, int u) {
8          if (l == r)
9              return;
10         int m = (l+r)/2;
11         r0[u].reserve(e-b+1); r0[u].push_back(0);
12         for (iter it = b; it != e; ++it)
13             r0[u].push_back(r0[u].back() + (*it<=m));
14         iter p = stable_partition(b, e, [=](int i){
15             return i<=m;});
16         build(b, p, l, m, u*2);
17         build(p, e, m+1, r, u*2+1);
18     }
19
20     int q, w;
21     int range(int a, int b, int l, int r, int u) {
22         if (r < q or w < l)
23             return 0;
24         if (q <= l and r <= w)
25             return b-a;
26         int m = (l+r)/2, za = r0[u][a], zb = r0[u][b];
27         return range(za, zb, l, m, u*2) +
28             range(a-za, b-zb, m+1, r, u*2+1);
29     }
30
31 public:
32     //arr[i] in [0,sigma)

```

```

33 WaveTree(vector<int> arr, int sigma) {
34     n = arr.size(); s = sigma;
35     r0.resize(s*2); arrCopy = arr;
36     build(arr.begin(), arr.end(), 0, s-1, 1);
37 }
38
39 //k in [1,n], [a,b) is 0-indexed, -1 if error
40 int quantile(int k, int a, int b) {
41     //extra conditions disabled
42     if (/*a < 0 or b > n or*/ k < 1 or k > b-a)
43         return -1;
44     int l = 0, r = s-1, u = 1, m, za, zb;
45     while (l != r) {
46         m = (l+r)/2;
47         za = r0[u][a]; zb = r0[u][b]; u*=2;
48         if (k <= zb-za)
49             a = za, b = zb, r = m;
50         else
51             k -= zb-za, a -= za, b -= zb,
52             l = m+1, ++u;
53     }
54     return r;
55 }
56
57 //counts numbers in [x,y] in positions [a,b)
58 int range(int x, int y, int a, int b) {
59     if (y < x or b <= a)
60         return 0;
61     q = x; w = y;
62     return range(a, b, 0, s-1, 1);
63 }
64
65 //count occurrences of x in positions [0,k)
66 int rank(int x, int k) {
67     int l = 0, r = s-1, u = 1, m, z;
68     while (l != r) {
69         m = (l+r)/2;
70         z = r0[u][k]; u*=2;
71         if (x <= m)
72             k = z, r = m;
73         else
74             k -= z, l = m+1, ++u;
75     }
76     return k;
77 }
78
79 //x in [0,sigma)
80 void push_back(int x) {
81     int l = 0, r = s-1, u = 1, m, p; ++n;
82     while (l != r) {
83         m = (l+r)/2;
84         p = (x<=m);
85         r0[u].push_back(r0[u].back() + p);
86         u*=2; if (p) r = m; else l = m+1, ++u;
87     }
88 }
89

```

```

90      //doesn't check if empty
91      void pop_back() {
92          int l = 0, r = s-1, u = 1, m, p, k; --n;
93          while (l != r) {
94              m = (l+r)/2; k = r0[u].size();
95              p = r0[u][k-1] - r0[u][k-2];
96              r0[u].pop_back();
97              u*=2; if (p) r = m; else l = m+1, ++u;
98          }
99      }
100
101      //swap arr[i] with arr[i+1], i in [0,n-1)
102      void swap_adj(int i) {
103          int &x = arrCopy[i], &y = arrCopy[i+1];
104          int l = 0, r = s-1, u = 1;
105          while (l != r) {
106              int m = (l+r)/2, p = (x<=m), q = (y<=m);
107              if (p != q) {
108                  r0[u][i+1] ^= r0[u][i] ^ r0[u][i+2];
109                  break;
110              }
111              u*=2; if (p) r = m; else l = m+1, ++u;
112          }
113          swap(x, y);
114      }
115  };

```

Capítulo 2

Grafos

2.1. DFS

```
1 void graphCheck(int u) {                                // DFS for checking graph
    edge properties
2     dfs_num[u] = DFS_GRAY;    // color this as DFS_GRAY (temp) instead
    of DFS_BLACK
3     for (int j = 0; j < (int)AdjList[u].size(); j++) {
4         ii v = AdjList[u][j];                            // weighted graph
5         if (dfs_num[v.first] == DFS_WHITE) {              // Tree Edge, DFS_GRAY
            to DFS_WHITE
6             dfs_parent[v.first] = u;                      // parent of this
            children is me
7             graphCheck(v.first);
8         }
9         else if (dfs_num[v.first] == DFS_GRAY) {          //
            DFS_GRAY to DFS_GRAY
10            if (v.first == dfs_parent[u])                  // to differentiate
                these two cases
11                printf("Bidirectional (%d, %d) - (%d, %d)\n", u, v.first,
                    v.first, u);
12            else // the most frequent application: check if the given
                graph is cyclic
13                printf("Back Edge (%d, %d) (Cycle)\n", u, v.first);
14        }
15        else if (dfs_num[v.first] == DFS_BLACK)           // DFS_GRAY
            to DFS_BLACK
16            printf("Forward/Cross Edge (%d, %d)\n", u, v.first);
17    }
18    dfs_num[u] = DFS_BLACK;    // after recursion, color this as
    DFS_BLACK (DONE)
19    topoSort.push_back(u);
20 }
```

2.2. Brexit

```
1 int main(){
2     int c,p,x,l;
```

```

3  cin>>c>>p>>x>>l;
4  int u,v;
5  vector<vector<int> > g(c,vector<int>());
6  for(int i=0;i<p;i++){
7      cin>>u>>v;
8      g[u-1].push_back(v-1);
9      g[v-1].push_back(u-1);
10 }
11 vector<int> d(c,0);
12 vector<int> ori(c,0);
13 for(int i=0;i<c;i++){
14     d[i]=ori[i]=g[i].size();
15 }
16 x--;
17 l--;
18 vector<bool> vivo(c,true);
19 vivo[l]=false;
20 queue<int> q;
21 q.push(l);
22 while(!q.empty()){
23     int nodo=q.front();
24     q.pop();
25     vivo[nodo]=false;
26     for(int i=0;i<g[nodo].size();i++){
27         int next=g[nodo][i];
28         if(vivo[next]){
29             d[next]--;
30             if(d[next]==ori[next]/2){
31                 q.push(next);
32             }
33         }
34     }
35 }
36 if(vivo[x]){
37     puts("stay");
38 }else{
39     puts("leave");
40 }
41 return 0;
42 }

```

2.3. Kruskal

```

1  vector< pair<int, ii> > EdgeList;    // (weight, two vertices) of
    the edge
2  for (int i = 0; i < E; i++) {
3      scanf("%d_%d_%d", &u, &v, &w);    // read the triple: (u,
        v, w)
4      EdgeList.push_back(make_pair(w, ii(u, v)));    // (w,
        u, v)
5      AdjList[u].push_back(ii(v, w));
6      AdjList[v].push_back(ii(u, w));
7  }
8  sort(EdgeList.begin(), EdgeList.end()); // sort by edge weight O(E
    log E)
9
    // note: pair object has built-in comparison
    function

```



```

10
11 int mst_cost = 0;
12 UnionFind UF(V);                                // all V are disjoint sets
    initially
13 for (int i = 0; i < E; i++) {                    // for each edge
    , O(E)
14     pair<int, int> front = EdgeList[i];
15     if (!UF.isSameSet(front.second.first, front.second.second)) { //
        check
16         mst_cost += front.first;                  // add the weight of e
            to MST
17         UF.unionSet(front.second.first, front.second.second); //
            link them
18 } }                                                // note: the runtime cost of UFDS is very
    light
19
20 // note: the number of disjoint sets must eventually be 1 for a
    valid MST
21 printf("MST cost = %d (Kruskal's)\n", mst_cost);

```

2.4. Single source shortest path

2.5. Edmond Blonsson

```

1  /*
2  GETS:
3  V->number of vertices
4  E->number of edges
5  pair of vertices as edges (vertices are 1..V)
6
7  GIVES:
8  output of edmonds() is the maximum matching
9  match[i] is matched pair of i (-1 if there isn't a matched pair)
10 */
11
12 // RECORDAR SETEAR LA VARIABLE GLOBAL V Y EL VECTOR DE VISITADOS
    ED O SINO NO FUNCIONA!!
13 #include <bits/stdc++.h>
14 using namespace std;
15 const int M=500;
16 struct struct_edge{int v;struct_edge* n;};
17 typedef struct_edge* edge;
18 struct_edge pool[M*M*2];
19 edge top=pool,adj[M];
20 int V,E,match[M],qh,qt,q[M],father[M],base[M];
21 bool inq[M],inb[M],ed[M][M];
22 void add_edge(int u,int v){
23     top->v=v,top->n=adj[u],adj[u]=top++;
24     top->v=u,top->n=adj[v],adj[v]=top++;
25 }
26 int LCA(int root,int u,int v){
27     static bool inp[M];
28     memset(inp,0,sizeof(inp));
29     while(1)
30     {
31         inp[u=base[u]]=true;

```

