Apunte ICPC

4 de noviembre de 2017

Índice general

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&(-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.

```
struct FenwickTree {
  vi ft;
  FenwickTree(){}
  FenwickTree(int n){
    ft.assign(n + 1, 0);
  }

  int query(int b) {
    int sum = 0;
    for (; b; b -= b&(-b)) sum += ft[b];
    return sum;
  }

  int query(int a, int b) { \\RSQ
    return query(b) - (a == 1 ? 0 : query(a - 1));
  }

  void update(int k, int v) {                    // note: n = ft.size() - 1
    for (; k < (int)ft.size(); k += k&(-k)) ft[k] += v;
  }
};
```

1.2. Union-Find

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

```
class UnionFind {
private:
  vi p, rank, setSize;
  int numSets;
public:
  UnionFind(int N) {
  setSize.assign(N, 1); numSets = N; rank.assign(N, 0);
  p.assign(N, 0); for (int i = 0; i < N; i++) p[i] = i; }
  int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j) {
  if (!isSameSet(i, j)) { numSets--;
  int x = findSet(i), y = findSet(j);
  // rank is used to keep the tree short
  if (rank[x] > rank[y]) { p[y] = x; setSize[x] += setSize[y]; }
  else                   { p[x] = y; setSize[y] += setSize[x];
                               if (rank[x] == rank[y]) rank[y]++; } } }
  int numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};
```

1.3. Segment Tree

1.3.1. Iterativo

```
1  struct prodsgn {
2      int sgn;
3      prodsgn() {sgn = 1;}
4      prodsgn(int x) {
5          sgn = (x > 0) - (x < 0);
6      }
7      prodsgn(const prodsgn &a,
8              const prodsgn &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 ansl, ansr;
72         for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
73             if (l&1) ansl = node(ansl, t[l++]);
74             if (r&1) ansr = node(t[--r], ansr);
75         }
76         return node(ansl, 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  };
```

1.4. 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, ii> 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

2.4.1. Dijkstra

Utilizamos la representacion vvii con pares (vecino,peso)

```
1 asd
```

Capítulo 3

Matemática

3.1. Pascal

```
for(int i=1;i<150;i++){
    for(int j=1;j<i+3;j++){
      pascal[i][j]=pascal[i-1][j-1]+pascal[i-1][j];
    }
  }
  cin>>n;
  for(ll i=0;i<n+1;i++){
      printf("%lld ",pascal[n-1][i]);
  }
```

3.2. Criba

```
vector<bool> isprime;
vector<int> primes;
void sieve(int n) {
  isprime.assign(n + 1,true);

  isprime[1] = false; isprime[2] = true;
  for (int i = 2; i <= n; i++) {
    if (isprime[i]) {
      for (int j = i*i;  j < n; j+=i) {
          isprime[j] = false;
      }
    }
  }
  for (int i = 2; i < n; i++) {
    if (isprime[i]) {
      primes.push_back(i);
    }
  }
  return;
}
```

Capítulo 4

Geometria

4.1. Closest pair

```
#define px second
#define py first
typedef pair<ll,ll> pairll;
int n;
pairll pnts[100000];
set<pairll> box;
double best;
int compx(pairll a,pairll b){
  return a.px<b.px;
}
int main(){
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%lld␣%lld",&pnts[i].px,&pnts[i].py);
  }
  sort(pnts,pnts+n,compx);
  best=100000000000;
  box.insert(pnts[0]);
  int left=0;
  for(int i=1;i<n;i++){
    while(left<i && pnts[i].px-pnts[left].px > best) box.erase(pnts[left++]);
    for(typeof(box.begin()) it=box.lower_bound(make_pair(pnts[i].py-best, pnts[i].px-best));
    it!=box.end() && pnts[i].py+best>=it->py; it++){
          best = min(best, sqrt(pow(pnts[i].py - it->py, 2.0)+pow(pnts[i].px - it->px, 2.0)));
    }
    box.insert(pnts[i]);
    printf("%.2f\n",best);
  }
}
```

4.2. Radial sweep example

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

```

```
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 } O;
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(), (O - B).abs());
56         return (O - inter(A, B, O, Point(cos(nowAng), sin(nowAng)))).abs();
57     }
58     bool operator < (const Line &rhs) const {
59         return sgn(dis() - rhs.dis()) < 0;
60     }
```

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

4.3. 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 siempre van de a pares
47          i++;
48        }else{
49          if(FT.query(poly[i].y)%2==0){
50            perdido+=poly[i].valor;
51          }
52        }
53      }
54      printf("%lld\n",perdido);
55    }
56    return 0;
57  }
```

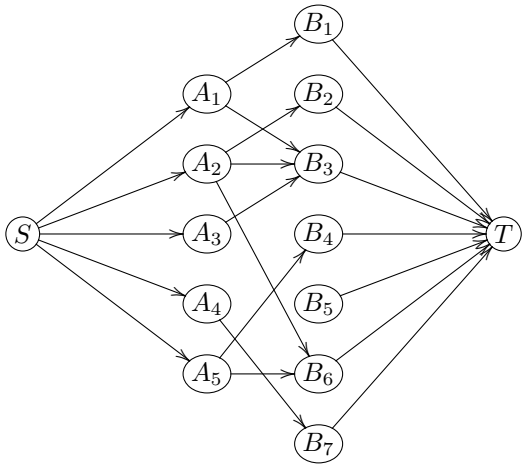
Capítulo 5

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.

Capítulo 6

Programación dinámica

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