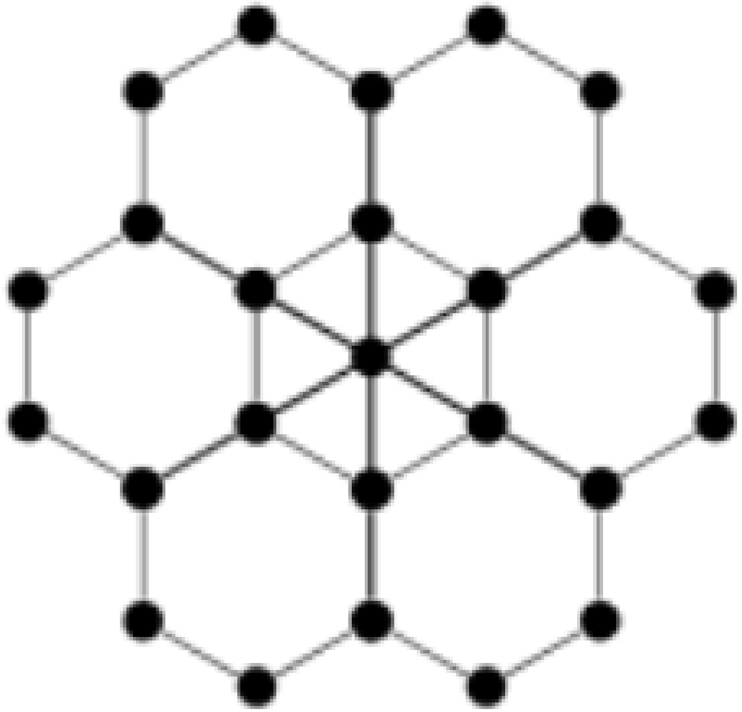
Planar Graphs and Doubly Connected Edge List (DCEL)



Planar graph. A graph $G = (V, E)$ (vertex set V, edge set E) is *planar* if it can be embedded in the plane without crossings (see Section 1.2.3.2). A straight line planar embedding of a planar graph determines a partition of the plane called *planar subdivision* or *map*. Let v, e, and f denote respectively the numbers of vertices, edges, and regions (including the single unbounded region) of the subdivision. These three parameters are related by the classical *Euler's formula* [Bollobás (1979)]

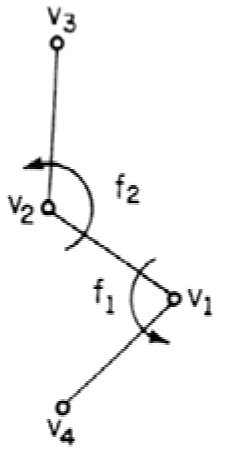
$$v - e + f = 2. \tag{1.1}$$

If we have the additional property that each vertex has degree ≥ 3, then it is a simple exercise to prove the following inequalities

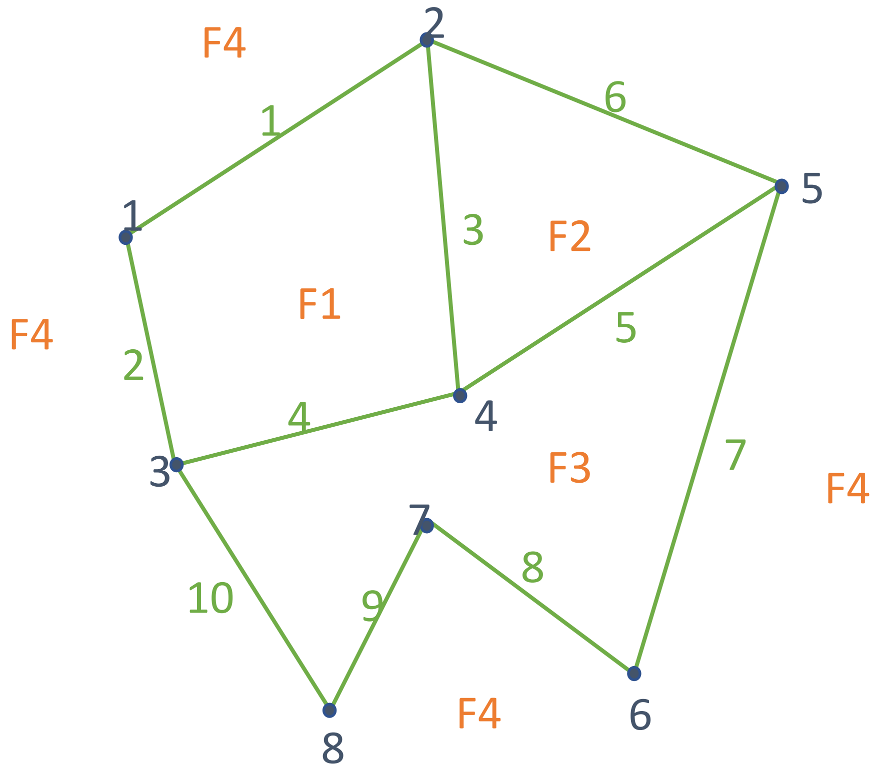
$$\begin{cases} v \leq \frac{2}{3}e, & e \leq 3v - 6 \\ e \leq 3f - 6, & f \leq \frac{2}{3}e \\ v \leq 2f - 4, & f \leq 2v - 4 \end{cases} \tag{1.2}$$