```

32         if (u==root) break;
33         u=father[match[u]];
34     }
35     while(1)
36     {
37         if (inp[v=base[v]]) return v;
38         else v=father[match[v]];
39     }
40 }
41 void mark_blossom(int lca,int u){
42     while (base[u]!=lca)
43     {
44         int v=match[u];
45         inb[base[u]]=inb[base[v]]=true;
46         u=father[v];
47         if (base[u]!=lca) father[u]=v;
48     }
49 }
50 void blossom_contraction(int s,int u,int v){
51     int lca=LCA(s,u,v);
52     memset(inb,0,sizeof(inb));
53     mark_blossom(lca,u);
54     mark_blossom(lca,v);
55     if (base[u]!=lca)
56         father[u]=v;
57     if (base[v]!=lca)
58         father[v]=u;
59     for (int u=0;u<V;u++)
60         if (inb[base[u]])
61         {
62             base[u]=lca;
63             if (!inq[u])
64                 inq[q[++qt]=u]=true;
65         }
66 }
67 int find_augmenting_path(int s){
68     memset(inq,0,sizeof(inq));
69     memset(father,-1,sizeof(father));
70     for (int i=0;i<V;i++) base[i]=i;
71     inq[q[qh=qt=0]=s]=true;
72     while (qh<=qt)
73     {
74         int u=q[qh++];
75         for (edge e=adj[u];e=e->n)
76         {
77             int v=e->v;
78             if (base[u]!=base[v]&&match[u]!=v)
79                 if ((v==s)|| (match[v]!=-1 && father[match[v]]!=-1))
80                     blossom_contraction(s,u,v);
81                 else if (father[v]==-1)
82                 {
83                     father[v]=u;
84                     if (match[v]==-1)
85                         return v;
86                     else if (!inq[match[v]])
87                         inq[q[++qt]=match[v]]=true;
88                 }

```

```

89     }
90 }
91 return -1;
92 }
93 int augment_path(int s,int t){
94     int u=t,v,w;
95     while (u!=-1)
96     {
97         v=father[u];
98         w=match[v];
99         match[v]=u;
100        match[u]=v;
101        u=w;
102    }
103    return t!=-1;
104 }
105 int edmonds(){
106     int matchc=0;
107     memset(match,-1,sizeof(match));
108     for (int u=0;u<V;u++){
109         if (match[u]==-1)
110             matchc+=augment_path(u,find_augmenting_path(u));
111     }
112     return matchc;
113 }
114 int main(){
115     int n,m;
116     scanf("%d %d",&n,&m);
117     string s;
118     vector < vector <string> > jueces(n);
119     vector < pair <int,int> > parejas;
120     int distintos = 0;
121     map <string,int> M;
122     int contador = 0;
123     for(int i = 0; i < n; i++){
124         for(int j = 0; j < m; j++){
125             cin >> s;
126             if(M.count(s) == 0){
127                 distintos++;
128                 // cout << s << endl;
129                 M[s] = contador;
130                 contador++;
131             }
132             jueces[i].push_back(s);
133         }
134     }
135     //printf("distintos = %d\n",distintos);
136     vector <vector <bool> > adj(distintos,vector <bool>(distintos,
137         true));
138     for(int i = 0; i < n; i++){
139         for(int j = 0; j < m; j++){
140             for(int k = j+1; k < m; k++){
141                 adj[M[jueces[i][j]]][M[jueces[i][k]]] = false;
142                 adj[M[jueces[i][k]]][M[jueces[i][j]]] = false;
143             }
144         }
145     }
146     V = distintos;

```

```

145     for(int i = 0; i < adj.size();i++){
146         for(int j = i+1; j < adj[i].size();j++){
147             if(adj[i][j] && !ed[i][j]){
148                 add_edge(i,j);
149                 ed[i][j]=ed[j][i]=true;
150             }
151         }
152     }
153     int final =edmonds();
154     if(final >= abs(m-distintos))puts("S");
155     else puts("N");
156     return 0;
157 }

```

2.6. Flood Fill

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  int n,m;
6  int mapa[2020][2020];
7  int total;
8  int w;
9  int sumar=507;
10 int xx[]={0,0,1,-1};
11 int yy[]={1,-1,0,0};
12 #define fix(x) (x)=2*((x)+sumar)
13 typedef pair<int,int> ii;
14 inline bool check(int x,int y){
15     return x>=0 && x<2020 && y>=0 && y<2020;
16 }
17
18 void fill(int x,int y){
19     total--;
20     queue<ii> q;
21     q.push(ii(x,y));
22     mapa[x][y]=-1;
23     while(!q.empty()){
24         x=q.front().first;
25         y=q.front().second;
26
27         q.pop();
28         for(int i=0;i<4;i++){
29             //printf("ini: %d %d\n",x+xx[i],y+yy[i]);
30             if(check(x+xx[i],y+yy[i]) && mapa[x+xx[i]][y+yy[i]]<w){
31                 //printf("a: %d %d\n",x+xx[i],y+yy[i]);
32                 //printf("%d\n",mapa[x+2*xx[i]][y+2*yy[i]]);
33                 if(check(x+2*xx[i],y+2*yy[i])&&mapa[x+2*xx[i]][y+2*yy[i]]
34                     ]!=-1){
35                     //printf("b: %d %d\n",x+2*xx[i],y+2*yy[i]);
36                     mapa[x+2*xx[i]][y+2*yy[i]]=-1;
37                     //puts("a");
38                     total++;
39                     q.push(ii(x+2*xx[i],y+2*yy[i]));
40                     //printf("%d %d\n",q.front().first,q.front().second);
41                 }
42             }
43         }
44     }
45 }

```

```

41
42     }
43
44     }
45 }
46 //puts("fin");
47 }
48
49 int main(){
50     int x1,y1,x2,y2,h,minX,maxX,minY,maxY;
51     while(scanf("%d",&n)){
52         total=0;
53
54         if(n==0){
55             break;
56         }
57         for(int i=0;i<2020;i++){
58             for(int j=0;j<2020;j++){
59                 mapa[i][j]=0;
60             }
61         }
62         minX=minY=10000;
63         while(n--){
64             scanf("%d_%d_%d_%d",&x1,&y1,&x2,&y2,&h);
65             fix(x1);
66             fix(y1);
67             fix(x2);
68             fix(y2);
69             if(x1==x2){
70                 for(int i=min(y1,y2);i<=max(y1,y2);i++){
71                     mapa[x1][i]=h;
72                 }
73             }else{
74                 for(int i=min(x1,x2);i<=max(x1,x2);i++){
75                     mapa[i][ y1]=h;
76                 }
77             }
78         }
79     }
80
81     scanf("%d",&w);
82     fill(1,1);
83
84     printf("%d\n",1010*1010-total-2);
85
86 }
87 return 0;
88 }

```

2.7. Dijkstra

Utilizamos la representacion vvii con pares (vecino,peso)

```
1 asd
```

Capítulo 3

Matemática

3.1. Pascal

```
1 for(int i=1;i<150;i++){
2     for(int j=1;j<i+3;j++){
3         pascal[i][j]=pascal[i-1][j-1]+pascal[i-1][j];
4     }
5 }
6 cin>>n;
7 for(ll i=0;i<n+1;i++){
8     printf("%lld□",pascal[n-1][i]);
9 }
```

3.2. Criba

```
1 vector<bool> isprime;
2 vector<int> primes;
3 void sieve(int n) {
4     isprime.assign(n + 1,true);
5
6     isprime[1] = false; isprime[2] = true;
7     for (int i = 2; i <= n; i++) {
8         if (isprime[i]) {
9             for (int j = i*i; j < n; j+=i) {
10                 isprime[j] = false;
11             }
12         }
13     }
14     for (int i = 2; i < n; i++) {
15         if (isprime[i]) {
16             primes.push_back(i);
17         }
18     }
19     return;
20 }
```

Capítulo 4

Geometria

4.1. Intersection

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 template< class T > bool inside(T a, T b, T c) { return a<=b && b<=
   c; }
4 typedef vector<int> vi;
5
6 class UnionFind{
7 private:
8     vi p, rank;
9 public:
10     UnionFind(int N){
11         rank.assign(N,0);
12         p.assign(N,0);
13         for(int i=0;i<N;i++){
14             p[i]=i;
15         }
16     }
17     int findSet(int i){
18         return (p[i]==i) ? i : (p[i] = findSet(p[i]));
19     }
20     bool isSameSet(int i, int j){
21         return findSet(i) == findSet(j);
22     }
23     void unionSet(int i,int j){
24         if(!isSameSet(i,j)){
25             int x = findSet(i), y = findSet(j);
26             if(rank[x]>rank[i]) p[y]=x;
27             else{
28                 p[x]=y;
29                 if(rank[x]==rank[y]) rank[y]++;
30             }
31         }
32     }
33     void imprimir(){
34         for(int i=0;i<p.size();i++) printf("%d_",p[i]);
35         puts("");
```

