Randomized Algorithms

Dr Hamed Vahdat-Nejad

Outline

- Different types of randomized algorithms .
- Randomized Quick sort (Las Vegas).
- Randomized Min-cut (Monte Carlo)

Advantages of randomized algorithms

- performance; Randomized algorithms run faster than the best-known deterministic algorithms.
- Simplicity; Many randomized algorithms are simpler to describe and implement than deterministic algorithms of comparable performance.
- Foiling an adversary; While the adversary may be able to construct an input that foils a deterministic algorithms, it is difficult to devise a single input that is defeat a randomly chosen algorithm.

Types of randomized algorithms

1. Las Vegas algorithm
   - A randomized algorithm that always return the correct result.
   - The running time might change between executions.
2. Monte Carlo algorithm
   - A randomized algorithm that might output an incorrect result.
   - the probability of error can be diminished by repeating the execution of the algorithm.

Las Vegas algorithm (an example)

- Sorting Algorithms: The problem is concerned with sorting an array of n elements into ascending order.
- Quick Sort (deterministic):
  1. picks an element from the array (the first element)
  2. partitions the remaining elements into those greater than and those less than this pivot.
  3. Then recursively sorts the partitions.

Quick Sort

{**26**	5	37	1	61	11	59	15	48	19}
{ **11**	5	19	1	15}	**26**	{59	61	48	37}
{1	5}	**11**	{19	15}	**26**	{59	61	48	37}
1	5	**11**	15	19	**26**	{48	37}	59	{61}
1	5	**11**	15	19	**26**	37	48	**59**	{61}
{1	5	**11**	15	19	**26**	48	37	**59**	61}

The adversary can provide a reverse-sorted array as input (worst-case).

The array will always be split into two unbalanced sets, a set of one element and a set consisting of the remainder.

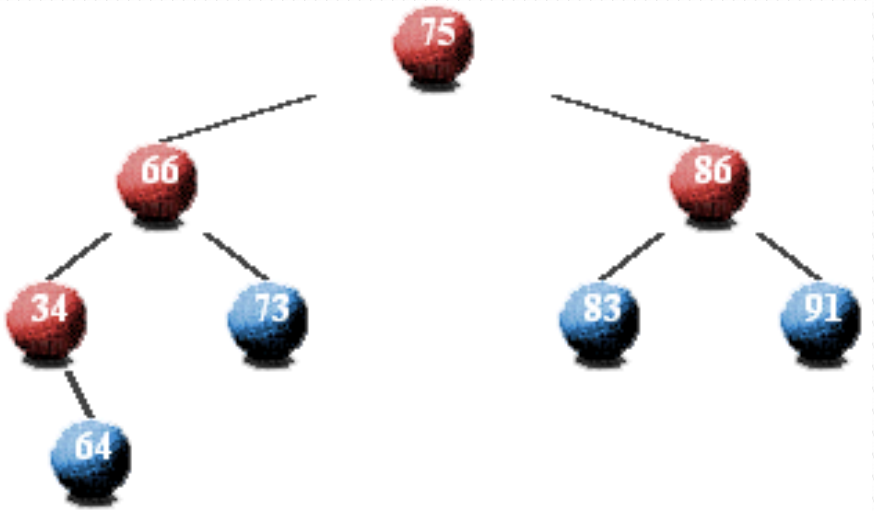
$T(n)=T(n-1)+n$

$T(n)=n(n-1)=O(n^2)$

Randomized Quick Sort

1. Choose an element Y uniformly at random from S.
2. Determine the set S1 of elements smaller than Y and the set S2 of elements larger than Y .
3. Recursively sort S1 and S2.