which show that v, e and f are pairwise proportional. (Note that the three rightmost inequalities are unconditionally valid.)

The doubly-connected-edge-list (DCEL)

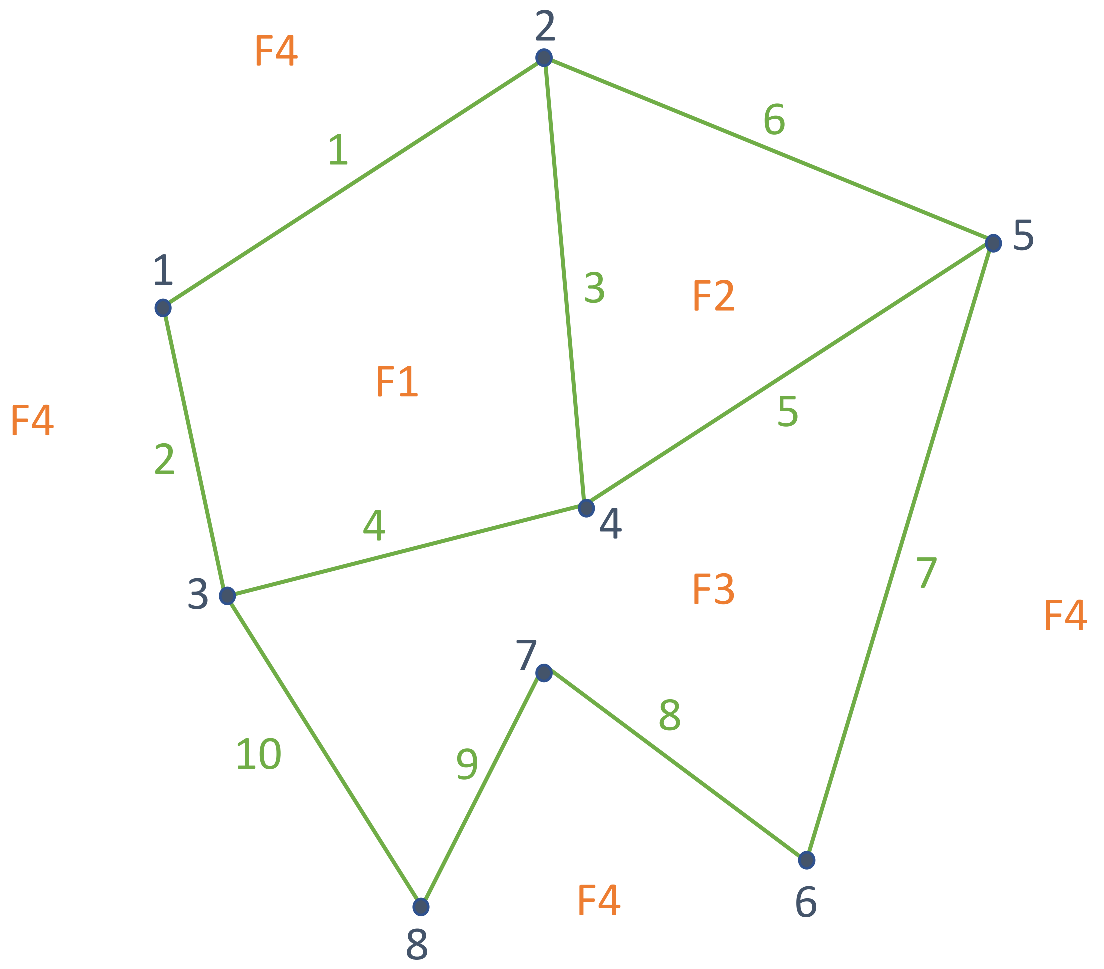


	V1	V2	F1	F2	P1	P2
1						
2						
⋮						
a_1	1	2	1	2	a_2	a_3
⋮						
a_2	4	1	1			
⋮						
a_3	2	3		2		



	V1	V2	F1	F2	P1	P2
1	1	2	4	1	2	3
2	3	1	4	1	10	1
3	4	2	1	2	4	6
4	3	4	1	3	2	5
5	4	5	2	3	3	7
6	5	2	2	4	5	1
7	6	5	3	4	8	6
8	6	7	4	3	7	9
9	8	7	3	4	10	8
10	8	3	4	3	9	4

V1: lower end of the edge
V2: upper end of the edge
F1: left face of the edge
F2: right face of the edge
P1: next edge going counterclockwise from lower end
P2: next edge going counterclockwise from upper end



	V1	V2	F1	F2	P1	P2
1	1	2	4	1	2	3
2	3	1	4	1	10	1
3	4	2	1	2	4	6
4	3	4	1	3	2	5
5	4	5	2	3	3	7
6	5	2	2	4	5	1
7	6	5	3	4	8	6
8	6	7	4	3	7	9
9	8	7	3	4	10	8
10	8	3	4	3	9	4

It is now easy to see how the edges incident on a given vertex or the edges enclosing a given face can be obtained from the DCEL. If the graph has N vertices and F faces, we can assume we have two arrays $HV[1:N]$ and $HF[1:F]$ of headers of the vertex and face lists: these arrays can be filled by a scan of arrays $V1$ and $F1$ in time $O(N)$. The following straightforward procedure, VERTEX(j), obtains the sequence of edges incident on v_j as a sequence of addresses stored in an array A.

```
        procedure VERTEX(j)