```

36     }
37 };
38
39 const int MAX = 1024;
40
41 struct Point { int x, y; };
42 struct Segment { Point a, b; };
43
44 Segment seg[MAX];
45 inline int direction(Point &pi, Point &pj, Point &pk) {
46     return (pk.x-pi.x)*(pj.y-pi.y)-(pj.x-pi.x)*(pk.y-pi.y);
47 }
48
49 inline bool onsegment(Point &pi, Point &pj, Point &pk) {
50     return (inside(min(pi.x,pj.x),pk.x,max(pi.x,pj.x)) && inside(min(
        pi.y,pj.y),pk.y,max(pi.y,pj.y)));
51 }
52
53 inline bool intersect(Point &p1, Point &p2, Point &p3, Point &p4) {
54     int d1, d2, d3, d4;
55     d1 = direction(p3, p4, p1);
56     d2 = direction(p3, p4, p2);
57     d3 = direction(p1, p2, p3);
58     d4 = direction(p1, p2, p4);
59     if(((d1>0 && d2<0)||(d1<0 && d2>0)) && ((d3>0 && d4<0)||(d3<0 &&
        d4>0))) return true;
60     if(!d1 && onsegment(p3, p4, p1)) return true;
61     if(!d2 && onsegment(p3, p4, p2)) return true;
62     if(!d3 && onsegment(p1, p2, p3)) return true;
63     if(!d4 && onsegment(p1, p2, p4)) return true;
64     return false;
65 }
66
67
68 int main(){
69     int t;
70     cin>>t;
71     int n,m;
72     while(t--){
73         Point p[n];
74         Point q[n];
75         cin>>n>>m;
76         for(int i=0;i<n;i++){
77             int a,b,c,d;
78             cin>>a>>b>>c>>d;
79             seg[i].a.x = a;
80             seg[i].a.y = b;
81             seg[i].b.x = c;
82             seg[i].b.y = d;
83         }
84         UnionFind uf(n);
85         for(int i=0;i<n;i++){
86             for(int j=i+1;j<n;j++){
87                 if( intersect(seg[i].a, seg[i].b, seg[j].a, seg[j].b)){
88                     uf.unionSet(i,j);
89                 }
90             }

```



```

91     }
92     // uf.imprimir();
93     for(int i=0;i<m;i++){
94         int a,b;
95         cin>>a>>b;
96         if(uf.isSameSet(a-1,b-1)) puts("YES");
97         else puts("NO");
98     }
99 }
100 return 0;
101 }

```

4.2. Rectangle union

```

1  #include <cstdio>
2  #include <algorithm>
3  using namespace std;
4  struct event {
5      int ind; // Index of rectangle in rects
6      bool type; // Type of event: 0 = Lower-left ; 1 = Upper-right
7      event() {} ;
8      event(int ind, int type) : ind(ind), type(type) {} ;
9  };
10 struct point {
11     int x, y;
12 };
13 int n, e; // n = number of rectangles; e = number of edges
14 point rects [1000][2]; // Each rectangle consists of 2 points: [0]
    = lower-left ; [1] = upper-right
15 event events_v [2000]; // Events of horizontal sweep line
16 event events_h [2000]; // Events of vertical sweep line
17 bool compare_x(event a, event b) { return rects[a.ind][a.type].x<
    rects[b.ind][b.type].x; }
18 bool compare_y(event a, event b) { return rects[a.ind][a.type].y<
    rects[b.ind][b.type].y; }
19 bool in_set [10000]; // Boolean array in place of balanced binary
    tree (set)
20 long long area; // The output: Area of the union
21 int main() { /// x -> v; y -> h
22     scanf("%d", &n);
23     for (int i=0;i<n;++i) {
24         scanf("%d_%d_%d_%d", &rects[i][0].x, &rects[i][0].y, // Lower-
            left coordinate
25             &rects[i][1].x, &rects[i][1].y); // Upper-right coordinate
26         events_v[e] = event(i, 0);
27         events_h[e++] = event(i, 0);
28         events_v[e] = event(i, 1);
29         events_h[e++] = event(i, 1);
30     }
31     sort(events_v, events_v+e, compare_x);
32     sort(events_h, events_h+e, compare_y); // Pre-sort set of
        horizontal edges
33     in_set[events_v[0].ind] = 1;
34     for (int i=1;i<e;++i) { // Vertical sweep line
35         event c = events_v[i];
36         int cnt = 0; // Counter to indicate how many rectangles are
            currently overlapping

```

```

37      // Delta_x: Distance between current sweep line and previous
           sweep line
38      int delta_x = rects[c.ind][c.type].x - rects[events_v[i-1].ind
           ][events_v[i-1].type].x;
39      int begin_y;
40      if (delta_x==0) continue;
41      for (int j=0;j<e;++j) if (in_set[events_h[j].ind]==1) { //
           Horizontal sweep line
42          if (events_h[j].type==0) {
43              if (cnt==0) begin_y = rects[events_h[j].ind][0].y; // Block
           starts
44              ++cnt;
45          } else {
46              --cnt;
47              if (cnt==0) { // Block ends
48                  int delta_y = (rects[events_h[j].ind][1].y-begin_y);
49                  area+=delta_x * delta_y;
50              }
51          }
52      }
53      in_set[c.ind] = (c.type==0);
54  }
55  printf("%lld\n", area);
56  return 0;
57  }

```

4.3. Closest pair

```

1  #define px second
2  #define py first
3  typedef pair<ll,ll> pairll;
4  int n;
5  pairll pnts[100000];
6  set<pairll> box;
7  double best;
8  int compx(pairll a,pairll b){
9      return a.px<b.px;
10 }
11 int main(){
12     scanf("%d",&n);
13     for(int i=0;i<n;i++){
14         scanf("%lld_%lld",&pnts[i].px,&pnts[i].py);
15     }
16     sort(pnts,pnts+n,compx);
17     best=1000000000000;
18     box.insert(pnts[0]);
19     int left=0;
20     for(int i=1;i<n;i++){
21         while(left<i && pnts[i].px-pnts[left].px > best) box.erase(pnts
           [left++]);
22         for(typeof(box.begin()) it=box.lower_bound(make_pair(pnts[i].py
           -best, pnts[i].px-best));
23         it!=box.end() && pnts[i].py+best>=it->py; it++){
24             best = min(best, sqrt(pow(pnts[i].py - it->py, 2.0)+pow(
           pnts[i].px - it->px, 2.0)));
25         }
26         box.insert(pnts[i]);

```

```

27     printf("%.2f\n",best);
28 }
29 }

```

4.4. Radial sweep example

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const double eps = 1e-10, PI = acos(-1.0);
5
6  inline int sgn(double x) {
7      if (fabs(x) <= eps) return 0;
8      else if (x > eps) return 1;
9      else return -1;
10 }
11
12 struct Point {
13     double x, y, ang;
14     Point() : x(0), y(0) {}
15     Point(double a, double b) : x(a), y(b) {
16         ang = atan2(y, x);
17     }
18     Point operator + (const Point &rhs) const {
19         return Point(x + rhs.x, y + rhs.y);
20     }
21     Point operator - (const Point &rhs) const {
22         return Point(x - rhs.x, y - rhs.y);
23     }
24     Point operator * (double k) const {
25         return Point(x * k, y * k);
26     }
27     Point operator / (double k) const {
28         return Point(x / k, y / k);
29     }
30     double dot(const Point &rhs) const {
31         return x * rhs.x + y * rhs.y;
32     }
33     double det(const Point &rhs) const {
34         return x * rhs.y - y * rhs.x;
35     }
36     double abs() const {
37         return hypot(x, y);
38     }
39     void read() {
40         scanf("%lf%lf", &x, &y);
41     }
42 } 0;
43
44 double nowAng;
45
46 Point inter(Point A, Point B, Point C, Point D) {
47     return A + (B - A) * ((D - C).det(C - A) / (D - C).det(B - A));
48 }
49
50 struct Line {
51     Point A, B;

