Fortgeschrittene Datenstrukturen — Vorlesung 12

Schriftführer: Johannes Bittner

26.1.2012

1 Sparse Bitmaps

Our final task is to prove the sparse bitmap theorem: represent a bit-vector $B[0, n-1]$ containing u 1's in $O(u * \lg(n/u)) + o(n)$ bits such that *rank*, *select* and access to any $B[i]$ can be answered in $O(1)$ time. Note that the space is $o(n)$ if $u = o(n)$. Our strategy is to compress B such that arbitrary $C = O(\lg n)$ consecutive bits $B[i \dots i + C - 1]$ can be accessed in $O(1)$ time. Then we can re-use the *rank* and *select* data structures from the previous section: whenever they need to make a table lookup on a block of size $\frac{\lg n}{2}$, we load those bits in $O(1)$ time. Accessing $B[i]$ works similar: extract the bit from its corresponding $\lg n$-sized chunk using bit-operations on words.

Again, we divide B into blocks of size $s = \frac{\lg n}{2}$. Each block B_i will be represented *individually* by two values, where i is the block index:

1. u_i: the *number* of 1's in the block.

2. o_i: an *index* in an enumeration of all $\binom{s}{u_i}$ bit-vectors of length s containing u_i 1's.

To recover the original block contents from a (u_i, o_i)-pair, we store a universal lookup table $BlkContents$, where $BlkContents[u_i][o_i]$ contains the original s bits of a block that is encoded by (u_i, o_i). We now show how to store and recover the (u_i, o_i)-pair efficiently.

The u_i's are stored in an array $U[0, \frac{n}{s}]$ containing numbers of size $\lg s$ bits, and the o_i's are stored in a *bit stream* O of variable-length numbers. In order to recover the o_i-values from O, we use again a 2-level storage scheme: group s consecutive blocks into *superblocks* of size $s' = s^2$ and store in $SBlk[i_{SBlk}]$ the beginning of o_i's in O, where $0 \leq i_{SBlk} \leq \lceil \frac{n}{s'} \rceil - 1$. In a second table $Blk[i]$, we store the beginning of the description of o_i in O, but this time only relative to the beginning of the corresponding superblock. Those two tables allow to recover the o_i's for any block i_{Blk}.

1.1 Space analysis

$$
\begin{aligned}
|U| &= \frac{n}{s} * \lg s = O(\frac{n * \lg\lg n}{\lg n}) \\
|SBlk| &= \frac{n}{s'} * \lg n = O(\frac{n}{\lg n}) \\
|Blk| &= \frac{n}{s} * \lg s' = O(\frac{n * \lg\lg n}{\lg n}) \\
|BlkContents| &= \sum_{u=0}^{s} \binom{s}{u} * s \\
&\leq s * 2^s * s = O(\sqrt{n} \lg^2 n) \\
|O| &= \sum_{i=0}^{n/s} \lceil \lg \binom{s}{u_i} \rceil \\
&\leq \sum \lg \binom{s}{u_i} + \frac{n}{s} \\
&\leq \lg \binom{n}{u} + \frac{n}{s} && (\text{since } \binom{n}{u} \leq \binom{s}{u_1} * \cdots * \binom{s}{u_{n/s}}) \\
&= \lg \frac{n!}{u! * (n-u)!} + \frac{n}{s} \\
&= \lg \frac{\overbrace{n * (n-1) * \cdots * (n-u+1)}^{u \text{ factors}} * (n-u) * \cdots * 1}{u! * (n-u) * \cdots * 1} + \frac{n}{s} \\
&\leq \lg \frac{n^u}{u!} + \frac{n}{s} \\
&\leq \lg \frac{n^u * e^u}{u^u} + \frac{n}{s} && (\text{Stirling's approximation}) \\
&= O(u \lg \frac{n}{u}) + O(\frac{n}{\lg n})
\end{aligned}
$$

1.2 Example of bit vector compression

With the data structures below, accessing a bit in B, for example $B[18]$, could be achieved as follows:

- Determine block $i = \frac{18}{s} = 4$ and superblock $i_{SBlk} = \frac{18}{s'} = 1$.
- We now want to recover o_4. The value in $SBlk$ is an index into the array O, so we then read $O[SBlk[i_{Sblk}]] = 10$. Furthermore, we need $Blk[i] = 0$. The index into O for retrieving o_4 is thus $10 + 0 = 10$, as $Blk[i]$ is relative to the beginning of the superblock. Therefore, $o_4 = 10$.
- Together with $u_4 = 1$, which can be retrieved from array U, we read $BlkContents[1][10] = 0010$.