begin a := HV[j];
      a0 := a;
      A[1] := a;
      i := 2;
      if (V1[a] = j) then a := P1[a] else a := P2[a];
      while (a ≠ a0) do
          begin A[i] := a;
                if (V1[a] = j) then a := P1[a] else a := P2[a];
                i := i + 1
          end
end.
```

	V1	V2	F1	F2	P1	P2
1	1	2	4	1	2	3
2	3	1	4	1	10	1
3	4	2	1	2	4	6
4	3	4	1	3	2	5
5	4	5	2	3	3	7
6	5	2	2	4	5	1
7	6	5	3	4	8	6
8	6	7	4	3	7	9
9	8	7	3	4	10	8
10	8	3	4	3	9	4

```
        procedure VERTEX(j)
  begin a := HV[j];
        a0 := a;
        A[1] := a;
        i := 2;
        if (V1[a] = j) then a := P1[a] else a := P2[a];
        while (a ≠ a0) do
             begin A[i] := a;
                   if (V1[a] = j) then a := P1[a] else a := P2[a];
                   i := i + 1
             end
  end.
```

	V1	V2	F1	F2	P1	P2
1	1	2	4	1	2	3
2	3	1	4	1	10	1
3	4	2	1	2	4	6
4	3	4	1	3	2	5
5	4	5	2	3	3	7
6	5	2	2	4	5	1
7	6	5	3	4	8	6
8	6	7	4	3	7	9
9	8	7	3	4	10	8
10	8	3	4	3	9	4

Clearly VERTEX(j) runs in time proportional to the number of edges incident on v_j. Analogously, we can develop a procedure, FACE(j), which obtains the sequence of edges enclosing f_j, by replacing HV and $V1$ with HF and $F1$, respectively, in the above procedure VERTEX(j). Notice that the procedure VERTEX traces the edges counterclockwise about a vertex while FACE traces them clockwise about a face.