```

```

52     Line() {}
53     Line(Point a, Point b) : A(a), B(b) {}
54     double dis() const {
55         if (sgn((O - A).det(O - B)) == 0) return min((O - A).abs(),
56             (O - B).abs());
57         return (O - inter(A, B, O, Point(cos(nowAng), sin(nowAng))))
58             .abs();
59     }
60     bool operator < (const Line &rhs) const {
61         return sgn(dis() - rhs.dis()) < 0;
62     }
63 };
64
65 struct Event {
66     double ang;
67     int id, type;
68     Event() : ang(0), id(0), type(0) {}
69     Event(double a, int b, int c) : ang(a), id(b), type(c) {}
70     bool operator < (const Event &rhs) const {
71         if (sgn(ang - rhs.ang) != 0) return sgn(ang - rhs.ang) < 0;
72         return type < rhs.type;
73     }
74 };
75
76 const int MAXN = 30000 + 10;
77
78 Point P[MAXN];
79 Line L[MAXN];
80 int S, N, M;
81
82 vector<Event> E;
83 set<Line> Seg;
84 set<Line>::iterator its[MAXN];
85
86 double fix(double x) {
87     if (x < 0) x += PI * 2;
88     if (x >= PI * 2) x -= PI * 2;
89     return x;
90 }
91
92 int gao(int id) {
93     int ret = 0;
94     E.clear();
95     for (int i = 0; i < N; ++ i) {
96         if (i == id) continue;
97         Point tmp = P[i] - P[id];
98         E.push_back(Event(tmp.ang, i, 1));
99     }
100     for (int i = 0; i < M; ++ i) {
101         Point A = L[i].A - P[id];
102         Point B = L[i].B - P[id];
103         double delta = fix(B.ang - A.ang);
104         if (sgn(delta - PI) > 0) swap(A, B);
105         if (sgn(A.ang - B.ang) > 0) {
106             E.push_back(Event(A.ang, i, 0));
107             E.push_back(Event(PI, i, 2));
108             E.push_back(Event(-PI, i, 0));

```

```

107         E.push_back(Event(B.ang, i, 2));
108     }
109     else {
110         E.push_back(Event(A.ang, i, 0));
111         E.push_back(Event(B.ang, i, 2));
112     }
113 }
114 sort(E.begin(), E.end());
115 Seg.clear();
116 for (int i = 0; i < (int)E.size(); ++ i) {
117     int nowID = E[i].id;
118     nowAng = E[i].ang;
119     if (E[i].type == 0) {
120         its[nowID] = Seg.insert(Line(L[nowID].A - P[id], L[
            nowID].B - P[id])).first;
121     }
122     else if (E[i].type == 1) {
123         ret += (Seg.empty() || sgn(Seg.begin()->dis() - (P[id]
            - P[nowID]).abs()) > 0);
124     }
125     else if (E[i].type == 2) {
126         Seg.erase(its[nowID]);
127     }
128 }
129 return ret;
130 }
131
132 int main() {
133     O = Point(0, 0);
134     while (scanf("%d%d%d", &S, &N, &M) == 3) {
135         for (int i = 0; i < N; ++ i) P[i].read();
136         for (int i = 0; i < M; ++ i) {
137             L[i].A.read();
138             L[i].B.read();
139         }
140         for (int i = 0; i < S; ++ i) {
141             printf("%d\n", gao(i));
142         }
143     }
144     return 0;
145 }

```

4.5. Line sweep example

```

1 struct point{
2     int x,y,valor;
3 };
4
5 bool comp1(const point &lhs,const point &rhs){
6     return lhs.y<rhs.y;
7 }
8
9 bool comp2(const point &lhs,const point &rhs){
10     return (lhs.x==rhs.x?lhs.y<rhs.y:lhs.x<rhs.x);
11 }
12
13 point poly[200010];

```

```

14
15 int main(){
16     int p,v;
17     while(scanf("%d %d",&p,&v)!=EOF){
18         for(int i=0;i<p;i++){
19             scanf("%d %d",&poly[i].x,&poly[i].y);
20             poly[i].valor=i+1;
21         }
22         for(int i=p;i<p+v;i++){
23             scanf("%d %d",&poly[i].x,&poly[i].y);
24             poly[i].valor=-1;
25         }
26
27         sort(poly,poly+p+v,comp1);           //orden por y;
28
29         int original=poly[0].y;                //compresion de puntos por y
30         int comprimido=1;
31         poly[0].y=comprimido;
32         for(int i=1;i<p+v;i++){
33             if(poly[i].y==original){
34                 poly[i].y=comprimido;
35             }else{
36                 original=poly[i].y;
37                 comprimido++;
38                 poly[i].y=comprimido;
39             }
40         }
41         FenwickTree FT(800010);
42         sort(poly,poly+p+v,comp2);           //orden por x
43         int perdido=0;
44         for(int i=0;i<p+v;i++){
45             if(poly[i].valor==-1){
46                 FT.update(poly[i].y,poly[i+1].y-1,1); //los vertices
47                 //siempre van de a pares
48                 i++;
49             }else{
50                 if(FT.query(poly[i].y)%2==0){
51                     perdido+=poly[i].valor;
52                 }
53             }
54             printf("%lld\n",perdido);
55         }
56         return 0;
57     }

```

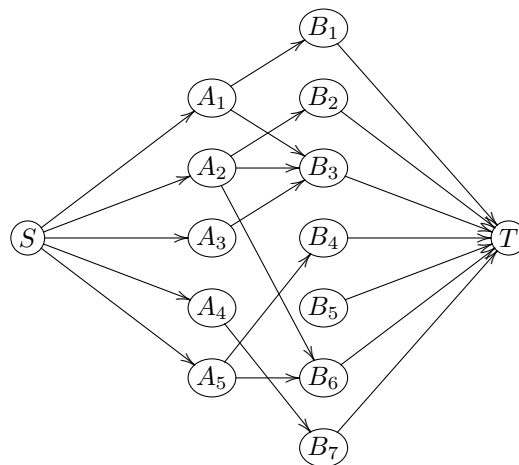
Capítulo 5

Flujo

5.1. Problemas de asignación

5.1.1. Bipartite matching

Tenemos dos conjuntos A y B , donde cada elemento de A es compatible con ciertos elementos de B . Además, tenemos la condición de que podemos asociar cada elemento de A con a lo más un solo elemento de B . Bipartite matching nos permite saber la cantidad máxima de asociaciones posibles.



Modelamiento utilizado. Todas las aristas llevan 1 de flujo.

5.2. Dinic

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long intt;
5  typedef pair<int, int> par;
6  typedef vector<vector<int> > graph;
7  typedef vector<vector<par> > wgraph;
8  #define pb push_back
9  #define ppb pop_back
10 vector <int> origen, destino, capacidad, costos, dia, orden, inicial;
11 class Dinic {
12     struct edge {
13         int to, rev;
14         intt f, cap;
15     };
16     vector<vector<edge>> g;
17     vector<intt> dist;
18     vector<int> q, work;
19     int n, sink;
20     bool bfs(int start, int finish) {
21         dist.assign(n, -1);
22         dist[start] = 0;
23         int head = 0, tail = 0;
24         q[tail++] = start;
25         while (head < tail) {
26             int u = q[head++];
27             for (const edge &e : g[u]) {
28                 int v = e.to;
29                 if (dist[v] == -1 and e.f < e.cap) {
30                     dist[v] = dist[u] + 1;
31                     q[tail++] = v;
32                 }
33             }
34         }
35         return dist[finish] != -1;
36     }
37     intt dfs(int u, intt f) {
38         if (u == sink) return f;
39         for (int &i = work[u]; i < (int)g[u].size(); ++i) {
40             edge &e = g[u][i];
41             int v = e.to;
42             if (e.cap <= e.f or dist[v] != dist[u] + 1)
43                 continue;
44             intt df = dfs(v, min(f, e.cap - e.f));
45             if (df > 0) {
46                 e.f += df;
47                 g[v][e.rev].f -= df;
48                 return df;
49             }
50         }
51         return 0;
52     }
53 public:
54     Dinic(int n) {
55         this->n = n;

```



```

56     g.resize(n);
57     dist.resize(n);
58     q.resize(n);
59 }
60 Dinic(){
61
62 }
63 // aristas bidireccionales si cap de edge b = cap, si es 0 no son
    bidireccionales!!
64 void add_edge(int u, int v, intt cap) {
65     edge a = {v, (int)g[v].size(), 0, cap};
66     edge b = {u, (int)g[u].size(), 0, 0};
67     g[u].pb(a);
68     g[v].pb(b);
69 }
70 intt max_flow(int source, int dest) {
71     sink = dest;
72     intt ans = 0;
73     while (bfs(source, dest)) {
74         work.assign(n, 0);
75         while (intt delta = dfs(source, 1000)) ans += delta;
76     }
77     return ans;
78 }
79 };
80
81 //contruyo el dinic
82
83 Dinic construir(int maximo,int tam,int g,int d){
84     Dinic D(tam+1);
85     for(int i = 0;i < inicial.size();i++){
86         D.add_edge(0,(i)*(d+1) +1,inicial[i]);
87     }
88     for(int i = 0; i < inicial.size();i++){
89         for(int j = 0; j < d; j++){
90             D.add_edge(i*(d+1) + 1 + j,i*(d+1)+2+j,g);
91         }
92     }
93     for(int i = 0;i < origen.size();i++)if(costos[i] <= maximo){
94         D.add_edge(origen[i],destino[i],capacidad[i]);
95     }
96     return D;
97 }
98
99 // busqueda binaria cuando quiero minimizar un valor X asociado al
    flujo
100
101 int BS(int lo,int hi, int tam,int tope,int d){
102     int mi;
103     while(lo<hi){
104         mi = (lo+hi)/2;
105         Dinic D = construir(orden[mi],tam,tope,d);
106         int flujo = D.max_flow(0,tam);
107         if(flujo==tope) hi = mi;
108         else lo = mi+1;
109     }
110     return lo;

```