Illustration



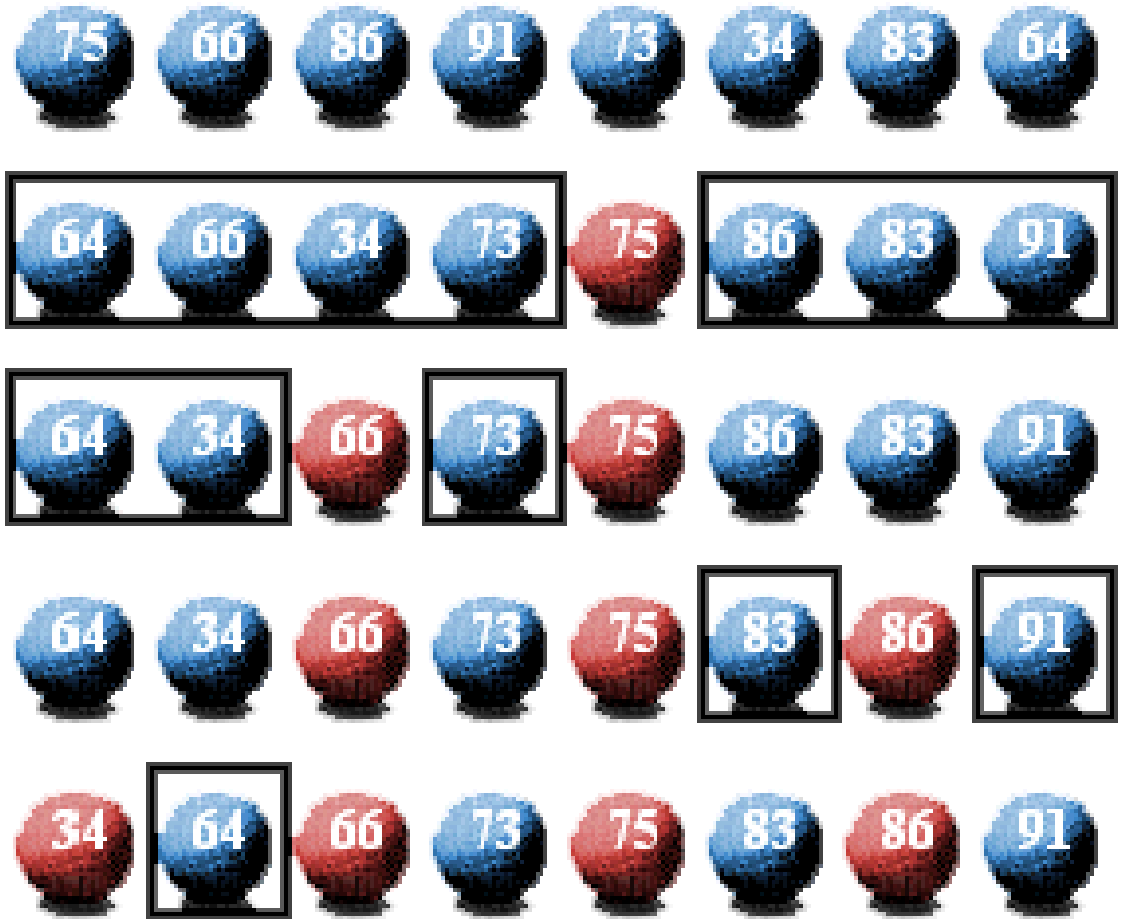
Complexity analysis

- We are interested in the expected number of comparisons in an execution of this algorithm.

- All the comparisons are performed in Step 2, in which we compare a randomly chosen partitioning element to the remaining elements.

- For $1 < i < n$, let S(i) denote the element of rank i (the ith smallest element) in the set S.
- Define the random variable X_{ij} to assume the value 1 if S(i) and S(j) are compared in an execution, and the value 0 otherwise.

- Two different elements are at most compared once.


75 66 86 91 73 34 83 64
64 66 34 73 75 86 83 91
64 34 66 73 75 86 83 91
64 34 66 73 75 83 86 91
34 64 66 73 75 83 86 91

Total number of comparisons is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