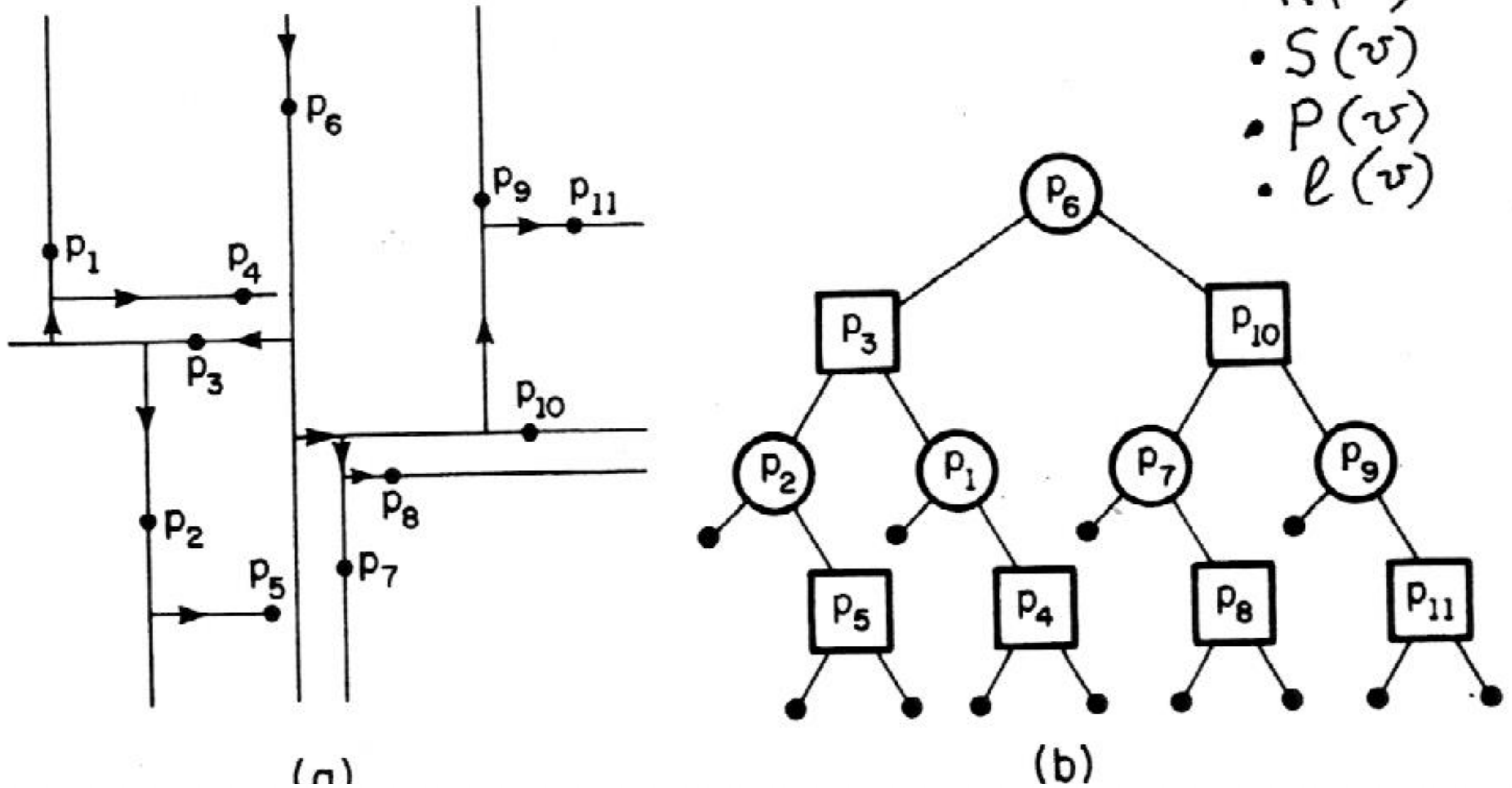
2.3 Range-Searching Problems (<u>Report</u>/Count)

- In 1D: $\Theta(\log N + m)$ query time, $\Theta(N)$ storage, $\Theta(N \log N)$ prepr.

2.3.2 The method of the multidimensional binary tree (k-D tree)

- In 2D: - given N points $v_i = (x_i, y_i)$ $i=\overline{1,N}$
  and rectangular range $D : [x', x''] \times [y', y'']$
  - report all points that are in D.