```

111 }
112 int main(){
113     int t,aux;
114     scanf("%d",&t);
115     int caso = 1;
116     while(t--){
117         int n,d,m,u,v,c,p,e;
118         scanf("%d_%d_%d",&n,&d,&m);
119         origen.clear();destino.clear();capacidad.clear();
120         costos.clear(),orden.clear(),inicial.clear();
121         set <int> ordenNR;
122         for(int i = 0; i < m; i++){
123             scanf("%d_%d_%d_%d",&u,&v,&c,&p,&e);
124             origen.push_back( (u-1)*(d+1) + e +1);
125             destino.push_back( (v-1)*(d+1) + e+2);
126             capacidad.push_back(c);
127             costos.push_back(p);
128             ordenNR.insert(p);
129         }
130         int total = 0;
131         for(int i = 0; i < n; i++){
132             scanf("%d",&aux);
133             inicial.pb(aux);
134             total += aux;
135         }
136         for(auto it = ordenNR.begin();it != ordenNR.end();it++){
137             orden.push_back(*it);
138         }
139         int tama = n*(d+1);
140         sort(orden.begin(),orden.end());
141         int sol = BS(0,orden.size()-1,tama,total,d);
142         printf("Case_%d:",caso);
143         Dinic D = construir(orden[sol],tama,total,d);
144         int flujo = D.max_flow(0,tama);
145         if(flujo>=total)printf("_%d\n",orden[sol]);
146         else puts("_Impossible");
147         caso++;
148     }
149     return 0;
150 }

```

Capítulo 6

Programación dinámica

6.1. 0/1 Knapsack

```
1  int f[1000]={0};
2  int n=0, m=0;
3  int main(void)
4  {
5      cin >> n >> m;
6      for (int i=1;i<=n;i++)
7      {
8          int price=0, value=0;
9          cin >> price >> value;
10         for (int j=m;j>=price;j--)
11             if (f[j-price]+value>f[j])
12                 f[j]=f[j-price]+value;
13     }
14     cout << f[m] << endl;
15     return 0;
16 }
```

6.2. Bitonic Sequence

```
1  /*
2   DP para obtener la secuencia bitonic mas larga
3   BitonicSubsequence = secuencia que en primera instancia crece, y
4                       luego decrece.
5   Complejidad  $O(n^2)$ 
6   Espacio lineal
7   */
8  #include <bits/stdc++.h>
9
10 using namespace std;
11
12 int bitonicSequence(vector <int> &bitonic){
13     int lis[bitonic.size()],lds[bitonic.size()];
14     for(int i = 0; i < bitonic.size();i++){
```

```

15     lis[i] = 1;
16     lds[i] = 1;
17 }
18 for(int i = 1; i < bitonic.size();i++){
19     for(int j = 0; j < i; j++){
20         if(bitonic[i] > bitonic[j])lis[i] = max(lis[i],lis[j]+1);
21     }
22 }
23 for(int i = bitonic.size()-2; i >= 0 ;i--){
24     for(int j = bitonic.size()-1; j > i; j--){
25         if(bitonic[i] > bitonic[j])lds[i] = max(lds[i],lds[j]+1);
26     }
27 }
28 int max = 0;
29 for(int i = 0; i < bitonic.size();i++){
30     if(max < lis[i] + lds[i]-1)max = lis[i] + lds[i]-1;
31 }
32 return max;
33 }
34
35 int main(){
36     vector <int> bitonic;
37     int n,aux;
38     scanf("%d",&n);
39     for(int i = 0; i < n; i++){
40         scanf("%d",&aux);
41         bitonic.push_back(aux);
42     }
43     printf("%d\n",bitonicSequence(bitonic));
44     return 0;
45 }
46
47 /*
48 16
49 0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 5
50 */

```

6.3. Box Stacking

```

1 package com.interview.dynamic;
2
3 import java.util.Arrays;
4
5 /**
6  * Date 05/09/2015
7  * @author tusroy
8  *
9  * Given different dimensions and unlimited supply of boxes for
10  * each dimension, stack boxes
11  * on top of each other such that it has maximum height but with
12  * caveat that length and width
13  * of box on top should be strictly less than length and width of
14  * box under it. You can
15  * rotate boxes as you like.
16  *
17  * 1) Create all rotations of boxes such that length is always
18  * greater or equal to width

```

```

15  * 2) Sort boxes by base area in non increasing order (length *
    width). This is because box
16  * with more area will never ever go on top of box with less area.
17  * 3) Take T[] and result[] array of same size as total boxes after
    all rotations are done
18  * 4) Apply longest increasing subsequence type of algorithm to get
    max height.
19  *
20  * If n number of dimensions are given total boxes after rotation
    will be 3n.
21  * So space complexity is O(n)
22  * Time complexity - O(nlogn) to sort boxes. O(n^2) to apply DP on
    it So really O(n^2)
23  *
24  * References
25  * http://www.geeksforgeeks.org/dynamic-programming-set-21-box-
    stacking-problem/
26  * http://people.cs.clemson.edu/~bcdean/dp\_practice/
27  */
28  public class BoxStacking {
29
30      public int maxHeight(Dimension[] input) {
31          //get all rotations of box dimension.
32          //e.g if dimension is 1,2,3 rotations will be 2,1,3 3,2,1
33              3,1,2 . Here length is always greater
34          //or equal to width and we can do that without loss of
35          generality.
36          Dimension[] allRotationInput = new Dimension[input.length *
37              3];
38          createAllRotation(input, allRotationInput);
39
40          //sort these boxes in non increasing order by their base
41          area.(length X width)
42          Arrays.sort(allRotationInput);
43
44          //apply longest increasing subsequence kind of algorithm on
45          these sorted boxes.
46          int T[] = new int[allRotationInput.length];
47          int result[] = new int[allRotationInput.length];
48
49          for (int i = 0; i < T.length; i++) {
50              T[i] = allRotationInput[i].height;
51              result[i] = i;
52          }
53
54          for (int i = 1; i < T.length; i++) {
55              for (int j = 0; j < i; j++) {
56                  if (allRotationInput[i].length < allRotationInput[j]
57                      .length
58                      && allRotationInput[i].width <
59                          allRotationInput[j].width) {
60                      if (T[j] + allRotationInput[i].height > T[i]){
61                          T[i] = T[j] + allRotationInput[i].height;
62                          result[i] = j;
63                      }
64                  }
65              }
66          }
67      }
68  }

```

```

59         }
60
61         //find max in T[] and that will be our max height.
62         //Result can also be found using result[] array.
63         int max = Integer.MIN_VALUE;
64         for(int i=0; i < T.length; i++){
65             if(T[i] > max){
66                 max = T[i];
67             }
68         }
69
70         return max;
71     }
72
73     //create all rotations of boxes, always keeping length greater
74     or equal to width
75     private void createAllRotation(Dimension[] input,
76         Dimension[] allRotationInput) {
77         int index = 0;
78         for (int i = 0; i < input.length; i++) {
79             allRotationInput[index++] = Dimension.createDimension(
80                 input[i].height, input[i].length, input[i].
81                 width);
82             allRotationInput[index++] = Dimension.createDimension(
83                 input[i].length, input[i].height, input[i].
84                 width);
85             allRotationInput[index++] = Dimension.createDimension(
86                 input[i].width, input[i].length, input[i].
87                 height);
88         }
89     }
90
91     public static void main(String args[]) {
92         BoxStacking bs = new BoxStacking();
93         Dimension input[] = { new Dimension(3, 2, 5), new Dimension
94             (1, 2, 4) };
95         int maxHeight = bs.maxHeight(input);
96         System.out.println("Max height is " + maxHeight);
97         assert 11 == maxHeight;
98     }
99 }
100
101 /**
102  * Utility class to hold dimensions
103  * @author tusroy
104  */
105 class Dimension implements Comparable<Dimension> {
106     int height;
107     int length;
108     int width;
109
110     Dimension(int height, int length, int width) {
111         this.height = height;
112         this.length = length;
113         this.width = width;
114     }

```

```

111     }
112
113     Dimension() {
114     }
115
116     static Dimension createDimension(int height, int side1, int
117         side2) {
118         Dimension d = new Dimension();
119         d.height = height;
120         if (side1 >= side2) {
121             d.length = side1;
122             d.width = side2;
123         } else {
124             d.length = side2;
125             d.width = side1;
126         }
127         return d;
128     }
129
130     /**
131      * Sorts by base area(length X width)
132      */
133     @Override
134     public int compareTo(Dimension d) {
135         if (this.length * this.width >= d.length * d.width) {
136             return -1;
137         } else {
138             return 1;
139         }
140     }
141
142     @Override
143     public String toString() {
144         return "Dimension[" + height + ", length=" + length
145             + ", width=" + width + "]";
146     }

```

6.4. Break multiple words with no space into space