0	4	8	12	16	20	24	28	32	36	40	44	
0110	0010	0011	0010	0010	1000	0110	0000	1000	1010	1100	1000	$\}\ B$

Figure 1: Bitmap B, with $s = 4$ and $s' = 16$.

2	1	2	1	1	1	2	0	1	2	2	1	$\}\ U$

Figure 2: Array U, containing the number of 1's for each block in B.

010	10	101	10	10	00	010	0	00	001	000	00	$\}\ O$

Figure 3: Array O, containing the o_i's.

0	10	18	$\}\ SBlk$

Figure 4: Array $SBlk$, the values are indices into the O array.

0	3	5	8	0	2	4	7	0	2	5	8	$\}\ Blk$

Figure 5: Array Blk, the values are arrays into the O array again, but this time relative to the superblock.

(a) $BlkContents[2]$

o_i			block			
0	0	0	1	1	0	0
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	1	0	1
1	0	1	0	0	1	1

(b) $BlkContents[1]$

o_i		block			
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(c) $BlkContents[0]$

o_i	block			
0	0	0	0	0

Table 1: $BlkContents[u_i][o_i]$

2 Distance Oracles in Graphs

In this chapter we show how to preprocess a graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ vertices such that subsequent approximate distance-queries in G can be answered efficiently.

2.1 Basic Definitions

Let $G = (V, E)$ be a *weighted* undirected graph with nonnegative edge weights $\omega(e)$ for $e \in E$. The distance $\delta(u, v)$ between two arbitrary nodes is the weighted path-length of the shortest path between u and v, in symbols:

$$\delta(u, v) = \min \left\{ \sum_{e \in \Pi} \omega(e) : \Pi \text{ is } u\text{-to-}v \text{ path} \right\}$$

Let $\hat{\delta}$ be an *estimate* to $\delta(u, v)$. We say that $\hat{\delta}(u, v)$ is of *stretch* t iff

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq t * \delta(u, v)$$

The aim of this chapter is to show the following theorem:

Theorem 1. *For any parameter* $k \geq 1$*, a graph* G *can be preprocessed in expected* $O(kn^{1/k}(n \lg n + m))$ *time, producing a data structure of* $O(kn^{1+1/k})$ *size, such that subsequent approximate distance queries can be answered in* $O(k)$ *time, with stretch* $t \leq 2k - 1$.

Note that the theorem only considers pure *distance* queries. However, it is also possible to return a corresponding path in constant time per edge.

2.2 Approximate Distance Oracles for Metric Spaces

Let us first assume that we are given an $(n \times n)$ distance matrix representing a finite *metric* δ on V. For example, we can assume that δ is the shortest path metric induced by the graph G. An example of a graph is shown in Figure 6, with its corresponding distance matrix in Table 2.

	A	B	C	D	E	F	G	H
A	0	1	2	3	4	3	8	6
B		0	2	3	5	3	8	6
C			0	1	3	1	6	4
D				0	4	2	5	3
E					0	2	4	6
F						0	6	4
G							0	2
H								0

Table 2: Example of distance matrix, representing δ of G.