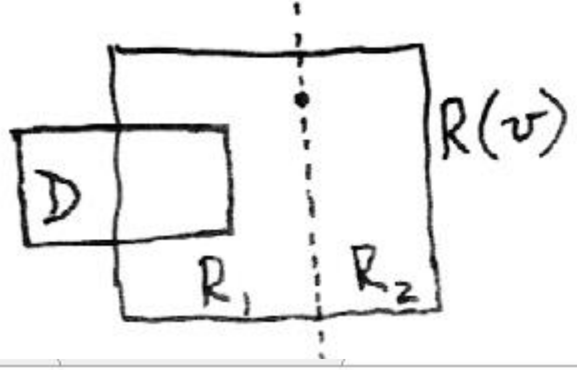
. two dimensional binary search tree

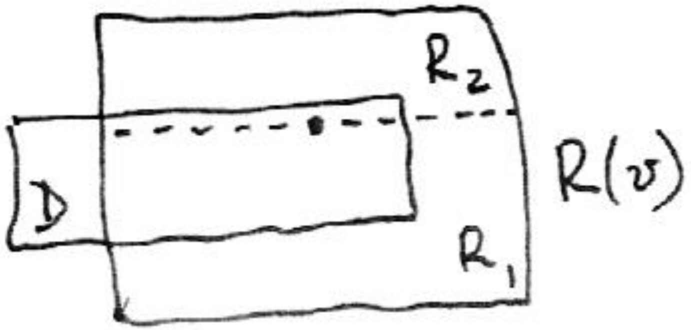


- $R(v)$
- $S(v)$
- $P(v)$
- $\ell(v)$

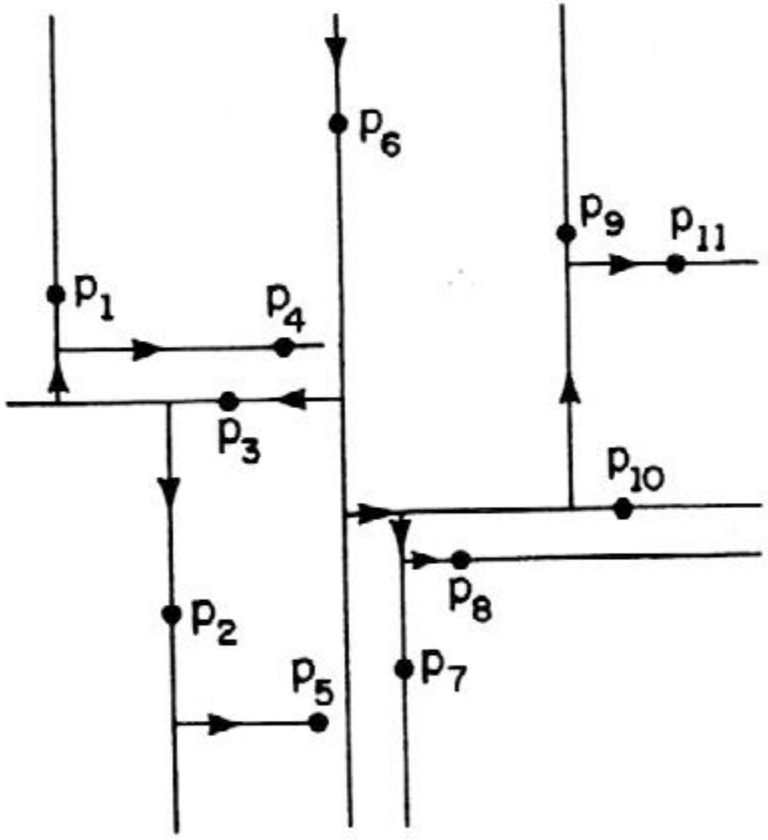
Figure 2.28 Illustration of the two-dimensional binary-tree search method. The partition of the plane in (a) is modeled by the tree in (b). Search begins with the vertical line through P_6. In (b) we have the following graphic convention: circular nodes denote vertical cuts; square nodes denote horizontal cuts; solid nodes are leaves.

if $D \cap R(v) \neq \emptyset$:

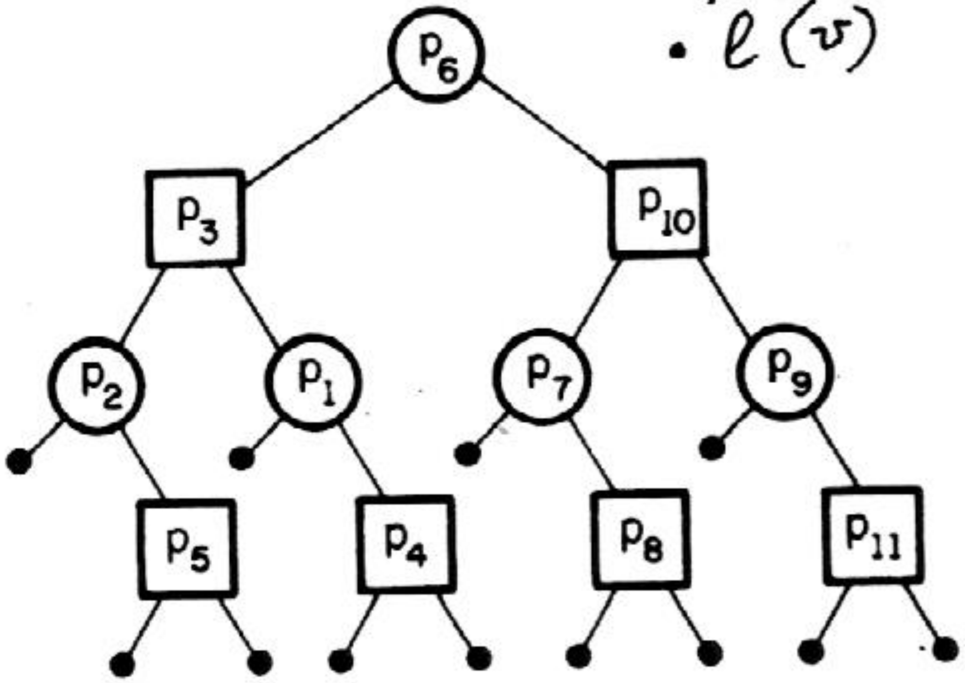




- $R(v)$
- $S(v)$
- $P(v)$
- $\ell(v)$



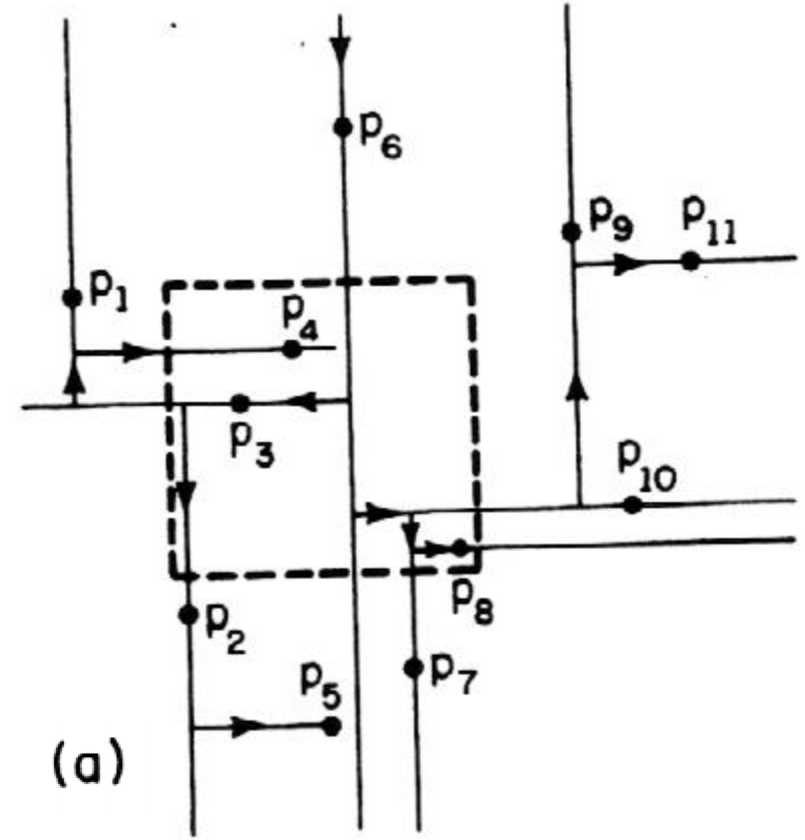
(a)