We are interested in the expected number of comparisons, which is clearly

$$E\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left[X_{ij}\right]$$

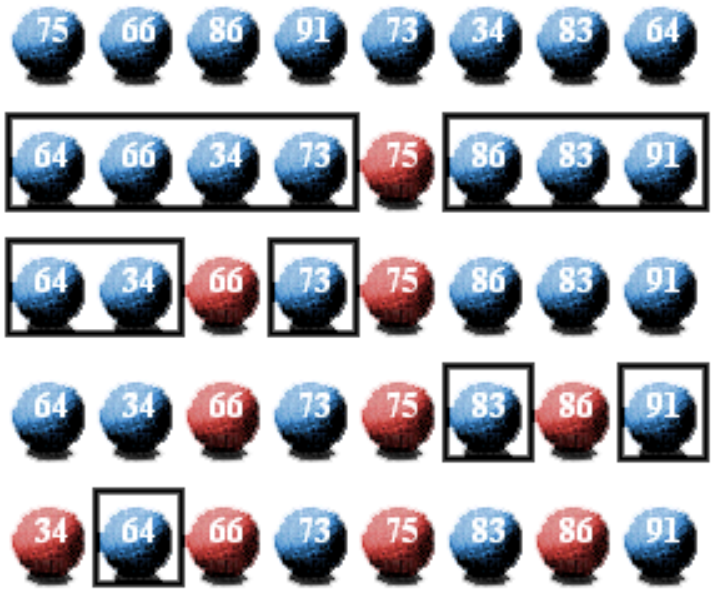
- Let p_{ij} denote the probability that S(i) and S(j) are compared in an execution.
- X_{ij} is a Bernoulli random variable that only assumes the values 0 and 1.

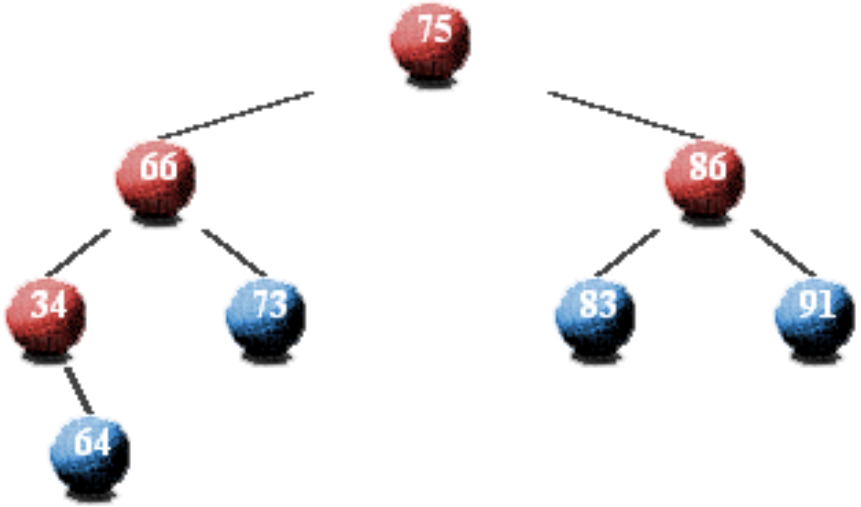
$$\mathbf{E}[X_{ij}] = p_{ij} \cdot 1 + (i - p_{ij}) \cdot 0 = p_{ij}$$

- Thus, the expected number of comparisons is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} p_{ij}$$

- There is a comparison between S(i) and S(j) if and only if one of these elements is an ancestor of the other.





- This happens only if S(i) or S(j) was selected first among the elements that are larger than S(i) and smaller than S(j).
- S(i) and S(j) goes into different partitions and are never compared if and only if an element in between them is picked first by the random algorithm.
- The probability of picking S(i) or S(j) first in the set {S(i), S(i + 1), ..., S(j)} is $2/(j - i + 1)$.

- Accordingly, The expected number of comparisons is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

- Let k = j − i + 1; thus the expected number of comparisons:

$$= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \leqslant \sum_{i=1}^{n} \sum_{k}^{n} \frac{2}{k}$$

$$\leqslant 2 \sum_{i=1}^{n} H_n \leqslant 2nH_n$$

$$\leqslant n \ln n$$

- H_n is the n_{th} Harmonic number, and is roughly ln n.
- the expected running time is O(n log n).
- The expected running time depends only on random choices made by the algorithm and not on any assumptions about the distribution of the input.
- Behavior can vary even on the same input.