```

1 package com.interview.dynamic;
2
3 import java.util.*;
4
5 /**
6  * Date 08/01/2014
7  * @author tusroy
8  *
9  * Given a string and a dictionary, split this string into multiple
10     words such that
11     * each word belongs in dictionary.
12     *
13     * e.g peanutbutter -> pea nut butter
14     * e.g Iliketoplay -> I like to play
15     *

```

```

15  * Solution
16  * DP solution to this problem
17  * if( input[i...j] belongs in dictionary) T[i][j] = i
18  * else{
19  *     T[i][j] = k if T[i][k-1] != -1 && T[k][j] != -1
20  *
21  * Test cases
22  * 1) Empty string
23  * 2) String where entire string is in dictionary
24  * 3) String which cannot be split into words which are in
    dictionary
25  * 3) String which can be split into words which are in dictionary
26  *
27  */
28  public class BreakMultipleWordsWithNoSpaceIntoSpace {
29
30
31  /**
32   * Recursive and slow version of breaking word problem.
33   * If no words can be formed it returns null
34   */
35  public String breakWord(char[] str,int low,Set<String>
    dictionary){
36      StringBuffer buff = new StringBuffer();
37      for(int i= low; i < str.length; i++){
38          buff.append(str[i]);
39          if(dictionary.contains(buff.toString())){
40              String result = breakWord(str, i+1, dictionary);
41              if(result != null){
42                  return buff.toString() + "_" + result;
43              }
44          }
45      }
46      if(dictionary.contains(buff.toString())){
47          return buff.toString();
48      }
49      return null;
50  }
51
52  /**
53   * Dynamic programming version for breaking word problem.
54   * It returns null string if string cannot be broken into
    multiple words
55   * such that each word is in dictionary.
56   * Gives preference to longer words over splits
57   * e.g peanutbutter with dict{pea nut butter peanut} it would
    result in
58   * peanut butter instead of pea nut butter.
59   */
60  public String breakWordDP(String word, Set<String> dict){
61      int T[][] = new int[word.length()][word.length()];
62
63      for(int i=0; i < T.length; i++){
64          for(int j=0; j < T[i].length ; j++){
65              T[i][j] = -1; //-1 indicates string between i to j
                cannot be split
66          }
67      }
68  }
69  }

```



```

67     }
68
69     //fill up the matrix in bottom up manner
70     for(int l = 1; l <= word.length(); l++){
71         for(int i=0; i < word.length() -l + 1 ; i++){
72             int j = i + l-1;
73             String str = word.substring(i,j+1);
74             //if string between i to j is in dictionary T[i][j]
              is true
75             if(dict.contains(str)){
76                 T[i][j] = i;
77                 continue;
78             }
79             //find a k between i+1 to j such that T[i][k-1] &&
              T[k][j] are both true
80             for(int k=i+1; k <= j; k++){
81                 if(T[i][k-1] != -1 && T[k][j] != -1){
82                     T[i][j] = k;
83                     break;
84                 }
85             }
86         }
87     }
88     if(T[0][word.length()-1] == -1){
89         return null;
90     }
91
92     //create space separate word from string is possible
93     StringBuffer buffer = new StringBuffer();
94     int i = 0; int j = word.length() -1;
95     while(i < j){
96         int k = T[i][j];
97         if(i == k){
98             buffer.append(word.substring(i, j+1));
99             break;
100         }
101         buffer.append(word.substring(i,k) + " ");
102         i = k;
103     }
104
105     return buffer.toString();
106 }
107
108 /**
109  * Prints all the words possible instead of just one
    combination.
110  * Reference
111  * https://leetcode.com/problems/word-break-ii/
112  */
113 public List<String> wordBreakTopDown(String s, Set<String>
    wordDict) {
114     Map<Integer, List<String>> dp = new HashMap<>();
115     int max = 0;
116     for (String s1 : wordDict) {
117         max = Math.max(max, s1.length());
118     }
119     return wordBreakUtil(s, wordDict, dp, 0, max);

```

```

120     }
121
122     private List<String> wordBreakUtil(String s, Set<String> dict,
123         Map<Integer, List<String>> dp, int start, int max) {
124         if (start == s.length()) {
125             return Collections.singletonList("");
126         }
127         if (dp.containsKey(start)) {
128             return dp.get(start);
129         }
130
131         List<String> words = new ArrayList<>();
132         for (int i = start; i < start + max && i < s.length(); i++)
133         {
134             String newWord = s.substring(start, i + 1);
135             if (!dict.contains(newWord)) {
136                 continue;
137             }
138             List<String> result = wordBreakUtil(s, dict, dp, i + 1,
139                 max);
140             for (String word : result) {
141                 String extraSpace = word.length() == 0 ? "" : " ";
142                 words.add(newWord + extraSpace + word);
143             }
144         }
145         dp.put(start, words);
146         return words;
147     }
148
149     /**
150      * Check if any one solution exists.
151      * https://leetcode.com/problems/word-break/
152      */
153     public boolean wordBreakTopDownOneSolution(String s, Set<String>
154         wordDict) {
155         Map<Integer, Boolean> dp = new HashMap<>();
156         int max = 0;
157         for (String s1 : wordDict) {
158             max = Math.max(max, s1.length());
159         }
160         return wordBreakTopDownOneSolutionUtil(s, wordDict, 0, max,
161             dp);
162     }
163
164     private boolean wordBreakTopDownOneSolutionUtil(String s, Set<
165         String> dict, int start, int max, Map<Integer, Boolean> dp)
166     {
167         if (start == s.length()) {
168             return true;
169         }
170         if (dp.containsKey(start)) {
171             return dp.get(start);
172         }

```

```

170         for (int i = start; i < start + max && i < s.length(); i++)
171             {
172                 String newWord = s.substring(start, i + 1);
173                 if (!dict.contains(newWord)) {
174                     continue;
175                 }
176                 if (wordBreakTopDownOneSolutionUtil(s, dict, i + 1, max
177                     , dp)) {
178                     dp.put(start, true);
179                     return true;
180                 }
181             }
182         dp.put(start, false);
183         return false;
184     }
185
186     public boolean wordBreakBottomUp(String s, List<String>
187         wordList) {
188         boolean[] T = new boolean[s.length() + 1];
189         Set<String> set = new HashSet<>();
190         for (String word : wordList) {
191             set.add(word);
192         }
193         T[0] = true;
194         for (int i = 1; i <= s.length(); i++) {
195             for (int j = 0; j < i; j++) {
196                 if (T[j] && set.contains(s.substring(j, i))) {
197                     T[i] = true;
198                     break;
199                 }
200             }
201         }
202         return T[s.length()];
203     }
204
205     public static void main(String args[]){
206         Set<String> dictionary = new HashSet<String>();
207         dictionary.add("I");
208         dictionary.add("like");
209         dictionary.add("had");
210         dictionary.add("play");
211         dictionary.add("to");
212         String str = "Ihadliketoplay";
213         BreakMultipleWordsWithNoSpaceIntoSpace bmw = new
214             BreakMultipleWordsWithNoSpaceIntoSpace();
215         String result1 = bmw.breakWordDP(str, dictionary);
216
217         System.out.print(result1);
218     }
219 }

```

6.5. Burst Balloon

```

1 package com.interview.dynamic;
2
3 /**
4  * Date 03/02/2016

```

```

5  * @author Tushar Roy
6  *
7  * Given n balloons, indexed from 0 to n-1. Each balloon is painted
   with a number on it represented
8  * by array nums. You are asked to burst all the balloons. If the
   you burst balloon i you will
9  * get nums[left] * nums[i] * nums[right] coins. Here left and
   right are adjacent indices of i. After the burst,
10 * the left and right then becomes adjacent.
11 * Find the maximum coins you can collect by bursting the balloons
   wisely.
12 *
13 * Time complexity  $O(n^3)$ 
14 * Space complexity  $O(n^2)$ 
15 *
16 * Reference
17 * https://leetcode.com/problems/burst-balloons/
18 */
19 public class BurstBalloons {
20
21     /**
22      * Dynamic programming solution.
23      */
24     public int maxCoinsBottomUpDp(int[] nums) {
25
26         int T[][] = new int[nums.length][nums.length];
27
28         for (int len = 1; len <= nums.length; len++) {
29             for (int i = 0; i <= nums.length - len; i++) {
30                 int j = i + len - 1;
31                 for (int k = i; k <= j; k++) {
32                     //leftValue/rightValue is initially 1. If there
                       is element on
33                     // left/right of k then left/right value will
                       take that value.
34                     int leftValue = 1;
35                     int rightValue = 1;
36                     if (i != 0) {
37                         leftValue = nums[i-1];
38                     }
39                     if (j != nums.length - 1) {
40                         rightValue = nums[j+1];
41                     }
42
43                     //before is initially 0. If k is i then before
                       will
44                     //stay 0 otherwise it gets value T[i][k-1]
45                     //after is similarly 0 initially. if k is j
                       then after will
46                     //stay 0 other will get value T[k+1][j]
47                     int before = 0;
48                     int after = 0;
49                     if (i != k) {
50                         before = T[i][k-1];
51                     }
52                     if (j != k) {
53                         after = T[k+1][j];

```