(b)

$D: [x_1, x_2] \times [y_1, y_2]$

```
      procedure SEARCH(v, D)
begin if (t(v) = vertical) then [l, r] := [x1, x2] else [l, r] := [y1, y2];
      if (l ≤ M(v) ≤ r) then if (P(v) ∈ D) then U ⇐ P(v);
      if (v ≠ leaf) then
         begin if (l < M(v)) then SEARCH(LSON[v], D);
               if (M(v) < r) then SEARCH(RSON[v], D)
         end
end.
```



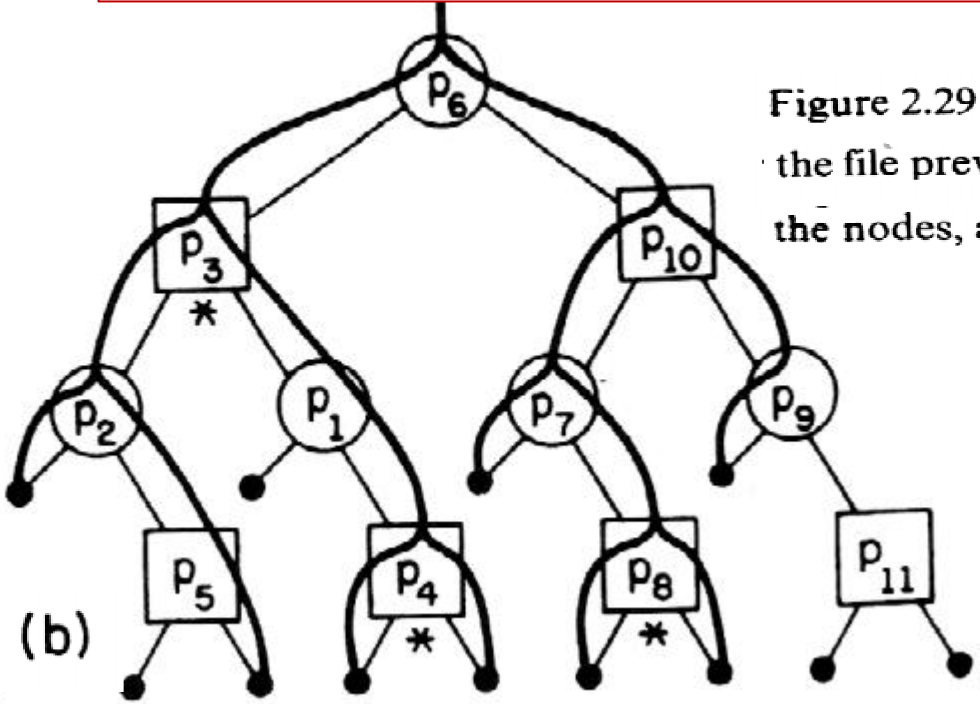


Figure 2.29 Illustration of a range search for the file previously given. Part (b) shows the nodes, actually visited by the search.

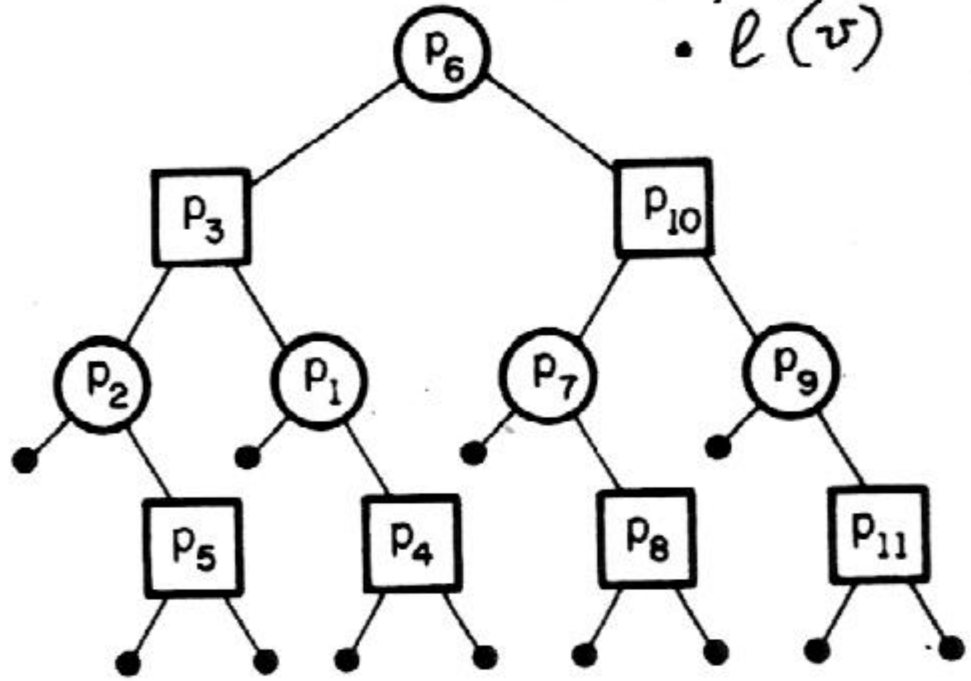
- $\Theta(N)$ storage (one node per point of S)
- $\Theta(N \log N)$ construction of the 2-D tree