A Monte Carlo algorithm

Randomized Min-cut

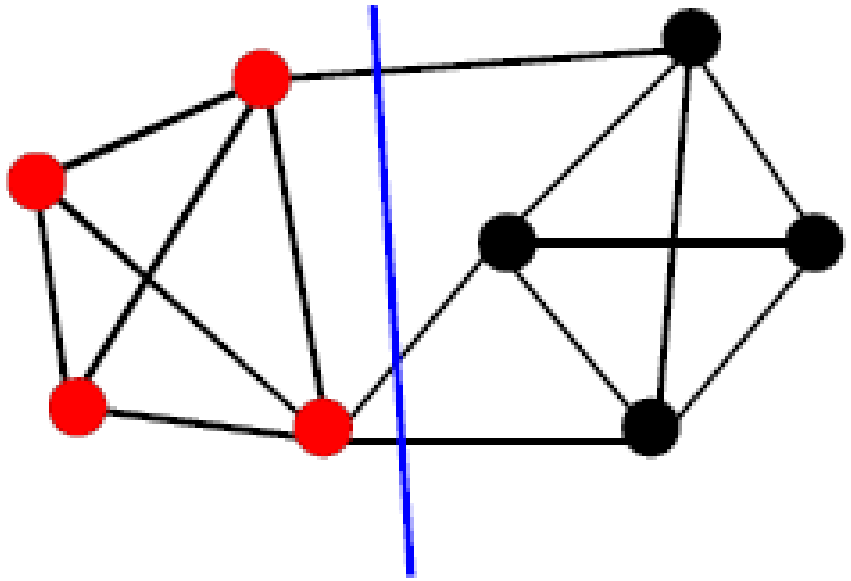
Definitions

A cut in G is a partition of the vertices of V into two sets S and V \ S, where the edges of the cut are

$(S, V \setminus S) = \{uv \mid u \in S, v \in V \setminus S, \text{ and } uv \in E\}$,

where $S \neq \emptyset$; and $V \setminus S \neq \emptyset$; We will refer to the number of edges in the cut (S, V \ S) as the size of the cut.

A cut



Problem

- Computing the minimum cut, that is, the cut in the graph with minimum cardinality.

Some introductory concepts

$$\mathbf{Pr}[X = x \,|Y = y] = \frac{\mathbf{Pr}[(X = x) \cap (Y = y)]}{\mathbf{Pr}[Y = y]}$$

$$\mathbf{Pr}[(X = x) \cap (Y = y)] = \mathbf{Pr}[X = x \,|Y = y] * \mathbf{Pr}[Y = y]$$

if X and Y are independent, then

$$\mathbf{Pr}\left[X = x \,\middle|\, Y = y\right] = \mathbf{Pr}[X = x]$$

Let $\eta_1, \ldots, \eta_n$ be n events which are not necessarily independent. Then,

$$\begin{aligned}\mathbf{Pr}[\cap_{i=1}^{n}\eta_i] &= \mathbf{Pr}[\eta_1] * \mathbf{Pr}[\eta_2 \,|\eta_1\,] * \\ &\quad \mathbf{Pr}[\eta_3 \,|\eta_1 \cap \eta_2] * \ldots * \mathbf{Pr}\left[\eta_n \;\middle|\; \eta_1 \cap \ldots \cap \eta_{n-1}\right]\end{aligned}$$

Edge contraction

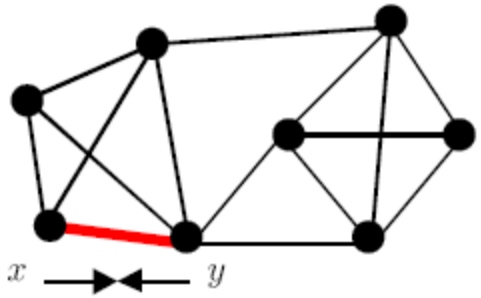
Edge contraction: We take an edge $e = xy$,and merge the two vertices into a single vertex.

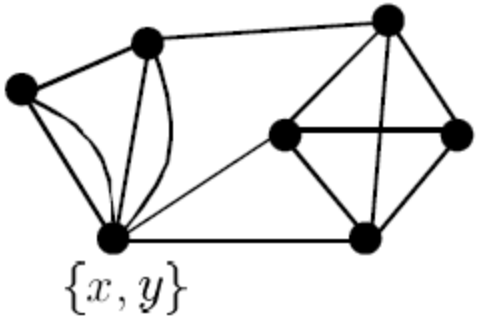
The new graph is denoted by G/xy. We remove self loops.