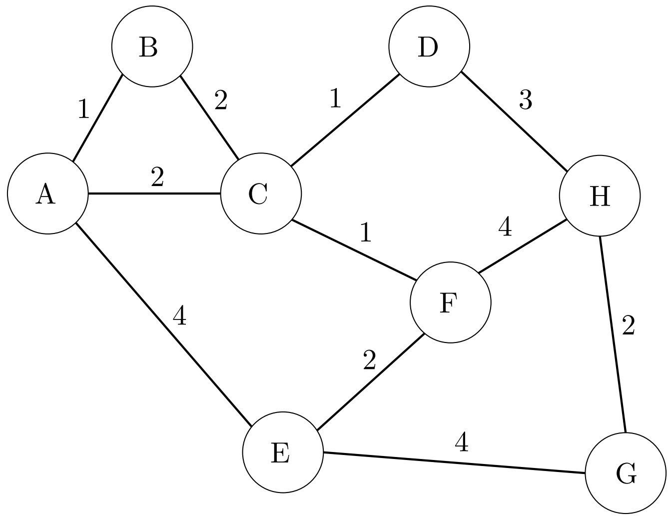


Figure 6: Example for graph G.

2.2.1 Preprocessing

The preprocessing algorithm starts by constructing a non-decreasing sequence of sets

$$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k = \emptyset$$

in a *randomized* manner. The rule is that each element of A_{i-1} is placed in A_i independently, with probability $n^{-1/k}$. We assume that $A_{k-1} \neq \emptyset$ (otherwise the construction has to be restarted). The *expected size* $Exp[|A_i|]$ of A_i, for $0 \leq i \leq k$, is

$$\begin{aligned} Exp[|A_i|] &= |V| * Prob[v \in A_j \;\forall\, 1 \leq j \leq i] \\ &= n * \underbrace{n^{-1/k} * n^{-1/k} * \cdots * n^{-1/k}}_{i \text{ times}} \\ &= n^{1-i/k} \end{aligned}$$

For each vertex $v \in V$ and every index $i = 0, \ldots, k-1$, we compute and store $\delta(A_i, v)$, the *smallest distance* from v to a vertex in A_i. The algorithm also computes and stores an element $p_i(v)$, the *witness*, that is nearest to A_i. That is, $\delta(p_i(v), v) = \delta(A_i, v)$. We define $\delta(A_k, v) = \infty$ for all $v \in V$ and leave $p_k(v)$ undefined.

Example 1. *Let $A_1 = \{B, E, F, G\}$, $A_2 = \{E, F\}$, $A_3 = \{E\}$ and $A_4 = \emptyset$. Then $\delta(A_i, v)$ and $p_i(v)$ have the values as shown in Table 3.*

The size of this table is $O(k * n)$.

2.2.2 Bunches

For each vertex $v \in V$, the algorithm also computes a *bunch* $B(v) \subseteq V$ as follows. Informally, a vertex w is put into the bunch of v if w is in A_i, but not in A_{i+1}, and it is closer to v than v is to A_{i+1}. In symbols,

		$\delta(A_i,v)$						$p_i(v)$				
v	$i=$	0	1	2	3	4	$i=$	0	1	2	3	4
A		0	1	3	4	∞		A	B	F	E	$\perp$
B		0	0	3	5	∞		B	B	F	E	$\perp$
C		0	1	1	3	∞		C	F	F	E	$\perp$
D		0	2	2	4	∞		D	F	F	E	$\perp$
E		0	0	0	0	∞		E	E	E	E	$\perp$
F		0	0	0	2	∞		F	F	F	E	$\perp$
G		0	0	4	4	∞		G	G	E	E	$\perp$
H		0	2	4	6	∞		H	G	F	E	$\perp$

Table 3: $\delta(A_i,v)$ and $p_i(v)$ of graph shown in Figure 6.

$$w \in B(v) \Leftrightarrow \exists i : w \in A_i \backslash A_{i+1} \text{ and } \delta(w,v) < \delta(A_{i+1},v)$$

A schematic view of bunches, assuming Euclidian distances, is shown in Figure 7. The arrows point to the elements which belong to $B(v)$. Note that since $\delta(A_k,v)=\infty$, we get that $A_{k-1} \subseteq B(v)$ for every $v \in V$. This is shown in Figure 7, where all elements of A_2 are included in $B(v)$.