$$T(N) \le 2\,T(N/2) + O(N)$$

- $\Theta(N^{1/2} + m)$ query time

SEARCH(root(T), D)

- $R(v)$
- $S(v)$
- $P(v)$
- $\ell(v)$



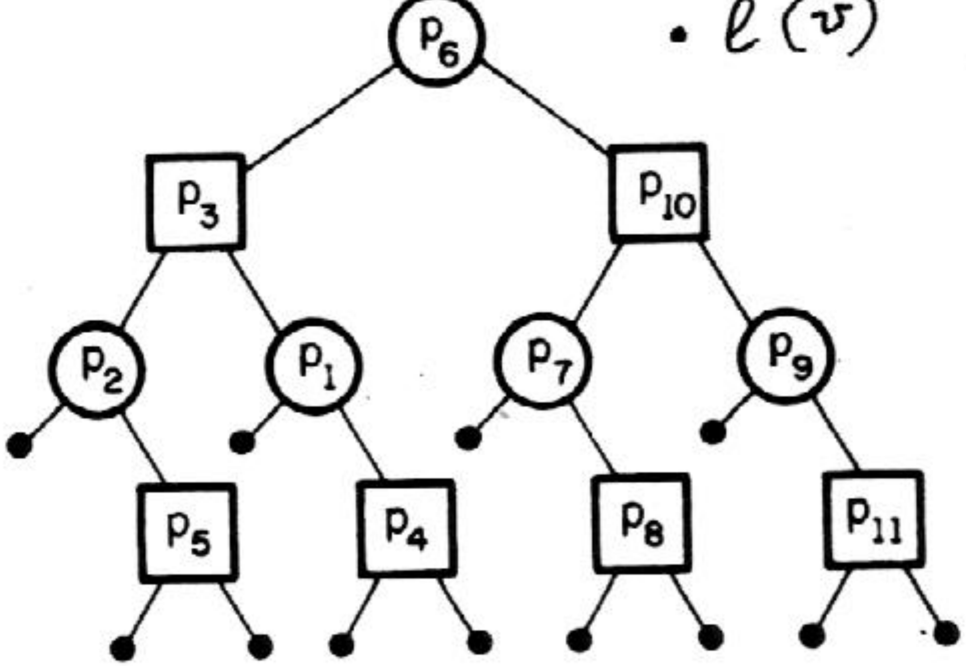
Query time

- clearly, it is proportional to the total number of nodes of T visited by the search algorithm,

  since at each node the search algorithm spends a constant amount of time.

- The time to traverse a subtree and report points is linear in the number of reported points $\Rightarrow \Theta(m)$
- To bound the number of other nodes visited, we bound the number of regions intersected by edges of the query rectangle.

- $R(v)$
- $S(v)$
- $P(v)$
- $\ell(v)$



- $Q(n) = O(1) \quad n = 1$
- $Q(n) = 2 + 2\,Q(n/4) \quad \text{if } n > 1$

$$\Rightarrow \quad Q(n) = O(\sqrt{n})$$

- in k-D:
  - $\Theta(kN)$ storage
  - $\Theta(kN \log N)$ prepr. time
  - $\Theta(kN^{1-1/k} + m)$ query time