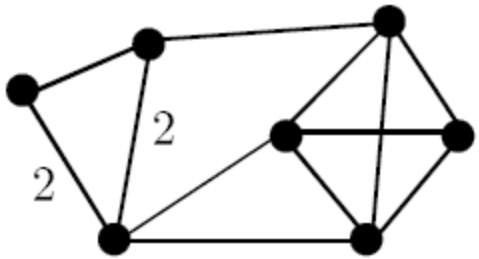
The resulting graph is no longer a regular graph, is a multi-graph.

We represent a multi-graph, as a regular graph with multiplicities on the edges.

Edge contraction







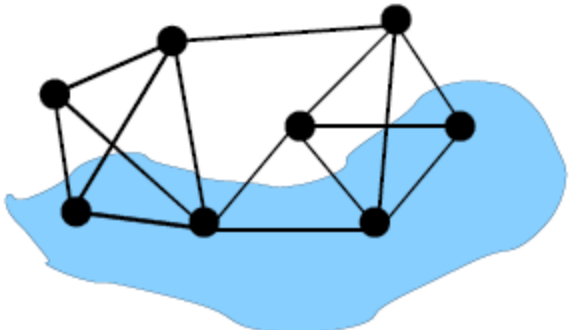
- If an edge is in the cut, and it has weight w we add w to the weight of the cut.
- Each vertex in the contracted graph, corresponds to a connected component in the original graph.
- The size of the minimum cut in G/xy is at least as large as the minimum cut in G. Since any cut in G/xy has a corresponding cut of the same cardinality in G.

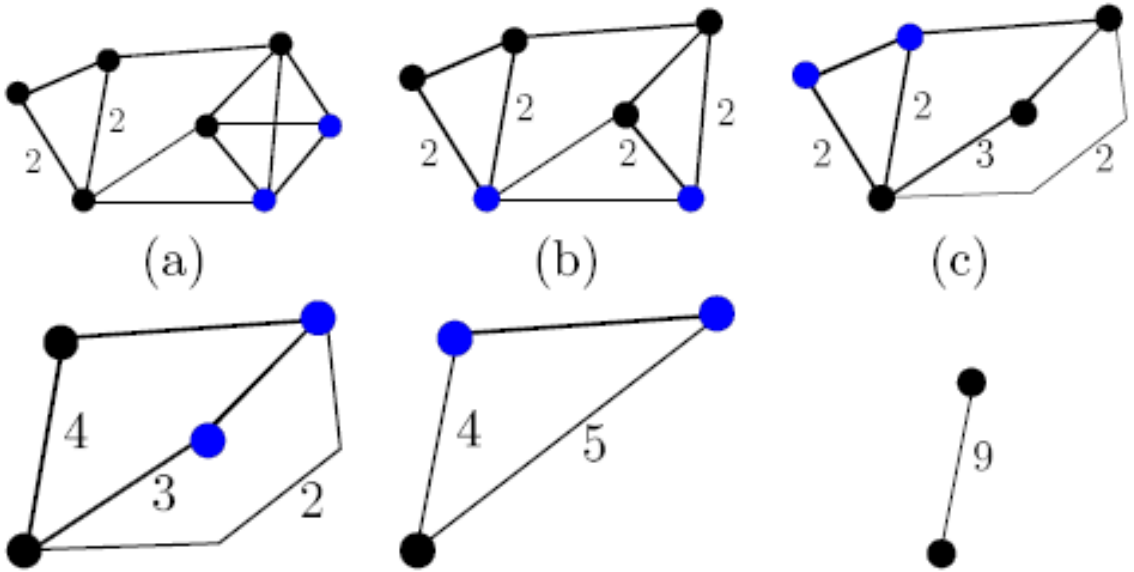
The idea of our algorithm is to repeatedly perform contraction, which is beneficial since it shrinks the graph.

We contract the graph into a single edge.