```

54         }
55         T[i][j] = Math.max(leftValue * nums[k] *
56                             rightValue + before + after,
57                             T[i][j]);
58     }
59 }
60 return T[0][nums.length - 1];
61 }
62
63 /**
64  * Recursive solution.
65  */
66 public int maxCoinsRec(int nums[]) {
67     int[] nums1 = new int[nums.length + 2];
68     nums1[0] = 1;
69     nums1[nums1.length - 1] = 1;
70     for (int i = 0; i < nums.length; i++) {
71         nums1[i+1] = nums[i];
72     }
73     return maxCoinsRecUtil(nums1);
74 }
75
76 private int maxCoinsRecUtil(int[] nums) {
77     if (nums.length == 2) {
78         return 0;
79     }
80
81     int max = 0;
82     for (int i = 1; i < nums.length - 1; i++) {
83         int val = nums[i - 1]*nums[i]*nums[i+1] +
84                 maxCoinsRecUtil(formNewArray(nums, i));
85         if (val > max) {
86             max = val;
87         }
88     }
89     return max;
90 }
91
92 private int[] formNewArray(int[] input, int doNotIncludeIndex)
93 {
94     int[] newArray = new int[input.length - 1];
95     int index = 0;
96     for (int i = 0; i < input.length; i++) {
97         if (i == doNotIncludeIndex) {
98             continue;
99         }
100         newArray[index++] = input[i];
101     }
102     return newArray;
103 }
104
105 public static void main(String args[]) {
106     BurstBalloons bb = new BurstBalloons();
107     int input[] = {2, 4, 3, 5};

```

```

108         System.out.print(bb.maxCoinsBottomUpDp(input));
109     }
110 }

```

6.6. Catalan

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  /*
4  1) Count the number of expressions containing n pairs of
      parentheses which are correctly matched. For n = 3,
5  possible expressions are ((())), ()(()), ()()(), (())(), (()()).
6  2) Count the number of possible Binary Search Trees with n keys (
      See this)
7  3) Count the number of full binary trees (A rooted binary tree is
      full if every vertex has either two children or no children)
      with n+1 leaves.
8  */
9  // A recursive function to find nth catalan number
10 unsigned long int catalan(unsigned int n)
11 {
12     // Base case
13     if (n <= 1) return 1;
14
15     // catalan(n) is sum of catalan(i)*catalan(n-i-1)
16     unsigned long int res = 0;
17     for (int i=0; i<n; i++)
18         res += catalan(i)*catalan(n-i-1);
19
20     return res;
21 }
22
23 // Driver program to test above function
24 int main()
25 {
26     for (int i=0; i<10; i++)
27         cout << catalan(i) << " ";
28     return 0;
29 }

```

6.7. Coin changing

```

1  /*
2  DP sobre la cantidad de formas de pagar X dado un set de monedas(
      cantidad infinita)
3  Complejidad X*Monedas/
4  */
5
6  #include <bits/stdc++.h>
7
8  using namespace std;
9
10 int solution(int total, vector<int>&monedas){
11     vector <int> temp(total+1,0);
12     temp[0] = 1;

```

```

13     for(int i = 0; i < monedas.size();i++){
14         for(int j = 1; j <= total; j++)
15             if(j >= monedas[i])temp[j]+= temp[j-monedas[i]];
16     }
17     return temp[total];
18 }
19
20 int main(){
21     int total = 15;
22     vector <int> monedas;
23     monedas.push_back(3);
24     monedas.push_back(4);
25     monedas.push_back(6);
26     monedas.push_back(7);
27     monedas.push_back(9);
28     printf("%d\n",solution(total,monedas));
29     return 0;
30 }

```

6.8. Cutting sticks

```

1  //dada una vara, con cortes dados, el cortar cuesta el tamaño de
   la vara, se busca minimizar el costo total de realizar los
   cortes
2
3  #include <bits/stdc++.h>
4
5  int n, l, C[52], DP[52][52];
6  int min(int a, int b){ return (a < b)? a:b; }
7
8  int BottomUp(int ini, int fin)
9  {
10     int i, j, k, L;
11
12     /*Caso Base*/
13     for(i = 0; i < fin; i++)
14         DP[i][i+1] = 0;
15
16     for(L = 2; L <= fin; L++){
17         for(i = 0; i <= (fin - L); i++){
18             j = i + L;
19             DP[i][j] = INT_MAX;
20             for(k = i+1; k < j; k++){
21                 DP[i][j] = min(DP[i][j], DP[i][k] + DP[k][j] + (C[j]
22                                     ] - C[i]));
23             }
24         }
25
26         return DP[0][fin];
27     }
28
29     int main()
30     {
31         int i, sol;
32
33         while(1)

```

```

34     {
35         scanf("%d", &l);
36         if(l == 0) break;
37         scanf("%d", &n);
38         C[0] = 0;
39         for(i = 1; i <= n; i++)
40             scanf("%d", &C[i]);
41         C[n+1] = 1;
42         sol = BottomUp(0, n+1);
43         printf("The minimum cutting is %d.\n", sol);
44     }
45     return 0;
46 }

```

6.9. Distinct subsequence

```

1  package com.interview.dynamic;
2
3  /**
4   * Date 03/20/2016
5   * @author Tushar Roy
6   *
7   * Given a string S and a string T, count the number of distinct
8   * subsequences of T in S.
9   * Time complexity  $O(n^2)$ 
10  * Space complexity  $O(n^2)$ 
11  *
12  * https://leetcode.com/problems/distinct-subsequences/
13  */
14  public class DistinctSubsequence {
15      public int numDistinct(String s, String t) {
16          if (s.length() == 0 || t.length() == 0) {
17              return 0;
18          }
19          int[][] T = new int[t.length() + 1][s.length() + 1];
20          for (int i = 0; i < T[0].length; i++) {
21              T[0][i] = 1;
22          }
23          for (int i = 1; i < T.length; i++) {
24              for (int j = 1; j < T[0].length; j++) {
25                  if (s.charAt(j - 1) == t.charAt(i - 1)) {
26                      T[i][j] = T[i-1][j-1] + T[i][j-1];
27                  } else {
28                      T[i][j] = T[i][j-1];
29                  }
30              }
31          }
32          return T[t.length()][s.length()];
33      }
34
35      public static void main(String args[]) {
36          DistinctSubsequence ds = new DistinctSubsequence();
37          System.out.println(ds.numDistinct("abdacgblc", "abc"));
38      }
39  }

```


6.10. Divide and conquer

```

1  / tags: DP + divide and conquer optimization
2  #include <bits/stdc++.h>
3  using namespace std;
4  #define rep(i,a,b) for(int i=a; i<=b; i++)
5  typedef long long int ll;
6  #define MAXN 6000
7
8  int N;
9  ll B,C;
10 ll H[MAXN];
11 ll accH[MAXN];
12 ll accKH[MAXN];
13 ll dp[MAXN+1][MAXN];
14
15 ll deltaAccH(int i, int j) { return i > 0 ? accH[j] - accH[i-1] :
    accH[j]; }
16 ll deltaAccKH(int i, int j) { return i > 0 ? accKH[j] - accKH[i-1]
    : accKH[j]; }
17
18 // Compute total distance cost between i and j assuming that
19 // there is a station in both i and j
20 // (if i < 0, only in j)
21 // (if j >= N, only in i)
22 ll cost(int i, int j) {
23     // station only in j
24     if (i < 0) return j * accH[j] - accKH[j];
25
26     // station only in i
27     if (j >= N) return deltaAccKH(i, N-1) - i * deltaAccH(i, N-1);
28
29     // normal case: both stations
30     int ml = (i+j)/2;
31     int mr = ml+1;
32     ll left_cost = deltaAccKH(i, ml) - i * deltaAccH(i, ml);
33     ll right_cost = j * deltaAccH(mr, j) - deltaAccKH(mr, j);
34     return left_cost + right_cost;
35 }
36
37 // Solve DP[k][i] where i1 <= i <= i2 using divide and conquer
    optimization
38 // DP[k][i] = min { DP[k-1][j-1] + cost(j, i+1) for p1 <= j <= p2 }
39 void fill(int k, int i1, int i2, int j1, int j2) {
40     if (i1 > i2) return;
41     int im = (i1+i2)/2;
42     int jmin = max(j1, k-1);
43     int jmax = min(j2, im);
44     ll min_val = LLONG_MAX;
45     int best_j = -1;
46     rep(j,jmin,jmax) {
47         ll val = cost(j,im+1) + (j > 0 ? dp[k-1][j-1] : 0);
48         if (val < min_val) min_val = val, best_j = j;
49     }
50     dp[k][im] = min_val;
51     fill(k,i1,im-1,j1,best_j);
52     fill(k,im+1,i2,best_j,j2);

