

Apunte ICPC

8 de octubre 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 |
| 2. Grafos | 3 |
| 2.1. Single source shortest path | 3 |
| 2.1.1. | 3 |
| 3. Flujo | 4 |
| 3.1. Problemas de asignación | 4 |
| 3.1.1. Bipartite matching | 4 |
| 4. Contenido adicional | 5 |
| 4.1. Usar en caso de emergencia | 5 |

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.size() - 1
19        for (; k < (int)ft.size(); k += k&(-k)) ft[k] += v;
20    }
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    bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
11    void unionSet(int i, int j) {
12        if (!isSameSet(i, j)) { numSets--;
13            int x = findSet(i), y = findSet(j);
14            // rank is used to keep the tree short
15            if (rank[x] > rank[y]) { p[y] = x; setSize[x] += setSize[y]; }
16            else { p[x] = y; setSize[y] += setSize[x];
17                if (rank[x] == rank[y]) rank[y]++; } } }
18    int numDisjointSets() { return numSets; }
19    int sizeOfSet(int i) { return setSize[findSet(i)]; }
20 };

```

Capítulo 2

Grafos

2.1. Single source shortest path

2.1.1. Dijkstra

Utilizamos la representacion vvii con pares (vecino,peso)

1 asd

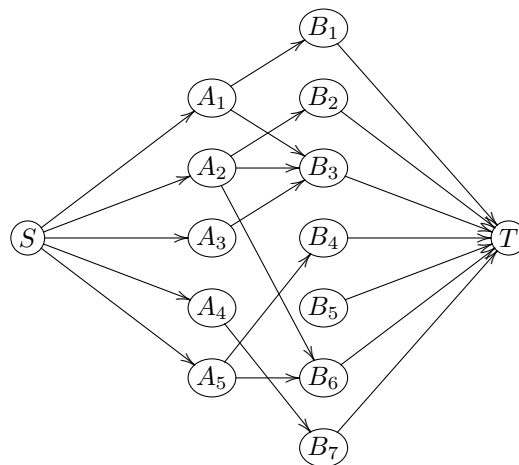
Capítulo 3

Flujo

3.1. Problemas de asignación

3.1.1. 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 4

Contenido adicional

4.1. Usar en caso de emergencia



GOD BLESS OUR SAVIOUR

Índice alfabético

Bipartite matching, 4

Componentes conexas, 2

Conjuntos disjuntos, 2

Fenwick Tree, 1

Particiones, 2

RSQ, 2