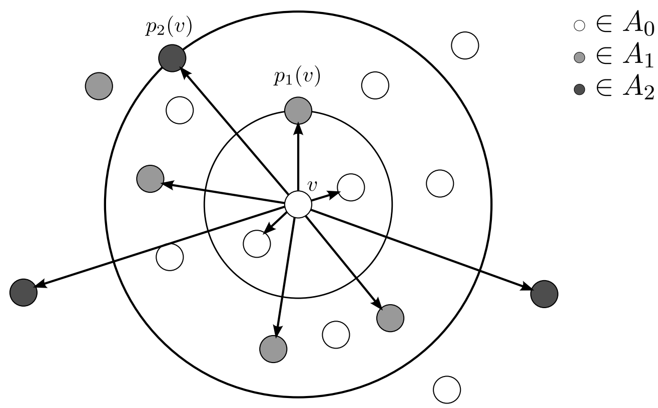


Figure 7: Schematic view of bunches

Example 2. *$B(A)$ is the bunch of A, using the values of Table 3.*

$$B(A) = \left\{ \underbrace{A}_{0=\delta(A,A)<\delta(A_1,A)=1}, \underbrace{B}_{1=\delta(A,B)<\delta(A_2,A)=3}, \underbrace{F}_{3=\delta(A,F)<\delta(A_3,A)=4}, \underbrace{E}_{4=\delta(A,E)<\delta(A_4,A)=\infty} \right\}$$

The bunch $B(v)$ is stored in a perfect hash table of size $O(|B(v)|)$, such that for an arbitrary $w \in V$ it is possible in $O(1)$ time to tell if $w \in B(v)$. If $w \in B(v)$, we also store the distance $\delta(v,w)$.

We now bound the *expected sizes* of the bunches.

Lemma 2. *The expected size of $B(v)$ is $k * n^{1/k}$.*

Proof. We show that in any iteration of the preprocessing algorithm, the bunch grows only by $n^{1/k}$ elements in expectation, in symbols:

$$Exp[|B(v) \cap (A_i \backslash A_{i+1})|] = n^{1/k} \; \forall\, 0 \leq i \leq k-1$$

For $i = k-1$ the claim is trivial, as all elements from A_{k-1} are in the bunch and $Exp[|A_{k-1}|] = n^{1-\frac{k-1}{k}} = n^{1/k}$. For $i < k-1$, let $w_1, \ldots, w_x$ be the elements of A_i arranged in nondecreasing order of distance from v. Figure 8 shows a schematic view of those nodes, again assuming Euclidian distances.

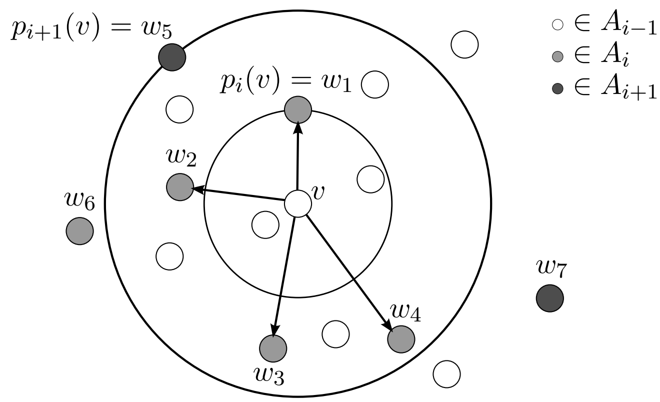


Figure 8: Sketch showing $w_1, \ldots, w_x$

If $w_j \in B(v)$, then $\delta(w_j, v) < \delta(A_{i+1}, v)$, and thus $w_1, \ldots, w_j \notin A_{i+1}$. So $Prob[w_j \in B(v)] \leq (1-p)^j$ for p being the probability that an element from A_i is placed into A_{i+1}, as all $w_1, \ldots w_j$ must not be in A_{i+1}. So the expected size of $B(v) \cap (A_i \backslash A_{i+1})$ is at most

$$\begin{aligned}
& \sum_{j=1}^{x} Prob[w_j \in B(v)] \\
\leq & \sum_{j=1}^{x} (1-p)^j \\
\leq & \sum_{j=0}^{\infty} (1-p)^j \\
< & \; p^{-1} && \text{(geometric series)} \\
= & \; n^{1/k} && \text{(by definition of } A_{i+1}\text{)}
\end{aligned}$$

□

Using this lemma, the total size of all hash tables is $\sum_{v \in V} |B(v)| = n^{1+1/k}$ in expectation. As usual by rerunning the algorithm until the data structure is small enough this is the space in the worst case; the expected number of trials to achieve this space is constant by *Markov's inequality*. The overall *running time* is $O(n^2)$.