```

```

53 }
54
55 int main() {
56     while (scanf("%d%lld%lld", &N, &B, &C) == 3) {
57         rep(i,0,N-1) scanf("%lld", &H[i]);
58
59         // -- precompute acc sums --
60         accH[0] = H[0];
61         accKH[0] = 0;
62         rep(k, 1, N-1) {
63             accH[k] = accH[k-1] + H[k];
64             accKH[k] = accKH[k-1] + H[k] * k;
65         }
66
67         // -- DP --
68         // k = 0
69         rep(i,0,N-2) dp[0][i] = cost(-1, i+1);
70         // k >= 1
71         rep(k,1,N) fill(k,k-1,N-1,0,N-1);
72
73         // -- print output --
74         rep(k,1,N) {
75             if (k > 1) printf("\n");
76             ll total_cost = dp[k][N-1] * C + k * B;
77             printf("%lld", total_cost);
78         }
79         puts("");
80     }
81
82     return 0;
83 }

```

6.11. Edit Distance

```

1 package com.interview.dynamic;
2
3 import java.util.List;
4
5 /**
6  * Date 07/07/2014
7  * @author Tushar Roy
8  *
9  * Given two strings how many minimum edits(update, delete or add)
10    is needed to convert one string to another
11  *
12  * Time complexity is O(m*n)
13  * Space complexity is O(m*n)
14  *
15  * References:
16  * http://www.geeksforgeeks.org/dynamic-programming-set-5-edit-
17  * https://en.wikipedia.org/wiki/Edit\_distance
18  */
19 public class EditDistance {
20
21     /**
22      * Uses recursion to find minimum edits

```

```

22     */
23     public int recEditDistance(char[] str1, char str2[], int len1,
24                               int len2){
25         if(len1 == str1.length){
26             return str2.length - len2;
27         }
28         if(len2 == str2.length){
29             return str1.length - len1;
30         }
31         return min(recEditDistance(str1, str2, len1 + 1, len2 + 1)
32                   + str1[len1] == str2[len2] ? 0 : 1, recEditDistance(
33                     str1, str2, len1, len2 + 1) + 1, recEditDistance(str1,
34                       str2, len1 + 1, len2) + 1);
35     }
36
37     /**
38      * Uses bottom up DP to find the edit distance
39      */
40     public int dynamicEditDistance(char[] str1, char[] str2){
41         int temp[][] = new int[str1.length+1][str2.length+1];
42
43         for(int i=0; i < temp[0].length; i++){
44             temp[0][i] = i;
45         }
46
47         for(int i=0; i < temp.length; i++){
48             temp[i][0] = i;
49         }
50
51         for(int i=1; i <= str1.length; i++){
52             for(int j=1; j <= str2.length; j++){
53                 if(str1[i-1] == str2[j-1]){
54                     temp[i][j] = temp[i-1][j-1];
55                 }else{
56                     temp[i][j] = 1 + min(temp[i-1][j-1], temp[i-1][j], temp[i][j-1]);
57                 }
58             }
59         }
60
61         printActualEdits(temp, str1, str2);
62         return temp[str1.length][str2.length];
63     }
64
65     /**
66      * Prints the actual edits which needs to be done.
67      */
68     public void printActualEdits(int T[][], char[] str1, char[]
69                                   str2) {
70         int i = T.length - 1;
71         int j = T[0].length - 1;
72         while(true) {
73             if (i == 0 || j == 0) {
74                 break;
75             }
76             if (str1[i-1] == str2[j-1]) {

```

```

73         i = i-1;
74         j = j-1;
75     } else if (T[i][j] == T[i-1][j-1] + 1){
76         System.out.println("Edit_" + str2[j-1] + "_" + str1[i-1] + "in_" +
77             string2_to_string1);
78         i = i-1;
79         j = j-1;
80     } else if (T[i][j] == T[i-1][j] + 1) {
81         System.out.println("Delete_in_string1_" + str1[i-1]);
82         i = i-1;
83     } else if (T[i][j] == T[i][j-1] + 1){
84         System.out.println("Delete_in_string2_" + str2[j-1]);
85         j = j-1;
86     } else {
87         throw new IllegalArgumentException("Some_wrong_with_" +
88             "given_data");
89     }
90 }
91
92 private int min(int a,int b, int c){
93     int l = Math.min(a, b);
94     return Math.min(l, c);
95 }
96
97 public static void main(String args[]){
98     String str1 = "azced";
99     String str2 = "abcdef";
100     EditDistance editDistance = new EditDistance();
101     int result = editDistance.dynamicEditDistance(str1.
102         toCharArray(), str2.toCharArray());
103     System.out.print(result);
104 }
105 }

```

6.12. Sum 2D Rectangle

```

1 package com.interview.dynamic;
2
3 /**
4  * Date 03/11/2016
5  * @author Tushar Roy
6  *
7  * Given a 2D array find the sum in given range defining a
8  * rectangle.
9  *
10 * Time complexity construction O(n*m)
11 * Time complexity of query O(1)
12 * Space complexity is O(n*m)
13 *
14 * Reference
15 * https://leetcode.com/problems/range-sum-query-2d-immutable/
16 */

```

```

16 public class Immutable2DSumRangeQuery {
17     private int[][] T;
18
19     public Immutable2DSumRangeQuery(int[][] matrix) {
20         int row = 0;
21         int col = 0;
22         if (matrix.length != 0) {
23             row = matrix.length;
24             col = matrix[0].length;
25         }
26         T = new int[row + 1][col + 1];
27         for (int i = 1; i < T.length; i++) {
28             for (int j = 1; j < T[0].length; j++) {
29                 T[i][j] = T[i - 1][j] + T[i][j - 1] + matrix[i -
30                     1][j - 1] - T[i - 1][j - 1];
31             }
32         }
33
34         public int sumQuery(int row1, int col1, int row2, int col2) {
35             row1++;
36             col1++;
37             row2++;
38             col2++;
39             return T[row2][col2] - T[row1 - 1][col2] - T[row2][col1 -
40                 1] + T[row1 - 1][col1 - 1];
41         }
42
43         public static void main(String args[]) {
44             int[][] input = {{3, 0, 1, 4, 2},
45                             {5, 6, 3, 2, 1},
46                             {1, 2, 0, 1, 5},
47                             {4, 1, 0, 1, 7},
48                             {1, 0, 3, 0, 5}};
49
50             int[][] input1 = {{2,0,-3,4}, {6, 3, 2, -1}, {5, 4, 7, 3},
51                             {2, -6, 8, 1}};
52             Immutable2DSumRangeQuery isr = new Immutable2DSumRangeQuery
53                 (input1);
54             System.out.println(isr.sumQuery(1, 1, 2, 2));
55         }
56     }
57 }

```

6.13. Sum Dice

```

1
2 package com.interview.dynamic;
3
4 /**
5  * @author Tushar Roy
6  * http://www.geeksforgeeks.org/dice-throw-problem/
7  * This solution assumes that 1,2,1 is different from 2,1,1 which
8  * is different from 1,1 2
9  * so total 3 ways are possible
10  * program to find number of ways to get sum 'x' with 'n'
11  */
12 public class DiceThrowWays {

```

```
12
13     public int numberOfWays(int n, int f, int k){
14
15         int T[][] = new int[n+1][k+1];
16         T[0][0] = 1;
17         /* for(int i=0; i < T.length; i++){
18             T[0][i] = 1;
19         }*/
20
21         for(int i=1; i <= n; i++){
22             for(int j=1; j <= i*f && j <= k ; j++){
23                 if(j == i){
24                     T[i][j] = 1;
25                     continue;
26                 }
27                 if(j < i){
28                     continue;
29                 }
30                 for(int l =1; l <=f ;l++){
31                     if(j >= l){
32                         T[i][j] += T[i-1][j-l];
33                     }
34                 }
35             }
36         }
37         return T[n][k];
38     }
39
40     public static void main(String args[]){
41         DiceThrowWays dtw = new DiceThrowWays();
42         System.out.println(dtw.numberOfWays(3, 3, 6));
43     }
44 }
```

Capítulo 7

Contenido adicional

7.1. Fast input

```
1
2 #define GETCHAR getchar_unlocked
3 #define PUTCHAR putchar_unlocked
4
5 inline void readInt(int &n){
6     n = 0;
7     bool flag=1;
8     char c;
9     int sign=1;
10    while (1){
11        c = GETCHAR();
12        if(c=='-') sign=-1;
13        else if(c>='0'&&c<='9') {n = n * 10 + c - '0';flag=0;}
14        else if(flag!=1) break;
15    }
16    n *= sign;
17 }
```

7.2. Usar en caso de emergencia



GOD BLESS OUR SAVIOUR

Índice alfabético

Bipartite matching, 27

Componentes conexas, 2

Conjuntos disjuntos, 2

Fenwick Tree, 1

Particiones, 2

RSQ, 2