Indeed, the resulting cut is not surely the minimum cut.

Example





The algorithm

The MinCut algorithm was developed by David Karger during his PhD thesis in Stanford.

Algorithm MINCUT(G)

$G_0 \leftarrow G$

$i = 0$

while G_i has more than two vertices **do**

Pick randomly an edge e_i from the edges of G_i

$G_{i+1} \leftarrow G_i / e_i$

$i \leftarrow i + 1$

Let $(S, V - S)$ be the cut in the original graph corresponding to the single edge in G_i

- The edge contraction operation can be implemented in O(n) time.
- MinCut runs in $O(n^2)$ time.
- The algorithm always outputs a cut, and the cut is not smaller than the minimum cut.

Lemmas

- Let $e_1, \ldots, e_{n-2}$ be a sequence of edges in G, such that none of them is in the minimum cut, and such that $G_0 = G/\{e_1, \ldots, e_{n-2}\}$ is a single multi-edge. Then, this multi-edge corresponds to the minimum cut in G.

Lemmas

- If a graph G has a minimum cut of size k, and it has n vertices, then $|E(G)| \geq kn/2$
- If we pick in random an edge e from a graph G, then with probability at most $2/n$ it belongs to the minimum cut.

- Proof
  - There are at least $nk/2$ edges in the graph and exactly k edges in the minimum cut

MinCut outputs the min cut in probability $\geq \frac{2}{n(n-1)}$.

Proof: Let η_i be the event that e_i is not in the minimum cut of G_i.

$$\mathbf{Pr}[\eta_i \,|\eta_0 \cap \ldots \cap \eta_{i-1}] \geq 1 - \frac{2}{|V(G_i)|} = 1 - \frac{2}{n-i}$$

- MinCut outputs the minimum cut if the events $\eta_0, \ldots, \eta_{n-3}$ all happen

$$
\begin{aligned}
\mathbf{Pr}[\eta_0 \cap \ldots \cap \eta_{n-3}] &= \mathbf{Pr}[\eta_0] \cdot \mathbf{Pr}[\eta_1 \,|\eta_0] \cdot \mathbf{Pr}[\eta_2 \,|\eta_0 \cap \eta_1] \\
&\cdot \quad \ldots \quad \cdot \mathbf{Pr}[\eta_{n-3} \,|\eta_0 \cap \ldots \cap \eta_{n-4}] \\
&\geq \prod_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{n-3} \frac{n-i-2}{n-i} \\
&= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} \cdots \cdot \frac{2}{4} \cdot \frac{1}{3} \\
&= \frac{2}{n \cdot (n-1)}.
\end{aligned}
$$

Definition

- Amplification is the process of running an experiment again and again till the things we want, happens with good probability.

- Let MinCutRep be the algorithm that runs MinCut $n(n-1)$ times and return the minimum cut computed in all those independent executions.

- The probability that MinCutRep fails to return the minimum cut is < 0.14.

- Proof

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)} \leq \exp\left(-\frac{2}{n(n-1)} \cdot n(n-1)\right) = \exp(-2) < 0.14,$$

since $1 - x \leq e^{-x}$ for $0 \leq x \leq 1$.

Conclusion theorem

- One can compute the minimum cut in $O(n^4)$ time with constant probability to get a correct result.

- In $O(n^4 \log n)$ time the minimum cut is returned with high probability.

conclusion

- The sorting algorithm always terminates with the correct solution, but its running time varies from an expected running time of O(nlog n) to a worst-case running time of $O(n^2)$.
- The min-cut algorithm sometimes generates an incorrect solution. The failure probability can be made arbitrarily small by executing the algorithm several times with independent random choices each time.

- A Las Vegas algorithm is efficient if on any input its expected running time is bounded by a polynomial function of the input size.

- a Monte Carlo algorithm is efficient if on any input its worst-case running time is bounded by a polynomial function of the input size.

References

- Ashraf M. Osman, Introduction to Randomized Algorithms, West Virginia University.
- Kumar, Analysis of Algorithms, Nov 03, 2003.
- Sariel Har-Peled, Class notes for Randomized Algorithms, University of Illinois, 2005.

Any questions?