2.3 Answering Distance Queries

The idea of the query algorithm is to iterate through the preprocessed layers until the bunches intersect, as illustrated in Figure 9. Note that $\delta(p_3(u), v)$ is stored in the hash table of $B(v)$, and $\delta(u, p_3(u))$ is stored in the global table of Section 2.2.1.

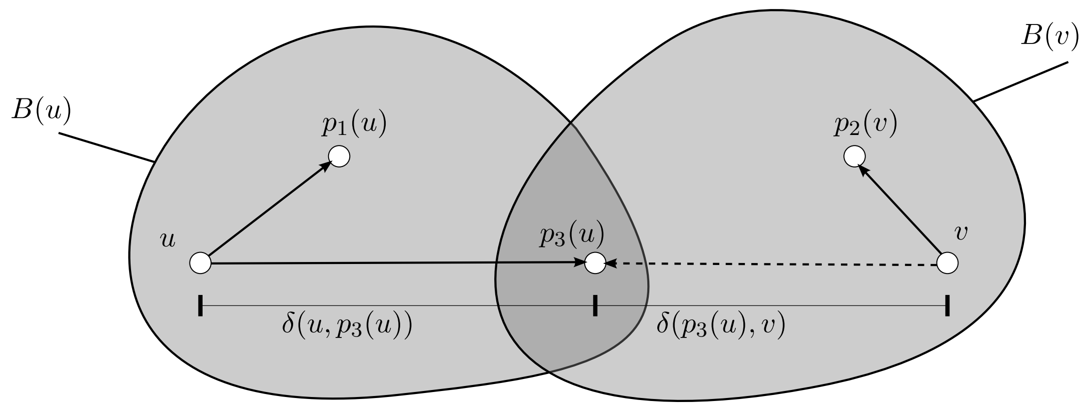


Figure 9: Sketch of the query algorithm

The complete algorithm is best shown by means of pseudo-code, which is shown in Algorithm 1. Note that the algorithm *always terminates*, as if $i = k-1$, $w \in A_{k-1}$ and $A_{k-1} \subseteq B(v)$ for every $v \in V$.

Algorithm 1: Computing $dist_k(u, v)$

$w \leftarrow u$;
$i \leftarrow 0$;
while $w \notin B(v)$ **do**
  $i \leftarrow i + i$;
  $w \leftarrow p_i(v)$;
  $(u, v) \leftarrow (v, u)$;
end
return $\delta(w, u) + \delta(w, v)$;

We finally show that the stretch produced by $dist_k(u, v)$ is at most $(2k - 1)$.

Lemma 3. $dist_k(u, v) \leq (2k - 1) * \delta(u, v)$

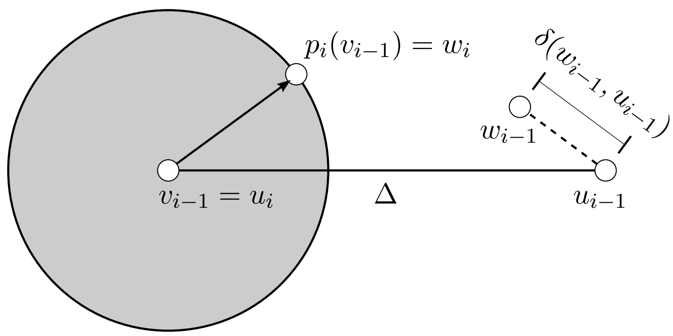
Proof. Let $\Delta = \delta(u, v)$. We show that each iteration increases $\delta(w, u)$ by at most Δ. This proves our claim, since in the beginning $\delta(w, u) = 0$ and there are at most $k - 1$ iterations, we will end up with $\delta(w, u) \leq (k - 1) * \Delta$. Now,

$$\begin{aligned}
\delta(w,v) &\leq \delta(w,u)+\delta(u,v) && \text{(triangle inequality)}\\
&\leq (k-1)*\Delta+\Delta\\
&= k*\Delta
\end{aligned}$$

so $dist_k(u,v) = \delta(u,w) + \delta(w,v) \leq (2k-1)*\Delta$.

Let u_i, v_i and w_i be the values of the variables u, v, w assigned with a given value of i ($u_0 = u, v_0 = v$ and $w_0 = u$), so $\delta(w_0, u_0) = 0$. We want to show $\delta(w_i, u_i) \leq \delta(w_{i-1}, u_{i-1}) + \Delta$ if the $i'th$ iteration passes the test of the while loop. Then $w_{i-1} \notin B(v_{i-1})$, so

$$\begin{aligned}
\delta(w_{i-1},v_{i-1}) &\geq \delta(A_i,v_{i-1})\\
&= \delta(p_i(v_{i-1}),v_{i-1}) \;=\; \delta(w_i,u_i)
\end{aligned}$$



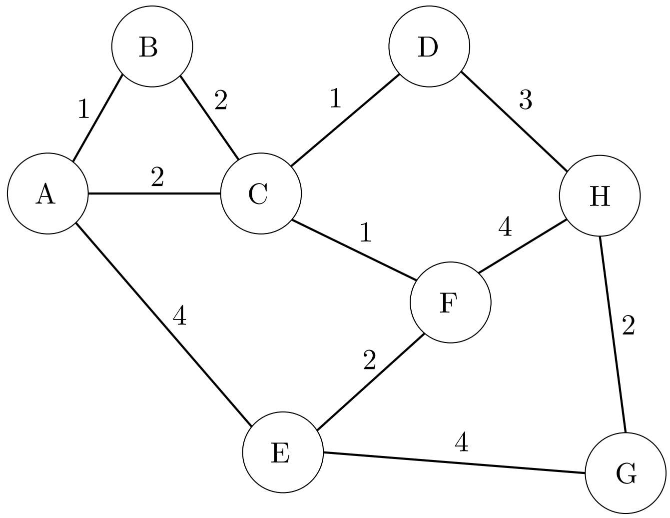
So by using the triangle inequality, we get

$$\begin{aligned}
\delta(w_i,u_i) &\leq \delta(w_{i-1},v_{i-1})\\
&\leq \delta(w_{i-1},u_{i-1}) + \delta(u_{i-1},v_{i-1})\\
&= \delta(w_{i-1},u_{i-1}) + \Delta
\end{aligned}$$

□

2.4 Example Distance Query

For the example distance query $dist_k(H, A)$, we use the same graph and sets A_i as in the previous subsections.



$$
\begin{aligned}
A_0 &= \{A, B, C, D, E, F, G, H\} \\
A_1 &= \{B, E, F, G\} \\
A_2 &= \{E, F\} \\
A_3 &= \{E\} \\
A_4 &= \emptyset
\end{aligned}
$$

Given those definitions, the following bunches $B(A)$ and $B(H)$ result:

$$
\begin{aligned}
B(A) &= \{A, B, F, E\} \\
B(H) &= \{H, G, F, E\}
\end{aligned}
$$

The following shows the query $dist_k(H, A)$. Note that $\delta(F, A)$ is stored with the bunch of $\delta(F, A)$, as $F \in B(A)$, whereas $\delta(F, H) = \delta(A_2, H)$ is stored with $p_2(H)$. Also note that there exists a shorter path from $A \to C \to D \to H$ with $\delta(A, H) = 6$.

$dist_k(H, A)$

$i = 0: \; w = H \notin B(A)$

$\Rightarrow i \leftarrow i + 1$

$\Rightarrow w \leftarrow p_1(A) = B$

$i = 1: \; w = B \notin B(H)$

$\Rightarrow i \leftarrow i + 1$

$\Rightarrow w \leftarrow p_2(H) = F$

$i = 2: \; w = F \in B(A)$

$\Rightarrow$ return $\underbrace{\delta(F,H)}_{4} + \underbrace{\delta(F,A)}_{3}$

References

[1] Thorup and Zwick. Approximate distance oracles. *JACM: Journal of the ACM*, 52, 2005.